INFORMATIKAI ALGORITMUSOK

2. kötet. Párhuzamos módszerek és optimalizáció

Bővített és javított második kiadás

HOUNTLER Kft.
Budapest, 2015

A könyv a Magyar Tudományos Akadémia támogatásával készült

Szerkesztő: Iványi Antal

Szerzők: © 2014 Bacsárdi László (18. fejezet), Domoszlai László (26. fejezet), Englert Burkhard (19. fejezet), Fohry Claudia (20. fejezet), Galambos Máté (18. fejezet), Gyires Tibor (17. fejezet), Horváth Zoltán (21. fejezet), Illés Tibor (24. fejezet), Imre Sándor (18. fejezet), Imreh Csanád (23. fejezet), Iványi Anna Barbara (Irodalomjegyzék), Iványi Antal (20. fejezet), Kowalski Dariusz (19. fejezet), Lakatos László (27. fejezet), Malewicz Grzegorz (19. fejezet), Nagy Marianna (24. fejezet), Shvartsman Alexander A. (19. fejezet), Szécsi László (16. fejezet), Szeidl László (27. fejezet), Szidarovszky Ferenc (25., 26. fejezet), Szirmay-Kalos László (16. fejezet), Tejfel Máté (21. fejezet), Telek Miklós (27. fejezet), Terlaky Tamás (24. fejezet), Vizvári Béla (23. fejezet), Zehendner Eberhard (22. fejezet)

Szakmai lektorok: Antal György (16. fejezet), Csirik János (28. fejezet), Iványi Antal (18. fejezet), Mayer János (24., 25. fejezet), Majzik István (19. fejezet), Molnár Sándor (26. fejezet), Pataricza András (21. fejezet), Sima Dezső (20., 22. fejezet), Sztrik János (27. fejezet), Vizvári Béla (23. fejezet)

Nyelvi lektor: Bíró Gabriella

Kiadó: Hountler Kft.

Tartalomjegyzék

Előszó

Nagy örömmel ajánlom az Olvasók figyelmébe az *Informatikai algoritmusok* című tankönyvet, Iványi Antal gondos szerkesztésében. A számítógépes algoritmusok az informatika igen fontos és igen gyorsan fejlődő területét alkotják. Hatalmas hálózatok tervezése és üzemeltetése, nagyméretű tudományos számítások és szimulációk, gazdasági tervezés, adatvédelmi módszerek, titkosítás és még sok más alkalmazás igényel hatékony, gondosan tervezett és pontosan elemzett algoritmusokat.

Sok évvel ezelőtt Gács Péterrel írtunk egy kis könyvecskét [184] *Algoritmusok* címmel. Az *Informatikai algoritmusok* három kötete mutatja, hogy milyen sokrétű és szerteágazó területté fejlődött ez a téma. Külön örömet jelent, hogy a magyar informatika ilyen sok kiváló képviselője fogott össze, hogy ez a könyv létrejöjjön. Nyilvánvaló számomra, hogy diákok, kutatók és alkalmazók egyik legfontosabb forrásmunkája lesz hosszú ideig.

Budapest, 2014. október
Lovász László

Bevezetés a 2. kötethez

A második kötet párhuzamos módszerekkel és optimalizációval foglalkozik és tizenhárom fejezetet tartalmaz.

A kötet Szirmay-Kalos László és Szécsi László (BME) *GPGPU: számítások grafikus processzorokon* című **??**, fejezetével kezdődik, majd Gyires Tibor (Illinois Egyetem) *Hálózatok szimulációja* című, 17. fejezetével folytatódik.

A *Kvantumalapú algoritmusok* című **??**. fejezet szerzői Imre Sándor, Bacsárdi László és Galambos Máté (BME), míg az *Osztott algoritmusok* című 21. fejezeté Englert Burkhard (Kaliforniai Állami Egyetem), Kowalski Dariusz (Liverpooli Egyetem), Grzegorz Malewicz (Alabamai Egyetem) és Alexander A. Shvartsman (Connecticuti Egyetem).

A *Párhuzamos számítások* című 20. fejezetet Claudia Fohry (Kasseli Egyetem) és Iványi Antal (ELTE) (ELTE) írták. Horváth Zoltán és Tejfel Máté (ELTE) a szerzői a *Petri-hálók* című 18. fejezetnek. A következő, 22. fejezet címe Szisztolikus rendszerek, szerzője Eberhard Zehendner (Jénai Friedrich Schiller Egyetem.

A *Versenyképességi elemzés* című 23. fejezet Imreh Csanád (Szegedi Tudományegyetem) műve, a *Belsőpontos algoritmusok* című 24. fejezeté pedig Illés Tibor és Nagy Marianna (BME), valamint Terlaky Tamás (Lehigh Egyetem).

A 25. fejezet címe *Játékelmélet*, szerzője pedig Szidarovszky Ferenc (Arizonai Egyetem). Domoszlai László (ELTE) és Szidarovszky Ferenc írták a *Konfliktushelyzetek kezelése* című 26. fejezetet.

A kötet a *Tömegkiszolgálás* című 27. és a *Ütemezéselmélet* című 28. fejezettel zárul. Előbbi szerzői Lakatos László (ELTE), Szeidl László (Széchenyi István Egyetem) és Telek Miklós (BME), utóbbié pedig Vizvári Béla (Keleti Mediterrán Egyetem).

A LaTeX style fájlt Belényesi Viktor, Csörnyei Zoltán, és Iványi Antal írták. Az ábrákat a szerzők, Csergő Bálint, Iványi Antal jr. rajzolták. Az irodalomjegyzék ugrópontjait Iványi Anna Barbara lektorálta és egészítette ki.

A 17. fejezetet Roszik János (Debreceni Egyetem), a 21. fejezetet Lencse

Zsolt, a 22. fejezetet Ruff Laura (Babeş-Bolyai Tudományegyetem), a 25. fejezetet pedig Pintér Miklós (Corvinus Egyetem) fordították.

A kolofon oldalon lévő ugrópontok segítségével az Olvasók kapcsolatba léphetnek a kötet alkotóival. Új gyakorlatokat és feladatokat, valamint a tartalomra vonatkozó észrevételeket köszönettel fogadunk.

A nyomtatott első két kötet és az elektronikus első kötet megjelenését az Oktatási Minisztérium,, a harmadik nyomtatott kötet megjelenését a Nemzeti Kulturális Alap, ezeknek az elektronikus kötetnek a megjelenését pedig a Magyar Tudományos Akadémia, a Neumann János Számítógéptudományi Társaság támogatta. Mindhárom kötet megjelent angolul is, mind nyomtatott, mind pedig elektronikus formában. Ezek megjelenését az Európai Unió (az Európai Szociális Alap társfinanszírozásával), a Magyar Tudmányos Akadémia és a Nemzeti Kulturális Alap támogatta.

A későbbiekben szeretnénk az eddigi nyomtatott kés elektronikus kiadások hibáit kijavítani. Ezért kérjük a könyv Olvasóit, hogy javaslataikat, észrevételeiket küldjék el a `tony@inf.elte.hu` címre – levelükben lehetőleg pontosan megjelölve a hiba előfordulási helyét, és megadva a javasolt szöveget. A javított kiadásokban közreadjuk minden hiba első felfedezőjének a nevét.

Olvasóink javaslataikkal, kérdéseikkel megkereshetik a könyv alkotóit is (címük megtalálható a kolofonoldalon).

Budapest, 2015. január

Iványi Antal (tony@inf.elte.hu)

II. PÁRHUZAMOS MÓDSZEREK ÉS OPTIMALIZÁCIÓ

16. GPGPU: Számítások grafikus processzorokon

A ***GPGPU*** angol betűszó a *General-Purpose computing on Graphics Processing Units* kifejezést takarja, vagyis általános célú számítások grafikus feldolgozóegységeken történő elvégzését jelenti. A grafikus processzorok (***GPU-k***) erősen párhuzamos, többszálú, sokmagos feldolgozóegységek, amelyek számítási teljesítménye sokkal nagyobb mint a hagyományos CPU-ké. Valaha ezeket az egységeket kizárólag a megjelenítés és a grafika céljaira tervezték, és csak a grafikus API-k segítségével voltak programozhatók. Mára azonban a GPU-k általános célú párhuzamos processzorokká váltak, amelyeket bármely programozó számára elérhető felületeken keresztül és ismert nyelveken – például C-ben – lehet programozni.

Gyakran egy alkalmazás GPU-ra történő átültetésével nagyságrendi gyorsulást lehet elérni még az optimalizált CPU implementációhoz képest is. Ez a különbség a GPU-k hatalmas lebegőpontos műveletvégző teljesítményének, illetve az elérhető nagy memória-sávszélességnek köszönhető. Ezek a kedvező tulajdonságok abból erednek, hogy a GPU-t műveletintenzív, erősen párhuzamos számításokra tervezték, hiszen a grafikus megjelenítéshez pontosan erre van szükség. Így több tranzisztort szenteltek az adatfeldolgozásnak – a gyorsítótárak (cache) és a programvezérlés (flow control) rovására. Fejlesztői szempontból ez azt jelenti, hogy a hardver késleltetéseket nincs ami elrejtse, így ezekkel közvetlenül kell számolni. Így hatékony GPU program írása nem képzelhető el az architektúra ismerete nélkül.

Azért is van értelme a GPGPU-ról, mint az informatika önálló területéről beszélni, mert bár a GPU tekinthető párhuzamos rendszernek, architektúrája mégsem tiszta megvalósítása a párhuzamos számítási modelleknek (A könyv külön fejezete foglalkozik a párhuzamos számításokkal). Ehelyett számos modell tulajdonságait ötvözi: csővezetéknek (pipeline), vektor-, illetve tömbprocesszornak (kiSingle-Instruction Multiple-Data, azaz ***SIMD*** gépnek) adatfolyam-feldolgozó processzornak, osztott memórián keresztül kommunikáló többprocesszoros rendszernek egyaránt használható, amelyben speciális rögzített funkciójú hardverelemek is jelen vannak. Így, ha erre a különleges architektúrára tervezünk algoritmust, azt számos különböző modellhez kötődő fogalom és módszer felhasználásával alkothatjuk meg.

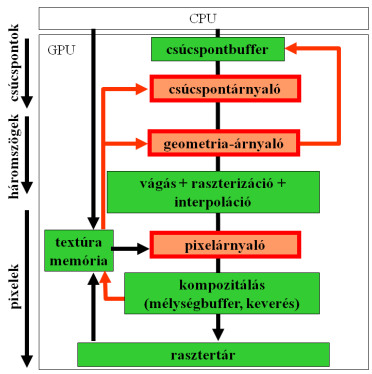
A grafikus kártya a programozható feldolgozóegységek mellett többféle memóriát és gyorsítótárat tartalmaz, valamint bizonyos grafikai feladatok vég-

rehajtására képes rögzített funkciójú egységeket is. A grafikus kártyát befogadó (host) számítógép központi processzorán (CPU) futó program a harver működését annak alkalmazásprogramozói felületén (API) keresztül vezérli. A központi processzor programokat tölt fel a GPU egységekre illetve adatokkal látja el őket. Ezeket a programokat számos nyelven meg lehet írni és a forrásnyelvről gépi kódra fordítani. A ***Cg*** [218] és a ***HLSL*** [203] nyelveket kifejezetten grafikára tervezték, de általános programozási nyelvekhez is léteznek a GPU programozáshoz kiterjesztések (például a CUDA C). A programozói környezet egyúttal meghatározza a ***programozási modellt,*** vagy másképpen a ***virtuális párhuzamos architektúrát,*** ami a programozható és a rögzített funkciójú elemeket, valamint azok kapcsolatait képezi le. Érdekes módon a különböző programozási modellek jelentősen eltérő virtuális párhuzamos architektúrákat láttatnak (16.1. ábra). A grafikus API-k a GPU-t mint egy csővezetéket jelenítik meg, amelynek az elemei folyamprocesszorként dolgozzák fel az adatokat, mivel ez illeszkedik legtermészetesebben a legtöbb grafikai feladathoz.

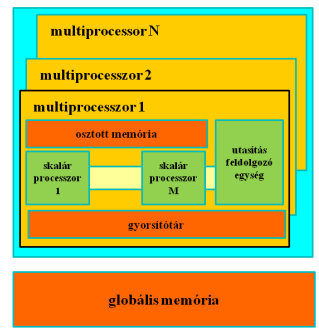
A ***CUDA*** [219] és az ***OpenCL*** [153] viszont azon az értelmezésen alapul, hogy a GPU multiprocesszorok halmaza, ahol minden multiprocesszor egy skaláregységekből álló, széles SIMD feldolgozóegység, ami ugyanazt a műveletet egyszerre több adaton hajtja végre. Ma a multiprocesszorok száma egy rendszerben akár néhány száz is lehet, míg egyetlen multiprocesszor általában 8 vagy 16, az utasításdekóderen osztozó skaláregységet tartalmaz.

A skalárprocesszorok teljes száma a multiprocesszorok számának és a SIMD skalárprocesszorok multiprocesszoronkénti számának szorzata, így könnyedén lehet ezernél is több. Ez a sok processzor mind ugyanazt a programot hajtja végre, de különböző adatokon. A program egy végrehajtását ***szálnak*** (thread) nevezzük. Egy multiprocesszor egy ***szálblokkot*** (thread block) futtat, vagyis az azonos multiprocesszoron futó szálak halmaza a blokk. Minden processzor rendelkezik valamennyi gyors, lokális memóriával, amely csupán a multiprocesszoron futó, tehát azonos blokkban lévő szálak számára elérhető. A globális memória sebessége a lokális memóriáénál sokkal alacsonyabb. A CPU vezérlő program a globális memóriába töltheti fel és innen töltheti le a bemenő és kimenő adatokat. Ezt a memóriát a multiprocesszorok többféle gyorsítótárazási és szinkronizációs stratégia igénybevételével is elérhetik. A CPU-val összevetve jóval kevesebb a gyorsítótárakra szánt tranzisztormennyiség, ezért összességében a gyorsítótár teljesítménye a CPU-étól elmarad. A programozó kreativitásán múlik, hogy a memóriaarchitektúrát milyen hatékonyan használja.

A fenti architektúra kompakt adatokat feldolgozó, rövid, koherens számítások párhuzamos végrehajtását szolgálja legjobban. Így algoritmu-



Grafikus API programozási modell



CUDA programozási modell

16.1. ábra. GPU programozási modellek: grafikus árnyalók és az általános célú CUDA. A grafikus árnyalókat a Shader Model 4 szabványnak megfelelően ábrázoltuk, amelyben a programozható egységek piros színnel, a rögzített funkciójúak pedig zölddel jelennek meg.

saink GPU-ra adaptálásának fő kihívása az, hogy ezeket sokezer szálra párhuzamosítsuk, illetve független és lehetőleg rövid számítási lépésekre bontsuk. Az ilyen számítási lépeseket megvalósító, GPU-n futó programokat gyakran ***kernelnek*** vagy ***árnyalónak*** nevezik. A kernel név a párhuzamos adatfeldolgozási vetületre utal, míg az árnyaló név egy alapvető grafikai feladat öröksége, amely a tárgyak felületére érkező fény visszaverődését szimulálja, vagyis az árnyal (shading).

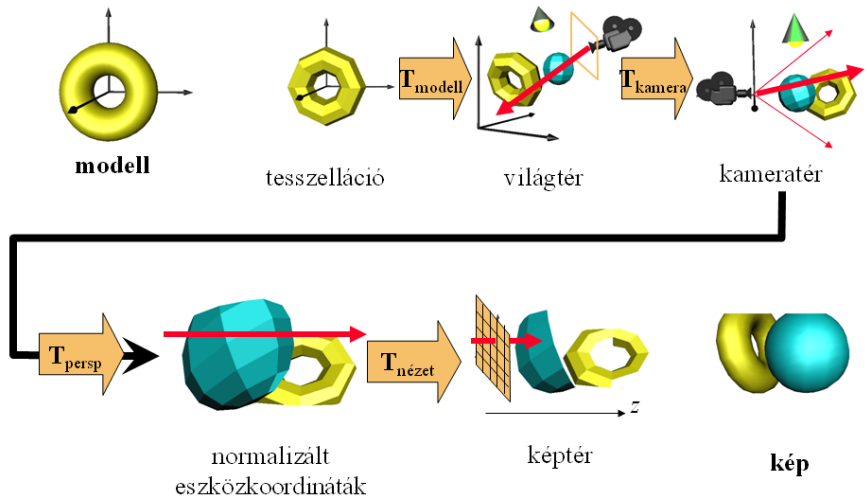
Bár a GPU-programozási nyelvek és vezérlésre használható API-k képességeikben és nyelvi elemeikben is egyre hasonlóbbak lettek, még mindig különválnak a grafikai és az általános célú megoldások. A két megközelítés két különböző programozói szemléletet takar. A grafikus API-k a korábban nagyon erősen korlátozott, de tisztán párhuzamos árnyalóprogramokból igyekeznek rugalmas számítóeszközöket faragni. A GPGPU keretrendszerek az általános programnyelvekhez tesznek hozzá a hagyományos kód párhuzamos futtatását lehetővé tevő elemeket. A grafikus megközelítés tűnhetne túlhaladottnak, vagy mondhatnánk, hogy csak a grafikához közeli területeken indokolt, de valójában a lényegének nincs köze a grafikához. A lényeg ugyanis az a benne foglalt ismeret, hogy hogyan is működnek az olyan hatékony párhuzamos algoritmusok, mint amilyen az inkrementális képalkotási csővezeték. Ezért tárgyaljuk először ezt a csővezetéket és bemutatjuk, hogy a grafika programozó szemével milyennek is látjuk a GPU-t. Ezzel nem-

csak az egyes alkatrészek célját és működését tisztázzuk, hanem egyúttal egy érvényes, kipróbált és általános célra is használható GPU programozási megközelítést is adunk. Ezután térünk át a GPGPU megközelítésre, a magasabb absztrakciós szint kedvéért elhagyva minden grafikával kapcsolatos fogalmat és lemondva a grafika feladataihoz rendelt kötött működésű hardverelemekről.

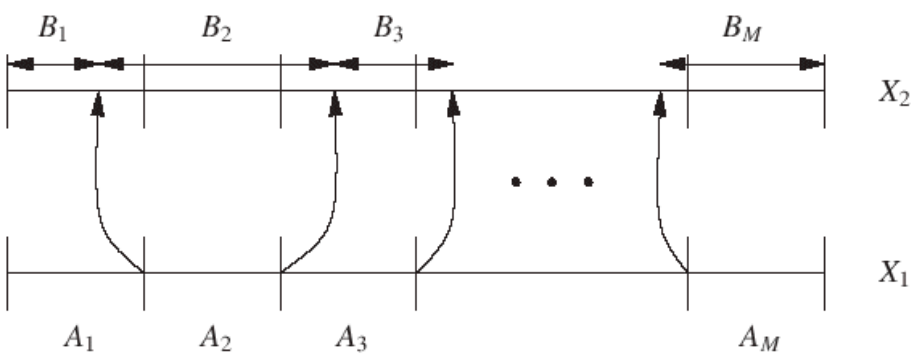
16.1. A grafikus csővezeték modellje

A grafikus csővezeték modellje a GPU hardvert azon az absztrakción keresztül szemléli, hogy az egy ***inkrementális képalkotást*** [285] (lásd a *Számítógépes grafika* című fejezetet a harmadik kötetben) végrehajtó eszköz. Az inkrementális képalkotás célja, hogy lefényképezzen egy virtuális világot. Ehhez a világot leíró numerikus modellt lineáris primitívekké (pontokká, vonalakká, háromszögekké) alakítja és ezeket egy diszkrét kép képpontjaira bontja, más szóval raszterizálja. A folyamat számos algoritmikus lépésből áll, ezek csoportjai alkotják a csővezeték szakaszait (pipeline stages). Néhány szakaszt dedikált hardverkomponensek valósítanak meg, míg másokat a GPU-n futó programok. A részletek mellőzésével, tekintsük át a képalkotás folyamatát (16.2. ábra):

- A virtuális világ modellpéldányok gyűjteménye. A modelleket háromszöghálókkal közelítjük. Ez a folyamat a ***tesszelláció.***
- Az árnyalás végrehajtásához az objektumokat abba a térbe kell transzformálnunk, amelyben a kamera és a fényforrások is adottak. Ez lehet a ***világtér*** vagy a ***kameratér.***
- A háromszögek csúcspontjait a kamerabeállításoknak megfelelően a képernyőre vetítjük. Azt, hogy a csúcspont hol látszik a képernyőn, a ***kamera transzformáció,*** a ***perspektív transzformáció*** és végül a ***nezetablak@nézetablak transzformáció*** végrehajtásával számítjuk ki. Kameratérben a kamera az origóban van és a $-z$ tengely irányába néz. A szempozícióból induló, a képernyőnek megfeleltetett ablak pontjain áthaladó sugarak egy perspektív köteget alkotnak. A perspektív transzformáció szerepe, hogy ezt a perspektív köteget párhuzamos sugárköteggé alakítsa és ezzel a következő lépésben szükséges perspektív vetítést párhuzamos vetítésre cserélje. A perspektív transzformáció után a csúcspontok normalizált eszköz-koordinátákban állnak elő. Ebben a térben a látható térfogatot a $-1 \leq x \leq 1$, $-1 \leq y \leq 1$, $-1 \leq z \leq 1$ egyenlőtlenségekkel meghatározott téglatest adja meg. A geometriai primitíveknek ezen a térfogaton kívül eső részeit a ***vágás*** művelete távolítja el.



16.2. ábra. Az inkrementális képalkotás folyamata.



16.3. ábra. Munkaoptimális összefésülő algoritmus.

A normalizált eszköz-koordinátákat tovább transzformáljuk a képtérbe, a célkép felbontását és elhelyezkedését is figyelembe véve. Az $x = -1$, $y = -1$ normalizált eszköz-koordinátájú pontot a nézeti ablak bal alsó sarkának képernyőn elfoglalt helyére képezzük, az $x = 1$, $y = 1$ pontot pedig

a jobb felsőére. Eközben a z koordinátát a $-1 \leq z \leq 1$ tartományból a $[0, 1]$ tartományba konvertáljuk.

- A képtérben minden vetített háromszöget ***képpontok*** (pixelek) egy halmazára bontunk. Egy belső pont kitöltésekor annak tulajdonságait – köztük a z koordinátát, vagyis a ***mélységet*** – a csúcspontok adataiból inkrementális lineáris interpolációval számoljuk. Minden képpont árnyalási színe ezekből az interpolált adatokból számolható, például a csúcspontokban számolt színek interpolált változata lehet a pixelszín. Emellett vagy ehelyett ***textúrának*** nevezett képeket is tapétázhatunk a háromszöghálókra. A textúraképek színrekordok kétdimenziós tömbjei, egy elemüket ***texelnek*** hívjuk. Azt, hogy a textúrát mi módon képezzük a háromszögek felületére, a csúcspontokhoz rendelt textúrakoordinátákkal adhatjuk meg.
- Végül a képpont színeket a képernyőn megjelenített ***rasztertár*** (frame buffer) írjuk. A rasztertár mellett egy ***mélységbufferre*** (depth buffer, z-buffer, depth-stencil texture) is szükség van. Ez tárolja a mélységinformációt, vagyis azon felületi pontok z koordinátáit, amelyek színét a rasztertár pixelei tárolják. Amikor újabb háromszöget raszterizálunk egy pixelre, a szín- és mélységinformációt csak akkor írjuk felül, ha az új mélység a mélységbufferben tároltnál kisebb, vagyis az új háromszög ebben a képpontban közelebb van a nézőponthoz. Ennek eredményeképpen egymást helyesen takaró háromszögek 3D képét kapjuk. A ***mélységbuffer algoritmus*** egyben példa arra az általánosabb műveletre, amikor a képpontban tárolt adatot valamilyen új adat és a korábban a képpontban tárolt adat függvényeként határozzuk meg. Az ilyen műveletek neve ***kompozitálás*** (merging).

16.1.1. A GPU mint az inkrementális képalkotás megvalósítása

A grafikus API által elénk tárt GPU architektúra a képalkotási csővezeték közvetlen megvalósítása (16.1. ábra, bal oldal). Ezt a csővezetéket a CPU a grafikus API hívásain keresztül konfigurálhajta, majd működését egy ***rajzoláshívás*** (draw call) révén indíthatja el. Az egymás utáni rajzoláshívások sorozatát, amelyek során a konfiguráció nem, csupán a bemenetek változnak, ***menetnek*** (pass) hívják. Egy rajzoláshívás csúcspontok sorozatát dolgozza fel. A csúcspontokhoz tartozó adatokat a csúcspontbuffer (vertex buffer) tárolja.

A csúcspontok megadása modellezési térben, homogén koordinátákkal történik. Az (x, y, z) ***Descartes-koordinátájú*** pont az $[xw, yw, zw, w]$ négyelemű ***homogén koordinátavektorral*** is megadható, ahol w egy

tetszőleges nemzérus skalár lehet (részletesebb magyarázatért lásd a számítógépes grafikáról szóló fejezetet). Ennek a reprezentációnak a neve onnan származik, hogy a számnégyes minden elemének ugyanazzal a skalárral való szorzása nem módosítja a kijelölt háromdimenziós pontot. Az $[X, Y, Z, w]$ homogén számnégyesből ugyanennek a pontnak a Descartes-koordinátáit a ***homogén osztás*** segítségével kaphatjuk meg: $(X/w, Y/w, Z/w)$. A homogén koordinátáknak számos előnyös tulajdonsága van a Descartes-koordinátákhoz képest. Homogén koordinátákkal még a párhuzamosok metszéspontja is kifejezhető (egy ***ideális pont***), vagyis az euklideszi geometria párhuzamosok okozta szingularitása kiküszöbölhető. A homogén lineáris transzformációk közé tartozik a ***perspektív vetítés,*** aminek nagyon fontos szerepe van a képalkotásban, de Descartes-koordinátákban nem lenne lineáris transzformáció. Ami igazán fontos, hogy az egyeneseket egyenesekbe, síkokat síkokba, háromszögeket háromszögekbe leképező transzformációk halmaza megegyezik a homogén koordinátákon végzett lineáris műveletek halmazával.

A csúcspontbuffer beállítása után a koordinátáikkal és más attribútumaikkal – például textúrakoordinátákkal vagy saját színnel – leírt csúcspontok megkezdik útjukat a grafikus csővezetékben, a programozható árnyalóprocesszorok és a rögzített funkciójú elemek révén megvalósított feldolgozószakaszokon keresztül. Ezen szakaszokat egyesével ismertetjük.

Tesszelláció

Ha a csúcspontjaink nem közvetlenül a végső háromszögháló adják meg, hanem csupán egy parametrikus felület vezérlőpontjai, vagy a háromszögháló durvább verziójának csúcsai, akkor az első lépés a végső háromszögháló előállítása, vagyis a . Mivel ennek a szakasznak a programozhatósága korlátozott és a GPGPU alkalmazhatósága nem különösebben jelentős, nem tárgyaljuk tovább, inkább feltételezzük, hogy a csúcspontbuffer a tesszellációt nem igénylő háromszögháló csúcsait tartalmazza.

A csúcspontárnyaló

Az objektumokat a vágás előtt normalizált eszköz-koordinátákba kell transzformálni, ezt pedig egy homogén lineáris transzformációval érhetjük el. Ezen felül a GPU feladata lehet a megvilágítás számítása a háromszögháló csúcspontjaiban. Ezeket a műveleteket a ***csúcspontárnyaló*** hajtja végre. Általánosabban tekintve, a csúcspont-árnyaló egység egyszerre egy csúcspont adatait kapja meg, módosíthatja őket új pozíció-, szín- és textúrakoordináta-értékek kiszámításával, és így egy megváltoztatott csúcspontot ad ki. A csúcspontok feldolgozása egymástól függetlenül, párhuzamosan történik.

A geometria-árnyaló

A geometria-árnyaló a csúcspontrekordok mellett a primitívekről is kap információt: például mely csúcsok alkotnak egy háromszöget a háromszöghálóban. Az árnyaló továbbíthatja a primitíveket, de új csúcspontokat, új primitíveket is létrehozhat. Lehetőség van arra, hogy ezeket egy kimenő bufferbe írjuk, amit későbbi menetek bemenetként használhatnak. A geometria-árnyaló tipikus felhasználási területe a procedurális modellezés, amely szabályok alapján összetett modellek alkot egyetlen pontból vagy háromszögből kiindulva [190].

A csúcspontárnyalók kis, specializált egységekből általános adatfolyam-feldolgozó processzorokká fejlődtek, mialatt megtartották az egy bemenet – egy kimenet sémát. A geometria-árnyaló azonban a csúcspont-árnyaló kimenő rekordjain (a feldolgozott csúcspontokon) dolgozik és változó (de korlátos) mennyiségű hasonló rekordot ad ki.

Vágás

A során a hardver a primitíveknek csak a normalizált eszköz-koordinátákban $(-1,-1,-1)$ és $(1,1,1)$ átellenes csúcsokkal jellemezhető, koordinátasíkokkal párhuzamos oldalú kockán belüli részét tartja meg. Egy pont akkor esik ezen belülre, ha homogén koordinátái kielégítik a következő egyenlőtlenségeket:

$$-w \leq x \leq w,\ -w \leq y \leq w,\ -w \leq z \leq w\ .$$

A harmadik egyenlőtlenség speciálisan az OpenGL konvencióihoz igazodik. Akkor érvényes pl. a Cg nyelvben, ha OpenGL csúcspontárnyaló-profilra illeszkedő gépi kódot fordítunk belőle. Az utolsó egyenlőtlenségpár lehetne $0 \leq z \leq w$ is, ahogy a Direct3D feltételezi. Ez érvényes Cg-ben Direct3D profilok esetén, illetve a HLSL szabványban. A különbséget a fordítóprogramok elrejtik, mivel a csúcspontárnyaló kimenetét (a használt konvenció, vagyis a profil ismeretében) olyanra alakítják, amilyent a vágást végző hardver vár.

A vágást a GPU rögzített funkciójú hardvere hajtja végre, így működését nem lehet megváltoztatni. Azok a primitívek folytathatják útjukat a csővezeték további szakaszai felé, amelyek a vágási feltételeket teljesítik. Párhuzamos programozási szemszögből a vágó hardver egy ***adatfolyam-szűrő***. Ha el szeretnénk hagyni egy, a csúcspont- vagy geometria-árnyaló által feldolgozott adatelemet, a csúcspontok pozícióját úgy kell beállítanunk, hogy azok a vágási kockán kívül essenek. Így a vágó hardver törli az elemet a csővezetékből.

A vágás után a művelete következik, vagyis a homogén koordinátákat Descartes-koordinátákká alakítjuk úgy, hogy az első három homogén koordinátát a negyedikkel elosztjuk. A csúcspontokat ezután képtérbe transzformáljuk, ahol az első két koordináta azt mondja meg, melyik képpontra esik

a csúcspont.

Raszterizáció lineáris interpolációval

A csővezeték szíve a nem-programozható raszterizáló szakasz. Ez képes a lineáris primitíveket (háromszögeket, szakaszokat, pontokat) a képernyő képpontjaira illeszkedő diszkrét pixelekké alakítani. Egyszerűbben fogalmazva, háromszögeket képes rajzolni a csúcspontok képtérbeli koordinátái alapján. A csővezeték raszterizáló előtti szakaszai ezeket a csúcspont-koordinátákat számolják ki, a későbbi szakaszok feladata pedig a képpont színek megállapítása lesz.

Bár így a raszterizáció kapcsán a csővezeték minden szakaszának meg lehet fogalmazni az alapfeladatát, a GPGPU alkalmazások nem feltétlenül igénylik háromszögek rajzolását. A raszterizálót mégis lehet párhuzamos programozási szempontból is értelmezni, mégpedig adatfolyamsokszorozóként (stream expander), hiszen minden primitív számítás nyomán független pixelszámítások sokaságát indítja el. Ehhez csak a háromszögeket kell ügyesen megkonstruálni.

A raszterizáció képtérben dolgozik, ahol a csúcspontok x, y koordinátái azon egész képpont-koordináták, ahova a képen a csúcsok vetülnek. A csúcspontokhoz további jellemzők is tartozhatnak, mint például a képtérbeli z koordináta, textúrakoordináták vagy színek. Egy háromszög raszterizációjakor mindazokat a képpontokat azonosítjuk, amelyek a háromszög vetített képén belülre esnek. Az egyes képpontok tulajdonságait lineáris interpolációval határozzuk meg.

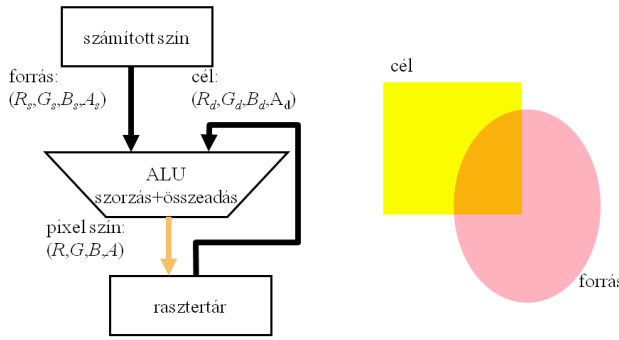
A pixelárnyaló

A csúcspontokhoz rendelt értékekből interpolációval állnak elő a képpontok tulajdonságértékei, és ezek alapján kell meghatározni a pixel színét, illetve módosítható a hozzá tartozó mélységérték is. A szín meghatározásának klasszikus módja az, hogy az interpolált textúrakoordináta alapján kiolvassuk a textúramemóriát, a kapott eredményt pedig az interpolált színnel moduláljuk.

Általában egy pixelárnyaló program a pixelhez tartozó interpolált értékeket kapja bementként, és kimenetként egy színt, illetve opcionálisan egy mélységet adhat ki. A csúcspontárnyalóhoz hasonlóan a pixelárnyaló is „egy bemeneti rekordhoz egy kimeneti rekord”’ típusú processzor. A pixelárnyaló a célképponthoz van kötve, a kimenetét sehova máshova nem tudja átirányítani.

Kompozitálás

A kiszámított képpontszínek nem közvetlenül kerülnek a képmemóriába, hanem a ***kompozitáló egység*** (output merger) felelős az új értékek és



16.4. ábra. Keverőegység, amely az új pixelszínt a számított és a korábban tárolt színek kombinálásával állítja elő.

a korábbi tartalom összeolvasztásáért. Először a mélységbuffer alapján a mélységtesztet hajtjuk végre. Megjegyzendő, hogy amennyiben a pixelárnyaló a z értéket nem módosítja, ez a művelet előrehozható a pixelárnyalás elé. Ez a ***korai z-teszt*** jobb teljesítményhez vezethet a felesleges pixelfeldolgozások elhagyása miatt.

Végül a kompozitáló kombinálja – például lineárisan, vagy minimumot, maximumot képezve – a kiszámított képpont színt a már meglevő pixel színnel, és az eredménnyel felülírja a rasztertárban tárolt színt. Ez a ***keverés*** (blending), amivel például átlátszó felületek megjelenítése lehetséges (16.4. ábra).

GPGPU céljaira a keverés főleg akkor hasznos, ha egymást követő számítások eredményeinek összegét, minimumát vagy maximumát keressük.

16.2. GPGPU a grafikus csővezeték modelljének felhasználásával

Az általános célú programozásban olyan fogalmakat szoktunk meg, mint például a bemenő és kimenő adat, valamint az ideiglenes és a kimenő adatokat előállító, paraméterezhető függvények. Ha a GPU-t a grafikus API-n keresztül szeretnénk használni, akkor a programozói fogalmainkat meg kell feleltetni az inkrementális képalkotás fogalmainak, köztük a geometriai primitíveknek,

csúcspont, geometria és képpont feldolgozásnak, raszterizációnak, textúrázásnak, kompozitálásnak és a kimeneti képnek. Számos különböző lehetőség van arra, hogy ezt a megfeleltetést megtegyük, melyek előnyei és hátrányai a konkrét algoritmustól függenek. Itt néhány olyan általános megközelítést tárgyalunk, amelyek sikeresnek bizonyultak teljesítményigényes számítási alkalmazásokban. Először megvizsgáljuk, hogy miként rokoníthatók a programozási fogalmaink a GPU elemeihez.

16.2.1. A kimenet

A GPU-k képeket rajzolnak, vagyis kétdimenziós képponttömböket állítanak elő. A ***rajzolási cél*** (render target) lehet a felhasználó felé megjelenített rasztertár, de lehet egy kimeneti textúra is (amely esetben a képpont (pixel) helyett textúraelemről, texelről szokás beszélni). A GPGPU esetében a kimenet jellemzően egy textúra, amelyben a fixpontos számábrázolású rasztertárral szemben lebegőpontos értékek is tárolhatóak. Ennél is fontosabb, hogy a textúrák a későbbi számítási lépésekben bemenetként használhatóak, vagyis a korábbi kétdimenziós kimenti textúra akár kétdimenziós bemeneti textúraként, vagy többet összefogva háromdimenziós bementi textúraként is szerepelhet.

Korábbi GPU-knál minden képponthoz legfeljebb öt lebegőpontos számot tudtunk tárolni, mivel a színt a vörös, zöld, kék és átlátszatlanság értékek írták le, a mélységbufferben pedig egy skalár mélységérték, a z képtérbeli koordináta szerepelhetett. Később a ***több rajzolási cél*** (multiple render targets) megjelenésével a képponthoz több, jellemzően négy textúrában is tartozhatott érték, amivel a legnagyobb kimeneti rekord már tizenhét lebegőpontos számot tudott tárgyalni. A Shader Model 5.0-ös GPU-k esetében ez a korlátozás is megkerülhető, a képpontokhoz akár változó hosszúságú adatlistákat is ki tudunk írni.

A geometriai elemek határozzák meg, hogy mely képpontokat veszi célba a rajzolási folyamat. Minden primitívet a képtérbe transzformálunk és a vetített képét raszterizáljuk, tehát azok lesznek a kimeneti képpontok, amelyek a vetületen belülre esnek. Ha több primitívet is átküldünk a csővezetéken, azok vetületei átlapolódhatnak, így egy képpontra több számítást is végzünk. A kompozitáló kombinálja ezeket az eredményeket, például csak a legkisebb képtérbeli mélységűt tartja meg a mélységteszt révén, vagy összegzi a részeredményeket a keverés segítségével.

A rajzolási célként kijelölt memóriát csak a pixelárnyaló írhatja a kompozitálási műveleten keresztül, miközben közvetlenül egyetlen árnyalóprocesszor sem olvashatja. Mivel különböző pixelárnyaló szálak különböző helyekre írnak a rajzolási célban, szinkronizációs probléma nem merülhet fel.

16.2.2. A bemenet

A képszintézis bemenetei a geometriát leíró adatfolyam és a felületek színét kódoló textúrák. Mivel a háromszöghálónak általában nincs konkrét szerepe GPGPU alkalmazásokban, a geometriát leíró adatokat csupán vezérlőmechanizmusként használjuk arra, hogy elosszuk a számítási terhelést az árnyalóprocesszorok között. A valódi GPGPU bemenet a textúrákban tárolt adat lesz. A textúra texelek egy-, két- vagy háromdimenziós tömbje, ahol egy texel egy, kettő vagy négy skalár értéket tárol a vörös, zöld, kék és átlátszatlanság csatornákban. A számértékek különböző formátumúak lehetnek, például előjel nélküli bájt vagy 32 bites lebegőpontos szám. GPGPU szempontból többnyire a 32 bites lebegőpontos ábrázolás a legmegfelelőbb.

Az egydimenziós lebegőpontos textúra hasonló a lineáris CPU memóriához, ahol a szokásos adatszerkezeteket, tömböket, listákat, fákat ugyanúgy lehet tárolni. A kétfajta memória hasonlósága azonban két ponton megtörik. Az egyik tekintetben a textúra gyengébb, a másik tekintetben viszont erősebb, mint a lineáris CPU memória.

Nyilvánvaló korlátozás az, hogy a textúra párhuzamosan bár, de csak olvasható minden programozható árnyaló számára, kivéve a rajzolási céltextúrát, ami viszont csak a kompozitáló egység számára írható. A CPU memóriában megszokott olvasás-módosítás-írás ciklust árnyalóprogramban nem használhatjuk. A GPU-tervezőknek jó oka volt rá, hogy ne tegyék lehetővé az olvasás-módosítás-írási ciklusokat, és hogy a textúrákat párhuzamosan olvasható, illetve kizárólagosan írható szerepekbe korlátozzák, hiszen így az írást soha nem kell gyorsítótárazni és az olvasási gyorsítótárak soha nem válnak érvénytelenné.

Másrészről viszont a textúramemória sokkal több címzési módot támogat, mint a lineáris memória, és ami még fontosabb, ***textúraszűrő*** hardveregységeken keresztül is olvasható. A szűrőkkel a textúra nemcsak elemek egy tömbje lehet, de egy-, két-, vagy háromdimenziós térbeli függvény végeselemes reprezentációja is (a 16.7. alfejezet fogja a végeselemes reprezentáció és textúrák közötti kapcsolatot bővebben tárgyalni).

Egydimenziós textúrákhoz lineáris szűrés alkalmazása azt jelenti, hogy ha az u textúra koordináta az U és $U+1$ texelkoordináták közé mutat, akkor a hardver automatikusan lineárisan interpolál a két texel értéke között. Legyenek a texelek értékei $T(U)$ és $T(U+1)$. Az u helyen visszaadott szűrt érték ekkor

$$T(u) = (1-u^*)T(U) + u^*T(U+1), \quad \text{ahol} \quad u^* = u - U\ .$$

A kétdimenziós textúrákon ***bilineáris szűrést*** használhatunk, az interpolált (u,v) textúrakoordináta-párhoz legközelebb eső négy texelérték között.

Legyenek ezek $T(U,V), T(U+1,V), T(U+1,V+1)$ és $T(U,V+1)$. Az (u,v)-re visszaadott szűrt érték ekkor

$$T(u,v) = T(U,V)u^*v^* + T(U+1,V)(1-u^*)v^* +$$

$$T(U+1,V+1)(1-u^*)(1-v^*) + T(U,V+1)u^*(1-v^*) \ ,$$

ahol $u^* = u - U$ és $v^* = v - V$.

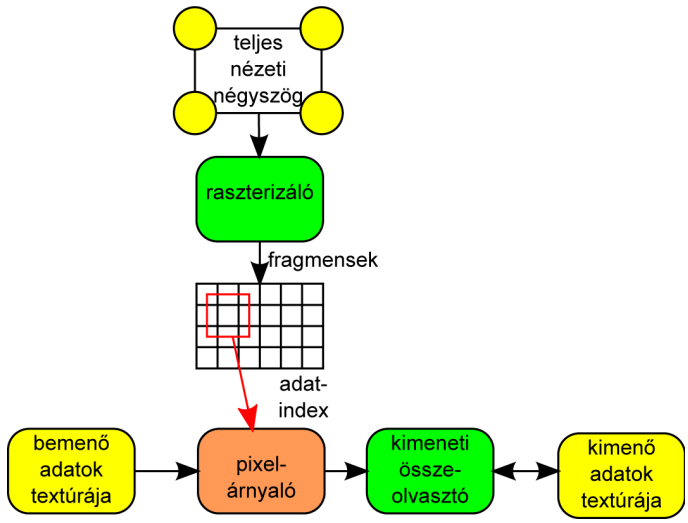
Háromdimenziós textúrákra alkalmazható ***trilineáris szűrés*** hasonlóképpen működik.

16.2.3. Függvények és paramétereik

Ahogy a primitívek végighaladnak a csővezetéken, árnyalóprocesszorok és rögzített funkciójú elemek dolgozzák fel őket, hogy végül meghatározzák az egyes képpontokhoz tartozó értékeket. Az árnyalóprocesszorok programjai nem változnak egy rajzolási menetben, így azt mondhatjuk, hogy minden egyes képpontot ugyanaz a program számít ki. A képpontok színeinek különbözősége az adatok különbözőségén múlik. Így a GPU olyan hardver, ami rekordok egy tömbjét számolja ki.

A GPU-n az adatok egy sor programozható vagy fix algoritmust végrehajtó processzoron haladnak keresztül amíg a kimeneti képpontok elő nem állnak. Ez azt jelenti, hogy a GPU-t lehet ***adatfolyam-feldolgozónak*** (stream processzor) is tekinteni. A primitíveket meghatározó csúcsok virtuális adatfolyamot alkotnak, amit elsőként a csúcspontárnyaló dolgoz fel. Az adatfolyam feldolgozás fogalomkészletét használva azt mondhatjuk, hogy a csúcspontárnyaló egy ***leképezés,*** mivel egy-egy csúcs adataira alkalmaz valamiféle függvényt és minden bemenetre egy módosított csúcspontadatot ad ki. Így a bemeneti és kimeneti adatfrekvencia azonos.

A geometria-árnyaló megváltoztathatja a primitív típusát és a csúcspontok számát. Az adatfrekvencia csökkenhet, ekkor a művelet neve ***redukció*** vagy ***adatfolyamszűrés***. Az adatfrekvencia növekedhet is, amikor ***adatfolyam-sokszorozásról*** beszélünk. A vágás megtarthat vagy eldobhat primitíveket, de ha azok részben lógnak be a vágási térfogatba, akkor meg is változtathatja őket. Ha ez utóbbi esettől eltekintünk, a vágás tekinthető adatfolyamszűrésnek. Ha a csúcsok koordinátáit a csúcspontárnyalóban úgy állítjuk be, hogy a vágási térfogaton kívülre essenek, ki tudjuk zárni a primitívet a további feldolgozási lépésekből. A raszterizálás a primitívet pixelekké bontja, így adatsokszorozóként működik. A pixelárnyaló a csúcspontárnyalóhoz hasonlóan ugyancsak leképezés. Végül a kompozitálás működhet ***kiválasztásként,*** például a mélységkoordináta alapján, vagy akár ***összegzésként*** is, ha a keverést használjuk.



16.5. ábra. A GPU mint vektorprocesszor.

Az árnyalóprocesszorok a bemenő adataikat regisztereken keresztül kapják meg, amelyeket az előző szakasz egységei töltenek fel. Az ilyen bemenetek neve ***változó bemenet*** (varying input). Ezek mellett a CPU programból is beállíthatunk globális paramétereket. Ezeket ***egységes bemenetnek*** (uniform input) hívják, mivel az adatfolyam minden elemére azonosak és a csővezeték működése közben nem, csupán a rajzolási menetek között változtathatók.

16.3. A GPU mint vektorprocesszor

Azt a párhuzamos gépet, amely a kimeneti értékeket ugyanazzal az algoritmussal, de egymástól függetlenül és köztes eredmények megosztása nélkül számítja, ***vektorprocesszornak*** vagy ***tömbprocesszornak*** nevezzük. Mivel a GPU hardverben a pixelárnyaló kapcsolható a kimeneti adatelemekhez, ezt az árnyalót használjuk a kimenet előállítására. Természetesen egy adott processzoron futó programnak tudnia kell, hogy melyik adatelemet számítja, alapvetően ebből adódik a számítás adatfüggősége (nem lenne értelme a párhuzamosításnak, ha ugyanazt számolnánk ki egyszerre több példányban). A pixelárnyaló számára az adatelem indexe nem más, mint a képpontot kijelölő koordináta pár.

A legegyszerűbb, de a gyakorlat számára a legfontosabb esetben a rajzolási cél minden képpontjára szeretnénk egy menetben eredményt kapni. Tehát olyan geometriai primitívet kell választanunk, ami a kép összes képpontjára leképződik, de mindegyikre csak egyszeresen, hogy elkerülhessük

egy kimeneti adat többszöri felesleges kiszámítását. Ilyen geometriai primitív maga a nézeti ablak. Képtérben ez nem más, mint a nézetablak (viewport), vágási térben (normalizált eszköz-koordinátákban) pedig a $[-1,-1,0,1]$, $[1,-1,0,1]$, $[1,1,0,1]$, $[-1,1,0,1]$ homogén koordinátás csúcsokkal rendelkező, x,y síkon elhelyezkedő négyzet. Ezt a két háromszögprimitívből összeállítható téglalapot hívják ***teljes képernyős téglalapnak*** vagy ***teljes nézeti téglalapnak*** (full-screen quad, full-viewport quad) (16.5. ábra).

Tegyük fel, hogy szeretnénk kiszámítani az N elemű $\mathbf{y}$ kimeneti tömböt az M elemű $\mathbf{x}$ bemeneti tömbből, a p globális paraméter mellett az F függvény segítségével:

$$\mathbf{y}_i = F(i, \mathbf{x}, p), \qquad i = 1, \ldots, N\ .$$

Ahhoz, hogy a GPU-t erre a számításra rábírjuk, rendeljük a rajzolási célként beállított kimeneti textúrát az $\mathbf{y}$ tömbhöz. A textúra méretét a kimeneti tömb mérete alapján válasszuk meg úgy, hogy a nézetablaknak le kell fednie a teljes rajzolási célt. Egy H vízszintes és V függőleges felbontással rendelkező kétdimenziós tömb $H \times V$ elem tárolására alkalmas. Tehát $H \times V \geq N$ betartása mellett elvileg bármilyen felbontást választhatunk. Azonban a GPU is rendelkezhet korlátozásokkal ezen a téren, például a felbontás legfeljebb $2^{12} \times 2^{12}$ lehet, vagy ha ugyanezt a textúrát később bemenetként is szeretnénk használni, vagy bináris redukciót kívánunk végrehajtani rajta, akkor előnyösebb lehet kettő hatvány felbontást választani. Ezt akár úgy is elérhetjük, hogy tömbjeinket nem használt elemekkel egészítjük ki.

A vektorfeldolgozás alapgondolatának megfelelően a különböző kimeneti értékek számítása egymástól függetlenül, köztes adatok megosztása nélkül zajlik. Mivel a GPU hardverben a pixelárnyaló végez a kimenet elemeihez kapcsolható független számításokat, ezt az árnyalót használjuk az F kiértékelésére. A paraméterek eléréséhez ismernünk kell az i indexet, vagyis hogy melyik kimeneti elemet számítjuk, és hozzá kell férnünk az $\mathbf{x}$ bemeneti tömbhöz. Ennek legegyszerűbb módja, hogy a tömböt egy bemenő textúrában tároljuk (vagy akár több textúrában, ha az kényelmesebb), hiszen a pixelárnyaló olvashat textúrákból.

A CPU feladata a céltextúra bekötése rajzolási célként, az egységes paraméterek beállítása, a nézetablak megadása és egy teljesnézetes téglalap csővezetékre küldése. Az egységes paramétereken jelölhetjük ki a bemenő textúrát és a p globális paraméter értékét. OpenGL API-t feltételezve a CPU programot C nyelven a következőképpen írhatjuk meg:

```
StartVectorOperation( ) {
    Az egységes p paraméter és a a bemeneti textúra arrayX azonosítójának beállítása

    glViewport(0, 0, H, V);     // Felbontás, azaz a kimeneti tömb mérete
    glBegin(GL_QUADS);          // A következő pontok egy téglalap csúcsai
        glVertex4f(-1,-1, 0, 1);   // normalizált eszköz-koordinátarendszerben
```

```
        glVertex4f(-1, 1, 0, 1);
        glVertex4f( 1, 1, 0, 1);
        glVertex4f( 1,-1, 0, 1);
    glEnd( );
}
```

Megjegyzendő, hogy a program közvetlenül normalizált eszköztérben, homogén koordináták használatával adja a *glVertex4f* függvény paramétereként a téglalap csúcsait. Így a csővezetékben ezeket már nem kell transzformálnunk.

Az árnyalóprogramok a változó bemenetet erre szánt regisztereikben kapják meg és a kimenethez rendelt regiszterekbe írják. Minden ilyen regiszter *float4* típusú, tehát négy lebegőpontos számot tárol. A regiszterek szerepüket jelölő nevekkel azonosíthatók, a csúcspont pozíciója például a ***POSITION*** regiszterből olvasható ki. A csúcspont színét a ***COLOR***, textúrakoordinátáit a ***TEXCOORD0..7*** regiszterek tárolják.

A csúcspontárnyaló feladata, hogy a csúcspont megkapott pozícióját normalizált eszköz-koordinátarendszerbe transzformálja. Mivel közvetlenül ebben a térben adtuk meg a csúcspontokat, a csúcspontárnyalónak csak át kell másolnia az értéket a bemeneti *POSITION* regiszterből a kimeneti *POSITION* regiszterbe. A kimeneti és bemeneti jelleget az ***in*** és ***out*** kulcsszavakkal jelezhetjük a regiszterekhez rendelt paraméterek előtt. A bemeneti pont pozícióját a kimenetre másoló csúcspont árnyaló:

vagy találkozunk

```
void VertexShader( in float4  inputPos   : POSITION,
                   out float4 outputPos  : POSITION )
{
    outputPos = inputPos; // változtatás nélküli másolás
}
```

A geometriaárnyalónak ugyanazt a téglalapot kell továbbadnia, amit maga kapott. Ez a viselkedés az alapértelmezett, ehhez nem kell programot megadnunk. A téglalap ezután a vágást végző egységhez ér, ami megtartja a téglalapot, hiszen a vágási térfogaton belül van. Vágás után a homogén osztás a homogén koordinátákból a Descartes-koordinátákat állítja elő. Mivel mind a négy csúcs negyedik homogén koordinátáját 1 értékűre állítottuk, az első három koordináta nem változik. Osztás után a GPU a téglalap csúcsait képtérbe transzformálja, amelynek során a csúcsok x, y koordinátái a nézetablak sarkaira kerülnek, a z tartomány pedig a $(-1, 1)$-ből a $(0, 1)$ intervallumra képződik le, így a $z = 0$ értékből $z = 0.5$ érték lesz. Végül ezt a téglalapot töltik ki a pixelek, tehát a nézetablak összes képpontjára lefut a pixelárnyaló.

A pixelárnyaló az igazi számítást végző egység. A bemenő tömböt és a p globális paramétert egységes bemenetként kapja meg, a ***WPOS*** regiszter alapján pedig azonosítja, hogy mely képpontot számolja éppen:

vagy találkozunk

```
float FragmentShaderF(
        in float2 index : WPOS,     // célpixel koordinátái
```

```
        uniform samplerRECT arrayX, // bemeneti tömb
        uniform float p             // globális p paraméter
        ) : COLOR   // a kimenet egy pixel szín, azaz tömbelem
{
    float yi = F(index, arrayX, p);  // F a kiértékelendő függvény
    return yi;
}
```

Ebben a programban két bemeneti paramétert deklaráltunk egységes bemenetnek az ***uniform*** kulcsszó segítségével: a p paramétert és az *arrayX* textúraazonosítót. A textúra típusa {bindexsamplerRECT, amivel azt is kijelöltük, hogy milyen címzéssel érhető el a textúra. Ebben a címzési módban a texelközéppontok kétdimenziós egységhálón helyezkednek el. Megjegyzendő, hogy itt a korábbiaktól eltérő szintaxist alkalmaztunk az árnyaló kimenetének magadására. Ahelyett, hogy a regisztert az *out* kulcsszóval jelöltük volna meg, itt a kimenet maga a függvény visszatérési értéke, ami a ***COLOR*** regiszterbe kerül.

16.3.1. A SAXPY BLAS függvény megvalósítása

Tekintsünk egy konkrét példát és valósítsuk meg a ***Basic Linear Algebra Subprograms*** (***BLAS***) könyvtár (http://www.netlib.org/blas/) első szintű funkcionalitását, ami a következő általános formában felírható vektorfüggvényeket képes kiértékelni:

$$\mathbf{y} = p\mathbf{x} + \mathbf{y}$$

ahol $\mathbf{x}$ és $\mathbf{y}$ vektorok és p skalárparaméter. Ezt a műveletet a BLAS könyvtárban ***SAXPY-nak*** hívják. Ezúttal a pixelárnyaló két textúrát fogad, az $\mathbf{x}$ vektort és az $\mathbf{y}$ vektor eredetijét.

Egy pixelárnyaló futtatás a kimeneti vektor egyetlen elemét számolja:

```
float FragmentShaderSAXPY(
        in float2 index : WPOS,      // célpixel koordinátái
        uniform samplerRECT arrayX,  // x bemeneti vektor
        uniform samplerRECT arrayY,  // az y vektor eredeti verziója
        uniform float p              // globális p paraméter
        ) : COLOR                    // a kimenet egy pixel szín, azaz vektorelem
{
    float yoldi = texRECT(arrayY, index);    // yoldi = arrayY[index]
    float xi    = texRECT(arrayX, index);    // xi = arrayX[index]
    float yi = p * xi + yoldi;
    return yi;
}
```

A CPU-stílusú programozás tömbindexelése helyett itt ***texRECT*** Cg függvényével olvasunk be egy elemet a tömböt reprezentáló textúrából. A *texRECT* függvény első paramétere a kétdimenziós textúra azonosítója, amit a CPU-ról egységes paraméterként adunk át, a második pedig a kiválasztandó texelre mutató textúracím.

A példában megfigyelhető, hogyan kezelhetjük azt a korlátot, hogy az árnyaló csak olvashat a textúrákból, de nem írhatja őket. A ***SAXPY***

műveletben az **y** vektor egyszerre szerepel bemenetként és kimenetként is. Ennek feloldására két textúrát rendelünk az **y** vektorhoz. Az egyik az eredeti vektor az *arrayY* textúrában, a másik pedig a rajzolási céltextúra. Az eredeti értéket olvassuk, így a kimenetet anélkül is elő tudjuk állítani, hogy a rajzolási céltextúrából kellene olvasnunk – ami nem lenne lehetséges.

16.3.2. Képszűrés

Másik fontos példa két textúrának, nevezetesen a kép- és a szűrőtextúrának a diszkrét konvolúciója, ami sok képfeldolgozási algoritmusban használt alapművelet.

$$\tilde{L}(X,Y) \approx \sum_{i=-M}^{M} \sum_{j=-M}^{M} L(X-i, Y-j) w(i,j) \ , \tag{16.1}$$

ahol $\tilde{L}(X,Y)$ a szűrt érték az X,Y képpontban, $L(X,Y)$ az ere deti kép és $w(x,y)$ a ***szűrőkernel,*** ami $(2M+1) \times (2M+1)$ képpontot fog át.

Ezúttal a pixelárnyaló az *Image* textúrában tárolt kép és a *Weight* textúrában tárolt szűrőkernel alapján egy kimeneti képpont kiértékeléséért felel. Az M értéket, ami a szűrőkernel méretének a fele, egységes paraméterként kapja meg az árnyaló:

```
float3 FragmentShaderConvolution(
        in float2 index : WPOS,     // célpixel koordinátái
        uniform samplerRECT Image,  // bemeneti kép
        uniform samplerRECT Weight, // szűrő kernel
        uniform float M             // a szűrő kernel mérete
        ) : COLOR  // a szűrt kép egyetlen pixele
{
    float3 filtered = float3(0, 0, 0);

    for(int i = -M; i <= M; i++)
        for(int j = -M; j <= M; j++) {
            float2 kernelIndex = float2(i, j);
            float2 sourceIndex = index - kernelIndex;
            filtered += texRECT(Image, sourceIndex) * texRECT(Weight, kernelIndex);
        }
    }
    return filtered;
}
```

Bár a példánk lineáris, vagyis konvolúciós szűrő volt, nemlineáris szűrőket, például mediánszűrést is megvalósíthatunk hasonlóan. A programban a két, illetve három lebegőpontos számot tároló ***float2*** és ***float3*** típusú változókon alkalmaztunk aritmetikai operátorokat (*, +=, =). A Cg fordító, illetve a GPU ezeket a műveleteket a vektorelemekre, azaz a koordinátákra, egyenként és egymástól függetlenül végzi el.

Megjegyezzük továbbá, hogy nem foglalkoztunk azzal a kérdéssel, hogy mi történik a kép szélein, hanem a textúrát mindig a célcím és a kernelcím különbségével megcímezve olvastuk ki. Egy ilyen, a határokat figyelmen kívül hagyó CPU implementáció nyilvánvalóan hibás lenne, hiszen túlindexelnénk

a forrástömböt. Itt azonban a *texRECT* függvényt megvalósító textúraolvasó hardver önállóan megoldja ezt a problémát. A textúra inicializálásakor megadhatjuk, mi történjen, ha a textúrakoordináta kilóg az értelmezési tartományból. A választott opciótól függően vagy a legközelebbi érvényes texelt, vagy egy alapértelmezett értéket kapunk, vagy akár a címet értelmezzük periodikusan.

16.4. A vektorfeldolgozáson túl

A példák szerint a GPU-t vektorprocesszorként egyszerű használni. Ha az algoritmusunk vektorfeldolgozásra alkalmas, akkor kézenfekvő ez a megközelítés. Ugyanakkor számos algoritmusra nem alkalmazható jól a vektorfeldolgozás elve, mégis lehetséges rájuk hatékony GPU implementációt adni. Ebben a szakaszban a vektorfeldolgozó keretrendszert algoritmusok szélesebb körére kiterjesztő koncepciókat tekintjük át.

16.4.1. SIMD vagy MIMD

A vektorprocesszorok általában ***SIMD*** (Single Instruction Multiple Data) gépek, vagyis egyszerre ugyanazt a gépi utasítást hajtják végre több adaton. Ez azt jelenti, hogy a vektorműveletek nem tartalmazhatnak adatfüggő feltételeket vagy dinamikus lépésszámú ciklusokat. SIMD párhuzamos program ágaiban csak egyetlen közös vezérlési szekvencia lehet.

Természetesen feltételes utasítások nélkül és konstans ciklushatárokkal programozni nagyon korlátozó. A legkorábbi GPU árnyalók ilyen SIMD processzorok voltak, és a programozó feladata volt az összes feltétel kiküszöbölése. A mai GPU-k és fordítók már maguktól megoldják ezt, így programozói szinten ugyanúgy használhatók a feltételes elágazások és dinamikus lépésszámú ciklusok, mintha az árnyalók a CPU-khoz hasonló ***MIMD*** (Multiple Instruction Multiple Data) processzorok lennének. A végrehajtás szintjén speciális vezérlési logika teszi lehetővé, hogy az egyes skalár-egységek végrehajtási szekvenciái eltérjenek: az egy multiprocesszorban lévő skalárprocesszorok továbbra is ugyanazt az utasítást hajtják végre az összes itt futó szálon, de nem minden egység dolgozik valójában, azaz nem változtathatja meg az adatait. A különböző vezérlési szekvenciák műveleteit sorosítják, hogy végül mindegyik végrehajtódjon. Az a szál, amely nem a saját szekvenciáját futtatná, letiltott állapotba kerül. Így – a sorosítás miatt nem dolgozó egységeknek felróhatóan – a teljesítmény erősen függ a végrehajtási szekvenciák hasonlóságától. Ez az ára annak, hogy a tranzisztorokat vezérlési logika helyett több feldolgozóegységre áldozza a GPU.

A feltételes szerkezet minden ágának végrehajtását az eredmény írásának tiltásával megoldó trükk neve ***predikáció***. Tegyük fel, hogy programunkban szerepel a következő részlet:

```
if (condition(i)) {
    F( );
} else {
    G( );
}
```

A bemenő adattól függően bizonyos processzorokon a ***condition(i)*** feltétel igaz, másokon hamis lesz, így vektorszámítógépünknek bizonyos processzorokon az F függvényt kellene végrehajtania, más processzorokon a G függvényt. Mivel a SIMD architektúra csak egy vezérlési utat enged meg, a párhuzamos rendszernek mindkét szekvenciát végre kell hajtania, úgy, hogy csak akkor engedélyezi az adatok átírását, amíg az érvényes fázisban van. Ezzel a módszerrel az eredeti program a következő, feltételes elágazásokat nem tartalmazó algoritmussá alakítható:

```
enableWrite = condition(i); // írásengedélyezés a feltétel alapján
F( );
enableWrite = !enableWrite; // írásengedélyezés a negált feltétel alapján
G( );
```

Mivel ebben a verzióban nincs feltételes végrehajtás, lefuttatható a SIMD gépen. A számítási idő azonban a két függvény számítási idejének összege lesz.

Ezt a szűk keresztmetszetet úgy lehet kiküszöbölni, ha a számítást több menetre bontjuk és kihasználjuk a korai z-teszt lehetőségét. A korai z-teszt összeveti a pixel z értékét a mélységbuffer tartalmával, és ha az kisebb a tárolt értéknél, a pixelre nem fut le a pixelárnyaló program, helyette a processzor másik adatelemet dolgoz fel. A korai z-teszt automatikusan bekapcsolódik olyan pixelárnyaló programok esetén, amelyekben nem módosítjuk a pixel z értékét (az összes eddig tárgyalt program ilyen volt).

Ennek a képességnek a kihasználásához három menetre bontjuk a számítást. Az első menetben csak a feltételt értékeljük ki és a mélységbuffert ennek megfelelően inicializáljuk. Idézzük fel, hogy ha a z értéket nem módosítjuk, a teljes nézetablakos téglalapunk normalizált eszközkoordinátarendszerben az x, y síkon, képtérben pedig a $z = 0.5$ síkon van. Tehát ahhoz, hogy a feltételnek megfelelő különbségtételt érjünk el, $(0.5, 1)$ tartományban levő értékeket állíthatunk be a feltétel teljesülése esetén, és $(0, 0.5)$ tartományban levőt, ha a feltétel hamis.

Az első menet pixelárnyalója csak a feltételt értékeli ki és írja a mélységbufferbe:

```
float FragmentShaderCondition(
        in float2 index : WPOS,     // célpixel koordinátái
        uniform samplerRECT Input,  // bemeneti vektor
```

```
        ) : DEPTH  // a kimenet a mélységbufferhez kötött
{
    bool condition = ComputeCondition(texRECT(Input, index));
    return (condition) ? 0.8 : 0.2; // 0.8 nagyobb, mint 0.5; 0.2 pedig kisebb
}
```

Ezután még két menetet kell futtatnunk az F és a G kiszámítására. Az első menetben a pixelárnyaló F-et számítja ki és a mélységellenőrzést azon pixelek átengedésére állítjuk be, amelyek $z = 0.5$ mélysége kisebb, mint a mélységbufferben levő érték. Ebben a menetben így csak azok a pixelek értékelődnek ki, amelyekhez a mélységbufferbe 0.8-at írtunk, vagyis ahol a feltétel igaz volt. Ezután a második menetben a pixelárnyaló G-t számítja ki, és a mélységtesztet a bufferben levőnél kisebb mélységű pixelek átengedésére állítjuk be.

A 16.7.1. szakaszban a korai z-teszt segítségével változó lépésszámú ciklust valósítunk meg a pixelárnyalóban.

16.4.2. Redukció

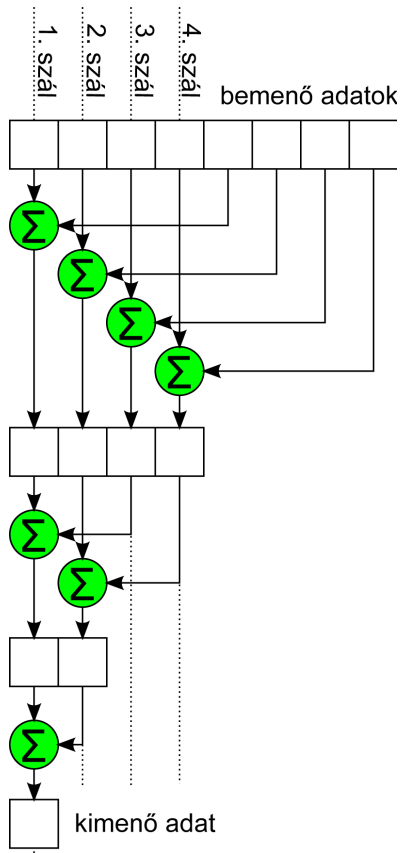
A vektorfeldolgozási megközelítésben a kimeneti tömb elemeit függetlenül számoljuk. A tömbnek elég nagynak kell lenni ahhoz, hogy minden árnyalóprocesszort ellásson munkával. Világos, hogy ha csak egy vagy néhány eleme van a tömbnek, akkor csak egy vagy néhány processzor dolgozhat egyszerre, vagyis elveszítjük a párhuzamos feldolgozás előnyét.

Számos algoritmus eredménye nem egy nagy tömb, hanem csak egyetlen, tömb alapján számolt érték. Ilyenkor az algoritmusnak csökkentenie, redukálnia kell a kimenet méretét. Ha ezt a ***redukciót*** egyetlen lépésben valósítjuk meg, egyetlen texelt kiértékelve, akkor nem használjuk ki optimálisan a párhuzamos architektúrát. Ezért a redukciót is érdemes párhuzamosan, több lépésben végezni. Ez akkor lehetséges, ha a tömbből az eredmény kiszámítására használt művelet asszociatív, ami a leggyakrabban előforduló műveletekre, az összeg, a minimum, a maximum vagy az átlag képzésére igaz.

Tegyük fel, hogy a tömböt egy kétdimenziós textúrában tároljuk. Egy számítási lépés abból áll, hogy textúra felbontását felére csökkentjük, vagyis négy szomszédos texelt egyetlen texelre cserélünk. A pixelárnyalók tehát mindig négy bementi texelt olvasnak be. Ha az eredeti tömb $2^n \times 2^n$ felbontású, akkor n redukciós lépés szükséges a kimeneti 1×1 méretű eredmény előállításához. A következő példában a bemeneti tömb elemeinek összegét számoljuk ki (16.6. ábra). A CPU program minden iterációban felezi a felbontást és egy teljes nézetablakos téglalapot rajzol.

```
Reduction( ) {
    Az arrayX bemeneti textúra azonosító beállítása

    for(N /= 2 ; N >= 1; N /= 2) { // log_2 N iteráció
        glViewport(0, 0, N, N);    // A kép méret beállítása, amely NxN pixeles
        glBegin(GL_QUADS);         // Egy teljes nézetablakos téglalap rajzolása
```



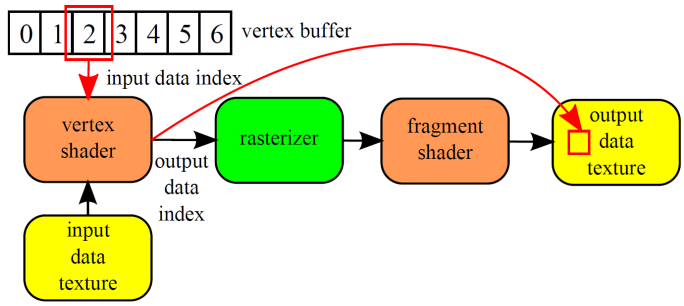
16.6. ábra. Példa a párhuzamos redukcióra: bemenő tömb elemeinek összegzése.

```
            glVertex4f(-1,-1, 0, 1);
            glVertex4f(-1, 1, 0, 1);
            glVertex4f( 1, 1, 0, 1);
            glVertex4f( 1,-1, 0, 1);
        glEnd( );

        A célterület átmásolása az arrayX azonosítójú textúrába
    }
}
```

A pixelárnyaló egyetlen redukált texelt számol ki négy texel összegzésével:

```
float FragmentShaderSum( ) (
        in float2 index : WPOS,       // célpixel koordinátái
        uniform samplerRECT arrayX,   // az x bemeneti tömb textúraazonosítója
        ) : COLOR                     // egy szín, amely a kimeneti tömb egy eleme
{
    float sum = texRECT(arrayX, 2 * index);
    sum += texRECT(arrayX, 2 * index + float2(1, 0));
    sum += texRECT(arrayX, 2 * index + float2(1, 1));
    sum += texRECT(arrayX, 2 * index + float2(0, 1));
    return sum;
}
```



16.7. ábra. Szórás megvalósítása.

Megjegyezzük, hogy ha kihasználnánk a textúramemória bilineáris szűrésének lehetőségét, három textúraolvasást megspórolva egyetlen művelettel is kiolvashatnánk a négy texel értékének átlagát.

16.4.3. Szórás

Vektorfeldolgozás esetén minden processzor egy kimeneti értékhez rendelt, vagyis minden processzornak tudnia kell, mely kimeneti elemet számolja, amit nem irányíthat máshova. Az ilyen statikus jellegű összerendelés ***gyűjtő*** jellegű számításokhoz megfelel. A gyűjtés sémában, a bemeneti halmaz választható dinamikusan, de előre ismernünk kell, hogy hova akarjuk kiírni az eredményt:

```
index = ComputeIndex( );      // A bemeneti tömb indexe
y = F(x[index]);
```

A gyűjtéssel szemben vannak ***szórás*** jellegű algoritmusok, ahol egy adott bemeneti érték valamely dinamikusan választott kimenethez járulhat hozzá. Egy egyszerű szórási jellegű művelet:

```
index = ComputeIndex( );      // A kimeneti tömb indexe
y[index] = F(x);
```

A vektorfeldolgozás általában nem alkalmas szórás (lásd 16.7. megvalósítására, hiszen a pixelárnyaló csak a hozzá rendelt képpontba írhat.

Ha szórási típusú algoritmust szeretnénk megvalósítani a GPU-n, két lehetőségünk van. Egyrészt átstrukturálhatjuk az algoritmust gyűjtő típusúvá. Ahhoz, hogy szórási típusú algoritmusokat gyűjtési típusúvá alakítsunk, máshogy kell tekintenünk a feladatra és megoldására. Integrálegyenletek és transzportproblémák esetén ez az adjungált probléma megoldását jelenti [287]. Másrészt, az index számítását előbbre hozhatjuk a csővezetékben, és dinamikusan rendelhetjük az indexet a rajzolási célhoz a raszterizáló segítségével (16.7. ábra).

Tekintsünk egy híres szórási típusú algoritmust, a ***hisztogram*** generálást. Tegyük fel, hogy egy M méretű $\mathbf{x}$ bemeneti tömb elemeire értékeljük ki

az F függvényt, és kiszámoljuk azt az N elemű $\mathbf{y}$ kimeneti tömböt, ami az $(Fmin, Fmax)$ tartomány N egyenlő részre osztásával kapott intervallumokba eső függvényértékeket tárolja.

A hisztogram generálás nem párhuzamos megvalósítása a következő lenne:

```
Histogram( x ) {
    for(int i = 0; i < M; i++) {
        index = (int)((F(x[i]) - Fmin)/(Fmax - Fmin) * N); // intervallum index
        index = max(index, 0);
        index = min(index, N-1);
        y[index] = y[index] + 1;
    }
}
```

Láthatjuk, hogy a fenti függvény nem előre meghatározott helyeken ír a kimeneti tömbbe, ezért ez a függvény a csak kötött indexű célhelyre írni képes pixelárnyalóval nem megvalósítható. A szórást úgy implementálhatjuk, hogy az indexet a csúcspontárnyalóban számoljuk ki, de a számláló növelésének feladatát a csővezeték többi részére hagyjuk. Az indexeket a raszterizáló hardver köti a kimeneti képpontokhoz. Az olvasás-módosítás-írás ciklusok problémáját megoldhatnánk úgy, hogy minden inkrementálási művelet után új menetet kezdünk és a jelenlegi célterületet a következő menet bemeneti textúrájába másoljuk. Ez a megoldás azonban nagyon gyenge teljesítményt nyújtana és egyáltalán nem használná ki a párhuzamos hardvert. Sokkal szerencsésebb, ha a kompozitáló egység aritmetikai képességeit aknázzuk ki. A pixelárnyaló csak az inkremenst (vagyis az 1 értéket) állítja elő, mégpedig ott, ahol a hisztogramot növelni kell, és ezt az értéket adja tovább a kompozitáló egységnek. A kompozitáló hozzáadja az inkremenst a rajzolási cél tartalmához.

A CPU program minden egyes bemenő adatelemre egy pontprimitívet generál. Ezen felül beállítja a kimenő tömböt rajzolási célnak és engedélyezi a kompozitáló egység összeadási műveletét:

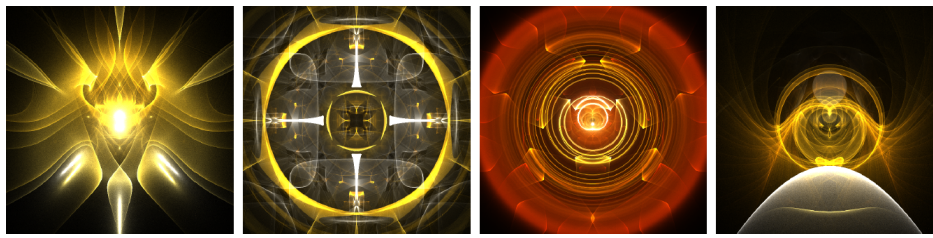
```
CPUHistogram( ) {
    Egységes paraméterek beállítása: Fmin, Fmax, N

    glDisable(GL_DEPTH_TEST);    // A mélységellenőrzés kikapcsolása
    glBlendFunc(GL_ONE, GL_ONE); // Keverő operátor: cél = forrás * 1 + cél * 1;
    glEnable(GL_BLEND);          // Keverés engedélyezése

    glViewport(0, 0, N, 1);      // A célterület méretének beállítása: N pixel
    glBegin(GL_POINTS);          // Minden egyes bemeneti elem egy pont primitív
    for(int i = 0; i < M; i++) {
        glVertex1f( x[i] );      // Egy pontprimitív feldolgozása
    }
    glEnd( );
}
```

A csúcspontok pozíciói itt még nem lényegesek, hiszen csak később derül ki, hova képezzük le a pontokat. Így a csúcspontok első koordinátáját az $x[i]$ bemenő elem átadására használjuk.

A csúcspontárnyaló megkapja a csúcspont pozícióját, ami ekkor a bemenő elemet tartalmazza és megállapítja a pont pozícióját normalizált eszköztér-



16.8. ábra. A hisztogram generálást felhasználó kausztika szimuláció. Az illumináció erőssége a beérkező fotonok számával arányos (Balambér Dávid képei).

ben. Ehhez először az F függvényt értékeli ki és megállapítja a függvényértéket tartalmazó intervallum indexét. Ezután ezt az indexet a $[-1, 1]$ tartományba konvertálja, hiszen normalizált eszköztérben ezek a nézetablak szélei:

```
void VertexShaderHistogram(
    in float   inputPos   : POSITION,
    out float4 outputPos  : POSITION,
    uniform float Fmin,
    uniform float Fmax,
    uniform float N )
{
    float xi = inputPos;
    int index = (int)((F(xi) - Fmin)/(Fmax - Fmin) * N); // részintervallum index
    index = max(index, 0);
    index = min(index, N-1);
    float nindex = 2.0 * index / N - 1.0;   // normalizált eszköz koordinátarendszer
    outputPos = float4(nindex, 0, 0, 1);    // kimeneti koordináták beállítása
}
```

A fenti példa nem optimalizált. Vegyük észre, hogy az index számítást és a normalizálást összevonhatnánk és így a kimenő tömb N méretére sem lenne szükségünk. A pixelárnyaló minden olyan képpontra le fog futni, amire pontprimitív vetül és egyszerűen az 1 inkremens értéket adja ki:

```
float FragmentShaderIncr( ) : COLOR   // A kimenet egy pixel szín
{
    return 1;  // inkremens, amit a keverő ad a célterület pixeléhez
}
```

16.4.4. Párhuzamosság vagy újrafelhasználás

Az egymástól függetlenül futó párhuzamos processzorok az egyszálas megvalósításhoz képest a processzorok számával arányos gyorsulást nyújtanak. Az egyszálas, nem párhuzamos megvalósítás azonban kihasználhatja, ha különböző kimeneti értékek számításában közös rész ismerhető fel, amit értelemszerűen csak egyszer kell kiszámítani. Így a részeredmény újrafelhasználásával a teljesítmény növelhető. Mivel a független párhuzamos processzorok nem hasznosíthatják újra a más processzorok által előállított adatokat, komparatív előnyeik csökkennek. A GPU-k jelentős adatfolyam-kezelési képességgel

rendelkező párhuzamos rendszerek, így ha az újrahasznosítható adat elég korán áll elő, a független párhuzamos feldolgozás és az újrahasznosítás előnyeit egyaránt kiaknázhatják.

A fő adatsokszorozónk a raszterizáló egység. Így bármi, ami a raszterizálás előtt történik, a raszterizált primitív által lefedett képpontok szempontjából globális számításnak tekinthető. Másik megoldás, hogy mivel egy menet eredményét a következő menetekben bemeneti textúraként használhatjuk, az újrahasznosítandó adatokat textúrákba kell írni és a folyamatot több képalkotási fázisra kell bontani.

16.5. A GPGPU programozási modell: CUDA és OpenCL

A ***CUDA*** (Compute Unified Device Architecture) és az interfészek kínálta programozási modell jelentősen eltér a grafikus csővezeték modelljétől (16.1. ábra, jobb oldal). Ezek multiprocesszorok gyűjteményeként láttatják a GPU-t, ahol minden multiprocesszor több SIMD skalárprocesszort tartalmaz. A skalárprocesszorok saját regiszterekkel rendelkeznek és a multiprocesszoron belül ***osztott memórián*** keresztül kommunikálhatnak, gyorsítótárazott textúrákból beépített szűréssel olvashatnak, valamint írhatják és olvashatják a lassú globális memóriát. Ha kívánjuk, akár olvasás-módosítás-írás műveleteket is használhatunk. A globális memóriában textúrákat hozhatunk létre, amelyek a létrehozásuk után csak olvasásra érhetők el.

A grafikus API modelljével szemben itt a globális memória írása nem kizárólagos, így versenyhelyzetet teremthet, viszont ***atomi műveletekkel*** működtethető szemaforokkal biztosíthatjuk az adatok foglaltságának jelzését. A vágás, raszterizáció, kompozitálás és más rögzített funkciójú elemek nem láthatóak és nem is használhatóak ebben a programozási modellben.

A GPGPU programozási modellt a grafikus API modelljéhez hasonlítva megállapíthatjuk, hogy tisztább és egyszerűbb. A GPGPU modellben a párhuzamos processzorok egy szinten vannak és korlátlanul elérhetik a globális memóriát, míg a grafikus modellben a processzorok és fix funkciójú elemek csővezetéket alkotnak, és memória írás csak a csővezeték végén lehetséges. Amikor a GPGPU modellben programozunk, kevesebb korlátozással szembesülünk. Ugyanakkor óvatosnak kell lennünk, hiszen a grafikus csővezeték modellje pontosan azokat a lehetőségeket tiltja, amelyek használata amúgy sem ajánlott nagy teljesítményű alkalmazásokban.

A GPGPU programozás művészete abban rejlik, hogy az eredeti algoritmust hatékonyan bontsuk párhuzamos szálakra úgy, hogy minimális mennyiségű adatkommunikációra és szinkronizációra legyen szükség, de a

lehető legtöbb processzort lássuk el munkával. A következő szakaszokban egy alapvető műveletet, a mátrix-vektor szorzást elemezzük ebből a szempontból.

16.6. Mátrix-vektor szorzás

A számítási feladatok matematikai modelleken és ezek numerikus megoldásán alapulnak. A numerikus módszerek általában valamiféle linearizálásra épülnek, ami lineáris egyenletrendszerek megoldását igénylő algoritmusokhoz vezet, ahol az iteratív megoldás lépéseiben mátrix-vektor szorzatokat kell számolnunk. Így a mátrix-vektor szorzás olyan alapművelet, amit ha párhuzamos architektúrán hatékonyan tudunk implementálni, akkor egy nagyon jól használható általános építőelemet kapunk. Definiálhatjuk úgy is a feladatot, hogy az $\mathbf{y}$ eredményvektort az $\mathbf{A}$ mátrixból, valamint az $\mathbf{x}$ és $\mathbf{b}$ vektorokból álló bemenet alapján kell számolnunk a következőképpen:

$$\mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{b} \, .$$

Nevezzük ezt MV feladatnak! Legyen az $\mathbf{A}$ mátrix mérete $N \times M$! Mivel minden bemenő vektorelem szerepel minden kimenő vektorelem számításában, a nem párhuzamos CPU-implementáció két egymásba ágyazott ciklust tartalmazna, ahol az egyik ciklus a bemenet, a másik a kimenet elemein iterálna végig. Ha úgy párhuzamosítjuk az algoritmust, hogy a kimenet elemeit párhuzamos szálakhoz rendeljük, akkor egy gyűjtő típusú algoritmust kapunk, ahol egy szál az összes bemenő elem hozzájárulását összegyűjti és a szál egyetlen kimeneti értékében akkumulálja. Ezzel szemben ha a párhuzamos szálakat a bemenet elemeihez rendelnénk, akkor a szál ennek a bemeneti elemnek minden kimenő elemhez tartozó hozzájárulását számolná ki, ami egy szórási művelet lenne. Gyűjtés esetén a szálak csak a bemenő adatokon osztoznak, de kimenetük kizárólagos, így nincs szükség szinkronizációra. Szórás esetén több szál adhatja hozzá eredményét ugyanahhoz a kimeneti elemhez, így atomi összeadásokra van szükség, ami teljesítményromlással jár.

Nem párhuzamos CPU-n a mátrix-vektor szorzás megvalósítása például a következő lehet:

```
void ScalarMV(int N, int M, float* y, const float* A, const float* x, const float* b)
{
    for(int i=0; i<N; i++) {
        float yi = b[i];
        for(int j=0; j<M; j++) yi += A[i * M + j] * x[j];
        y[i] = yi;
    }
}
```

Az algoritmus párhuzamos gépre alkalmazásának első lépése az egy szálra eső feladat meghatározása. A szórás és gyűjtés lehetőségei közül a gyűjtést kell

előnyben részesítenünk, mivel automatikusan kiküszöböli az írási ütközések problémáját. Gyűjtő típusú megoldás esetén egy szál az **y** vektor egy elemét számolja, így N szálat kell indítanunk. A GPU gyakorlatilag korlátlan számú szálat képes indítani. A szálak blokkokba vannak csoportosítva, egy blokk szálai ugyanazon a multiprocesszoron futnak. Így a következő tervezési döntés az, hogy az N szálat hogyan osszuk blokkokba. Egy multiprocesszor tipikusan 32 szálat hajt végre párhuzamosan, így a blokkokban levő szálak számának a 32 többszörösének kell lennie. Amikor a szálak a lassú memóriaelérés miatt várakozni kényszerülnek, a hardveridőzítő más szálakat próbál futtatni. Ezért hasznos több mint 32 szálat rendelni egy multiprocesszorhoz, hogy mindig legyenek futásra kész szálak. Ugyanakkor az egy blokkban lévő szálak számának növelése arra is vezethet, hogy túl kevés blokkunk marad, így a program csupán néhány multiprocesszoron fut. Ezeket figyelembe véve, rendeljünk 256 szálat egy blokkhoz és reméljük, hogy az $N/256$ meghaladja a multiprocesszorok számát és így teljesen kihasználjuk a párhuzamos hardvert.

Kis problémát jelent, ha N nem 256 egész számú többszöröse. A vektor maradék elemeit is processzorhoz kell rendelnünk, így a szálblokkok számának $N/256$ felső egész részének kell legyen. Így viszont lesznek olyan szálaink, amelyekhez nem rendelünk vektorelemeket. Ez nem jelent gondot, ha az extra szálak ezt felismerik és nem okoznak kárt például a kimeneti tömb túlindexelésével.

A korábban tárgyalt vektorfeldolgozási esethez hasonlóan itt is minden szálnak tudnia kell, hogy melyik kimeneti elemet számítja. A CUDA könyvtár ezt az információt implicit bemeneti elemek formájában teszi elérhetővé: a *blockIdx* a szálblokk indexe, a *blockDim* a blokkban levő szálak száma, és a *threadIdx* a szál indexe a blokkon belül.

A kimeneti vektor egy elemét számító CUDA kernel program most nem különálló árnyalóprogramként, hanem a hagyományos CPU program speciális függvényeként jelenik meg:

```
__global__ void cudaSimpleMV(int N, int M, float* y, float* A, float* x, float* b)
{
    // A kiszámítandó elem indexének meghatározása a szál és blokk indexekből
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if(i < N) {  // Ha az index túlcímez a tömbön, akkor ugord át.
        float yi = b[i];
        for(int j=0; j<M; j++) yi += A[i * M + j] * x[j];
        y[i] = yi;
    }
}
```

A ***global*** kulcsszó azt jelzi a fordítónak, hogy ez a függvény nem a CPU-n, hanem a GPU-n fog futni, de a CPU-ról is hívható. A paramétereket hagyományos C szintaxissal adjuk át. Az egyetlen különlegesség az, hogy az implicit paramétereket használjuk a szál azonosítószámának, vagyis a kimeneti tömbelem indexének meghatározásához.

A kerneleket a CPU programból indítjuk, megadva nemcsak a paramétereket, hanem a szálblokkok és a blokkon belüli szálak számát is:

```
__host__ void run_cudaSimpleMV()
{
    int threadsPerBlock = 256; // number of threads per block
    int blockNum = (N +  threadsPerBlock - 1)/threadsPerBlock; // blokkok száma
    cudaSimpleMV<<<blockNum, threadsPerBlock>>>(N, M, y, A, x, b);
}
```

A fordító a ***host*** kulcsszó alapján ismeri fel, hogy a függvény a CPU-n fut. A párhuzamos szálak indítása C függvényhíváshoz hasonlóan történik, kivéve a <<<blockNum, threadsPerBlock>>> elemet, ami azt adja meg, hogy hány szálat kell indítani és hányat kell belőlük egy multiprocesszorra elosztani.

16.6.1. A mátrix-vektor szorzás további párhuzamosítása

Eddig a mátrix sorait rendeltük párhuzamos szálakhoz, az $\mathbf{A}_i\mathbf{x}$ skalárszorzatot a szálak sorosan számították. Amennyiben a mátrix sorainak száma kisebb, mint a párhuzamos skalárprocesszorok száma, akkor ekkora párhuzamosítás nem elégséges ahhoz, hogy minden feldolgozóegységet munkával lássunk el. A skalárszorzat számításának átfogalmazása jól ismert, de fogósabb párhuzamosítási probléma, mivel az összeadásokat nem lehet függetlenül végrehajtani és egyetlen skalárt kell kiírnunk a mátrix minden sorára. A szumma részei viszont számolhatók függetlenül, majd ezután a részösszegek összeadhatók. Ez a redukció klasszikus példája. Arra van szükség, hogy a részösszegeket számoló szálak a műveletet előbb befejezzék, az eredményeiket olyan memóriába írják, ahol az azokat összeadó szál elérheti őket és az összeadás csak ezután történjen meg. Ezért a szálak közötti szinkronizációt és az azonos blokkban lévő szálak számára elérhető osztott memóriát kell használnunk.

Kezdetben tegyük fel, – ideális körülményeket és hardvert feltételezve – hogy egy sor feldolgozására lehet M darab szálunk és az osztott memória képes M darab lebegőpontos érték tárolására! Legyen $\mathbf{Q}$ egy M elemű, osztott memóriában tárolt vektor. Ekkor minden szál egy $Q_j = A_{ij}x_j$ elemet számolhat. Végül $\mathbf{Q}$ elemeit összegzéssel kell redukálnunk. Ehhez tegyük fel azt is, hogy $M = 2^k$, azaz kettő hatvány! Így a redukciót k lépésben hajthatjuk végre, minden lépésben a szálak felét leállítva, míg minden túlélő szál összeadja $\mathbf{Q}$ saját maga által és egy leállított szál által számolt elemét. A végül utoljára maradt szál kiírja az értéket a globális memóriába.

```
#define M THE_NUMBER_OF_MATRIX_COLUMNS
__global__ void cudaReduceMV(int N, float* y, float* A, float* x, float* b)
{
    int i = blockIdx.x;
    int j = threadIdx.x;

    __shared__ float Q[M];  // a multiprocesszor közös memóriájában legyen
    Q[j] = A[i * M + j] * x[j];  // a matrix-vektor szorzás párhuzamos része
```

```
    for(int stride = M / 2; stride > 0; stride >>= 1) { // redukció
        __syncthreads();    // várj amíg a blokk többi szálja is ide ér
        if(j + stride < M) Q[j] += Q[j + stride];
    }
    if(j == 0) y[i] = Q[0] + b[i]; // ha egyetlen elemre sikerült redukálni
}

__host__ void run_cudaReduceMV()
{
    cudaReduceMV<<< N, M >>>(N, y, A, x, b);
}
```

A gyakorlatban előforduló mátrixméretekre ($M > 10^4$) sem az egyetlen multiprocesszoron futtatható szálak száma, sem az osztott memória mérete nem elég az összes elem párhuzamos feldolgozására. Következő példánkban egyetlen, korlátozott méretű szálblokkot használunk egy nagy mátrix feldolgozására. Egyrészt a kimeneti vektort T méretű szegmensekre osztjuk. Az egy szegmensen belüli elemeket párhuzamosan értékeljük ki, majd a szálak a következő szegmenssel folytatják a munkát. Másrészt minden skalárszorzat számításnál az $\mathbf{A}_i$ és $\mathbf{x}$ vektorokat Z hosszú szegmensekre osztjuk. Minden párhuzamosan feldolgozott sorra egy Z hosszú, közös $\mathbf{Q}_t$ vektort tartunk fent. $\mathbf{A}_i$ és $\mathbf{x}$ szegmenseinek elemenkénti szorzatát párhuzamosan számíthatjuk, majd hozzáadhatjuk $\mathbf{Q}_t$-hez. Mivel T darab sort egyenként Z darab szál dolgoz fel, a blokk $T \times Z$ szálból fog állni. Egyetlen szál szemszögéből nézve ez azt jelenti, hogy az $\mathbf{y}$ vektoron T lépéshosszal kell végigmennie és $\mathbf{y}$ minden elemére $\mathbf{A}_i$-n és $\mathbf{x}$-en Z lépéshosszal kell végigiterálnia. Továbbá $\mathbf{y}$ minden elemére $\mathbf{Q}_t$ értékét – a korábbiakhoz hasonlóan – redukcióval össze kell adnia. A teljes nagy mátrixokon működő kernel tehát:

```
__global__ void cudaLargeMV(int N, int M, float* y, float* A, float* x, float* b)
{
    __shared__ float Q[T * Z]; // a multiprocesszor közös memóriájában legyen

    int t = threadIdx.x / Z;
    int z = threadIdx.x % Z;

    for(int i = t; i < N; i += T) {
        Q[t * Z + z] = 0;
        for(int j = z; j < M; j += Z) Q[t * Z + z] += A[i * M + j] * x[j];

        for(int stride = Z / 2; stride > 0; stride >>= 1) {
            __syncthreads();
            if(z + stride < Z) Q[t * Z + z] += Q[t * Z + z + stride];
        }
        if(z == 0) y[i] = Q[t * Z + 0] + b[i];
    }
}

__host__ void run_cudaLargeMV()
{
    cudaReduceMV<<< 1, T*Z >>>(N, M, y, A, x, b);
}
```

Ezt könnyen kiterjeszthetjük több szálblokk használatára, ha a külső hurkot a mátrix sorainak részhalmazára korlátozzuk a *blockIdx* paraméter

alapján.

A fenti algoritmus kézenfekvően használja az osztott memóriát és lehetővé teszi a blokkméret megválasztásával a szálak memória elérésének a hardver szószélességéhez igazítását. Azonban az $\mathbf{x}$ vektor minden elemét minden sor számításához egyszer be kell olvasnunk. Ezen tudunk javítani, ha az $\mathbf{x}$ értékeit az osztott memóriába olvassuk és az egy blokkban lévő szálakat a mátrix több során működtetjük. Ez azonban azt jelenti, hogy kevesebb osztott memóriát tudunk az összegzés párhuzamosítására használni. A kompromisszum vizsgálata túlmutat ezen fejezet keretein, de megemlítjük, hogy a [94] cikk a 64×8-as blokkméretet javasolta. Ilyen stratégia használata esetén további előnyt jelent az $\mathbf{A}$ mátrix textúraként való elérése, mivel az adatolvasás kétdimenziós lokalitást fog mutatni, ami a textúra-gyorsítótár szempontjából ideális.

A számítási alkalmazások széles köre vezethető vissza olyan mátrix-vektor szorzásra, ahol a mátrixok hatalmasak, de ritkák, azaz nagyon nagy részben zérus elemeket tartalmaznak. Ritka mátrixok esetén a fentebb bemutatott mátrix-vektor szorzási algoritmusok nem hatékonyak, mivel a zérus elemekkel való szorzást is explicit módon számolják. A ritka mátrixok reprezentációit és az azokon működő MV algoritmusokat a [22] publikáció tárgyalja.

16.7. Esettanulmány: Számítógépes áramlásdinamika

A fizikában vagy mérnöki területen felmerülő problémákat általában parciális differenciál- vagy integrálegyenletekkel írhatjuk le matematikailag. Mivel a fizikai rendszereknek időbeli és térbeli kiterjedése is van, a deriváltakat vagy integrálokat az idő- és tértartományban egyaránt ki kell értékelni. Térben és időben kiterjedt adatok reprezentációjához olyan függvényeket kell használnunk, amelyeknek a térbeli pozíció és az idő a szabad változója. Általános függvények tárolásához végtelen mennyiségű adatra lenne szükség, így numerikus módszerekben ezeket csupán véges számú értékkel közelítjük. Szemléletesen ezek az értékek gyakran elképzelhetők úgy is, mint diszkrét pontokban és időpillanatokban vett függvényérték minták. A mögöttes elméleti keretet a végeselem módszer adja. Ha egy $f(\vec{r})$ függvényt véges mennyiségű adattal szeretnénk reprezentálni, a következő véges sorral közelíthetjük:

$$f(\vec{r}) \approx \tilde{f}(\vec{r}) = \sum_{i=1}^{N} f_i B_i(\vec{r})$$

ahol $B_1(\vec{r}), \ldots, B_N(\vec{r})$ előre definiált ***bázisfüggvények*** és $f_1, \ldots, f_N$ az $\tilde{f}$-et leíró együtthatók.

Különösen egyszerű végeselemes reprezentáció a szakaszonként lineáris séma, amely a tartományban – gyakran szabályosan elhelyezett – $\vec{r}_1, \ldots, \vec{r}_N$ mintapontokkal dolgozik. Ezekben a pontokban a függvény kiértékelésével kaphatóak az $f_i = f(\vec{r}_i)$ komponensek és az $\vec{r}_i$ pontok közötti lineáris interpolációval kapjuk a közelítő függvényt.

Ha a rendszer dinamikus, akkor f megoldásának időfüggőnek kell lennie, így különböző időpillanatokban a végeselemes reprezentáció eltér. Alapvetően két lehetőségünk van erre. A mintapontokat rögzíthetjük statikusan és csak az f_i együtthatók változhatnak időben. Ezt a modellt ***eulerinek*** nevezik. Másrészt azt is megengedhetjük, hogy a mintapontok a rendszer mozgásával együtt elmozduljanak, így az $\vec{r}_i$ mintapontok is időfüggővé válnak. Ez a ***lagrange-i*** megközelítés, amelyben gyakran nevezik a mintapontokat ***részecskéknek***.

Az euleri és lagrange-i diszkretizálási sémákra szemléletes példát a meteorológiai adatok (pl. szélerősség, hőmérséklet) mérése alapján adhatunk. A földi állomások az adatokat rögzített helyeken mérik. A meteorológiai ballonok viszont hasonló adatokat éppen a levegő áramlását követve változó pozícióból szolgáltatnak.

Ebben a szakaszban egy GPU-alapú tudományos számítás megvalósításáról szóló esettanulmányt tárgyalunk. A kiválasztott feladat a ***számítógépes áramlásdinamika.*** A füst, a felhőképződés, a tűz, robbanások és sok más természeti jelenség mutat áramlásszerű viselkedést. Így érthető, hogy jó és gyors áramlásszámításra van szükség a mérnöki és animációs területeken egyaránt.

Az áramlás matematikai modelljét a Navier–Stokes-egyenlet adja. Először bemutatjuk ezt a parciális differenciálegyenletet, azután a GPU-alapú euleri és lagrange-i megoldások fejlesztésének lehetőségeit tárgyaljuk.

Egy állandó sűrűségű és hőmérsékletű közeget a $\vec{v} = (v_x, v_y, v_z)$ sebesség- és p nyomásmezővel írhatunk le. A sebesség és nyomás térben és időben egyaránt változik:

$$\vec{v} = \vec{v}(\vec{r}, t), \qquad p = p(\vec{r}, t)\ .$$

Koncentráljunk a közeg egy egységnyi térfogatú elemére, az $\vec{r}$ helyen és t időben. A korábbi, $t - \mathrm{d}t$ időpillanatban ez az elem $\vec{r} - \vec{v}\mathrm{d}t$ helyen volt, és — a dinamika alaptörvényének megfelelően — a sebessége az F eredő erő és az egységnyi térfogatú közeg ρ tömegének hányadosával egyenlő gyorsulásnak megfelelően változott.

$$\vec{v}(\vec{r}, t) = \vec{v}(\vec{r} - \vec{v}\mathrm{d}t, t - \mathrm{d}t) + \frac{\vec{F}}{\rho}\mathrm{d}t.$$

Az egységnyi térfogatú elem ρ tömege a ***közeg sűrűsége.*** A sebességet leíró

tagokat bal oldalra áthozva és az egyenletet $\mathrm{d}t$-vel elosztva kifejezhetjük a sebesség teljes deriváltját:

$$\frac{\vec{v}(\vec{r},t) - \vec{v}(\vec{r} - \vec{v}\mathrm{d}t, t - \mathrm{d}t)}{\mathrm{d}t} = \frac{\vec{F}}{\rho}.$$

Az eredő erő különböző tényezőkből adódik össze. Ébredhet erő a nyomáskülönbség miatt:

$$\vec{F}_{\text{pressure}} = -\vec{\nabla}p = -\left(\frac{\partial p}{\partial x}, \frac{\partial p}{\partial y}, \frac{\partial p}{\partial z}\right),$$

ahol $\vec{\nabla}p$ a nyomásmező gradiense. A negatív előjel azt jelzi, hogy a nyomáskülönbség az alacsonyabb nyomású régió felé gyorsítja a közeget. Itt a ***nabla operátort*** használtuk, ami Descartes-koordinátákban a következő formában írható fel:

$$\vec{\nabla} = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}\right).$$

A súrlódás csillapítja a közeg mozgását. Ez a csillapítás a közeg ν ***viszkozitásától*** függ. Az erősen viszkózus folyadékok szirupként tapadnak egybe, míg az alacsony viszkozitásúak szabadon folynak. A teljes csillapítási erőt diffúziós tagként fejezzük ki, mivel a súrlódási erő a sebességmező a $\vec{\nabla}^2$ ***Laplace-operátorral*** kifejezett második deriváltjával arányos:

$$\vec{F}_{\text{viscosity}} = \nu\vec{\nabla}^2\vec{v} = \nu\left(\frac{\partial^2\vec{v}}{\partial x^2} + \frac{\partial^2\vec{v}}{\partial y^2} + \frac{\partial^2\vec{v}}{\partial z^2}\right).$$

Végül egy $\vec{F}_{\text{external}}$ külső erő is gyorsíthatja a közeget. A Föld felszínén – feltéve, hogy a z a függőleges tengely – ez tipikusan a föld tömegvonzásából származó külső gyorsulás $(0, 0, -g)$, ahol $g = 9.8\ [m/s^2]$.

Az erők összeadásával megkapjuk a térfogat elem sebességére vonatkozó ***Navier-Stokes-egyenletet***:

$$\rho\frac{\vec{v}(\vec{r},t) - \vec{v}(\vec{r} - \vec{v}\mathrm{d}t, t - \mathrm{d}t)}{\mathrm{d}t} = -\vec{\nabla}p + \nu\vec{\nabla}^2\vec{v} + \vec{F}_{\text{external}}.$$

Valójában ez az egyenlet nem más, mint a dinamika alaptörvényének áramló közegre alkalmazása, ezért hívják ezt az egyenletet ***lendületmegmaradási egyenletnek*** is.

A zárt fizikai rendszerek nem csupán a lendületet, hanem a tömeget is megőrzik, így ezt is be kell építenünk áramlási modellünkbe. Egyszerűen fogalmazva a tömeg megmaradása annyit jelent, hogy ami egy térfogatba befolyik, annak onnan ki is kell folynia, vagyis a tömegáram ***divergenciája*** zérus. Amennyiben a közeg összenyomhatatlan, akkor a sűrűsége állandó. Így a tömegáram a sebességmezővel arányos. Összenyomhatatlan közegekre tehát

a tömegmegmaradás miatt a sebességmező ***divergenciamentes***:

$$\vec{\nabla} \cdot \vec{v} = \frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} + \frac{\partial v_z}{\partial z} = 0 \; . \tag{16.2}$$

16.7.1. Az áramlásdinamika euleri megoldása

Az euleri megközelítés a sebesség- és nyomásmező változását rögzített szabályos rácspontokban követi. A rács lehetőséget ad arra, hogy a térbeli deriváltakat egyszerűen véges differenciákkal közelítsük. Ha a rácspontok Δx, Δy és Δz távolságokra helyezkednek el a koordinátatengelyek mentén és a p skalármező, valamint a $\vec{v}$ vektormező értékei az (i, j, k) rácspontban $p^{i,j,k}$ és $\vec{v}^{i,j,k}$, akkor a gradiens-, divergencia- és Laplace-operátorokat a következőképpen közelíthetjük:

$$\vec{\nabla} p \approx \left(\frac{p^{i+1,j,k} - p^{i-1,j,k}}{2\Delta x}, \frac{p^{i,j+1,k} - p^{i,j-1,k}}{2\Delta y}, \frac{p^{i,j,k+1} - p^{i,j,k-1}}{2\Delta x} \right) \; , \tag{16.3}$$

$$\vec{\nabla} \cdot \vec{v} \approx \frac{v_x^{i+1,j,k} - v_x^{i-1,j,k}}{2\Delta x} + \frac{v_y^{i,j+1,k} - v_y^{i,j-1,k}}{2\Delta y} + \frac{v_z^{i,j,k+1} - v_z^{i,j,k-1}}{2\Delta z} \; , \tag{16.4}$$

$$\vec{\nabla}^2 p \approx \frac{p^{i+1,j,k} - 2p^{i,j,k} + p^{i-1,j,k}}{(\Delta x)^2} + \frac{p^{i,j+1,k} - 2p^{i,j,k} + p^{i,j-1,k}}{(\Delta y)^2} + \frac{p^{i,j,k+1} - 2p^{i,j,k} + p^{i,j,k-1}}{(\Delta z)^2} \; . \tag{16.5}$$

A Navier-Stokes-egyenlet és a divergenciamentesség követelménye a diszkretizálás után minden rácspontban és minden időpillanatban négy ismeretlen skalárváltozóra fogalmaz meg követelményeket, amelyek a sebesség három komponense és a nyomás: (v_x, v_y, v_z, p). A numerikus megoldásban az aktuális mezőket az idő Δt-vel való léptetésével számoljuk:

$$\vec{v}(\vec{r}, t) = \vec{v}(\vec{r} - \vec{v}\Delta t, t - \Delta t) + \frac{\nu \Delta t}{\rho} \vec{\nabla}^2 \vec{v} + \frac{\Delta t}{\rho} \vec{F}_{\text{external}} - \frac{\Delta t}{\rho} \vec{\nabla} p \; .$$

A sebességmezőt több lépésben módosítjuk, ahol minden lépés ezen egyenlet jobb oldalán levő egyik tagnak felel meg. Tekintsük ezeket egyenként!

Advekció

Az új sebességmezőt úgy inicializáljuk az $\vec{r}$ pontban, hogy az előző mezőt $\vec{r} - \vec{v}\Delta t$-ben kiolvassuk, mivel az $\vec{r}$-be érkező közegelem ott volt az előző mintavételi időpontban [275]. Ez a lépés az advekciót szimulálja, vagyis azt a jelenséget, hogy az áramlás magával viszi a sebességmezőt:

$$\vec{w}_1(\vec{r}) = \vec{v}(\vec{r} - \vec{v}\Delta t, t - \Delta t) \; .$$

Diffúzió

A diffúzió a sebességmezőnek a közeg belső súrlódásából, viszkozitásából fakadó csillapítása. Megtehetnénk, hogy a diffúziós tagot hozzáadjuk az eddig számolt sebességmezőhöz:

$$\vec{w}_2 = \vec{w}_1 + \frac{\nu \Delta t}{\rho} \vec{\nabla}^2 \vec{w}_1 \ .$$

Ez az ***előrelépéses Euler-integrálás*** azonban numerikusan instabil. Az instabilitás oka, hogy az előrelépéses módszerek jelenlegi értékek alapján jósolják a jövőt és minden szimulációs lépés hozzátesz a hibához, ami minden határon túl növekedhet (részletesebben lásd a tudományos számításokról szóló fejezetben).

Az előrelépéses integrátorokkal szemben a visszalépéses módszerrel garantálható a stabilitás. A visszafele néző megközelítés stabil, mivel a jövő jóslása közben egyben a múltat is korrigálja. Így az összes hiba véges értékhez konvergál és korlátok közt marad. A mi esetünkben a visszalépéses módszer azt jelenti, hogy a Laplace-operátort a jövőbeli, még ismeretlen sebességmezőre alkalmazzuk, nem pedig az aktuális sebességmezőre:

$$\vec{w}_2 = \vec{w}_1 + \frac{\nu \Delta t}{\rho} \vec{\nabla}^2 \vec{w}_2 \ . \tag{16.6}$$

A számítás ezen lépcsőjében a $\vec{w}_1$ advektált mező a rácspontokban ismert, viszont a diffúzió utáni $\vec{w}_2^{i,j,k}$ sebességek ismeretlenek. A (16.5) egyenletet felhasználva megállapíthatjuk, hogy az ismeretlen $\vec{w}_2$ vektormezőre a Laplace-operátort alkalmazva az (i, j, k) rácspontbeli értékre a saját és a szomszédok $\vec{w}_2$ értékeinek lineáris kombinációját kapjuk. Így a (16.6). egyenlet egy ritka lineáris egyenletrendszer:

$$\mathbf{w}_2 = \mathbf{w}_1 + \mathbf{A} \cdot \mathbf{w}_2 \tag{16.7}$$

ahol a $\mathbf{w}_1$ vektor az advekcióval nyert ismert sebességek vektora, $\mathbf{w}_2$ az ismeretlen sebességek vektora, végül az $\mathbf{A} \cdot \mathbf{w}_2$ mátrix-vektor szorzat pedig a $(\nu \Delta t/\rho)\nabla^2 \vec{w}_2(\vec{r})$ deriváltnak a diszkrét formája.

Az ilyen rendszerek kiváló alanyai a ***Jakobi-iterációnak*** (lásd a tudományos számításokról szóló fejezetben). Kezdetben a $\mathbf{w}_2$ vektort zérus értékkel töltjük fel és iteratívan kiértékeljük a (16.7) egyenlet jobb oldalát, az eggyel korábbi lépés eredményét mindig a jobb oldalon szereplő $\mathbf{w}_2$ vektorba másolva. Így az egyenletrendszer megoldását ritka mátrix-vektor szorzások sorozatára vezetjük. Figyeljük meg, hogy az $\mathbf{A}$ mátrixot nem kell tárolnunk. Amikor a $\mathbf{w}_2$ sebességmező egy rácspontbeli értékére van szükség, a szomszédok megkeresésével a (16.5) egyenlet egyszerű képlete megadja az eredményt.

Egy rácspont értékének a saját és szomszédos korábbi értékek alapján történő módosítását ***képszűrésnek*** hívjuk. Így tehát a Jakobi-iteráció egy lépése képszűrési művelettel egyenértékű. A képszűrés GPU megvalósítását a 16.3.2. szakaszban tárgyaltuk.

A külső erőtér

A külső erő minden rácspontban gyorsítja a sebességmezőt:

$$\vec{w}_3 = \vec{w}_2 + \frac{\Delta t}{\rho}\vec{F}_{\text{external}} \, .$$

Vetítés

Az eddigiekben a $\vec{w}_3$ új sebességmezőt az ismeretlen nyomásmező figyelembe vétele nélkül számítottuk. A vetítési lépésben kiszámoljuk az ismeretlen p nyomásmezőt és neki megfelelő módon módosítjuk a sebességmezőt:

$$\vec{v}(t) = \vec{w}_3 - \frac{\Delta t}{\rho}\vec{\nabla} p \, .$$

A nyomásmezőt a sebességmező divergenciamentességének követelményéből nyerjük. Alkalmazzuk az egyenlet mindkét oldalára a divergencia-operátort. Ezután a bal oldal zérus lesz, mivel divergenciamentes vektormezőt keresünk, amire $\vec{\nabla} \cdot \vec{v} = 0$:

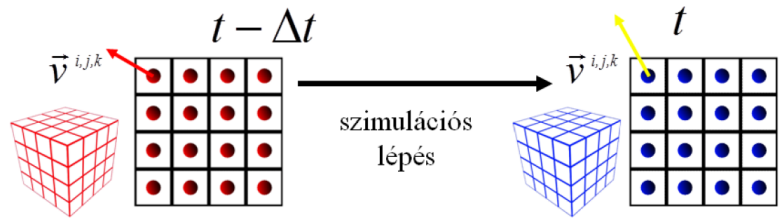
$$0 = \vec{\nabla} \cdot \left(\vec{w}_3 - \frac{\Delta t}{\rho}\vec{\nabla} p \right) = \vec{\nabla} \cdot \vec{w}_3 - \frac{\Delta t}{\rho}\vec{\nabla}^2 p \, .$$

Feltételezve, hogy a $\vec{w}_3$ vektormező értékei szabályos rácson adottak, a rácspontokban az ismeretlen nyomás meghatározásával és a divergencia és a Laplace-operátor a (16.4) és (16.5) egyenleteknek megfelelő véges differenciás kiértékelésével ismét ritka lineáris egyenletrendszert kapunk a diszkrét nyomásértékekre. Ezt az egyenletrendszert is Jakobi-iterációval oldjuk meg. A diffúziós lépéshez hasonlóan a vetítésbeli Jakobi-iteráció is egyszerű képfeldolgozási művelet.

Euleri szimuláció a GPU-n

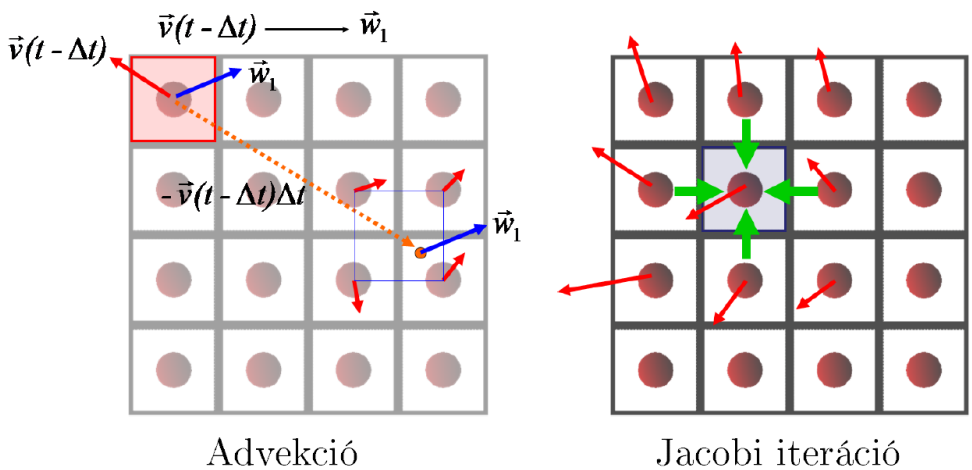
A diszkretizált sebesség- és nyomásmezőket kényelmesen tárolhatjuk háromdimenziós textúrákban, ahol a diszkrét változókat szabályos rácson elhelyezkedő térfogatelemek (***voxelek***) középpontjaihoz rendeljük [115]. Minden lépésben ezen adathalmazok tartalmát számoljuk újra (16.9. ábra).

Az advekció számításánál nagy előnyt jelent, hogy a mezőket textúrában tároljuk. Az $\vec{r}_i$ voxelközéppontban úgy kapjuk az advektált mezőt, hogy az $\vec{r}_i - \vec{v}_i \Delta t$ pozícióból kiolvassuk a mező értékét. Az így számolt pozíció nem feltétlenül esik voxel középpontra, hanem rácspontok között helyezkedik el. A



16.9. ábra. Az euleri módszer egy időlépése a sebességmező textúráját frissíti.

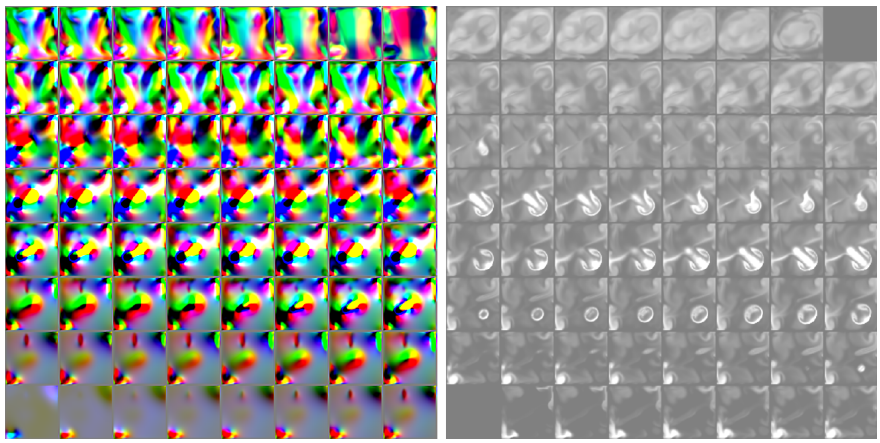
végeselemes megközelítésnek megfelelően az itteni érték a mező végeselemes rekonstrukciójából nyerhető. Ha szakaszonként lineáris bázisfüggvényeket feltételezünk, akkor a textúraszűrő hardver ezt a feladatot további számítási költség nélkül megoldja (16.10. ábra).



16.10. ábra. A szimulációs lépések mint háromdimenziós textúrák módosításai. Az advekció számítása kiaknázza a textúraszűrő hardvert. A súrlódás és vetítés lineáris egyenleteit Jacobi-iterációval oldjuk meg, ahol egy texelt (illetve három dimenzióban voxelt) a szomszédai súlyozott átlagával módosítunk, ami szemléletesen egy képszűrési lépésnek felel meg.

A vektor- és skalármezők háromdimenziós textúrában tárolásának hátránya, hogy a textúrákat a GPU csak olvasni tudja, akár a grafikus API-t, akár a GPGPU megközelítést használjuk. Grafikus API esetén a módosított mezőt a rajzolási célba, GPGPU esetben pedig a globális memóriába kell írni. Ezután a következő szimulációs lépésben a korábbi rajzolási célt vagy globális memóriaterületet bemenő textúrának kell deklarálni.

Az írási ütközések elkerülésére gyűjtő megközelítést alkalmazunk és min-



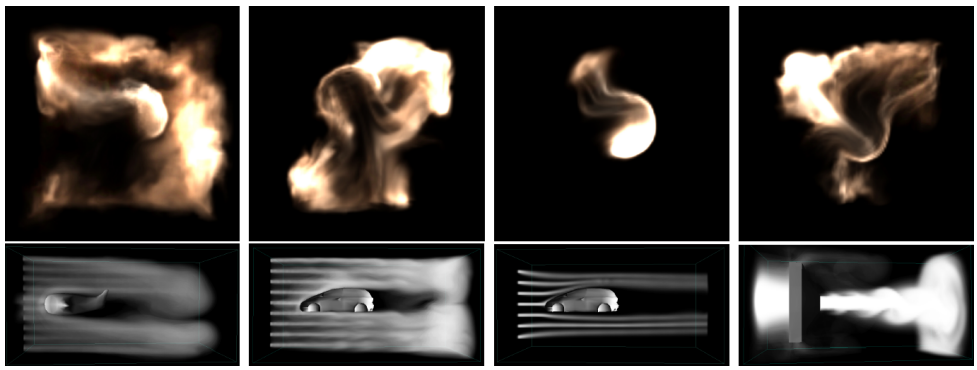
16.11. ábra. Kiterített háromdimenziós sebességmező (bal oldali kép) és a megjelenített sűrűségmező (jobb oldali kép).

den kimeneti értéket tartalmazó rácsponthoz rendelünk egy számot. Ha a GPU-k textúrákon keresztül olvassák a globális adatokat, akkor a szál által kiírt új érték azután válik láthatóvá, hogy a menet vagy a kernelek futása véget ért és az eddigi kimenetet innentől bemenő textúrának nyilvánítjuk. Ezért egyetlen időlépcső számítását elemi módosító lépésekre kell bontanunk mindenhol, ahol egy rácspont korábban számolt értékét fel kívánjuk használni. Ez azt jelenti, hogy szükség van egy advekciós menetre, Jakobi-iterációs menetek sorozatára a diffúziós lépésben, egy külső erőt számító menetre és egy második Jakobi-iterációs menetsorozatra a vetítési lépésben.

A GPGPU keretrendszerben egy szál ugyan olvashatja a más szálak által előállított adatokat, de ekkor szinkronizálni kell, hogy biztosak lehessünk abban, hogy az olvasott érték már érvényes, nem pedig az írás előtti értéket olvassuk. Ilyenkor a szinkronizációs pontok töltik be ugyanazt az ellenőrző szerepet, amit esetünkben a menetek, illetve kernelek szétválasztása.

A grafikus API esetében van még egy korlátozás. A rajzolási cél csak kétdimenziós lehet, így vagy kiterítjük a háromdimenziós voxeltömb rétegeit egy hatalmas kétdimenziós textúrába, vagy egy lépésben csak egy réteget frissítünk. A 16.11. ábra kiterített háromdimenziós textúrát ábrázol. Miután a textúrákat beállítottuk, az egész térfogatra végrehajtható egy szimulációs lépés a teljes kiterített rácsot lefedő téglalap rajzolásával.

A grafikus API-t használó megközelítésnek nemcsak hátrányai, de előnye is van a GPGPU módszerhez képest, mégpedig a lineáris egyenletrendszerek Jakobi-iterációval történő megoldásakor. Minden rácsponton egy pixelárnyaló fut, amely a rácsponthoz rendelt texel értékét számítja. Az olyan rácspon-



16.12. ábra. Euleri folyadékszimulációval készült animáció képkockái.

tokban, ahol a szomszédok hatása elhanyagolható, kevesebb iterációs lépésre lenne szükség, mint ahol a szomszédok jelentősek. Egy kvázi-SIMD gépen, amilyen a GPU, célszerűtlen a processzorokra eltérő mennyiségű számítási munkát bízni. A korai z-teszt kihasználásával azonban ezt a problémát megkerülhetjük és javíthatjuk a teljesítményt [298]. A mélységértéket a környezet legnagyobb elemével és az iterációs számmal arányosan állítjuk be. Ily módon az iteráció során a GPU kevesebb pixelt dolgoz fel, miközben a fontos területekre koncentrálhat. Méréseink szerint ez az optimalizáció a teljes szimulációs időt körülbelül 40%-kal csökkenti).

Ha szeretnénk az áramlást megjeleníteni, feltételezhetjük, hogy az áramlás egy skalár indikátormezőt is magával sodor. Ez úgy működik, mintha némi festéket vagy konfettit öntenénk az áramló közegbe. Az indikátormezőt lebegőpontos voxeltömbben tároljuk.

Az advekciós képletet a D indikátorváltozóra alkalmazva ezt a mezőt is frissíthetjük a Δt időlépcső szimulációja során:

$$D(\vec{r}, t) = D(\vec{r} - \vec{v}\Delta t, t - \Delta t) \ .$$

Egy pont színét és átlátszóságát az indikátorváltozó értékéből egy, a felhasználó által szabályozható ***átviteli függvény*** segítségével számolhatjuk.

A kapott indikátormezőt a háromdimenziós textúrát szeletenként rajzoló módszerrel jeleníthetjük meg. Ez félig átlátszó, a nézeti irányra merőleges sokszögeket rajzol hátulról előre, a keverés bekapcsolásával (16.12. ábra). A háromdimenziós textúra színe és átlátszósága az indikátorváltozó függvénye.

16.7.2. A differenciálegyenletek lagrange-i megoldása

A lagrange-i megközelítésben a teret részecske segítségével diszkretizáljuk, vagyis csak véges számú áramló folyadékelemet követünk. Jelöljük a ré-

szecskék számát N-nel, az i-edik diszkrét közegelem pozícióját és sebességét $\vec{r}_i$-vel és $\vec{v}_i$-vel! Feltesszük, hogy minden részecske ugyanolyan m tömegű közegelemet reprezentál. Mivel a sűrűség térben változik, minden részecskéhez eltérő $\Delta V_i = m/\rho_i$ térfogat tartozik. A lendületmegmaradás egyenlete ebben az esetben a következő formájú:

$$\frac{\mathrm{d}\vec{r}_i}{\mathrm{d}t} = \vec{v}_i \ ,$$

$$m\frac{\mathrm{d}\vec{v}_i}{\mathrm{d}t} = \left(-\vec{\nabla}p(\vec{r}_i) + \nu\vec{\nabla}^2\vec{v}(\vec{r}_i) + \vec{F}_{\mathrm{external}}(\vec{r}_i)\right)\Delta V_i \ . \tag{16.8}$$

Ha a szimuláció során részecskék nem vesznek el, a tömeg automatikusan megmarad, de mivel ez a tömeg kis térfogaton összpontosulhat, a szimulált közeg nem lesz összenyomhatatlan. Lagrange-i szimulációban általában összenyomható gázt feltételezünk.

A diszkrét pontokban ismert rendszer tetszőleges pontban érvényes jellemzőit interpolációval nyerhetjük. Tegyük fel, hogy egy A jellemző a részecskepozíciókban ismert, vagyis $A_1, \ldots, A_N$ adott. Egy tetszőleges $\vec{r}$ pozícióban az A jellemzőt a részecskék hozzájárulásainak súlyozott összegeként számoljuk:

$$A(\vec{r}) = \sum_{i=1}^{N} A_i \Delta V_i W(|\vec{r} - \vec{r}_i|) \ .$$

ahol ΔV_i az $\vec{r}_i$ pontban levő részecske által elfoglalt térfogat, $W(d)$ egy ***simító kernel***, más néven ***radiális bázisfüggvény***, amely a részecske pozíciója és a vizsgált pont közötti d távolságtól függ. Más szemszögből nézve a simító kernel azt fejezi ki, hogy milyen gyorsan enyészik el a részecske hatása a távolsággal. A simító kernel normalizált, ha a simítás a jellemző összértékét megőrzi, ami akkor igaz, ha a kernel a teljes tértartományon integrálva 1-et ad. A lehetséges kernelek egy példája a h legnagyobb sugarú *kúpos kernel*:

$$W(d) = \frac{15}{\pi h^6}(h-d)^3, \text{ ha } 0 \le d \le h \text{ és zérus egyébként}$$

Normalizált kernelekre az $\vec{r}_j$ pontban érvényes részecskesűrűség következőképpen becsülhető:

$$\rho_j = \rho(\vec{r}_j) = \sum_{i=1}^{N} m W(|\vec{r}_j - \vec{r}_i|) \ .$$

Mivel minden részecske ugyanolyan m tömegű, a j indexű részecskéhez tartozó térfogat:

$$\Delta V_j = \frac{m}{\rho_j} = \frac{1}{\sum_{i=1}^{N} W(|\vec{r}_j - \vec{r}_i|)} \ .$$

Az ***ideális gáztörvény*** szerint állandó hőmérsékleten a nyomás fordítottan arányos a térfogattal, így a j indexű részecskénél a nyomás:

$$p_j = \frac{k}{\Delta V_j} ,$$

ahol a k konstans a hőmérséklettől függ. A nyomás egy tetszőleges $\vec{r}$ pontban:

$$p(\vec{r}) = \sum_{i=1}^{N} p_i \Delta V_i W(|\vec{r} - \vec{r}_i|) .$$

A nyomáskülönbségek miatti gyorsuláshoz szükséges a nyomásmező gradiensének számítása. Mivel az $\vec{r}$ térbeli változó csak a simító kernelben szerepel, a gradiens a simítókernel gradienséből számítható:

$$\vec{\nabla} p(\vec{r}) = \sum_{i=1}^{N} p_i \Delta V_i \vec{\nabla} W(|\vec{r} - \vec{r}_i|) .$$

Így – első közelítésben – a j indexű részecskére a nyomásból származó erő:

$$\vec{F}_{\text{pressure},j} = -\vec{\nabla} p(\vec{r}_j) = -\sum_{i=1}^{N} p_i \Delta V_i \vec{\nabla} W(|\vec{r}_j - \vec{r}_i|) .$$

Ezzel viszont van egy kis gond. A közelítő sémánk nem garantálhatja a fizikai szabályok teljesítését, így az erők szimmetriáját és következésképpen a lendületmegmaradását sem. Gondoskodnunk kell arról, hogy az i-edik részecskére a j-edik részecske miatt ható erő mindig egyenlő legyen a j-edik részecskére az i-edik részecske miatt ható erővel. A szimmetrikus viszonyt a nyomásból származó erő következő módosításával biztosíthatjuk:

$$\vec{F}_{\text{pressure},j} = -\sum_{i=1}^{N} \frac{p_i + p_j}{2} \Delta V_i \vec{\nabla} W(|\vec{r}_j - \vec{r}_i|) .$$

A viszkozitási taghoz a vektormezőre a Laplace-operátort kell alkalmazni, amit a Laplace-operátornak a simító kernelre való végrehajtásával számíthatunk:

$$\vec{F}_{\text{viscosity},j} = \nu \vec{\nabla}^2 \vec{v} = \nu \sum_{i=1}^{N} \vec{v}_i \Delta V_i \vec{\nabla}^2 W(|\vec{r}_j - \vec{r}_i|) .$$

A nyomásból eredő erőhöz hasonlóan itt is inkább egy szimmetrikus változatot használunk, hogy az erők szimmetrikusak maradjanak:

$$\vec{F}_{\text{viscosity},j} = \nu \sum_{i=1}^{N} (\vec{v}_i - \vec{v}_j) \Delta V_i \vec{\nabla}^2 W(|\vec{r}_j - \vec{r}_i|) .$$

A külső erőket közvetlenül alkalmazhatjuk a részecskékre. A részecske–test ütközéseket a részecske sebességnek a test felületére merőleges komponensének tükrözésével kezelhetjük.

Miután minden erőt kiszámolunk és a (16.8) egyenlet idő szerinti deriváltjait véges differenciákkal közelítjük a következőképpen kaphatjuk meg a részecskék új pozícióit és sebességeit:

$$\begin{aligned} \vec{r}_i(t+\Delta t) &= \vec{r}_i(t) + \vec{v}_i(t)\Delta t \ , \\ \vec{v}_i(t+\Delta t) &= \vec{v}_i(t) + (\vec{F}_{\text{pressure},i} + \vec{F}_{\text{viscosity},i} + \vec{F}_{\text{external},i})\Delta V_i \Delta t/m \ . \end{aligned}$$

Vegyük észre, hogy ez is egy rossz stabilitási tulajdonságokkal rendelkező előrelépéses euleri integrálási séma. Ehelyett stabil alternatívát, például ***Verlet-integrálást*** is használhatunk [69].

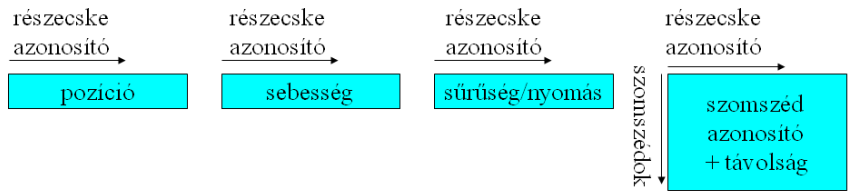
A lagrange-i megközelítés véges számú részecskét követ, ahol a részecskékre ható erők a többi részecske pozíciójától és tulajdonságaitól függenek. Így egy N részecskéből álló rendszer időléptetéséhez elég $O(N^2)$ kölcsönhatást megvizsgálni. Az ilyen jellegű feladatokat ***többtest problémának*** hívják.

Lagrange-i megoldás a GPU-n

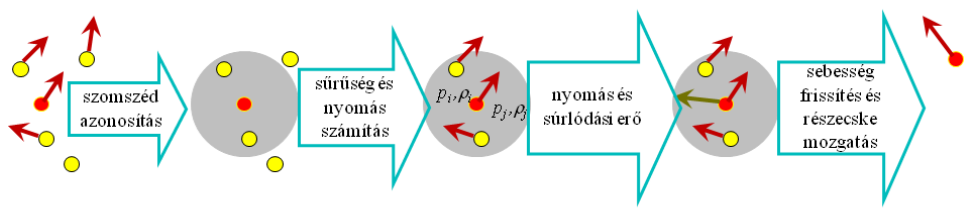
GPGPU keretrendszerben a részecskék tulajdonságait egydimenziós tömbként tárolhatjuk a globális memóriában, vagy egydimenziós textúrából olvashatjuk ki őket. Grafikus API esetén csak textúrával reprezentálhatjuk a részecskejellemzőket. Az adatok textúrából olvasásának csak a jobb gyorsítótárazás miatt van előnye, mivel a textúraszűrő hardver itt nem hasznos. Egy gyűjtőtípusú módszer minden vezérelt részecskéhez rendelne egy szálat és a szál a többi részecske hatását számolná. Mivel a simító függvény csak egy korlátos tartományban nem zérus, csak a simító függvény maximális sugaránál közelebbi részecskéknek lehet hatása. Érdemes ezeket a közeli, úgynevezett szomszédos részecskéket csak egyszer azonosítani és a globális memória kétdimenziós textúrájában eltárolni, majd a későbbi kernelekben a szomszédsági információt újabb számítások nélkül felhasználni.

A GPGPU megközelítésben három egydimenziós tömbre van szükségünk a részecskék pozíciójának, sebességének, sűrűségének és nyomásának reprezentációjára, valamint egy kétdimenziós tömbre a szomszédos részecskék tárolásához (16.13. ábra). A grafikus megközelítésben ezek egy- vagy kétdimenziós textúrák. Minden részecskére futtathatunk egy kernelt, vagy egy pixelárnyalót.

Minden olyan lépésnél, ahol a részecskéknek valamely korábban számított tulajdonságát szeretnénk felhasználni, a részecske feldolgozását menetekre vagy kernelekre kell osztani. Az első menet a részecskék szomszédainak azonosítása, vagyis a simítókernel sugarán belüli többi részecske megtalálása.



16.13. ábra. Tömbökben vagy textúrákban tárolt adatstruktúrák. A részecske pozíció és sebesség egydimenziós float3 típusú textúrába kerül. A számított sűrűséghez és nyomáshoz egydimenziós float2 típusú textúrát használunk. Végül, egy kétdimenziós textúra azonosítja minden részecskéhez a hozzá közel lévő és ezért hatással bíró további részecskéket.

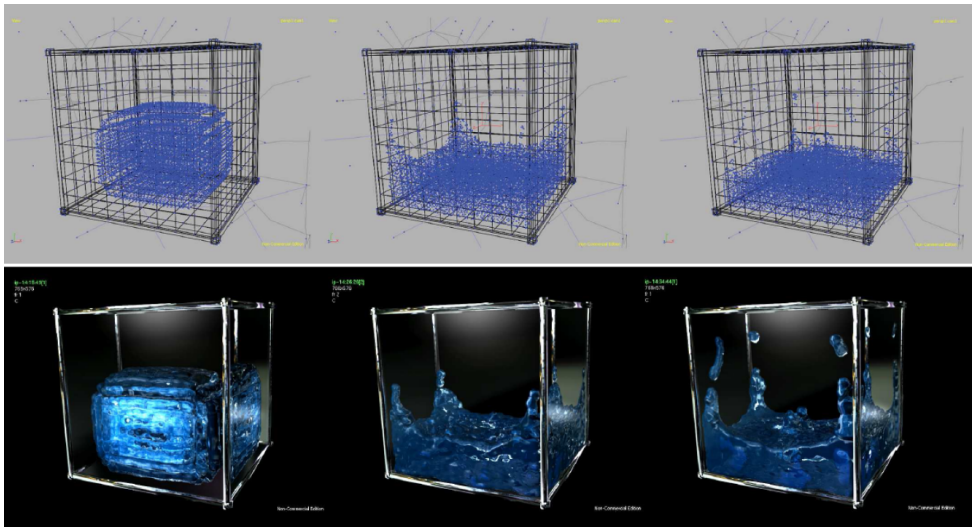


16.14. ábra. A lagrange-i megoldás egy időlépése, amely a pirossal (feketével) színezett részecskét a sárga (fehér) szomszédai szerint mozgatja.

Ennek a lépésnek a kimenete egy olyan kétdimenziós tömb, ahol az oszlopok a részecske indexével címezhetők és az oszlop elemei a közeli részecskék indexei és távolságai.

A második menet a sűrűséget és nyomást számítja ki, a közeli részecskék száma és távolságai alapján. Ezután minden részecske nyomása minden szál számára elérhető lesz. A harmadik menet az erőket számolja a közeli részecskék nyomása és sebessége alapján. Végül minden részecske megkapja az újonnan kiszámolt sebességét és új pozíciójára mozog (16.15. ábra).

A részecskepozíciók ismeretében a rendszert többféleképpen megjeleníthetjük. Rajzolhatunk például pontot, vagy apró gömböt minden részecskéhez



16.15. ábra. Lagrange-i megoldóval készült animációk a részecskéket gömbökkel megjelenítve (felső kép) és izofelület megkeresésével (alsó kép) [121].

(16.15. ábra felső képe). Egy másik megoldás a részecskéket a képernyőre keni az euleri megoldásban látott rajzolási stílushoz hasonló eredménnyel (16.12. ábra). Végül az áramló közeg felszínét is megkereshetjük és a felületen a geometriai optika törvényeit követve fényvisszaverődést és törést számolhatunk (16.15. ábra alsó képe). A közeg felszíne a sűrűségmező szintfelülete, amely az alábbi implicit egyenlet megoldásaként definiálható:

$$\rho(\vec{r}) = \rho_{\text{iso}} \ .$$

Ezt az egyenletet a virtuális kamerából látható pontokra kisugármasírozás [284] (ray marching) segítségével oldhatjuk meg. A szem pozíciójából a képponton keresztül haladó sugár mentén kis lépéseket teszünk. Minden $\vec{r}_s$ mintapontban ellenőrizzük, hogy a $\rho(\vec{r}_s)$ sűrűség meghaladja-e a megadott ρ_{iso} értéket. Az első olyan lépésben, amikor ez megtörténik, megtaláltuk a sugár-szintfelület metszéspontot. A sugarakat innen a visszaverődési és törési irányokban folytathatjuk. Ezen irányok számításához a szintfelület normálvektorára is szükség van, amit a sűrűségmező gradienseként számolhatunk.

Megjegyzések a fejezethez

A GPU-k fix transzformációs és multitextúrázó hardverei egy jó évtizeddel ezelőtt váltak programozható csúcspont- és pixelárnyalókká. A GPU-k nagy lebegőpontos számítási teljesítménye gyorsan megteremtette az igényt arra, hogy ne csak az inkrementális képalkotásra, de más algoritmusokra is használni lehessen őket. Az első GPGPU algoritmusok még kapcsolódtak a grafikához, mint például a ***sugárkövetés***, vagy a természeti jelenségek szimulációja. A GPGPU számítások korai éveiről kiváló összefoglaló található a [225]-ban. A számítógépes grafika kutatói azóta is lelkesen dolgoznak az új hardverrel, hiszen annak általános célú képességei lehetővé tették az inkrementális képalkotástól alapvetően különböző algoritmusok megvalósítását. A legfontosabb irányok a fizikai alapú fényterjedés szimulációja, amit globális illuminációnak [286] hívnak, valamint a merev testek mozgásának és ütközéseinek fizikai szimulációja és az áramlásszámítás, amelyek valós idejű rendszerekben és játékokban valószerű szimulációt tettek lehetővé. A GPU Gems könyvsorozat [77, 237, 215] és ShaderX (ma már GPU Pro [70]) sorozat hasonló eljárások terjedelmes gyűjteményét alkotják.

A CUDA és OpenCL platformok megjelenése óta a magas teljesítményigényű számítások minden területén megjelentek GPU alapú megoldások. A GPGPU.org weboldal és NVIDIA weboldala [219, 218] cikkek és programok tárházai, egyben a GPU alapú megközelítés minden területen meglevő elfogadottságának bizonyítékai. Sikeres GPU alkalmazások születtek a következő, sok számítást igénylő területeken, a teljesség igénye nélkül: mindenfajta fizikai jelenség szimulációja, differenciálegyenletek megoldása, tomográfiás rekonstrukció, számítógépes látás, adatbáziskeresés, adattömörítés, lineáris algebra, jelfeldolgozás, molekuladinamika és molekuladokkolás, pénzügyi informatika, víruskeresés, végeselemes módszerek, Monte-Carlo módszerek, számítógépek szimulációja (CNN, neurális hálózatok, kvantumszámítógépek), mintaillesztés, DNS-szekvenciálás, kriptográfia, digitális holográfia, kvantumkémia stb.

Ha skálázható rendszert szeretnénk, amit nem határol be egyetlen GPU kártya memóriájának mérete, GPU klasztert is építhetünk. Egyetlen PC-be akár 4 GPU-t is szerelhetünk, és az összekapcsolható PC-k száma korlátlan [322]. Az ilyen rendszerekben azonban már a kommunikáció lesz a szűk keresztmetszet, mert a jelenlegi kommunikációs csatornák nem versenyezhetnek a GPU-k számítási teljesítményével.

17. Hálózatok szimulációja (Gyires Tibor)

Ebben a fejezetben olyan módszereket és technikákat tárgyalunk, melyekkel szimulálhatjuk különböző hálózati számítógéprendszerek és alkalmazások működését. Egy hálózati rendszer vagy alkalmazás teljesítményének a hálózat fizikai megépítését, illetve az alkalmazás kibocsátását megelőző jellemzésére a hálózattervezés és menedzselés területén a szimuláció az egyik legelterjedtebb módszer.

17.1. A szimuláció típusai

Egy hálózati rendszer hálózati elemek – mint például a forgalomirányítók, kapcsolók, kapcsolatok, felhasználók és alkalmazások – olyan halmaza, melynek elemei együttműködnek valamilyen cél érdekében. A szimulációs tanulmány tárgya lehet egy olyan rendszer is, mely része egy másik rendszernek, mint például egy alhálózat. A hálózati rendszer állapota azoknak a lényeges változóknak és paramétereknek a halmaza, amelyek leírják a rendszert egy bizonyos időben, amely magában foglalja az adott dolog vizsgálatát. Például, ha egy kapcsolat kihasználtságára vagyunk kíváncsiak, akkor csak a link által másodpercenként átvitt bitek számát és a kapcsolat teljes kapacitását akarjuk tudni, és nem a kapcsolat által összekötött kapcsolókban az egyes kapuk számára rendelkezésre álló puffermennyiséget.

A hálózat fizikai modelljének megépítése helyett egy matematikai modellt építünk, mely tükrözi a hálózatelemek működését és a közöttük lévő logikai és mennyiségi kapcsolatokat. A hálózatelemek közötti kapcsolatok megváltoztatásával a hálózatot annak fizikai megépítése nélkül elemezhetjük azt feltételezve, hogy a modell a valós rendszerhez hasonlóan viselkedik, tehát ez egy érvényes modell. Például, a kapcsolat kihasználtságát analitikusan kiszámíthatjuk az $U = D/T$ képlet felhasználásával, ahol D az adott idő alatt elküldött adatmennyiség, T pedig a kapcsolat kapacitása bit/másodpercben. Ez nagyon egyszerű modell, amely ritkán alkalmazható valós feladatok esetén. Sajnos, a problémák nagy része túl bonyolult ahhoz, hogy a felmerülő kérdéseket egyszerű matematikai egyenletekkel megválaszolhassuk. A nagyon bonyolult esetekben a szimulációs technika jobb.

A szimulációs modellek többféleképpen osztályozhatók. A leggyakoribb osztályozások a következők.

- *Statikus és dinamikus szimulációs modellek.* A statikus modell az időtől

függetlenül jellemzi a rendszert, a dinamikus modell pedig egy időben változó rendszert ír le.

- *Sztochasztikus és determinisztikus modellek.* Ha a modell egy olyan rendszert ír le, amely véletlentől függő elemeket tartalmaz, akkor sztochasztikus modellnek nevezzük, egyébként determinisztikusnak. A hálózati modelleket megalapozó sorbanállási rendszerek véletlenszerű komponenseket tartalmaznak, mint például a csomagok beérkezési ideje egy adott sorba, a sorok kiszolgálási ideje, egy kapcsoló kapu kimenete stb.

- *Diszkrét és folytonos modellek.* A folytonos modell egy olyan rendszert ábrázol, melynek változói időben folyamatosan változnak. Például a differenciálegyenletek definiálják az összefüggéseket egyes állapotváltozók változásának a mértékére az idő változása szerint. Egy diszkrét modell egy olyan rendszert jellemez, ahol az állapotváltozók diszkrét időpillanatokban változnak meg. Az egyes események ezekben a diszkrét időpontokban fordulhatnak elő és változtathatják meg a rendszer állapotát. Például, egy csomag érkezése egy forgalomirányítóhoz egy bizonyos időpillanatban egy olyan esemény, amely megváltoztatja az forgalomirányító kapu-pufferének állapotát.

A továbbiakban mi dinamikus, sztochasztikus és diszkrét modelleket tételezünk fel, és diszkrét-esemény szimulációs modellekként hivatkozunk rájuk.

A számítógépes kommunikáció összetett természetének következtében a hálózati modellek is összetettek. A számítógépes programok fejlesztése az egyes szimulációs problémákra egy lehetőség, de ez nagyon sokáig tart és nem hatékony. Az utóbbi időben a szimulációs és modellező csomagok alkalmazása szokásossá vált, ami megtakarítja a kódolási időt és lehetővé teszi a modellező számára, hogy a programozási részletek helyett a modellezési problémára koncentráljon. Első pillantásra ezeknek a hálózatszimulációs és modellezési csomagoknak – mint például a COMNET, OPNET – használata azzal a veszéllyel jár, hogy a modellezőnek meg kell bíznia olyan modellezési technikákban és rejtett eljárásokban, amelyek jogilag védettek lehetnek, valamint nem nyilvánosak. A következő alfejezetekben azt tárgyaljuk, hogy a szimulációs módszertan hogyan győzi le az előbb említett félelmeinket az érvényesítési eljárások alkalmazásával, melyek célja, hogy megbizonyosodjunk arról, vajon a valós hálózati rendszer ugyanúgy működik-e majd, mint ahogyan azt a szimulációs modell előrejelezte.

17.2. Hálózatok modellezésének és szimulációjának szükségessége

Az egyre több adat, számítógép, tároló rendszer és hálózat világában a rendszerek tervezése és menedzselése egyre nagyobb kihívássá válik. Amint a hálózatok gyorsabbá, nagyobbá és bonyolultabbá válnak, a hagyományos számítások már nem ésszerű megközelítései az új hálózati technológiáknál egy új hálózatterv megvalósításának és több millió dolláros befektetéseknek az érvényesítésénél. Ezek a bonyolult számítások és táblázatok már nem megfelelő eszközök a hálózati forgalom sztochasztikus természete, valamint az egész rendszer összetettsége miatt.

A különböző szervezetek egyre inkább azoktól az új hálózati technológiáktól és hálózati alkalmazásoktól függenek, melyek támogatják az üzleti szükségleteiket. Ennek eredményeképp, a gyenge hálózat-teljesítménynek súlyos következményei lehetnek az üzleti tevékenységükre nézve. Az egyes tervezési célok eléréséhez tartozó különböző alternatív megoldások kiértékelésében a hálózattervezők egyre inkább számítanak az olyan módszerekre, melyek segítik őket a számos javaslat kiértékelésében a végső döntés meghozatala és a valódi rendszer megépítése előtt. Egy széleskörűen elfogadott módszer a szimulációs teljesítmény-előrejelzés. A hálózattervező használhat egy szimulációs modellt a tervezési változatok elemzésére és egy új, vagy egy már létező rendszer módosításainak tanulmányozására azok fizikai megépítése nélkül. Egy szimulációs modell tükrözheti egy hálózat topológiáját és a hálózatban elvégzett feladatokat, hogy segítségével statisztikai eredményeket kapjuk a hálózat teljesítményéről.

Fontos a szimuláció és az emuláció közötti különbség megértése. Az emuláció célja az, hogy utánozza az eredeti hálózatot, és reprodukáljon minden eseményt, amely az egyes hálózatelemekben és alkalmazásokban bekövetkezik. Szimuláció esetén a cél az, hogy olyan statisztikai eredményeket állítsunk elő, amelyek kifejezik az egyes hálózatelemek viselkedését és funkcióit. Egy diszkrét-esemény szimuláció során meg akarjuk figyelni az eseményeket, amint azok megtörténnek az adott időben, és teljesítmény-jellemzőket szeretnénk meghatározni, hogy következtetéseket vonhassunk le a hálózat teljesítményéről, mint például a kapcsolat-kihasználtságokról, a válaszidőkről, a forgalomirányítók puffereinek méreteiről stb.

A nagyon sok hálózatelemet tartalmazó hálózatok szimulációja esetén olyan nagy modellt kaphatunk, amelynek elemezése a szimuláció alatt előállított nagy mennyiségű statisztika miatt bonyolult. Ezért ajánlatos a hálózatnak csak azon részeit modellezni, amelyek lényegesek a szimulációval megkapni kívánt statisztikákat illetően. Továbbá szükségszerű, hogy a modell csak azokat a részleteket tartalmazza, amelyek a szimuláció céljai miatt

fontosak. A hálózattervezők általában a következő célokat tűzik ki:

- *Teljesítménymodellezés.* Statisztikák előállítása a kapcsolatok, forgalomirányítók, kapcsolók, pufferek stb. teljesítmény paramétereiről.
- *Hibaelemzés.* A hálózatelemek hibás működése következményeinek elemzése.
- *Hálózattervezés.* Különböző hálózattervek statisztikáinak összehasonlítása az egyes tervezési javaslatok követelményeinek kiértékelése céljából.
- *Hálózati erőforrások tervezése.* Olyan változtatások hatásának mérése a hálózat teljesítményén, mint pl. az új felhasználókkal, alkalmazásokkal vagy hálózatelemekkel való bővítés.

A céloktól függően ugyanahhoz a hálózathoz különböző szimulációs modellekre lehet szükség. Például, ha a modellező meg akarja határozni egy protokoll új szolgáltatása által okozott többletterhelést a kommunikációs kapcsolatokon, akkor a modell ugrópontjainak csak az új szolgáltatás által előállított forgalmat kell ábrázolnia. Ha viszont a modellező egy alkalmazás válaszidejét akarja vizsgálni maximális forgalom-terhelés esetén, akkor mellőzheti az előző modellbeli protokoll új szolgáltatásának megfelelő forgalmat.

Egy másik fontos kérdés a modell finomsága, azaz hogy milyen szintű részletekig modellezzük az egyes hálózatelemeket. Például el kell döntenünk, hogy egy forgalomirányító belső felépítését akarjuk modellezni, vagy pedig egy egész csomagkapcsolt hálózatot. Az első esetben meg kell adnunk a forgalomirányító belső komponenseit, a processzorok számát és sebességét, a buszok típusait, a kapuk számát, a kapu-pufferek mennyiségét továbbá az ezen komponensek közötti kölcsönhatásokat is. De ha alkalmazásszinten akarjuk vizsgálni a válaszidőt az egész csomagkapcsolt hálózatban, akkor a forgalomirányítók belső részletei helyett inkább az alkalmazások és protokollok típusait, a hálózat topológiáját és a kapcsolatok kapacitását kell meghatároznunk. Habár a forgalomirányítók alacsony szintű működése hatással van a végpontok közötti válaszidőre, ennek részletes modellezése az egész hálózatot tekintve nem járul hozzá meghatározóan a szimulációs eredményekhez. A nanoszekundum nagyságrendű belső működési részletek modellezése nem járul hozzá jelentősen a végpontok közötti mikroszekundum vagy másodperc nagyságrendű késleltetés vizsgálatához. A nagyobb modell-finomságból eredő további pontosságot erősen ellensúlyozza a modell bonyolultsága, és ezeknek a részleteknek a modellbe foglalása által igényelt munka és idő.

Az egyszerűsítés történhet statisztikai függvények alkalmazásával is. Például a cellahibákat egy ATM hálózatban nem feltétlenül kell úgy modellezni, hogy egy kommunikációs kapcsolaton a cella fejlécében megváltoztatunk egy bitet, ezáltal idézve elő rossz CRC-t a fogadónál. Használhatunk

egy statisztikai függvényt is annak eldöntésére, hogy egy cella megsérült vagy elveszett-e, tehát így a cella részleteit nem kell megadnunk a cellahibák modellezéséhez.

Ezek a példák azt szemléltetik, hogy a hálózatszimuláció célja nem a hálózat emulációja, hanem a funkcionalitás bizonyos vizsgálatok céljára történő reprodukciója.

17.3. A telekommunikációs hálózatok típusai

Egy kommunikációs hálózat hálózatelemekből, csomópontokból (küldőkből és fogadókból) és az őket összekötő kommunikációs közegekből áll. A számos hálózat osztályozási kritérium közül mi kettőt használunk: az átviteli technológiát és méretet. A méret vagy távolság szintén meghatározza a hálózatban alkalmazott technikát, amely például lehet vezetékes vagy vezetéknélküli. Két vagy több hálózat összekapcsolását ***internetworknek*** nevezzük. A legismertebb internetwork az Internet.

Az átviteli technológiától függően nagyjából két csoportba oszthatjuk a hálózatokat, üzenetszórásos és pont-pont kapcsolatú hálózatok:

- Az *üzenetszórásos hálózatokban* egy kommunikációs csatorna van megosztva az összes csomópont között. A kommunikáció úgy történik, hogy a csomópontok a minden más csomópont által küldött csomagokat vagy kereteket is megkapják. A keretben lévő cím határozza meg a címzettet vagy címzetteket, és csak a címzett csomópontok dolgozzák fel a keretet. Az üzenetszórásos technológiák azt is lehetővé teszik, hogy a keretet az összes csomópontnak címezzük. Az ilyen üzenetszórásos kereteket a hálózat összes csomópontja feldolgozza. A kereteket címezhetjük csomópontok egy csoportjának a tagjainak is, vagy közülük egy tetszőleges csomópontnak is. Az előbbit többesküldésnek, az utóbbit bárkinek való küldésnek nevezzük.

- A *pont-pont kapcsolatú hálózatok* csomópontpárok közötti kapcsolatok sokaságából állnak. Itt egy adott küldőtől egy adott címzetthez küldött csomagot vagy keretet esetleg más csomópontokon keresztül kell továbbítani. Ezek tárolják és továbbítják azt, amíg el nem éri célját.

A másik osztályozási szempont, vagyis a lefedett fizikai terület nagysága alapján a következőképpen csoportosíthatjuk a hálózatokat:

- A *személyes hálózati környezetek* (**P**ersonal **A**rea **N**etwork – PAN) egy adott személy igényeit elégítik ki. Például egy billentyűzetből, egérből és *digitális személyi asszisztensből* (**P**ersonal **D**igital **A**ssistant – PDA) álló, vezetéknélküli hálózat tekinthető egy személyes hálózati környezetnek.

- A *0helyi halozatok@helyi hálózatok* (**L**ocal **A**rea **N**etwork – LAN) egy korlátozott méretű területet fednek le, tipikusan magánszemélyek, részlegek, kisebb otthoni szervezetek vagy épületek adott emeletei birtokolják őket. A helyi hálózat munkaállomásokat, kiszolgálókat és megosztott erőforrásokat kapcsol össze. A LAN-ok csoportosíthatók továbbá az átviteli technológia (melynek sebességét bit/másodpercben mérjük) és a hálózati topológia szerint. Az átviteli technológiák sebessége a hagyományos 10 Mbps-tól 10 Gbps sebességűig terjed. Topológia szerint vannak busz és gyűrű hálózatok, továbbá kapcsolt LAN-ok.
- A *városi hálózatok* (**M**etropolitan **A**rea **N**etwork – MAN) nagyobb területet hidalnak át, például egy várost. Egy széleskörűen alkalmazott MAN a kábel-tv hálózat, amely nem csak az egyirányú TV adásokat, hanem kétirányú Internet szolgáltatásokat is biztosít az átviteli spektrum nem használt részében. Más MAN technológiák például az *optikai szálas elosztott adatinterfész* és a következőkben tárgyalt vezetéknélküli IEEE (**I**nstitute of **E**lectrical and **E**lectronical **E**ngineers) technológiák.
- A *nagy kiterjedésű hálózatok* (**W**ide **A**rea **N**etwork – WAN) nagy földrajzi területet fednek le, egy államot, országot vagy akár egy kontinenst is. Egy WAN alhálózatokban összekapcsolt hosztokból (kliensekből és szerverekből) áll. Az alhálózatok továbbítják az üzeneteket a forrástól a cél hosztig. Egy alhálózat tartalmazhat számos átviteli vonalat, melyek mindegyike speciális hardver eszközöket, úgynevezett forgalomirányítókat kapcsol össze. Az átviteli vonal több különböző közeg is lehet, rézdrót, optikai üvegszál, vezetéknélküli kapcsolat stb. Mikor egy üzenetet el kell küldeni egy vagy több hosztnak, a küldő az üzenetet kisebb részekre osztja, melyeket ***csomagoknak*** nevezünk. Amikor egy csomag megérkezik egy átviteli vonalon, a forgalomirányító tárolja azt, kiválaszt egy kimenő vonalat és azon továbbítja. A kimenő vonal kiválasztása egy forgalomirányítási algoritmus alapján történik. Végül az egyesével továbbított csomagokból a cél hoszt(ok) újra összeállítják az eredeti üzenetet.

A vezetéknélküli hálózatok a következő osztályokba sorolhatók: rövidtávú rádió-hálózatok, vezetéknélküli LAN-ok és vezetéknélküli WAN-ok.

- Rövidtávú hálózatoknál az egyes eszközök 6–10 méteren belül vannak összekapcsolva rövidtávú rádiókapcsolatokkal. Ilyenek például a Bluetooth, a különböző komponensek, digitális kamerák, a ***globális helymeghatározó rendszer*** (**G**lobal **P**ositioning **S**ystem – GPS) eszközei, a fejhallgatók, számítógépek, szkennerek, monitorok és billentyűzetek. A komponensek elsődleges-másodlagos viszonyban vannak. A fő rendszeregység, az elsődleges komponens irányítja a másodlagos komponensek működését. Az elsődleges komponens határozza meg, hogy milyen címeket

használnak a másodlagos eszközök, továbbá, hogy melyik frekvenciákon továbbíthatnak adatokat.

- A vezetéknélküli LAN számítógépekből és rádió-modemmel és antennával felszerelt hozzáférési pontokból áll, melyek lehetővé teszik az adatok küldését és fogadását. A számítógépek kommunikálhatnak egymással közvetlenül vagy egy hozzáférési ponton keresztül, ami más hálózatokhoz is kapcsolhatja őket. A lefedett terület általában 100 méter körül van. A vezetéknélküli LAN protokollok az IEEE 802.11 szabványok családjába tartoznak, és 11–108 Mbps sebességet biztosítanak.
- A vezetéknélküli WAN-ok lehetnek kis és nagy sávszélességű hálózatok is. A kis sávszélességű rádió-hálózatok eddig kifejlesztett három generációját a celluláris telefonrendszerek használják. Az első generációt csak hangkommunikációra tervezték, ez analóg jelzést használt. A második generáció szintén csak hangot továbbított, de digitális átviteli technológiára alapozva. Az aktuális harmadik generáció digitális és hangot és adatokat is továbbít legfeljebb 2 Mbps sebességgel. Negyedik és további generációs celluláris rendszerek is fejlesztés alatt vannak. A nagy sávszélességű WAN-ok a telefonrendszerek használatát elkerülve biztosítanak nagy sebességű hozzáférést az otthonokból és cégektől. Az IEEE 802.16 szabvány az IEEE 802.11 szabvánnyal ellentétben épületekhez és nem mobil állomásokhoz továbbítja a szolgáltatásokat, és sokkal magasabb, 10–66 GHz frekvenciatartományban üzemel. A távolság az épületek között több kilométer is lehet.
- A vezetékes és vezetéknélküli otthoni hálózatok a különböző, Interneten keresztül elérhető eszközök összekapcsolásával egyre inkább népszerűbbek lesznek. Az otthoni hálózatok állhatnak PC-kből, laptopokból, PDA-kból, TV-kből, DVD-kből, videókamerákból, MP3 lejátszókból, mikrohullámú sütőkből, hűtőszekrényekből, lámpákból, riasztókból, kihasználtságmérőkből stb. Sok háztartás ma már fel van szerelve nagy sebességű Internet-hozzáféréssel (kábel modem, DSL stb.), amin keresztül igény szerint tölthetnek le zenét és filmeket.

A kommunikációs hálózatok különböző komponensei és típusai megfelelnek a szimulációs modellépítés konstrukcióinak és különböző lépéseinek. Tipikusan a hálózati topológiát építik fel először, majd hozzáadják a forgalom forrásait, céljait, a terhelést és a hálózati működés paramétereinek beállítását. A szimulációt vezérlő paraméterek meghatározzák a kísérletet és a szimuláció lefutását. A szimuláció indítása előtt a szimuláció alatti és utáni elemzéshez különböző statisztikai jelentések aktiválhatók. Statisztikai eloszlások állnak rendelkezésre ahhoz, hogy leírjuk a beépített analitikus eloszlások speciális paraméterezéseit. Amint a modell elkészült, a modellező új modellkönyvtára-

kat készíthet, melyek újra felhasználhatók más modellekben is.

17.4. Teljesítményjellemzők szimulációhoz

Ebben az alfejezetben a hálózatok olyan jellemzőinek a ***teljesítménymértékeknek*** – egy nem kimerítő listáját tekintjük át, melyek meghatározó hatással vannak a hálózat teljesítményére. Általában ezek a jellemzők képezik a hálózat-modellezésnek, azaz a számítógéphálózatok statisztikai elemzésének, tervezésének és optimalizálásának céljait. Alapvetően a hálózatmodelleket úgy készítjük, hogy megadjuk egy sorbanállási rendszerben az igény érkezési és kiszolgálási intenzitások statisztikai eloszlását, ami azután meghatározza ezeket a jellemzőket.

- *Kapcsolat kapacitás.* A csatorna vagy kapcsolat-kapacitás a kapcsolat által időegységenként továbbított üzenetek száma. Ezt általában bit/másodpercben mérik. Az információelmélet egyik leghíresebb eredménye Shannon csatornakódolási tétele: „Minden csatornára létezik olyan kód, ami lehetővé teszi a csatornán való hibamentes adattovábbítást R intenzitással, feltéve, hogy $R \leq C$, ahol C a csatorna kapacitása." Egyenlőség csak akkor lehetséges, ha a jel/zaj hányados (**S**ignal-to-**N**oise **R**atio – SNR) végtelen.
- *Sávszélesség.* A sávszélesség a hálózati jelek számára rendelkezésre álló legalacsonyabb és legmagasabb frekvenciák közötti különbség. A sávszélesség kifejezést szintén használják egy adott kapcsolat vagy protokoll áteresztőképességének leírására, kilobit, megabit, terabit stb. másodpercben mérve.
- *Válaszidő.*A ***válaszidő*** az az idő, amire a hálózati rendszernek szüksége van ahhoz, hogy megválaszolja egy igényforrás kérését. A válaszidő tartalmazza a célállomáshoz való továbbítás idejét, a feldolgozási időt a forrásnál, a célnál, valamint a közbeeső hálózatelemeknél, továbbá tartalmazza a válasznak a forráshoz való átvitelének idejét. Az átlagos válaszidő a hálózat teljesítményének fontos mértéke. A felhasználók számára a minél kisebb válaszidő a legkedvezőbb. A válaszidő statisztikáknak (átlag és szórás) állandóknak kell lenniük, és nem szabad, hogy napszaktól függjenek. Meg kell jegyeznünk viszont, hogy az alacsony átlagos válaszidő nem garantálja azt, hogy nem lesznek hálózati torlódások miatt rendkívül hosszú válaszidők is.
- *Késés.* A ***késés*** vagy késleltetés az az időmennyiség, ami egységnyi adatnak egy hálózati kapcsolaton való továbbításához szükséges. A késleltetés és a sávszélesség az a két tényező, ami meghatározza egy kapcso-

lat sebességét. A késleltetés tartalmazza a terjedési késleltetést (azt az időt, ami alatt az elektronikus vagy optikai jelek megteszik a két pont közötti távolságot) és a feldolgozási időt. Például egy földi állomásnak egy másik földi állomással való műholdas kommunikációs kapcsolatának késése (legalább 34000 km mindkét út) körülbelül 270 milliszekundum. Az USA keleti és nyugati partjai közötti válaszolási késleltetés körülbelül 100 milliszekundum, globálisan 125 milliszekundum. Egy több szegmensből álló adatútnak a forrás és cél végpontok közötti késleltetését nem csak a jel terjedési sebessége, hanem az útvonalon lévő hálózati eszközök, forgalomirányítók, kapcsolók is befolyásolják, melyek tárolják, feldolgozzák, irányítják, kapcsolják, és becsomagolják az adatokat. A hibás csomagok és cellák, a jelveszteség, a kapcsolat és eszközhibák, valamint túlterhelések szintén hozzájárulnak a hálózati késleltetéshez. Hibás cellák és csomagok esetén szükség van azoknak a forrástól való teljes újraküldésére. Az ilyen a csomagokat általában eldobják – azt feltételezve, hogy később újraküldik őket. Ez lassulást és a pufferek túlcsordulását okozhatja.

- *Forgalomirányító protokollok.* A forrástól a célig a hálózati forgalom valamilyen útvonalon halad keresztül. Egy helyi hálózatban ez az útvonal nem egy problémás kérdés, mivel csak egy út lehetséges bármely forrás és cél között. Ha viszont a hálózat számos céget kapcsol össze számos útvonallal, forgalomirányítóval és kapcsolattal, a legjobb út vagy útvonalak megkeresése igen fontossá válik. Egy útvonal többféle különböző kapacitású, késleltetésű és megbízhatóságú kapcsolatból állhat. Létesítésük forgalomirányító protokollok segítségével történik. Ezeknek a protokolloknak az a célja, hogy a torlódást elkerülve egy optimális, vagy közel optimális útvonalat találjanak a forrás és a cél között.
- *Forgalomszervezés.* A forgalomszervezés koncepcióját felhasználva jelenleg egy újfajta forgalomirányítási technikát fejlesztenek. A folyamatos hálózatkapacitás-növelés helyett inkább a hálózati erőforrások optimális kiosztását alkalmazzák a torlódások elkerülésére. A forgalomszervezést úgy valósítják meg, hogy a forgalomfolyamokat a fizikai hálózati topológiára képezik le előre meghatározott útvonalak mentén. Fő cél a forgalomirányítók és kapcsolók továbbító kapacitásainak optimális kiosztása. A forgalomszervezés megteremti annak lehetőségét, hogy a forgalomfolyam eltérjen a hagyományos forgalomirányítási protokollok által kiszámított optimális útvonaltól egy kevésbé zsúfolt hálózatrész felé. Kiegyenlíti a terhelést a kapcsolatokon, forgalomirányítókon és kapcsolókon úgy, hogy ezen hálózatelemek egyike se legyen túl kevéssé vagy túlságosan is kihasználva.
- *Protokoll többletterhelés.* A protokollüzenetek és az alkalmazások adatai

protokoll-adategységekbe vannak ágyazva, mint például keretek, csomagok és cellák. A hálózattervezők egyik fő aggálya a protokollok többletterhelése. Ez a többletterhelés a következő kérdést veti fel. Valójában milyen gyorsan tudunk adatokat továbbítani egy adott kommunikációs útvonallal és protokollcsomaggal, azaz mekkora sávszélesség marad valójában az alkalmazásoknak? A legtöbb protokoll további többletterhelést is bevezet sávszélességen belüli protokoll-menedzselési funkciókkal kapcsolatban. A kapcsolatfenntartó csomagok, hálózati riasztások, vezérlő és ellenőrző üzenetek, a poll, select és különböző jelzőüzenetek az adatfolyamokkal együtt továbbítódnak.

- *Erős ingadozás.* A hálózati torlódás leggyakoribb oka a forgalom erős ingadozása. Egyes nem régi eredmények nyilvánvalóvá tették a nagy sebességű Internet forgalmának erősen ingadozó voltát és azt, hogy a változékonysága a korábbi feltételezésekkel szemben nem jelezhető előre. Megmutatták, hogy a hálózati forgalom gyakran hasonló statisztikai tulajdonságokkal rendelkezik. A gyakran vagy mindig erősen ingadozó forgalom statisztikailag leírható a *hosszú távú függőség* fogalmának felhasználásával. A hosszú távon függő forgalomnak mindig megfigyelhető erős ingadozása. Az előbbiek egyik következménye az, hogy a különböző adatfolyamok összekeverése, mint ahogyan az az Interneten is történik, nem eredményez egyenletes forgalmat. A helyi és nagy kiterjedésű hálózatokban végzett mérések bebizonyították, hogy a széleskörűen használt Markov-folyamaton alapuló modellek nem alkalmazhatók napjaink hálózati forgalmának leírására. Ha a forgalom Markov-folyamat lenne, az erős ingadozás kiegyenlítődne egy hosszú időintervallum átlagolása során, ellentmondva a megfigyelt forgalomjellemzőknek. Az erősen ingadozó forgalom káros következményeit a 17.9. alfejezetben lévő esettanulmányban vizsgáljuk meg.

- *Keret méret.* A hálózattervezők gyakran aggódnak a nagyméretű keretek miatt, ugyanis az ilyen elveszett keretek és újraküldésük esetén sokkal gyorsabban fel tudják tölteni a forgalomirányítók puffereit mint a kisebbek. Mivel a feldolgozási késleltetés ugyanakkora a nagyobb kereteknél, mint a kisebbeknél, azaz a nagyobb csomagok látszólag hatékonyabbak, de a forgalomirányítók és kapcsolók gyorsabban fel tudják dolgozni a belső soraikat kisebb csomagok esetén. A nagyobb keretek esetén szükség lehet azok feldarabolására is, mivel a *maximális adatátviteli egység* (**M**aximum **T**ransmission **U**nit – MTU) mérete korlátozza a méretüket. Az MTU egy olyan paraméter, ami meghatározza egy IP interfész által átvihető legnagyobb csomag méretét. Másrészt, a kisebb keretek több ütközést okozhatnak az Ethernet hálózatokban vagy kisebb

kihasználtságot a WAN kapcsolatokon.

- *Az eldobott csomagok aránya.* A csomagok az OSI architektúra adatkapcsolati és hálózati rétegében kerülhetnek eldobásra. A szállítási réteg puffereket kezel a nyugtázatlan csomagok számára, és szükség esetén újraküldi őket, hogy ezáltal egy hibamentes kapcsolatot biztosítson a küldő és a fogadó között. Az alsóbb rétegekben eldobott csomagok aránya meghatározza a szállítási rétegben újraküldött csomagok arányát. A forgalomirányítók és kapcsolók a belső pufferek hiánya miatt is eldobhatnak csomagokat. Ha a WAN kapcsolatok zsúfolttá válnak, akkor a pufferek gyorsabban feltöltődnek, ami pedig időtúllépéseket és újraküldéseket okoz a szállítási rétegben. A TCP *lassú indulás algoritmusa* a válaszidő folyamatos becslésével és a továbbítási intenzitásnak a korábbi ingadozásától függő beállításával próbálja elkerülni a torlódást.

17.5. A forgalom jellemzése

A kommunikációs hálózatok véletlentől függő jellemzőkkel továbbítják az adatokat. A hálózat jellemzőinek mértékei véletlen folyamatokból vett statisztikai minták. Ilyen például a válaszidő, a kapcsolat kihasználtság, az üzenetek beérkezési időköze stb. Ebben az alfejezetben áttekintjük a hálózatmodellezésben és teljesítménybecslésben legfontosabb statisztikákat.

A vizsgált hálózatjellemzőnek megfelelő eloszlástípus kiválasztása után a következő lépés az eloszlás paramétereinek becslése. Erre sok esetben a minta átlagát vagy közepét és szórását használják. A fejlettebb szoftver-eszközök tartalmazzák a szükséges számításokat ezekhez a becslésekhez. Az átlag úgy értelmezhető, mint az az érték, ami körül a minta értékei csoportosulnak. A következő egyenletek akkor alkalmazhatók, ha rendelkezésre állnak diszkrét vagy folytonos alapadatok. Legyen $X_1, X_2, \ldots, X_n$ a rendelkezésre álló, n méretű minta. Ennek az átlagát a következő képlet adja meg:

$$\overline{X} = \frac{\sum_{i=1}^{n} X_i}{n} .$$

A minta S^2 szórásnégyzete pedig a következőképpen definiálható:

$$S^2 = \frac{\sum_{i=1}^{n} {X_i}^2 - n\overline{X}^2}{n-1} .$$

Ha az adatok diszkrétek és egy gyakorisági eloszlásba vannak csoportosítva,

eloszlás	paraméter(ek)	becslés(ek)
Poisson	α	$\hat{\alpha} = \overline{X}$
exponenciális	λ	$\hat{\lambda} = 1/\overline{X}$
egyenletes	b	$\hat{b} = ((n+1)/n)[\max(X)]$ (torzítatlan)
normális	μ, σ^2	$\hat{\mu} = \overline{X}$ $\hat{\sigma}^2 = S^2$ (torzítatlan)

17.1. ábra. Leggyakoribb eloszlások paramétereinek becslése.

akkor az előbbi egyenletek a következőképpen módosulnak:

$$\overline{X} = \frac{\sum_{j=1}^{k} f_j X_j}{n} ,$$

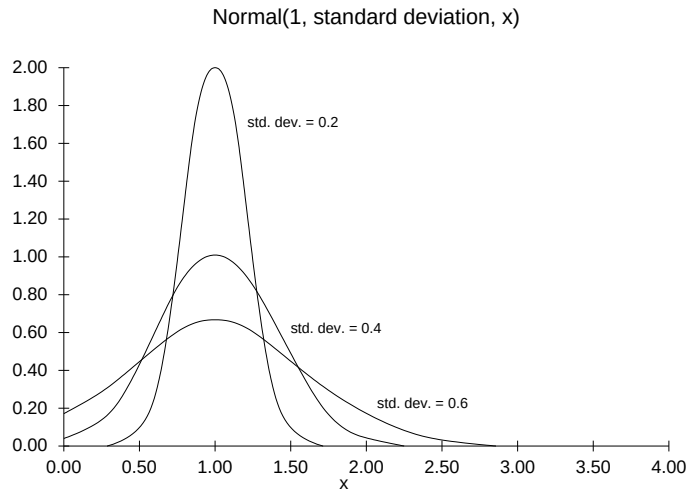
$$S^2 = \frac{\sum_{j=1}^{k} f_j {X_j}^2 - n\overline{X}^2}{n-1} ,$$

ahol k az X különböző értékeinek száma, f_j pedig az X_j érték gyakorisága. Az S szórás az S^2 szórásnégyzet négyzetgyöke.

A szórásnégyzet és a szórás szemlélteti a mintáknak az átlagérték körüli eltérését. A kis eltérés a mintáknak egy erős, középre irányuló tendenciáját mutatja. A nagy eltérés viszont kis tendenciát és nagy statisztikai véletlenszerűséget mutat.

Az eloszlás paramétereinek numerikus becslései azért szükségesek, hogy az adott eloszláscsaládot egy eloszlásra redukálhassuk és tesztelhessük az adott feltevést. Az 17.1. ábra a hálózatmodellezésben előforduló leggyakoribb eloszlások paramétereinek becsléseit foglalja össze. Jelölje α az adott paramétert, $\hat{\alpha}$ pedig a becslését. Ha eltávolítjuk a torzítást a σ^2 becslésétől a normális eloszlásnál és a b-étől az egyenletes eloszlásnál, akkor ezek a becslések a mintaadatokon alapuló maximum likelihood becslések.

A valószínűségeloszlások a valós világban előforduló véletlenszerű változásokat írják le. Habár a változásokat véletlennek nevezzük, a véletlenszerűségnek vannak különböző mértékei; a különböző eloszlások megfelelnek annak, ahogyan a változások bekövetkeznek. Ezért a különböző eloszlásokat különböző szimulációs célokra használják. A valószínűségeloszlások eloszlásfüggvényekkel vannak reprezentálva. Ezek megmutatják, hogy egy bizonyos érték mennyire valószínű. A kumulatív eloszlásfüggvény annak a valószínűségét adja meg, hogy egy bizonyos értékkel egyező, vagy annál kisebb értéket választunk. Például, ha a kumulatív eloszlásfüggvény az 1 értéknél 0.85, akkor ebből az eloszlásból választva az esetek 85%-ában 1-nél kisebb

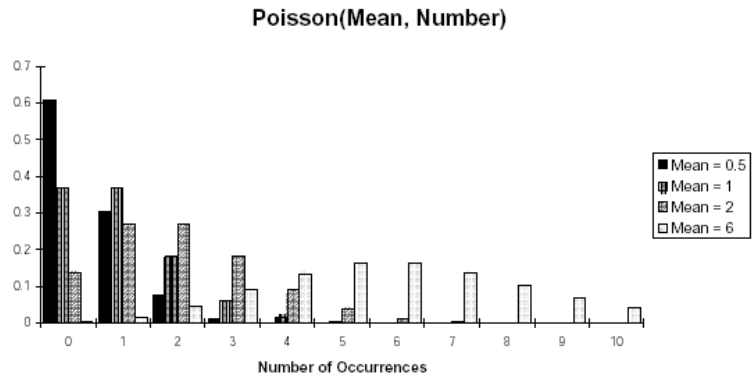


17.2. ábra. Normális eloszlás.

számot kapunk. A kumulatív eloszlásfüggvény értéke egy adott pontban megegyezik a megfelelő sűrűségfüggvény-görbe alatt a ponttól balra lévő területtel. Mivel a sűrűségfüggvény alatti teljes terület egyenlő 1-gyel, ezért a kumulatív eloszlásfüggvény 1-hez tart, ahogyan a pozitív irányban haladunk előre. A modellezési esetek többségében a modellezőnek nem kell tudnia minden részletet ahhoz, hogy sikeresen felépítsen egy szimulációs modellt. Elég azt tudnia, hogy melyik eloszlás a legmegfelelőbb az adott esetre.

Most ismertetjük a leggyakoribb statisztikai eloszlásokat. A megfelelő sűrűségfüggvények ábrázolására a COMNET szimulációs modellezési eszközt használjuk. Gyakorlati szempontból egy sűrűségfüggvény egy hisztogrammal közelíthető az összes előfordulás gyakoriságainak valószínűségekké való konvertálásával.

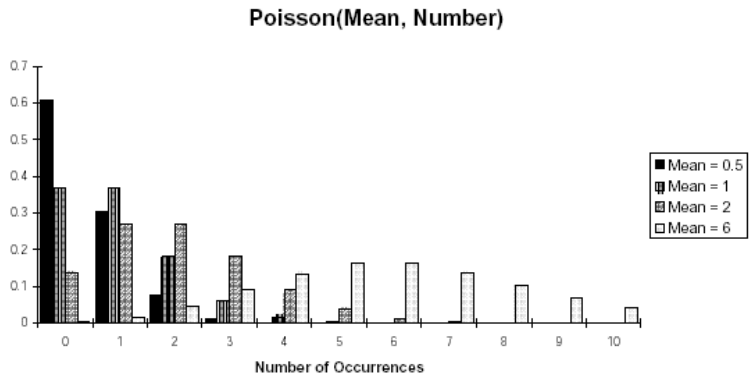
- *Normális eloszlás.* Tipikusan egy összetett folyamat eloszlásának modellezésére használható, ahol a folyamat komponens folyamatok összegeként írható le. Például, egy fájl hálózaton való átvitelének ideje (válaszidő) a fájlt felépítő blokkok átviteli idejeinek az összegével egyezik meg. A modellező eszközökben a normális eloszlás két pozitív valós számmal adható meg: a várható értékkel és a szórással. A függvényértékek szintén pozitív valós számok. Az x paraméter azt adja meg, hogy melyik véletlen számsorozatot alkalmazzuk a mintakészítéshez. Ez az eloszlás szintén gyakran használt üzenetek méretének modellezésére. Például, egy üzenet leírható 20000 bájt átlagos mérettel és 5000 bájt szórással.
- *Poisson-eloszlás.* Ez az eloszlás az egy bizonyos időintervallumban bekövetkező független események előfordulásának számát modellezi.



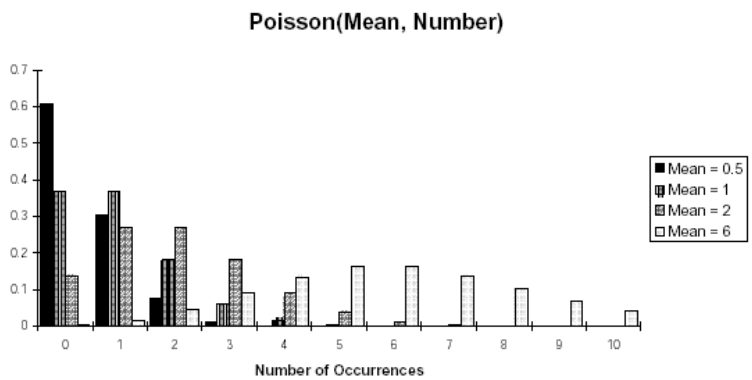
17.3. ábra. Poisson eloszlás.

Például egy csomagfolyamból egy cél által egy másodperc vagy perc alatt megkapott csomagok számát. A modellező eszközökben a Poisson-eloszlás egy pozitív valós számmal, a várható értékkel adható meg. A „szám" paraméter az 17.3. ábrán azt adja meg, hogy melyik véletlen számsorozat lesz alkalmazva a mintakészítéshez. Amikor ez az eloszlás egy időintervallummal együtt van megadva és egy egész számot ad, olyankor gyakran használják az adott intervallumban bekövetkező beérkezések számának leírására. Szimulációkor viszont sokkal hasznosabb, ha ez az információ a beérkezések közötti időközként van kifejezve. Erre a célra az exponenciális eloszlást használják.

- *Exponenciális eloszlás.* Ez az eloszlás a független események között eltelt időt modellezi, mint például a csomagfolyam egy forrása által küldött csomagok esetén a beérkezési időközt. Fontos megjegyezni, hogy ha az események bekövetkezése közötti idő exponenciális, akkor a bekövetkezett események száma Poisson-eloszlású. A modellező eszközökben az exponenciális eloszlás (lásd a 17.4. ábrát) egy pozitív valós számmal, az eloszlás várható értékével, és egy x paraméterrel adható meg, ami pedig azt mondja meg, melyik véletlen számsorozatot alkalmazzuk a mintakészítéshez. Az eloszlás további alkalmazási területei még például: adatbázis tranzakciók, billentyű leütések, fájl hozzáférések, elektronikus levelek, névfeloldási kérések, HTTP lekérdezések, X-window protokoll üzenetváltások stb. közötti idők modellezése.
- *Egyenletes eloszlás.* Az egyenletes eloszlás (lásd a 17.5. ábrát) olyan adatokat modellez, melyek egy intervallumból vesznek fel értékeket, mindegyiket egyenlő valószínűséggel. Az eloszlás teljesen meghatározható a legkisebb és a legnagyobb lehetséges érték megadásával. Diszkrét ada-



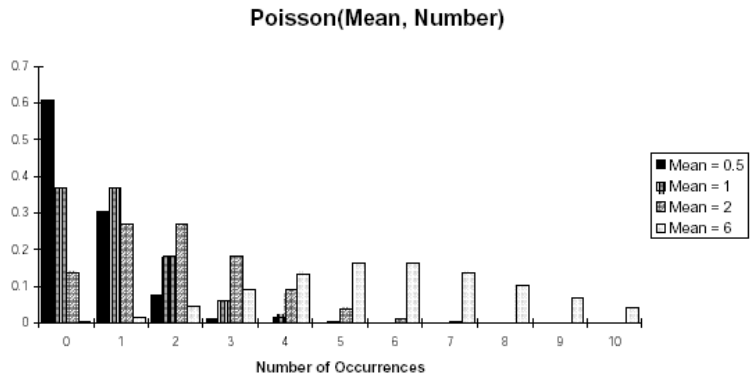
17.4. ábra. Poisson eloszlás.



17.5. ábra. Poisson eloszlás.

tok esetén a megfelelő diszkrét egyenletes eloszlás is használható. A csomaghosszakat gyakran modellezik egyenletes eloszlással. A modellező eszközökben ez az eloszlás három pozitív valós számmal adható meg. Az első kettő a legkisebb és a legnagyobb lehetséges értéket adja meg, a harmadik pedig azt, hogy melyik véletlen számsorozat legyen alkalmazva a mintakészítéshez.

- *Pareto-eloszlás.* A Pareto-eloszlás (lásd a 17.6. ábrát) egy hatvány típusú eloszlás erősen ingadozó tulajdonsággal rendelkező források modellezésére (nem hosszú távon függő forgalomra). Az eloszlás erősen csúcsos, de a vége lassan csökken. Megadása a következő három paraméterrel történik: hely, alak és eltolás. A hely megadja ahol az eloszlás kezdődik, az alak meghatározza, hogy a vége milyen gyorsan ereszkedik, az eltolás pedig



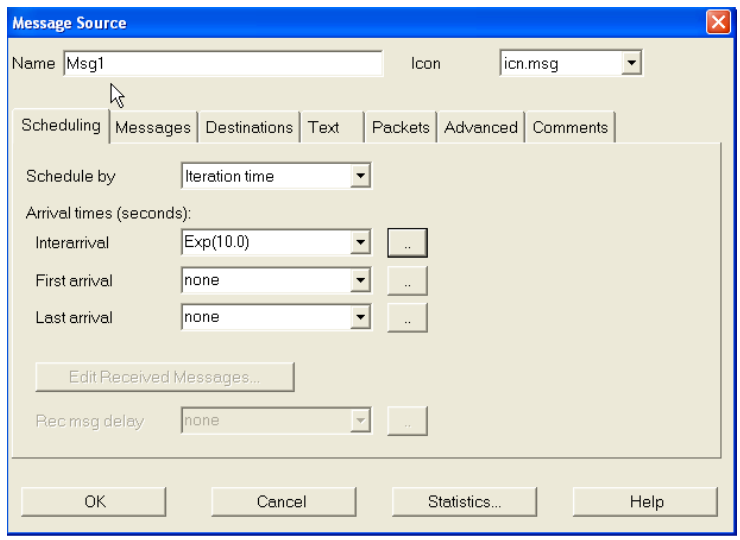
17.6. ábra. Poisson eloszlás.

eltolja az eloszlást.

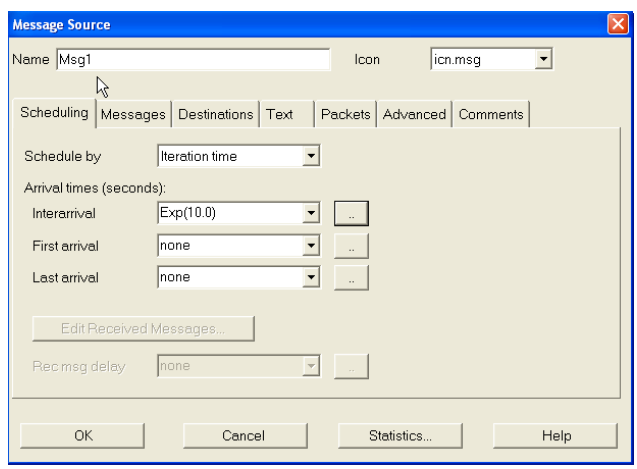
A valószínűségeloszlások egy közös alkalmazási célja az, hogy definiálják a hálózatok különböző paramétereit. A hálózatok egy tipikus modellezési célú paramétere több üzenet esetén az üzenetek közötti idő. A megadott időn egy üzenet indulásától a következő üzenet indulásáig eltelt időt értjük. Ahogyan azt korábban tárgyaltuk, a beérkezési időközökre leggyakrabban használt eloszlás az exponenciális. Az exponenciális eloszlás számára megadandó paraméterek a várható érték és a véletlen folyamszám. A hálózati forgalmat gyakran írják le Poisson folyamatként. Ez általában azt jelenti, hogy az üzenetek számát figyelték meg egymást követő időintervallumokban, és a megfigyelések száma egy intervallumban Poisson-eloszlású. A modellező eszközökben nem az időegységenkénti üzenetek számát nem adják meg, hanem inkább az üzenetek beérkezési időközét. Bebizonyítható, hogy ha az egységnyi időintervallumonkénti üzenetszám Poisson-eloszlású, akkor a beérkezési időköz exponenciális eloszlású. A 17.7. ábrán látható párbeszédablakban a beérkezési időköz eloszlás COMNET-ben az Exp(10.0) kifejezéssel van definiálva. Ez azt jelenti, hogy az egy üzenet és az utána következő üzenet indulása közötti idő exponenciális eloszlást követ 10 másodperces átlaggal. A 17.8. ábra a megfelelő sűrűségfüggvényt szemlélteti.

Több szimulációs modell is a különböző forgalomfolyamok szimulációjára összpontosít. A forgalomfolyamok szimulálhatók egyrészt a forgalom feltételezett jellemzőinek megadásával, vagy pedig a tanulmányozott alkalmazás működése során nyert valódi adatok felhasználásával megalapozva. Ez utóbbit a következő alfejezetben fogjuk tárgyalni.

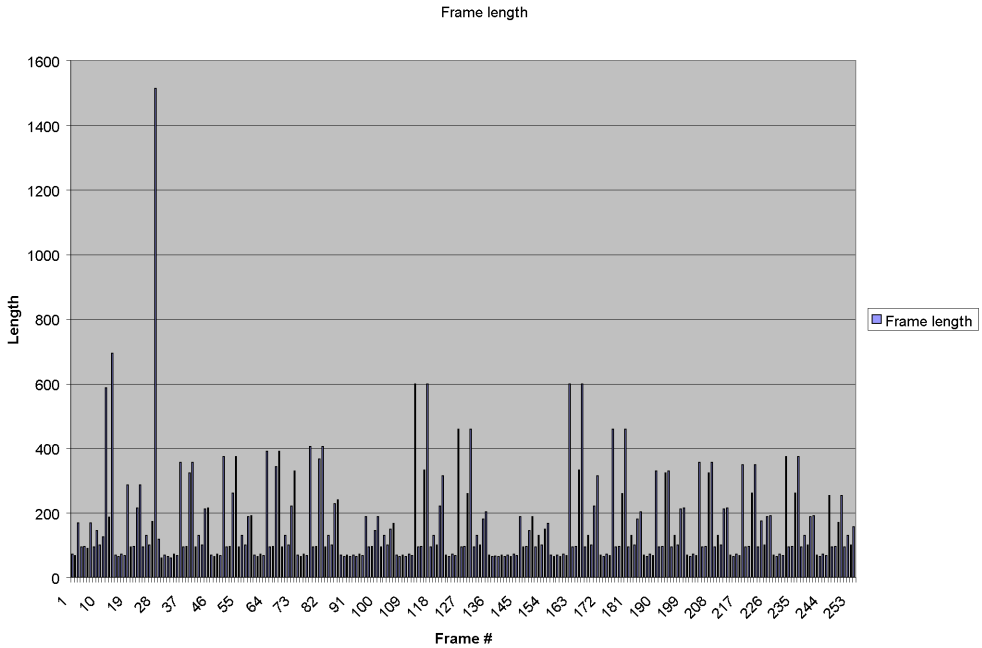
A hálózatmodellezők általában a hálózatról meglévő adatok elemzésével kezdik a modellezést, hogy ezáltal képet kapjanak a hálózat jellemzőiről. Ez



17.7. ábra. Exponenciális eloszlás átlagosan 10 másodperces beérkezési időközzel.



17.8. ábra. Az Exp(10.0) beérkezési időköz sűrűségfüggvénye.



17.9. ábra. .

segíti az alkalmazási szinten lévő folyamatok elég mély megértését ahhoz, hogy az egyes hálózatelemeket a modellezési konstrukciókhoz tudják rendelni. Különféle eszközök is használhatók a modellépítés előtt. Az előzetes elemzés után a modellező figyelmen kívül hagyhatja az olyan folyamatokat és eseményeket, amelyek nem fontosak a szóban forgó tanulmányban. Például, az adatbázis tranzakciókról nyert adatok a kerethossz nagy változékonyságát mutatják. Az 17.9. ábra segít elképzelni ezeket anomáliákat.

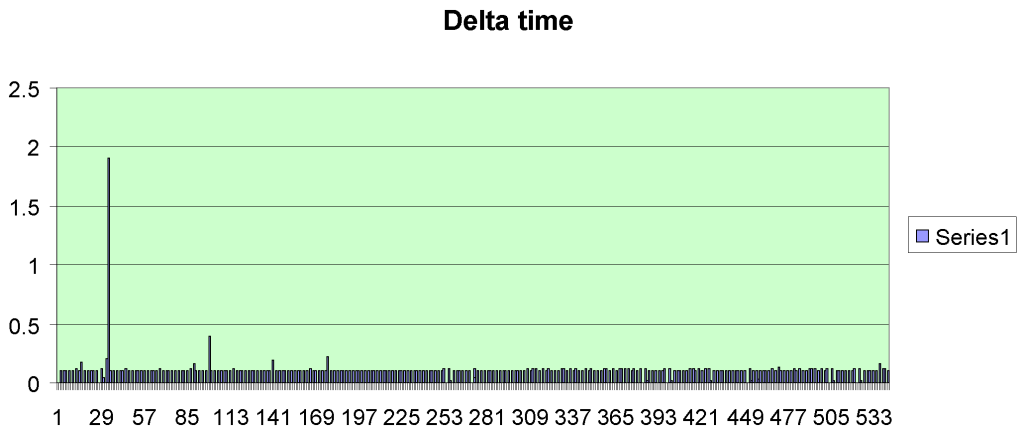
Ugyanezeknek az adatoknak a vizsgálata (17.10. ábra) a keretek beérkezési időközeinek nagy változékonyságát is felfedi.

A kumulatív valószínűségeloszlás-függvény közelítésével például a kerethosszak hisztogramja (17.11. ábra) segíti a modellezőt az eloszlástípus meghatározásában.

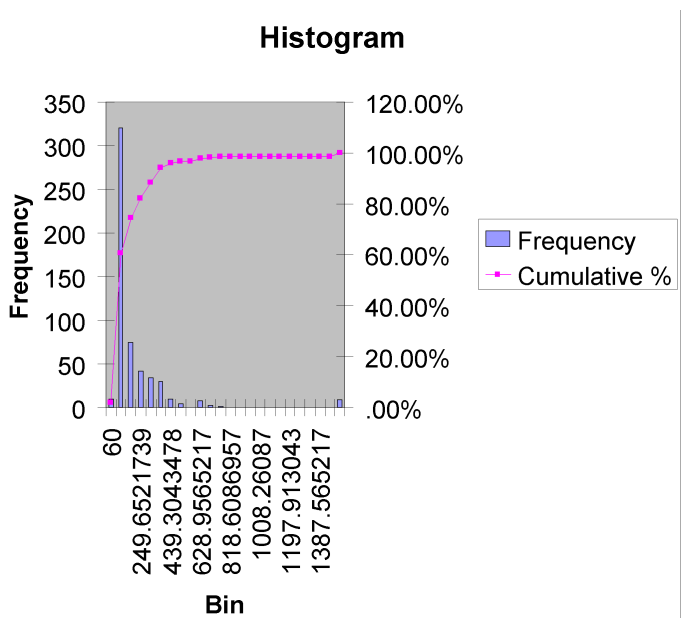
17.6. Szimulációs modellező rendszerek

17.6.1. Adatgyűjtő eszközök és hálózatelemzők

Ebben az alfejezet összefoglaljuk a széleskörűen alkalmazott OPNET és COMNET diszkrét-esemény szimulációs eszközök fő jellemzőit, és az ezeket támo-



17.10. ábra. A beérkezési időközök nagy változékonysága.



17.11. ábra. A kerethosszak hisztogramja.

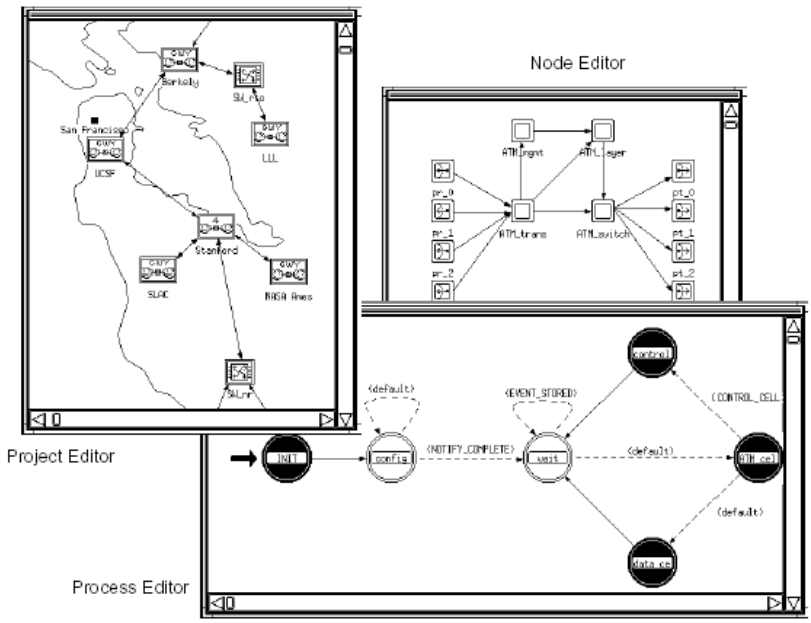
gató hálózatelemzőket, a Network Associates Sniffer-ét és a az OPNET Alkalmazásjellemző Környezetét.

Az OPNET (**OP**timized **N**etwork **E**ngineering **T**ools – *Optimalizált hálózattervező eszközök*) egy széleskörűen alkalmazható szimulációs rendszer, mely alkalmas kommunikációs hálózatok és elosztott rendszerek modellezésére részletes protokoll és teljesítményelemzéssel. Az OPNET számos eszközt tartalmaz, melyek a modellezési és szimulációs projektek egyes szakaszainak megfelelően három kategóriába sorolhatók. Ezek a szakaszok a következők: modellspecifikáció, adatgyűjtés és szimuláció, valamint az elemzés.

17.6.2. Modellspecifikáció

A modellspecifikáció során a modellező a tanulmányozott hálózati rendszer egy reprezentációját hozza létre. Az OPNET támogatja a modellek újrafelhasználását, azaz a modellek alapozhatók olyan beágyazott modellekre, melyeket korábban készítettek el és modell-könyvtárakban tároltak. A specifikációszerkesztőkkel a modellek különböző szintű részletezettséggel adhatók meg. Ezek a szerkesztők az aktuális hálózati rendszer hierarchikus struktúrájának megfelelően osztályokba sorolják az modellezési információkat. A legmagasabb szintű szerkesztő, a *Projektszerkesztő* alakítja ki a hálózati topológiák, alhálózatok, és kapcsolatok modelljeiből álló hálózatmodelleket, melyeket pedig a *Csomópontszerkesztővel* adhatunk meg. A Csomópontszerkesztő írja le a csomópontok belső szerkezetét, funkcionális elemeiket és a közöttük lévő adatfolyamokat. A csomópontok olyan folyamatmodell-modulokból állnak, melyeket a *Folyamatszerkesztővel* adhatunk meg. A hálózat hierarchiájának legalacsonyabb szintjén a folyamatmodellek írják le az adott modul viselkedését a protokollokkal, algoritmusokkal és alkalmazásokkal kapcsolatban, véges automatákat és egy magasszintű nyelvet felhasználva.

Számos más szerkesztő is elérhető a folyamat- vagy csomópont-szintű modellek által hivatkozott különböző adatmodellek definiálására, mint amilyenek például a csomagformátumok és a folyamatok közötti vezérlőinformációk. További szerkesztőkkel készíthetünk, módosíthatunk vagy csak megtekinthetünk különböző sűrűségfüggvényeket, melyekkel különböző eseményeket irányíthatunk. Ilyen például a csomagok küldése vagy fogadása között eltelt idő meghatározása. A modellspecifikáció-szerkesztők egy grafikus felületet biztosítanak a felhasználónak, mellyel változtathatja a modelleket ábrázoló objektumokat és a megfelelő folyamatokat. Mindegyik szerkesztő a modell egy adott absztrakciós szintjének megfelelő objektumokat és műveleteket adhatja meg. Ezért a Projektszerkesztő adja meg a hálózat csomópontjait és kapcsolatait, a Csomópontszerkesztő a processzorokat, sorokat és a hálózat csomópontjaiban lévő adó és fogadó egységeket, a Folya-



17.12. ábra. A három absztrakciós szintet a Projekt-, Csomópont- és Folyamatszerkesztők adják meg..

matszerkesztő pedig a folyamatok állapotait és átmeneteit. A 17.12. ábra az egyes szerkesztők absztrakciós szintjeit szemlélteti.

17.6.3. Adatgyűjtés és szimuláció

Az OPNET a szimuláció alatt többféle kimenetet tud készíteni attól függően, hogy a modellező ezt hogyan definiálja. A modellezők legtöbb esetben beépített adattípusokat is használhatnak: kimenetvektorokat és skalárokat, valamint animációkat.

- A kimenetvektorok idősorozatok szimulációs adatait reprezentálják. Ezek a vektorok idő-érték párokat tartalmazó bejegyzések listájából állnak. A bejegyzések első értéke tekinthető a független, a második pedig a függő változónak.

- A skaláris statisztikák a szimuláció alatt gyűjtött statisztikákból származtatott egyedi értékek. Ilyen például az átlagos átviteli ráta, az eldobott cellák számának maximuma, az átlagos válaszidő és más egyéb statisztikák.

- Az OPNET a szimuláció alatt vagy az után megtekinthető animációkat is tud készíteni. A modellező definiálhat többféle animációt is, például csomagfolyamokat, állapotátmeneteket és statisztikákat.

17.6.4. Elemzés

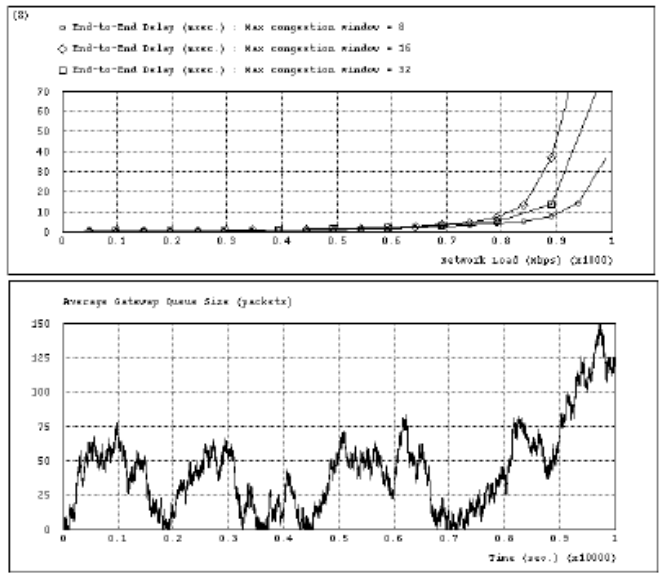
A szimuláció alatt összegyűjtött adatok többsége kimenetvektor és skalár állományokban tárolódik. Ezeknek az adatoknak az elemzésére az OPNET egy *Elemzőeszköz* nevű segédeszközt nyújt, amely ábrázoló és numerikus feldolgozó-funkciók gyűjteménye. Az Elemzőeszköz az adatokat grafikonok és nyomkövetések formájában jeleníti meg. A nyomkövetések az X és Y tengelyek értékpárjainak listáját tartalmazzák. Tárolásukra és megjelenítésükre elemzőtáblákat használ. Az Elemzőeszköz a szimulációs eredmények feldolgozására és új nyomkövetések készítésére módszerek széles választékát támogatja. Ebben benne van a hisztogramok, sűrűség és kumulatív eloszlásfüggvények, valamint a konfidencia intervallumok számítása is. Támogatja továbbá a matematikai szűrők használatát is a vektor vagy nyomkövetési adatok feldolgozásához. A matematikai szűrők előre definiált számításokra, valamint statisztikai és aritmetikai operátorokra alapozott hierarchikus blokkdiagrammokként vannak definiálva. A következő két ábra (17.13. és 17.14.) az Elemzőeszköz által készített grafikonokat szemlélteti.

A 17.14. ábrán látható Elemzőeszköz négy grafikont jelenít meg egyszerre.

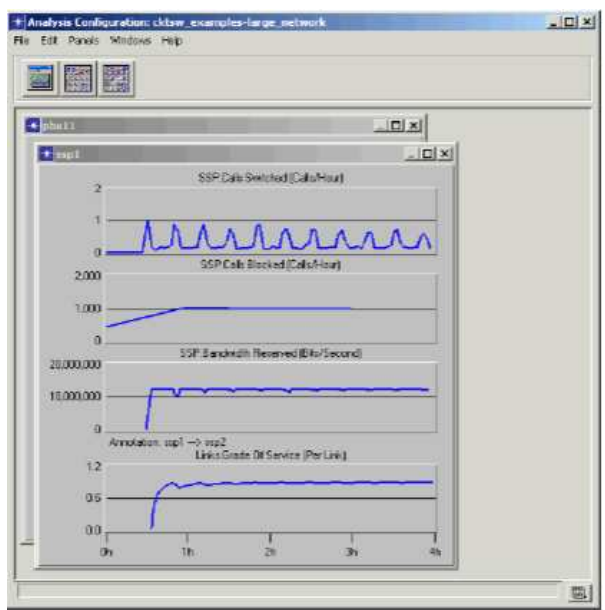
Népszerű diszkrét-esemény szimulációs rendszer a COMNET is, melyet majd az 17.9. alfejezetben fogunk röviden tárgyalni.

17.6.5. Hálózatelemzők – az Alkalmazásjellemző Környezet

Egyre nagyobb az érdeklődés az alkalmazás életciklusán keresztüli, a fejlesztéstől az üzembe helyezésig tartó teljesítmény-előrejelzésre, mérésre, modellezésre és megállapításra. Az alkalmazások teljesítményének előrejelzése különösen fontos olyan alkalmazási területeken, mint amilyen például az elektronikus kereskedelem. Az egyre inkább versengő elektronikus kereskedelemnél az alkalmazás teljesítménye különösen fontos lehet ott, ahol nagyon szoros a verseny. A teljesítmény így a bevételekre is hatással van. Ha egy alkalmazás gyengén teljesít, akkor az alkalmazás helyett inkább mindig a hálózatot hibáztatják. Ezeket a teljesítményproblémákat sokminden okozhatja, például az alkalmazás tervezési hibája vagy a lassú adatbázisszerverek. Az *Alkalmazásjellemző környezethez* („Application Characterization Environment” – ACE) és a Network Associates Sniffer-jéhez hasonló eszközök használatával a modellezők módszertanokat fejleszthetnek ki arra, hogy azonosítani tudják az alkalmazás-lelassulások forrását és megoldják az azokat előidéző problémákat.



17.13. ábra. Egy példa a skaláris adatok (felső grafikon) valamint vektor adatok (alsó grafikon) grafikus megjelenítésére..



17.14. ábra. Négy grafikont megjelenítő Elemzőeszköz..

Az alkalmazás elemzése után a modellezők javaslatokat tehetnek a teljesítmény optimalizálására, aminek eredményei gyorsabb alkalmazások és jobb válaszidők lehetnek.

Az Alkalmazásjellemző környezet a hálózati alkalmazások szemléltetésére, elemzésére és hibáinak elhárítására alkalmas eszköz. A hálózatok szervezői és alkalmazásfejlesztői a következőkre használhatják.

- Hálózatok és alkalmazások szűk keresztmetszetének meghatározására.
- Hálózati és alkalmazásproblémák felderítésére.
- Várható hálózati módosításoknak a meglévő alkalmazások válaszidejeire gyakorolt hatásának elemzésére.
- Az alkalmazásoknak változó konfigurációk és hálózati feltételek melletti teljesítményének előrejelzésére.

Egy alkalmazás teljesítményét a hálózat jellemzői határozzák meg, melyeket különböző komponensek befolyásolnak. A következő lista néhány ilyen tulajdonságot és a kapcsolódó hálózatelemeket tartalmazza:

- *Hálózati közeg*
  - Sávszélesség (torlódás, erős ingadozás)
  - Késleltetés (TCP ablakméret, nagy késleltetésű eszközök, csevegő alkalmazások)
- *Csomópontok*
- *Kliensek*
  - A felhasználók számára szükséges idő
  - Feldolgozási idő
  - Kiéheztetés
- *Kiszolgálók*
  - Feldolgozási idő
  - Többrétegű várakozó adatok
  - Kiéheztetés
- *Alkalmazások*
  - Alkalmazás-fordulók (túl sok forduló – csevegő alkalmazások)
  - Szálak (egy és többszálúság)
  - Adat profil (erősen ingadozó, túl sok adatfeldolgozás)

  Az alkalmazások elemzése két fázist igényel:

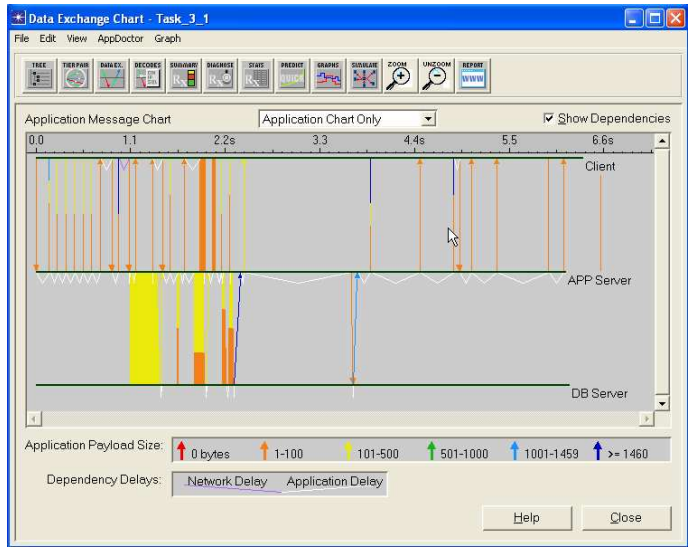
- Az alkalmazás futása közben csomag-nyomkövetések készítése ahhoz, hogy egy alapkonfigurációs modellt készíthessünk az alkalmazás modellezéséhez. Ehhez használhatjuk az ACE vagy bármely más hálózatelemzők eszközeit. Ezek a nyomkövetések stratégiailag telepített ügynök-alkalmazásokkal készíthetők el.
- A nyomkövetés-állomány importálásával az alkalmazás tranzakcióinak ábrázolása, amit alkalmazásfeladatnak nevezünk, az alkalmazás által generált üzenetek és protokoll adategységek további elemzése céljára.

Az alkalmazásfeladat elkészítése után a következő műveleteket végezhetjük el a forgalom nyomkövetési eredményein:

- A csomag-nyomkövetés eredményeinek megjelenítése és szerkesztése a protokoll-verem különböző szintjein, külön ablakokban. Ezeket az ablakokat arra is használhatjuk, hogy eltávolítsuk vagy töröljük az alkalmazásfeladat különböző részeit. Így a minket érdeklő tranzakciókra tudjuk fordítani a legnagyobb figyelmet.
- Alkalmazási szintbeli elemzés elvégzése a szűk keresztmetszetek felismerésére. Külön megmérhetjük a válaszidő egyes összetevőit, mint például az alkalmazási szinten eltöltött időt, a feldolgozási időt és a hálózati adattovábbítás idejét, továbbá részletes statisztikákat tekinthetünk meg a hálózatról és az alkalmazásról. A csomag-nyomkövetések eredményei alapján elemezhetjük a hálózati és alkalmazásprotokollok adategységeit is.
- Az alkalmazás teljesítményének előrejelzése különböző módosítási ötletek és tervezett változtatások esetén.

A következőkben a részletekbe bocsátkozás nélkül illusztráljuk egy egyszerű háromrétegű alkalmazáson keresztül az előbb említettek jellemzőit. Egy távoli alkalmazásszerverhez hozzáférő kliens (ami adatbázis szervertől igényel információkat) esetén szeretnénk meghatározni a nagy válaszidő okait. A kliens egy ADSL vonalon csatlakozik az Internetre, az alkalmazásszerver és az adatbázis szerver között pedig egy 100 Mbps sebességű Ethernet kapcsolat van. Azonosítani akarjuk a nagy válaszidő okát, valamint megoldási lehetőségeket szeretnénk ajánlani. Ehhez nyomkövetési ügynököket telepítünk a kliens és az alkalmazásszerver, valamint a két szerver közé. Az ügynökök egyidejűleg mindkét helyen gyűjtik az információkat a tranzakciók alatt. Ezután a nyomkövetési információk egyesíthetők és szinkronizálhatók, hogy ezáltal a hálózat és az egyes rétegek késleltetésének a lehető legjobb elemzési lehetőségét tudják nyújtani.

A nyomkövetési információknak az ACE-ba való importálása után a tranzakciókat az *Adatcsere diagramon* (**D**ata **E**xchange **C**hart) elemezzük, ami



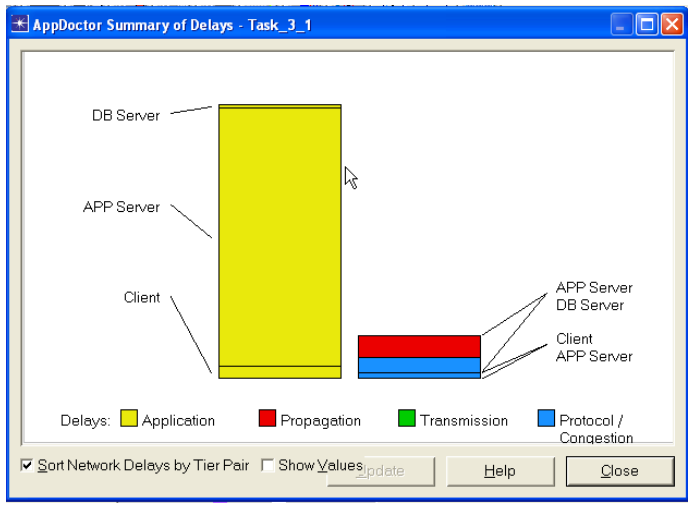
17.15. ábra. Adatcsere diagram.

ábrázolja az alkalmazás üzeneteinek a rétegek közötti továbbítását (lásd a 17.15. ábrát).

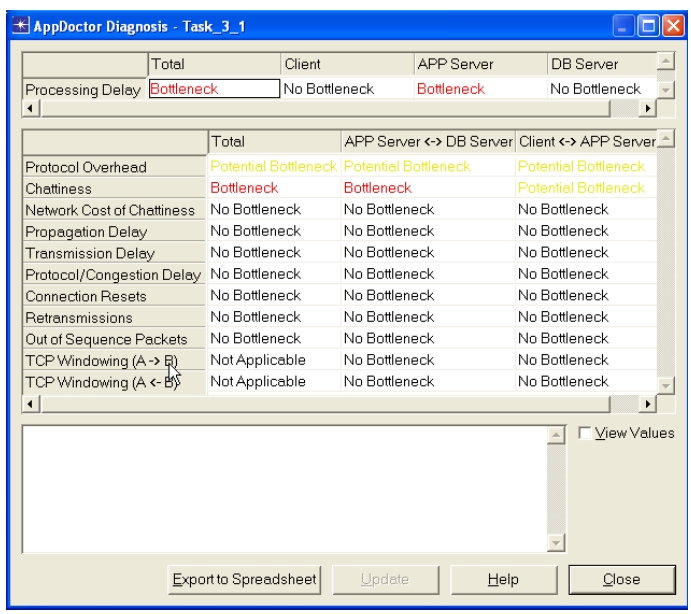
Az Adatcsere diagram megmutatja a kliens és a szerverek között átvitt különböző méretű csomagokat. A tranzakció teljes válaszideje körülbelül 6 másodperc. A „Függőségek megjelenítése” jelölőnégyzet bejelölése esetén fehér vonalak jelzik a nagy feldolgozási késleltetéseket az alkalmazásszerver és a kliens rétegeknél. További elemzés céljára készíttethetünk egy „Késleltetések összegzése” ablakot, melyben az alkalmazás válaszideje a következő négy általános kategóriára van osztva: az alkalmazás késleltetése, terjedési késleltetés, átviteli késleltetés és protokoll/torlódás miatti késleltetés (lásd az 17.16. ábrát). Ennek a diagramnak a segítségével megfigyelhetjük az alkalmazáshoz és a hálózathoz kapcsolódó késleltetések viszonyát a kliens és a szerverek közötti tranzakció alatt. Látható, hogy az alkalmazás késleltetése jóval több, mint a terjedési, átviteli és a protokoll/torlódás miatti késleltetés.

A „Diagnózis” funkció (17.17. ábra) a lehetséges szűk keresztmetszeteknek egy sokkal részletesebb elemzési lehetőségét biztosítja olyan tényezők elemzésével, melyek gyakran okoznak teljesítményproblémákat a hálózati alkalmazásokban. Az egy adott küszöb feletti értékek szűk keresztmetszetként, vagy lehetséges szűk keresztmetszetként vannak megjelölve.

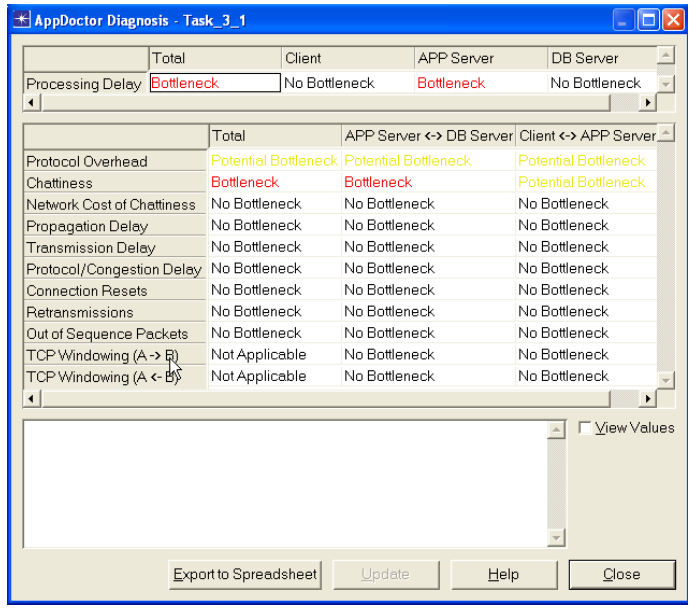
A tranzakció diagnózisa igazolja, hogy az elsődleges szűk keresztmetszet az alkalmazásszerver feldolgozási késleltetése miatt van. A feldolgozási késleltetést lassú állomány input/output, CPU feldolgozás vagy memória-



17.16. ábra. Késleltetések összegzése..



17.17. ábra. Diagnózis ablak..



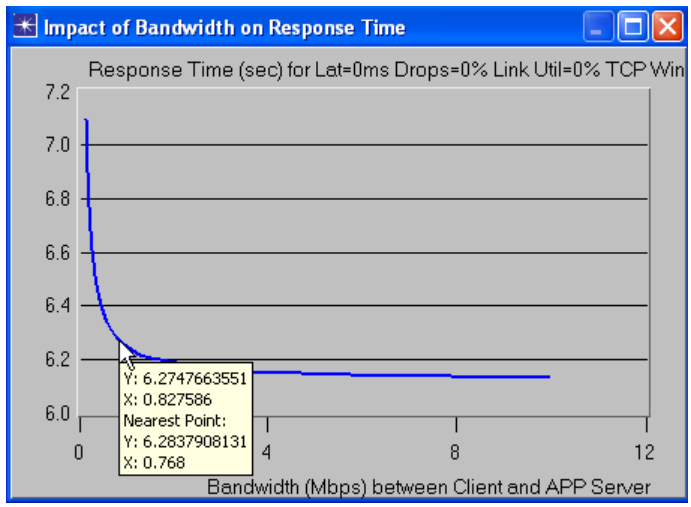
17.18. ábra. Statisztikák ablak..

hozzáférés okozhatja. Az elemzés egy másik szűk keresztmetszetet is felfed, ami az alkalmazás csevegőssége. Ez tehát a következő feladatunk. Az alkalmazás viselkedését az alkalmazás-fordulókkal kapcsolatban vizsgáljuk, amihez a tranzakció-statisztikákból juthatunk hozzá. Egy alkalmazás-forduló az alkalmazás-üzenetfolyam irányának megváltozását jelenti.

A tranzakció statisztikáiból (17.18. ábra) kiderül, hogy az alkalmazás-fordulók száma nagy, azaz a tranzakció által egy időben küldött adatok mérete kicsi. Ez jelentős alkalmazásbeli és hálózati késleltetéseket okozhat. Továbbá, az alkalmazás feldolgozási idejének jelentős része telhet el a sok kérés és válasz feldolgozásával. A Diagnózis ablak egy „Csevegősség" szűk keresztmetszetet jelez „Csevegősség hálózati költsége" szűk keresztmetszet nélkül, ami a következőket jelenti:

- Az alkalmazás nem generál jelentős hálózati forgalmat a csevegősség miatt.
- Az alkalmazás jelentős feldolgozási késleltetést okoz a sok kis alkalmazási szintbeli kérés és válasz kezelése miatt.
- Az alkalmazás „Csevegősség hálózati költsége" drasztikusan nőhet egy nagyobb késleltetésű hálózatban.

Azt lehet ajánlani, hogy az alkalmazásnak kevesebb és nagyobb üzeneteket



17.19. ábra. A sávszélesség hozzáadásának hatása a válaszidőre..

kellene küldenie. Ezzel hatékonyabban használná ki a rétegek és a hálózat erőforrásait. Például, egy adatbázis alkalmazásnak nem szabad egy rekordhalmazt rekordonként egyesével elküldenie.

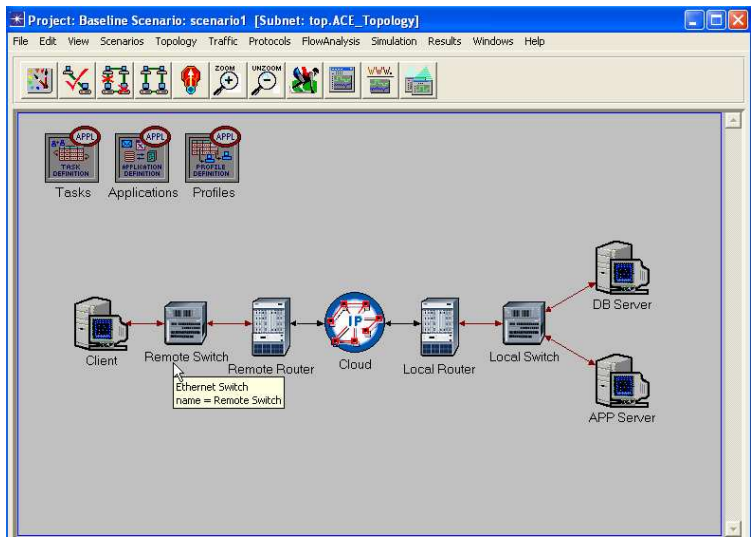
Vajon jelentősen csökkenne-e a válaszidő akkor, ha nagyobb sávszélességet adnánk (17.19. ábra) a kliens és az alkalmazásszerver közötti kapcsolathoz? Ennek a kérdésnek a megválaszolása azért fontos, mert a nagyobb sávszélesség sokba kerül. A becslés funkciót felhasználva megválaszolhatjuk ezt a kérdést. A következő diagramon a sávszélességet 128 Kbps-ról 10 Mbps-ra növeltük. Látható, hogy körülbelül 827 Kbps után nincs jelentős javulás a válaszidőben, azaz az ajánlott legnagyobb sávszélesség ennél az alkalmazásnál nem nagyobb, mint 827 Kbps, ami pedig biztosítható egy nagyobb sebességű DSL vonallal is.

Az alkalmazás teljesítményének elemzése után a forgalom nyomkövetési adataiból azonnal el tudjuk készíteni az induló alapkonfigurációs modellt további szimulációs célokra, amit a 17.20. ábra szemléltet.

17.6.6. Sniffer

Egy másik népszerű hálózatelemző a Network Associates cég *Sniffer*nevű terméke. (A Network Associates ezt nemrég nevezte át Netasyst-re.) Ez egy hatékony hálózatmegjelenítő eszköz, ami a következő feladatok megoldását teszi lehetővé:

- A hálózati forgalom részletes elemzés céljára történő nyomkövetését.



17.20. ábra. Alapkonfigurációs modell további szimulációs tanulmányokra..

- Az *Expert Analyzer* segítségével problémák diagnosztizálását.
- A hálózati tevékenységek valós idejű megfigyelését.
- Részletes kihasználtság és hibastatisztikák gyűjtését az egyes állomásokról, párbeszédekről vagy a hálózat bármely részéről.
- A korábbi kihasználtság és hibainformációk mentését alapkonfigurációs elemzésre.
- Látható és hallható riasztások létrehozását, továbbá a hálózat adminisztrátorainak értesítését problémák észlelése esetén.
- A hálózat aktív eszközökkel való vizsgálatát forgalomszimulációs célokra, válaszidő mérésre, hop-ok számlálására és problémák elhárítására.

17.7. Modellfejlesztési életciklus

A hálózatmodellezésnek számos megközelítése van. Egy lehetséges módszer egy modell készítése a hálózati topológia alapján a hálózati forgalom statisztikai közelítésével. Néhány módosítás után a modellező meg tudja vizsgálni néhány paraméter változtatásának a hálózat vagy az alkalmazás teljesítményére gyakorolt hatását. Ennél a megközelítésnél fontosabb a teljesítménykülönbségek vizsgálata annál, hogy egy olyan modellt használjunk, ami valós hálózati forgalmon alapszik. Például, bizonyos kliens-szerver tranzak-

ciókat feltételezve meg akarjuk mérni a válaszidő változását a kapcsolat-kihasználtság függvényében. Ebben az esetben nem különösebben fontos, hogy egy a valódi hálózati forgalmon alapuló modellt alkalmazzunk. Elég megadni egy gyakori felhasználó vagy tervező által becsült adatforgalom mennyiségét, majd ezen adatmennyiség esetén vizsgáljuk meg, hogy a válaszidő mennyivel nő, amikor a kapcsolat-kihasználtság növekszik az eredetihez képest.

A hálózatmodellezés leggyakrabban használt megközelítése a megelőző hálózatszervezés módszertanát követi. Ez magában foglalja a valós hálózati forgalmon alapuló hálózati modell készítését a hálózat jelenlegi és a jövőbeni viselkedésének szimulálására és az új alkalmazások hozzáadásának hatásainak előrejelzésére. A modellező és szimulációs eszközök alkalmazásával a hálózatszervezők változtatni tudnak a modellen, új berendezéseket, munkaállomásokat, kiszolgálókat és alkalmazásokat adhatnak hozzá. Ezen kívül az egyes kapcsolatokat nagyobb sebességűekre cserélhetik, és tesztelhetik őket a hálózat tényleges megváltoztatása előtt. A következőkben ezt az egyetemek, vállalatok és az ipar által széleskörűen elfogadott megközelítési módot követjük. A következő bekezdésekben a ***modellfejlesztési életciklusnak*** nevezett modellezési lépéseket tekintjük át, amelyeket már több nagyméretű vállalati hálózat modellezésében alkalmaztunk. A modellfejlesztési életciklus tehát a következő lépésekből áll:

- A hálózat topológiájának és komponenseinek azonosítása.
- Adatgyűjtés.
- Az alapkonfigurációs modell elkészítése és érvényesítése, és ennek felhasználásával szimulációs tanulmányok végzése.
- Az alkalmazásmodell elkészítése az alkalmazások által generált forgalom részleteinek felhasználásával.
- Az alapkonfigurációs és az alkalmazásmodell integrációja valamint a szimulációs tanulmányok befejezése.
- További adatgyűjtés a hálózat növekedése és változásai után, továbbá miután további részleteket tudtunk meg az alkalmazásokról.
- Ezeknek a lépéseknek a megismétlése.

A következőkben a fenti lépéseket részletezzük.

A hálózat topológiájának és komponenseinek azonosítása

A topológia a hálózat fizikai komponenseit (forgalomirányítók, áramkörök és kiszolgálók) és ezek kapcsolatát írja le. Tartalmazza az egyes hálózati berendezések helyét és konfigurációjuk leírását, azt hogy milyen típusú és sebességű áramkörökkel vannak összekapcsolva, a LAN-ok és WAN-ok típusát, a kiszol-

	RMON1	RMON2	Enterprise RMON
Ethernet/Token Ring	X	X	X
MAC réteg megfigyelése	X	X	X
hálózati réteg megfigyelése		X	X
alkalmazási réteg megfigyelése		X	X
kapcsolt LAN, Frame Relay, ATM			X
VLAN támogatása			X
alkalmazás válaszideje			X

17.21. ábra. Az RMON szabványok összehasonlítása.

gálók helyét, a címzési módokat, az alkalmazások és protokollok listáját stb.

Adatgyűjtés

Az alapkonfigurációs modell felépítéséhez szükség van a topológia és forgalom adataira. A modellezők a topológia jellemzőit megadhatják manuálisan, vagy pedig hálózatmenedzselő eszközök és a hálózati berendezések konfigurációs állományainak felhasználásával. Számos hálózatmenedzselő eszköz használja az *Egyszerű hálózati menedzsment protokollt* (**S**imple **N**etwork **M**anagement **P**rotocol – SNMP), mellyel a forgalomirányítókban és más berendezésekben futó SNMP ügynökök által karbantartott *Menedzsment információs adatbázisból* (**M**anagement **I**nformation **B**ase – MIB) lehet lekérdezni. Ezt a folyamatot nevezik SNMP hálózat-feltérképezésnek. A kérdéses hálózat topológiájának felméréséhez topológia-adatokat importálhatunk a forgalomirányítók konfigurációs állományaiból. Egyes teljesítménymenedzselő eszközök hálózatmenedzsment platformok (például HP OpenView vagy IBM NetView) térkép állományaiból is tudnak adatokat importálni. Ezek exportálási funkcióját alkalmazva a kapott térkép fájl importálható lesz a modellezéshez.

Az alapkonfigurációs modellhez szükséges forgalmi adatok különböző forrásokból származhatnak: interjúkból és hálózati dokumentumokból származó forgalomleírásokból, tervezési vagy karbantartási dokumentumokból, MIB/SNMP jelentésekből és hálózatelemző vagy *Távoli megfigyelő* (**R**emote **M**onitoring – RMON) nyomkövetési eredményekből. Az RMON egy hálózatmenedzselő protokoll, ami lehetővé teszi a hálózati információk gyűjtését az egyes csomópontokban. Az RMON nyomkövetési eredményeit RMON szondák gyűjtik, melyek szabványuktól függően a hálózati architektúra különböző szintjein gyűjtik az adatokat. A 17.21. ábra a leggyakrabban használt szabványokat és adatgyűjtési szinteket foglalja össze.

A hálózati forgalom osztályozható használat és alkalmazás alapú adatokként. A legfontosabb különbség a két osztály között az, hogy milyen fokú

részleteket biztosítanak az adatok, valamint hogy milyen következtetések vonhatók le belőlük. A felosztás egyértelműen megadható két egymás melletti OSI réteg, a szállítási és a viszonyréteg segítségével. A használat alapú adatok a szállítási réteggel kapcsolatos teljesítménykérdésekkel kapcsolatos vizsgálatokra szolgálnak, az alkalmazás alapú adatok pedig a hálózati architektúra szállítási rétege fölötti rétegekkel kapcsolatos elemzésekre. (Intenet terminológiában ez ekvivalens a TCP szint és az e fölötti alkalmazási szint közötti vágással.)

A használat alapú adatok gyűjtésének célja a teljes forgalom méretének meghatározása az alkalmazásoknak a hálózaton történő megvalósítása előtt. Ezek az adatok a forgalomirányítókban vagy egyéb hálózati berendezésekben lévő SNMP ügynököktől gyűjthetők össze. A forgalomirányítókhoz vagy kapcsolókhoz küldött SNMP kérdések statisztikákat biztosítanak az egyes LAN interfészeken, WAN áramkörökön vagy permanens virtuális áramkör (**P**ermanent **V**irtual **C**ircuit – PVC) interfészeken keresztül küldött bájtok pontos számáról. Ezen adatok segítségével kiszámíthatjuk az egyes áramkörökön elérhető sávszélesség kihasználtságát.

Az alkalmazás alapú adatok gyűjtésének célja pedig az egy alkalmazás által generált adatok mennyiségének és az alkalmazás igényeinek a típusának meghatározása. Ez lehetővé teszi a modellezők számára az alkalmazás viselkedésének megértését és az alkalmazási szintbeli forgalom jellemzését. A forgalomelemzőkből, RMON2 kompatibilis szondákból, a Sniffer-ből vagy a NETScout Manager-ből származó adatok különböző részleteket biztosítanak az alkalmazás hálózati forgalmáról. A stratégiailag elhelyezett adatgyűjtő berendezések elég adatot tudnak gyűjteni ahhoz, hogy világosan lássuk az alkalmazás forgalmának viselkedését. A forgalomelemzők tipikusan a következő alkalmazási szintbeli adatokat gyűjtik:

- Az alkalmazások típusait.
- A hálózati rétegbeli címmel (azaz IP címmel) rendelkező hosztokat.
- Két hoszt közötti hálózati párbeszédek hosszát (a kezdés és a befejezés időpontja).
- Az egyes párbeszédek során mindkét irányban az elküldött bájtok számát.
- A párbeszédek során mindkét irányban a csomagok átlagos méretét.
- A forgalom erős ingadozását.
- Csomaghossz eloszlásokat.
- Csomag-beérkezési időköz eloszlásokat.
- Csomagtovábbítási protokollokat.
- Forgalom profilt, azaz üzenet és csomag méreteket, beérkezési időközöket

és feldolgozási késleltetést.

- Egy alkalmazásnak egy tipikus felhasználó általi használatának gyakoriságát.
- A résztvevő csomópontok fontosabb kölcsönhatásait, események sorozatait.

Az alapkonfigurációs modell elkészítése és érvényesítése, és ennek felhasználásával szimulációs tanulmányok végzése

Az alapkonfigurációs modell készítésének célja az, hogy a vizsgált hálózat pontos modelljét adjuk meg. Ez a modell a hálózat jelenlegi állapotát tükrözi. A tanulmányok pedig az ezen a modellen elvégzett módosítások hatásait mérik fel. A modell könnyen érvényesíthető, ugyanis az előrejelzéseinek összhangban kell lenniük az aktuális hálózaton végzett mérésekkel. Az alapkonfigurációs modell általában csak olyan alapvető teljesítmény-mértékeket jelez elő, mint az erőforrás kihasználtság és a válaszidő.

Az alapkonfigurációs modell a topológiának és a korábban összegyűjtött használat alapú forgalomadatoknak az egyesítésével jön létre. Ezt érvényesíteni kell az aktuális hálózat teljesítmény-paramétereivel, azaz igazolni kell, hogy a modell a valós hálózati működéshez hasonlóan működik. Az alapkonfigurációs modell használható az aktuális hálózat elemzésére, és szolgálhat a további alkalmazás és kapacitástervek alapjául is. A modellezőeszközök importálási funkcióját alkalmazva a modellkészítés kezdhető az életciklus adatgyűjtési lépésében összegyűjtött topológiaadatok importálásával. A különböző hálózatmenedzselő rendszerek, mint például a HP OpenView vagy a Network Associate Sniffer-je, a topológia adatait általában topológiaállományokban (.top vagy .csv) tárolják. A forgalomadatokat tartalmazó állományok a következőképpen csoportosíthatók:

- Párbeszédek résztvevőinek kommunikációjával kapcsolatos állományok, melyek összegyűjtve tartalmazzák a hálózat terheltségének információit, a hosztneveket, a küldött csomagok és bájtok számát az egyes hosztpárok esetén. Az adatok lehetővé teszik a modellezőeszköz számára, hogy megtartsa a forgalom erősen ingadozó voltát. Ezek az állományok különböző adatgyűjtő eszközök segítségével nyerhetők.
- Esemény-nyomkövetési állományok, melyek összegzett információk helyett az egyes párbeszédekhez kapcsolódó hálózatterheltségéről tartalmaznak információkat. A szimuláció során az állomány alapján eseményről eseményre lejátszhatók a hálózati tevékenységek.

A szimuláció előtt a modellezőnek a következő szimulációs paraméterekkel

kapcsolatban kell döntenie:

- *Futási idő.* A ***futási időnek*** nagyobbnak kell lennie, mint a leghosszabb üzenetnek a hálózatban való késése. Ezen idő alatt a szimulációnak elegendő számú eseményt kell létrehoznia, hogy a modell elég mintát tudjon generálni minden eseményből.
- *Felmelegedési periódus.* A szimuláció ***felmelegedési periódusa*** az az idő, ami a csomagok, pufferek, üzenetsorok, áramkörök és a modell különböző egyéb elemeinek inicializálásához kell. A felmelegedési periódus megegyezik egy tipikus üzenet hosztok közötti késleltetésével. A szimuláció felmelegedési ideje azért szükséges, hogy biztosítsuk az egyensúlyi állapot elérését az adatgyűjtés megkezdése előtt.
- *Többszöri ismétlések.* Olyan esetekben, amikor a statisztikák nem eléggé közelítik a valós értékeket, szükség lehet egy adott modell többszöri lefuttatására is. Szintén többszöri ismétlésekre van szükség az érvényesítés előtt, amikor több másolatot futtatunk le azért, hogy meghatározzuk a statisztikáknak a másolatok közötti ingadozását. Ennek okai leggyakrabban a ritka események.
- *Konfidenciaintervallum.* A konfidenciaintervallum segítségével becsüljük meg a populáció-paraméter valószínűsíthető méretét. Ez megad egy becsült értéktartományt, ami egy adott valószínűséggel tartalmazza a becsült paramétert. A leggyakrabban használt intervallumok a 95% és 99% konfidenciaintervallumok, melyek 0.95 és 0.99 valószínűséggel tartalmazzák az adott paramétert. A szimulációban a konfidenciaintervallum egy mutatót biztosít a szimulációs eredmények pontosságára nézve. A kevesebb ismétlés szélesebb konfidenciaintervallumot és kisebb pontosságot eredményez.

Több modellezőeszközben a topológia és forgalomadatokat tartalmazó állományok importálása után az alapkonfigurációs modell automatikusan elkészül. Ezt ellenőrizni kell a konstrukciós hibák elkerülése végett, majd érvényesíteni a következő lépések végrehajtásával:

- Egy előzetes futtatással ellenőrizni kell, hogy minden forrás-cél pár jelen van-e a modellben.
- Egy felmelegedési periódust is tartalmazó hosszabb szimulációval meg kell mérni a küldött és fogadott üzenetek számát és a kapcsolatok kihasználtságát, hogy ezzel igazoljuk a megfelelő méretű forgalommennyiség átvitelét.

Az alapkonfigurációs modell érvényesítése igazolja, hogy a szimuláció ugyanazokat a teljesítmény-paramétereket szolgáltatja, mint amiket a fizikai

hálózaton mértünk. Általában a következő hálózatjellemzők mérhetők mind a modellben, mind pedig a fizikai hálózatban.

- Küldött és fogadott csomagok száma.
- Pufferhasználat.
- Csomagkésleltetés.
- Kapcsolat-kihasználtság.
- Csomópontok CPU kihasználtsága.

A konfidenciaintervallumok és az egymástól független minták száma befolyásolja azt, hogy modell és a valós hálózat között milyen szoros egyezés várható. A legtöbb esetben a legjobb, amit várhatunk, egy átfedés a szimulációval előrejelzett értékek és a mért adatok konfidenciaintervalluma között. A nagyon szoros egyezés elérése túl sok mintát igényelne a hálózatból, és a szimuláció túl sokszori ismétlését.

Az alkalmazásmodell elkészítése az alkalmazások által generált forgalom részleteinek felhasználásával

Az alkalmazásmodelleket akkor tanulmányozzák, ha egy hálózati alkalmazásnak a hálózat teljesítményére gyakorolt hatását, vagy pedig ha a hálózatnak az alkalmazás teljesítményére gyakorolt hatását kell kiértékelni. Az alkalmazásmodellek a hálózati csomópontok között az alkalmazás futása alatt generált forgalomról szolgáltatnak részleteket. Az alkalmazásmodellek építésének lépései hasonlóak az alapkonfigurációs modellnél tárgyaltakhoz:

- Adatok gyűjtése az alkalmazás eseményeiről és a felhasználói profilokról.
- Az alkalmazás adatainak importálása a szimulációs modellbe manuálisan vagy automatikusan.
- A modellezési hibák azonosítása és javítása.
- A modell érvényesítése.

Az alapkonfigurációs és az alkalmazási modell integrációja és a szimulációs tanulmányok befejezése

Az alkalmazásmodell(ek) és az alapkonfigurációs modell integrációja a következő lépések szerint történik:

- Kezdjük a használat alapú adatokból készített alapkonfigurációs modellel.
- Az alkalmazáshasználati esetek információit felhasználva (a felhasználók helye, száma, tranzakciók gyakorisága) meghatározzuk, hogy hová és hogyan töltsük be az alkalmazásprofilokat az alapkonfigurációs modellbe.
- Az előző lépésben generált alkalmazásprofilokat az alapkonfigurációs mo-

dellhez adjuk, melyek a tanulmányozott alkalmazás által generált forgalmat ábrázolják.

A szimulációs tanulmányok befejezése a következő lépésekből áll:

- Futtassuk a modellt vagy szimulációt egy modellezési eszköz felhasználásával.
- Elemezzük az eredményeket: hasonlítsuk össze a cél tranzakciók teljesítmény paramétereit a szimuláció kezdetekor felállított célokkal.
- Elemezzük a különböző hálózatelemek kihasználtságát és teljesítményét, különösen ott ahol a célokat nem értük el.

A tipikus szimulációs tanulmányok a következő eseteket tartalmazzák:

- Kapacitáselemzés

  A kapacitás elemzésekor a hálózat paramétereinek változásait tanulmányozzuk, mint például:

  – Felhasználók számának és helyének változása.
  – Hálózatelemek kapacitásának változása.
  – Hálózati technológiák változása.

  A modellezőt érdekelhetik például a különböző változtatásoknak a következő hálózatparaméterekre gyakorolt hatásai:

  – Kapcsolók és forgalomirányítók kihasználtsága.
  – Kommunikációs kapcsolatok kihasználtsága.
  – Pufferkihasználtság.
  – Az újraküldött és az elveszett csomagok száma.

- Válaszidő elemzése

  A válaszidő elemzésének köre az üzenet és csomagtovábbítási késleltetések tanulmányozása:

  – Alkalmazási és hálózati rétegbeli csomagtovábbítási késleltetés.
  – Csomag válaszidő.
  – Üzenet/csomag késleltetések.
  – Az alkalmazás válaszideje.

- Alkalmazáselemzés

  Egy alkalmazással kapcsolatos tanulmányhoz az alkalmazás válaszidő arányának meghatározása tartozik, relatívan az egyes hálózati komponensek és alkalmazások késleltetéséhez. Az alkalmazások elemzése

statisztikákat biztosít a hálózat és az alkalmazások különböző teljesítményjellemzőiről, beleértve az előző alfejezetben tárgyaltakat is.

További adatgyűjtés a hálózat növekedése és változásai után, továbbá miután további részleteket tudtunk meg az alkalmazásokról
A következőkben bemutatott fázisnak a célja az, hogy elemezze vagy előrejelezze a hálózat teljesítményét az aktuális feltételek és a hálózat terhelésének megváltozása (új alkalmazások, felhasználók vagy hálózati struktúra) esetén:

- Azonosítsuk a hálózati infrastruktúra azon módosításait, melyek megváltoztatják a hálózati erőforrások kapacitásigényét.
- Az áttervezés tartalmazhatja a megnövekedett vagy lecsökkent kapacitást, a hálózatelemek áthelyezését vagy a megváltozott kommunikációs technológiát.
- Módosítsuk a modellt ezeknek a változásoknak megfelelően.
- Mérjük fel az alkalmazás fejlesztési vagy telepítési terveknek a hálózatra gyakorolt hatását.
- Mérjük fel az üzleti feltételek és tervek (pl. új felhasználók, telephelyek hozzáadása) hatását a hálózatra.
- Használjunk folyamatos mérési technikákat a használati tendenciák, különösen az Internet és intranet használathoz kapcsolódók figyelésére.

17.8. A forgalom ingadozásának hatása

A helyi hálózati és nagy kiterjedésű hálózati forgalommal kapcsolatos jelenlegi mérések azt bizonyítják, hogy a széleskörűen elterjedt Markov-folyamat modellek nem alkalmazhatók napjaink hálózati forgalmának leírására. Ha a forgalom Markov-folyamat lenne, akkor a forgalom erős ingadozása hosszú idő átlagolása során kiegyenlítődne, ami ellentétes a forgalom megfigyelt jellemzőivel. A valós fogalommal kapcsolatos mérések azt is bizonyítják, hogy a forgalom erős ingadozása nagyon gyakran jelen van. A gyakran vagy mindig erősen ingadozó forgalom statisztikailag jellemezhető az *önhasonlóság* fogalmának felhasználásával. Az önhasonlóságot gyakran összekapcsolják fraktálgeometriai objektumokkal, melyek a nagyításuktól függetlenül egyformának látszanak. A sztochasztikus folyamatok, idősorok esetében az önhasonlóság kifejezés a folyamat eloszlására utal, amely azonos marad, ha különböző időintervallumban nézünk. Az önhasonló idősorok jelentős ingadozásokat tartalmaznak, bármely időintervallumban kiugróan magas

értékek sokaságával. A hálózati forgalom jellemzői – mint például a csomagok száma másodpercenként, a bájtok száma másodpercenként vagy a keretek hossza – tekinthetők sztochasztikus idősoroknak. Ezért a forgalom ingadozásának meghatározása megegyezik a megfelelő idősorok önhasonlóságának jellemzésével.

A hálózati forgalom önhasonlóságával foglalkozó cikkek megmutatják, hogy a csomagvesztés, a puffer kihasználtság és a válaszidő teljesen más lesz akkor, ha a szimuláció valós forgalomadatokat vagy az önhasonlóságot is tartalmazó mesterséges adatokat használ.

A háttérről

Legyen $X = (X_t : t = 0, 1, 2, \ldots)$ egy stacionárius kovariancia folyamat. Egy ilyen folyamathoz tartozik egy $\mu = \mathrm{E}[X_t]$ várható érték konstans, egy $\sigma^2 = \mathrm{E}\left[(X_t - \mu)^2\right]$ szórásnégyzet és egy $r(k) = \mathrm{E}\left[(X_t - \mu)(X_{t+k} - \mu)\right] / \mathrm{E}\left[(X_t - \mu)^2\right]$ $(k = 0, 1, 2, \ldots)$ autokorrelációs függvény, ami csak k-tól függ. Feltételezzük, hogy X-nek van egy

$$r(k) \sim \alpha k^{-\beta}, \quad k \to \infty \tag{17.1}$$

alakú autokorrelációs függvénye, ahol $0 < \beta < 1$ és α egy pozitív konstans. Reprezentálja $X^{(m)} = (X_{(k)}^{(m)} : k = 1, 2, 3, \ m = 1, 2, 3, \ldots)$ az új idősorokat, melyeket az eredeti X sorok feletti m méretű, egymást nem fedő blokkok átlagolásával kapunk. Minden $m = 1, 2, 3, \ldots, X^{(m)}$-re $X_k^{(m)} = (X_{km-m+1} + \cdots + X_{km})/m, (k \geq 1)$. Továbbá jelölje $r^{(m)}$ az egyesített $X^{(m)}$ idősorok autokorrelációs függvényeit.

Az önhasonlóság definíciója

Az X folyamatot ***önhasonlónak*** nevezzük $H = 1 - \beta/2$ önhasonlósági paraméterrel, ha a megfelelő $X^{(m)}$ egyesített folyamatnak azonos korrelációs struktúrája van, mint X-nek, azaz $r^{(m)}(k) = r(k)$ minden $m = 1, 2, \ldots$ $(k = 1, 2, 3, \ldots)$ esetén.

Az X stacionárius kovariancia folyamatot ***aszimptotikusan önhasonlónak*** nevezzük $H = 1 - \beta/2$ önhasonlósági paraméterrel, ha minden eléggé nagy k-ra $r^{(m)}(k) \to r(k)$, ha $m \to \infty$, $0.5 \leq H \leq 1$.

A hosszú távú függőség definíciója

Egy stacionárius folyamat ***hosszú távon függő,*** ha az autokorrelációs értékek összege végtelenhez tart: $\sum_k r(k) \to \infty$. Egyébként a folyamat ***rövid távon függő.*** A definíciókból származtatható, hogy míg a rövid távon függő folyamatoknak exponenciálisan fogyó, addig a hosszú távon függő folyamatoknak hiperbolikusan fogyó autokorrelációik vannak, azaz a kapcsolódó eloszlás nehéz farkú. A nehéz farokkal rendelkező eloszlású valószínűségi változók nagy

valószínűséggel generálnak különösen nagy értékeket.

Az önhasonlóság fokát a H vagy ***Hurst-paraméter*** fejezi ki. A paraméter a folyamat autokorrelációs függvényének a fogyás sebességét mutatja. Amint $H \to 1$, az önhasonlóság és a hosszú távú függőség mértéke is nő. A hosszú távon függő önhasonló folyamatok esetén $H > 0.5$.

Forgalommodellek

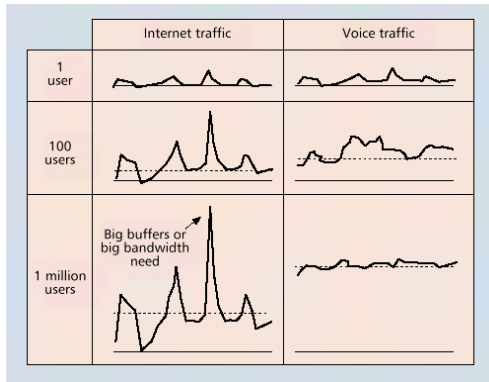
A forgalommodellezés a hagyományos hangátviteli hálózatok modellezéséből ered.

Korábban a modellek legtöbbje azon a feltevésen alapult, hogy a modellezett folyamatok markoviak (vagy általánosabban rövid távon függők). Azonban napjaink nagy sebességű digitális csomag-alapú hálózatai a hálózati szolgáltatások és technológiák különbözőségének köszönhetően sokkal bonyolultabbak és erősebben ingadozók, mint a hagyományos hangforgalom.

Válaszul az új fejlesztésekre számos kifinomult sztochasztikus modellt hoztak létre, mint például a Markov-modulált Poisson-folyamatok, a fluid-flow modellek, a markovi érkezési folyamatok, a kötegelt markovi érkezési-folyamat modellek, a csomagsorozat modellek és a *Transzformáció-kiterjesztés-minta* („Transform-Expand-Sample") modellek. Ezek a modellek leginkább a kapcsolódó sorbanállási probléma analitikus megoldására helyezik a hangsúlyt. Általában nem vetik őket össze a forgalom valós jellemzőivel, és nem bizonyítják, hogy eredményeik összhangban vannak a valós forgalomadatok statisztikai jellemzőivel.

A modellek egy másik csoportja megpróbálja követni a valós forgalomadatok statisztikai jellemzőit. Viszont hosszú ideig a hálózatokkal kapcsolatos kutatások nem rendelkeztek megfelelő forgalmi mérésekkel. Azonban az utóbbi években nagy mennyiségű – a Webre és a nagy sebességű hálózatokra vonatkozó – mérési adatot gyűjtöttek össze és tettek elérhetővé. Ezeknek az adatoknak egy része nagy felbontású, órákon, napokon vagy heteken keresztül mért forgalmi adatokat tartalmaz. Más részük hetekről, hónapokról, évekről biztosít információkat. A nagy felbontású adatok statisztikai elemzései bebizonyították, hogy a csomag-alapú hálózatok valós forgalmi adatai alátámasztják az önhasonlóságot. Ezek az eredmények rámutatnak a tradicionális modellek és a mért forgalmi adatok közötti eltérésekre. Míg a hagyományos modellekben a feltételezett folyamatok rövid távon függők, a mért forgalmi adatok bizonyítják a hosszú távú függőséget. A 17.22. ábrán az Internet-forgalom és a hangforgalom közötti különbség figyelhető meg különböző felhasználószám esetén. Amint a hangfolyamok száma nő, a forgalom egyre egyenletesebb lesz, ellentétben az Internet-forgalommal.

A rövid távon függő sorbanállási modellekkel ellentétben a hosszú távú függőséggel rendelkező modellekkel kapcsolatban eddig jóval kevesebb elméleti



17.22. ábra. Az internetes hálózati forgalom önhasonló természete.

eredmény született.

Az önhasonló modelleknek két fő csoportja van: a fraktális Gauss-zajok és a fraktális ARIMA folyamatok. A Gauss-modellek pontosan leírják több forgalomfolyam egyesítését. Az M/Pareto modellt olyan hálózati forgalommodellezésben használták, ahol a Gauss-modell alkalmazásához nem állt rendelkezésre elegendő számú egyesített adat.

Fekete doboz és strukturális modellek

A hagyományos idősorok elemzését ***fekete doboz modellezésnek*** nevezzük. Ezzel ellentétben a szerkezeti modellezés arra a környezetre összpontosít, amelyben a modell adatait gyűjtötték, azaz a hálózati komponensek hierarchiájára, melyekből napjaink kommunikációs rendszerei felépülnek. Míg az előbbi szerzők elismerik, hogy a fekete doboz modellek hasznosak lehetnek más környezetekben, a modern csomag-alapú hálózatok dinamikus és bonyolult természetének megértésére alkalmatlannak tartják őket. Ezeknek a modelleknek nem sok haszna van napjaink hálózatainak tervezésében, irányításában és ellenőrzésében sem. Hogy az empirikusan megfigyelt jelenségeknek, mint amilyen a hosszú távú függőség is, fizikai magyarázatát adjuk, a fekete doboz modelleket strukturális modellekre kell cserélnünk. A strukturális forgalommodelleknek egy jól alkalmazható jellemzőjük, hogy figyelembe veszik napjaink hálózatainak rétegelt felépítését, és elemezni tudják a hozzájuk kapcsolódó hálózatparamétereket, amelyek alapvetően meghatározzák a hálózat teljesítményét és működését. Az idősor-modellek általában ezeket a részleteket fekete dobozokként kezelik. Mivel a valós hálózatok bonyolult rendszerek, a fekete doboz modellek sok esetben számos paraméterről feltételezik, hogy pontosan ábrázolják a valós rendszert. A

hálózattervezők számára, akik igen fontos alkalmazói a forgalommodellezésnek, a fekete doboz modellek nem túl hasznosak. Egy bonyolult hálózati környezetben ritkán lehetséges megmérni vagy megbecsülni a modell nagy számú paraméterét. A hálózattervezők számára a modellnek egyszerűnek és értelmesnek kell lennie az adott hálózatra nézve. A modell támaszkodhat valós hálózati mérésekre, és az eredményeknek helytállóknak kell lenniük a valós hálózat teljesítményére és működésére nézve.

Sokáig a forgalommodelleket valódi hálózatokban gyűjtött adatok felhasználása nélkül készítették. Ezek a modellek nem voltak alkalmazhatók gyakorlati hálózattervezésre. Napjainkban a nagy mennyiségű hálózati forgalmi mérések elérhetősége, és a hálózati struktúra növekvő bonyolultsága miatt egyre inkább alkalmazzák az ***Ockham borotvája*** nevű elvet. (Ockham borotvája egy középkori filozófus, William Ockham elve volt. Eszerint az elv szerint a modellezőknek nem szabad a minimálisan szükségesnél több feltételezést tenniük. Ez az alapelv, melyet a takarékosság elvének is hívnak, motiválja az összes tudományos modellezést és elméletépítést. A modellezőknek az adott jelenség leírására alkalmas ekvivalens modellek közül a legegyszerűbbet kell választaniuk. Ez az elv bármely modell esetén segíti a modellezőket abban, hogy csak azokat a változókat vegyék be a modellbe, amelyek valóban szükségesek a jelenség magyarázatához. Így a modellek kifejlesztése egyszerűbbé válik, csökken az inkonzisztencia, a félreérthetőség és a redundancia lehetősége.)

A szerkezeti modellek bemutatják, hogyan magyarázza a forgalom dinamikájának részleteit az egyes hosztok szintjén a hosztok közötti párbeszédek hálózati forgalomának önhasonló természete. A szerkezeti forgalommodelleknek fizikai jelentése van az adott hálózati környezetben, és nyomatékosítják a hosszú távú függőség túlsúlyát az egyes hosztok közötti párbeszédek által generált csomag érkezési mintákban. Ezek a modellek betekintést nyújtanak abba, hogy az egyes hálózati kapcsolatok hogyan viselkednek helyi és nagy kiterjedésű hálózatokban. Habár ezek a modellek az összeállított forgalmi minták fizikai struktúrájának figyelembevételével túlmennek a fekete doboz modellezési módszertanon, nem tartalmazzák a kapcsolatok, forgalomirányítók, kapcsolók és ezek véges kapacitásainak a forgalmi útvonalakon egybefonódó struktúráját.

Crovella és Bestavros megmutatták, hogy a World Wide Web forgalma az önhasonlóságával egyező tulajdonságokat mutat, továbbá hogy a Weben elérhető állományméretek eloszlása miatt az átviteli idők nehéz farkúak lehetnek. Szintén megmutatták, hogy elsődlegesen a felhasználók „gondolkodási idejének” hatása miatt a csendes idők szintén nehéz farkúak lehetnek.

Az erős ingadozás hatása a nagy sebességű hálózatokra
Az erős ingadozásnak a hálózati torlódásokra gyakorolt hatásai a következők:

- A veszteségeket is tartalmazó torlódási periódusok meglehetősen hosszúak lehetnek, és erősen koncentráltak.
- A pufferméret lineáris növekedése nem okozza a csomagvesztés jelentős csökkenését.
- Az aktív kapcsolatok számának csekély növekedése is okozhat jelentős csomagvesztés-növekedést.

Az eredmények azt mutatják, hogy a csomagforgalom hullám jellegű. A hegyes csúcsok okozzák a tényleges adatvesztéseket, a kisebb hullámzások pedig a kiemelkedéseken lovagolnak tovább.

Egy másik terület, ahol az erős ingadozás befolyásolhatja a hálózat teljesítményét, az olyan ütemezésű kapcsolat, amely forgalomosztályok közötti prioritásokat tartalmaz. Egy olyan környezetben, ahol a nagyobb prioritású osztályra nincs sávszélességkorlátozás (a fizikai sávszélességen kívül), az interaktív forgalom prioritást kaphat a nagy mennyiségű adatforgalommal szemben. Ha a magasabb prioritású osztály hosszú időn keresztül erősen ingadozik, akkor ezen osztály erős ingadozásai hosszú időre akadályozhatják az alacsonyabb prioritású forgalmat.

Az erős ingadozás olyan hálózatokra is hatással lehet, ahol az engedélyezésvezérlési mechanizmus nem az egyes kapcsolatok forgalom-paraméterein, hanem a közelmúlt forgalmi mérésein alapszik. Az az engedélyezésvezérlés, amely csak a közelmúlt forgalmát veszi figyelembe, félrevezethető egy hosszú, egészen alacsony forgalmi intenzitású periódust követően.

17.8.1. Modellparaméterek

A kliensek és szerverek közötti tranzakciók aktív és azt követő inaktív periódusokból állnak. Ezek a tranzakciók az egyes irányokban küldött csomagok csoportjaiból tevődnek össze. A csomagcsoportokat erős ingadozásoknak nevezzük. A forgalom erősen ingadozó volta a következő időparaméterekkel jellemezhető:

- ***Tranzakció érkezési időköz*** (**T**ransaction **I**nter**A**rrival **T**ime – TIAT): Egy adott tranzakció első csomagja és a következő tranzakció első csomagja között eltelt idő.
- ***Erős ingadozás érkezési időköz*** (**B**urst **I**nterarrival **T**ime): az erős ingadozások között eltelt idő, $1/\lambda$, ahol λ az erős ingadozások beérkezési intenzitása.

- ***Csomag érkezési időköz*** (**P**acket **I**nterarrival **T**ime): az erős ingadozásokban a csomagok érkezése között eltelt idő, $1/r$, ahol r a csomagok beérkezési intenzitása.

A Hurst-paraméter

Előre látható, hogy az egyre több és többféle forgalom gyorsan folytatódó többszolgáltatású hálózatokba való egyesítése végül a forgalom kiegyenlítődését fogja eredményezni. Ha elegendő az egyesítés mértéke, a folyamat Gauss-folyamatokkal modellezhető. Jelenleg viszont a hálózati forgalom nem mutat a gaussihoz közeli jellemzőket. A hálózatok nagy részében az egyesítés mértéke nem elég nagy ahhoz, hogy kiegyenlítse az erősen ingadozó forgalom negatív hatását. Azonban addig, amíg a forgalom gaussivá nem válik, a létező módszerek pontos méréseket és előrejelzéseket nyújthatnak az erősen ingadozó forgalomról.

A módszerek többsége a Hurst-paraméter becslésén alapszik – minél nagyobb a H értéke, annál nagyobb az ingadozás, következésképpen rosszabb a kapcsolók és a forgalomirányítók teljesítménye az egyes forgalmi útvonalak mentén. Egyes módszerek megbízhatóbbak másoknál. A megbízhatóság számos tényezőtől függ, például a becslési technikától, a mintaérettől, az időintervallumtól, a forgalmi politikától stb. A publikált mérések alapján megvizsgáltuk a legkisebb becslési hibával rendelkező módszereket.[1] Ezek közül az *Újraskálázott módosított tartomány* („Rescaled Adjusted Range" – R/S)) módszert választottuk, mivel ennek megvalósítását megtaláltuk a hálózatról letölthető Benoit csomagban. Módszerünkhöz az ezen csomag által kiszámított Hurst-paraméter szolgál bemenetként.

Az M/Pareto forgalommodell és a Hurst-paraméter

Ismert, hogy az M/Pareto modell alkalmas a hosszú erős ingadozásokat tartalmazó, hosszú távon függő forgalomfolyamok modellezésére. A modellt eredetileg az ATM pufferszintjeinek az elemzésére javasolták, később az Ethernet, VBR videó és egykiszolgálós sorbanállási rendszerekben IP csomagfolyamok teljesítményének előrejelzésére is használták. A modellt itt nem csak egykiszolgálós sorra alkalmazzuk, hanem olyan összetett rendszerre, melyben kapcsolatok, kapcsolók és forgalomirányítók befolyásolják az egyes hálózatelemek teljesítményét.

Az M/Pareto modell egymást átfedő, λ beérkezési intenzitású erős ingadozások Poisson-folyamata. Az ingadozások r intenzitással generálnak csomagokat. Minden ingadozás az intervalluma kezdetétől egy Pareto-eloszlású ideig folytatódik. A Pareto-eloszlás alkalmazása eredményezi a hosszú távon

[1] Szórásnégyzet, együttes szórásnégyzet, Higuchi, maradék-szórásnégyzet, Újraskálázott módosított tartomány, Whittle-becslés, periodogram, regressziós maradéktag.

függő forgalmat jellemző nagyon hosszú ingadozásokat.

Annak a valószínűsége, hogy egy Pareto-eloszlású X valószínűségi változó túllép egy x határt:

$$\Pr\{X > x\} = \begin{cases} \left(\frac{x}{\delta}\right)^{\gamma}, & x \geq \delta \\ 1, & \text{egyébként} \end{cases}, \tag{17.2}$$

$$1 < \gamma < 2, \delta > 0 \; .$$

Az X várható értéke, az erős ingadozások $\mu = \delta\gamma/(\gamma - 1)$ várható hossza és ennek szórásnégyzete végtelen. Egy t intervallumot feltételezve ebben az intervallumban a csomagok M átlagos száma

$$M = \lambda t r \delta\gamma/(\gamma - 1) \; , \tag{17.3}$$

ahonnan

$$\lambda = \frac{M(\gamma - 1)}{tr\delta\gamma} \; . \tag{17.4}$$

Az M/Pareto modell aszimptotikusan önhasonló, és a Hurst-paraméterre teljesül, hogy

$$H = \frac{3 - \gamma}{2} \; koz. \tag{17.5}$$

17.8.2. A Hurst-paraméter megvalósítása a COMNET modellezőeszközben

A Hurst-paramétert és az M/Pareto modell egy módosított változatát a COMNET diszkrét-esemény szimulációs rendszerben valósítottuk meg. A diszkrét-esemény szimuláció segítségével valósághű hálózatjellemzőket kaphatunk. Ilyen jellemző például a kapcsolatok kihasználtsága és a kapcsolók, forgalomirányítók teljesítménye. Módszerünkkel felmérhetjük az összeállított erősen ingadozó forgalom káros következményeit, és előrejelezhetjük hatását a teljes hálózat teljesítményére.

Forgalmi mérések

Az alapkonfigurációs modell felépítéséhez egy nagyméretű intézményi hálózatban a Concord Network Health nevű hálózatelemző rendszerrel gyűjtöttünk össze nyomkövetési információkat. Különböző széles és keskeny sávú kapcsolatokon végeztünk méréseket, például 45 Mbps ATM, 56 Kbps, és 128 Kbps Frame Relay összeköttetéseken. A Concord Network Health rendszer segítségével adott ideig forgalmi méréseket végezhetünk az egyes hálózati csomópontoknál, mint például forgalomirányítóknál, kapcsolóknál. Az időintervallumot 6000 másodpercre állítottuk be, és a következőket mértük: a

eltelt idő (másodperc)	átlagos sávszélesség kihasználtság %	összes bájt/ másodperc	bejövő bájtok/ másodperc	kimenő bájtok/ másodperc
299	2.1	297.0	159.2	137.8
300	2.2	310.3	157.3	153.0
301	2.1	296.8	164.4	132.4
302	2.7	373.2	204.7	168.5
...	...	...	...	...

17.23. ábra. Nyomkövetési eredmények.

bájtok átlagos száma	üzenet késés (ms)	pufferszint (bájt)	eldobott csomagok száma	a kapcsolat sávszélesség kihasználtsága (%)		
				56 Kbps Frame Relay	ATM DS-3 szegmens	100 Mbps Ethernet
440.4279	78.687	0.04	0	3.14603	0.06	0.0031

17.24. ábra. A mért hálózatjellemzők.

küldött és fogadott bájtok és csomagok másodpercenkénti számát, a csomag késleltetést, az eldobott csomagok számát stb. A rendszer nem képes mérni az ingadozásokban lévő csomagok számát és az ingadozások hosszát, mint ahogy azt az előbbi M/Pareto modellben feltételeztük. Emiatt a korlátozás miatt a rendelkezésre álló adatoknak megfelelően kissé módosítjuk a forgalommodellt. ötpercenként készítünk pillanatfelvételeket a egy keskeny sávú Frame Relay kapcsolat forgalomáról egy távoli kliens és egy intézmény szervere között a 17.23. ábra szerinti formában.

A küldött bájtok átlagos számát, az üzenetkésést, a kliens helyi forgalomirányítójának a pufferszintjét, az eldobott csomagok számát, és az 56 Kbps sebességű kapcsolatnak, az ATM hálózat DS-3 szegmensének és a 100 Mbps sebességű Ethernet kapcsolatnak a célállomásnál lévő átlagos kihasználtságait a 17.24. ábra foglalja össze.

A COMNET a tranzakciókat a következőkkel reprezentálja: üzenetforrás, cél, üzenet méret, valamint az útvonalon lévő kommunikációs berendezések és kapcsolatok. Az üzenetküldési intenzitás egy beérkezési időköz eloszlással van megadva, azaz az egymást követő csomagok között eltelt idővel. Az M/Pareto modellben a Poisson-eloszlás λ intenzitással generál ingadozásokat vagy üzeneteket. Ezt az információt a szimulációban úgy adjuk meg, hogy az egymást követő érkezések közötti időintervallum hossza átlagosan $1/\lambda$. Erre a célra az exponenciális eloszlást használjuk. Az exponenciális eloszlásnak a beérkezések közötti időre való alkalmazásával egy Poisson-eloszlás szerinti érkezési mintát fogunk kapunk. COMNET-ben a beérkezések közötti időt az Exp($1/\lambda$) függvénnyel implementáltuk. A modellben a Concord Network Health-ben lévő mintának megfelelően 1 másodpercnek állítottuk be a

beérkezési időközt, ami megegyezik $\lambda = 1/$másodperc beérkezési intenzitással.

Az M/Pareto modellben az egyes erős ingadozások Pareto-eloszlású ideig tartanak. Mivel a Concord Network Health eszköz nem tudta mérni az ingadozások hosszát, ezért azt feltételezzük, hogy az ingadozást az egy üzenetben küldött vagy fogadott bájtok másodpercenkénti száma jellemzi. Mivel az ATM cellakezelő algoritmusa biztosítja, hogy azonos hosszúságú üzenetek azonos idő alatt legyenek feldolgozva, ezért a hosszabb üzenetek hosszabb feldolgozási időt igényelnek. Így azt mondhatjuk, hogy az ingadozások időtartamának eloszlása megegyezik az ingadozások méretének eloszlásával. Ezért módosíthatjuk az M/Pareto modellt úgy, hogy a Pareto-eloszlású ingadozásidőtartamot a Pareto-eloszlású ingadozás-mérettel helyettesítjük. Ezután δ-t nem az ingadozások átlagos időtartamából, hanem azok átlagos méretéből kapjuk.

COMNET-ben a Pareto-eloszlású ingadozásméretet két paraméterrel definiáljuk, a hely és az alak paraméterekkel. A hely paraméter megfelel (17.2) δ-jának, az alak pedig a γ-nak, ami a (17.5) egyenlettel a következőképpen számítható ki:

$$\gamma = 3 - 2H \ . \tag{17.6}$$

A Pareto-eloszlásnak végtelen várható értéke és szórása lehet. Ha viszont az alak paraméter nagyobb, mint 2, akkor mindkettő véges lesz. Ha az alak paraméter nagyobb, mint 1 és kisebb vagy egyenlő, mint 2, akkor a várható érték véges, viszont a szórásnégyzet végtelen. Továbbá ha ez kisebb vagy egyenlő 1-nél, akkor mind a várható érték, mind a szórásnégyzet végtelen.

A Pareto-eloszlás várható értékéből a következőt kapjuk:

$$\delta = \frac{\mu \cdot (\gamma - 1)}{\gamma} \ . \tag{17.7}$$

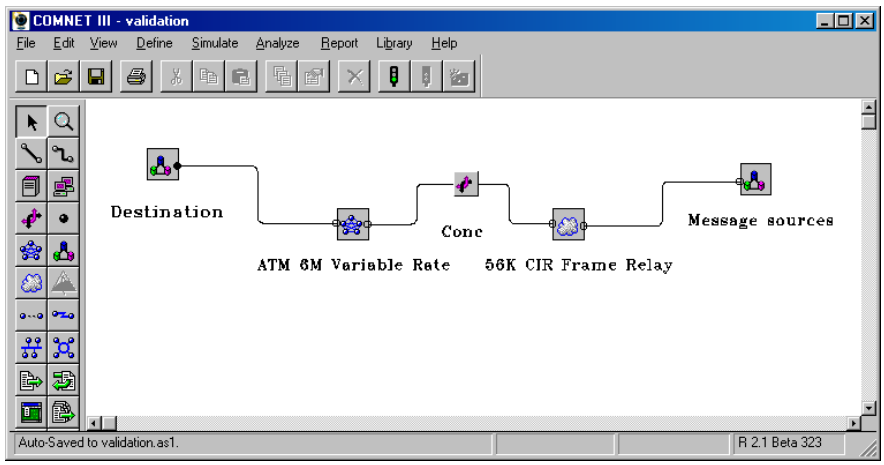
A (17.6) és (17.7) összefüggések lehetővé teszik az erősen ingadozó forgalom modellezését a valós nyomkövetési eredmények alapján a következő lépésekben:

a. Nyomkövetési adatok gyűjtése a Concord Network Health hálózatelemző felhasználásával.

b. A H Hurst-paraméter kiszámítása a nyomkövetési adatokból a Benoit csomag segítségével.

c. A COMNET eszköz exponenciális és Pareto-eloszlásainak felhasználásával, az előzőleg kiszámított paraméterekkel az érkezések közötti idő és az üzenetek méterének eloszlásának megadása.

d. Forgalomgenerálás a módosított M/Pareto modell szerint, és a hálózat teljesítményjellemzőinek mérése.



17.25. ábra. A hálózati topológiának az a része, ahol a méréseket végeztük..

Az előbbi lépesekkel generált forgalom erősen ingadozó olyan H paraméterrel, melyet valós forgalmi adatokból számítottunk ki.

17.8.3. Az alapkonfigurációs modell érvényesítése

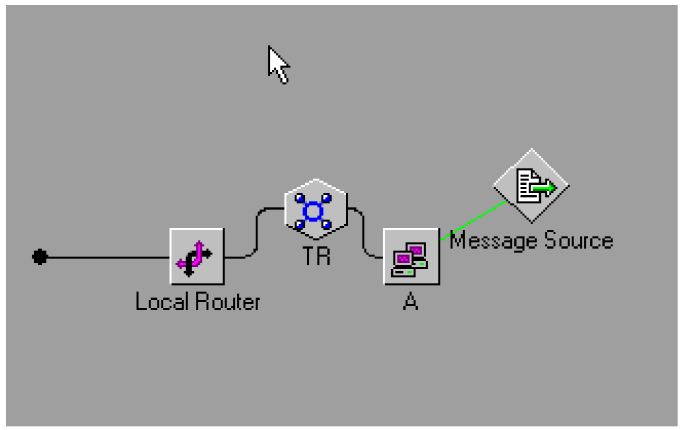
Az alapkonfigurációs modellt úgy érvényesítjük, hogy az 56 Kbps sebességű Frame Relay és a 6 Mbps sebességű ATM kapcsolatok különböző paramétereit összehasonlítjuk a valós hálózatról a Concord Network Health hálózatelemző segítségével kapott adatokkal. Az egyszerűség kedvéért csak az „összes bájtok/másodperc" oszlopot vizsgáljuk. A valós forgalomnak a Benoit csomag által kiszámított Hurst-paraméter értéke $H = 0.55$. A hálózat topológiája a 17.25. ábrán látható.

Az „üzenetforrások" ikon egy alhálózatot reprezentál, mely egy token ring hálózatból, egy helyi forgalomirányítóból és egy A kliensből áll, mely a „Célhálózat" alhálózatban lévő B kiszolgálónak küld üzeneteket (17.26. ábra).

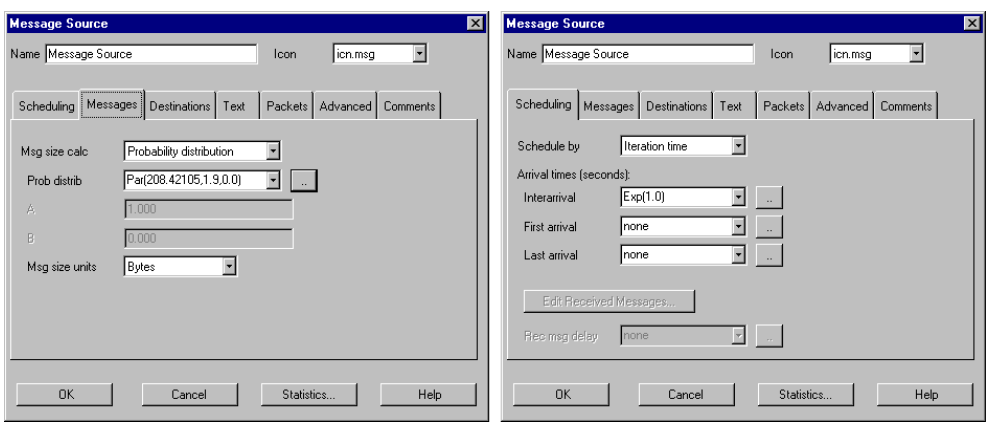
A beérkezési időköz és az üzenetek mérete az Exp(1) exponenciális és a Par(208.42, 1.9) Pareto függvényekkel van definiálva. A Pareto-eloszlás helyét (208.42) és alakját (1.9) a (17.7) és (17.7) képletek segítségével számítottuk ki az erős ingadozások átlagos hosszának (ez 440 bájt a 17.24. ábra alapján) és a $H = 0.55$ paraméternek a behelyettesítésével (17.27. ábra).

A 17.28. grafikon a megfelelő nehéz farkú Pareto-eloszlást és a kumulatív eloszlásfüggvényt szemlélteti (Az X tengelyen a bájtok számát ábrázoltuk).

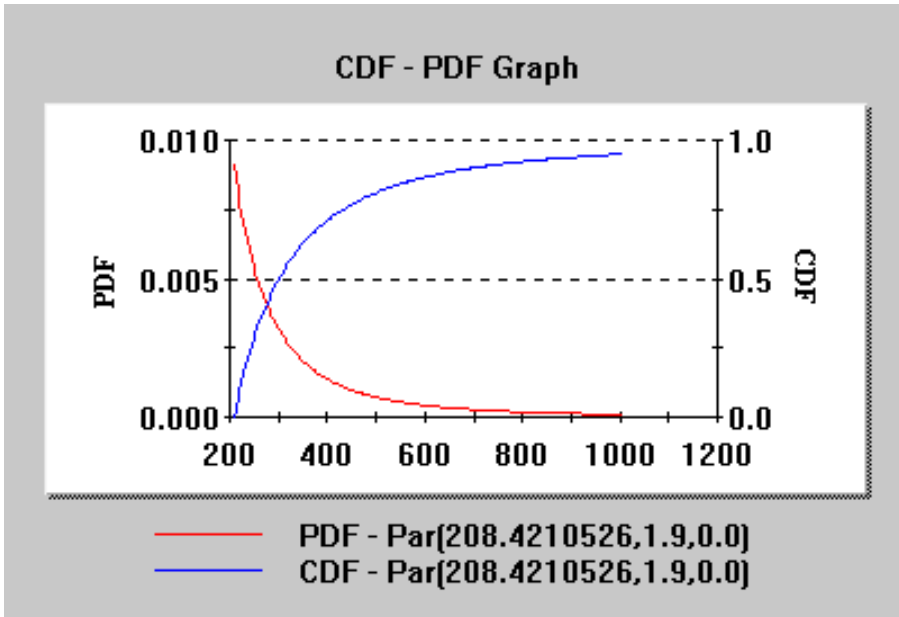
A „Frame Relay" ikon egy kerettovábbítási felhőt reprezentál 56 K információátviteli intenzitással (**C**ommitted **I**nformation **R**ate – CIR). A „Conc"



17.26. ábra. Az „üzenetforrás” távoli kliens.



17.27. ábra. A kliens által küldött üzenetek beérkezési időköze és mérete..



17.28. ábra. A Pareto-eloszlás 400 bájt várható érték és $H = 0.55$ Hurst-paraméter esetén.

forgalomirányító a Frame Relay hálózatot egy 6 Mbps sebességű, *változó intenzitásirányítással* (**V**ariable **R**ate **C**ontrol – VBR) rendelkező ATM hálózathoz kapcsolja. Ezt a 17.29. és 17.30. ábra szemlélteti.

A „Célhálózat" jelöli a B kiszolgálót tartalmazó alhálózatot, melyet a 17.31. ábra szemléltet.

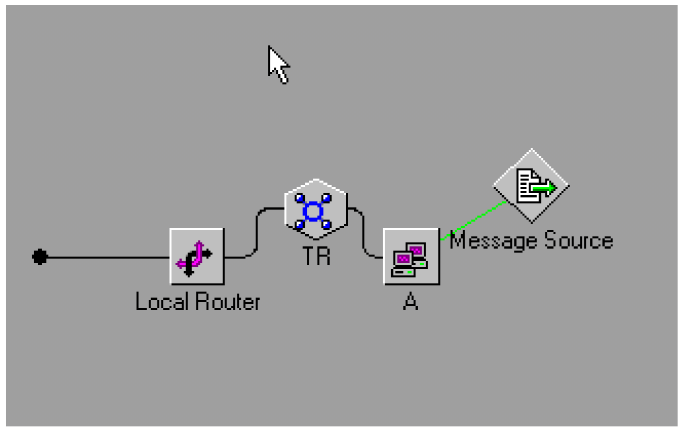
A modell eredményei a Frame Relay kapcsolat kihasználtságát tekintve ($0.035 \sim 3.5\%$) majdnem azonosak a valós mérésekkel (3.1%) (17.32. ábra).

Az üzenetek késleltetése szintén nagyon közel van a kliens és a szerver közötti mért értékhez (78 milliszekundum) (17.33. ábra).

A 17.34. ábrán látható, hogy a kliens forgalomirányítójának input puffer szintje körülbelül azonos a mért értékkel.

Hasonlóan, az ATM hálózat DS-3 kapcsolat-szegmensének és a célhálózat Ethernet kapcsolatának kihasználtsága is jól közelíti a valós hálózat méréseit (17.35. ábra).

Az is megfigyelhető a modell nyomkövetési eredményeiből, hogy a $H = 0.55$ Hurst-paraméterrel a modell majdnem ugyanolyan erősen ingadozó forgalmat generál, mint a valós hálózat. Továbbá az eldobott csomagok száma a modellben és a méréseknél is nulla. Így tehát van egy olyan kiinduló modellünk, amely jól reprezentálja a valós hálózatot.



17.29. ábra. A 6 Mbps sebességű, változó intenzitásirányítású ATM hálózat belső kapcsolatai.

17.8.4. A forgalom erős ingadozásának következményei

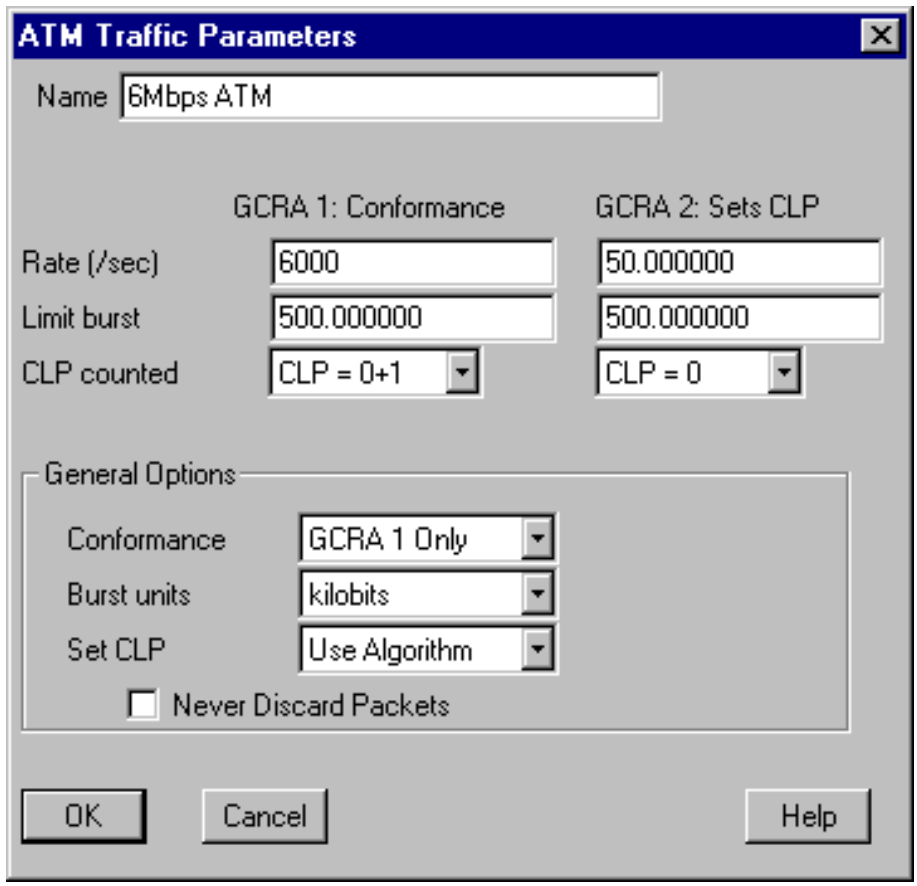
Módszerünk illusztrálására kifejlesztettünk egy COMNET szimulációs modellt a forgalom erős ingadozásának a hálózati kapcsolatokra, az üzenetek késleltetésére, a forgalomirányítók input pufferere és a nagy számú felhasználótól származó összetett forgalom miatt eldobott csomagok számára gyakorolt következményeinek mérésére. A modell az 17.3. alfejezetben leírt módon valósítja meg a Hurst-paramétert. Hogy a ritka események is megfelelő számban előforduljanak, a szimulációt 6000, 16000 és 18000 másodpercig ismételtük. Az eredményt mindegyik esetben nagyon hasonlónak találtuk.

Az erősen ingadozó forgalom forrásainak topológiája

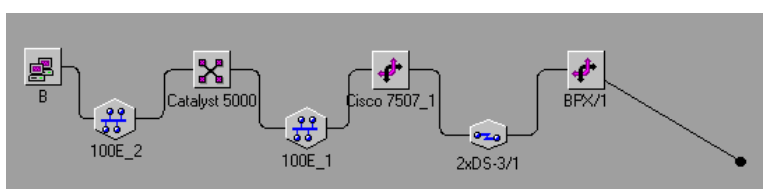
Az „üzenetforrás" alhálózatok az előbbi alapkonfigurációs modell szerint továbbítják az üzeneteket, azonos mérettel de különböző ingadozási paraméterrel: $H = 0.95$, $H = 0.75$ és $H = 0.55$. Kezdetben négy alhálózat működését szimuláltuk, alhálózatonként négy felhasználóval, melyek mindegyike az előzőekkel megegyezően ugyanolyan mennyiségű adatot küldött (átlagosan 440 bájtot másodpercenként) (17.36. ábra).

Kapcsolat-kihasználtság és üzenetkésleltetés

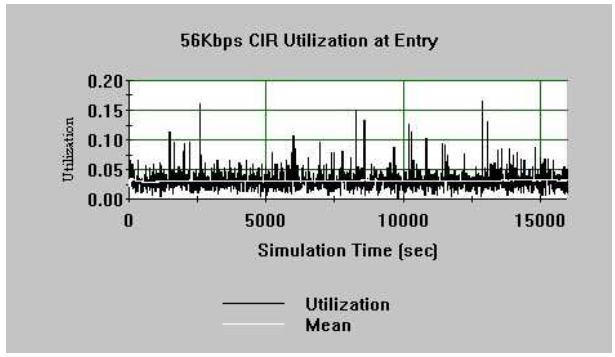
Először is egy Frame Relay kapcsolaton szeretnénk mérni és illusztrálni a különösen magas kihasználtság és üzenetkésleltetés csúcsokat. A modell forgalmát adó üzenetek méretét különböző Hurst-paraméterek határozzák meg. Az üzenetek az összehasonlíthatóság érdekében azonos méretűek. A COMNET eszköznek van egy nyomkövetési opciója is, mellyel adatokat tud gyűjteni a modell által generált forgalomról. Azt is ellenőriztük, hogy a



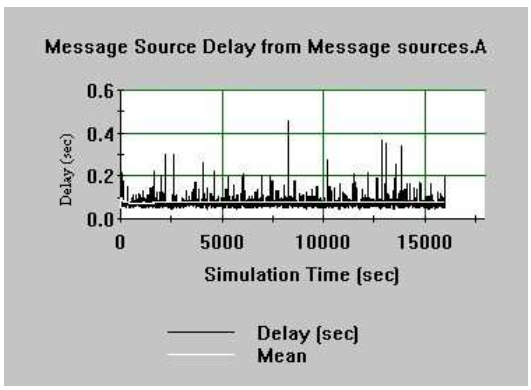
17.30. ábra. A 6 Mbps sebességű ATM kapcsolat jellemzői..



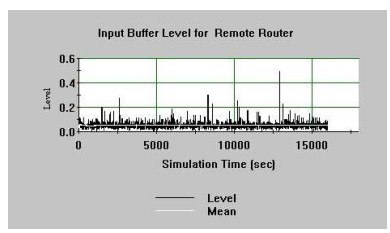
17.31. ábra. A „Célhálózat” alhálózat.



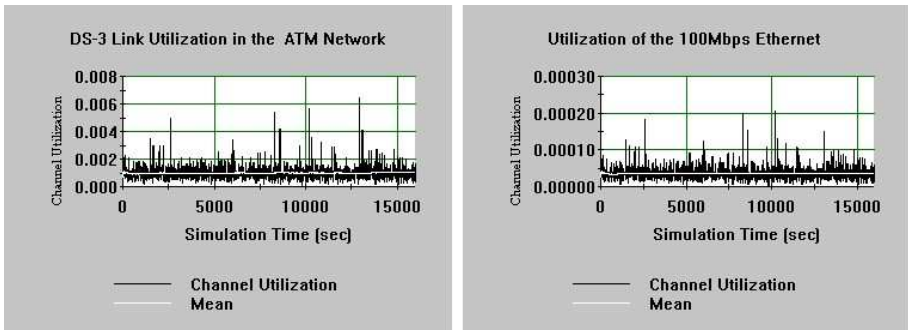
17.32. ábra. A Frame Relay kapcsolat kihasználtsága az alapkonfigurációs modellben.



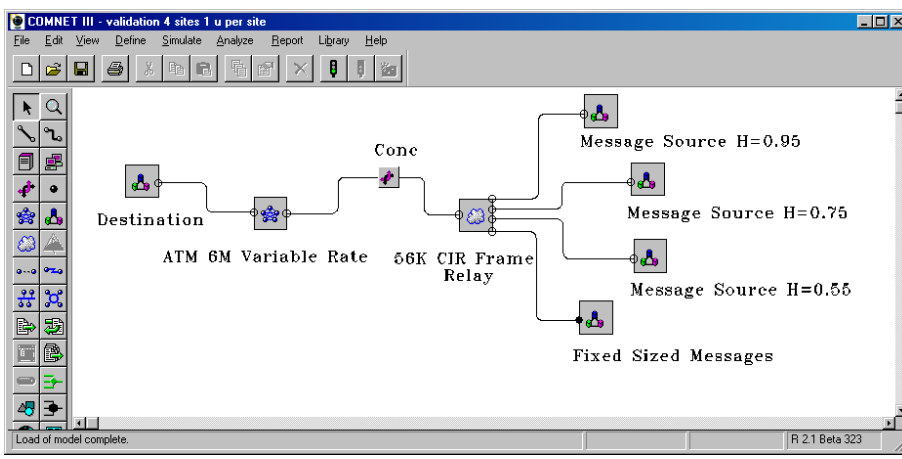
17.33. ábra. Az üzenetek késleltetése a kliens és a szerver között.



17.34. ábra. A távoli forgalomirányító input puffer szintje.



17.35. ábra. A DS-3 kapcsolat és a célhálózat Ethernet kapcsolatának kihasználtsága.



17.36. ábra. Erősen ingadozó forgalom forrásainak topológiája különböző Hurst-paraméterekkel.

	azonos méretű üzenetek	$H = 0.55$	$H = 0.75$	$H = 0.95$
átlagérték	0.12	0.13	0.13	0.14
csúcsérték	0.18	0.48	1	1

17.37. ábra. A szimulált kapcsolat-kihasználtságok átlagos és csúcsértékei.

	azonos méretű üzenetek	$H = 0.55$	$H = 0.75$	$H = 0.95$
átlagos válaszidő (ms)	75.960	65.61	87.880	311.553
válaszidő csúcsérték (ms)	110.06	3510.9	32418.7	112458.08
standard eltérés	0.470	75.471	716.080	4341.24

17.38. ábra. Válaszidő és erős ingadozás.

	azonos méretű üzenetek	$H = 0.55$	$H = 0.75$	$H = 0.95$
elfogadott csomagok	13282	12038	12068	12622
blokkolt csomagok	1687	3146	3369	7250
átlagos pufferhasználat (bájt)	56000858	61001835	62058222	763510495

17.39. ábra. Az eldobott cellák száma és az erős ingadozás közötti kapcsolat.

különböző Hurst-paraméterek esetén generált forgalomfolyamokból a Benoit-csomag hasonló Hurst-paramétereket számít ki.

Az 17.37. ábrán az egyes szimulált esetek kapcsolat-kihasználtságainak átlagos és csúcsértékei láthatók. A kihasználtságok nem százalékban, hanem [0,1] intervallumbeli értékekként vannak kifejezve.

A következő alfejezetben található ábrák jól láthatóvá teszik, hogy annak ellenére hogy a kapcsolat átlagos kihasználtsága közel azonos, a csúcsértékek gyakorisága és mérete növekszik az ingadozás erősödésével, ez pedig cellavesztéseket okoz a forgalomirányítókban és a kapcsolókban. A válaszidőre a 17.38. ábrán látható eredményeket kaptuk.

Az A függelék grafikonjai grafikusan illusztrálják a különböző Hurst-paraméterek és a válaszidők közötti kapcsolatot.

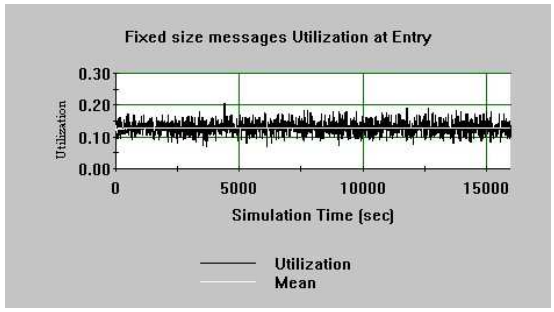
Az input pufferek szintje nagy számú felhasználó esetén

Megmértük az erősen ingadozó cellaforgalom miatt az ATM hálózat egy forgalomirányítójának input pufferénél eldobott cellák számát is. Körülbelül 600 felhasználó összegzett forgalmát szimuláltuk, melyek a valós forgalmi mérésekkel egyezően ugyanannyi bájtot küldtek el másodpercenként. Az 17.39. ábra a blokkolt csomagok számát foglalja össze az egyes eseteknél.

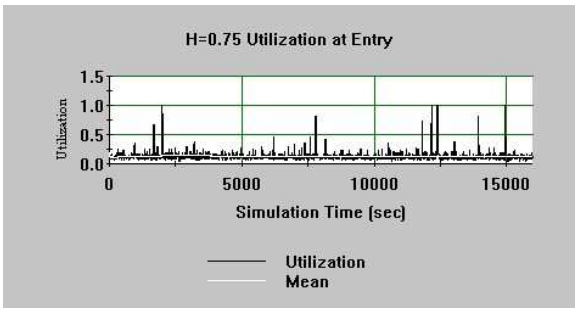
17.8.5. Következtetések

Az előzőekben egy diszkrét-esemény szimulációs módszert mutattunk be, amely alkalmas különböző erősen ingadozó forgalmat továbbító hálózatok jellemzőinek mérésére. Korábbi tanulmányok bebizonyították, hogy az erősen ingadozó adatfolyamok egyesítése szintén erősen ingadozó adatfolyamot eredményez. Ezért a hagyományos, hálózattervezésre használt módszerek és modellek módosítására van szükség. A mi módszertanunkat a fekete doboz modellek helyett a strukturált modellek csoportjába soroljuk. A strukturális modellek arra a környezetre összpontosítanak, amiben a modellek adatai össze lettek gyűjtve, azaz a hálózati komponensek hierarchiájára, melyekből napjaink kommunikációs rendszerei felépülnek. Habár a fekete doboz modellek hasznosak lehetnek más környezetekben, nem olyan könnyű őket alkalmazni napjaink hálózatainak tervezésében, irányításában és ellenőrzésében. Egy jól ismert modellt, az M/Pareto modellt implementáltuk a COMNET diszkrét-esemény szimulációs csomag felhasználásával Ez lehetővé teszi az önhasonló forgalom káros következményeinek elemzését nem csak egy egykiszolgálós sorra, hanem különböző kapcsolódó hálózati komponensek összteljesítményére nézve is. Valós hálózati nyomkövetési információk felhasználásával olyan modellt készítettünk és érvényesítettünk, mellyel mérni és grafikusan illusztrálni tudtuk az erősen ingadozó forgalomnak a kapcsolatok kihasználtságra, az üzenetkésleltetésekre és a pufferek teljesítményére gyakorolt hatását Frame Relay and ATM hálózatokban. Megmutattuk, hogy az erősödő ingadozás nagyon nagy kapcsolat-kihasználtságot, válaszidőt és sok eldobott csomagot eredményez, továbbá szimulációval meghatároztunk különböző teljesítményjellemzőket.

A csomagválasztás hangsúlyozza olyan eszközöknek a szükségességét, melyek hasznosak lehetnek nem csak elméleti szakemberek, hanem hálózattervezők és mérnökök számára is. Ez a cikk a meglévő, jól ismert elméleti eredmények és ezeknek a mindennapi gyakorlati hálózatelemzésben és modellezésben való alkalmazása közötti rést szeretné csökkenteni. Meglehetősen jó volna, ha a mérő, megfigyelő és irányító eszközökben rendelkezésre állnának a megfelelő forgalommodellek. Az itt tárgyalt modell segítheti a forgalommodellezés elsődleges felhasználóit, a hálózattervezőket és mérnököket abban, hogy megértsék a hálózati forgalom dinamikus természetét, továbbá támogathatja őket mindennapi munkájukban.



17.40. ábra. A Frame Relay link kihasználtsága azonos méretű üzenetek esetén.



17.41. ábra. A Frame Relay link kihasználtsága $H = 0.75$ Hurst-paraméter esetén (magasabb csúcsokkal).

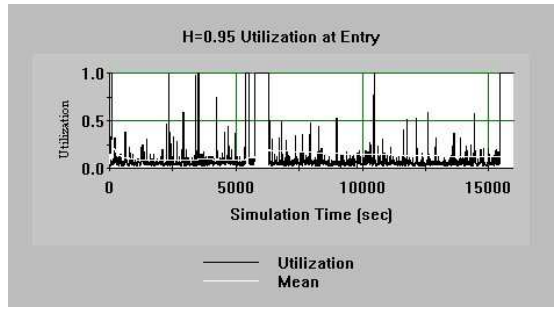
17.9. Mérési adatok bemutatása

17.9.1. Kapcsolat-kihasználtsági mérések

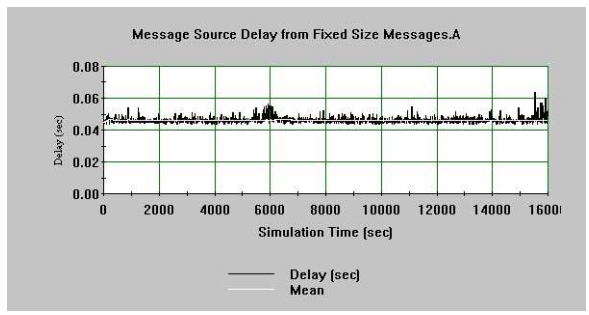
Az 17.40–17.42. ábrák azt szemléltetik, hogy bár a különböző Hurst-paraméterek esetén az átlagos kapcsolat-kihasználtságok majdnem azonosak, az ingadozás növekedésével a csúcsértékek gyakorisága és mérete nő, ez pedig a forgalomirányítókban és kapcsolókban cellavesztéseket okoz. A kihasználtságok nem százalékban, hanem [0,1] intervallumbeli értékekként vannak kifejezve.

17.9.2. Üzenetkésleltetési mérések

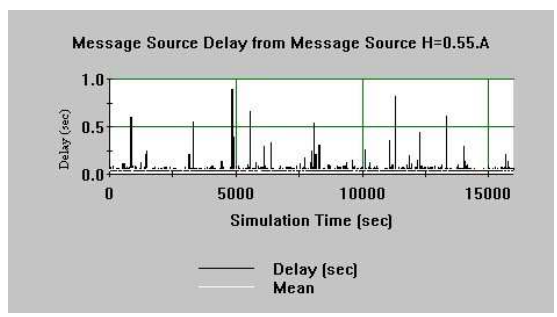
A 17.43–17.45. ábrák a különböző Hurst-paraméterek és a válaszidők közötti kapcsolatot szemléltetik.



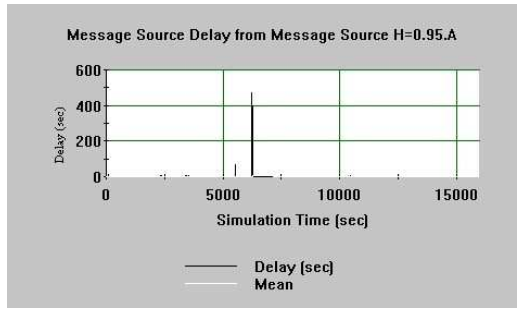
17.42. ábra. A Frame Relay link kihasználtsága $H = 0.95$ Hurst-paraméter esetén (sok magas csúccsal).



17.43. ábra. Üzenetkésleltetés azonos méretű üzenetek esetén.



17.44. ábra. Üzenetkésleltetés $H = 0.55$ esetén (magasabb válaszidő csúcsokkal).



17.45. ábra. Üzenetkésleltetés $H = 0.95$ esetén (nagyon magas válaszidő csúccsal).

Gyakorlatok

17.9-1. Nevezzünk meg néhány, a következő fogalmakhoz kapcsolódó tulajdonságot, eseményt, tevékenységet és állapotváltozót:

- Kiszolgáló
- Kliens
- Ethernet
- Csomagkapcsolt hálózat
- Híváslétesítés celluláris mobil hálózatban
- A TCP *lassú indulás algoritmusa*

17.9-2. Olvassunk el egy cikket a hálózatok szimulációjának alkalmazásáról, és írjunk beszámolót arról, hogyan közelíti meg a cikk a modell érvényesítést.

17.9-3. Ez a gyakorlat feltételezi, hogy rendelkezésre áll valamilyen hálózatelemző szoftver (például Lananalyzer for Windows vagy bármilyen más eszköz), mely elemezni tudja a hálózati forgalmat és információkat tud gyűjteni róla. A következőkben mi az előbb említett eszközt használjuk.

- Kezdjük el egy állomány továbbítását a helyi hálózatban egy kliens és egy kiszolgáló között. Figyeljük meg a részletes statisztikákat az adatátviteli vonal kihasználtságáról és a másodpercenként átvitt csomagok számáról, majd mentse el a megfelelő grafikonokat.
- A Lananalyzer Help menüjénél válasszuk ki és olvassuk el a *Capturing and Analyzing Packets* fejezetet.
- Az állomány átvitele során csak a kliens és a kiszolgáló közötti csomagokat vizsgáljuk.

- Mentsük el a csomagokról gyűjtött nyomkövetési információkat .csv formátumban. Elemezzük ezt az állományt táblázatkezelő felhasználásával. Figyeljük meg, vannak-e szokatlan protokoll-események, mint például a csomagok között eltelt túl hosszú idő, túl sok hibás csomag stb.

17.9-4. Ebben a gyakorlatban a Sniffer különböző hálózatelemzési és alapkonfiguráció készítési funkcióit vizsgáljuk. Az alapkonfiguráció definiálja a hálózatot jellemző tevékenységeket, és ennek ismeretében fel tudjuk ismerni a tipikustól eltérő működést. Ezt okozhatja valamilyen probléma, vagy a hálózat növekedése is. Az alapkonfigurációs adatokat akkor kell gyűjteni, amikor a hálózati működése tipikusnak mondható. Egyes statisztikák készítéséhez, mint például a sávszélesség-kihasználtság vagy a csomagok száma másodpercenként, egy olyan grafikont kell készíteni, amely egy adott időintervallumban ábrázolja az információkat. Erre azért van szükség, mert az olyan mintavétel, amely túl rövid időintervallumban gyűjt adatokat, félrevezető lehet. Egy vagy több hálózatkomponens hozzáadása után érdemes egy alapkonfigurációt készíteni, így később össze lehet hasonlítani a hozzáadás előtti és utáni tevékenységeket. Az összegyűjtött adatok exportálhatók más programok, például táblázatkezelők és modellezőeszközök számára, amelyekkel további elemzéseket készíthetünk és amelyek segítik az összegyűjtött adatok kezelését.

A Sniffer egy nagyon jól alkalmazható hálózatelemző eszköz. Számos jól integrált funkciót tartalmaz, melyeket a következőkre használhatunk:

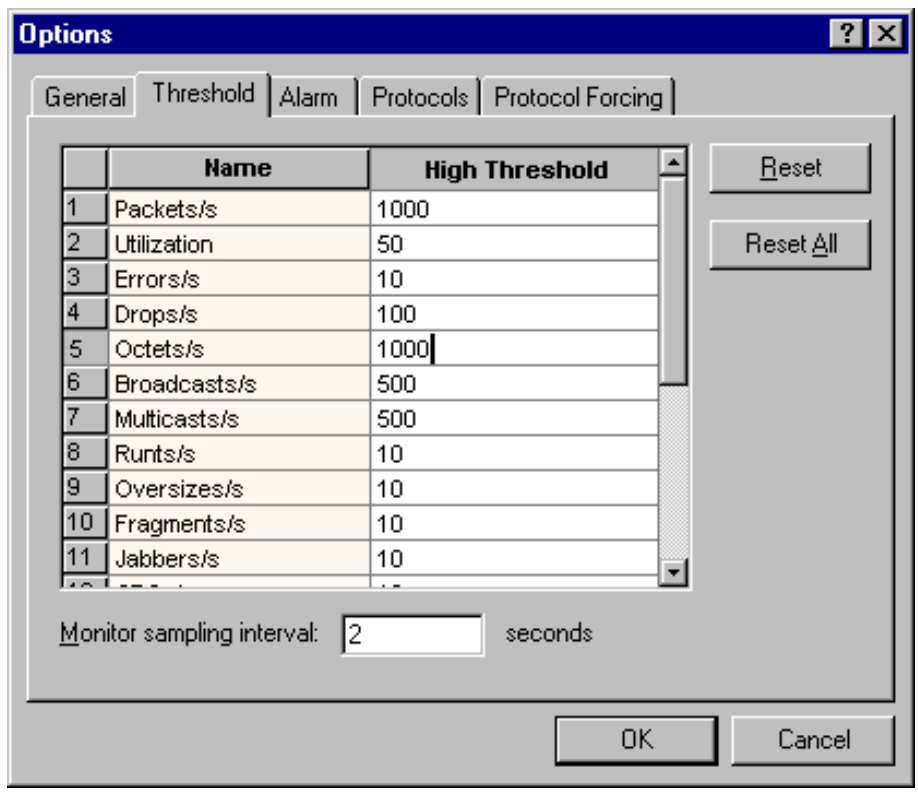
- Forgalomnyomkövetési információk gyűjtése részletes elemzés céljára.
- Problémák megállapítására az Expert Analyzer alkalmazásával.
- A hálózati tevékenységeknek valós idejű megfigyelésére.
- Részletes kihasználtsági és hibastatisztikák gyűjtésére az egyes állomásokról, párbeszédekről vagy a hálózat bármely részéről.
- A korábbi kihasználtsági és hibainformációknak alapkonfigurációs elemzés céljára történő tárolására.
- Problémák esetén az adminisztrátorokat figyelmeztető látható és hallható riasztások létrehozására.
- A hálózat aktív eszközökkel történő forgalomszimulációs vizsgálatára, válaszidő mérésre, hop számlálásra és hibaelhárításra.
- A Monitor menü History Samples pontja lehetővé teszi a hálózati tevékenységeknek egy időintervallumon keresztül való rögzítését. Ezek az adatok használhatóak az alapkonfiguráció elkészítéséhez, ami segíti az egyes határértékek beállítását, melyeknek a normálistól eltérő működés

esetén történő átlépése kiváltja az egyes riasztásokat. Továbbá ezek az adatok szintén hasznosak a hálózat terhelésének a hosszú távú változásainak meghatározására, így tervezhetővé válnak a jövőbeli hálózatbővítések.

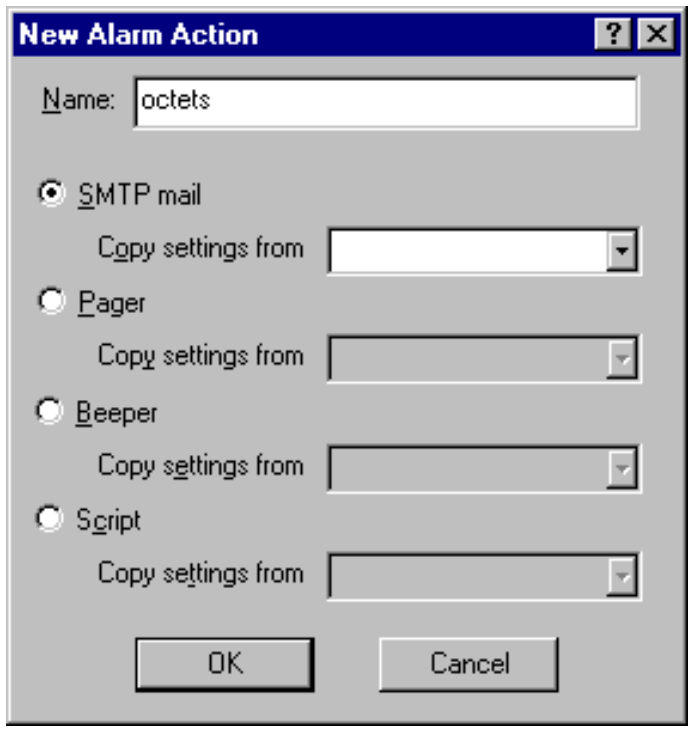
- Egyidejűleg legfeljebb 10 hálózati tevékenység figyelhető meg vele. Egy adott tevékenység megfigyelésére több statisztikakészítés is elindítható, így egyidejűleg mind a rövid, mind a hosszú távú tendenciák rögzíthetők. A korábbi minták megfigyelérése rendelkezésre álló hálózati események az Adapter párbeszédablakban kiválasztott adaptertől függenek. Például, egy token ring hálózat esetén a különböző token ring kerettípusok mintái (mint például a beacon keretek) figyelhetők meg, Frame Relay hálózat esetén pedig a különböző Frame Relay kerettípusok (például LMI keretek) mintái. A megfigyelhető események adapterenként változnak.

Gyakorlati feladatok

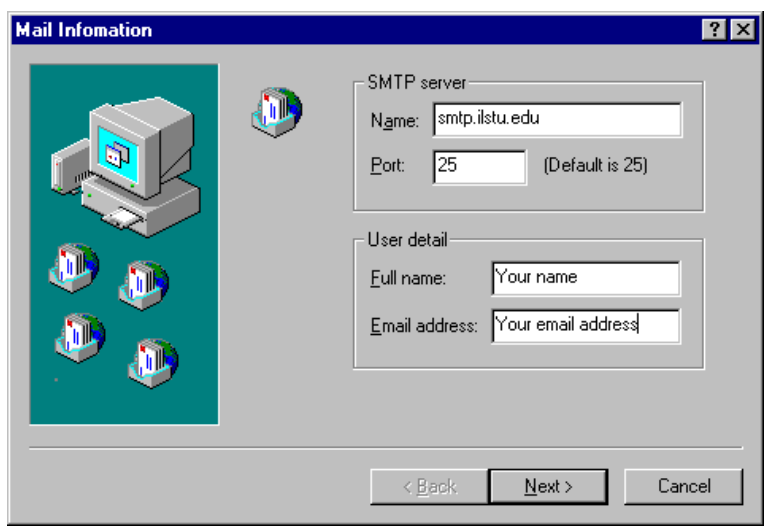
- Állítsunk be egy szűrőt (Capture/Define filter) a saját PC-je és valamelyik távoli munkaállomás között az IP forgalom mintavételezésére.
- Állítsuk be a Monitor/History Samples/Multiple History-nál a következőket: Octets/s (Oktet/másodperc), Utilization (Kihasználtság), Packets/s (Csomag/másodperc), Collision/s (Ütközés/másodperc), és Broadcasts/s (Üzenetszórás/másodperc).
- Állítsuk be a mintavételi intervallumot 1 másodpercre (jobb klikk a Multiple ikonon és ott properties, Sample).
- Indítsuk el a hálózat megfigyelését (jobb klikk a Multiple ikonon, majd Start Sample).
- Szimuláljunk valamilyen szokásos hálózati forgalmat, például töltsünk le egy nagyméretű állományt egy kiszolgálótól.
- Rögzítsük a „Multiple History"-t ezalatt a „szokásos hálózati forgalom" alatt. Ezt tekintjük az alapkonfigurációnak.
- Állítsuk a Tools/Options/MAC/Threshold-nál az oktet/másodperc értékét az alapkonfigurációs érték 10-szeresére. Definiáljunk egy riasztást az oktet/másodpercre: Amikor eléri ezt a határértéket, küldessünk egy levelet a saját elektronikus címünkre. Az 17.46. ábrán azt feltételezzük, hogy ez a határérték 1000.
- A riasztást a 17.47. ábrán látható módon adjuk meg.
- Ezután állítsuk be az SMTP kiszolgálót a helyi, saját levelezőszerverére (17.50. ábra).
- A probléma komolyságát (Severity) állítsuk kritikusra (Critical) (17.49. ábra).



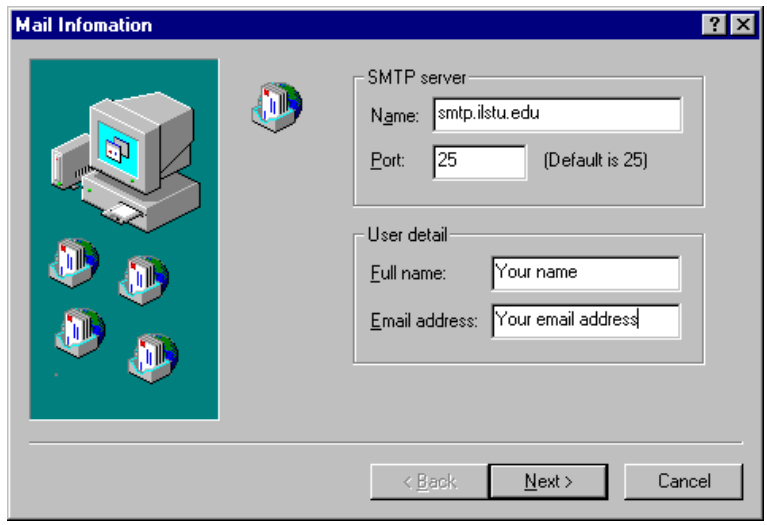
17.46. ábra. Beállítások.



17.47. ábra. Új riasztás akció.



17.48. ábra. Levelezési információk.



17.49. ábra. Beállítások.

- Gyűjtsünk nyomkövetési információkat a forgalomról (Capture/Start) az állomány letöltése alatt.
- A letöltés befejeződése után állítsuk le az információgyűjtést (Capture/Stop majd Display).
- Az Expert Decode opcióval elemezzük a csomagok TCP/IP rétegeit!
- Nézzük meg, megérkezett-e a Sniffer Pro-tól a „riasztás e-mail". Várhatóan az alábbihoz hasonló levelet fogjuk kapni, mely jelzi az oktet/másodperc határérték túllépését:

```
From: ...
Subject: Octets/s: current value = 22086, High Threshold = 9000
To: ...

This event occurred on ...
```

Mentse el a következő állományokat:

- A „Baseline screens"-t
- A Baseline Multiple History.csv állományt
- A „riasztás e-mail"-t

17.9-5. A gyakorlat célja egy alapkonfigurációs modell felépítése és érvényesítése egy hálózatmodellező eszköz felhasználásával. Feltételezzük, hogy a modellező számára elérhető egy szimulációs modellezőeszköz, például

a COMNET vagy az OPNET.

Először gyűjtsünk válaszidő statisztikákat egy távoli számítógép pingelésével. A ping parancs a hálózaton egy adott klienstől egy kiszolgálóhoz küldött csomagok oda-vissza útjának idejét méri. A parancs egy lehetséges formátuma a következő: ping hosztnév -n x -l y -w z > fájlnév. Itt „x” a küldendő csomagok száma, „y” a csomaghossz bájtokban, „z” az időtúllépési érték és a „fájlnév” az állomány neve, amibe az összegyűjtött statisztikák kerülnek.

Például, a ping 138.87.169.13 -n 5 -l 64 > c:
ping.txt parancs a következő állományt hozza létre:

```
Pinging 138.87.169.13 with 64 bytes of data:
Reply from 138.87.169.13: bytes=64 time=178ms TTL=124
Reply from 138.87.169.13: bytes=64 time=133ms TTL=124
Reply from 138.87.169.13: bytes=64 time=130ms TTL=124
Reply from 138.87.169.13: bytes=64 time=127ms TTL=124
Reply from 138.87.169.13: bytes=64 time=127ms TTL=124
```

- Egy táblázatkezelő felhasználásával készítsünk egy hisztogramot ezekről az időkről és a csomagok sorszámáról.
- Készítsünk hisztogramot a válaszok számáról és a válaszidőkről.
- Készítsük el a válaszidők kumulatív sűrűségfüggvényét, az eloszlás farkánál a részleteket is feltüntetve.
- Készítsük el az átvitelek alapkonfigurációs modelljét. A forgalomjellemzőket az előző lépésben készített kumulatív sűrűségfüggvény határozzuk meg.
- Érvényesítsük a modellt.
- Mennyi a kapcsolat kihasználtsága 32 és 64 bájt hosszúságú üzenetek esetén?

17.9-6. Feltételezzük, hogy a modellező számára elérhető egy szimulációs modellezőeszköz, mint például a COMNET, OPNET stb. Ebben a gyakorlatban néhány gyakran használt kép-állomány helyét szeretnénk meghatározni egy laborban. Az előrejelzések szerint a következő évben az új kliensek hozzáadása megháromszorozza ezeknek az állományoknak a használatát. Ezek tárolhatók vagy a kiszolgálón, vagy a kliens munkaállomásokon. A könnyebb karbantartás miatt a kiszolgálón való tárolást részesítjük előnyben. Az aktuális hálózatnak egy alapkonfigurációs modelljét fogjuk elkészíteni, és megmérjük a kapcsolat-kihasználtságot az állományátvitelek alatt. Továbbá érvényesítjük a modellt az aktuális forgalomjellemzőkkel. A

forgalom skálázásával előrejelezhetjük a kapcsolat-kihasználtságot a munkaállomások hozzáadása után megháromszorozódott forgalom esetén.

- Készítsük el az alapkonfigurációs modell topológiáját.
- Gyűjtsünk forgalom-nyomkövetési információkat az átvitel alatt, és importáljuk őket.
- Futtassuk és érvényesítsük a modellt (Az átvitt üzenetek számának a modellben meg kell egyeznie a nyomkövetési állományban lévő számmal, a szimuláció ideje egyenlőnek kell lennie a csomagok közötti idők – (**I**nterpacket **T**ime– IT) – összegével, és a kapcsolat-kihasználtságnak közel egyenlőnek kell lennie a nyomkövetés alatti átlagos kihasználtsággal).
- Írassuk ki az átvitt üzenetek számáról, az üzenetkésleltetésről, a protokollok általi kapcsolat-kihasználtságról és a teljes kapcsolat-kihasználtságról szóló jelentéseket.
- Háromszorozzuk meg a forgalmat.
- Írassuk ki az átvitt üzenetek számáról, az üzenetkésleltetésről, a protokollok általi kapcsolat-kihasználtságról és a teljes kapcsolat-kihasználtságról szóló jelentéseket.
- Ha a kapcsolat-kihasználtság az alapkonfigurációs határérték alatt van, akkor a kiszolgálón hagyjuk a kép-állományokat, egyébként a munkaállomásokra helyezzük át őket.
- Kérdés: Hol érdemesebb tárolni ezeket, a klienseken vagy a kiszolgálón?

17.9-7. Ennek a gyakorlatnak a célja az osztott és a kapcsolt Ethernet teljesítményének összehasonlítása. Megmutatjuk, hogy az örökség („legacy") vagy osztott Ethernet átalakítása kapcsolt Ethernetté csak akkor indokolt, ha az ütközések száma meghalad egy adott határértéket. *a.* Készítsük el egy osztott Ethernetet használó helyi hálózati kliens/szerver alkalmazás modelljét. A modell tartalmaz egy 10Base5 Ethernetet, mely egy kiszolgálót (Webszerver) és három számítógépcsoportot kapcsol össze (Kliens 1, Kliens 2, Kliens 3). Minden csoportnak három számítógép a tagja, továbbá minden csoportnak van egy „Web Request" nevű üzeneteket generáló forrása. A kiszolgálónak ezekre egy „Web Server" alkalmazása válaszol. Minden „Web Request" forgalmat generál a kiszolgálóhoz. Amikor a kiszolgáló megkap egy „Web Request" üzenetet, olyankor egy „Web Response" üzenetet generál és elküldi a megfelelő kliensnek.

- Minden „Web Request" egy 10000 byte hosszú üzenetet jelent, amit a forrás minden Exp(5) másodpercben küld a Webszerverhez. Állítsa az

üzenet szövegét „Web Request”-re.

- A Webszerver visszaküld egy üzenetet „Web Response” szöveggel. Az üzenet mérete 10000 és 100000 bájt között változik, ezt a Geo(10000, 100000) eloszlás határozza meg. A szerver csak a kapott „Web Request” üzenetekre válaszol. Állítsa a válaszüzenetet „Web Response”-ra.
- A többi paraméternél használjuk az alapértelmezett értékeket.
- Válasszuk a „Csatornakihasználtságot” („Channel Utilization”) és az „Ütközési statisztikák”-at („Collision Stats”) a „Kapcsolatok beszámolói”-nál („Links Reports”).
- Válasszuk az „Üzenetkésleltetés”-t az „Üzenet + válaszjelentés”-nél („Message + Response Source Report”).
- Futtassuk a szimulációt 100 másodpercig. Megadhatjuk az animáció opciót is.
- Írassuk ki a „Kapcsolat-kihasználtság”-ot és az „Ütközési statisztikák”-at megjelenítő jelentést, valamint a forgalom forrásai közötti üzenetkésleltetésről szóló jelentést.

b. A válaszidő csökkentése érdekében alakítsuk át az osztott LAN-t kapcsolt LAN-ná. A kliens-szerver paramétereket és az Ethernet sebességét változatlanul hagyva helyezzünk el egy Ethernet kapcsolót a kliensek és a szerver közé. (A szerver egy full-duplex 10Base5 kapcsolattal csatlakozik a kapcsolóhoz.)

- Írassuk ki a „Kapcsolat-kihasználtság”-ot és az „Ütközési statisztikák”-at megjelenítő jelentést, valamint a forgalom forrásai közötti üzenetkésleltetésről szóló jelentést.

c. Mindkét modellben cseréljük a 10Base5 kapcsolatokat 10 BaseT kapcsolatokra. Ekkor az előző esetekkel ellentétben nem azonos mértékű relatív javulást fogunk tapasztalni a válaszidőben. Magyarázzuk meg, miért.

17.9-8. Egy vállalat helyi hálózatának egy része két alhálózatból áll. Mindkettő egy-egy osztályt szolgál ki. Az egyik hálózat az IEEE 802.3 CSMA/CD 10BaseT Ethernet szabvány szerint üzemel, a másik pedig az IEEE 802.5 16Mbps Token Ring szabvány szerint. A két hálózat egy Cisco 2500-as forgalomirányítóval van összekapcsolva. Az Ethernet LAN 10 számítógépet tartalmaz, melyek közül egy e-mail szervernek van kijelölve mindkét osztály számára. A token ring LAN szintén 10 számítógépet tartalmaz, melyek közül egy fájlszervernek van kijelölve az osztályok számára.

A vállalat jelenleg mindkét osztályra alkalmazottak felvételét tervezi. Habár a jelenlegi hálózati konfiguráció valószínűleg nem lesz képes kiszolgálni az új alkalmazottakat, a vállalatnak nincs semmilyen módszere a hálózat

kihasználtságának vagy a késleltetésének mérésére. A vállalat az új alkalmazottak felvétele előtt szeretné felmérni ezeket az aktuális alapkonfigurációs szinteket. Az alkalmazottak egyébként is már mindkét osztályról panaszkodtak a fájlszervertől való letöltés lassúsága miatt.

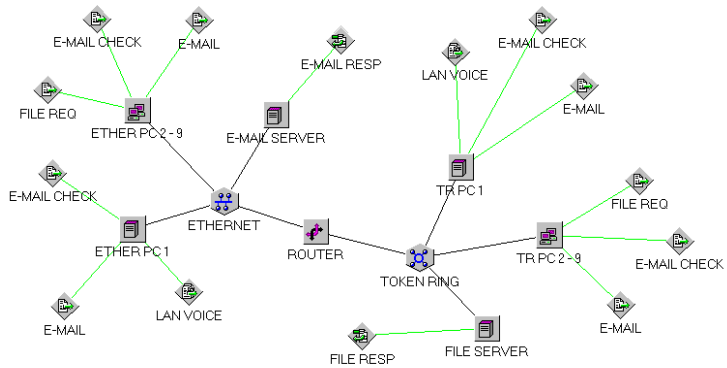
A LAN-okon keresztülfolyó közös forgalomnak egy felmérése szerint a forgalom többsége a következő forrásokból származik: elektronikus levelezésből, a különböző alkalmazások állomány-átviteléből és a hang alapú levelezési rendszerből, amely lehetővé teszi a vezetők számára, hogy hallható üzeneteket küldjenek az osztályukon dolgozó alkalmazottaknak. Az üzenetjellemzők statisztikai leírásának alapjául az alkalmazottakkal folytatott beszélgetések és az üzenetek átlagos méretének becslése szolgált.

Az elektronikus levelezést minden alkalmazott használja mindkét osztályon. Az interjúkból kiderült, hogy a levélküldések időköze leírható egy 900 másodperc várható értékű exponenciális eloszlással. A levelek mérete pedig leírható egy olyan egyenletes eloszlással, mely szerint a levélméret 500 és 2000 bájt között van. Az összes e-mail az Ethernet hálózaton lévő e-mail szerverhez továbbítódik, ahol pedig a megfelelő felhasználó postafiókjában kerül tárolásra.

Ha egy felhasználó el akar olvasni egy levelet, akkor azt az e-mail szervertől kell lekérnie. A postafiók ellenőrzéseinek ideje leírható egy Poisson-eloszlással, melynek várható értéke 900 másodperc. Az erre használt üzeneteknek a mérete 60 bájt. Ha egy felhasználó le szeretne tölteni egy levelet, akkor a szerver beolvassa a felhasználóhoz tartozó állományt és továbbítja a kért levelet az adott alkalmazott számítógépére. Az állomány beolvasásának és a benne lévő üzenetek feldolgozásának ideje leírható egy egyenletes eloszlással, mely 3 és 5 másodperc közötti értékeket vehet fel. A levelek mérete egy normális eloszlással írható le, melynek várható értéke 40000 bájt és standard eltérése 10000 bájt.

Mindkét osztályon nyolc alkalmazott van, akiknek saját számítógépe van, és akik a fájlszervertől is kérnek le állományokat. Ezeknek a kéréseknek az érkezési időköze leírható egy 900 másodperc várható értékű exponenciális eloszlással. A kérések mérete egyenletes eloszlást követ, 10 bájt minimummal és 20 bájt maximummal. Ezeket a kéréseket kizárólag a token ring hálózatban lévő fájlszervernek küldik. Egy kérés érkezésekor a szerver beolvassa a kért állományt és elküldi az adott számítógépnek. Ez a feldolgozás csak nagyon kis késleltetést jelent. Az állományok mérete egy normális eloszlással írható le, melynek várható értéke 200000 bájt és standard eltérése 25000 bájt.

A hang alapú üzenetküldést mindkét osztályon csak a vezetők használják, akik általában csak a saját részlegükben lévő alkalmazottaknak küldenek ilyen üzeneteket. A továbbító alkalmazás először kapcsolatot létesít az alkalmazott



17.50. ábra. Levelezési információk.

számítógépével. Ha ez felépült, akkor továbbítja az üzenetet. Ezek mérete normális eloszlással írható le, melynek várható értéke 50000 bájt és standard eltérése 1200 bájt. Beérkezési időközük szintén normális eloszlással írható le, melynek várható értéke 1000 másodperc és standard eltérése 10 bájt.

Az összes üzenetforrás a TCP/IP protokollkészletet használja, és a csomagok elkészítésének becsült ideje 0.01 milliszekundum.

A hálózat topológiájának hasonlónak kell lennie a 17.50. ábrán a COMNET-ben láthatóhoz.

A következő jelentések használhatók a szimulációnál:

- Kapcsolatjelentések: Csatorna-kihasználtság és Ütközési statisztikák az egyes kapcsolatok esetén.
- Csomópontjelentések: A beérkezett üzenetek száma az egyes csomópontok esetén.
- Üzenet és válasz jelentések: Az üzenetek késleltetése az egyes csomópontok esetén.
- Session Source jelentések: Az üzenetek késleltetése az egyes csomópontok esetén.

A modell futtatásakor jóval nagyobb válaszidőt fog észlelni a fájlszervernél. Ha a szolgáltatás-minőségi szint kisebb válaszidőt igényel, akkor milyen

megoldást lehet javasolni annak csökkentésére? Jó megoldás lenne még egy fájlszerver üzembe helyezése az Ethernet hálózatban? Mi mást lehetne még módosítani?

Megjegyzések a fejezethez

Law és Kelton monográfiája [172] jó áttekintést ad a hálózatoki rendszerekről.

A hálózatok osztályozásával kapcsolatban két, magyarul is megjelent monográfiát ajánlunk, melyek szerzői Sima, Fountain és Kacsuk [269],

illetve Tanenbaum [293].

A valószínűségszámítás alapjaival kapcsolatban Prékopa András [242] és Rényi Alfréd [249] könyveit ajánljuk. A leggyakoribb statisztikai eloszlásokat Banks könyve [20] alapján foglaltuk össze. A sűrűségfüggvények ábrázolására használt COMNET szimulációs modellezési eszközt ismertetése megtalálható a CACI két kiadványában [41, 141]

A szimuláció matematikai hátterével kapcsolatban Kátai Imre jegyzetét [162], jegyzetét [162], a sorbanállás elméletével kapcsolatban pedig Kleinrock könyvét [156] és Sztrik János hálózatról letölthető tananyagát [288] ajánljuk.

A csatornakapacitás definíciója megtalálható például az Interneten is elérhető szótárakban [129, 311]. Az információ- és kódelmélettel kapcsolatos részletek megtalálhatók például Jones és Jones könyvében [141].

A hosszú távú függőséggel foglalkoznak például Taqqu és társai [179, 296].

A hálózatmodellezésben előforduló leggyakoribb eloszlások becsléseit leíró 17.1. ábra Banks, Carson és Nelson könyvéből [20] származik.

Az OPNET szoftver és annak dokumentációja letölthető a [224]-ben megadott címről. Ez a dokumentáció részletesen tárgyalja a szimuláció egyes szakaszait is.

A forgalom ingadozásának hatását Gyires Tibor cikke [112] alapján elemeztük. A hálózati forgalommal kapcsolatos mérésekről számolnak be Leland és társai [178, 177], valamint Crovella és Bestavros [55].

Hálózatok önhasonlóságával foglalkoznak Erramilli, Narayan és Willinger [71], Willinger és társai [314], valamint Beran [23]. Mandelbrot [192], Paxson és Floyd [231], valamint Mandelbrot és van Ness [193] tanulmányozták a hosszú távon függő folyamatokat.

Forgalomirányítási modellek találhatók például a következő művekben: [5, 117, 134, 202, 213, 214, 228, 314].

Forgalmi adatokat tartalmaz [23, 68, 110, 231]. Az 17.22. ábra Listanti és Eramo cikkéből [183] származik.

A hosszú távú függőséget Addie, Zukerman és Neame [1], Duffield és O'Connell [67], valamint Erramilli, Narayan és Willinger [71] elemezték.

A *fekete doboz* modellezés kifejezést Willinger és Paxson [312] vezette be 1997-ben.

Az *Ockham borotvája* nevű elvről például Francis Heylighen honlapja [118] tartalmaz adatokat. A Snifferről a Network Associates cég honlapján [198] találhatók további részletek.

Willinger, Taqqu, Sherman és Wilson [313] egy szerkezeti modellt elemeznek. Crovella és Bestavros [55] a World Wide Web forgalmát elemezték.

Az erős ingadozásnak a hálózati torlódásokra gyakorolt hatásával foglalkozott például Neuts [213], valamint Molnár Sándor, Vidács Attila és Nilsson [206].

A Hurst-paraméter hatását Addie, Zukerman és Neame [1] elemezte.

A Benoit-csomag letölthető a hálózatról [306].

A Pareto-modellt Addie, Zukerman és Neame [1] vizsgálták.

18. Kvantumalapú algoritmusok

A kvantummechanikai alapokon nyugvó információelmélet olyan új lehetőségeknek nyit kaput, amelyek a hagyományos (klasszikus) világban elképzelhetetlenek, és éppen ezért hagyományos módszerekkel nem lehet őket leírni.

Ebben a fejezetben különböző, kvantummechanikai elveken alapuló algoritmusokat mutatunk be, amelyek lehetővé teszik, hogy gyorsan és hatékonyan oldjunk meg különböző, algoritmikusan nehéz feladatokat.

18.1. Bevezetés

A kvantum-számítástechnika alapját különféle kvantumfizikai effektusok képezik, melyek forradalmi megoldásokat kínálnak számos matematikai és mérnöki problémára, mint például a prímfaktorizáció, keresés rendezetlen adatbázisban, kulcsszétosztás és kódolás. A kvantumpárhuzamosság lehetővé teszi, hogy klasszikusan bonyolult problémákat oldjunk meg hatékonyan, a kvantum-összefonódás pedig olyan kommunikációs algoritmusok ígéretét hordozza, mint például a teleportáció (egy kvantumbitnyi információ továbbítása csupán összefonódás és két klasszikus bit segítségével), vagy a szupersűrű kódolás (két klasszikus bitnyi információ továbbítása egy kvantumbit felhasználásával). Bár a teljes egészében működő általános kvantumszámítógép, melyet David Deutsch 1985-ben leírt, még mindig a nem túl távoli jövő eszköze, számos részmegoldást már kísérletileg is bemutattak.

Ez a fejezet egy rövid áttekintést ad az információ kvantum alapú feldolgozásáról, és betekintést enged néhány kiválasztott algoritmus működésébe. Célunk elsősorban az, hogy bemutassuk a kvantuminformatika világának különlegességeit, így nem mindegyik algoritmust tárgyaljuk teljes mélységében. Aki részletesebb információra vágyik, az megtalálja ezt az [127] könyvben, amely az algoritmusokkal foglalkozik, illetve a [14] munkában, melynek fókuszában a kommunikáció áll. További irodalmi hivatkozásokat a fejezet végén talál az érdeklődő Olvasó.

A fejezet felépítése a következő: a 18.2. alfejezet rész tárgyalja a kvantuminformatika posztulátumait, és az összefonódás jelenségét. A 18.3. alfejezetben található az elemi kvantumkapuk leírása, melyekből kvantumáramköröket építhetünk. A kvantumpárhuzamosság jelenségét a 18.4.

alfejezetben tárgyaljuk, míg a 18.5. alfejezet rész a kvantum Fourier-transzformációval foglalkozik. A kvantum alapú keresésről a 18.6. alfejezetben írunk bővebben.

A könyvfejezetben a manapság elterjedt módszerekre a klasszikus jelzőt fogjuk használni (ezek információs alapegysége a klasszikus bit), a kvantummechanikán alapuló eljárásokat pedig kvantumosnak nevezzük majd (ilyen esetben az információ alapegységének a kvantumbitet tekintjük).

Gyakorlatok

18.1-1. Keressünk rá az interneten a legújabb, kvantumszámítógépekkel kapcsolatos hírekre, és hasonlítsuk össze őket a Deutsch által leírt kvantumszámítógéppel. Létezik-e jelenleg valóban működő kvantumszámítógép?

18.2. Rövid bevezetés a kvantumalapú informatikába

18.2.1. Kvantuminformatikai posztulátumok

A kvantummechanika matematikai leírásához először szükségünk van néhány alapfeltevésre, melyeket igaznak fogadunk el, bármilyen különösnek is tűnjenek. Ezek a kvantummechanikai ***posztulátumok.*** Ezeket az alapfeltevéseket többféle, egymással ekvivalens módon is meg lehet fogalmazni, mi itt most négy állításba sűrítjük őket.

1. **posztulátum:** *Minden zárt fizikai rendszer állapota leírható egy úgynevezett* $\mathbf{v}$ *állapotvektorral, mely a komplex számtest felett definiált* V ***Hilbert-tér*** *egy egységnyi vektora.*

Például egy fizikai rendszerhez, melynek 2, egymást kizáró, és egymástól jól megkülönböztethető állapota van, hozzárendelhetünk egy 2-dimenziós állapotvektort. A lineáris algebrában ezt a vektort reprezentálhatjuk egy oszlopvektorral: $\mathbf{v} = [a, b]^T = a\mathbf{0} + b\mathbf{1}$, ahol $\mathbf{0} = [1, 0]^T$ és $\mathbf{1} = [0, 1]^T$ a V Hilbert-tér standard ortonormált bázisa, míg $a, b \in \mathbb{C}$. A $\mathbf{v}$ vektor egységnyi hosszának megtartásához a koordináták között a következő összefüggésnek kell fennállnia: $|a|^2 + |b|^2 = 1$. Az állapotvektor koordinátáit ***valószínűségi amplitúdóknak*** nevezzünk, míg az ilyen kétdimenziós állapotvektorok neve a kvantuminformatikában ***kvantumbit,*** vagy angol elnevezéssel ***qubit.***

2. **posztulátum:** *Minden zárt rendszer időevolúcióját le lehet írni az állapotvektoron ható unitér transzformációk segítségével, úgy, hogy a végállapot csak a kezdeti állapottól és az evolúció időtartamától függ.*

A második posztulátumot matematikailag a következőképpen fogalmazhatjuk meg: $\mathbf{v}'(t_2) = U(t_1, t_2)\mathbf{v}(t_1)$ ahol mind $\mathbf{v}$ kezdeti, mind $\mathbf{v}'$ végál-

lapot a Hilbert-tér vektora $\mathbf{v}, \mathbf{v}' \in V$.

A fenti definíció diszkrét időpontok közötti evolúciót ír le, ami az informatika számára a legpraktikusabb. A valóságban ez nem más, mint az időfüggő ***Schrodinger-egyenlet@Schrödinger-egyenlet*** egy egyszerűsített változata.

A lineáris algebrai leírásban az U unitér operátort reprezentálhatjuk egy $\mathbf{U}$ négyzetes mátrixszal, melynek U_{ij} eleme írja le, hogy mi lenne a végállapothoz tartozó vektor **i**-edik komponense, ha a kezdeti állapot a **j**-edik bázisvektor lenne.

3. **posztulátum:** *Minden kvantummérés leírható **mérési operátorok** egy $\{M_m\}$ halmazával, ahol az m index jelöli a mérés egy lehetséges kimenetelét. Annak valószínűsége, hogy a mérés előtt $\mathbf{v}$ állapotvektorral jellemezhető rendszert a mérés után az m-edik állapotban találjuk:*

$$P(m \mid \mathbf{v}) = \mathbf{v}^\dagger M_m^\dagger M_m \mathbf{v} \ ,$$

ahol $\dagger$ az adjungálás műveletét jelöli. A mérés során a rendszer állapota megváltozik, a megváltozott állapot a mérés eredményének függvényében:

$$\mathbf{v}' = \frac{M_m \mathbf{v}}{\sqrt{\mathbf{v}^\dagger M_m^\dagger M_m \mathbf{v}}}.$$

Az m mérési állapotokat általában úgy választjuk, hogy teljes eseményrendszert alkossanak, így a klasszikus valószínűségelmélet szerint a valószínűségek összege 1:

$$\sum_m P(m \mid \mathbf{v}) = \sum_m \mathbf{v}^\dagger M_m^\dagger M_m \mathbf{v} \equiv 1 \ ,$$

Ennek tetszőleges $\mathbf{v}$ állapotra igaznak kell lennie, ezért a mérési operátoroknak tejesíteniük kell az úgynevezett teljességi relációt:

$$\sum_m M_m^\dagger M_m \equiv I \ .$$

A klasszikus világ és kvantumvilág között mérések teremtenek kapcsolatot. Fontos megjegyezni, hogy a kvantummechanikában a mérés megváltoztatja a rendszer állapotát, ezért az informatikai alkalmazásokban nagyon fontos lesz, hogy mikor és mit mérünk. Általános esetben ez a változás nem megfordítható (azaz nem reverzibilis), így a mérés kimeneteléhez tartozó operátor sem unitér.

A lineáris algebrai leírásban a mérési operátorokat négyzetes mátrixokkal reprezentálhatjuk, azonban erre nincs mindig szükség. Egy példa erre, ha mérésünkkel azt próbáljuk eldönteni, mi az esélye annak, hogy a rendszer állapota azonos egy bázisvektorral abból az ortonormált bázisból, amiben a vektorunkat felírtuk. Ekkor a valószínűség nem más, mint a bázisvektorhoz tartozó valószínűségi amplitúdó abszolút érték négyzete, a mérés után pedig a végállapot a mért bázisvektor lesz.

A fentiekből az is nyilvánvaló, hogy miért kell az első posztulátum szerint egységvektorokkal dolgoznunk: ha a vektor komponensei a valószínűségi amplitúdók, melyek a mérési valószínűségekkel állnak szoros kapcsolatban, leírásunkban pedig a teljes eseményrendszert le akarjuk fedni, akkor a megfelelő valószínűségek összegének pontosan egyet kell adnia.

4. **posztulátum:** *Egy összetett rendszer állapotvektora leírható a W Hilbert-tér egy egységnyi hosszúságú* $\mathbf{w}$ *elemeként. Ez a W tér nem más, mint a részrendszerek állapotterének tenzorszorzata: $W = V \otimes Y$. Ha definiálható az alrendszerek $\mathbf{v} \in V$ és $\mathbf{y} \in Y$ állapota, akkor a teljes rendszer állapotvektora $\mathbf{w} = \mathbf{v} \otimes \mathbf{y}$, ahol $\otimes$ jelöli a tenzorszorzást.*

A negyedik posztulátumból következik, hogy a kvantummechanikában a teljes rendszer több a részek összegénél. Ez matematikailag azt jelenti, hogy a részrendszerek állapotvektorát nem mindig lehet definiálni, a tejes, zárt rendszerét azonban igen. Ez a különös jelenség ad lehetőséget az összefonódásra, mellyel a későbbi fejezetekben részletesen is foglalkozunk.

18.2.2. Kvantumbit és kvantumregiszter

A fent bemutatott posztulátumok távol állnak a hétköznapi számítástudománytól és algoritmusoktól. Éppen ezért bevezetünk egy magasabb szintű absztrakciót, mely lehetővé teszi, hogy bitekről, regiszterekről, kapukról, áramkörökről és kommunikációs csatornákról beszéljünk, anélkül, hogy tisztáznánk, pontosan milyen fizikai részecskék és folyamatok állnak a háttérben.

A klasszikus információ legkisebb alapegysége a ***bit***. Ennek értéke lehet 0 vagy 1, de a kettő közül egyszerre mindig csak egy és pontosan egy értéket vesz fel. A kvantuminformatikában a helyzet egészen más. Az első posztulátumnak megfelelően a legegyszerűbb, nem triviális kvantumrendszer jellemezhető egy 2-dimenziós Hilbert-tér egy komplex együtthatós egységvektorával. Ezt a kvantumbit. A Dirac-féle jelöléssel a $\mathbf{v}$ oszlopvektort a következőképpen írjuk: $|v\rangle$, szóban pedig „ket v” vektornak olvassuk.

A kvantumbit értékét egy ortonormált bázis két vektorának segítségével írhatjuk fel, ezek a $|0\rangle$ és $|1\rangle$ vektorok. (A félreértések elkerülésére hangsúlyoz-

zuk, hogy a zárójelben álló számok mindig sorszámok, nem pedig nullvektorra vagy egységvektorra utalnak. A kvantuminformatikában bevett konvenció szerint a bázisvektorokat sorszámozzuk, ez a számozás 0-tól indul, és bináris.) A bázis ismeretében egy tetszőleges $|\varphi\rangle$ kvantumbit a következőképpen írható fel:

$$|\varphi\rangle = a|0\rangle + b|1\rangle = a\begin{bmatrix} 1 \\ 0 \end{bmatrix} + b\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} a \\ b \end{bmatrix} . \tag{18.1}$$

Az a és b együtthatók komplex számok ($a, b \in \mathbb{C}$), és ***valószínűségi amplitúdónak*** nevezzük őket. A harmadik posztulátum értelmében annak valószínűsége, hogy $|0\rangle$ állapotban találjuk a rendszert, az $|a|^2$, a $|1\rangle$ állapot valószínűsége pedig $|b|^2$. Éppen ezért a valószínűségi amplitúdók nem vehetnek fel tetszőleges értéket, hiszen teljesíteniük kell a következő relációt: $|a|^2 + |b|^2 = 1$.

Azokra az állapotokra, melyek egynél több bázisvektor súlyozott lineáris kombinációját tartalmazzák, a ***szuperpozíció*** kifejezést használjuk.

A sorvektorokat $\langle\varphi|$-ként szokás jelölni, és „bra φ"-ként szokás kiolvasni. A relációt az oszlop és sorvektorok között az adjungálás művelete teremti meg: $|\varphi\rangle = (\langle\varphi|)^\dagger$.

A jelölés előnye, hogy a belső (skaláris) szorzat természetesen adódik, $|\varphi\rangle$ és $|\psi\rangle$ skalárszorzata $\langle\varphi|\psi\rangle$. A jelölés neve is innen eredeztethető, a bra és ket szavak az angol bracket, vagyis zárójel kifejezés első és második felének torzított alakjából származnak.

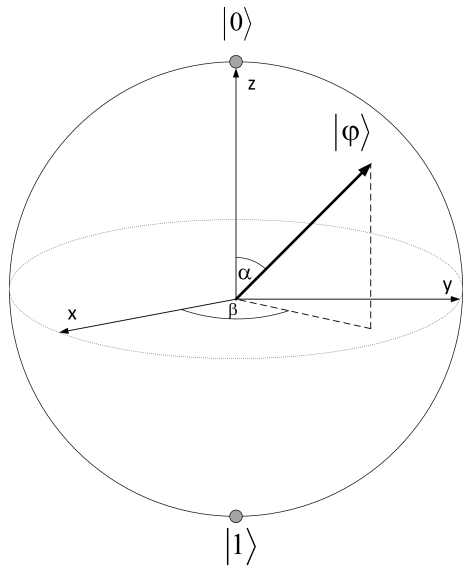
Ezen a ponton említjük meg az úgynevezett ***No-cloning tételt*** mely kimondja, hogy a kvantuminformatikában csak olyan állapotokat lehet másolni, amiket vagy ismerünk, vagy egy ismert, ortogonális állapotokból álló halmaz elemei. A második posztulátum szerint minden kvantumos művelet unitér, ebből az állításból pedig levezethető, hogy tetszőleges, ismeretlen állapotot nem lehet másolni.

A No-cloning tétel tulajdonképpen nem más, mint a kvantummechanika sztochasztikus természetének egy megfogalmazása, hiszen ha tudnánk másolni az állapotot, elég lenne nagyszámú másolaton megismételni a mérést ahhoz, hogy tetszőleges pontossággal megismerjük az eredményt. A valóságban ez nem lehetséges, a később tárgyalt algoritmusoknál pedig látni fogjuk, hogy milyen nagy gonddal kell megterveznünk, mit is fogunk mérni, hiszen másolni nem tudunk, a harmadik posztulátum pedig kimondja, hogy a mérés megváltoztatja az állapotot, így csak egyetlen lehetőségünk van a mérés elvégzésére.

A továbbiakban egy kvantumbit egy geometriai reprezentációjával fogunk foglalkozni, ehhez írjuk át a (18.1)-es vektorunkat a következő alakba:

$$|\varphi\rangle = e^{j\gamma}\left[\cos\left(\frac{\alpha}{2}\right)|0\rangle + e^{j\beta}\sin\left(\frac{\alpha}{2}\right)|1\rangle\right], \tag{18.2}$$

ahol $\alpha, \beta, \gamma \in \mathbb{R}$ és $e^{j\gamma}$ az úgynevezett ***gobális fázis***. Mivel a globális fázis



18.1. ábra. Kvantumbit vizuális megjelenítése a Bloch-gömbön.

abszolút értéke 1, ezért nem befolyásolja a mérési statisztikát. Éppen ezért a globális fázist legtöbbször elhanyagoljuk (nem tüntetjük fel a levezetésekben), vagy ha egy levezetés úgy egyszerűbb, minden további nélkül megváltoztathatjuk a γ fázisszöget.

Míg a (18.1)-es írásmód egy 2-dimenziós Descartes-koordinátarendszerben írható fel, melynek minden koordinátáját a komplex számsík egy vektorával reprezentálhatjuk, mely összesen négy geometriai tengelyt jelent, addig a (18.2)-es képlet a globális fázisfaktor elhanyagolásával egy 3-dimenziós vektort ír le gömbi koordinátákban. A két valós szög szerepét α és β játssza, míg a vektor hosszát az első posztulátum értelmében egységnyinek vesszük. Ez a fajta vizualizáció Felix Bloch nevéhez köthető, éppen ezért a 18.1 ábrán látható megjelenítést Bloch-gömbnek nevezzük. A gömbi koordinátákat a következőképpen lehet Descartes-koordinátákra átírni:

$$|\varphi\rangle = [x, y, z]^T = [\cos(\beta)\sin(\alpha), \sin(\beta)\sin(\alpha), \cos(\alpha)]^T. \tag{18.3}$$

Fontos kiemelnünk, hogy a Bloch-gömb csak egy globális fázisfaktor erejéig ad egyértelmű leírást, a gömb minden egyes pontjához végtelen sok γ fázisszög tartozhat.

A klasszikus információelmélethez hasonlóan n darab kvantumbitet együtt egy n hosszúságú ***kvantumregiszternek*** hívunk. Ez tartalmazhatja az

$N = 2^n$ dimenziós tér bármelyik bázisvektorát, vagy a bázisvektorok tetszőleges szuperpozícióját. Természetesen az első posztulátum értelmében a regiszter állapotvektorának egységnyi hosszúnak kell lennie, a negyedik posztulátum szerint pedig, ha ismerjük az egyes kvantumbitek állapotvektorait, akkor a teljes regiszter állapotvektora ezek tenzorszorzataként adódik: $|\varphi\rangle = |qubit_{N-1}\rangle \otimes |qubit_{N-2}\rangle \otimes ... \otimes |qubit_1\rangle \otimes |qubit_0\rangle$.

Nézzünk meg egy egyszerű példát egy két kvantumbitből álló regiszterre:

$$|\varphi_1\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}}, |\varphi_2\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}}.$$

Amikor a két kvantumbitet egyetlen regiszterként kezeljük, a következő 4-dimenziós $|\varphi\rangle$ vektort kapjuk eredményül:

$$\begin{aligned} |\varphi\rangle &\equiv |\varphi_1\rangle|\varphi_2\rangle \equiv |\varphi_1, \varphi_2\rangle \equiv |\varphi_1\varphi_2\rangle \\ &= \frac{|0\rangle \otimes |0\rangle + |1\rangle \otimes |0\rangle + |0\rangle \otimes |1\rangle + |1\rangle \otimes |1\rangle}{2} \\ &= \frac{|00\rangle + |01\rangle + |10\rangle + |11\rangle}{2} . \end{aligned} \tag{18.4}$$

A példában látható állapotvektorunk mind a négy bázisvektort tartalmazza. A bázisok sorszáma – 00, 01, 10 és 11 – nem más, mint a klasszikus kétbites regiszter lehetséges tartalma. Kvantumos esetben azonban az összes lehetőség egyszerre van jelen, annak valószínűségét pedig, hogy egy mérés során ezek közül melyiket milyen eséllyel találjuk a regiszterben, a harmadik posztulátum alapján dönthetjük el.

Azokat az állapotokat, melyeket alacsonyabb dimenziós állapotok tenzorszorzataként állíthatunk elő, ***szorzatállapotoknak*** nevezzük.

Általánosítva egy n darab kvantumbitből álló regiszter állapota a következőképpen írható le:

$$|\varphi\rangle = \sum_{i=0}^{2^n-1} \varphi_i |i\rangle ,$$

ahol φ_i az i-edik $|i\rangle$ bázisvektor valószínűségi amplitúdója.

18.2.3. Összefonódás

Felvértezve a kvantumregiszterekkel kapcsolatos tudásunkkal, bontsuk fel a $|\varphi\rangle = a|00\rangle + b|11\rangle$ kvantumregisztert az alkotó kvantumbitek szorzatára. Ha a felbontás nem sikerülne, ne keseredjünk el, ez ugyanis nem szorzatállapot, így nem tudjuk egyszerű kvantumbitek tenzorszorzatára bontani.

De nézzük meg közelebbről ebben a speciális regiszterben rejlő

lehetőségeket. Ha mérést hajtunk végre az egyik kvantumbiten, akkor az véletlenszerűen vagy 0, vagy 1 állapotba fog kerülni, a megfelelő $|a|^2$ és $|b|^2$ valószínűséggel. Azonban ettől a pillanattól kezdve a másik kvantumbiten végrehajtott mérés eredménye már nem lesz véletlenszerű, hiszen az első mérés eredményének ismeretében tökéletesen meg tudjuk jósolni azt. Ha az első kvantumbit értéke a mérés szerint 0, akkor a második is 0 lesz, ha 1, akkor a második mérés eredménye is 1 lesz. Gondosan tervezett fizikai kísérletek megmutatták, hogy ez a jelenség akkor is fennáll, ha jelentős távolságban van egymástól a két kvantumbit. Ebben az az igazán zavarba ejtő, hogy jelenlegi ismereteink szerint a mérés végeredménye csak a mérés pillanatában dől el, ami látszólag azt jelenti, hogy a két részrendszert összekapcsoló hatás terjedéséhez nincs szükség időre. Ugyanakkor épp a véletlenszerűség biztosítja, hogy fénynél gyorsabb információterjedés ne valósulhasson meg.

Azokat a kvantumbiteket/kvantumregisztereket, amiket ilyen különös effektus köt egymáshoz, ***összefonódott állapotoknak*** nevezzük.

18.2.4. Bell-állapotok

Végtelen számú összefonódott állapotot lehet alkotni pusztán a és b valószínűségi amplitúdók értékének változtatásával, melyek mind a $|\varphi\rangle = a|00\rangle + b|11\rangle$ formulát követik, ugyanakkor a $|01\rangle$ és $|10\rangle$ bázisvektorok is ugyanúgy alkalmasak összefont állapotok létrehozására. Van azonban négy kitüntetett összefonódott pár, melyeket együtt ***Bell-állapotoknak*** vagy ***EPR-pároknak*** nevezünk:

$$|\beta_{00}\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}} ,$$

$$|\beta_{01}\rangle = \frac{|01\rangle + |10\rangle}{\sqrt{2}} ,$$

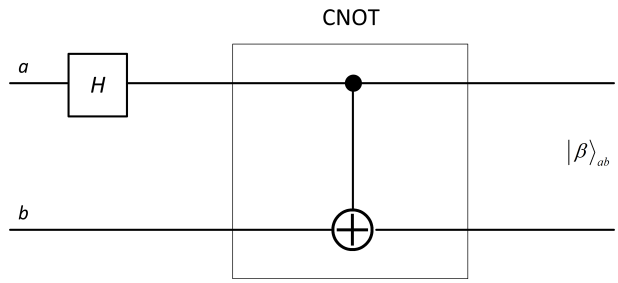
$$|\beta_{10}\rangle = \frac{|00\rangle - |11\rangle}{\sqrt{2}} ,$$

$$|\beta_{11}\rangle = \frac{|01\rangle - |10\rangle}{\sqrt{2}} .$$

A Bell-állapotokat a 18.2. ábrán látható kvantumáramkör segítségével lehet előállítani.

A Bell-állapotok ortonormált rendszert alkotnak, ami azt jelenti, hogy egyértelműen meg tudjuk különböztetni őket.

Az EPR-párokat természetesen tovább lehet általánosítani összefont bithármasokra, mint például a $(|000\rangle + |111\rangle)/\sqrt{2}$, melyekre az irodalomban



18.2. ábra. Bell-állapotok előállítására szolgáló áramkör.

GHZ vagy *Greenberg-Horne-Zeilinger állapotokként* szokás hivatkozni.

Gyakorlatok

18.2-1. A belső szorzat definíciójának felhasználásával bizonyítsuk be, hogy bármely két Bell-állapot merőleges.

18.2-2. Vegyük a következő állapotot $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$. Meg akarjuk határozni, hogy ehhez a Bloch-gömb melyik vektora tartozik. Ha az állapotvektort a (18.2) képlet szerint írjuk fel, mi lesz α, β és γ értéke?

18.3. Elemi kvantumkapuk

18.3.1. Pauli-kapuk

Ebben a részben az elemi kvantumkapukat járjuk körbe, és megnézzük, milyen hatást fejtenek ki egy kvantumbitre. A második posztulátum szerint a rendszer időbeli fejlődését modellezhetjükunitér operátorok segítségével.

Az általánosság kedvéért a kapuk hatását egy tetszőleges 1 kvantumbites állapoton mutatjuk be $|\varphi\rangle = a|0\rangle + b|1\rangle$, és a végállapotra úgy hivatkozunk, mint $|\psi\rangle = U|\varphi\rangle$. Lássuk először a bit-flip, vagy más néven ***Pauli-X kaput.***

$$|\psi\rangle = X|\varphi\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} \begin{bmatrix} b \\ a \end{bmatrix} = b|0\rangle + a|1\rangle.$$

Ahogy a leírásból is látható, a bit-flip kapu megcseréli a két bázisvektorhoz tartozó valószínűségi amplitúdókat.

A ***Pauli-Z kapu,*** vagy más szóval fázis-flip kapu megváltoztatja a kezdeti

állapot egyik bázisvektorához tartozó valószínűségi amplitúdó előjelét.

$$|\psi\rangle = Z|\varphi\rangle = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} \begin{bmatrix} a \\ -b \end{bmatrix} = a|0\rangle - b|1\rangle.$$

A Pauli-kapuk listájának teljessé tételéhez definiálhatjuk a ***Pauli-Y kaput*** úgy mint:

$$|\psi\rangle = Y|\varphi\rangle = \begin{bmatrix} 0 & -j \\ j & 0 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} \begin{bmatrix} -jb \\ ja \end{bmatrix} = -jb|0\rangle + ja|1\rangle \ .$$

A Pauli-kapuk rejtélyes X, Y és Z elnevezését rögtön megértjük, mihelyt ránézünk a Bloch-gömbre, ezekkel ugyanis az x, y és z tengelyek körüli forgatását valósíthatunk meg. Például az x tengely körül α szöggel történő forgatást kifejezhetjük úgy, mint:

$$e^{-j\frac{\alpha}{2}X} = \cos\left(\frac{\alpha}{2}\right) I - j \sin\left(\frac{\alpha}{2}\right) X.$$

A *fázisforgató kapu*, vagy röviden *fáziskapu* a következő műveletet hajtja végre:

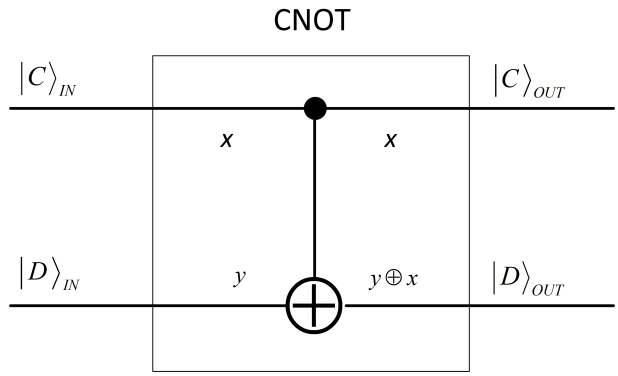
$$|\psi\rangle = P(\alpha)|\varphi\rangle = \begin{bmatrix} 1 & 0 \\ 0 & e^{j\alpha} \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} \begin{bmatrix} a \\ e^{j\alpha}b \end{bmatrix} = a|0\rangle + e^{j\alpha}b|1\rangle \ .$$

18.3.2. Hadamard-kapu

A *Hadamard-kapu* a következő állapotot állítja elő:

$$|\psi\rangle = H|\varphi\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} \begin{bmatrix} \frac{a+b}{\sqrt{2}} \\ \frac{a-b}{\sqrt{2}} \end{bmatrix} = \frac{a+b}{\sqrt{2}}|0\rangle + \frac{a-b}{\sqrt{2}}|1\rangle \ . \tag{18.5}$$

Könnyű belátni, hogy mind a Pauli-kapuknak, mind a Hadamard-kapunak van néhány különleges matematikai tulajdonsága. Mátrixuk nem csak unitér ($HH^\dagger = H^\dagger H = I$), hanem hermitikus is ($H^\dagger = H$), amiből következik, hogy önmaga inverze ($HH = I$). Ráadásul a Pauli- és Hadamard-kapuknak



18.3. ábra. Vezérelt NEM kapu (CNOT-kapu).

különleges kapcsolata van: $HXH = Z$, $HYH = -Y$ és $HZH = X$.

Klasszikus bementek esetén a Hadamard-kapu kimenete:

$$H|0\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}},$$
$$H|1\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}}. \tag{18.6}$$

Egy Hadamard-kapu a klasszikus bemenethez az összes bázisvektor egyenletes eloszlású szuperpozícióját rendeli kimenetként. Az, hogy mi volt a klasszikus bemenet, csak a valószínűségi amplitúdók előjeléből olvasható le. Egy n-kvantumbitből álló $|\varphi\rangle = |00\ldots0\rangle$ regiszter esetén a kapu kimenete úgy írható le, mint:

$$|\psi\rangle = H^{\otimes n}|\varphi\rangle = \frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} |i\rangle \ , \tag{18.7}$$

ahol $H^{\otimes n}$ az összes kvantumbitre ható Hadamard-transzformációt jelöli.

18.3.3. A CNOT kapu

A legfontosabb két qubites kapu az úgynevezett ***vezérelt NEM kapu*** vagy ***CNOT kapu.*** Áramköri jelölése a 18.3. ábrán látható, a hozzá tartozó igazságtáblát pedig a 18.1. táblázatban foglaltuk össze.

A CNOT kapu mátrixa az igazságtábla alapján könnyen felírható:

$$\mathbf{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}. \tag{18.8}$$

IN		OUT	
x	y	x	$y \oplus x$
0	0	0	$0 \oplus 0 = 0$
0	1	0	$1 \oplus 0 = 1$
1	0	1	$0 \oplus 1 = 1$
1	1	1	$1 \oplus 1 = 0$

18.1. táblázat. A CNOT kapu igazságtáblája.

A transzformáció unitér, hiszen a mátrix sor- és oszlopvektorai kölcsönösen ortogonálisak egymásra.

A kapu legfontosabb tulajdonsága, hogy összefont állapotokat lehet vele előállítani, úgy, ahogy azt a 18.2 ábrán már láthattuk.

Gyakorlatok

18.3-1. Határozzuk meg a 2-kvantumbitre ható Hadamard-kapu ($H^{\otimes 2} = H \otimes H$) mátrixát.

18.3-2. Tegyük fel, hogy a kvantumbitünk a $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$ állapotban van. Ha végrehajtunk rajta egy X transzformációt, hogyan változik meg vektora a Bloch-gömbön? (A 18.2-es képlet használatával határozzuk meg α, β, γ értékeit.)

18.4. Kvantum-párhuzamosság

Számos kvantum alapú algoritmus kihasználja az úgynevezett ***kvantum-párhuzamosság*** jelenségét. Ennek alapja, hogy ha egy kvantumáramkör bemenetére az összes lehetséges bemenet szuperpozícióját adjuk, akkor a kimenet az összes lehetséges kimenet szuperpozíciója lesz. Sajnos az összes lehetséges kimenetet egyforma valószínűséggel kapjuk, éppen ezért a mérés előtt szükség van egy úgynevezett amplitúdóerősítési lépésre, amely megnöveli a számunkra hasznos kimenet valószínűségi amplitúdóját. A jelenség szemléltetésére három algoritmust vizsgálunk meg ebben a fejezetben, melyek alapja a Fourier-mintavételezés,l és az f-vezérelt kapu.

18.4.1. Fourier-mintavételezés

Amint azt a 18.7. részben láttuk, a $|\varphi\rangle = |00\ldots0\rangle$ állapot Hadamard-transzformált alakja:

$$H^{\otimes n}|\varphi\rangle = \frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} |i\rangle \ ,$$

ahol $H^{\otimes n}$ az n darab kvantumbitre ható Hadamard-kaput jelöli. Egy tetszőleges $|x\rangle$ bázisvektor esetén a transzformáció végeredménye:

$$H^{\otimes n}|x\rangle = \frac{1}{\sqrt{2^n}} \sum_{y\in\{0,1\}^n} (-1)^{x\cdot y}|y\rangle \ , \tag{18.9}$$

ahol $x\cdot y$ nem más, mint az x és y bináris sorszámok skalárszorzata (vagy más szóval bitenkénti szorzatának összege modulo 2).

Fourier-mintavételezésnekl nevezzük azt az eljárást, amikor egy n kvantumbitből álló $|\phi\rangle = \sum_{x\in\{0,1\}^n} \alpha|x\rangle$ állapoton $H^{\otimes n}|\phi\rangle$ Hadamard-transzformációt alkalmazunk, majd a végeredményen – amely nem más mint $\sum_{y\in\{0,1\}^n} \beta_y|y\rangle$ – egy mérést hajtunk végre. A mérés után $|\beta_y|^2$ valószínűséggel találjuk a rendszerünket y állapotban.

18.4.2. f-vezérelt kapu

A kvantumpárhuzamosság fontos építőeleme az U_f függvény által vezérelt transzformáció, ahol f egy bináris függvényt jelöl. Az ezt megvalósító logikai kapu áramköri rajza a 18.4. ábrán látható. A transzformáció egyenlete:

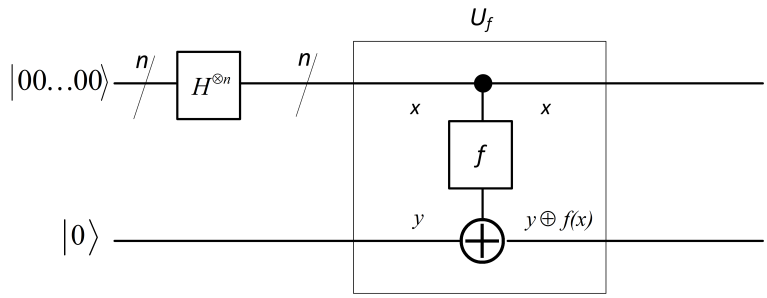
$$U_f : |x\rangle_N|y\rangle \to |x\rangle_N|y \oplus f(x)\rangle \ , \tag{18.10}$$

ahol $x \in \{0,1\}^n$. Amennyiben az f függvény kimenete egyetlen bit, és a vezérelt kvantumbit $|0\rangle$ állapotban van, a kapu kimenete a következő lesz:

$$\begin{aligned} U_f \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} |x\rangle|0\rangle &= \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} |x\rangle|0 \oplus f(x)\rangle \\ = \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} |x\rangle|f(x)\rangle &= \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle|f(x)\rangle \ . \end{aligned} \tag{18.11}$$

Ez egy olyan különleges szuperpozíció, amely az f függvény összes lehetséges kimeneti értékét egyszerre tartalmazza, úgy, hogy az első n kvantumbit a függvény bemeneti értékéhez tartozik, míg az utolsó kvantumbit értéke a kimeneti függvényérték lesz. Ez azt jelenti, hogy elméletileg egyetlen lépésben kiszámítottuk f összes lehetséges függvényértékét, függetlenül attól, hogy hány kvantumbites rendszerrel dolgozunk.

Sajnálatos módon a gyakorlatban, amikor megpróbáljuk kiolvasni a végeredményt, akkor a mérés eredménye a szuperpozíciónak csak egyetlen, véletlenszerű, egyenletes eloszlással választott eleme lesz. Éppen ezért sok esetben még szükség van egy amplitúdó-erősítési lépésre is, melyben az egyenletes eloszlást megváltoztatva megnöveljük annak a valószínűségét, hogy az általunk keresett válasz legyen a mérés eredménye. Nem mindig kézenfekvő, hogyan tudjuk ezt elérni.



18.4. ábra. f-vezérelt CNOT kapu, amely egy N-dimenziós vezérlő bemenettel kiszámítja f értékeit.

18.4.3. Bernstein-Vazirani algoritmus

Az alábbiakban ismertetett probléma Bernstein és Vazirani nevéhez köthető. Adott egy f függvény, amely minden, n bit hosszú $x \in \{0,1\}^n$ bináris vektorhoz egyetlen bit kimenetet rendel, azaz $f(x) : \{0,1\}^n \to \{0,1\}^1$. Tudjuk azt is erről a függvényről, hogy $f(x) = u \cdot x$, és arra vagyunk kíváncsiak, hogy mi ez a bizonyos ismeretlen $u \in \{0,1\}^n$. Mivel f egy bool-algebrai függvény, így interpretálható úgy, mint: $f(x) = u \cdot x = u_{n-1} \cdot x_{n-1} \oplus u_{n-2} \cdot x_{n-2} \oplus \cdots \oplus u_0 \cdot x_0$. Hogyan találjuk meg u-t?

A kapcsolódó kvantumáramkör blokkvázlata a 18.5-ös ábrán látható. A rendszer kezdeti állapota:

$$|\varphi_0\rangle = |0\rangle_n |1\rangle \; . \tag{18.12}$$

A Hadamard-transzformáció eredménye:

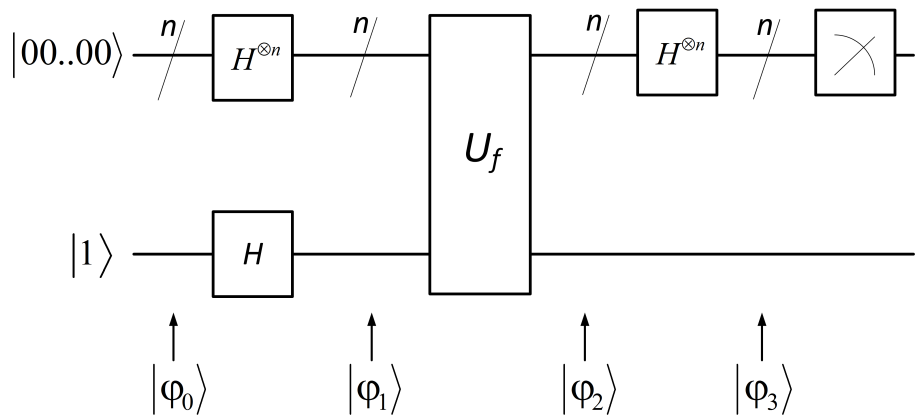
$$|\varphi_1\rangle \;\; = \;\; H^{\otimes(n+1)}|\varphi_0\rangle = \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} |x\rangle \otimes \frac{|0\rangle - |1\rangle}{\sqrt{2}}. \tag{18.13}$$

Az f-vezérelt kapu hatása:

$$|\varphi_2\rangle \;\; = \;\; U_f|\varphi_1\rangle = \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} (-1)^{f(x)}|x\rangle \otimes \frac{|0\rangle - |1\rangle}{\sqrt{2}}. \tag{18.14}$$

Felhasználva, hogy $f(x) = u \cdot x$,

$$\frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} (-1)^{f(x)}|x\rangle = \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} (-1)^{u\cdot x}|x\rangle, \tag{18.15}$$



18.5. ábra. A Bernstein-Vazirani algormust megvalósító kvantumáramkör.

Ahhoz, hogy megkapjuk a keresett, bináris u vektort, Fourier-mintavételeznünkl kell a $|\varphi_2\rangle$ állapotvektort. A felső vezérlő biteken egy n-kvantumbites Hadamard-transzformációt elvégezve (hatásához lásd a (18.9) képletet), továbbá a legalsó kvantumbiten identitás transzformációt végrehajtva az eredmény:

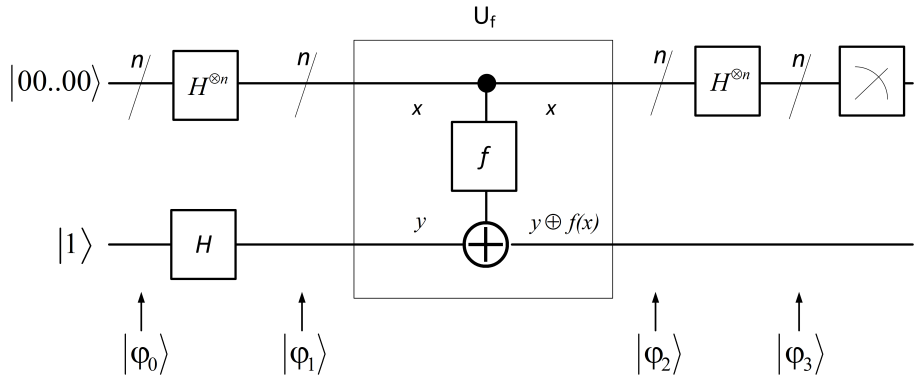
$$\begin{aligned} |\varphi_3\rangle &= \left(H^{\otimes n} \otimes I\right) |\varphi_2\rangle \\ &= \frac{1}{2^n} \sum_{x\in\{0,1\}^n} \sum_{y\in\{0,1\}^n} (-1)^{f(x)+x\cdot y} |y\rangle \otimes \frac{|0\rangle - |1\rangle}{\sqrt{2}} \ . \end{aligned} \tag{18.16}$$

A (18.15) összefüggés felhasználásával a szorzótényezőt az alábbi módon alakíthatjuk át:

$$\sum_{x=0}^{2^n-1} (-1)^{u\cdot x} (-1)^{y\cdot x} = \prod_{j=1}^{n} \sum_{x_j=0}^{1} (-1)^{(u_j+y_j)x_j}. \tag{18.17}$$

Vizsgáljuk meg azokat az eseteket, amikor y és u nem egyezik meg. Másképpen megfogalmazva azokat az eseteket keressük, amikor legalább egy bitértékük nem egyezik meg, vagyis $y_j \neq u_j$. Ekkor (18.17) jobboldalán a kitevő értéke nem független x_j értékétől. Ebből következik, hogy az x_j két lehetséges értékére való összegzés eredménye 0, hiszen az ellentétes előjelek éppen kiejtik egymást. Ennek fényében látható, hogy a teljes $|\varphi_3\rangle$ szuperpozícióból csak azok a tagok maradnak, ahol $y = u$:

$$|\varphi_3\rangle = |u\rangle_n \otimes \frac{|0\rangle - |1\rangle}{\sqrt{2}} \ . \tag{18.18}$$



18.6. ábra. A Deutsch-Jozsa algoritmust megvalósító kvantumáramkör.

A vezérlőbiteken végrehajtott mérés így pontosan a keresett u vektort fogja adni.

18.4.4. Deutsch-Jozsa algoritmus

Egy másik, kvantumpárhuzamosságon alapuló algoritmus az úgynevezett Deutsch-Jozsa algoritmus. Legyen egy n hosszúságú, $x \in \{0,1\}^n$ bináris vektorunk és egy $f(x) : \{0,1\}^n \to \{0,1\}^1$ függvényünk. Tegyük fel, hogy függvényünk vagy konstans (minden bemeneti értékhez azonos kimeneti értéket rendel), vagy kiegyensúlyozott (a lehetséges bemeneti értékekhez fele-fele arányban rendel 0-át és 1-et). Hogyan tudjuk megállapítani, hogy a függvényünk a kettő közül melyik csoportba tartozik?

A kapcsolódó kvantumáramkör a 18.6. ábrán látható. A rendszer kezdeti állapota legyen

$$|\varphi_0\rangle = |0\rangle_n |1\rangle \ . \tag{18.19}$$

Az összes kvantumbitre ható Hadamard-transzformáció hatása:

$$\begin{aligned} |\varphi_1\rangle &= H^{\otimes(n+1)} |\varphi_0\rangle = \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} |x\rangle \otimes \frac{|0\rangle - |1\rangle}{\sqrt{2}} \\ &= \frac{1}{\sqrt{2^{(n+1)}}} \sum_{x\in\{0,1\}^n} |x\rangle|0\rangle - \frac{1}{\sqrt{2^{(n+1)}}} \sum_{x\in\{0,1\}^n} |x\rangle|1\rangle \ . \end{aligned} \tag{18.20}$$

Az U_f kapu vezérlő bemenete minden lehetséges bemenet szuperpozícióját tartalmazza. A vezérelt kvantumbit szintén szuperpozícióban van, így az U_f kapu ezen szuperpozíció mindkét $|D\rangle_{IN} = \frac{|0\rangle - |1\rangle}{\sqrt{2}}$ elemére külön-külön hat,

a következőképpen:

$$\begin{aligned} |\varphi_2\rangle &= U_f|\varphi_1\rangle \\ &= \frac{1}{\sqrt{2^{(n+1)}}}\sum_{x\in\{0,1\}^n}|x\rangle|f(x)\rangle - \frac{1}{\sqrt{2^{(n+1)}}}\sum_{x\in\{0,1\}^n}|x\rangle|1\oplus f(x)\rangle \\ &= \frac{1}{\sqrt{2^n}}\sum_{x\in\{0,1\}^n}|x\rangle\frac{|f(x)\rangle - |1\oplus f(x)\rangle}{\sqrt{2}} \\ &= \frac{1}{\sqrt{2^n}}\sum_{x\in\{0,1\}^n}(-1)^{f(x)}|x\rangle\otimes\frac{|0\rangle-|1\rangle}{\sqrt{2}}\ , \end{aligned} \tag{18.21}$$

A vezérlő kvantumbitekre alkalmazott $H^{\otimes n}$ Hadamard-transzformáció eredménye:

$$\begin{aligned} |\varphi_3\rangle &= (H^{\otimes n}\otimes I)|\varphi_2\rangle \\ &= \frac{1}{\sqrt{2^n}}\sum_{x\in\{0,1\}^n}(-1)^{f(x)}H^{\otimes n}|x\rangle\otimes\frac{|0\rangle-|1\rangle}{\sqrt{2}} \\ &= \frac{1}{\sqrt{2^n}}\sum_{x\in\{0,1\}^n}(-1)^{f(x)}\frac{1}{\sqrt{2^n}}\sum_{y\in\{0,1\}^n}(-1)^{x\cdot y}|y\rangle\otimes\frac{|0\rangle-|1\rangle}{\sqrt{2}} \\ &= \sum_{y\in\{0,1\}^n}\underbrace{\left(\frac{1}{2^n}\sum_{x\in\{0,1\}^n}(-1)^{x\cdot y+f(x)}\right)}_{z_y}|y\rangle\otimes\frac{|0\rangle-|1\rangle}{\sqrt{2}}\ . \end{aligned} \tag{18.22}$$

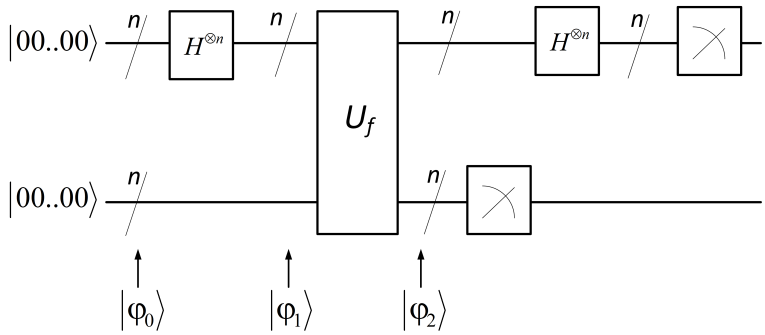
Mindezek után egy mérést kell végrehajtanunk. Az algoritmus érdekessége, hogy elég azt megnézni, mi annak az esélye, hogy a vezérlőbitek értéke $|y\rangle_n = |\mathbf{0}\rangle$. (Az egyszerűség kedvéért bevezetjük a klasszikus 0 és 1 értékkel kitöltött regiszterünkre a következő jelölést: $|0\rangle_n = |\mathbf{0}\rangle$, és $|1\rangle_n = |\mathbf{1}\rangle$.) Lássuk, miért érdemes a $|\mathbf{0}\rangle$ állapotot megvizsgálni. Ebben az esetben $xy = x\mathbf{0} \equiv 0$, vagyis a $z_\mathbf{0}$ együtthatója:

$$z_\mathbf{0} = \frac{1}{2^n}\sum_{x\in\{0,1\}^n}(-1)^{x\cdot y+f(x)} = \frac{1}{2^n}\sum_{x\in\{0,1\}^n}(-1)^{f(x)}\ . \tag{18.23}$$

Ekkor, ha az f függvény *konstans*, $z_\mathbf{0}$ is konstans, az értéke pedig

$$z_\mathbf{0} = \begin{cases} -1 & \text{ha } f(x)\equiv 1 \\ 1 & \text{ha } f(x)\equiv 0\ . \end{cases} \tag{18.24}$$

A mérési valószínűség a második posztulátum értelmében az együttható abszolút érték négyzete, mely $z_\mathbf{0}$ előjelétől függetlenül 1, vagyis konstans esetben a vezérlőbiteket teljes bizonyossággal $|\mathbf{0}\rangle$ állapotban találjuk.



18.7. ábra. A Simon-algoritmust megvalósító kvantumáramkör.

Ha f függvényünk *kiegyensúlyozott*, akkor a (18.23) képletben a szumma alatt azonos arányban szerepelnek a pozitív és negatív előjelű tagok, így $z_{\mathbf{0}} = 0$, tehát az $|y\rangle_n = |\mathbf{0}\rangle$ állapot valószínűsége 0. Ezzel a módszerrel egyetlen lépésben eldöntöttük, hogy a függvényünk konstans-e, vagy kiegyensúlyozott.

Összehasonlításképpen, klasszikus esetben a függvényt szerencsés esetben legalább kétszer ki kellene értékelni (szerencsés eset az, ha a függvény kiegyensúlyozott, és az első két eredmény különbözik egymástól), legrosszabb esetben pedig $2^{n-1}+1$-szer kell kiértékelnünk (ez akkor áll elő, ha a függvény konstans, hiszen az összes lehetséges bemenet felénél legalább eggyel többször ki kell értékelnünk ahhoz, hogy ebben biztosak lehessünk).

18.4.5. A Simon-algoritmus

Lássunk most egy példát arra, amikor a függvényünk kimenete is n bitből áll úgy, ahogy a 18.8-as ábrán látható. Legyen f egy $f(x) : \{0,1\}^n \to \{0,1\}^n$ típusú függvény. Tudjuk, hogy a függvényünk kimenete $f(x) = x \oplus s$, valamilyen ismeretlen s-re, és mi erre az $s \in \{0,1\}^n$ vagyunk kíváncsiak.

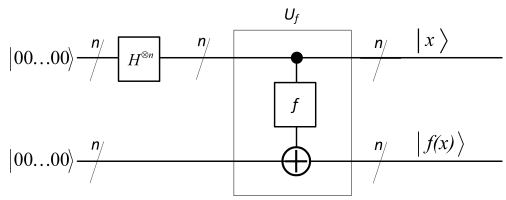
Megjegyzés: ha egy f-vezérelt kaput több bites kimenettel értelmezünk, ügyelnünk kell arra, hogy a kapu mátrixa továbbra is unitér legyen.

Ha a kapu vezérlő bemenete $|x\rangle$, és a vezérelt bit $|\mathbf{0}\rangle$, akkor a kapu kimenete $|x\rangle \otimes |f(x)\rangle$.

A Simon-algoritmus során stratégiánk az lesz, hogy először egy n-bit hosszú, véletlenszerű r bitsorozatot választunk, majd létrehozzuk a következő szuperpozíciót:

$$\frac{1}{\sqrt{2}}|r\rangle + \frac{1}{\sqrt{2}}|r \oplus s\rangle \ . \tag{18.25}$$

Ezután Fourier-mintavételezéssel egy véletlenszerű y-hoz jutunk, amelyre igaz, hogy $y \cdot s = 0 \mod 2$. Ezt $n-1$-szer megismételve egy $n-1$ tagból álló



18.8. ábra. Általánosított f-vezérelt kapu.

lineáris egyenletrendszerhez jutunk, melyet már megoldhatunk s-re.

Klasszikus esetben n növekedésével hatványfüggvény szerint növekszik a szükséges kiértékelések száma, ezzel szemben – mint azt látni fogjuk – a kvantumos Simon-algoritmus számára elég n-szer kiértékelni $f(x)$ függvényünket.

Nézzük meg a Simon-algoritmust részleteiben is. A rendszer kezdeti állapota $|\varphi_0\rangle = |\mathbf{0}\rangle$. A Hadamard-transzformáció után az U_f kapu vezérlő bemenete szuperpozícióban tartalmazni fogja az összes lehetséges x értéket, míg az összes vezérelt kvantumbit értéke $|\mathbf{0}\rangle$, azaz

$$|\varphi_1\rangle = \left(H^{\otimes n} \otimes I\right)|\varphi_0\rangle = \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} |x\rangle|\mathbf{0}\rangle \ . \tag{18.26}$$

Az f-vezérelt függvény alkalmazása után:

$$|\varphi_2\rangle = U_f|\varphi_1\rangle = \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} |x\rangle|f(x)\rangle \ . \tag{18.27}$$

Ha végrehajtunk egy mérést a vezérelt kvantumbiteken, egy $|f(x_0)\rangle$ állapotot találunk, amelyik valamilyen véletlenszerű x_0 bemenethez tartozik. A mérést követően a rendszer a következő állapotba kerül:

$$|\psi\rangle = \frac{1}{\sqrt{2}}|x_0\rangle + \frac{1}{\sqrt{2}}|x_0 \oplus s\rangle \ . \tag{18.28}$$

A regiszter tehát tartalmazza s-et, de nincs egy olyan egyszerű mérés, amivel hozzá tudnánk férni. Mivel a kapcsolódó levezetés meghaladja e könyvfejezet kereteit, bizonyítás nélkül fogadjuk el, hogy optimális megoldás, ha Fourier-mintavételezéstl hajtunk végre. Miután alkalmaztuk az n-kvantumbites Hadamard-kaput a $|\psi\rangle$ vezérlőbitekre, a következő állapotot kapjuk: $\sum_{y\in\{0,1\}^n} \beta_y|y\rangle$. A β_y együtthatókat úgy írhatjuk fel, mint:

$$\beta_y = \frac{(-1)^{r\cdot y}}{\sqrt{2^{(n+1)}}} + \frac{(-1)^{(r\oplus s\cdot y)}}{\sqrt{2^{(n+1)}}} = \frac{(-1)^{r\cdot y}}{\sqrt{2^{(n+1)}}}\left[1 + (-1)^{s\cdot y}\right] \ . \tag{18.29}$$

Ebből látható, hogy ha $s \cdot y = 1 \mod 2$, akkor $\beta_y = 0$. Ha $s \cdot y = 0 \mod 2$, akkor $\beta_y = \frac{(-1)^{r \cdot y}}{\sqrt{2^{(n+1)}}}$. Tehát a mérés után valamelyik olyan y állapotban fogjuk találni a rendszert, hogy $y \cdot s = 0 \mod 2$, méghozzá a következő valószínűséggel: $|\beta_y|^2 = \frac{1}{2^{(n-1)}}$.

A fenti eljárást még $n-1$-szer meg kell ismételnünk, így egy lineáris egyenletrendszerhez jutunk, melyet megoldhatunk s-re.

Megjegyzés: Sajnálatos módon a vezérelt biteken végzett kvantummérés véletlenszerűen választ az $|x\rangle$ bázisvektorai közül, ami lehetővé teszi, hogy egy x_0 a mérések sorozatában egynél többször is előforduljon. Szerencsére annak valószínűsége, hogy a lineáris egyenletrendszert még mindig nem lehet megoldani n lépés után, n függvényében exponenciálisan kicsi.

Gyakorlatok

18.4-1. Bizonyítsuk, hogy a következő transzformáció unitér: $U_f : |x\rangle_N |y\rangle \to |x\rangle_N |y \oplus f(x)\rangle$.

18.4-2. Számoljuk ki a Simon-algoritmus futtatási értékeit a következő paraméterekkel: $n = 4$ és $s = 1011$.

18.5. Kvantum Fourier-transzformáció

18.5.1. QFT

Röviden ismételjük át a klasszikus diszkrét Fourier-transzformáció (DFT) definícióját. Ez egy olyan transzformáció, mely két vektor között teremt kapcsolatot. Vegyünk egy $\mathbf{x} = [x_0, x_1, ..., x_{N-1}]^T$ vektort komplex $x_i \in \mathbb{C}$ koordinátákkal. $\mathbf{x}$ diszkrét Fourier-transzformációját így jelöljük: $\mathbf{y} = \mathrm{DFT}\{\mathbf{x}\}$, ahol az $\mathbf{y}$ vektor k-adik eleme:

$$y_k \triangleq \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} x_i e^{j \frac{2\pi}{N} ik} \ . \tag{18.30}$$

Ismerkedjünk meg ennek a kvantumos megfelelőjével, a ***kvantum-Fourier-transzformációval,*** amit az angol elnevezéséből (Quantum Fourier Transformation) QFT-nek rövidítünk. Tekintsük a $|\varphi\rangle$ szuperpozíciót az $|i\rangle, i = 0...N-1$ bázisvektorok terében:

$$|\varphi\rangle = \sum_{i=0}^{N-1} \varphi_i |i\rangle,$$

majd (18.30)-hoz hasonlóan transzformáljuk át $|\psi\rangle = F|\varphi\rangle$ módon, ahol

$$\psi_k \triangleq \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} \varphi_i e^{j \frac{2\pi}{N} ik} \ . \tag{18.31}$$

Ennek eredménye a következő szuperpozíció:

$$|\psi\rangle = \sum_{k=0}^{N-1} \psi_k |k\rangle = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \sum_{i=0}^{N-1} \varphi_i e^{j\frac{2\pi}{N}ik} |k\rangle \ . \tag{18.32}$$

Ha szuperpozíció helyett egyetlen $|i\rangle$ bázisvektorra alkalmazzuk a QFT-t, az eredmény:

$$F|i\rangle = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{j\frac{2\pi}{N}ik} |k\rangle \ . \tag{18.33}$$

A QFT inverzének (IQFT) együtthatóit a következőképpen definiálhatjuk:

$$\varphi_i \triangleq \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \psi_k e^{-j\frac{2\pi}{N}ik} \ , \tag{18.34}$$

ami egyetlen bázisvektorra alkalmazva a következő eredményre vezet:

$$F^\dagger |k\rangle = \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} e^{-j\frac{2\pi}{N}ik} |i\rangle. \tag{18.35}$$

Jelen fejezetben nem térünk ki rá, de be lehet bizonyítani, hogy a QFT művelet unitér, az adjungáltja pedig nem más, mint az inverze, így jogos az $F^\dagger$ jelölés használata. Sok kvantumalgoritmus egyébként nem a QFT-t, hanem az IQFT-t használja.

Fontos megjegyeznünk, hogy a kvantum Fourier-transzformációl nem állítja elő explicite a Fourier-koefficienseket. A DFT és QFT közötti különbség éppen az, hogy utóbbiban csak egy valószínűségi eloszlást állítunk elő, amelyet egy későbbi mérés során majd mintavételezünk.

Ugyan a QFT egy nagyon összetett transzformációnak tűnik, meglepő módon mégis fel tudjuk építeni elemi kapukból. Ahhoz, hogy ezt megtehessük, fel kell bontani a QFT-t tenzorszorzat alakra.

Ehhez először vezessük be egy nemnegatív egész k szám $k \in \{0, 1, \ldots, 2^n - 1\}$ bináris reprezentációját a következőképpen: $(k_1, k_2, \ldots, k_n) = k_1 2^{n-1} + k_2 2^{n-2} + \cdots + k_n 2^0$, ahol $k_l \in \{0, 1\}$. Emellett vezessük be a bináris tizedes tört fogalmát $h \geq 0$-ra a következő módon:

$$0.k_l k_{l+1} \ldots k_{l+h} \triangleq \frac{k_l}{2^1} + \frac{k_{l+1}}{2^2} + \cdots + \frac{k_{l+h}}{2^{h+1}}; k_m \in \{0, 1\} \ . \tag{18.36}$$

A (18.33)-as képletben helyettesítsük 2^n-t N helyére, és az egész számokat írjuk át bináris formába.

$$F|i\rangle = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{j\frac{2\pi}{N}ik} |k\rangle = \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} e^{j2\pi i \sum_{l=1}^{n} k_l \frac{2^{n-l}}{2^n}} |k\rangle \ .$$

Felhasználva azokat a triviális összefüggéseket, miszerint $\frac{2^{n-l}}{2^n} = 2^{-l}$, valamint $|k\rangle = |k_1, k_2, .., k_n\rangle = |k_1\rangle \otimes |k_2\rangle \otimes ... \otimes |k_n\rangle$ és $e^{\alpha+\beta} \equiv e^{\alpha} e^{\beta}$, felírhatjuk, hogy:

$$F|i\rangle = \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} \prod_{l=1}^{n} e^{j2\pi i k_l 2^{-l}} \bigotimes_{l=1}^{n} |k_l\rangle = \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} \bigotimes_{l=1}^{n} e^{j2\pi i k_l 2^{-l}} |k_l\rangle \ .$$

Mivel $k_l \in \{0, 1\}$ bináris, összegyűjthetjük a tenzorszorzat elemeit annak függvényében, hogy együtthatójuk $|0\rangle$ vagy $|1\rangle$:

$$\begin{aligned} F|i\rangle &= \frac{1}{\sqrt{2^n}} \bigotimes_{l=1}^{n} \left(e^{j2\pi i (k_l=0) 2^{-l}} |0\rangle + e^{j2\pi i (k_l=1) 2^{-l}} |1\rangle \right) \qquad (18.37) \\ &= \frac{1}{\sqrt{2^n}} \bigotimes_{l=1}^{n} \left(|0\rangle + e^{j2\pi i 2^{-l}} |1\rangle \right) \ . \end{aligned}$$

A fenti összefüggésből látszódik, hogy minden $|i\rangle$ bázisvektor Fourier-transzformáltjának létezik tenzorszorzat-felbontása. Mivel a QFT lineáris művelet, ezért elég megvizsgálni hatását a bázisvektorokra, ebből már következik, hogy tetszőleges szuperpozíciónak is létezik tenzorszorzat-felbontása.

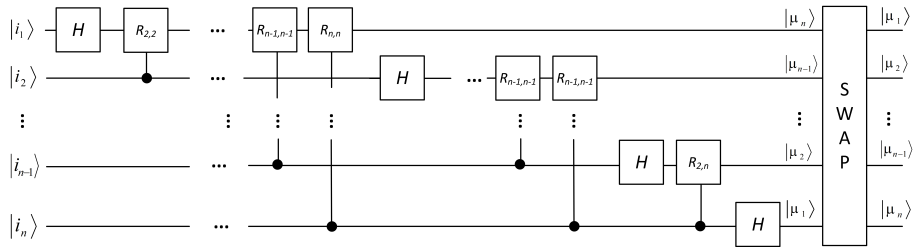
A tenzorszorzat tényezőire vezessük be a következő jelölést:

$$|\mu_l\rangle \triangleq \frac{1}{\sqrt{2}} \left(|0\rangle + e^{j2\pi i 2^{-l}} |1\rangle \right) \ . \qquad (18.38)$$

Ezt tovább egyszerűsíthetjük, ha felhasználjuk a következő összefüggést: $i = \sum_{l=1}^{n} i_l 2^{n-l}$, illetve a 2π bináris tizedes tört formáját, ahogy azt a (18.36) képletben definiáltunk. Ezzel $(2\pi i 2^{-l}) \mod 2\pi = 0.i_{l-n} i_{l-n+1} ... i_n$, ahol csak az $i_h, h \geq 0$ tagok számítanak. Mindezt felhasználva:

$$\begin{aligned} F|i\rangle &= \underbrace{\left(\frac{|0\rangle + e^{j2\pi 0.i_n} |1\rangle}{\sqrt{2}} \right)}_{|\mu_1\rangle} \otimes \underbrace{\left(\frac{|0\rangle + e^{j2\pi 0.i_{n-1} i_n} |1\rangle}{\sqrt{2}} \right)}_{|\mu_2\rangle} \ldots \qquad (18.39) \\ &\otimes \underbrace{\left(\frac{|0\rangle + e^{j2\pi 0.i_1 i_2 \ldots i_n} |1\rangle}{\sqrt{2}} \right)}_{|\mu_n\rangle} \ . \end{aligned}$$

Az Olvasó mindezek fényében kíváncsi lehet arra, milyen elemi kapukból lehet megépíteni egy QFT áramkört. Nincs szükség másra, mint Hadamard-kapukra, az egyes biteket felcserélő ún. SWAP kapura, és speciális, vezérelt



18.9. ábra. Kvantum Fourier-transzformációt megvalósító áramkör.

fázisforgató kapukra, amelyeket a következő módon írhatunk le:

$$R_{h,p} \triangleq \begin{cases} P(2\pi \frac{1}{2^h}) & \text{if } i_p = 1 \\ 1 & \text{if } i_p = 0 \end{cases} \tag{18.40}$$

A fent említett elemi kapukból összesen $O(n^2)$-re van szükség, az áramkör elrendezése pedig a 18.9-es ábrán látható.

Mivel az inverz kvantum Fourier-transzformáció a QFT inverze, és minden felhasznált elemi kapunak ismerjük az inverzét, könnyen beláthatjuk, hogy az IQFT áramkörét úgy építhetjük meg, ha a QFT-t megvalósító elemi kapuk inverzét fordított sorrendben alkalmazzuk.

18.5.2. A QFT, mint általánosított Hadamard-transzformáció

Emlékezzünk vissza a Hadamard-kapu hatására. Bázisvektorok esetén ez $H|0\rangle = (|0\rangle + |1\rangle)/\sqrt{2}$, $H|1\rangle = (|0\rangle - |1\rangle)/\sqrt{2}$. Az általános esetet leíró (18.7) képletet $n = 1$ darab kvantumbitre alkalmazva:

$$H^{\otimes 1}|i\rangle = \frac{1}{\sqrt{2^1}} \sum_{k \in \{0,1\}^1} (-1)^{ik}|k\rangle = \frac{1}{\sqrt{2}} \sum_{k=0}^{1} (-1)^{ik}|k\rangle \ , \tag{18.41}$$

ahol $i \in \{0, 1\}$. Felhasználva, hogy $-1 = e^{j2\pi\frac{1}{2}}$, (18.41) átírható a következő alakba:

$$H|i\rangle = \frac{1}{\sqrt{2}} \sum_{k=0}^{1} e^{j2\pi\frac{ik}{2}}|k\rangle \ . \tag{18.42}$$

Ha összehasonlítjuk (18.42)-t a QFT (18.33)-as definíciójával, látható, hogy $H|i\rangle$ nem más, mint $|i\rangle$ állapot QFT-ja. Ez egyrészt azt jelenti, hogy az egyetlen kvantumbiten végrehajtott QFT nem más, mint a Hadamard-transzformáció, másrészt azt, hogy $H^{\otimes n}$ nem más, mint n darab kvantumbiten külön-külön végrehajtott QFT.

18.5.3. Kvantum-fázisbecslés

Vegyünk egy U unitér operátort. Ennek sajátvektora $|u\rangle$, sajátértéke pedig $e^{j\alpha_u}$. Minket ez a bizonyos α_u fázis érdekel, vagy másképp megfogalmazva az a $\kappa_u \in [0,1)$ fázisarány, amire $\alpha_u = 2\pi\kappa_u$. Tegyük fel, hogy nem ismerjük U mátrixát, de a sajátvektorait igen, és van egy vezérelhető fekete dobozunk, mely végrehajtja ezt a kvantumműveletet.

Idealizált fázisbecslés

Kezdjük egy idealizált esettel, amikor is κ_u nem tetszőleges, hanem felírható $\kappa_u = i/2^n$ alakban, ahol $i \in \{0,1,...,2^n-1\}$. Ha elő tudnánk állítani a (18.33) jobb oldalát úgy, hogy $i/2^n = \kappa_u$ (ahol $N = 2^n$), elég lenne egy IQFT transzformációt végrehajtani, hogy megkapjuk $|i\rangle$-t, amiből κ_u könnyen kiszámolható. Ha az $i/2^n = \kappa_u$ megszorítás érvényes, akkor a (18.38)-ban definiált $|\mu_l\rangle$ vektort tovább alakíthatjuk a következő módon:

$$|\mu_l\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle + e^{j2\pi 2^{n-l}\kappa_u}|1\rangle\right) . \tag{18.43}$$

Célunk az, hogy a tenzorszorzat-felbontást alkalmazva egy olyan áramkört tervezzünk, mely előállítja az összes $|\mu_l\rangle$ állapotvektort, melyeket majd az IQFT bemeneteként használunk fel. Ehhez először próbáljunk egy olyan áramkört keresni, amely a legnagyobb sorszámú, $|\mu_n\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle + e^{j2\pi\kappa_u}|1\rangle\right)$állapotvektort állítja elő. Amint az látható, ez a $|\mu_l\rangle$ nagyon hasonló egy Hadamard-kapu kimenetéhez, melynek bemenete $|0\rangle$ volt, az egyetlen különbség az $|1\rangle$ bázisvektorhoz tartozó exponenciális faktor. Jobban megnézve észrevehetjük, hogy ez a szorzófaktor nem más, mint az U transzformáció sajátértéke, amely a következő sajátvektorhoz tartozik: $|u\rangle : U|u\rangle = e^{j2\pi\kappa_u}|u\rangle$. Mindezek alapján használjuk a 18.10. ábrán látható áramkört, mely két regisztert tartalmaz. A felső, vezérlő regiszter egyetlen kvantumbitből áll, melynek bemeneti értéke legyen $|0\rangle$. Az alsó, vezérelt regiszterünk kezdeti állapota legyen a t darab kvantumbitből álló $|u\rangle$ sajátvektor. A teljes kezdeti állapot így $|\varphi_0\rangle = |0\rangle \otimes |u\rangle$.

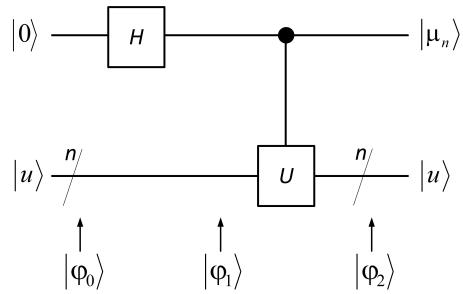
A Hadamard-kapu hatása a vezérlő bitre:

$$|\varphi_1\rangle = (H \otimes I^{\otimes t})|\varphi_0\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}}|u\rangle = \frac{|0\rangle|u\rangle + |1\rangle|u\rangle}{\sqrt{2}}.$$

U mátrixát nem ismerjük, de hatását a sajátvektorára igen, így a szuperpozíció elvét felhasználva kiszámíthatjuk $|\varphi_2\rangle$-t:

$$|\varphi_2\rangle = (I\otimes U)|\varphi_1\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle|u\rangle + e^{j2\pi\kappa_u}|1\rangle|u\rangle\right) = \frac{1}{\sqrt{2}}\left(|0\rangle + e^{j2\pi 2^0\kappa_u}|1\rangle\right)\otimes|u\rangle ,$$

Ebből látható, hogy a vezérlő kvantumbiten éppen a $|\mu_n\rangle$ állapot áll elő.



18.10. ábra. $|\mu_n\rangle$ vektort előállító kvantumáramkör.

A problémát tehát megoldottuk egy speciális esetre. Ahhoz, hogy képesek legyünk egy tetszőleges $|\mu_l\rangle$ állapot előállítására, az exponensben a $2^0 = 1$ szorzótényezőt kell helyettesítenünk 2^{n-l}-lel. Ehhez vizsgáljuk meg az U transzformáció ismételt hatását az $|u\rangle$ sajátvektorra:

$$U^h \triangleq \underbrace{UU...U}_{h},$$

$$U^h|u\rangle = \underbrace{e^{j2\pi\kappa_u}e^{j2\pi\kappa_u}...e^{j2\pi\kappa_u}}_{h}|u\rangle = e^{j2\pi h\kappa_u}|u\rangle.$$

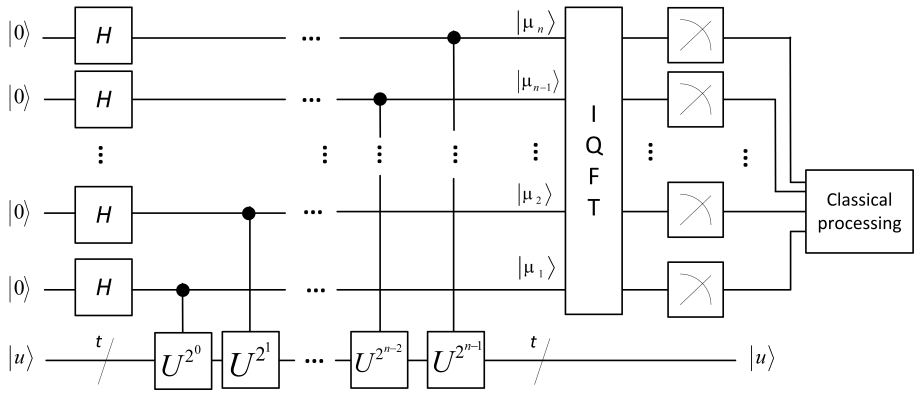
Ebből leolvashatjuk, hogy ha áramkörünkben az $U = U^{2^0}$ kaput helyettesítjük $U^{2^{n-l}}$-el, akkor a vezérlő kvantumbit kimenete éppen a $|\mu_l\rangle$ állapot lesz.

A teljes áramkör, mely az α_u fázis kiszámítására szolgál, a 18.11. ábrán látható. Ez egy n darab kvantumbitből álló vezérlő kvantumregisztert tartalmaz, mely az IQFT bemenetét adja.

Mivel az IQFT után a mérésünk egy m egész számot ad vissza, még egy további klasszikus logikai kapura van szükségünk, amely ebből megbecsüli az $\tilde{\alpha}_u = 2\pi m/2^n = 2\pi i/2^n = \alpha_u$ fázist, ami nem más, mint a pontos eredmény. A mérés után a legnagyobb helyi értékű bit a vezérlő regiszter legalsó, míg a legkisebb helyi értékű bit a legfelső vezetéken található majd. Még egyszer hangsúlyozzuk, hogy áramkörünk azért ad pontos becslést, mert ebben a speciális, idealizált esetben $\kappa_u = i/2^n$.

Fázisbecslés praktikus esetben

Ahogy többször hangsúlyoztuk, az idealizált fázisbecslés csak akkor működik, ha $\kappa_u = i/2^n$. Most azonban egy tetszőleges $\kappa_u \in [0, 1)$ fázisra vagyunk kíváncsiak. Az IQFT ebben az esetben pontatlanul fog működni, az eredmény pedig csak közel lesz a valódi fázishoz, nem fog tökéletesen megegyezni vele.



18.11. ábra. Fázisbecslő kvantumáramkör.

A felső, vezérlő kvantumregiszterünkben növelhetjük ugyan a kvantumbitek n számát, így tetszőlegesen megközelíthetjük az $i/2^n \to \kappa_u$ fázist. Azonban ez a növelés bonyolultabb architektúrával és magasabb költségekkel jár, ráadásul a hétköznapi alkalmazásokban a közelítő megoldás is tökéletesen megfelel. Éppen ezért mérnöki szemmel nézve érdemes a pontos megoldás helyett inkább arra koncentrálni, hogy mi az összefüggés a hiba, és a kvantumbitek száma között. Az alábbiakban csak a kapcsolódó eredményt közöljük, de az érdeklődő Olvasó megtalálhatja a pontos levezetést a fejezet végén felsorolt hivatkozások között.

Jelöljük a $\kappa_u 2^n$-hez legközelebbi n-bit hosszú egész számot a-val, ahol $0 \leq a \leq 2^n - 1$. Ha $\Delta_u \triangleq \kappa_u - a/2^n$, akkor $|\Delta_u| \leq 2^{-(n+1)}$. Feltéve, hogy az IQFT kimenetéről leolvasható m érték hibája ε korláton belül van, vagyis $a - \varepsilon \leq m < a + \varepsilon$, akkor annak P_ε feltételes valószínűsége, hogy m kívül esik ezen az ε korláton, nem más, mint

$$P_\varepsilon \triangleq P(m < a - \varepsilon \vee a + \varepsilon \leq m | a) \ , \tag{18.44}$$

Annak P_s valószínűsége, hogy a mért m az $i = a$ körüli, elfogadható régióba esik, nem más, mint:

$$P_s = \sum_{h=-\varepsilon+1}^{\varepsilon} \frac{1}{2^{2n}} \frac{\sin^2(\pi(2^n \Delta_u + h))}{\sin^2(\pi(\Delta_u + \frac{h}{2^n}))}, \Delta_u \neq 0 \ . \tag{18.45}$$

ahol $h \in (a - 2^{n-1}, a + 2^{n-1}]$, Δ_u pedig

$$\Delta_u = \kappa_u - \frac{\lfloor \frac{\alpha_u}{\pi} 2^{n-1} \rceil}{2^n}. \tag{18.46}$$

A fázisbecslő felső, n darab kvantumbitjének tartalmaznia kell $(c-1)$ kvantumbitet ahhoz, hogy a klasszikus számábrázolási pontosság 2^{-c} legyen, de emellett a kvantumos pontatlansággal is számolnunk kell. A mérésnek mindig van egyfajta bizonytalansága, és ahhoz, hogy ezt a mérési bizonytalanságot egy adott $\breve{P}_\varepsilon$ valószínűség alatt tartsuk, további p kvantumbitre van szükség $(n = c - 1 + p)$, aminek függvényében ε úgy adható meg, mint $\varepsilon = 2^{p-1}$.

18.5.4. Kvantum-faktorizáció

Peter Shor algoritmusa, amelynek kvantumos és klasszikus része is van, lehetővé teszi, hogy egy nagy N számot $O(\mathrm{ld}^3(N))$ lépésben faktorizáljunk. Ebben a részben először megmutatjuk, hogyan lehet a klasszikus prímfelbontást visszavetni egy x egész szám multiplikatív rendjének megkeresésére, majd megismerkedünk a kapcsolódó kvantumalgoritmussal is.

Faktorizáció és multiplikatív rend

Vegyünk két pozitív egész számot $x < N$, melyek relatív prímek, vagyis $\mathrm{lnko}(x, N) = 1$. Az x szám ***multiplikatív rendjét*** modulo N definiáljuk úgy, mint a legkisebb r számot, melyre igaz, hogy

$$x^r \mod N = 1 \tag{18.47}$$

és $1 < r < N$. x rendje szoros kapcsolatban áll a $f(z) = x^z \mod N$ függvény periodicitásával, mert

$$\begin{aligned} f(z+r) &= x^{z+r} \mod N \\ &= ((x^z \mod N) \cdot (\underbrace{x^r \mod N}_{\equiv 1})) \mod N \\ &= f(z)\,. \end{aligned} \tag{18.48}$$

Tegyük fel, hogy meg akarjuk határozni A prímtényezőit. Ha A páros, akkor ismételten eloszthatjuk 2-vel, egészen addig, míg egy páratlan B_1 számhoz nem jutunk. Ezután, ha egy B_1-nél kisebb $(x_1 < B_1)$, véletlenszerűen választott egész x_1-re teljesül, hogy $\mathrm{lnko}(x_1, B_1) = b_1$, és $b \neq 1$, vagyis x_1 és B_1 nem relatív prímek, akkor B_1-t elosztva b_1-el definiálhatunk egy új B_2-t. Ezt n-szer ismételhetjük, amíg nem találunk egy esetet, amikor $b_n{=}1$, ekkor megkaptuk $B_n = N$-et. Innentől kezdve azzal az esettel állunk szemben, amikor x és N relatív prímek, és persze $0 < x < N$. Tegyük fel, hogy az $r = x$ modulo N multiplikatív rend páros, vagyis definiálható egy y a következőképpen:

$$y \triangleq x^{\frac{r}{2}}.$$

A (18.47)-es képletből tudjuk, hogy

$$y^2 \mod N = 1\,, \tag{18.49}$$

amit átrendezhetünk úgy, hogy

$$(y^2-1) \mod N = 0 \Rightarrow ((y+1)(y-1)) \mod N = 0\,, \tag{18.50}$$

Az oszthatóság továbbra is igaz marad, ha a (18.50)-es összefüggést átrendezzük a következőképpen:

$$([\underbrace{(y+1) \mod N}_{b_{+1}}][\underbrace{(y-1) \mod N}_{b_{-1}}]) \mod N = 0, \tag{18.51}$$

ahol $0 \leq b_{+1}, b_{-1} < N$.

Szükségünk lesz emellett a következő legnagyobb közös osztókra: $c_{+1} = \text{lnko}(b_{+1}, N)$ és $c_{-1} = \text{lnko}(b_{-1}, N)$. A (18.51) összefüggésben 0 csak háromféleképpen jöhet ki:

1. $b_{+1} = 0$, ekkor $c_{+1} = N$, $b_{-1} = N-2$, $c_{-1} = 1$,
2. $b_{-1} = 0$, ekkor $c_{-1} = N$, $b_{+1} = 2$, $c_{+1} = 1$,
3. $b_{+1}b_{-1} = kN, 0 < k < N \Rightarrow 0 < b_{-1} < b_{+1} < N$, vagyis sem b_{+1} sem b_{-1} nem osztója N -nek. Hogy teljesítsük a (18.51) feltételt $b_{+1}b_{-1}$-nek kell, hogy legyen közös osztója N-el, nevezetesen c_{+1} és c_{-1}, melyek N nem triviális faktorai.

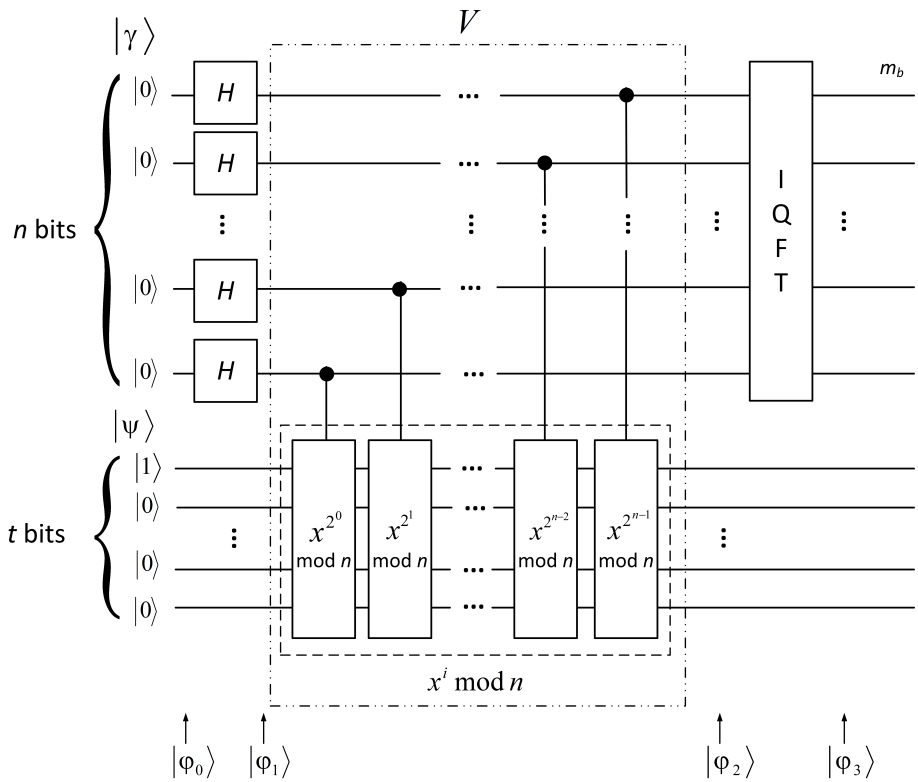
Ha x-re nem tejesül, hogy $0 < c_{-1}, c_{+1} < N$, vagy N-nek vannak további nem triviális faktorai, akkor egy új x-et kell választani, amíg N összes prímtényezőjét meg nem találjuk.

Ez az eljárás azonban feltételezi, hogy már ismerjük r-et. Ennek megkereséséhez a következő fejezetben egy polinomiális idejű kvantumos rendkeresést fogunk használni.

Kvantumos rendkeresés

A Shor-algoritmusnak három fő része van. Első lépésként ki kell számolnunk az összes $x^k \mod N$ értéket $0 \leq k < N$ tartományban, és el kell tárolnunk őket a k értékekkel együtt két kvantumregiszterben, úgy, hogy a két, $t = \lceil \text{ld}(N) \rceil$ kvantumbit nagyságú regiszter összefont állapotban legyen. Másodszor, a $|x^k \mod N = 1\rangle$ állapot valószínűségi amplitúdóját meg kell növelünk úgy, hogy a mérési valószínűség olyan közel legyen 1-hez amennyire csak lehetséges. Utolsó lépésként a második regiszteren elvégzünk egy mérést, ami nagy valószínűséggel a keresett r rendet adja vissza. A teljes folyamatot végrehajtó kvantumáramkör a 18.12. ábrán látható.

A rendszer kezdeti állapota $|\varphi_0\rangle = |\gamma_0\rangle|\psi_0\rangle$, ahol $\gamma_0, \psi_0 \in [0, 2^t - 1)$. Praktikus okokból válasszuk a $|\gamma_0\rangle = |\mathbf{0}\rangle$ bemenetet, hiszen ebből egy n-qubitre



18.12. ábra. A Shor-algoritmust megvalósító kvantumáramkör.

ható Hadamard-kapu felhasználásával előállíthatjuk az összes lehetséges k értéket, amikhez tartozó r rendet keressük. Azért, hogy egyszerűsítsük a számolást, feltételezzük, hogy $n = t$. A rendszer állapota így:

$$|\varphi_1\rangle = \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} |k\rangle \otimes |\psi_0\rangle \ . \tag{18.52}$$

A 18.12. ábrán V-vel jelölt kapu két részből áll, egy alsó részből, mely előállítja az $x^k \mod N$ értékeket, és egy felső részből, mely az összes lehetséges k értéket tartalmazza. Egy ilyen kapu matematikailag úgy definiálható, mint $V|\varphi_1\rangle = |\varphi_2\rangle$, ahol

$$|\varphi_2\rangle = \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} |k\rangle |x^k \mod N\rangle \ . \tag{18.53}$$

Moduláris hatványozást alkalmazva

$$\begin{aligned} x^k \mod N &= \prod_{l=1}^{2^n} \left(x^{k_l 2^{n-l}} \mod N \right) \\ &= \left(x^{k_1 2^{n-1}} \mod N \right) \cdot \left(x^{k_2 2^{n-2}} \mod N \right) \cdot \ldots \\ &\quad \cdot \left(x^{k_n 2^0} \mod N \right) , \end{aligned} \tag{18.54}$$

ahol $k = k_1 2^{n-1} + k_2 2^{n-2} + \cdots + k_n 2^0$ és $k_l \in \{0, 1\}$. Ezt az összefüggést tudjuk kapuk sorozataként is értelmezni, a következőképpen: az $(n-l)^{th}$-edik kapu az $f(l) = x^{k_l 2^{n-l}}$ függvényt valósítja meg, és az l-edik kvantumbit vezérli a kitevőben lévő k_l-en keresztül. Ez nem más, mint a fázisbecslő áramkörünk első fele. A nagyfokú hasonlóság miatt szánjunk rá egy kevés időt arra, hogy összehasonlítjuk a fázisbecslő és faktorizációs áramköröket. A faktorizáció esetén az ismeretlen U transzformáció:

$$U : |q\rangle \rightarrow |(qx) \mod N\rangle . \tag{18.55}$$

Ha IQFT-t alkalmazunk a felső regiszterre, úgy, ahogy azt a fázisbecslés esetén tettük, U sajátértékeit kapjuk eredményül. Jelöljük a fázisbecslő áramkör állapotát az IQFT előtt $|\varphi_{P2}\rangle$, ahogy az a 18.11 ábrán látható, míg a faktorizációs áramkör állapota az IQFT előtt legyen $|\varphi_2\rangle$.

Mivel az $x^k \mod N$ függvény r szerint periodikus, és csak r darab különböző értéke lehet, (18.53) átalakítható:

$$|\varphi_2\rangle = \frac{1}{\sqrt{2^n}} \sum_{k=0}^{r-1} \left(\sum_{z=0}^{Z_k} |zr + k\rangle \right) |x^k \mod N\rangle , \tag{18.56}$$

Z_k meghatározásához észre kell vennünk, hogy $0 \leq zr + k \leq 2^n - 1$ és z egész szám, amiből következik, hogy $0 \leq z \leq \lfloor \frac{2^n - 1 - k}{r} \rfloor = Z_k$.

Ezzel szemben a fázisbecslő $|\varphi_{P2}\rangle$ állapotát úgy írhatjuk fel, mint:

$$|\varphi_{P2}\rangle = \sum_{k=0}^{2^n-1} \frac{1}{\sqrt{2^n}} e^{j2\pi k \kappa_u} |k\rangle |u\rangle. \tag{18.57}$$

A (18.57) egyenlet csak akkor érvényes, ha a fázisbecslő bemenete az $|u\rangle$ sajátvektor, ezzel szemben a faktorizációs áramkör alsó felének $|\psi_0\rangle$ bemenetéről egyelőre nem tudunk semmit. A következőkben meghatározzuk a megfelelő sajátértékeket és sajátvektorokat, amelyek majd a kapu bemeneteként szolgálnak majd.

Tudjuk, hogy a faktorizációs áramkör $|\psi_0\rangle$-hoz tartozó kimenete

$|x^k \mod N\rangle$, ezt szeretnénk előállítani a fázisbecslő kimeneteként. A $|\psi_0\rangle$ bemenet ennek fényében legyen a sajátvektorok egyenlő arányú szuperpozíciója:

$$|\psi_0\rangle = \sum_{b=0}^{B-1} \frac{1}{\sqrt{B}} |u_b\rangle, \tag{18.58}$$

ahol B a felhasznált sajátvektorok számát jelöli (B pontos értéke majd csak a későbbi vizsgálataink során dől el). A fázisbecslő kimenetét $|\psi_0\rangle$ bemenet mellett úgy határozhatjuk meg, hogy a szuperpozícióban az összeg minden tagjára alkalmazzuk a (18.57)-es lineáris összefüggésünket.

$$|\varphi_{P2}\rangle = \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} \sum_{b=0}^{B-1} \frac{e^{j2\pi k\kappa_b}}{\sqrt{B}} |k\rangle|u_b\rangle = \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} |k\rangle \sum_{b=0}^{B-1} \frac{e^{j2\pi k\kappa_b}}{\sqrt{B}} |u_b\rangle, \tag{18.59}$$

ahol κ_b jelöli az $|u_b\rangle$ sajátvektorhoz tartozó fázisarányt. A periodicitást kihasználva (18.59)-et átírhatjuk (hasonlóan ahhoz, ahogy azt (18.53)-ban tettük).

$$|\varphi_{P2}\rangle = \frac{1}{\sqrt{2^n}} \sum_{k=0}^{r-1} \sum_{z=0}^{Z_k} |zr+k\rangle \sum_{b=0}^{B-1} \frac{e^{j2\pi k\kappa_b}}{\sqrt{B}} |u_b\rangle. \tag{18.60}$$

Ha most összehasonlítjuk a fázisbecslő $|\varphi_{P2}\rangle$ kimenetét (18.60)-ban a faktorizációs áramkör $|\varphi_2\rangle$ kimenetének (18.56)-ban felírt alakjával, akkor látható, hogy a kettő akkor egyenlő, ha a következő feltétel teljesül:

$$|x^k \mod N\rangle = \sum_{b=0}^{B-1} \frac{e^{j2\pi k\kappa_b}}{\sqrt{B}} |u_b\rangle, k = 0 \ldots r-1. \tag{18.61}$$

Az összefüggés jobboldalán még meg kell keresnünk az együtthatók egy megfelelő értékét. Ehhez használjuk fel a következő matematikai összefüggést:

$$\sum_{b=0}^{B-1} \frac{e^{-j2\pi \frac{b}{B}s}}{B} = \sum_{b=0}^{B-1} \frac{e^{+j2\pi \frac{b}{B}s}}{B} = \delta(s-B) = \begin{cases} 1, & \text{ha } s \text{ többszöröse } B\text{-nek} \\ 0, & \text{egyébként,} \end{cases} \tag{18.62}$$

amivel átírhatjuk (18.61) bal oldalát:

$$|x^k \mod N\rangle = \sum_{s=0}^{B-1} \underbrace{\sum_{b=0}^{B-1} \frac{e^{-j2\pi \frac{b}{B}s}}{B}}_{\delta(s-0)} |x^k \mod N\rangle.$$

Mivel k 0-tól $r-1$-ig fut, legyen $B = r$

$$|x^k \mod N\rangle = \sum_{s=0}^{r-1} \underbrace{\sum_{b=0}^{r-1} \frac{e^{-j2\pi\frac{b}{r}(s-k)}}{r}}_{\delta(s-k)} |x^s \mod N\rangle,$$

amit tovább alakíthatunk:

$$|x^k \mod N\rangle = \sum_{b=0}^{r-1} \frac{e^{j2\pi\frac{k}{r}b}}{\sqrt{r}} \sum_{s=0}^{r-1} \frac{e^{-j2\pi\frac{b}{r}s}}{\sqrt{r}} |x^s \mod N\rangle. \tag{18.63}$$

Ha ezt az eredményt, amit (18.61) bal oldalának átalakításával kaptunk, összehasonlítjuk (18.61) jobb oldalával, levonhatjuk a következtetést, hogy $k = 1...r-1$ esetén a fázisarányok és sajátvektorok:

$$\kappa_b = \frac{b}{r}, |u_b\rangle = \sum_{s=0}^{r-1} \frac{e^{-j2\pi\frac{b}{r}s}}{\sqrt{r}} |x^s \mod N\rangle, \tag{18.64}$$

Ebből látható, hogy a fázisarányok, melyeket az IQFT eredményeként kapunk, tartalmazzák az r multiplikatív rendet, azonban még mindig nem ismerjük a megfelelő $|\psi_0\rangle$ kezdeti állapotot. Ennek meghatározásához helyettesítsük vissza a kiszámított $|u_b\rangle$ sajátvektorokat, és a $B = r$ összefüggést (18.58)-ba:

$$\begin{aligned} |\psi_0\rangle &= \sum_{b=0}^{r-1} \frac{1}{\sqrt{r}} |u_b\rangle = \sum_{b=0}^{r-1} \frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} \frac{e^{-j2\pi\frac{b}{r}s}}{\sqrt{r}} |x^s \mod N\rangle \\ &= \frac{1}{r} \sum_{s=0}^{r-1} \underbrace{\left(\sum_{b=0}^{r-1} e^{-j2\pi\frac{b}{r}s} \right)}_{r\delta(s-0)} |x^s \mod N\rangle \\ &= |x^0 \mod N\rangle = |1\rangle_{2^t}. \end{aligned} \tag{18.65}$$

Ez azt jelenti, hogy az alsó regiszter bemeneti állapota a klasszikus $|1\rangle_{2^t}$ állapot. Érdemes megjegyezni, hogy nem használtunk semmiféle explicit ismeretet U transzformációs szabályával kapcsolatban, amikor meghatároztuk U sajátvektorait, sajátértékeit és $|\psi_0\rangle$-at. Helyette arra az indirekt információra hagyatkoztunk, ami $|\varphi_1\rangle$-ben és $|\varphi_2\rangle$-ben el volt rejtve.

Az IQFT kimenetén végzett mérés egy m_b becslést ad a $\kappa_b \approx m_b/2^n$ fázisszögre, ahol κ_b-t egyenletes eloszlással választottuk $b \in [0...r)$halmazból. Ahhoz, hogy $m_b/2^n$-ből meghatározzuk az r multiplikatív rendet, a következő tételt használjuk fel:

18.1. tétel. *Ha $\frac{m_b}{2^n}$ egy racionális tört, melyben b és r pozitív egész számok, melyek kielégítik a következő összefüggést:*

$$\left|\frac{b}{r} - \frac{m_b}{2^n}\right| \leq \frac{1}{2r^2} \tag{18.66}$$

akkor $\frac{b}{r}$ nem más, mint az $\frac{m_b}{2^n}$ lánctört határértéke.

A feltétel teljesítéséhez az kell, hogy

$$\frac{1}{2^{(n+1)}} \leq \frac{1}{2r^2} \Rightarrow r^2 \leq 2^n \underbrace{\Rightarrow}_{r<N} N^2 \leq 2^n \Rightarrow n = \lceil \mathrm{ld}(N^2) \rceil. \tag{18.67}$$

Mivel 2^n-t úgy választottuk, hogy N^2-nél nagyobb, vagy azzal egyenlő legyen, és $r < N$, csak egy olyan b van, ami teljesíti (18.66)-ot. Tegyük fel, hogy a határérték alakja b'/r'. $x^{r'} \mod N$ kiszámításakor vagy 1-et kapunk eredményül, ami bizonyítja, hogy x rendje $r' = r$, vagy bármilyen más értéket, amikor is r többszöröse r'-nek. A második esetben meg kell ismételni az algoritmust, hogy egy másik r'-t keressünk. Annak valószínűsége, hogy r és r' relatív prímek $P(\mathrm{lnko}(r, r') = 1) = \frac{1}{O(\mathrm{ld}(\mathrm{ld}(N)))}$, és ha az algoritmust megismételjük $O(\mathrm{ld}(\mathrm{ld}(N)))$ alkalommal, nagy valószínűséggel megfelelő r'-t kapunk.

Gyakorlatok

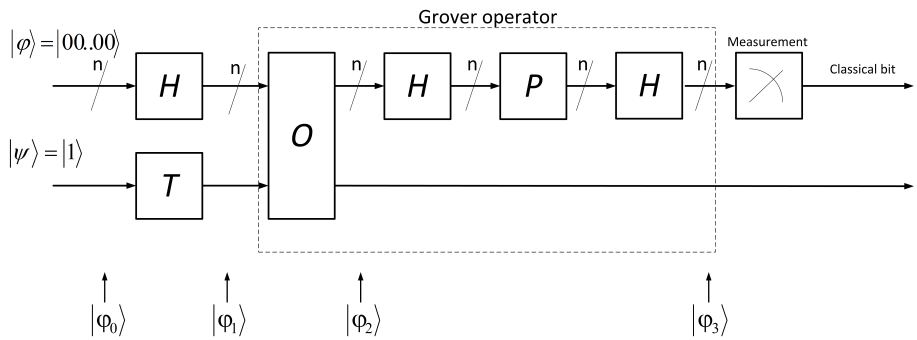
18.5-1. Határozzuk meg a QFT transzformáció mátrixát.

18.5-2. Bizonyítsuk be, hogy a kvantum Fourier transzformáció unitér.

18.6. Kvantum alapú keresés

Vegyünk egy N különböző bejegyzést tartalmazó DB adatbázist, melynek bejegyzéseit $x \in [0, N-1]$-ként indexeltük. Arra vagyunk kíváncsiak, hogy melyik x_0 index mutat a keresett $\mathrm{DB}[x_0] = \mathit{Alice}$ bejegyzésre.

Mint minden kvantumalgoritmus esetén, itt is a kvantumpárhuzamosság megteremtésével kezdünk neki a feladatnak. Ez lehetővé teszi majd, hogy az összes lehetséges bemeneti érték azonos amplitúdójú szuperpozícióját előállítsuk. Ez lesz majd az U_f unitér transzformációnk bemenete, amelyet ebben a konkrét esetben O-val jelölünk, feladata pedig az, hogy minden bemeneti bázisvektorhoz a megfelelő adatbázis bejegyzést adja eredményül. U_f-hez mindig tartozik egy kiegészítő (alsó) kvantumregiszter, melyen a kimenet megjelenik, és mely biztosítja a kapu reverzibilitását. Ez után általában egy Hadamard-művelet vagy IQFT következik a felső kvantumregiszteren,



18.13. ábra. A Grover-keresést megvalósító áramkör.

amivel megnöveljük a megfelelő állapotok mérési valószínűségét. Algoritmusunk a felső, vezérlő kvantumregiszter mérésével ér véget. Amint az a 18.13. ábrán látható, a kvantumos keresés G -vel jelölt Grover-operátorába a fenti lépések közül kettőt is beleértünk: a műveletvégzés O operátorát, és a mérési valószínűség növelésének lépését.

Inicializálás

Ha az adatbázis mérete $N = 2^n$, akkor szükségünk lesz egy n darab kvantumbitből álló kvantumregiszterre, amely a vezérlőbiteket tartalmazza. Ha N nem 2 hatványa, az adatbázist egyszerűen kiterjesztjük néhány lényegtelen bejegyzéssel, hogy teljesüljön ez a feltétel. A felső, vezérlőbitek kezdeti állapota legyen $|\gamma_0\rangle = |\mathbf{0}\rangle$. Továbbá szükségünk van egy vezérelt kvantumbitre is, melynek kezdeti állapotát $|\psi_0\rangle$-al jelöljük. Ezután minden egyes kvantumbitre alkalmazunk egy Hadamard-kaput.

$$|\varphi_1\rangle = (H^{\otimes n} \otimes T^{\otimes t})\,(|\gamma_0\rangle \otimes |\psi_0\rangle) = |\gamma_1\rangle \otimes H|\psi_0\rangle, \tag{18.68}$$

ahol jelen esetben $|\psi_1\rangle = T|\psi_0\rangle = H|1\rangle$ az alsó, $t = 1$ bites regiszter állapota, és

$$|\gamma_1\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle. \tag{18.69}$$

Ezzel a teljes rendszer állapota:

$$|\varphi_1\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle \otimes \frac{|0\rangle - |1\rangle}{\sqrt{2}}.$$

A szakirodalomban azokat az állapotokat, melyek megoldásai a problémának, marked, azaz ***jelölt állapotnak*** szokták nevezni, míg az összes többit unmarked, azaz ***jelöletlen állapotnak*** hívják.

Az orákulum

A következő lépésben alkalmazzuk az orákulum kaput, amely megszorozza a keresett bejegyzés valószínűségi amplitúdóját -1-gyel, és változatlanul hagyja az összes többit, így különböztetve meg a jelölt és jelöletlen állapotokat.

$$O : |x\rangle|y\rangle \to (-1)^{f(x)}|x\rangle|y\rangle, \tag{18.70}$$

ahol

$$f(x) = \begin{cases} 1 & \text{ha } x = x_0 \text{ (vagyis DB}[x_0] \text{ megegyezik a keresett bejegyzéssel),} \\ 0 & \text{egyébként.} \end{cases} \tag{18.71}$$

Az orákulum kapu operátora a keresett x_0 állapot függvényében:

$$O = I - 2|x_0\rangle\langle x_0|. \tag{18.72}$$

Egy ilyen kapu hatását bináris műveletekkel a következőképpen fogalmazhatunk meg:

$$O : |x\rangle|y\rangle \to |x\rangle|y \oplus f(x)\rangle, \tag{18.73}$$

ahol $\oplus$ a modulo 2 szerinti összeadás, amely egy szuperpozíció minden egyes tagjára külön-külön hat.

Esetünkben az orákulum kapu bemeneti állapota:

$$|\varphi_1\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle \otimes \frac{|0\rangle - |1\rangle}{\sqrt{2}}, \tag{18.74}$$

a kimeneti állapot pedig (18.70) felhasználásával:

$$|\varphi_2\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} (-1)^{f(x)}|x\rangle \otimes \frac{|0\rangle - |1\rangle}{\sqrt{2}} = |\gamma_2\rangle \otimes \frac{|0\rangle - |1\rangle}{\sqrt{2}}. \tag{18.75}$$

Átlagra való tükrözés

A Grover-operátoron belül a második lépés a kívánt eredményt adó $|x_0\rangle$ méréshez tartozó valószínűségek megnövelése, és a szuperpozíció összes többi állapotának csökkentése. Az előző részben láttuk, hogy az orákulum kapu megjelölte a keresett állapotot egy negatív előjellel. Fontos megjegyeznünk, hogy ezzel ugyan a súlyfaktorok átlaga is megváltozik (méghozzá kicsit kisebb lesz, mint az egyenletes eloszlás esetén), de az egyes súlyfaktorok abszolút értéke nem változik. Márpedig, ha a mérési valószínűségeket akarjuk elnyomni vagy erősíteni, épp az abszolút érték változására lenne szükségünk. Ennek egy módja lehet, ha a szuperpozíció minden súlyozott vektorát tükrözzük az átlagra. Mivel feltesszük, hogy a keresett bejegyzés ritkán fordul elő az

adatbázisban, az átlag továbbra is közel lesz a jelöletlen állapotokhoz, de távol lesz a jelöltekétől. Ebből következik, hogy a tükrözés a jelöletlen állapotok valószínűségét kicsit csökkenti, míg a jelöltekét jelentősen növeli. Ennek az eljárásnak a neve ***átlagra való tükrözés,*** amelyet most részletesen is megvizsgálunk.

A $|\gamma_2\rangle$ állapot esetén a valószínűségi amplitúdók $\overline{a}$ átlaga:

$$\overline{a} = \frac{1}{N}\sum_{x=0}^{N-1}\gamma_{2x}, \tag{18.76}$$

ahol γ_{2x} jelöli az $|x\rangle$ bázisvektorhoz tartozó valószínűségi amplitúdót a $|\gamma_2\rangle$ szuperpozícióban:

$$|\gamma_2\rangle = \frac{1}{\sqrt{N}}\sum_{x=0}^{N-1}(-1)^{f(x)}|x\rangle = \sum_{x=0}^{N-1}\gamma_{2x}|x\rangle. \tag{18.77}$$

Bár még nem tudjuk, hogyan lehet ezt megvalósítani, de ha minden állapotvektort tükrözni tudnánk $\overline{a}$-ra, akkor a következő amplitúdókat kapnánk:

$$\gamma_{3x} = 2\overline{a} - \gamma_{2x},$$

amivel

$$|\gamma_3\rangle = \sum_{x=0}^{N-1}(2\overline{a} - \gamma_{2x})|x\rangle = 2\sum_{x=0}^{N-1}\overline{a}|x\rangle - \sum_{x=0}^{N-1}\gamma_{2x}|x\rangle. \tag{18.78}$$

(18.78) jobb oldalán, a második összeg pedig nem más, mint $|\gamma_2\rangle$ (lásd (18.76) és (18.77)). Vizsgáljuk meg közelebbről az első összeget is. Ehhez írjuk fel a $\langle\gamma_1|\gamma_2\rangle$ skalárszorzatot. Felhasználva a skalárszorzatban szereplő vektorok (18.69) és (18.77) definícióit, az átlag (18.76) képletét, valamint azt a tényt, hogy az $|x\rangle$ bázisvektorok ortonormált rendszert alkotnak, beláthatjuk, hogy:

$$\langle\gamma_1|\gamma_2\rangle = \frac{1}{\sqrt{N}}\sum_{x=0}^{N-1}\gamma_{2x} = \sqrt{N}\overline{a}.$$

Ebből látható, hogy (18.78) jobb oldalán az első szumma nem más, mint $2|\gamma_1\rangle\langle\gamma_1||\gamma_2\rangle$, így $|\gamma_3\rangle$:

$$|\gamma_3\rangle = 2|\gamma_1\rangle\langle\gamma_1||\gamma_2\rangle - |\gamma_2\rangle.$$

Ebből már leolvashatjuk az átlagra való tükrözés operátorát, hiszen ahhoz, hogy $|\gamma_2\rangle$-ből $|\gamma_3\rangle$-at állítsunk elő, a következő operátorra van szükség:

$$U_\gamma = 2|\gamma_1\rangle\langle\gamma_1| - I. \tag{18.79}$$

Figyelembe véve, hogy $|\gamma_1\rangle = H|\mathbf{0}\rangle$, $H = H^\dagger$ és $HIH \equiv I$, (18.79) tovább alakítható

$$U_\gamma = 2H|\mathbf{0}\rangle\langle\mathbf{0}|H - HIH = H(2|\mathbf{0}\rangle\langle\mathbf{0}| - I)H. \tag{18.80}$$

(18.80) fontos információkkal szolgál arról, hogyan lehet megvalósítani az átlagra való tükrözést elemi kapuk segítségével. Ennek alapján két Hadamard-kapura van szükség, melyek közrefognak egy P *vezérelt fázistolót*, úgy, ahogy az 18.13. ábrán látható. A P kapu transzformációs szabálya meglehetősen egyszerű, minden valószínűségi amplitúdót változatlanul hagy kivéve a $|\mathbf{0}\rangle$ állapothoz tartozót, melynek megváltoztatja az előjelét.

Megjegyzés: most, hogy már ismerjük az átlagra való tükrözés operátorát, feltehetjük a kérdést, hogy P unitér-e egyáltalán. A válasz igen, hiszen P-t sikerült unitér elemi operációk szorzatára visszavezetni.

Most már készen állunk, hogy megalkossuk a teljes G Grover-operátort:

$$G = HPHO = (2H|\mathbf{0}\rangle\langle\mathbf{0}|H - I)O = (2|\gamma_1\rangle\langle\gamma_1| - I)(I - 2|x_0\rangle\langle x_0|). \tag{18.81}$$

Végül megemlítjük, hogy ha egynél több bejegyzés megegyezik az adatbázisunkban a keresett állapottal, akkor a Grover-algoritmus ezeket mind azonos valószínűséggel adja vissza, hiszen az áramkör nem tesz különbséget a jelölt állapotok között.

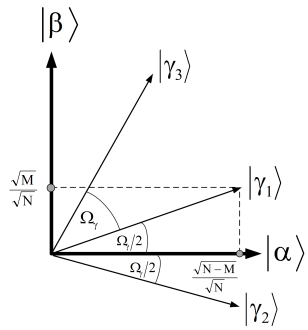
A szükséges iterációk száma

A Grover-operátor egyszeri alkalmazása megnöveli ugyan annak a valószínűségét, hogy a keresett állapotokat mérjük, de – mint azt látni fogjuk – sokkal jobb eredményt érhetünk el, ha a P kapu kimenetét visszacsatoljuk egy következő P kapu bemenetére, és iteratívan többször megismételjük a Grover-operátort. Nézzük meg, hogy hány iterációs lépésre van szükség ahhoz, hogy γ_{3x_0} valószínűsége olyan közel legyen 1-hez, amilyen közel csak lehetséges. Ennek vizsgálatához tegyük fel, hogy több jelölt bejegyzésünk van az adatbázisban ($M \geq 1$).

Először alkossunk két halmazt a bázisvektorok indexeiből. Az S halmaz elemei legyenek a jelölt bázisvektorok indexei (melyekre $f(x) = 1$), a $\overline{S}$ halmaz pedig tartalmazza a jelöletlen ($f(x) = 0$) állapotvektorok indexeit. Ezek segítségével két szuperpozíciót írhatunk le, az egyik az összes jelölt, a másik az összes jelöletlen állapotvektort tartalmazza egyenlő valószínűséggel:

$$|\alpha\rangle \triangleq \frac{1}{\sqrt{N-M}} \sum_{x\in\overline{S}} |x\rangle, \tag{18.82}$$

$$|\beta\rangle \triangleq \frac{1}{\sqrt{M}} \sum_{x\in S} |x\rangle. \tag{18.83}$$



18.14. ábra. A Grover-operátor geometriai megjelenítése.

$|\alpha\rangle$ és $|\beta\rangle$ ortonormált bázist alkot, amely egy 2-dimenziós Hilbert teret feszít ki. A tér vázlatos rajza a 18.14-es ábrán látható.

A G kapu bemeneti $|\gamma_1\rangle$ állapota ebben a bázisban úgy írható fel, mint:

$$\begin{aligned} |\gamma_1\rangle &= \frac{1}{\sqrt{N}} \sum_{x \in \overline{S}} |x\rangle + \frac{1}{\sqrt{N}} \sum_{x \in S} |x\rangle, \\ &= \sqrt{\frac{N-M}{N}} |\alpha\rangle + \sqrt{\frac{M}{N}} |\beta\rangle. \end{aligned} \tag{18.84}$$

Tegyük fel, hogy $|\alpha\rangle$ és $|\beta\rangle$ vektorunk is létezik, vagyis egyik halmaz sem üres.

Ebben a térben $|\gamma_1\rangle$ koordinátái szorosan kötődnek a $|\gamma_1\rangle$ és $|\alpha\rangle$ közti szöghöz, melyet a 18.14 ábrán $\frac{\Omega_\gamma}{2}$ jelöl. $|\gamma_1\rangle$ vetületét a tengelyekre meghatározhatjuk alapvető trigonometriai összefüggések segítségével

$$\cos\left(\frac{\Omega_\gamma}{2}\right) = \frac{\sqrt{\frac{N-M}{N}}}{1},$$

$$\sin\left(\frac{\Omega_\gamma}{2}\right) = \frac{\sqrt{\frac{M}{N}}}{1} \Rightarrow \Omega_\gamma = 2 \arcsin\left(\sqrt{\frac{M}{N}}\right), \tag{18.85}$$

ahol a nevezőben álló 1-esek azt jelzik, hogy $|\gamma_1\rangle$ egységnyi hosszú. Több jelölt állapot esetén az orákulum a következőképpen definiálható:

$$O = I - 2 \sum_{x \in S} |x\rangle\langle x|. \tag{18.86}$$

Ez minden jelölt valószínűségi amplitúdó előjelét megváltoztatja, amit

tekinthetünk egy $|\alpha\rangle$ tengelyre való tükrözésnek (lásd 18.14. ábra). Tehát

$$O(a|\alpha\rangle + b|\beta\rangle) = a|\alpha\rangle - b|\beta\rangle.$$

Az átlagra való tükrözés HPH operátora a $|\gamma_2\rangle$ állapotot tükrözi $|\gamma_1\rangle$-re. Ez a két tükrözés együtt egy forgatást eredményez, mely $|\gamma_1\rangle$ felől $|\beta\rangle$ felé forgat Ω_γ szöggel. Ebben a geometriai interpretációban úgy fogalmazhatjuk meg célunkat, hogy az indexregiszterünk $|\gamma_1\rangle$ állapotát olyan közel szeretnénk forgatni $|\beta\rangle$-hoz, amilyen közel csak lehetséges. Ha ez sikerült, az eredeti n-dimenziós bázisában végrehajtott mérés nagy valószínűséggel a jelölt állapotok egyikét adja vissza.

A Grover-operátor egyszeri hatása után a kezdeti $|\gamma_1\rangle$ állapot a következőképpen változik:

$$G^l|\gamma_1\rangle = \cos\left(\frac{2l+1}{2}\Omega_\gamma\right)|\alpha\rangle + \sin\left(\frac{2l+1}{2}\Omega_\gamma\right)|\beta\rangle. \tag{18.87}$$

A mérést akkor érdemes elvégezni, ha $G^l|\gamma_1\rangle$ egyenlő a $|\beta\rangle$ bázisvektorral, vagy másképp megfogalmazva, ha merőleges az $|\alpha\rangle$ vektorra, vagyis

$$\langle\alpha|G^l|\gamma_1\rangle = \cos\left(\frac{2l+1}{2}\Omega_\gamma\right) = 0.$$

Ez akkor igaz, ha

$$\frac{2l+1}{2}\cdot\Omega_\gamma = \frac{\pi}{2} + i\pi,$$

ahol $i = 0, 1, \ldots$. Így az iterációk optimális száma

$$l_{opt_i} = \frac{\frac{\pi}{2} + i\pi - \frac{\Omega_\gamma}{2}}{\Omega_\gamma}. \tag{18.88}$$

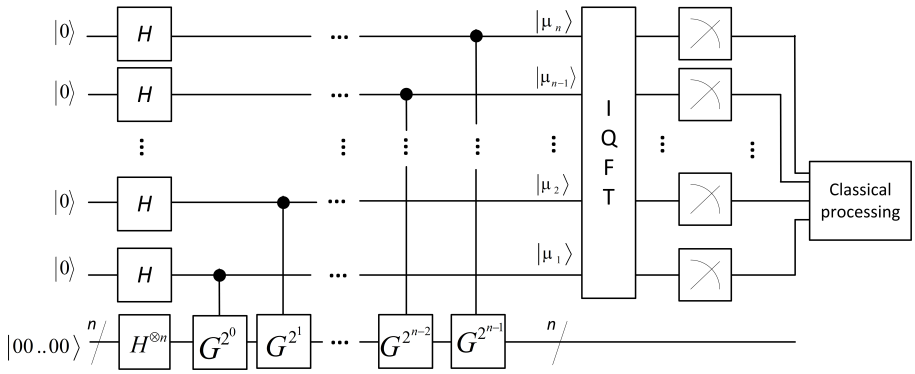
Természetesen igyekszünk olyan kevés iterációt végezni, amilyen keveset csak lehetséges: $\min_i l_{opt_i} = l_{opt_0}$. Továbbá figyelembe kell vennünk, hogy csak egész számú iterációt lehet végezni, vagyis:

$$L_{opt_0} = \lfloor l_{opt_0} \rceil = \left\lfloor \frac{\frac{\pi}{2} - \frac{\Omega_\gamma}{2}}{\Omega_\gamma} \right\rceil. \tag{18.89}$$

Praktikus estekben az adatbázis mérte lényegesen nagyobb, mint a jelölt állapotok száma $M \ll N$. Ebből következik, hogy $\Omega_\gamma \ll 1$, vagyis

$$\frac{\Omega_\gamma}{2} \simeq \sin\left(\frac{\Omega_\gamma}{2}\right) = \sqrt{\frac{M}{N}}, \tag{18.90}$$

(18.90)-et visszahelyettesítve (18.89)-be egy meglepő és érdekes eredményre



18.15. ábra. Kvantumkereső rendszerünk áramköri vázlata.

jutunk:

$$L_{opt_0} = \left\lfloor \frac{\pi}{4}\sqrt{\frac{N}{M}} - 1 \right\rfloor \simeq \frac{\pi}{4}\sqrt{\frac{N}{M}}. \tag{18.91}$$

Más szóval az iterációk száma $L_{opt_0} = O\left(\sqrt{\frac{N}{M}}\right)$, ami jól mutatja a kvantumkeresés magas hatásfokát, hiszen ez négyzetesen hatékonyabb mint a legjobb klasszikus algoritmus.

Kvantum-számlálás

Előfordulhat, hogy nem ismerjük a keresett bejegyzés előfordulásának M számát. Ebben az esetben egy módosított fázisbecslést alkalmazhatunk ahhoz, hogy megszámoljuk a jelölt elemeket. Nézzük meg ezt most közelebbről.

A (18.90) összefüggésből láthatjuk, hogy M és Ω_γ között szoros kapcsolat van, éppen ezért elsősorban Ω_γ-ra leszünk kíváncsiak. Ha a Grover-operátor mátrixát felírjuk az $|\alpha\rangle$ és $|\beta\rangle$ bázisban, ebben az Ω_γ természetesen megjelenik:

$$\mathbf{G} = \begin{bmatrix} \cos(\Omega_\gamma) & -\sin(\Omega_\gamma) \\ \sin(\Omega_\gamma) & \cos(\Omega_\gamma) \end{bmatrix}.$$

Megmutatható, hogy G-nek két sajátértéke van, nevezetesen $e^{\pm j\Omega_\gamma}$. Ebben a sajátértékben Ω_γ még mindig ismeretlen, de emlékezzünk vissza a korábbi fejezetben a fázisbecslésre (18.11-es ábra), mellyel meghatároztuk egy ismeretlen operátor adott sajátértékéhez tartozó fázist. Ha ebbe az áramkörbe behelyettesítjük az $U = G$ kaput úgy, ahogy az a 18.15 ábrán látható, meghatározhatjuk Ω_γ-t.

A fázisbecslés működéséhez azonban szükségünk van a sajátvektorokra is, hogy az alsó, vezérelt kvantumregiszter bemenetét inicializálni tudjuk. Ezeket

elméletben könnyű meghatározni G mátrixából, ha $|\alpha\rangle$ és $|\beta\rangle$ bázisában van felírva:

$$|g_1\rangle = \frac{e^{j\xi}}{\sqrt{2}}\begin{bmatrix} j \\ 1 \end{bmatrix}, |g_2\rangle = \frac{e^{j\xi}}{\sqrt{2}}\begin{bmatrix} -j \\ 1 \end{bmatrix}, \xi \in \mathbb{R},$$

de sajnos a gyakorlatban $|g_1\rangle$ és $|g_2\rangle$ vektorok egyikét sem használhatjuk külön-külön bemenetként, hiszen ezek előállításához ismernünk kellene $|\alpha\rangle$ és $|\beta\rangle$ állapotokat, ami azt jelentené, hogy tökéletesen ismerjük az összes jelölt és jelöletlen bejegyzést az adatbázisban.

Ha viszont $|\alpha\rangle$ és $|\beta\rangle$ egy szuperpozícióját alkalmazzuk az alsó regiszter bemeneteként, akkor a felső regiszteren végrehajtott mérés eredménye véletlenszerűen valamelyik sajátérték lesz, de nem tudjuk, hogy melyik. Szerencsére ebben az esetben csak két sajátértékünk van: Ω_γ és $-\Omega_\gamma = 2\pi - \Omega_\gamma$, ezeket pedig könnyű megkülönböztetni, mert $\Omega_\gamma \ll \frac{\pi}{4}$. (Mint azt korábban részleteztük, ha a jelölt állapotok száma az összes bejegyzéshez képest kicsi, akkor Ω_γ is kicsi.)

Így tehát Ω_γ-át kiszámíthatjuk a mérés eredményéből minden különösebb előismeret nélkül. Az egyszerűség kedvéért használjuk $|\gamma_1\rangle$ állapotvektort az alsó regiszter inicializálására, hiszen (18.84)-ben láttuk, hogy ez kifejezhető $|\alpha\rangle$ és $|\beta\rangle$ szuperpozíciójaként. Mivel $|g_1\rangle$ és $|g_2\rangle$ ortonormált bázist alkot a térben, melyet $|\alpha\rangle$és $|\beta\rangle$ feszít ki, így $|\gamma_1\rangle$ kifejezhető $|g_1\rangle$ és $|g_2\rangle$ szuperpozíciójaként is.

Végül még beszélnünk kell egy keveset a felső kvantumregiszter méretéről. Azért, hogy elkerüljük a keveredést, ezt jelöljük n_s-sel. Amint azt korábban láttuk, n_s annak függvényében, hogy Ω_γ-t milyen 2^{-c} klasszikus pontossággal és $\breve{P}_{\varepsilon P}$ kvantumos bizonytalansággal akarjuk meghatározni:

$$n_s = c - 1 + \left\lceil \mathrm{ld}(2\pi) + \mathrm{ld}\left(3 + \frac{1}{\breve{P}_{\varepsilon P}}\right)\right\rceil . \tag{18.92}$$

Gyakorlatok

18.6-1. A Grover-algoritmust a következő kezdeti paraméterekkel szeretnénk futtatni: $N = 8$, $M = 1$, $x_0 = 4$. Határozzuk meg O és G mátrixokat.

18.6-2. Tegyük fel, hogy az adatbázisunk negyede jelölt: $M = N/4$. Ha egyetlen iterációt végzünk a Grover-algoritmussal, mekkora az esélye annak, hogy a mérés után egy jelölt elemet kapunk eredményül?

Feladatok

18-1 Kvantum-komplexitás

Meg tudják-e oldani a kvantumszámítógépek az NP-teljes problémákat?

18-2 Összefonódás a környezettel
Nézzünk utána a tudományos szakirodalomban, hogy mi lesz a környezettel való összefonódás hatása egy kvantumregiszterre? (A kapcsolódó jelenség neve dekoherencia, angol szakkifejezéssel decoherence.) Létezik valamiféle a módszer a kvantumos „szigetelésre" és a jelenség kiküszöbölésére?

18-3 Tetszőleges szuperpozíció létrehozása
Különböző elemi kvantumkapuk felhasználásával tervezzünk olyan áramkört, amely $|0\rangle$ bemenetből előállítja a következő kimenetet:$|\phi\rangle = a|0\rangle + b|1\rangle$.

18-4 Fourier-mintavételezés
Tervezzünk programot, mely végrehajtja a Fourier-mintavételezéstl egy klasszikus n-bites bemenetre.

18-5 Shor-algoritmus
Tervezzünk és írjunk egy programot, ami a Shor-algoritmust szimulálja.

18-6 Egy különös adatbázis
Tegyük fel, hogy adatbázisunkban a bejegyzéseknek több mint a fele jelölt állapot. Hogyan fog működni a Grover-operátor? Hogyan érdemes jelölt elemeket keresni egy ilyen adatbázisban?

Megjegyzések a fejezethez

A kvantumalgoritmusok számos érdekes megoldást kínálnak klasszikus problémákra. Ezek tanulmányozásához egy jó kezdőpont lehet Imre Sándor és Balázs Ferenc könyve [127], mely az alapvető algoritmusokat tárgyalja mérnöki szemszögből. A könyv második kötete Imre Sándor és Gyöngyösi László tollából [128] a kvantum-információelméletet tekinti át mérnöki megközelítésben. A szakirodalom egyik megalapozó műve Michael Nielsen és Isaac Chuang kézikönyve is [216], míg Bacsárdi László [14] könyve a világűrben zajló kvantumkommunikáció elméletébe enged betekintést.

A Bernstein-Vazirani-algoritmust eredeti formájában a [26] cikkben találja meg az olvasó, míg a Deutsch-Jozsa-algoritmus a [63] cikkben található (illetve ennek későbbi, továbbfejlesztett változata itt: [230]). A Simon-algoritmus leírása itt található: [270]. A Shor-algoritmus eredeti leírása megtekinthető a [268] cikkben, míg Grover eredeti munkája itt olvasható: [108]. Számos további hivatkozás és újdonság található a quant-ph honlapon [6] is.

A szerzők szívesen értékelik az Olvasók megoldásait a fejezetben lévő gyakorlatokra és feladatokra, a megoldásokat a quant-course@mlc.hu e-mail címen várják.

19. Osztott algoritmusok

Osztott rendszernek nevezzük az egymással kommunikáló önálló számító eszközöket. Ez a meghatározás nagyon tág, magában foglalja a VLSI áramköröket, a többprocesszoros rendszereket, a helyi hálózattal klaszterbe kötött munkaállomásokat és az Internetet. Itt most a lazábban összekötött rendszerekre koncentrálunk. Úgy tekintjük, hogy egy osztott rendszerben az egyes processzorok lényegében függetlenek, de bizonyos okokból – ilyenek például az erőforrás-megosztás, a hozzáférhetőség, a hibatűrés – összhangba kell hozniuk a tevékenységüket.

Bár hatalmas igény mutatkozik az osztott rendszerekre, közismerten nehéz olyan hatékony osztott algoritmusokat készíteni, melyek jól teljesítenek valódi rendszereken. A nehézségek nem csak gyakorlati természetűek, hanem elméletiek is. Nevezetesen, a következő három tényező sok problémát okoz: aszinkronitás, korlátozott helyi információk, meghibásodások. Az aszinkronitás azt jelenti, hogy nem feltétlenül áll rendelkezésre egy globális idő, valamint, hogy az egyes számító eszközökön bekövetkező események abszolút és relatív időpontjai gyakran nem pontosan ismertek. Továbbá, az egyes számító eszközök csak a kapott információkat ismerik, így csak helyi nézetük van a rendszer globális állapotáról. Végül, a számító eszközök és a hálózati komponensek egymástól függetlenül meghibásodhatnak, azaz némelyek működőképesek maradnak, míg mások nem.

Az osztott rendszerek elemzésére használt modellek leírását a számítás üzenetátadási modelljével kezdjük. Bemutatunk és elemzünk néhány, ezeken a modelleken alapuló, osztott algoritmust. Tárgyaljuk a hibatűrést az osztott rendszerekben és megvizsgálunk néhány olyan algoritmust, mely megoldást jelent az üzenetátadási modellekben a hibák kezelésére. Mivel az osztott rendszerekben a globális idő gyakran nem elérhető, bemutatunk néhány megközelítést az ún. logikai idő meghatározására, mely lehetővé teszi az okozati viszony és az ellentmondásmentes állapotok megállapítását. Ezután haladóbb témákat veszünk górcső alá. Bemutatjuk azokat az üzenetszóró szolgáltatásokat, melyeket az osztott rendszerek gyakran használnak, és bemutatjuk azokat az algoritmusokat, melyek ezeket a szolgáltatásokat megvalósítják. Bemutatunk néhány információgyűjtő algoritmust is. Végül az osztott számítások közös memóriát használó modelljének kölcsönös kizárási problémáját tárgyaljuk.

19.1. Üzenetküldő rendszerek és algoritmusok

Az osztott számítás első modelljeként az üzenetküldő rendszerek meghibásodás nélküli üzenetküldési modelljét tárgyaljuk. Figyelembe vesszük mind a szinkron, mind az aszinkron rendszereket, és bemutatunk néhány kiválasztott algoritmust, amelyek tetszőleges hálózati topológiájú üzenetküldő rendszerekben alkalmazhatók – mind szinkron, mind pedig aszinkron esetekben.

19.1.1. Üzenetküldő rendszerek modellezése

Egy üzenetküldő rendszerben a processzorok[1] kommunikációs csatornákon keresztül küldött üzenetekkel kommunikálnak, ahol az egyes csatornák kétirányú kapcsolatot biztosítanak a két kérdéses processzor között. A csatornákkal meghatározott kapcsolódási mintát a rendszer ***topológiájának*** nevezzük. Ezt a topológiát egy irányítatlan gráffal ábrázoljuk, ahol az egyes csúcsok egy processzort jelentenek, és akkor és csak akkor van egy él két csúcspont között, ha van egy csatorna a csúcspontokkal ábrázolt processzorok között. A csatornák összességét ***hálózatnak*** is nevezzük. Egy adott topológiájú üzenettovábbító rendszer algoritmusa a rendszer egyes processzoraira vonatkozó helyi programból áll. Ez a helyi program lehetővé teszi, hogy a processzor helyi számításokat végezzen és az adott topológia melletti szomszédainak üzeneteket küldjön, illetve azoktól üzeneteket kapjon.

A rendszer egyes processzorait egy-egy olyan automatával modellezzük, amely végtelen is lehet. ***Konfigurációnak*** nevezünk egy $C = (q_0, \ldots, q_{n-1})$ vektort, ahol a q_i-k, az egyes P_i processzorok állapotai. A rendszerben zajló tevékenységeket oszthatatlan rendszerműveleteket leíró ***eseményeknek*** (vagy ***műveleteknek***) nevezzük. Esemény például a helyi számítási esemény vagy az olyan kézbesítési esemény, ahol egy processzor egy üzenetet kap. A rendszer időbeli működését egy ún. ***végrehajtási sorozattal*** (röviden: végrehajtás) modellezzük, ami C_i konfigurációkat és a_i eseményeket váltakozva tartalmazó, véges vagy végtelen sorozat: $C_0, a_1, C_1, a_2, C_2, \ldots$. A rendszer modelljétől függően, a végrehajtásnak meg kell felelnie számos, helyességi tulajdonságokat leíró feltételnek. E feltételeket vagy az ún. *it biztonsági* vagy az ún. *élőségi* osztályokba sorolhatjuk. Egy rendszer ***biztonsági*** feltétele egy olyan feltétel, melynek igaznak kell lennie a rendszer bármely eseménye előtt. Informálisan ez azt jelenti, hogy még semmi *rossz* nem történt. Egy ***létezési*** feltétel egy olyan feltétel, melynek igaznak kell lennie számos (valószínűleg végtelen) alkalommal. Informálisan ez azt jelenti, hogy végül is valami *jó* be fog következni. Egy fontos létezési feltétel a ***tisztaság,*** mely megköveteli,

[1] A rendszer elemeit a továbbiakban processzoroknak tekintjük. *A lektor.*

hogy egy (végtelen) végrehajtás végtelen sok processzorműveletet tartalmazzon, hacsak néhány konfiguráció után az adott processzoron nincs engedélyezett művelet.

19.1.2. Aszinkron rendszerek

Egy rendszerre akkor mondjuk, hogy ***aszinkron,*** ha egy üzenet kézbesítési idejének vagy egy processzor egymás után tett lépései közti időnek nincs rögzített felső korlátja. Az aszinkron rendszer nyilvánvaló példája az Internet. Egy osztott rendszer megvalósításában gyakran létezik felső korlát az üzenetek kézbesítési idejére és a processzorlépések idejére. Mivel ezek a felső korlátok gyakran nagyon nagyok és időben változnak, gyakran kívánatos egy olyan algoritmus kifejlesztése, mely független az időzítési paraméterektől, vagyis ***aszinkron***..

Az aszinkron modellben egy végrehajtás ***elfogadható,*** ha az egyes processzorok végtelen számú számítási eseménnyel rendelkeznek, és az összes küldött üzenet végül kézbesítődik. Az első követelmény modellezi azt a tényt, hogy a processzorok nem hibásodnak meg. (Ez nem azt jelenti, hogy a processzor helyi programja végtelen ciklust tartalmaz. Egy algoritmus még mindig megállhat, ha olyan az átmenetfüggvénye, amely egy bizonyos pont után nem változtatja meg a processzor állapotát.)

Feltesszük, hogy minden processzor állapothalmaza tartalmaz egy olyan részhalmazt, mely a *megállt* állapotokat tartalmazza. Azt mondjuk, hogy az algoritmus ***megállt***, ha az összes processzor megállt állapotban van, és nincs kézbesítés alatt álló üzenet.

Az aszinkron modellben egy algoritmus ***üzenetszáma*** az elfogadható végrehajtási sorozatokban csúcsponttól csúcsponthoz küldött üzenetek számának maximuma.

Az ***időzített végrehajtás*** olyan végrehajtás, amelynél minden eseményhez egy nemnegatív valós számot rendelünk, az esemény bekövetkezésének ***iidőpontját***. Egy aszinkron algoritmus ***futási idejének*** méréséhez először feltesszük, hogy bármely végrehajtás esetén az üzenetek kézbesítési ideje egy időegység. Így a futási idő az a maximális idő, amíg az összes olyan időzített elfogadható végrehajtás megáll, ahol a üzenetek kézbesítési ideje legfeljebb egy. Intuitív módon ezt úgy tekinthetjük, hogy vesszük az algoritmus bármelyik végrehajtását, és úgy normalizáljuk, hogy a leghosszabb kézbesítési időt tekintjük egy időegységnek.

19.1.3. Szinkron rendszerek

A szinkron modellben a processzorok zárt lépéseket végeznek. A végrehajtást olyan menetekre bontjuk, ahol az egyes processzorok egy-egy üzenetet küldhetnek szomszédaiknak, az üzenetek megérkeznek, és minden processzor az épp kapott üzeneteket alapul véve számol. Ez a modell nagyon kényelmes az algoritmusok tervezése során. Az ebben a modellben tervezett algoritmusok sok esetben könnyen szimulálhatók úgy, hogy más, a valóságot jobban tükröző időzítési modellben is működjenek.

A szinkron modellben azt mondjuk, hogy egy végrehajtás elfogadható, ha az végtelen. A menet alapú struktúrából az következik, hogy minden processzor végtelen sok számítási lépést tesz, és minden egyes üzenet valamikor megérkezik. Így egy hiba nélküli szinkron rendszerben ha a (determinisztikus) algoritmus rögzítésre került, az egyetlen megváltoztatható szempont egy végrehajtás meghatározásánál a kezdőkonfiguráció. Másrészt egy aszinkron rendszerben ugyanannak az algoritmusnak több, különböző végrehajtása lehet még azonos kezdőkonfiguráció és hibamentesség esetén is, mivel itt a processzorlépések egymásutánisága és az üzenetek késése nem rögzített.

A ***megállt állapot*** fogalma és az algoritmus ***megállása*** eltér az aszinkron modellétől.

A szinkron modellben az algoritmus ***üzenetszáma*** ugyanúgy az összes küldött üzenet számának maximuma az algoritmus összes elfogadható végrehajtása esetén, mint az aszinkron modellben.

Szinkron esetben az idő méréséhez egyszerűen összeszámoljuk a megállásig megtett menetek számát. Így a szinkron modellben egy algoritmus futási ideje az algoritmus összes elfogadható végrehajtásainak megállásig megtett menetei számának maximuma.

19.2. Alapvető algoritmusok

Az üzenetküldő modell néhány egyszerű algoritmusával kezdjük.

19.2.1. Üzenetszórás

Egy, az (együzenetes) üzenetszórási problémára vonatkozó egyszerű algoritmussal kezdjük. Feltesszük, hogy az n csúcsú hálózatgráf feszítőfája már adott. Később majd eltekintünk ettől a feltevéstől. A P_r processzor egy M üzenetet kíván küldeni az összes többi processzornak. A P_r gyökerű feszítőfa osztott módon van karbantartva: Minden egyes processzor rendelkezik egy megkülönböztetett csatornával a fabeli *szülője* felé, illetve csatornák egy halmazával a fabeli *gyerekei* felé. A P_r gyökér az M üzenetet elküldi a gyerekei

felé vezető összes csatornán. Amikor egy processzor megkapja az M üzenetet egy csatornán a szülőjétől, akkor azt elküldi az összes gyerekének.

FESZÍTŐFA-ÜZENETSZÓRÓ

Kezdetben M útban van P_r-től az összes feszítőfabeli gyereke felé.
Kód P_r esetén:
1 üres üzenet fogadásakor: // P_r első számítási eseménye
2 leállás

Kód P_j, $0 \leq j \leq n-1$, $j \neq r$ esetén:
3 M szülőtől való fogadásakor:
4 M küldése az összes gyerekhez
5 leállás

A FESZÍTŐFA-ÜZENETSZÓRÓ algoritmus helyes akár szinkron akár aszinkron rendszerről van szó. Továbbá, az üzenetszámok és a futási idők is azonosak mindkét modellben.

Egyszerű induktív bizonyítással először bebizonyítunk egy lemmát, amely szerint a t-edik menet végén az M eléri az összes, a feszítőfában P_r-től t (vagy rövidebb) távolságra lévő processzort.

19.1. lemma. *A szinkron modellben az üzenetszóró algoritmus minden elfogadható végrehajtási sorozatában az összes, a feszítőfában P_r-től t távolságra lévő processzor megkapja az M üzenetet a t-edik menetben.*

Bizonyítás. A tetszőleges P_r processzortól számított t távolság szerinti indukciós feltétellel fogunk haladni. Először legyen $t = 1$. Az algoritmusból következik, hogy P_r minden egyes gyereke megkapja az üzenetet az első menetben.

Tegyük fel, hogy minden $t-1$ távolságra lévő processzor megkapta az M üzenetet a $(t-1)$-edik menetben. Meg kell mutatnunk, hogy minden P_t processzor, mely t távolságra van, megkapja az üzenetet a t-edik menetben. Legyen P_s P_t szülője a feszítőfában. Mivel P_s $t-1$ távolságra van P_r-től, az indukciós feltétel szerint P_s megkapta az M üzenetet a $(t-1)$-edik menetben. Az algoritmus alapján így P_t megkapja M-et a t-edik menetben. ∎

A 19.1. lemma alapján az üzenetszóró algoritmus futási ideje d, ahol d a feszítőfa magassága. Mivel d legfeljebb $n-1$ (amikor a feszítőfa egy lánc), azt kapjuk, hogy:

19.2. tétel. *Ha egy d magasságú gyökeres feszítőfa előre ismert, n procesz-*

szor esetén létezik egy szinkron üzenetszóró algoritmus, melyre az üzenetszám $n-1$ *és a futási idő* d.

Most megvizsgáljuk az aszinkron rendszert és egy hasonló elemzést végzünk.

19.3. lemma. *Az aszinkron modellben az üzenetszóró algoritmus minden egyes elfogadható végrehajtása esetén minden –* P_r*-től a feszítőfában* t *távolságra lévő – processzor megkapja az* M *üzenetet a* t *időpontig.*

Bizonyítás. A P_r processzortól számított t távolság szerinti indukcióval fogunk eljárni. Az algoritmusból következik, hogy M kezdetben útban van P_r-től 1 távolságra lévő P_i processzorok felé. Az aszinkron modell futási idejének meghatározása szerint P_i megkapja az M üzenetet az 1 időpontig.

Tegyük fel, hogy minden, $t-1$ távolságra lévő processzor megkapta az üzenetet a $t-1$ időpillanatban. Meg kell mutatnunk, hogy minden t távolságra lévő P_t processzor megkapja az üzenetet a t időpontig. Legyen P_s P_t szülője a feszítőfában. Mivel P_s $t-1$ távolságra van P_r-től, az indukciós feltevés miatt, P_s M-et továbbküldi P_t-nek, amikor a $t-1$ időpontban megkapja azt. Az algoritmus szerint P_t így megkapja M-et a t időpontig. ∎

Ebből közvetlenül kapjuk a következő tételt:

19.4. tétel. *Ha egy* d *magasságú gyökeres feszítőfa előre ismert,* n *processzor esetén létezik egy aszinkron üzenetszóró algoritmus, melyre az üzenetszám* $n-1$ *és a futási idő* d.

19.2.2. A feszítőfa megkonstruálása

A következőkben tárgyalt ELÁRASZTÁS nevű aszinkron algoritmus felépíti a kijelölt P_r csúcsnál lévő gyökerű feszítőfát. Az algoritmus hasonlít a MÉLYSÉGI-KERESÉS (MK) algoritmushoz. Az MK algoritmustól eltérően azonban, ahol csak egyetlen processzor rendelkezik „globális ismerettel" a fáról, az ELÁRASZTÁS algoritmusban minden egyes processzornak „helyi ismerete" van a gráfról, a processzorok a munkájukat üzenetküldéssel koordinálják és a processzorok, valamint az üzenetek tetszőlegesen sokat késhetnek. Mindezek miatt az ELÁRASZTÁS algoritmus megtervezése és elemzése igazi kihívás, mivel meg kell mutatnunk, hogy az algoritmus ténylegesen felépít egy feszítőfát a késések kedvezőtlen összejátszása esetén is.

Az algoritmus leírása

Minden egyes processzor négy helyi változóval rendelkezik.

Az egy processzorhoz kapcsolódó összeköttetéseket 1-től kezdődő egyedi

számokkal azonosítjuk, és egy *szomszédok*-nak nevezett helyi változóban tároljuk. Azt mondjuk, hogy a *feszítőfát felépítettük,* ha a *szülő* változó a processzor feszítőfabeli szülőjére mutató összeköttetés azonosítószámát tartalmazza, kivéve, ha a P_r kijelölt processzorról van szó. Ez utóbbi esetben a *szülő* értéke NINCS. A *gyerekek* változó a fában a gyerekekre mutató élek azonosítóinak halmazát, az *egyéb* változó pedig az összes többi összeköttetés azonosítóinak halmazát tárolja. Így a feszítőfa ismeretét „oszthatjuk" a processzorok között.

Az egyes processzorokra vonatkozó kód szegmensekből áll össze. Az első szegmens (1–4. sorok) leírja, a processzorok helyi változóinak kezdőértékadását. Emlékezzük rá, hogy a helyi változók a nulladik időpillanat előtt kapnak kezdőértéket. A következő három szegmens (5–10., 11–14. és 15–18. sorok) leírja azokat a műveleteket, melyeket az egyes processzoroknak kell végrehajtani egy üzenet fogadásakor: <*kijelöl*>, <*jóváhagyva*> vagy <*visszautasítva*> . Az utolsó szegmens (19–21. sorok) csak a P_r processzor kódjára vonatkozik. Ez a szegmens csak akkor hajtódik végre, ha a helyi *szülő* változó értéke a P_r processzor esetén NIL. Valamely időpillanatban megtörténhet, hogy egy processzornak egynél több szegmenst kell végrehajtania (például akkor, ha a processzor két másik processzortól kapott <*kijelöl*> üzenetet). Ekkor a processzor a szegmenseket egymás után egyesével hajtja végre (egy processzoron futó szegmensek sohasem hajtódnak végre egyszerre). Azonban egy másik processzor műveletei tetszőlegesen hajtódhatnak végre a végrehajtás során. A feldolgozható összes üzenet végül feldolgozásra kerül és a végrehajtható szegmensek végül végrehajtódnak (tisztaság).

Elárasztás

```
     Kód a P_k, 1 ≤ k ≤ n processzorok esetén
1    kezdőértékek beállítása
2        szülő ← NIL
3        gyerekek ← ∅
4        egyéb ← ∅

5    üzenet feldolgozása, <kijelöl> érkezett a j élen
6        if szülő = NIL
7           then szülő ← j
8                <jóváhagyva> küldése a j élre
9                <kijelöl> az összes szomszédok
                 \{j} halmazbeli összeköttetésre
```

```
10          else<visszautasítva> küldése a j összeköttetésre

11  üzenet feldolgozása <jóváhagyva> érkezett a j élen
12      gyerekek ← gyerekek ∪ {j}
13      if gyerekek ∪ egyéb = szomszédok \{szülő}
14          then terminate

15  üzenet feldolgozása < visszautasítva> érkezett a j élen
16      egyéb ← egyéb ∪ {j}
17      if gyerekek ∪ egyéb = szomszédok \{szülő}
18          then terminate

    Extra kód a kijelölt P_r processzorra
19  if szülő = NIL
20      then szülő ← NINCS
21           <kijelöl> küldése az összes szomszédok-beli élre
```

Az alábbiakban körvonalazzuk, hogyan működik az algoritmus. A kijelölt processzor egy *<kijelöl>* üzenetet küld az összes szomszédjának, és a NINCS értéket rendeli a *szülő* változóhoz (a NIL és a NINCS eltérő értékek, és eltérnek bármely természetes számtól), úgy, hogy sohasem küldi az üzenetet újra egyetlen szomszédjának sem.

Amikor egy processzor először feldolgoz egy *<kijelöl>* üzenetet, a saját *szülő* változóhoz rendeli annak az összeköttetésnek az azonosítóját, melyen az üzenet érkezett, és ugyanazon az összeköttetésen válaszol egy *<jóváhagyva>* üzenettel, valamin továbbítja a *<kijelöl>* üzenetet az összes többi összeköttetésen. Azonban, amikor egy processzor újra feldolgoz egy *<kijelöl>* üzenetet, a processzor a *<visszautasítva>* üzenettel válaszol, mert a *szülő* változó már nem NIL értékű.

Amikor egy processzor feldolgoz egy *<jóváhagyva>* üzenetet, a saját *gyerekek* halmazához adja annak az összeköttetésnek az azonosítóját, melyen az üzenet érkezett. Előfordulhat, hogy a *gyerekek* halmaz és az *egyéb* halmaz együtt tartalmazzák a processzorból induló összes összeköttetés azonosítóját a *szülő* változóban tárolt azonosítót kivéve. Ebben az esetben a processzor egy megállási állapotba kerül.

Amikor egy processzor feldolgoz egy *<visszautasítva>* üzenetet, a saját *egyéb* halmazához adja annak az összeköttetésnek az azonosítóját, melyen az üzenet érkezett. Ha a *gyerekek* és az *egyéb* halmazok uniója elég nagy, akkor a processzor szintén egy megállási állapotba kerül.

Helyesség bizonyítása

Most bebizonyítjuk, hogy az ELÁRASZTÁS algoritmus felépíti a feszítőfát. Az algoritmus végrehajtásának kulcspillanatai azok, amikor az egyes processzorok értéket rendelnek a *szülő* változóhoz. Ezek a hozzárendelések határozzák meg a feszítőfa „alakját" Azok a tények, hogy bármely processzor valamikor végül végrehajt egy utasítást, valamint hogy minden üzenet valamikor végül megérkezik, és hogy minden üzenet feldolgozásra kerül, biztosítják, hogy ezek a hozzárendelések végigfutnak a szomszédokon. Így az algoritmus kiterjed a gráf egy részfájára, jóllehet a terjedés meglehetősen lassú lehet. Végül a feszítőfa létrejön. Ha a feszítőfa létrejött, végül az egyes processzorok megállnak, bár néhány processzor még a fa létrejötte előtt megáll.

19.5. lemma. *Bármely $1 \leq k \leq n$ esetén létezik olyan t_k időpont, hogy az az első olyan pillanat, amikor pontosan k processzor esetén nem* NIL *a szülő változójának értéke, és ezek a processzorok és a szülő változók egy P_r gyökerű fát alkotnak.*

Bizonyítás. A bizonyítást k szerinti indukcióval végezzük. Az alapesetben tegyük fel, hogy $k = 1$. Figyeljük meg, hogy a P_r processzor valamikor *nincs* értéket rendel a *szülő* változójához. Legyen t_1 az a pillanat, amikor ez bekövetkezik. Ekkor a P_r kivételével az összes processzor esetén a *szülő* változó értéke még mindig NIL, mivel még <*kijelöl*> üzenet nem került elküldésre. A P_r processzor és a *szülő* változó egy egy csúcsú, él nélküli fát alkot. Így ezek egy gyökeres fát alkotnak. Ezzel beláttuk, hogy az induktív feltevés igaz a $k = 1$ esetben.

Induktív lépésként tegyük fel, hogy $1 \leq k < n$ és azt, hogy az induktív feltétel igaz k-ra. Tekintsük azt a t_k pillanatot, amikor pontosan k olyan processzor van, melynél a *szülő* változó értéke nem NIL. Mivel $k < n$, létezik egy olyan processzor, mely nincs a fában. Mivel a G hálózatgráf összefüggő, létezik olyan – nem a fában lévő – processzor, mely szomszédos a fával. (A processzorok bármely T részhalmaza esetén egy P_i T-vel akkor és csak akkor *szomszédos,* ha létezik egy olyan él a G gráfban, amely P_i-től egy T-beli processzorba vezet.) Emlékezzünk rá, hogy a meghatározás szerint egy ilyen processzor *szülő* változójának értéke NIL. Az induktív feltétel szerint a k processzoroknak a kódjuk 7. sorát már végre kellett hajtaniuk, így már küldtek, vagy valamikor a jövőben küldenek egy <*kijelöl*> üzenetet az egyes szomszédaiknak a *szülő* változóban meghatározott összeköttetés kivételével az összes összeköttetésen. Így a fával szomszédos nem fabeli processzorok már kaptak vagy valamikor a jövőben kapni fognak egy <*kijelöl*> üzenetet. Emiatt végül az összes szomszédos processzor egy NIL-től eltérő értéket fog rendelni a *szülő* változójához. Legyen $t_{k+1} > t_k$ az első pillanat, amikor valamely processzor végrehajt egy ilyen hozzárendelést, és jelöljük ezt a processzort P_i-vel.

Ez nem lehet egy fabeli processzor, mivel a fabeli processzorok még egyszer már nem rendelnek értéket a *szülő* változójukhoz. Lehet P_i olyan processzor, mely nem része a fának, de nem is szomszédos vele? Nem, mert egy ilyen processzor nem rendelkezik a fával való közvetlen összeköttetéssel, így nem kaphat <*kijelöl*> üzenetet közvetlenül a fából, és így ez azt jelentené, hogy valamely t_k és t_{k+1} közti t' időpontban valamely nem fabeli P_j <*kijelöl*> üzenetet küldött P_i-nek, s így P_j-nek NIL-től eltérő értéket kellett rendelnie a *szülő* változójához valamikor t_k után, de t_{k+1} előtt, ami ellentmond annak a ténynek, hogy t_{k+1} az első ilyen pillanat. Következésképp P_i egy olyan nem fabeli processzor, mely szomszédos a fával, s mint ilyen a t_{k+1} pillanatban P_i a *szülő* változójához rendeli egy olyan összeköttetés azonosítószámát, mely egy fabeli processzorhoz kapcsolja. Tehát a t_{k+1} az első olyan pillanat, amikor pontosan $k+1$ olyan processzor van, melynél a *szülő* változó értéke nem NIL, és ekkor ezek a processzorok és a *szülő* változóik egy P_r gyökerű fát alkotnak. Ezzel teljes az induktív lépés és a lemma bizonyítása. ■

19.6. tétel. *Minden processzor véges időn belül megáll, és amikor már minden processzorok megállt, a szülő változók által indukált részgráf egy P_r gyökerű feszítőfa lesz.*

Bizonyítás. A 19.5. lemma alapján tudjuk, hogy létezik egy t_n pillanat, amikor az összes processzor és a *szülő* változóik értéke egy feszítőfát alkotnak.

Előfordulhat-e, hogy minden processzor megáll t_n előtt? A kód vizsgálatával az kapjuk, hogy egy processzor csak akkor áll meg, ha <*visszautasítva*> vagy <*jóváhagyva*> üzenetet kapott az összes szomszédjától a *szülő* változó által mutatott szomszédja kivételével. Egy processzor ilyen üzenetet csak a processzor által küldött <*kijelöl*> üzenetre adott válaszként kaphat. A t_n időpillanatban van egy olyan processzor, mely még nem küldött <*kijelöl*> üzenetet. Emiatt nem igaz az, hogy minden processzor már megállt a t_n pillanatig.

Megáll-e minden processzor végül? Megjegyezzük, hogy a t_n pillanatig minden egyes processzor vagy már küldött, vagy végül küld egy <*kijelöl*> üzenetet a *szülő* változó által mutatott szomszéd kivételével az összes szomszédnak. Amikor egy processzor megkap egy <*kijelöl*> üzenetet, a processzor válaszol egy <*visszautasítva*> vagy egy <*jóváhagyva*> üzenettel, még akkor is, amikor a processzor leállt. Így minden egyes processzor valamikor végül kap egy <*visszautasítva*> vagy egy <*jóváhagyva*> üzenetet az összes olyan összeköttetésen, melyen a processzor egy <*kijelöl*> üzenetet küldött. Így végül minden processzor leáll. ■

Figyeljük meg, hogy az a tény, hogy egy processzor leállt, nem jelenti azt, hogy a feszítőfa elkészült. Valójában előfordulhat, hogy a hálózat egyik

részében lévő processzorok megállnak akkor, amikor a hálózat egy másik részének processzorai még egy üzenet sem kaptak.

19.7. tétel. *Az* ELÁRASZTÁS *algoritmus üzenetszáma* $O(e)$, *ahol* e *a* G *gráfban lévő élek száma.*

A tétel bizonyítását meghagyjuk feladatnak (lásd *19-1* feladat).

Gyakorlatok

19.2-1. Előfordulhat, hogy egy processzor már leállt, bár még nem kapott egyetlen üzenetet sem. Egy egyszerű hálózatpéldán mutassuk be, hogyan lehet az üzenettovábbítást és a processzor számítását úgy késleltetni, hogy ez valójában megtörténjen.

19.3. Gyűrűs algoritmusok

Egy osztott rendszerben gyakran kell koordinálni a processzorok tevékenységét. Ez gyakran egyszerűsíthető, ha van egy olyan processzor, amely koordinátorként működik. Előfordulhat, hogy kezdetben nincs a rendszernek koordinátora vagy egy meglévő koordinátor meghibásodik, és egy másikat kell választani. Ez felveti azt a problémát, hogy a processzoroknak ki kell választaniuk pontosan egyet maguk közül: egy ***vezetőt***. Ebben a szakaszban speciális típusú hálózatokban vizsgáljuk ezt a problémát: gyűrűkben. A probléma megoldására kifejlesztünk egy aszinkron algoritmust. Mint ahogy demonstrálni fogjuk, az algoritmus aszimptotikusan optimális üzenetszámmal rendelkezik. Ebben a szakaszban megismerjük a soros algoritmusokban gyakran használt, a futási időt alacsonyan tartó, jól ismert oszd-meg-és-uralkodj technikának az osztott analógját. Az osztott rendszereknél ez a technika segít csökkenteni az üzenetszámot.

19.3.1. A vezetőválasztási probléma

A vezetőválasztási probléma az, hogy választani kell egy vezetőt egy processzorhalmazban. Formálisan minden processzor rendelkezik egy *vezető* nevű változóval, melynek kezdőértéke NIL. Egy algoritmusról azt mondjuk, hogy *megoldja a vezetőválasztási problémát*, ha kielégíti a következő feltételeket:

1. Bármely végrehajtásban pontosan egy processzor valamikor IGAZ értéket rendel a *vezető* változójához, és

2. bármely végrehajtásban ha egy processzor egyszer már értéket rendelt a *vezető* változójához, akkor a változó értéke már nem fog változni.

Gyűrű modell

A vezetőválasztás problémáját egy speciális hálózatban, a gyűrűben fogjuk vizsgálni. Formálisan az n csúcsú osztott rendszert modellező G gráf egy egyszerű kört alkot, más él nincs a gráfban. A processzorhoz kapcsolódó két összeköttetést ÓMJ (óramutató járása szerinti), illetve ÓMJE (óramutató járásával ellentétes) címkével látjuk el. A processzorok megegyeznek a gyűrű irányultságán, azaz ha egy üzenet ÓMJ irányban halad tovább n lépésben, akkor ezzel bejárja az összes csúcspontot és visszakerül az kezdetiküldőhöz. Ugyanez igaz az ÓMJE irányra is. Minden processzor rendelkezik egy egyedi *azonosítóval*, amely egy természetes szám. Az egyes processzorok azonosítója tehát különbözik bármely más processzorétól. Az azonosító értékeknek nem kell egymás után következő $1, \ldots, n$ számoknak lenniük. Kezdetben egyetlen processzor sem ismeri egyetlen más processzor azonosítóját sem. A processzorok szintén nem ismerik a gyűrű n hosszát.

19.3.2. A vezetőválasztó algoritmus

A BULLY algoritmus megválaszt egy vezetőt a $P_1, \ldots, P_n$ processzorok közül. A processzorazonosítókat az algoritmus kritikus módon használja. Röviden: mindegyik processzor megpróbál vezető lenni, a legnagyobb azonosítójú processzor blokkolja a többiek kísérletét, magát vezetőnek deklarálja, és kényszeríti a többieket, hogy ne legyenek vezetők.

Az algoritmus néhány ötletének szemléltetéséhez kezdjük az algoritmus egy egyszerűbb változatával. Tegyük fel, hogy minden egyes processzor végigküld egy üzenetet a gyűrűben, mely tartalmazza a processzor azonosítóját. Bármely processzor *csak* akkor küld tovább egy ilyen üzenetet, ha az üzenetben lévő azonosító nagyobb, mint a processzor azonosítója. Így a gyűrűben a legnagyobb azonosítójú processzor által küldött üzenet mindig továbbítódik, végül körbeutazik a gyűrűn, és visszatér ahhoz a processzorhoz, mely eredetileg küldte. A processzor észlelni tudja, hogy egy ilyen üzenet visszaérkezett, mivel más processzor nem küld üzenetet ezzel az azonosítóval (az azonosítók egyediek). Vegyük észre, hogy más üzenet nem utazik körbe a gyűrűn, mivel a legnagyobb azonosítójú nem fogja továbbadni. Azt mondhatjuk, hogy a legnagyobb azonosítójú processzor „lenyeli" azokat az üzeneteket, melyek kisebb azonosítót szállítanak. Ezután a processzor vezető lesz, és végigküld egy speciális üzenetet a gyűrűn, amellyel arra veszi rá a többieket, hogy ne döntsenek úgy, hogy vezetők lesznek. Az algoritmus üzenetszáma legrosszabb esetben $\Theta(n^2)$, mivel az egyes processzorok legfeljebb n üzenetet küldenek, és a vezető küld még n további üzenetet, másrészt lehet úgy azonosítót rendelni a processzorokhoz és késleltetni a processzorokat és az üzeneteket, hogy az üzeneteket az n processzor egy konstans hányada küldi az

n összeköttetés egy konstans hányadán. Az algoritmus javítható úgy, hogy az üzenetszám $O(n \lg n)$-re csökkenjen. Egy ilyen javított algoritmust mutatunk be ennek a pontnak a hátralévő részében.

A BULLY algoritmus kulcsgondolata az, hogy kevés üzenet utazik hosszan, így biztosítva az $O(n \lg n)$ üzenetszámot. Speciálisan, a processzorok tevékenységét fázisokra bontjuk. Egy fázis elején egy processzor küld egy „próba" üzenetet mind az ÓMJ, mind az ÓMJE irányba. Ezek az üzenetek a küldő azonosítóját tartalmazzák és egy bizonyos „élettartam" értéket, mely korlátozza, hogy hány lépést tehetnek az egyes üzenetek. A próba üzenetet akkor adhatja tovább egy processzor, ha a benne szereplő processzorazonosító nagyobb, mint a saját azonosítója. Amikor az üzenet eléri a korlátját, és nem nyelték le, akkor „visszapattan". Így, ha az eredeti küldő két visszapattant üzenetet kap a két különböző irányból, akkor a processzor biztos benne, hogy a korlátnyi távolságra nincs nagyobb azonosítójú processzor sem az ÓMJ, sem az ÓMJE irányban, mivel akkor az lenyelte volna a próbaüzenetet. Csak ezután lép a processzor a következő fázisba, amelyben egy nagyobb élettartam korlátú próbaüzenetet küld, hogy megtudja, van-e nagyobb azonosítójú processzor egy kétszer nagyobb sugarú környezetben. Ennek eredményeképpen egy processzor által küldött próbaüzenet csak akkor lép sokat, ha nincs nagyobb azonosítójú processzor a processzor egy nagy sugarú környezetében. Így tehát egyre kevesebb processzor küld üzeneteket, melyek egyre távolabb jutnak. Következésképp, ahogy nemsokára bebizonyítjuk, az algoritmus üzenetszáma $O(n \lg n)$ lesz.

A következőkben részletezzük a BULLY algoritmust. Minden egyes processzornak öt helyi változója van. Az *id* változó tárolja a processzor egyedi azonosítóját. A *vezető* változó IGAZ értéket vesz fel, amikor a processzor úgy dönt, hogy ő a vezető, egyébként a HAMIS értéket veszi fel. A maradék három változó az állapotkövetést szolgálja. Az *alszik* azt határozza meg, hogy a processzor küldött-e olyan <*próba,id,*0,0> üzenetet, mely a processzor *id*-jét hordozza. Bármely processzor küldhet <*próba,id,fázis,*$2^{fázis} - 1$> üzenetet bármely irányban (ÓMJ, ÓMJE) – különböző *fázis* különböző értékei mellett. Bármely üzenetre a processzor egy <*válasz,id,fázis*> üzenettel válaszolhat. Az *ÓMJválaszolt* és a *ÓMJEválaszolt* változók annak a nyilvántartására szolgálnak, hogy a választ feldolgozta-e a processzor.

Az egyes processzorokra vonatkozó kód öt részből áll. Az első rész (1–5. sorok) beállítja a processzor helyi változóinak kezdeti értékét. A második részt (6–8. sorok) csak akkor szabad végrehajtani, ha az *alszik* helyi változó értéke IGAZ. A fennmaradó három rész (9–17., 18–26. és 27–31. sorok) írja le azokat a lépéseket, melyeket a processzor a három különböző üzenet, nevezetesen a <*próba,ids,fázis,élettartam*>, <*válasz, ids, fázis*> és a <*megállás*> hatására

végez. Az üzenetek az *ids fázis* és az *élettartam* paramétereket használják, melyek természetes számok.

Most bemutatjuk, hogyan működik az algoritmus. Emlékezzünk vissza, hogy az egyes processzorok helyi változói a globális idő szerinti 0 időpillanat előtt kaptak kezdőértéket. Minden egyes processzor valamikor a jövőben küld egy <*próba, id, 0, 0*> üzenetet, mely a processzor *id* azonosítóját tartalmazza. Ekkor azt mondjuk, hogy a processzor a 0 fázisba *lépett*. Általában, amikor a processzor egy <*próba,id,fázis,*$2^{fázis}-1$> üzenetet küld, azt mondjuk, hogy a processzor a *fázis* sorszámú fázisba *lépett*. A <*próba, id, 0, 0*> üzenet sohasem kerül újraküldésre, mivel a 7. sorban az *alszik* változó a HAMIS értéket veszi fel. Előfordulhat, hogy mire ez az üzenet küldésre kerül, a processzor már más üzeneteket is feldolgozott.

Amikor egy processzor feldolgoz egy, az ÓMJ (vagyis az óramutató járásával megegyező irányhoz tartozó) kapcsolaton érkező <*próba, ids, fázis, élettartam*> formátumú üzenetet, akkor az *ids* paramétertől és a processzor *id* azonosítójától függően tevékenykedik. Ha az *ids* kisebb, mint az *id*, a processzor nem csinál semmit (azaz lenyeli az üzenetet). Ha az *ids* egyenlő az *id*-del, és a processzor még nem döntött, akkor, mint majd látjuk, a processzor által küldött *próba*-üzenet körbement a teljes gyűrűn. Ekkor a processzor küld egy <*megállás*> üzenetet, magát vezetőnek jelöli meg, és leáll (de a leállás után még dolgozhat fel üzeneteket). Ha az *ids* nagyobb, mint az *id*, akkor a tevékenység az *élettartam* paraméter értékétől függ. Ha ez az érték nagyobb, mint nulla, a processzor csökkenti az *élettartam* paraméter értékét, és így adja tovább az üzenetet. Ha az *élettartam* értéke nulla, akkor a processzor visszaküld (ÓMJ irányban) egy válaszüzenetet. Hasonlóak a műveletek, abban az értelemben, hogy az üzenetküldési irányok felcserélődnek, amikor a <*próba, ids, fázis, élettartam*> üzenet az ÓMJE kapcsolaton érkezik. A részleteket lásd a programkódban.

Bully

Kód a P_k, $1 \leq k \leq n$ processzorok esetén:

```
1  kezdőértékek beállítása
2      alszik = IGAZ
3      ÓMJválaszolt = HAMIS
4      ÓMJEválaszolt = HAMIS
5      vezető = NIL
```

```
6  if alszik then
7      alszik = HAMIS
8      <próba, id, 0,0> küldése az ÓMJ és az ÓMJE élekre

9  üzenet feldolgozása <próba, ids, fázis, élettartam>
   érkezett az ÓMJ (ill. ÓMJE) élre
10     if id = ids és vezető = NIL then
11         <megállás> küldése az ÓMJ élre
12         vezető = IGAZ
13         terminate
14     if ids > id és élettartamttl > 0 then
15         <próba, ids, fázis, élettartam-1> küldése az ÓMJE (ill. ÓMJ) élre
16     if ids > id és élettartam = 0 then
17         <válasz, ids, fázis> küldése az ÓMJ (ill. ÓMJE) kapcsolatra

18 üzenet feldolgozása <válasz, ids, fázis>
   érkezett az ÓMJ (ill. ÓMJE) élen
19     if id ≠ ids then
20         <válasz, ids, fázis> küldése az ÓMJE (ill. ÓMJ) élre
21     else
22         ÓMJválaszolt = IGAZ (ill. ÓMJEválaszolt)
23         if ÓMJválaszolt és ÓMJEválaszolt then
24             ÓMJválaszolt = HAMIS
25             ÓMJEválaszolt = HAMIS
26             <próba, id, fázis+1, 2^(fázis+1) − 1>
               küldése az ÓMJ és ÓMJE élekre

27 üzenet feldolgozása <megállás> érkezett az ÓMJ élen
28     if vezető = NIL then
29         <megállás> küldése az ÓMJE élre
30         vezető = HAMIS
31         terminate
```

Amikor egy processzor az ÓMJ kapcsolaton érkező *<válasz, ids, fázis>* üzenetet kap, először ellenőrzi, hogy a *ids* értéke eltér-e a saját *id* értékétől. Ha igen, a processzor egyszerűen továbbadja az üzenetet. Ha azonban $ids = id$, akkor a processzor feljegyzi, hogy egy válaszüzenet érkezett az ÓMJ kapcsolaton úgy, hogy az *ÓMJválaszolt* változóhoz IGAZ értéket rendel. Ezután

a processzor ellenőrzi, hogy az *ÓMJválaszolt* és az *ÓMJEválaszolt* változók értéke IGAZ-e. Ha igen, akkor már mindkét irányból érkezett válasz. Ekkor a processzor mindkét változóhoz a HAMIS értéket rendeli, majd egy *próba* üzenetet küld. Ez az üzenet a processzor *id* azonosítójából, a következő fázis *fázis*+1 számából, és a $(2^{fázis+1} - 1)$-re növelt élettartamértékből áll. Hasonló tevékenységet kell végezni, ha a <*válasz, ids, fázis*> üzenet az ÓMJE kapcsolatra érkezik.

Az utolsó, a processzor által feldolgozandó üzenet a <*megállás*> . A processzor ellenőrzi, hogy már döntött-e arról, hogy ő-e a vezető. Ha még nem volt döntés, továbbadja a <*megállás*> üzenetet, és úgy dönt, hogy nem ő a vezető. Ez az üzenet egyszer majd egy olyan processzorhoz jut, mely már döntött, és így az nem kerül továbbadásra.

19.3.3. A vezetőválasztási algoritmus elemzése

Az elemzést azzal kezdjük, hogy belátjuk, a BULLY algoritmus megoldja a vezetőválasztási problémát.

Helyesség bizonyítása

Először a BULLY algoritmus helyességét látjuk be.

19.8. tétel. *A* BULLY *algoritmus megoldja a vezetőválasztási problémát bármely gyűrűben aszinkron processzorok esetén.*

Bizonyítás. Azt kell belátnunk, hogy az alfejezet elején említett két feltétel megáll. A bizonyítás kulcsa, mely egyszerűsíti az érvelést, hogy egy processzorra koncentrálunk. Tegyük fel, hogy a P_i processzor rendelkezik a gyűrűben a legnagyobb *id* azonosítóval. Ez a processzor valamikor feltétlenül végrehajtja a 6–8. lépéseket. Ekkor a processzor küld egy <*próba, id, 0, 0* > üzenetet az ÓMJ és az ÓMJE kapcsolatokon. Jegyezzük meg, hogy amikor a processzor egy <*próba, id, fázis,* $2^{fázis} - 1$> üzenetet küld, az ilyen üzeneteket a többi processzor mindig továbbadja, amíg az *élettartam* értéke nullára nem esik. Ekkor a processzor visszaküld egy választ a P_i-nek. Ekkor a P_i végső soron minden irányból megkapja a <*válasz, id, fázis*> üzenetet, és a *fázis*+1 fázisba lép úgy, hogy <*próba, id, fázis+1,* $2^{fázis+1}-1$> üzenetet küld mindkét irányba. Ezek az üzenetek magasabb *élettartam* értékkel rendelkeznek, mint az előző, *fázis* sorszámú fázis élettartamértéke. Mivel a gyűrű véges, az *élettartam* olyan nagy lesz, hogy a P_i processzor kap egy olyan üzenetet, melyben a P_i az azonosító. Jegyezzük meg, hogy P_i két ilyen üzenetet is kap. Az első alkalommal, amikor a P_i egy ilyen üzenetet feldolgoz, a processzor küld egy <*megállás*> üzenetet, és leáll, mint vezető. A második alkalommal, amikor a P_i egy ilyen üzenetet feldolgoz, a 11–13. sorok nem kerülnek végrehajtásra,

mivel a *vezető* változó értéke már nem NIL. Megjegyezzük, hogy más P_j processzor nem hajthatja végre a 11–13. sorokat, mert egy, a P_j-ból küldött *próba* üzenet nem mehet végig az egész gyűrűn, mivel az útjában lévő P_i lenyeli azt. Ugyanakkor mivel az azonosítók egyediek, más processzor nem küldhet *próba* üzenetet P_i azonosítóval. Így a P_i-n kívül más processzor nem rendelhet IGAZ értéket a *vezető* változójához. Bármely a P_i-től különböző processzor, mely a <*megállás*> üzenetet megkapja, HAMIS értéket rendel a *vezető* változójához, és továbbadja ez üzenetet. Végül, a <*megállás*> üzenet megérkezik P_i-hez, és az nem adja tovább. A fenti érvelés azt bizonyítja, hogy végül pontosan egy processzor rendeli a IGAZ értéket a *vezető* változójához, és az összes többi HAMIS értéket rendel ehhez a változójához. Ha egy processzor már értéket rendelt a *vezető* változójához, az már változatlan marad. ■

A következő feladatunk, hogy felső korlátot találjunk az algoritmus által küldött üzenetek számára. A következő lemma megmutatja, hogy az egy fázisba belépő processzorok számára a fázis sorszámának növekedésével exponenciálisan csökkenő korlát adható.

19.9. lemma. *Adott egy n csúcsú gyűrű. Az $i \geq 0$ fázisba belépő processzorok k száma legfeljebb $n/2^{i-1}$.*

Bizonyítás. Pontosan n processzor lép az $i = 0$ fázisba, mivel minden egyes processzor végső soron elküldi a <*próba, id, 0, 0*> üzenetet. A lemmában említett korlát azt állítja, hogy a 0 fázisba lépő processzorok száma legfeljebb $2n$, így a korlát triviálisan igaz $i = 0$-ra. Vizsgáljuk a többi esetet, azaz tegyük fel, hogy $i \geq 1$. Tegyük fel továbbá, hogy a P_j processzor belép az i-edik fázisba. Így az a definíció alapján küld egy <*próba*,$id, i, 2^i - 1$> üzenetet. Ahhoz, hogy a processzor egy ilyen üzenetet küldjön, a processzor által a megelőző fázisban a két irányba küldött két <*próba*, $id, i-1, 2^{i-1}-1$> üzenetnek 2^{i-1} ugrást kellett végrehajtania, mindig elérve egy, a P_j azonosítónál kisebb azonosítójú processzort. (Máskülönben ha egy *próba* üzenet egy, az üzenetben szereplőnél nagyobb vagy egyenlő azonosítójú processzorhoz érkezik, az lenyeli az üzenetet, és nem generál válaszüzenetet. Következésképp P_j nem léphetne az i-edik fázisba.) Ennek eredményeképp, ha egy processzor belép az i-edik fázisba, nincs más olyan processzor tőle mindkét irányban 2^{i-1} ugrásnyi távolságra, mely valaha is ebbe a fázisba lépne. Tegyük fel, hogy $k \geq 1$ processzor van az i-edik fázisban. Minden egyes ilyen p_j processzorhoz hozzárendelhetjük a tőle ÓMJ irányba lévő 2^{i-1} következő processzort. Ezért legalább $k + k \cdot 2^{i-1}$ különböző processzor van a gyűrűben. Így $k(1 + 2^{i-1}) \leq n$. Gyengíthetjük a korlátot az 1 elhagyásával, így a kívánt $k \cdot 2^{i-1} \leq n$ eredményt kapjuk. ■

19.10. tétel. *A* BULLY *algoritmus üzenetszáma $O(n \lg n)$, ahol n a gyűrű mérete.*

Bizonyítás. Megjegyezzük, hogy bármely, az i-edik fázisban lévő processzor, a két (ÓMJ és ÓMJE) irányba, 2^i távolságra küld üzeneteket. Ez az i-edik fázisba lépő processzoronként legfeljebb $4 \cdot 2^i$ üzenetet jelent. Az üzenetek száma kisebb is lehet, mint $4 \cdot 2^i$, ha egy próbaüzenetet lenyelnek út közben. A 19.9. lemma felső korlátot ad a k-adik fázisba lépő processzorok darabszámára. Mi az a legmagasabb fázis, amelybe egy processzor még beléphet? Az i-edik fázisban lévő processzorok k száma legfeljebb $n/2^{i-1}$. Így, ha $n/2^{i-1} < 1$, nem lehet olyan processzor, mely az i-edik fázisba lép. Tehát egyik processzor sem léphet magasabb fázisba, mint $h = 1 + \lceil \lg n \rceil$, mivel $n < 2^{(h+1)-1}$. Végül, egyetlen processzor küld egy megállási üzenetet, mely egyszer körbemegy a gyűrűn. Így az algoritmus által küldött összes üzenetek számára a

$$n + \sum_{i=0}^{1+\lceil \lg n \rceil} \left(n/2^{i-1} \cdot 4 \cdot 2^i \right) = n + \sum_{i=0}^{1+\lceil \lg n \rceil} 8n = O(n \lg n)$$

felső korlátot kapjuk. ∎

Burns azt is megmutatta, hogy a BULLY algoritmus üzenetszáma aszimptotikusan optimális, mivel bármely, a processzorok számát nem ismerő vezetőválasztó algoritmus egy aszinkron gyűrűben $\Omega(n \lg n)$ üzenetet küld.

19.11. tétel. *Bármely, a processzorok számát nem ismerő vezetőválasztó algoritmus egy aszinkron gyűrűben $\Omega(n \lg n)$ üzenetet küld.*

Gyakorlatok

19.3-1. Bizonyítsuk be, hogy az egyszerűsített BULLY algoritmus üzenetszáma $\Omega(n^2)$, ha az n méretű gyűrűben megfelelően rendeljük az azonosítókat a processzorokhoz, és meghatározzuk, hogy hogyan késleltessük a processzorokat és az üzeneteket.

19.3-2. Mutassuk meg, hogy a BULLY algoritmus üzenetszáma $\Omega(n \lg n)$.

19.4. Hibatűrő egyetértés

Az eddig bemutatott algoritmusok azon a feltételezésen alapulnak, hogy megbízható rendszeren futnak. Most bemutatunk néhány kiválasztott algoritmust nem megbízható rendszerek esetén, ahol az aktív (vagy helyes) processzoroknak koordinálniuk kell a tevékenységüket, azaz közös döntéseket kell hozniuk.

A probléma természeténél fogva nehéz olyan processzoroknak megegyezni, melyek egy hibáknak kitett osztott rendszerben vannak. Tekintsük például

azt a megtévesztően egyszerű példát, amikor két hibamentes processzor megpróbál megegyezni egy közös bitről, miközben olyan kommunikációs médiát használ, ahol az üzenetek elveszhetnek. Ez a probléma a ***két tábornok problémája*** néven ismert. Ebben két tábornoknak kell koordinálnia egy támadást olyan futárok segítségével, akiket elfoghat az ellenség. Az derül ki, hogy véges számú üzenetküldéssel nem sikerül megoldani a problémát. Ezt indirekt módon bizonyítjuk. Tegyük fel, hogy van olyan, az A és B processzorok között használt protokoll, mely véges számú üzenetküldésből áll. Tekintsük azt az ilyen protokollt, mely a legkevesebb – mondjuk, k darab – üzenetet használja. Az általánosság megszorítása nélkül feltételezhetjük, hogy az utolsó, k-adik üzenetet A küldi B-nek. Mivel ezt az utolsó üzenetet B nem nyugtázza, A-nak attól függetlenül kell meghoznia döntését, hogy B megkapta-e ezt az üzenetet. Mivel az üzenet elveszhet, B-nek az utolsó üzenettől függetlenül kell döntenie. Ez alapján azonban A és B a k-adik üzenet felhasználása nélkül döntött, azaz létezik egy olyan protokoll, mely csak $k-1$ üzenetet használ a probléma megoldására. Ez ellentmond annak a feltételnek, hogy a probléma megoldásához k darab üzenetre van szükség.

Az alfejezet hátralévő részében azokat a problémákat vizsgáljuk, ahol a kommunikációs média megbízható, de a processzoroknál a következő kétfajta meghibásodás léphet fel: ***megállás*** (vagy összeomlás), amikor a processzor megáll, és nem hajt végre további lépéseket és a ***bizánci hiba,*** ahol a processzor a hibánál fogva tetszőleges (akár rosszindulatú) tevékenységet végezhet.

A bemutatott algoritmusok az úgynevezett ***egyetértési problémát*** (vagy konszenzus problémát) oldják meg, amely egy alapvető koordinációs probléma: azt kívánja a processzoroktól, hogy megegyezzenek egy közös kimenetben – annak ellenére, hogy a bemenetük különböző lehet.

19.4.1. Az egyetértési probléma

Tekintsünk egy olyan rendszert, amelyben minden egyes P_i processzor rendelkezik egy speciális x_i állapotkomponenssel, amit *bemenetnek* nevezünk, és rendelkezik egy y_i komponenssel, melyet *kimenetnek* (más néven *döntésnek*) nevezünk. Az x_i változó kezdetben a lehetséges bemenetek valamely jól rendezett halmazából származó értéket vesz fel, az y_i értéke nem definiált. Ha egyszer már rendeltek értéket az y_i változóhoz, az később már nem változtatható meg. Az egyetértési probléma bármely megoldásának a következőket kell garantálnia:

- **Megállás:** Minden megengedett végrehajtás és minden hibamentes P_i processzor esetén y_i kap értéket.
- **Megegyezés:** Minden végrehajtás és minden hibamentes P_i és P_j processzor esetén, ha y_i és y_j már kapott értéket, akkor $y_i = y_j$, azaz a

hibamentes processzorok nem eredményeznek különböző kimeneteket.

- **Érvényesség:** Minden végrehajtásban, ha valamely v értékre $x_i = v$ minden P_i processzor esetén, azaz, ha minden processzor ugyanazt a bemenőértéket kapja, és y_i értéket kap valamely hibamentes P_i processzor esetén, akkor az eldöntött érték a közös bemenet lesz.

Megjegyezzük, hogy a megállási hibák esetén ez az érvényességi feltétel gyengébb annál a követelménynél, hogy a hiba nélkül működő processzorok valamely processzor bemenetét határozzák meg kimenetnek. Ha egy processzor megáll, az nem érdekli az algoritmust, ezért nincs követelmény a meghibásodott processzor kimenetére sem.

Egy egyszerű algoritmussal kezdjük egy megállási hibát megengedő szinkron üzenetküldő rendszerben.

19.4.2. Egyetértés megállási hibák esetén

Mivel a rendszer szinkron, a végrehajtás menetek sorozatából áll. Minden egyes menet az összes üzenet kézbesítéséből, és az azt követő egyetlen számítási lépésből áll az összes processzorra. A meghibásodott processzorok a különböző végrehajtásokban különbözők lehetnek, azaz előre nem ismertek, számuk legfeljebb f. Legyen F a hibás processzorok halmaza. Minden egyes menet pontosan egyetlen számítási lépést tartalmaz azon processzorok esetén, melyek nincsenek F-ben, és legfeljebb egy lépést az F-ben lévő processzorok esetén. Továbbá, ha egy processzor F-ben van, és valamely menetben nem hajt végre számítási lépést, a további menetekben már nem hajt végre ilyen lépést. Az utolsó olyan menetben, melyben egy meghibásodott processzor számítási lépést hajt végre, a kimenő üzeneteinek tetszőleges részhalmaza kerül kézbesítésre.

EGYETÉRTÉS-MEGÁLLÁSI-HIBÁKNÁL

Kód a P_i, $0 \leq i \leq n-1$ processzorok esetén.
Kezdetben $V = \{x\}$
k-adik menet, $1 \leq k \leq f+1$
1 $\{v \in V : p_i$ még nem küldte el v-t$\}$ küldése az összes processzorhoz
2 S_j fogadása P_j-től, $0 \leq j \leq n-1$, $j \neq i$
3 $V = V \cup \bigcup_{j=0}^{n-1} S_j$
4 **if** $k = f+1$ **then** $y = min(V)$

Ebben az algoritmusban miden egyes processzor fenntart egy halmazt a rendszerben ismert értékek tárolására. Ez kezdetben csak a saját bemenetet tartalmazza. A későbbi menetekben a processzor ezt frissíti a már processzoroktól kapott halmazokkal. Ezután az új megismert étékeket közvetíti az

összes processzor felé. Ez $f+1$ menetig tart, ahol f a meghibásodható processzorok számának maximuma. Ezen a ponton a processzor meghatározza, hogy melyik a legkisebb érték az értékhalmazában.

Az algoritmus helyességének bizonyításához először azt kell észrevenni, hogy az algoritmus pontosan $f+1$ menetet igényel. Ez maga után vonja a megállást. Továbbá, az érvényességi feltétel nyilvánvalóan megáll, mivel a meghatározott érték valamely processzor bemenő értéke. Már csak azt kell megmutatni, hogy a megegyezési feltétel teljesül. A következő lemmát fogjuk bizonyítani:

19.12. lemma. *Minden egyes végrehajtásban az $(f+1)$-edik menet végén $V_i = V_j$, minden hibamentes P_i és P_j processzorpárra.*

Bizonyítás. Az állítást azzal bizonyítjuk, hogy megmutatjuk: ha $x \in V_i$ az $(f+1)$-edik menet végén, akkor $x \in V_j$ is az $(f+1)$-edik menet végén.

Legyen r az első olyan menet, amelyben x-et V_i-hez adták valamely p_i hibamentes proces szor esetében. Ha x kezdetben V_i-ben van, legyen $r = 0$. Ha $r \leq f$, akkor az $r+1 \leq f+1$ menetben P_i elküldi x-et az összes P_j-nek. Ezáltal P_j x-et V_j-hez adja, ha már nem lenne benne.

Egyébként tegyük fel, hogy $r = f+1$, és legyen P_j egy hibamentes processzszor, mely x-et először az $f+1$-edik menetben kapja meg. Ekkor kell lenni egy olyan $f+1$ elemből álló $P_{i_1}, \ldots, P_{i_{f+1}}$ processzorláncnak, melyből az x érték P_j-hez érkezik. Így P_{i_1} elküldi x-et P_{i_2}-hez az első menetben stb., amíg $P_{i_{f+1}}$ el nem küldi x-et P_j-hez az $(f+1)$-edik menetben. Ugyanakkor $P_{i_1}, \ldots, P_{i_{f+1}}$, egy $f+1$ processzorból álló lánc. Így legalább az egyik processzor a láncból, mondjuk P_{i_k}, hibamentes. Ezáltal P_{i_k} felveszi x-et a $k-1 < r$ menetben, ami ellentmond annak, hogy r minimális. ∎

Ez a lemma a korábban említett megfigyeléssel együtt bizonyítja a következő tételt.

19.13. tétel. *Az* EGYETÉRTÉS-MEGÁLLÁSI-HIBÁKNÁL *algoritmus megoldja az egyetértési problémát legfeljebb f megállási hiba esetén egy $f+1$ menetből álló üzenetküldő rendszerben.*

A következő tétel megmutatja, hogy az előző algoritmus az adott modell mellett optimális.

19.14. tétel. *Nincs olyan algoritmus, mely az egyetértés problémát kevesebb, mint $f+1$ menetben, f megállási meghibásodás mellett, ha $n \geq f+2$.*

Mi van, ha meghibásodások nem jóindulatúak? Azaz, megoldható-e az egyetértési probléma *bizánci* típusú meghibásodások mellett? Ha igen, hogyan?

19.4.3. Egyetértés bizánci típusú meghibásodások mellett

A bizánci modellben egy hibás processzor egy számítási lépés után ismeretlen állapotba kerül, és az általa küldött üzenet tetszőleges. Mint a megbízható esetben, minden egyes processzor egy számítási lépést tesz minden menetben, és minden, általa küldött üzenet kézbesítésre kerül az adott menetben. Így a meghibásodott processzor tetszőlegesen (akár rosszindulatúan is) viselkedhet. Például, különböző üzeneteket küldhet különböző processzorokhoz. Még az is előfordulhat, hogy a meghibásodott processzorok együttműködnek. Egy hibás processzor még egy összeomlott processzor működését is utánozhatja azzal, hogy egy adott pont után nem küld üzeneteket.

Ebben az esetben a konszenzus probléma definíciója ugyanaz, mint az összeomlásos hibák melletti üzenetküldési modellben. Az érvényességi feltétel ebben a modellben azonban nem ekvivalens annak megkövetelésével, hogy a hibamentes processzorok kimeneti értéke megfelel valamely processzor bemenetének. Mint a megállásos esetben, nincs feltétel a meghibásodott processzorok kimenetére.

19.4.4. Alsó korlát a hibás processzorok arányára

Pease, Shostak és Lamport bizonyította először a következő tételt:

19.15. tétel. *Ha $n \leq 3f$, akkor nincs olyan algoritmus, amely egy n processzorból és f bizánci processzorból álló rendszerben megoldja az egyetértési problémát.*

19.4.5. Egy polinomiális algoritmus

A következő algoritmus konstans méretű üzeneteket használ, $2(f+1)$ menetet igényel, és feltételezi, hogy $n > 4f$. Az algoritmust Berman és Garay mutatta be.

Ez a bizánci típusú meghibásodások melletti egyetértési algoritmus $f+1$ fázisból áll, minden fázis két menetet tartalmaz. Minden egyes processzor minden egyes fázisban rendelkezik egy kedvezményezett döntéssel, mely kezdetben a bemenő értéke. Minden egyes fázis első menetében a processzorok elküldik ezt a kedvezményezett adatot egymásnak. Legyen v_i^k a leggyakoribb érték, melyet a P_i processzor kapott a k fázis első menetében. Ha nincs ilyen érték, a $v_\perp$ érték lesz a használatos. A fázis második menetében a menet *királyának* nevezett P_k processzor elküldi a v_k^k többségi értékét az összes processzornak. Ha P_i (a fázis első menetében) $(n/2+f)$-nél több példányt kap v_i^k-ból, akkor a következő fázisban v_i^k lesz a kedvezményezett értéke. Egyébként a kedvezményezett értéke a fázis királyának v_k^k kedvezményezettje lesz, melyet

a fázis második menetében kapott. $f+1$ fázis után a processzor dönt a kedvezményezettje felől. Minden processzor fenntart egy *pref*$[0\,..\,n-1]$ helyi tömböt.

A helyességet a következő lemmákkal bizonyítjuk. A megállás közvetlen. A következőkben a döntés állandóságát látjuk be.

19.16. lemma. *Ha a k-adik fázis elején az összes hibamentes processzor kedvezményezettje v, akkor a k-adik fázis végén is v lesz a kedvezményezett érték minden processzor és minden k, $1 \leq k \leq f+1$ esetén.*

Bizonyítás. Mivel az összes hibamentes processzor kedvezményezettje a k fázis elején v, ezért mind legalább $n-f$ példányt kapnak (a sajátjukkal együtt) v-ből a k fázis első menetében. Mivel $n > 4f$ és $n - f > n/2 + f$, ezért az összes hibamentes processzor kedvezményezettje v lesz a k-adik fázis végén. ■

EGYETÉRTÉS-BIZÁNCI-HIBÁKNÁL

```
    Kód a P_i, 0 ≤ i ≤ n − 1 processzorok esetén:
    Kezdetben pref[j] = v_⊥, mindenj ≠ i esetén
    2k − 1, 1 ≤ k ≤ f + 1 menet
1   ⟨pref[i]⟩ küldése az összes processzornak
2   ⟨v_j⟩ fogadása P_j-től, és hozzárendelés pref[j]-hez
    minden 0 ≤ j ≤ n − 1, j ≠ i esetén
3   legyen maj a többségben lévő érték a pref[0],...,pref[n − 1]
            tömbelemekben, illetve v_⊥, ha nincs ilyen
4   legyen mult maj multiplicitása

    2k, 1 ≤ k ≤ f + 1 menet
5   if i = k then ⟨maj⟩ küldése az összes processzorhoz
6   ⟨király-maj⟩ fogadása p_k-tól, (v_⊥ ha nincs)
7   if mult > n/2 + f
8      then pref[i] = maj
9      else  pref[i] = király-maj
10  if k = f + 1 then y = pref[i]
```

Mindez maga után vonja az érvényességi feltétel teljesülését: ha az összes processzor a v bemenettel indul, akkor a továbbiakban is v lesz a kedvezményezettjük, és az $(f+1)$-edik fázisban a v lesz a eldöntött érték. A megegyezés meglétét a király biztosítja. Mivel minden fázisban más a király és $f+1$ fázis van, legalább egy menet rendelkezik hibamentes királlyal.

19.17. lemma. *Legyen g egy olyan fázis, amelyben a P_g király hibamentes. Ekkor az összes hibamentes processzor befejezi a g fázist, és a kedvezményezettje ugyanaz lesz.*

Bizonyítás. Tegyük fel, hogy az összes hibamentes processzor a királytól kapott többségi értéket veszi kedvezményezettnek. Mivel a király hibamentes, ugyanazt az üzenetet küldi, így az összes hibamentes processzornál a kedvezményezett ugyanaz.

Tegyük fel, hogy egy P_i hibamentes processzor a saját v többségi értékét tekinti kedvezményezettnek. Így P_i a g fázis első menetében v-re $(n/2+f)$-nél több üzenetet kap. Ezáltal az összes processzor, P_g-t is beleértve, a g fázis első menetében több mint $n/2$ üzenetet kap v-re, és a többségi értékét v-re állítja. Így minden hibamentes processzor v-t választja kedvezményezettnek. ∎

Ezáltal a $(g+1)$-edik fázisban az összes processzor ugyanazzal a kedvezményezettel rendelkezik, és a 19.16. lemma alapján ugyanazt az értéket határozzák meg kedvezményezettnek. Tehát az algoritmus rendelkezik a megegyezés tulajdonsággal, és megoldja az egyetértési problémát.

19.18. tétel. *Ha $n > 4f$, akkor létezik olyan algoritmus, mely n processzor és f bizánci típusú meghibásodás esetén megoldja az egyetértési problémát $2(f+1)$ menetben konstans méretű üzenetekkel.*

19.4.6. Lehetetlenség az aszinkron rendszerekben

Mint ahogy korábban bemutattuk, az egyetértési probléma megoldható szinkron rendszerekben mind megállási (jóindulatú), mind bizánci (súlyos) típusú meghibásodások mellett. Mi a helyzet az aszinkron rendszerekkel? Azon feltételezés mellett, hogy a kommunikációs rendszer teljesen megbízható, és a meghibásodásokat csak a megbízhatatlan processzorok okozzák, belátható, hogy ha a rendszer teljesen aszinkron, nincs egyetértési algoritmus még akkor sem, ha csak egyetlen processzor hibásodhat meg. Ez akkor is igaz, ha a processzorok csak megállás típusú hibának vannak kitéve. A lehetetlenség bizonyítása főleg a rendszer aszinkron voltán nyugszik.

A lehetetlenség fennáll mind a csak az olvasás/írás regisztereket használó közös memóriájú, mind az üzenetküldő rendszerekre. Az állítás először a közös memóriájú rendszerekre vonatkozik. Az üzenetküldő rendszerekre az eredmény szimulációval vihető át.

19.19. tétel. *Nincs olyan egyetértési algoritmus egy olvasás/írás aszinkron közös memóriájú rendszerre, mely akár egyetlen megállás típusú meghibásodást tolerálni tudna.*

Szimulációval a következő is belátható:

19.20. tétel. *Nincs algoritmus, mely egy n processzorból álló aszinkron üzenetküldő rendszerben megoldja az egyetértési problémát, ha akár csak egy processzor is megállási hibás.*

Megjegyezzük, hogy ezek az eredmények nem jelentik azt, hogy az egyetértés megoldhatatlan az aszinkron rendszerek esetén. Az eredmények inkább azt jelentik, hogy nincs olyan algoritmus, mely garantálná a megállást, a megegyezést és az érvényességet az összes végrehajtásban. Ésszerű azt feltételezni, hogy a megegyezés és az érvényesség szükséges, azaz ha egy konszenzus algoritmus megáll, akkor a megegyezési és az érvényességi követelmények garantáltak. Valójában vannak olyan hatékony és hasznos algoritmusok az egyetértés problémára, melyek nem garantálják a megállást minden végrehajtás esetén. A gyakorlatban ez elégséges lehet, mivel a nemmegállást okozó speciális feltételek meglehetősen ritkák. Továbbá, mivel sok valóságos rendszerben az ember időzítési feltételekkel élhet, lehet, hogy nem szükséges megoldást adni az aszinkron egyetértésre.

Gyakorlatok

19.4-1. Bizonyítsuk be a EGYETÉRTÉS-MEGÁLLÁSI-HIBÁKNÁL algoritmus helyességét.

19.4-2. Bizonyítsuk be a EGYETÉRTÉS-BIZÁNCI-HIBÁKNÁL algoritmus helyességét.

19.4-3. Bizonyítsuk be a 19.20. tételt.

19.5. Logikai idő, okság és konzisztens állapot

Egy osztott rendszerben gyakran hasznos megállapítani az összes processzorok állapotaiból álló globális állapotot. Ha hozzáférünk a globális állapothoz, megállapításokat tehetünk az összes processzortól függő rendszertulajdonságokról, például észlelhetünk egy holtpontot. Az egyik lehetőség a globális állapot meghatározására az összes processzor megállítása, és az állapotainak összegyűjtése egy központi helyre. Egy ilyen módszer megfelel a legtöbb olyan osztott rendszerben, mely örökké számol. Ez az alfejezet arról szól, hogyan lehet megállapítani a globális állapotot, ami meglehetősen intuitív, ugyanakkor konzisztens egy pontos értelemben.

Először egy olyan osztott algoritmust vizsgálunk, mely processzorok által végrehajtott utasítások egy globális sorrendjét határozza meg. Ez az algoritmus egy olyan illúziót kelt, mintha a processzorok számára egy globális óra állna rendelkezésre. Ezután bevezetjük a más utasításra hatással lévő utasítás fogalmát, és egy olyan algoritmust, amely kiszámítja, hogy melyik

utasítás van hatással melyik másik utasításra. Erről a fogalomról kiderül, hogy nagyon hatásos egy osztott rendszer konzisztens globális állapotának maghatározásában. Az alfejezetet olyan osztott algoritmusokkal zártjuk, melyek egy osztott rendszer egy konzisztens globális állapotát határozzák meg.

19.5.1. Logikai idő

Osztott algoritmusok tervezése könnyebb, ha a processzorok hozzáférnek egy (newtoni) globális órához, mivel az osztott rendszerben előforduló események az óra állásával megcímezhetők, a processzorok megegyezhetnek bármely események sorrendjéről, és ezt az egyetértést az algoritmusok felhasználják döntések meghozatalára. Ugyanakkor egy globális óra elkészítése nehéz. Vannak algoritmusok, melyek közelítik az ideális globális órát helyi hardverórák periodikus szinkronizálásával. Ugyanakkor lehetséges az eseményeket hardver órák használata nélkül teljesen rendezni. Ezt az fogalmat ***logikai órának*** nevezik.

Emlékezzünk, vissza, hogy egy végrehajtás n program utasításainak egymásba fonása. Minden egyes utasítás egy processzor egy számítási lépése, üzenetküldése vagy üzenetfogadása lehet. Bármely utasítást a globális idő egy meghatározott pontján hajtanak végre. Ugyanakkor a globális óra olvasása nem lehetséges a processzorok számára. A célunk az, hogy a logikai óra állását rendeljük az egyes utasításokhoz úgy, hogy ezek az értékek a globális óra értékeinek tűnjenek. Azaz az utasítások végrehajtásának pillanatait előre vagy hátra mozgathatjuk úgy, hogy minden egyes x utasítás, melyhez a lokális óra t_x időt rendeli, pontosan a globális óra t_x időpillanatában hajtódik végre, és a létrejövő végrehajtás érvényes abban az értelemben, hogy ténylegesen előfordulhat, amikor az algoritmust késleltetve futtatják.

A Logikai-óra nevű algoritmus logikai időt rendel minden egyes utasításhoz. Az egyes processzorok rendelkeznek egy *számlálónak* nevezett helyi változóval. Ennek a változónak az értéke kezdetben nulla, és a processzszor minden utasítás végrehajtása után növekszik. Amikor egy processzor üzenetküldéstől és üzenetfogadástól eltérő utasítást hajt végre, a *számláló* értéke pontosan eggyel növekszik. Amikor egy processzor üzenetet küld, a változót eggyel növeli, és az eredményt csatolja az üzenethez. Amikor egy processzor üzenetet fogad, akkor az üzenethez csatolt értéket beolvassa, meghatározza a *számláló* aktuális értékének és a kapott értéknek a maximumát, növeli ezt a maximumértéket eggyel, és hozzárendeli a *számláló* változóhoz. Megjegyezzük, hogy minden utasítás végrehajtásakor a *számláló* változó értéke legalább eggyel növekszik, és tovább nő, amíg a processzor utasításokat hajt végre. Az x utasításhoz rendelt logikai időt a *(számláló,id)*

pár határozza meg, ahol a *számláló* a *számláló* változó értéke közvetlenül az utasítás végrehajtása után, az *id* pedig a processzor azonosítója. A logikai idő értékei egy teljes rendezést alkotnak, ahol a párokat lexikografikusan hasonlítjuk össze. Ezt a logikai időt Lamport-időnek is nevezik. t_x-et a *számláló* $+1/(id+1)$ hányadosként határozzuk meg, ami a pár reprezentálásának egy ekvivalens módja.

19.21. állítás. *Bármely végrehajtás esetén a logikai idő kielégíti az alábbi három feltételt:*

1. *ha egy x utasítást egy processzor egy y utasítás előtt hajt végre, akkor x logikai ideje kisebb, mint y-é.*
2. *bármely két processzor bármely két különböző utasításához különböző logikai időt kell rendelni.*
3. *Ha egy x utasítás egy üzenetet küld, és y fogadja ezt az üzenetet, akkor x logikai ideje kisebb, mint y-é.*

Most az a célunk, hogy belássuk, hogy a logikai óra a processzorok számára a globális óra illúzióját kelti. Intuitív módon egy ilyen illúzió létrehozhatóságának az oka, hogy vehetjük egy determinisztikus algoritmus bármely végrehajtását, kiszámolhatjuk a t_x logikai időt az összes x utasítás esetén, majd újrafuttathatjuk a végrehajtást a processzorok és üzenetek olyan lassításával, illetve felgyorsításával, hogy az x utasítások a globális óra t_x időpillanatában kerülnek végrehajtásra. Így a hardverórához vagy más, a modellünkben nem ismertetett, külső mérőeszközhöz való hozzáférés nélkül a processzorok nem tudják megkülönböztetni a logikai és a globális órák által mutatott időt. Hogy miért érvényes formálisan az újraidőzített végrehajtás, vagyis miért megkülönböztethetetlen az eredeti végrehajtástól, azt az alábbi következményben összegezzük, mely a 19.21. állításból közvetlenül következik.

19.22. következmény. *Bármely α végrehajtás esetén legyen T az utasításokhoz történő logikai idő hozzárendelése, és legyen β az α-beli utasítások logikai idő szerint rendezett sorozata. Ekkor minden processzor esetén, a processzor által α-ban végrehajtott utasítások részsorozata azonos a β-beli részsorozattal. Továbbá, minden egyes, β-ban fogadott üzenet a β-beli elküldés után kerül fogadásra.*

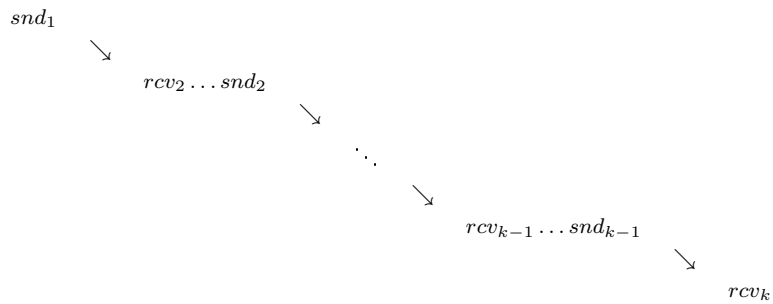
19.5.2. Okság

Egy rendszervégrehajtásban egy utasítás hatással lehet egy másik utasításra úgy, hogy megváltoztatja annak a számításnak az állapotát, melyet a másik

utasítás végrehajt. Azt mondjuk, hogy egy utasítás ***oksági hatással*** (más néven befolyással) van egy másikra, ha az első utasítás által létrehozott információ átadható a másik utasításnak. Emlékezzünk vissza, hogy az osztott rendszerünk modelljében minden utasítást a globális idő egy jól meghatározott pontján hajtunk végre, de a processzorok nem férnek hozzá egy globális órához. Illusztráljuk az okságot. Ha két utasítást ugyanaz a processzor hajt végre, akkor azt mondhatjuk, hogy a korábban végrehajtott utasítás okozati módon befolyásolja a később végrehajtott utasítást, mivel lehetséges, hogy a korábban végrehajtott utasítás eredményét a később végrehajtott utasítást felhasználja. A lehetséges szót hangsúlyoznunk kell, mivel valójában lehet, hogy a későbbi utasítás nem használja a korábbi által előállított információt. Azonban az okság meghatározásánál egyszerűsítjük azt a problémát, hogyan befolyásolnak az egyes processzorok más processzorokat, és csak arra koncentrálunk, hogy mi lehetséges. Ha az x és y utasítást különböző processzorokon hajtjuk végre, azt mondhatjuk, hogy az x utasítás oksági hatással van az y utasításra, amikor az x-et végrehajtó processzor x végrehajtása közben vagy után üzenetet küld, és az üzenet megérkezett y másik processzoron történő végrehajtása előtt vagy közben. Az is lehetséges, hogy a befolyást más processzorok közvetítik vagy a processzorok több utasítást hajtanak végre a másik processzor elérése előtt.

Azt az intuíciót, hogy egy utasítás oksági hatással van egy másikra, formálisan az utasításpárokhoz kapcsolódó ***korábban történt*** reláció segítségével határozzuk meg. A relációt egy adott végrehajtásra határozzuk meg, vagyis az algoritmus által végrehajtott utasítások sorozatát és az utasítások végrehajtása alatti globális idő állásait rögzítjük, majd meghatározzuk, hogy mely utasításpárok elégítik ki a *korábban történt* relációt. A relációt két lépésben vezetjük be. Ha az x és y utasításokat ugyanaz a processzor hajtja végre, úgy pontosan akkor mondjuk azt, hogy x ***korábban történt*** y-nál, ha x y előtt hajtódik végre. Ha x és y különböző processzorokon hajtódik végre, úgy pontosan akkor mondjuk azt, hogy x *korábban történt* y-nál, ha létezik olyan utasítás- és üzenetláncolat $k \geq 2$ esetén, hogy snd_1 vagy egyenlő x-szel vagy ugyanazon processzoron x után hajtódik végre; rcv_k vagy egyenlő y-nal vagy y előtt hajtódik végre ugyanazon a processzoron, mint y, rcv_h snd_h előtt hajtódik végre ugyanazon a processzoron, $2 \leq h < k$, és snd_h egy olyan üzenetet küld, melyet rcv_{h+1} fogad, $1 \leq h < k$ esetén (lásd a 19.1 alfejezetet).

Megjegyezzük, hogy egyetlen utasítás sem hajtódik végre önmaga előtt. Azt, hogy x y előtt történik, úgy jelöljük, hogy $x <_{HB} y$. Arra a végrehajtásra vonatkozó hivatkozást, melyben a reláció definiálva van, kihagyjuk, mivel a környezetből egyértelmű, hogy mely végrehajtást tekintjük. Azt mondjuk, hogy az x és y utasítások ***konkurensek,*** ha sem $x <_{HB} y$, sem pedig $y <_{HB} x$.



19.1. ábra. Utasítás- és üzenetláncolat.

Felmerül a kérdés, hogy a processzorok a meghatározás alapján egy adott végrehajtás mellett hogyan tudják eldönteni, hogy egy utasítás egy másik előtt történik. Ez a korábban bemutatott LOGIKAI-ÓRA algoritmus általánosításával válaszolhatjuk meg. Az általánosítást VEKTORÓRA-nak nevezzük.

A VEKTORÓRA algoritmus lehetővé teszi a processzorok számára, hogy utasításokat kössenek össze, és ez az összekötés pontosan a *korábban történik* relációt adja. Minden P_i processzor fenntart egy n egészből álló V_i vektort. A vektor j-edik koordinátáját $V_i[j]$-vel jelöljük. A vektort kezdeti értéke a $(0, \ldots, 0)$ nullvektor. A vektor mindig módosul, ha egy processzor egy utasítást hajt végre úgy, ahogy a *számláló* módosult a LOGIKAI-ÓRA algoritmusban. Speciálisan, amikor egy P_i processzor végrehajt egy üzenetküldéstől vagy egy üzenetfogadástól eltérő utasítást, a $V_i[j]$ érték eggyel növekszik, míg a más koordinátán lévő értékek változatlanul maradnak. Amikor a processzor üzenetet küld, $V_i[j]$ értékét növeli eggyel, és az eredményül létrejövő V_i vektort csatolja az üzenethez. Amikor a P_j processzor üzenetet kap, beolvassa a csatolt V vektort, és koordinátánként kiszámítja a V és a V_j maximumát a $V_j[j]$ koordináta kivételével, mely utóbbi értékét növeli eggyel. Ezután a keletkező eredményt hozzárendeli a V_j vektorhoz.

$V_j[j] \leftarrow V_j[j] + 1$
minden $k \in [n] \setminus \{j\}$ értékre
$\quad V_j[k] = \max\{V_j[k], V[k]\}$

A P_i processzor által végrehajtott minden x utasítást a V_i vektor utasítás végrehajtása utáni értékével megcímkézzük. A címkét $VT(x)$-vel jelöljük, és *vektor időbélyegnek* nevezzük. Intuitív módon $VT(x)$ a P_i processzor ismereteit reprezentálja arról, hogy az egyes processzorok hány utasítást hajtottak végre abban a pillanatban, amikor P_i az x utasítást hajtotta végre. Ez

a tudás elavult lehet.

A vektor-időbélyegek felhasználhatók a végrehajtott utasítások sorrendbe rendezéséhez. Specifikusan, ha adott két utasítás x és y, és a megfelelő vektor-időbélyegük $VT(x)$ és $VT(y)$, azt írjuk, hogy $x \leq_{VT} y$, amikor a $VT(x)$-t majorálja a $VT(y)$ vektor, azaz minden k-ra a $VT(x)[k]$ koordináta értéke legfeljebb a megfelelő $VT(y)[k]$ értéke. Ha $x \leq_{VT} y$, de $VT(x) \neq VT(y)$, akkor azt írjuk, hogy $x <_{VT} y$.

A következő tétel megmutatja, hogy a VEKTORÓRA algoritmus valóban megvalósítja a *korábban történt* relációt, mivel két utasításról eldönthetjük, hogy egyik a másik előtt hajtódott-e végre vagy sem, egyszerűen úgy, hogy összehasonlítjuk az utasítások időbélyegeit.

19.23. tétel. *Bármely végrehajtás és bármely két x és y utasítás esetén $x <_{HB} y$ akkor és csak akkor, ha $x <_{VT} y$.*

Bizonyítás. Először az előreirányuló következményt látjuk be. Tegyük fel, hogy $x <_{HB} y$. Így x és y két különböző utasítás.

Ha a két utasítást ugyanazon a processzoron hajtják végre, akkor x-et y előtt kell, hogy végrehajtsák. y végrehajtásának idejére csak véges számú utasítást hajtottak végre. A VEKTORÓRA algoritmus eggyel növel egy koordinátát és kiszámolja az x és y közötti utasítások vektor-időbélyegeit az x-hez és y-hoz tartozót is beleértve, és egyetlenegy koordinátaérték sem kerül csökkentésre. Emiatt $x <_{VT} y$.

Ha az x és y utasítást különböző processzorokon hajtják végre, akkor a *korábban történik* reláció meghatározása szerint létezik egy x-ből y-ba vezető utasítás- és üzenetlánc. De a vektoróra algoritmus szerint egy vektor-időbélyeg koordinátájának értéke minden lépésben növekszik, ahogy a láncban előre haladunk, és így szintén $x <_{VT} y$.

Most belátjuk a visszafelé irányuló következményt is. Tegyük fel, hogy nem igaz, hogy $x <_{HB} y$. Tekintsük azt a néhány alesetet, amelynél nem mindig igaz a $x <_{VT} y$ konklúzió.

Először, előfordulhat, hogy x és y ugyanazon utasítás. De ekkor az x-hez és az y-hoz rendelt vektorórák is nyilvánvalóan azonosak, tehát nem igaz az, hogy $x <_{VT} y$.

Tegyük fel tehát, hogy x és y különböző utasítások.

Ha ugyanazon a processzoron hajtják végre őket, akkor x-et nem lehet y előtt végrehajtani, így x y után kerül végrehajtásra. Így, az időbélyegek monotonitása miatt $y <_{VT} x$, és nem igaz, hogy $x <_{VT} y$.

Az utolsó aleset az, amikor x-et és y-t két különböző processzor hajtja végre – P_i és P_j. Koncentráljunk a P_i processzor V_i vektorórájának i-edik komponensére közvetlenül az x végrehajtása után. Legyen ez az érték k. Emlékezzünk vissza, hogy a processzorok az i-edik komponensük értékét csak

úgy tudják növelni, ha átvesznek egy más processzor által küldött értéket. Így ahhoz, hogy a P_j processzor i-edik komponensének értéke k vagy több legyen, y végrehajtásának pillanatában léteznie kell egy olyan, P_i-ből induló, utasításokból és üzenetekből álló láncnak, mely legalább k nagyságú értéket küld. Ez a lánc az x vagy a P_i által x után közvetlenül végrehajtott utasítással kezdődik. De egy ilyen lánc létezése maga után vonja, hogy x y előtt történik, amiről feltettük, hogy nem igaz. Így a $VT(y)$ vektoróra i-edik komponense kisebb, mint a $VT(x)$ vektoróra i-edik komponense. Ezért nem lehet igaz az, hogy $x <_{VT} y$. ■

A tétel szerint úgy dönthetjük el, hogy a különböző x és y utasítások konkurensek-e, hogy ellenőrizzük, hogy $VT(x) < VT(y)$ és $VT(x) > VT(y)$ igaz-e.

19.5.3. Konzisztens állapot

A korábban történik reláció felhasználható az osztott rendszerek egy globális állapotának megállapítására úgy, hogy ez az állapot valamilyen értelemben konzisztens legyen. Rövidesen formálisan meg fogjuk határozni a konzisztencia fogalmát. Minden processzor utasításokat hajt végre. A K ***vágatot*** úgy definiáljuk, mint a nemnegatív egészekből álló cut $K = (k_1, \ldots, k_n)$ vektort. Intuitív módon, a K a processzorok állapotait jelöli. Formálisan k_i azon utasítások száma, melyeket a P_i processzor végrehajtott. Nem minden vágat felel meg osztott processzorok természetesnek vagy konzisztensnek tekintett állapotgyűjteményének. Például, ha a P_i processzor üzenetet kapott P_j-től, és P_i állapotát rögzítjük a vágatban azzal, hogy k_i-t megfelelően nagyra választjuk az üzenet fogadása után, de k_j-t olyan kicsire választjuk, hogy a vágat a küldő azon állapotát tartalmazza, mielőtt az üzenetet elküldte, akkor azt mondhatjuk, hogy az ilyen vágat nem természetes: olyan utasítások vannak a vágatban rögzítve, melyekre a vágatban nem rögzített utasítások oksági hatással vannak. Az ilyen vágatokat nem tekintjük konzisztensnek, s így nem kívánatosak. Formálisan egy $K = (k_1, \ldots, k_n)$ vágat ***inkonzisztens,*** ha léteznek olyan P_i és P_j processzorok, hogy a P_i k_i számú utasítására oksági hatással van a P_j processzor egy k_j utáni utasítása. Így egy inkonzisztens vágatban van olyan üzenet, mely visszafelé „szeli át” a vágatot. A nem inkonzisztens vágatokat *konzisztens vágatnak* nevezzük.

A Konzisztens-vágat algoritmus vektor-időbélyegeket használ egy konzisztens vágat megkeresésére. Feltesszük, hogy minden processzor megkapja ugyanazt a $K = (k_1, \ldots, k_n)$ vágatot, mint bemenetet. Ezután a processzoroknak meg kell határozniuk egy olyan K' konzisztens vágatot, melyet K majorál. Minden P_i processzor rendelkezik egy $VT_i[0, 1, 2, \ldots]$ végtelen vektortáblával. A processzor utasításokat hajt végre, és tárolja a vektor-

időbélyegeket a tábla egymásutáni bejegyzéseiként. Specifikusan, a táblázat m bejegyzése a processzor m-edik végrehajtott utasításához tartozó $VT_i[m]$ vektor-időbélyeg. $VT_i[0]$-t nullvektornak határozzuk meg. A P_i processzor elkezdi egy vágat kiszámítását, miután végrehajtotta a k_i utasítást. A processzor meghatározza azt a legnagyobb $k'_i \geq 0$ számot, amelyik legfeljebb k_i, és amelyre K majorálja a $VT_i[k'_i]$ vektort. A processzorok által együtt megtalált $K' = (k'_1, \ldots, k'_n)$ vektorról kiderül, hogy egy konzisztens vágat.

19.24. tétel. *Bármely K vágat esetén a* Konzisztens-vágat *algoritmus által megtalált K' vágat egy, a K által majorált konzisztens vágat.*

Bizonyítás. Figyeljük meg először, hogy a k_i-n túli VT_i bejegyzésekre nincs szükség. Ezeket a bejegyzéseket nem majorálja K, mivel e vektorok i-edik koordinátája kifejezetten nagyobb, mint k_i. Így valójában a keresésben VT_i első k_i bejegyzésére koncentrálhatunk. Legyen $k'_i \geq 0$ a legnagyobb olyan bejegyzés, hogy a $VT_i[k'_i]$ vektort a K vektor majorálja. Tudjuk, hogy létezik ilyen vektor, mivel $VT_i[0]$ nullvektor, és azt bármilyen K vágat majorálja.

Azt, hogy $(k'_1, \ldots, k'_n)$ konzisztens vágat, ellentmondásra jutással bizonyítjuk. Tegyük fel, hogy a $(k'_1, \ldots, k'_n)$ vektor inkonzisztens vágat. Ekkor definíció szerint vannak olyan P_i és P_j processzorok, melyekre létezik a P_i processzornak egy olyan x utasítása, mely a k'_i számú utasítás után következik, és a végrehajtása a P_j processzor k'_j utasításánál korábban történik. Emlékezzünk vissza, hogy k'_i a legtávolabbi olyan VT_i-beli érték, melyet K majorál. Így a k'_i+1 értéket nem majorálja K, és mivel az összes következő, ideértve az x utasítást is, bejegyzés csak nagyobb koordinátával rendelkezhet, K ezeket sem majorálja. De mivel x végrehajtása a k'_j számú utasítás előtt történik, így a k'_j bejegyzés csak az x-hez tartozó bejegyzés megfelelő koordinátáinál magasabb koordinátával rendelkezhet, s így K nem majorálhatja $VT_j[k'_j]$-t sem. Ez ellentmond annak a feltételnek, hogy K majorálja $VT_j[k'_j]$-t. Tehát $(k'_1, \ldots, k'_n)$-nak konzisztens vágatnak kell lennie. ∎

A konzisztens vágat megtalálására van egy triviális algoritmus. Az algoritmus a $K' = (0, \ldots, 0)$ vágatot választja ki. Azonban a Konzisztens-vágat algoritmus jobb abban az értelemben, hogy a talált konzisztens vágat maximális. Annak belátását, hogy ez valóban igaz, feladatnak hagyjuk.

A konzisztens vágat megtalálására van egy másik mód is. A Konzisztens-vágat algoritmus megköveteli, hogy az üzenetekhez vektor-időbélyegeket csatoljunk, és emlékezzünk azon A algoritmus által eddig végrehajtott összes utasításhoz tartozó vektor-időbélyegekre, melyekhez tartozó konzisztens vágatot ki akarjuk számolni. Ez túl költséges lehet. Az Osztott-pillanatkép algoritmus elkerüli ezeket a költségeket. Az algoritmusban egy processzor úgy kezdeményezi a konzisztens vágat kiszámítását, hogy elárasztja

a hálózatot egy olyan speciális üzenettel, mely úgy viselkedik, mint egy, az A algoritmust konzisztensen vágó kard. Ahhoz, hogy belássuk, a vágat tényleg konzisztens, megköveteljük, hogy az üzenetek fogadása azok küldésének sorrendjében történjen. Egy ilyen rendezés sorozatszámokkal megvalósítható.

Az OSZTOTT-PILLANATKÉP algoritmusban minden P_i processzor rendelkezik egy *számlálónak* nevezett változóval, mely megszámolja az A algoritmusbeli, a processzor által eddig végrehajtott utasításokat. Továbbá, a processzor rendelkezik egy k_i változóval, mely a vágat i-edik koordinátáját tárolja. Ennek a változónak a kezdőértéke $\perp$. Mivel a *számláló* csak az A algoritmus utasításait számlálja, az OSZTOTT-PILLANATKÉP algoritmus utasításai nincsenek hatással a *számláló* változókra. Valamilyen értelemben az OSZTOTT-PILLANATKÉP algoritmus a „háttérben" fut. Tegyük fel, hogy létezik pontosan egy olyan processzor, mely úgy dönthet, pillanatképeket készít az osztott rendszerről. Ha így dönt, a processzor „elárasztja" a hálózatot egy speciális, <*pillanatkép*> üzenettel. Specifikusan, a processzor összes szomszédjának elküldi ezt az üzenetet, és *számláló* változóhoz a k_i értéket rendeli. Amikor a P_j processzor megkapja az üzenetet, és a k_j változó értéke még mindig $\perp$, akkor a processzor összes szomszédjának elküldi a <*pillanatkép*> üzenetet és k_j-hoz az *aktuális* számláló értéket rendeli. A <*pillanatkép*> üzenet küldése és az értékadás közben a processzor nem hajt végre utasítást A-ból. (Az OSZTOTT-PILLANATKÉP algoritmusra úgy tekinthetünk, mint egy „megszakítás"-ra.) Az algoritmus egy konzisztens vágatot számít ki.

19.25. tétel. *Bármely két P_i és P_j processzorra a P_i-ből P_j-be küldött üzenetek érkezzenek meg a küldés sorrendjében. Az* OSZTOTT-PILLANATKÉP *algoritmus végül megtalál egy $(k_1, \ldots, k_n)$ konzisztens vágatot. Az algoritmus $O(e)$ darab üzenetet küld, ahol e a gráf éleinek száma.*

Bizonyítás. Az a tény, hogy a k_i változó valamikor el fog térni $\perp$-től, a modellből következik, mivel feltételezzük, hogy az utasítások valamikor végrehajtódnak és az üzenetek kézbesítésre kerülnek, így a <*pillanatkép*> üzenet valamikor minden csúcshoz megérkezik.

Tegyük fel, hogy $(k_1, \ldots, k_n)$ nem konzisztens vágat. Ekkor létezik egy olyan P_j processzor, hogy a $(k_j + 1)$-edik vagy későbbi utasítás küld egy, a <*pillanatkép*> üzenettől eltérő, $< M >$ üzenetet, és az üzenetet megkapják, mielőtt egy P_i processzor a k_i-edik utasítást hajtja végre, vagy éppen aközben. Így az $< M >$ üzenetet azután kellett elküldeni, miután a P_j elküldte a <*pillanatkép*> üzenetet a P_i-hez. De mivel az üzeneteket a küldés sorrendjében kapják meg, a P_i a <*pillanatkép*> üzenetet $< M >$ előtt dolgozza fel. De ekkor az $< M >$ azután érkezik, hogy pillanatképet vettek P_i-nél. Ez a kívánt ellentmondás. ■

Gyakorlatok

19.5-1. Mutassuk meg, hogy a logikai idő megőrzi a *korábban történik* relációt, azaz mutassuk meg, hogy ha $x <_{HB} y$, akkor $LT(x) < LT(y)$, ahol LT a logikai időt jelöli.

19.5-2. Mutassuk meg, hogy bármely n processzor közti konkurenciát rögzítő vektoróra legalább n koordinátájú.

19.5-3. Mutassuk meg, hogy a KONZISZTENS-VÁGAT algoritmus által kiszámolt K' vektor valójában egy maximális konzisztens vágat, melyet K majorál, azaz nincs olyan K'-tól eltérő K'', mely majorálja K'-t miközben K majorálja K''-t.

19.6. Kommunikációs szolgáltatások

Az üzenetküldéssel kommunikáló processzorokból álló osztott rendszerek alapvető problémái közé tartozik az információk terjesztése és összegyűjtése. Sok kommunikációs hálózatra készített osztott algoritmus felépíthető üzenetszórási és többes üzenetszórási szolgáltatások implementálásával. Ebben az alfejezetben bemutatunk néhány alapvető kommunikációs szolgáltatást az üzenetküldő modellben. Az ilyen szolgáltatásoknak tipikusan valamilyen szolgáltatásminőségi követelményeket kell kielégíteniük, melyek az üzenetsorrendre és a megbízhatóságra vonatkoznak. Először az üzenetszórási szolgáltatásokra koncentrálunk, majd az általánosabb többes szórási szolgáltatásokat tárgyaljuk.

19.6.1. Az üzenetszóró szolgáltatások tulajdonságai

Az üzenetszórási problémában egy választott P_i processzor, melyet ***forrásnak*** vagy küldőnek neveznek, egy m üzenetet akar a rendszerben lévő összes processzorhoz (beleértve a saját magát is) elküldeni. Az üzenetszóró szolgáltatás interfészét a következőképpen határozzák meg:

bc-send$_i(m, qos)$: a P_i processzor egy eseménye, mely egy m üzenetet küld az összes processzornak.

bc-recv$_i(m, j, qos)$: a P_i processzor egy eseménye, mely egy – a P_j processzor által küldött – m üzenetet fogad.

A fenti meghatározásokban a qos a rendszer által nyújtott ***szolgáltatás minőségét*** (**Q**uality **O**f **S**ervice) jelöli.

Kétféle szolgáltatásminőséget fogunk vizsgálni:

Sorrend: hogyan függ az üzenetek fogadásának sorrendje az a forrás által küldött üzenetek sorrendjétől?

Megbízhatóság: hogyan függ a megkapott üzenetek halmaza a rendszerben előforduló hibáktól?

Az üzenetküldésen alapuló osztott rendszerek alapmodellje nem garantál semmit az üzenetek rendezésére vagy a megbízhatóságára nézve. Az alapmodellben csak azt tesszük fel, hogy az összes processzorpárt egy-egy vonal köti össze, és az üzenetkézbesítés független az egyes vonalakon. Az üzenetek megkapásának sorrendje nem áll összefüggésben az üzenetek elküldésének sorrendjével, az üzenetek elveszhetnek a küldők vagy a fogadók megállása miatt.

Bemutatunk néhány hasznos követelményt az üzenetszóró szolgáltatások rendezésére és megbízhatóságára nézve. A fő kérdés az, hogy az alap rendszermodellből kiindulva hogyan implementáljunk egy erősebb szolgáltatást egy gyengébb szolgáltatás tetején.

A rendezésre vonatkozó követelmények változatai

A ***korábban történt*** definícióját üzenetekre alkalmazva azt mondjuk, hogy az m üzenet m'-nél korábban történt, ha vagy m-et és m'-t ugyanaz a processzor küldte, és m-t korábban küldte, mint m'-t, vagy az m-re vonatkozó bc-recv esemény korábban következik be, mint az m'-re vonatkozó bc-send esemény (a *korábban történt* kifejezés utasításokra vonatkozó definícióját a 19.5.2 pontban adtuk meg).

Az üzenetküldések sorrendje szempontjából az általános üzenetszórási szolgáltatásokat négyféleképpen osztályozhatjuk:

Alap üzenetszórás: az üzenetek sorrendje nem garantált.

Egyforrású FIFO (*először be, először ki*): az egy processzor által küldött üzeneteket az egyes processzorok a küldés sorrendjében kapják meg. Pontosabban, minden P_i, P_j processzorpárra és m, m' üzenetekre, ha a P_i processzor m-et m' előtt küldi, akkor a P_j nem kapja meg hamarabb m'-t, mint m-et.

Oksági sorrend: az üzenetek előfordulásuk sorrendjében érkeznek meg. Pontosabban, minden m, m' üzenetpárra és minden P_i processzorra, ha m m' előtt történik, akkor P_i nem kapja meg m'-t m előtt.

Teljes sorrend: az összes processzor a kapott üzeneteknek azonos sorrendjét biztosítja. Pontosabban, minden P_i, P_j processzor-, és minden m, m' üzenetpárra, ha a P_i processzor az m üzenetet m' előtt kapja meg, akkor a P_j processzor sem kapja meg m'-t m előtt.

Könnyű belátni, hogy az oksági sorrend maga után vonja az egyforrású

FIFO követelményeket (mivel a *korábban történik* reláció az üzenetek esetén magában foglalja az egyetlen processzor által küldött üzenetsorrendet), valamint, hogy az összes felsorolt szolgáltatás magától értődően maga után vonja az alap üzenetszórást.

A négy szolgáltatás közt nincs további reláció. Például, vannak olyan végrehajtások, melyek maguk után vonják az egyetlen forrás FIFO tulajdonságot, de az oksági sorrendet nem. Tekintsük a P_0 és P_1 processzorokat. Az első eseményben a P_0 üzenetszórással elküldi az m üzenetet, majd a P_1 processzor megkapja m'-t. Ebből az következik, hogy m m' előtt történik. Ugyanakkor, ha a P_0 processzor m'-t kapja meg m előtt, ami meg is történhet, akkor ez a végrehajtás megsérti az oksági sorrend követelményét. Megjegyezzük, hogy az egyetlen forrás FIFO követelmény triviálisan megáll, hiszen az egyes processzorok csak egy üzenetet küldenek.

Az alapvető üzenetszóró szolgáltatást ***bb***-vel, az egyetlen forrás FIFO-t ***ssf***-fel, az oksági sorrendet ***co***-val, a teljes sorrendet pedig ***to***-val jelöljük.

Megbízhatósági követelmények

A meghibásodás nélküli modellben az üzenetszóró szolgáltatások következő tulajdonságait szeretnénk garantálni:

Integritás: a bc-recv eseményben kapott minden egyes m üzenetet valamilyen bc-send eseményben küldték.

Üzenetek duplikálásának elkerülése: egyetlen processzor sem kap meg egy üzenetet többször.

Létezés: az összes küldött üzenetet megkapja az összes processzor.

A hibák melletti modellben definiáljuk a ***megbízható*** üzenetszóró szolgáltatást, mely kielégíti az integritási, a kettősségmentességi és a kétféle létezési tulajdonságot:

Hibamentes létezés: bármely, egy hibamentes P_i processzor által küldött m üzenetet meg kell, hogy kapjon minden hibamentes processzor.

Meghibásodás melletti létezés: bármely, egy meghibásodott processzor által küldött üzenetet meg kell, hogy kapjon minden hibamentes processzor vagy egyik ilyen processzor sem kaphatja meg az üzenetet.

A megbízható alap üzenetszóró szolgáltatást ***rbb***-vel, a megbízható egyetlen forrás FIFO-t ***rssf***-fel, a megbízható oksági sorrendet ***rco***-val, a megbízható teljes sorrendet pedig ***rto***-val jelöljük.

19.6.2. Rendezett üzenetszóró szolgáltatások

A továbbiakban leírjuk a különböző üzenetszóró szolgáltatások algoritmusainak implementációit.

Alap üzenetszórás megvalósítása aszinkron pont-pont üzenetküldésre épülve

A bb szolgáltatást a következőképpen valósítják meg. A bc-send$_i(m, bb)$ esemény előfordulásakor a P_i processzor az összes vonalon elküldi az m üzenetet P_i-ből P_j-be, ahol $0 \leq i \leq n-1$. Ha egy m üzenet megérkezik a P_j processzorhoz, akkor az engedélyezi a bc-recv$_j(m, i, bb)$ eseményt.

A megbízhatóság biztosításához a következőt tesszük. Létrehozunk egy megbízható üzenetszórás szolgáltatást az alap üzenetszórás szolgáltatásra épülve. Amikor a bc-send$_i(m, rbb)$ esemény bekövetkezik, a processzor engedélyezi a bc-send$_i(\langle m, i\rangle, bb)$ eseményt. Ha a bc-recv$_j(\langle m, i\rangle, k, bb)$ esemény bekövetkezik, és az m üzenetkoordináta először jelenik meg, akkor a P_j processzor először engedélyezi a bc-send$_j(\langle m, i\rangle, bb)$ eseményt (hogy informálja a többi hibamentes processzort az m üzenetről abban az esetben, ha a P_i processzor hibás), majd engedélyezi a bc-recv$_j(m, i, rbb)$ eseményt.

Bebizonyítjuk, hogy a fenti algoritmus megbízhatóságot biztosít az alap üzenetszórás szolgáltatás számára. Először figyeljük meg, hogy az integritás és a kettősségmentesség tulajdonságok közvetlenül következnek abból a tényből, hogy az összes P_j processzor csak akkor engedélyezi bc-recv$_j(m, i, rbb)$-t, ha az m üzenetkoordináta először érkezett. A hibamentes létezés megőrződik, hiszen a hibamentes processzorok közti vonalak helyesen engedélyezik a bc-recv$_j(\cdot, \cdot, bb)$ eseményt.

A meghibásodás melletti létezést az a tény garantálja, hogy ha van egy olyan P_j hibamentes processzor, mely a meghibásodott P_i-től egy m üzenetet kap, akkor a bc-recv$_j(m, i, rbb)$ engedélyezése előtt a P_j processzor a bc-send$_j$ esemény segítségével elküldi az m üzenetet. Mivel P_j hibamentes, az összes hibamentes p_k processzor megkapja az m üzenetet valamely bc-recv$_k(\langle m, i\rangle, \cdot, bb)$ eseményben, majd elfogadja azt (a bc-recv$_k(m, i, rbb)$ esemény engedélyezésével) az első ilyen esemény során.

Egyetlen forrás FIFO megvalósítása az alap üzenetszóró szolgáltatásra épülve

Minden egyes P_i processzor rendelkezik egy számlálóval (időbélyeg), melynek kezdőértéke 0. Ha bekövetkezik a bc-send$_i(m, ssf)$ esemény, akkor a P_i processzor úgy küldi el az m üzenetet, hogy csatolja az aktuális időbélyeget a bc-send$_i(< m, \textit{időbélyeg} >, bb)$ segítségével. Ha egy bc-recv$_j(< m, t >, i, bb)$ esemény bekövetkezik, akkor a P_j processzor engedélyezi a bc-recv$_j(m, i, ssf)$

eseményt éppen a bc-recv$_j(m_0, i, ssf)$, ..., bc-recv$_j(m_{t-1}, i, ssf)$ események engedélyezése után, ahol $m_0, \ldots, m_{t-1}$ azok az üzenetek, melyeknél az bc-recv$_j(< m_0, 0 >, i, bb)$,...,bc-recv$_j(< m_{t-1}, t-1 >, i, bb)$ események engedélyezve vannak.

Megjegyezzük, hogy ha háttérszolgáltatásként a megbízható alap üzenetszórást alkalmazzuk az alap üzenetszórás helyett, az egyetlen forrás FIFO fenti implementációja megbízható egyetlen forrás FIFO szolgáltatást eredményez. A bizonyítást gyakorlatként az Olvasóra bízzuk.

Oksági sorrend és teljes sorrend implementálás az egyetlen forrás FIFO szolgáltatásra épülve

Bemutatunk egy rendezett üzenetszórási algoritmust, mely az egyetlen forrás FIFO üzenetszóró rendszert biztosító aszinkron üzenetküldő rendszerre épül. Ez is időbélyegeket használ, de kifinomultabb módon, mint az *ssf*. Az oksági és teljes rendezési feltételeknek megfelelő szolgáltatást *cto*-val jelöljük.

Minden egyes P_i processzor egy helyi T tömbben tartja nyilván a növekvő számlálóját (időbélyeg) és a többi processzor becsült értékeit. Az időbélyegek arra szolgálnak, hogy küldés előtt ezekkel címkézik meg az üzeneteket: ha P_i üzenetszórással akar küldeni egy üzenetet, növeli az időbélyegét, és ezzel címkézi meg a küldeni kívánt üzenetet (11–13. sorok). A végrehajtás során a P_i processzor megbecsüli a többi processzor időbélyegeinek értékét a T helyi vektorban. Ha a P_i processzor egy t (P_j időbélyege) címkével címkézett üzenetet kap P_j-től, elhelyezi t-t a $T[j]$-ben (23. és 32. sor). A P_i processzor úgy állítja be az aktuális időbélyegét, hogy a T vektorban becsült időbélyegek maximumához hozzáad egyet (24–26. sorok). Az időbélyeg frissítése után a processzor egy frissítés üzenetet küld. Egy processzor akkor fogad el egy t időbélyeggel címkézett m üzenetet a P_j processzortól, ha a (t, j) pár a legkisebb a többi fogadott üzenet között (42. sor) és minden egyes processzor legalább akkora nagy időbélyeggel rendelkezik, mint amit a P_i ismer (43. sor). A részletek az alábbi kódban találhatók.

Rendezett-üzenetszórás

```
    Kód a P_i processzorokra, 0 ≤ i ≤ n − 1
 1  kezdőértékek beállítása
 2      T[j] = 0 minden 0 ≤ j ≤ n − 1 esetén

11  if bc-send_i(m, cto) előfordul then
12      T[i] = T[i] + 1
13      enable bc-send_i(< m, T[i] >, ssf)
```

```
21  if bc-recv_i(< m, t >, j, ssf) előfordul then
22      add (m, t, j) hármas hozzáadása a függőkben levőkhöz
23      T[j] = t
24      if t > T[i] then
25          T[i] ← t
26          enable bc-send_i(< update, T[i] >, ssf)

31  if bc-recv_i(< update, t >, j, ssf) előfordul then
32      T[j] = t

41  if
42      (m, t, j) függő hármas, melyre (t, j) a legkisebb és
43      t ≤ T[k] minden 0 ≤ k ≤ n − 1 esetén
44  then
45      enable bc-recv_i(m, j, cto)
46      remove (m, t, j) hármas eltávolítása a függők listájáról
```

A Rendezett-üzenetszórás algoritmus kielégíti az oksági sorrend követelményt. A bizonyítást az Olvasóra hagyjuk gyakorlatként (a későbbiekben bemutatjuk, hogyan lehet elérni az erősebb megbízható rendezett oksági sorrend szolgáltatást, és a bizonyítást adunk arra az erősebb esetre).

19.26. tétel. *A* Rendezett-üzenetszórás *algoritmus kielégíti a teljes sorrend feltételt.*

Bizonyítás. Az integritás abból a tényből következik, hogy az egyes processzorok csak akkor engedélyezhetik a bc-recv$_i(m, j, cto)$ eseményt, ha a (m, t, j) hármas függő (41–45. sorok), ami csak a P_j processzortól érkező m üzenet fogadása után következhet be.

A kettősségmentesség tulajdonságot az a tény garantálja, hogy legfeljebb egy olyan hármas van, amelyet a P_j processzor küldött és tartalmazza az m üzenetet (13. és 21–22. sorok).

A létezési feltétel kielégítése abból a tényből következik, hogy minden egyes függő hármas a végrehajtásvalamely pillanatában kielégíti a 42–43. sorokban lévő feltételt.

E tény bizonyítása a végrehajtásban szereplő eseményekre vonatkozó indukción alapul. Tegyük fel, ellentmondó módon, hogy a (m, t, j) hármas az, amelyre a (t, j) a legkisebb és a végrehajtás során sohasem elégíti ki a 42–43. sorokban lévő feltételeket. Ebből az következik, hogy van olyan pillanat, mikor a (m, t, j) a legkisebb (t, j) koordinátákkal rendelkezik a P_i processzor

függő hármasai között. Emiatt, ettől a pillanattól fogva valamely k-ra meg kell sértenie a 43. sorban lévő feltételt. Megjegyezzük, hogy a 23–25. sorok frissítési szabályai miatt $k \neq i, j$. Ebből az következik, hogy a P_i processzor sohasem kap olyan üzenetet P_k-tól, mely $(t-1)$-nél nagyobb időbélyeggel van címkézve. Ez a 24–26. sorok frissítési szabályai miatt azt jelenti, hogy a P_k processzor sohasem kapja meg P_j-től az $< m, t >$ üzenetet, ami ellentmond a *ssf* szolgáltatás létezési feltételének.

A teljes rendezés tulajdonság bizonyításához elég azt belátni, hogy minden P_i processzor és minden, a P_k, P_l processzorok által t, t' időbélyeggel küldött megfelelő m, m' üzenet esetén a (m, t, k), (m', t', l) hármasokat $(t, k), (t', l)$ lexikografikus sorrendjében elfogadják. Két eset van.

1. eset. A végrehajtás valamely pillanatában mindkét hármas függő lesz a P_i processzornál. Ekkor a 42. sorban lévő feltétel garantálja a $(t, k), (t', l)$ sorrendben történő elfogadást.

2. eset. Az általánosság megszorítása nélkül feltehetjük, hogy a P_i processzor elfogadja a (m, t, k) hármast mielőtt a (m', t', l) hármas függő lesz. Ha $(t, k) < (t', l)$, akkor az elfogadás még mindig a $(t, k), (t', l)$ sorrendnek felel meg. Máskülönben $(t, k) > (t', l)$, és a 43. sor feltétele miatt azt kapjuk, hogy $t \leq T[l]$ és következményképpen $t' \leq T[l]$. Ez nem fordulhat elő az *ssf* követelmény és amiatt a feltételezés miatt, hogy a P_i processzor még nem kapta meg az $< m', t' >$ üzenetet l-től az *ssf* üzenetszóró szolgáltatáson keresztül. ■

Most az oksági rendezés és a teljes rendezés szolgáltatások megbízható változatait vizsgáljuk. A megbízható oksági rendezés követelményei a processzormegállás melletti aszinkron üzenetküldő rendszerben a következő algoritmussal valósíthatók meg a megbízható alap üzenetszórásra épülve. Ugyanazok az adatszerkezetek, mint a korábbi rendezett üzenetszóró algoritmusnál. A MEGBÍZHATÓ-OKSÁGI-RENDEZÉS algoritmus és a RENDEZETT-ÜZENETSZÓRÁS algoritmus között fő különbségek a következőek: Az egészből álló időbélyegek helyett vektoros T időbélyeget használ és nem becsüli más processzorok időbélyegeit, csak lexikografikusan összehasonlítja a saját (vektoros) időbélyegeit a kapottakkal. A P_i processzor vektoros időbélyegei mögötti intuíció az, hogy az tárolja azt az információt, hogy hány üzenetet küldött P_i és hány üzenetet fogadott el a P_i az egyes P_k-któl, ahol $k \neq i$.

Az algoritmus végrehajtása folyamán a P_i processzor azelőtt növeli a megfelelő i pozíciót a T időbélyeg-vektorában, mielőtt egy új üzenetet küld (12. sor), valamint növeli a vektor-időbélyeg j-edik pozícióját, miután egy új üzenetet fogadott el P_j-től (38. sor). Egy új, a P_j-től származó, $\hat{T}$ vektor-időbélyeggel rendelkező, üzenet fogadása után P_i felveszi a $(m, \hat{T}, j)$ hármast a függők listájára, és akkor fogadja el ezt a hármast, ha az a P_j-től származó

első, el nem fogadott üzenet (feltétel a 33. sorban), és a P_j-től származó elfogadott üzenetek száma (a $P_k \neq P_i$ processzorokra) m küldésének pillanatában nem nagyobb, mint a P_i aktuális időpillanatában (feltétel a 34. sorban). A következőkben bemutatjuk az algoritmus részletes kódját.

MEGBÍZHATÓ-OKSÁGI-RENDEZÉSŰ-ÜZENETSZÓRÁS

```
     Kód a P_i processzorokra, 0 ≤ i ≤ n − 1 esetén
 1   kezdőértékek beállítása
 2       T[j] = 0 minden 0 ≤ j ≤ n − 1 esetén
 3       a függőek listája üres

11   if bc-send_i(m, rco) előfordul then
12       T[i] = T[i] + 1
13       enable bc-send_i(< m, T >, rbb)
21   if bc-recv_i(< m, T̂ >, j, rbb) előfordul then
22       add (m, T̂, j) hármas felvétele a függőek listájába

31   if
32       (m, T̂, j) függő hármas és
33       T̂[j] = T[j] + 1, és
34       T̂[k] ≤ T[k] minden k ≠ i esetén
35   then
36       enable bc-recv_i(m, j, rco)
37       remove (m, T̂, j) hármas eltávolítása a függőek listájából
38       T[j] ← T[j] + 1
```

Most bebizonyítjuk, hogy a MEGBÍZHATÓ-OKSÁGI-RENDEZÉSŰ-ÜZENETSZÓRÁS algoritmus megbízható oksági üzenetszórási szolgáltatást nyújt a megbízható alap üzenetszórásra épülő rendszerben. Az integritás és a kettősségmentesség tulajdonságokat az *rbb* üzenetszóró szolgáltatás és azok a tények garantálják, hogy minden egyes üzenet legalább egyszer felkerül a függők listájára, és a nem fogadott üzenetek sohosem kerülnek fel erre a listára. A hibamentesség és a hiba melletti létezési tulajdonságokat a végrehajtáson vett indukcióval lehet bizonyítani azon tények felhasználásával, hogy a hibamentes processzorok az összes küldött üzenetet megkapják, melyek garantálják, hogy a 33–34. sorokban szereplő feltételek valamikor végül teljesülnek. Az oksági sorrend feltétel teljesül, mivel ha egy m üzenet m' előtt történik, akkor minden egyes P_i processzor az m, illetve m' üzenetekhez tartozó vektorok, azaz $\hat{T}$ és $\hat{T}'$ lexikografikus sorrendjének megfelelően fogadja el az m, m' üzeneteket, valamint ebben az esetben e

vektortömbök összehasonlíthatóak. A bizonyítás részleteit az Olvasóra bízzuk (lásd 19.6-6 gyakorlat).

Megjegyezzük, hogy a megbízható teljes rendezéses üzenetszórás szolgáltatás nem valósítható meg a processzor-meghibásodások melletti általános aszinkron beállítások mellett, mivel ez megoldaná az egyetértési problémát ebben a modellben. Az először elfogadott üzenet határozná meg a döntési értéket (ami ellentmond annak a ténynek, hogy az egyetértés nem oldható meg az általános modellben – lásd a 19.4.6. pontban).

19.6.3. Többes üzenetküldő szolgáltatások

A többes üzenetküldő szolgáltatások hasonlóak az üzenetszóró alkalmazásokhoz azzal a különbséggel, hogy az egyes többesküldésű üzenetek címzettje a processzorok halmazának egy adott részhalmaza. A többes üzenetküldő szolgáltatásokban kétfajta esemény van, ahol a *qos* a megkívánt szolgáltatásminőséget jelöli:

mc-send$_i(m, D, qos)$: a P_i processzor egy eseménye, mely az m üzenetet küldi az azonosítókkal együtt a $D \subseteq \{0, \ldots, n-1\}$ célhalmazban szereplő processzorokhoz.

mc-recv$_i(m, j, qos)$: a P_i processzor egy eseménye, mely a P_j processzor által küldött m üzenetet fogadja.

Megjegyezzük, hogy az mc-recv esemény hasonló a bc-recv eseményhez.

Az üzenetszórásos szolgáltatás esetén is hasznos rendezési és megbízhatósági tulajdonságokat szeretnénk nyújtani a többes küldés szolgáltatások számára. A rendezési követelményeket átvehetjük az üzenetszóró szolgáltatásoktól. Az alap többes üzenetküldés nem igényel rendezési tulajdonságokat. Az Egyforrású FIFO megköveteli, hogy ha egy processzor többes küldést végez (valószínűleg különböző célhalmazokba), akkor az egyes processzorok által fogadott üzeneteket (ha léteznek) ugyanabban a sorrendben kell fogadni, amelyben a forrás azokat küldte. Az oksági sorrend meghatározása ugyanaz marad. A teljes sorrend helyett, melyet a célhalmazok különbözősége miatt nehéz elérni, egy másik rendezési tulajdonságot definiálunk.

Részben teljes rendezés: az összes processzor által fogadott üzenetek sorrendje kiterjeszthető az üzenetek teljes rendezésére. Pontosabban, bármely m, m' üzenet, és P_i, P_j processzorpárra ha P_i és P_j is fogadta az m, m' üzeneteket, akkor azokat P_i és P_j is ugyanabban a sorrendben fogadta.

A többes üzenetküldés megbízhatósági tulajdonságai valamelyest eltérnek a megbízható üzenetszórásra vonatkozó feltételektől.

Integritás: minden, mc-recv$_i$ esemény során fogadott m üzenetet olyan mc-

send esemény küldött, melyben a P_i processzor szerepelt a célhalmazban.

Üzenetek duplikálásának elkerülése: egyik processzor sem kap meg egy üzenetet többször.

Hibamentes létezés: minden, a P_i hibamentes processzor által küldött m üzenetet megkap minden, a célhalmazban szereplő, processzor.

Meghibásodás melletti létezés: bármilyen üzenetet, melyet egy meghibásodott processzor küldött, vagy megkap minden, a célhalmazban szereplő, hibamentes processzor, vagy közülük egyik sem kapja meg.

A rendezett és megbízható többes küldés implementálásának egyik módja a megfelelő üzenetszórási szolgáltatás felhasználása. (A *részben teljes rendezés* esetén megfelelő üzenetszórási követelmény a teljes rendezés.) Pontosabban, az mc-send$_i(m, D, qos)$ esemény előfordulásakor a P_i processzor engedélyezi a bc-send$_i(< m, D >, qos)$ eseményt. Egy bc-recv$_j(< m, D >, i, qos)$ esemény előfordulásakor a P_j processzor engedélyezi az mc-recv$_j(m, i, qos)$ eseményt, ha $P_j \in D$, különben figyelmen kívül hagyja az eseményt. Annak bizonyítását, hogy egy ilyen módszer biztosítja a kívánt szolgáltatásminőségi követelményt, gyakorlatként az Olvasóra bízzuk.

Gyakorlatok

19.6-1. Tervezzünk futtatást azt bemutatandó, hogy nincs kapcsolat az Oksági sorrend és a Teljes sorrend között, valamint az Egyforrású FIFO és a Teljes sorrendű üzenetszóró szolgáltatások között. Az egyszerűség kedvéért tekintsünk két processzort és két elküldött üzenetet.

19.6-2. Az Egyforrású FIFO-t és az Oksági sorrend követelményeit kielégítő üzenetszóró szolgáltatás kielégíti a Teljes sorrend tulajdonságot? Az Egyforrású FIFO-t és a Teljes sorrend követelményeit kielégítő üzenetszóró szolgáltatás kielégíti az Oksági sorrend tulajdonságot? Ha igen, adjunk rá bizonyítást, ha nem, mutassunk be egy ellenpéldát.

19.6-3. Mutassuk meg, hogy Egyforrású FIFO szolgáltatás megvalósításában Alap-üzenetszórás helyett Megbízható-alap-üzenetszórás-t alkalmazva megbízható Egyforrású FIFO üzenetszórást kapunk.

19.6-4. Bizonyítsuk be, hogy Rendezett-üzenetszórás algoritmus az Egyforrású FIFO szolgáltatáson alapulva oksági sorrendű szolgáltatást valósít meg.

19.6-5. Mennyi a Rendezett-üzenetszóró-szolgáltatás algoritmusban csúcsról csúcsra küldött üzenetek teljes száma k számú üzenetszórás esetén?

19.6-6. Bizonyítsuk be részletesen, hogy a Megbízható-oksági-rendezésű-üzenetszórás algoritmus megbízható oksági rendezésű üzenetszórást biztosít a megbízható alap üzenetszórásra épülő rendszerben.

19.6-7. Becsüljük meg a MEGBÍZHATÓ-OKSÁGI-SORRENDŰ-ÜZENETSZÓRÁS végrehajtása közben csúcsról csúcsra küldött üzenetek teljes számát, ha az k számú üzenetszórást hajt végre, és a végrehajtás során $f < n$ processzor omlik össze.

19.6-8. Mutassuk be a MEGBÍZHATÓ-OKSÁGI-SORRENDŰ-ÜZENETSZÓRÁS algoritmus egy olyan végrehajtási sorozatát, amely megsérti a Teljes sorrend követelményt.

19.6-9. Írjunk kódot megbízható részleges sorrendű többes üzenetküldés szolgáltatás megvalósítására.

19.6-10. Mutassuk meg, hogy a megfelelő üzenetszolgáltatásra épülő többes üzenetküldés megvalósításának leírt módszere helyes.

19.7. Szóbeszédgyűjtő algoritmusok

A fejlettebb kommunikációs problémák algoritmusának összeállításánál a megbízható többes üzenetküldő szolgáltatások felhasználhatók építő blokként. Ebben a fejezetben ezt a módszert mutatjuk be a szóbeszédek megállásra hajlamos szinkron processzorokkal történő gyűjtésének problematikájára vonatkozóan. (Mivel csak tiszta végrehajtással foglalkozunk, feltételezzük, hogy legalább egy processzor működőképes marad a számítások végéig.)

19.7.1. Szóbeszédgyűjtési (pletyka) probléma és követelményei

A ***szóbeszédgyűjtés*** vagy *pletyka* klasszikus problémája a következőképpen határozható meg: Kezdetben minden processzor a saját részinformációjával rendelkezik, nevezzük ezt ***szóbeszédnek.*** A cél az, hogy minden egyes processzorral tudassuk az összes szóbeszédet.

A processzor megállásos modellben mindemellett szükségünk van a pletyka probléma átfogalmazására úgy, hogy tekintetbe vegyük a processzorok megállását. Mind az integritás, mind a duplikált üzenetek elkerülése tulajdonság ugyanaz itt is, mint a megbízható üzenetszóró szolgáltatásban, az egyetlen különbség (ami következik a pletyka probléma specifikációjából) a létezés követelményei között van:

Hibamentes létezés: Minden egyes hibamentes processzornak ismernie kell minden hibamentes processzor szóbeszédét.

Meghibásodás melletti létezés: Ha a P_i processzor végrehajtás közben összeomlott, valamennyi hibamentes processzor vagy tudja a P_i szóbeszédét, vagy azt tudja, hogy a P_i összeomlott.

A *pletyka* algoritmusok hatékonyságát a futási idővel és az üzenetszám-

mal jellemezzük. A futási idő méri a (szinkron) lépések számát a kezdettől a befejezésig. Az üzenetszám méri a csúcsról csúcsra küldött üzenetek teljes számát (pontosabban ha egy processzor három további processzornak küld el egy üzenetet, ez az üzenetszám szempontjából háromnak számít).

A következő egyszerű algoritmus mindössze egy szinkron lépésben végzi el a pletyka funkciót: minden egyes processzor elküldi a saját szóbeszédét az összes processzornak. Az algoritmus kifogástalan, mivel minden egyes megkapott üzenet tartalmaz egy szóbeszédet, és egy meg nem érkezett üzenet a küldő hibáját jelenti. Egy ilyen megoldásnak az a hátránya, hogy négyzetes számú üzenetelküldést kíván, ami igen rossz hatékonyságot jelent.

Mi úgy szeretnénk lefolytatni a pletykát, hogy ne csak gyors legyen, hanem kevesebb üzenetet kelljen csúcsról csúcsra küldeni. Van egy magától értetődő kompromisszum az idő és a kommunikáció között. Figyeljük meg, hogy egy processzormegállás-mentes rendszerben egy ilyen kompromisszum elérhető például az üzeneteknek egy (majdnem) teljes bináris fán való küldésével, ekkor a futási idő $O(\lg n)$, míg az üzenetszám $O(n \lg n)$. Így tehát a futási idő kismértékű növelésével elérhetünk egy közel lineáris növekedést az üzenetszámot illetően.

Ha az alap kommunikációs hálózat komponensei meghibásodhatnak, akkor a szabálytalan hibaminták zavarják az információáramlást, ezzel jóval hosszabbá téve a pletykázás folyamatát. Ebben az alfejezetben azzal a kérdéssel foglalkozunk, hogy mi a legmegfelelőbb kompromisszum a futási idő és az üzenetszám között egy processzor-megállást megengedő modellben.

19.7.2. Hatékony pletyka algoritmus

Ebben a részben a *pletyka* algoritmusoknak a családját írjuk le, amelyben találhatunk néhány hatékonyat is. Mindegyikük ugyanazon az általános kódon alapszik, hatékonyságuk pedig az általános algoritmusba beépített két adatstruktúra minőségétől függ. Célunk annak bebizonyítása, hogy találhatunk néhány olyan adatstruktúrát, amellyel kapott algoritmus mindig helyes, valamint hatékony is, ha a végrehajtás alatti megállások száma legfeljebb f, ahol $f \leq n - 1$ egy paraméter.

Az említett két struktúra, a kommunikációs gráf és a kommunikációs ütemezések leírásával kezdjük.

Kommunikációs gráf

Egy $G = (V, E)$ gráf csúcsok V halmazából és élek E halmazából áll. Ebben a fejezetben gráfon mindig ***egyszerű gráfot*** értünk, ami azt jelenti, hogy minden él egy csúcspár, és az élhez nincs irány rendelve. A gráfok kommunikációs minták leírására szolgálnak. Egy gráf csúcsainak V halmazát a szóban forgó

osztott rendszer processzorai alkotják. Az E-ben lévő élek azon processzorpárokat határozzák meg, amelyek üzenetváltás útján egymással közvetlenül kommunikálnak, ez azonban nem feltétlenül jelenti azt, hogy közöttük létezik fizikai kapcsolat. A kommunikációs mechanizmust vonatkoztatjuk el: lehet, hogy az – az E-ben lévő éllel összekötött csúcsokon váltott – üzeneteket továbbítani kell és lehet, hogy át kell vinni a kommunikációs hálózat alapjául szolgáló – esetleg hosszú – útvonalon.

Egy adott n számú processzorhoz általunk használt gráftopológiák az egy végrehajtás közben tolerálni kívánt megállások f felső korlátjától függően különbözőek. A végrehajtás egy adott pontján ez egy olyan gráf, amelyet azon processzorok hoztak létre, amelyek a végrehajtás e lépéséig még nem álltak meg.

Ahhoz, hogy hatékony algoritmust kapjunk, a kommunikációs gráfnak ki rendelkeznie kell néhány szükséges tulajdonsággal, például a következő $\mathcal{R}(n,f)$ tulajdonsággal:

19.27. definíció. *Legyen $f < n$ pozitív egészek egy párja. A G gráfról azt mondjuk, hogy kielégíti az $\mathcal{R}(n,f)$ tulajdonságot, ha G-nek van n gráfpontja, és ha minden legalább $n-f$ méretű $R \subseteq G$ részgráfhoz létezik G-nek egy olyan $P(R)$ részgráfja, amelyre az alábbiak teljesülnek:*

1:	$P(R) \subseteq R$	– öröklődés
2:	$\lvert P(R)\rvert = \lvert R\rvert/7$	– nagy méret
3:	*A $P(R)$ átmérője legfeljebb* $2 + 30 \ln n$	– logaritmikus kommunikáció
4:	*Ha* $R_1 \subseteq R_2$, *akkor* $P(R_1) \subseteq P(R_2)$	– monotonitás.

Figyeljük meg, hogy $P(R)$ egy összefüggő gráf – akkor is, ha R nem az –, mivel átmérője véges. A következő eredmény igazolja, hogy létrehozhatók $\mathcal{R}(n,f)$ tulajdonságot kielégítő gráfok.

19.28. tétel. *Minden $f < n$-hez létezik $\mathcal{R}(n,f)$ tulajdonságot kielégítő $G(n,f)$ gráf. A $G(n,f)$ gráf Δ maximális fokszáma $O(n/(n-f))^{1.837}$.*

Kommunikációütemezés

A ***lokális permutáció*** a $[0\,..\,n-1]$ intervallumban lévő egész számok permutációja. Feltesszük, hogy a számítást megelőzően létezik az n lokális permutációk egy adott Π halmaza. Minden P_i processzornak létezik a Π-ből egy ilyen π_i permutációja. Az egyszerűség kedvéért tegyük fel, hogy $\pi_i(0) = P_i$. A lokális permutáció a szóbeszéd módszeres – a permutáció által megadott sorrendben való – összegyűjtésére alkalmazható, míg a kommunikációs gráf inkább a már összegyűjtött szóbeszédek cseréjére használható nagy és összefüggő, hibamentes gráfkomponensekben.

Általános algoritmus

Annak a célnak a meghatározásával kezdjük, amit a pletykáló algoritmusnak teljesítenie kell.

Akkor mondjuk, hogy a P_i ***processzor már hallott a*** P_j ***processzorról,*** ha P_i ismeri a P_j eredeti bemeneti szóbeszédét, vagy ha P_i tudja, hogy P_j már meghibásodott. Újraszövegezhetjük a pletykáló algoritmus helyességét abból a szempontból, hogy hallott-e már a többi processzorról: az algoritmus helyes, ha teljesíti az integritás és a kettősségmentesség tulajdonságokat, és ha az algoritmus befejeztével minden processzor hallott minden másik processzorról.

A pletykáló algoritmus kódja tartalmaz olyan objektumokat, amelyek függenek a rendszerben lévő processzorok n számától, valamint a „hatékonyan tolerált" meghibásodások $f < n$ felső korlátjától (ha a meghibásodások száma maximum f, akkor a tervezőalgoritmus üzenetbonyoultsága kicsi). A hozzávett paraméter egy τ megállási küszöb, ami befolyásolja a általános pletyka séma adott implementációjának futási idejét. Célunk egy olyan ÁLTALÁNOS-PLETYKA algoritmus tervezése, amely helyes bármely hozzávett f, τ paraméter, tetszőleges kommunikációs gráf és ütemezési halmaz esetén, miközben hatékonyf, τ bizonyos értékeire, valamint bizonyos $G(n, f)$ és Π struktúrákra.

Minden processzor ***gyűjtőként*** kezd pletykálni. Egy gyűjtő processzor a többi processzor szóbeszédeire vonatkozó információk után kutat – közvetlen kérdéseket küldve néhányukhoz. A gyűjtő, miután minden processzorról hallott már, ***terjesztővé*** válik. Az ezzel a státusszal rendelkező processzorok terjesztik ismereteiket – lokális véleményeket küldve kiválasztott másik processzoroknak.

Lokális vélemény

Minden P_i processzor kezdetben csak a saját azonosítóját, valamint saját bemenő ***szóbeszéd***$_i$ információját ismeri. A beérkező adatok tárolására a P_i processzor a következő tömböket építi fel: Szóbeszédek$_i$, Aktív$_i$ és Függő$_i$. Mindegyik töm mérete n. Ezen tömbök mindegyike kezdeti értékként a NIL értéket tárolja. A P_i processzor egy X_i tömbjének j-edik bejegyzését $\mathsf{X}_i[j]$-vel jelöljük – természetszerűen ez a bejegyzés a P_j processzorról tartalmaz valamilyen információt. A Szóbeszéd tömb a processzor által ismert összes szóbeszéd tárolására szolgál. Kezdetben a P_i processzor a Szóbeszédek$_i[i]$ értékét a saját bemenő *szóbeszéd*$_i$ értékére állítja. Minden alkalommal, amikor a P_i processzor megtud valamilyen *szóbeszéd*$_j$ információt, azonnal átállítja a Szóbeszédek$_i[j]$ értékét erre az értékre. Az Aktív tömb azon processzorok halmazát tárolja, amelyekről a tömb tulajdonosa úgy tudja, hogy összeomlott. Amikor a P_i processzor értesül arról, hogy a P_j processzor meghibásodott,

Aktív$_i[j]$ értékét azonnal *meghibásodott* értékre állítja. Vegyük észre, hogy a P_i processzor akkor hallott a P_j processzorról, ha a Szóbeszédek$_i[j]$ és Aktív$_i[j]$ értékek valamelyike nem egyenlő a NIL értékkel.

A Függő tömb rendeltetése az, hogy elősegítse a terjesztést. Minden alkalommal, amikor a P_i processzor értesül arról, hogy valamely másik P_j processzor teljesen informált, azaz hogy az vagy egy terjesztő, vagy egy terjesztőként bejelentett processzor, ez az információ bekerül a Függő$_i[j]$ értékbe. A P_i processzor arra használja a Függő$_i$ tömböt, hogy szisztematikus módon küldhessen terjesztő üzeneteket, s teszi ezt úgy, hogy végignézi a Függő$_i$ tömböt, hogy megtalálja azokat a processzorokat, amelyek feltételezhetően még nem hallottak valamely másik processzorról.

A következő egy hasznos fogalommeghatározás az Aktív és a Függő tömb aktuális tartalmáról. A p_j processzor p_i ***szerint aktív,*** ha a p_i még nem kapott arra vonatkozó információt, hogy p_j összeomlott; ami azonos azzal, hogy az Aktív$_i[j]$ értéke NIL. A p_j processzort a p_i ***által értesítendőnek*** nevezzük, ha az a p_i szerint aktív, és ha a Függő$_i[j]$ értéke NIL. **Fázisok.** A pletykáló algoritmus egy végrehajtása a processzorok összes lokális objektumot inicializáló műveletével kezdődik. Ap_i processzor Szóbeszédek$_i$ listáját a NIL kezdeti értékkel tölti fel minden helyen, kivéve az i-ediket, ahol az érték $szóbeszéd_i$ lesz. A végrehajtás fennmaradó része egy ciklusként épül fel, amelyben fázisok ismétlődnek. Minden fázis három részből áll: üzenetek fogadása, helyi számítás, csoportos üzenetek. A fázisoknak két fajtája van: ***normál fázis*** és ***záró fázis***. Egy normál fázis közben a processzor üzeneteket fogad, frissíti helyi ismereteit, ellenőrzi a státuszukat, valamint elküldi ismereteit, a szóbeszédekkel kapcsolatos kérdéseit és a saját szóbeszédével kapcsolatos válaszokat a kommunikációs gráfban lévő szomszédaihoz. A záró fázis közben a processzor üzeneteket fogad, kérdéseket küld az összes olyan processzorhoz, amelyekről eddig még nem hallott, és válaszokat küld a saját szóbeszédével kapcsolatosan. A normál fázis τ-szor hajtódik végre; a τ szám egy ***megállási küszöb.*** Ezek után a záró fázis négyszer hajtódik végre. Ez egy általános pletykáló algoritmust határoz meg.

ÁLTALÁNOS-PLETYKA

```
    Kód a P_i, 0 ≤ i ≤ n − 1 processzorok esetén
1   kezdőértékek beállítása
2       a P_i processzor gyűjtővé válik
3       Szóbeszédek_i, Aktív_i és Függő_i tömbök kezdőértékének megadása
```

11 **repeat** τ-szor
12 normál fázis végrehajtása

20 **repeat** 4-szer
21 záró fázis végrehajtása

Most leírjuk a normál és a zárófázisban alkalmazott kommunikációt és üzenetfajtákat.

A normál fázis alatt használt gráf- és tartományüzenetek

A P_i processzor küldhet üzenetet a $G(n, f)$ gráfban lévő szomszédjának, amennyiben az P_i szerint aktív. Az ilyen üzenetet ***gráfüzenetnek*** hívjuk. Csak ilyen üzeneteket küldve a pletykálást nem szükségszerűen végzi el teljesen, mivel a megállások következtében a kommunikációs gráf válhat nem összefüggő gráffá. Így tehát másféle üzeneteket is kell küldeni annak érdekében, hogy szisztematikusan le tudjuk fedni az összes processzort. Egy ilyen típusú kommunikációban a P_i processzor a processzorokat a saját lokális π_i permutációja szerinti, azaz $\pi_i(0), \pi_i(1), \ldots, \pi_i(n-1)$ sorrendben rendezetteknek tekinti. Az ebben a folyamatban elküldött néhány további üzenetet ***tartományüzenetnek*** hívjuk.

A normál fázis alatt a processzorok a következő típusú tartományüzeneteket küldik: érdeklődő, válasz és értesítő üzenet. A P_i gyűjtő küld egy ***érdeklődő*** üzenetet az első olyan processzornak, amelyikről P_i eddig még nem hallott. Egy ilyen üzenet minden fogadója visszaküld egy tartományüzenetet, amelyet ***válaszüzenetnek*** nevezünk.

A terjesztők szintén küldenek a processzorok egy részhalmazának tartományüzeneteket. Az ilyen üzeneteket kiértesítő üzeneteknek nevezzük. A P_i terjesztő által kiválasztott célprocesszor az első olyan processzor, amit P_i-nek még értesítenie kell. Az értesítő üzeneteket nem szükséges megválaszolni: A küldő már ismeri az összes, szerinte aktív processzor szóbeszédét, az üzenet célja pedig az ismeretterjesztés.

A normál fázis pszeudokódja a következő oldalon van.

Utolsó remény üzenetek használata a záró fázis alatt

A záró fázis alatt küldött üzeneteket ***utolsó remény*** üzeneteknek nevezzük. Ezen üzenetek az érdeklődő, válasz és értesítő kategóriákba sorolhatók csakúgy, mint a megfelelő tartományüzenetek, mivel ugyanazokat a célokat szolgálják. Azok a gyűjtők, amelyek nem hallottak még néhány processzorról, közvetlen érdeklődő üzenetet küldenek egyszerre az *összes* ilyen processzorhoz. Ezeket az üzeneteket ***érdeklődő*** üzeneteknek nevezzük. Ezeket

az üzenetek a meg nem hibásodott fogadók a következő lépésben megválaszolják egy ***válasz*** üzenet küldésével. A következő fázisban minden egyes terjesztő küld egy üzenetet az *összes* általa értesítendő processzornak. Az ilyen üzeneteket ***értesítő*** üzeneteknek nevezzük.

A normál fázis egy lépésében egy processzor által küldött gráfüzenetek száma legfeljebb olyan nagy, mint a maximális csúcsok száma a kommunikációs gráfban. A normál fázis egy lépésében egy processzor által küldött tartományüzenetek száma legfeljebb olyan nagy, mint a fogadott érdeklődő üzenetek száma plusz egy konstans – így az összes processzor által a normál fázisokban elküldött üzenetek teljes száma számítható úgy, hogy az az elküldött érdeklődések számának (ami fázisonként és processzoronként egy) konstansszorosa. Ezzel ellentétben azonban nincs *eleve meghatározott* felső korlátja a záró fázis közben elküldött üzenetek számának. A τ megállási küszöb elég nagyra történő választásával ellenőrizhető, hogy a záró fázisban hány szóbeszédet lenne szükséges még begyűjteni.

Normál-fázis

Kód a P_i, $0 \leq i \leq n-1$ processzorok esetén
1 **receive** üzenetek

11 helyi számítások elvégzése
12 helyi tömbök frissítése
13 **if** P_i egy gyűjtő, ami az összes processzorról hallott már
14 **then** p_i terjesztővé válik
15 a célprocesszorok halmazának meghatározása:
for minden P_j processzorra
16 **if** P_j a P_i szerint aktív és P_j a $G(n,t)$ gráfban szomszédja P_i-nek
17 **then** adja hozzá P_j-t egy gráfüzenet célhalmazához
18 **if** P_i egy gyűjtő és P_j azon első processzor,
19 amelyről P_i még nem hallott
20 **then** küldjön egy érdeklődő üzenetet p_j-nek
21 **if** P_i terjesztő és P_j azon első processzor,
22 amely P_i által értesítendő
23 **then** küldjön egy értesítő üzenetet p_j-nek
24 **if** P_j egy gyűjtő, amelytől érdeklődő üzenet
érkezett e fázis fogadó fázisában
25 **then** küldjön egy válasz üzenetet P_j-nek

30 **send** gráf/érdeklődő/értesítő/válasz üzenet küldése
a célhalmazra vonatkozóan

Lokális vélemény frissítése

Egy processzor által elküldött üzenet magával hordozza a saját jelenlegi lokális ismereteit. Pontosabban a P_i processzor által küldött üzenet a következőket hordozza magával: a P_i azonosítót, a $\mathsf{Szóbeszédek}_i$, az $\mathsf{Aktív}_i$ és a $\mathsf{Függő}_i$ tömböt, valamint egy címkét, amely a fogadót értesíti az üzenet karakteréről. A címke a következők egyike lehet: *gráfüzenet, érdeklődés_gyűjtőtől, értesítés_terjesztőtől, ez_egy_válasz* – nevük magyarázza jelentésüket. Egy P_i processzor újonnan érkezett, valamely P_j processzor által küldött üzenet után kutat, hogy ismereteket szerezzen szóbeszédekről, meghibásodásokról, valamint más processzorok jelenlegi állapotáról. A kapott $\mathsf{Szóbeszédek}_j$ példányból átmásol minden szóbeszédet a $\mathsf{Szóbeszédek}_i$ tömbbe, abban az esetben, ha az még nincs ott. Beállítja az $\mathsf{Aktív}_i[k]$-t a *meghibásodott* értékre, amennyiben ez az értéke az $\mathsf{Aktív}_j[k]$-nak. Beállítja a $\mathsf{Függő}_i[k]$-t a *kész* értékre, amennyiben ez az értéke a $\mathsf{Függő}_j[k]$-nak. Beállítja a $\mathsf{Függő}_i[j]$-t a *kész* értékre, ha a P_j egy terjesztő és a kapott üzenet egy tartományüzenet. Ha a P_i maga egy terjesztő, akkor $\mathsf{Függő}_i[j]$-t *kész* értékre állítja rögtön azután, miután elküldött P_j-nek egy tartományüzenetet. Hogyha a P_i processzor üzenetet vár a P_j processzortól – például egy gráfüzenetet a kommunikációs gráfban lévő szomszédjától, vagy egy válaszüzenetet –, de nem érkezik üzenet P_j-től, akkor P_i tudja, hogy a P_j processzor meghibásodott, és ekkor azonnal beállítja az $\mathsf{Aktív}_i[j]$ értékét *meghibásodottra*.

Záró-fázis

Kód a P_i, $0 \leq i \leq n-1$ processzorok esetén
1 **receive** üzenetek

11 **helyi számítások** elvégzése
12 helyi tömbök frissítése
13 **if** P_i egy gyűjtő, ami az összes processzorról hallott már
14 **then** P_i terjesztővé válik
15 a célprocesszorok halmazának meghatározása:
for minden P_j processzorra

16 **if** P_i egy gyűjtő és még nem hallott P_j-ről
17 **then** küldjön egy érdeklődő üzenetet P_j-nek
18 **if** P_i egy terjesztő és P_j egy processzor,
amely P_i által értesítendő
19 **then** küldjön egy értesítő üzenetet P_j-nek
20 **if** egy érdeklődő üzenet érkezett P_j-től e fázis fogadó lépésében
21 **then** küldjön egy válasz üzenetet P_j-nek

30 **send** érdeklődő/értesítő/válasz üzenet küldése
a célhalmazra vonatkozóan

Helyesség

A záró fázis garantálja a helyességet, amint azt a következő állítás mutatja.

19.29. lemma. *Az* Általános-pletyka *algoritmus helyes minden* $G(n, f)$ *kommunikációs gráfra és* Π *ütemezéshalmazra.*

Bizonyítás. Az integritás és a kettősségmentesség tulajdonság közvetlenül következik a kódból és a szinkron üzenetküldő rendszerben a többes üzenetküldés szolgáltatásból. Már csak azt kell bizonyítani, hogy minden egyes processzor hallott minden processzorról. Tekintsük az első záró fázisokat közvetlenül megelőző lépést. Ha a P_i processzor nem hallott még valamely más P_j processzorokról, akkor az első záró fázisban az küld P_j-nek egy utolsó remény üzenetet. Erre válasz érkezik a második záró fázisban, hacsak a P_j processzor már meg nem állt. Mindegyik esetben a P_i processzor a harmadik záró fázisban már vagy ismeri a P_j bemenő szóbeszédét, vagy megtudja, hogy a P_j meghibásodott. A negyedik záró fázik gondoskodik annak lehetőségéről, hogy ezeket a P_i által elküldött értesítő üzeneteket megkapják azok a processzorok, amelyeknek a P_i ezeket az üzeneteket elküldte. ∎

A $G(n, f)$ kommunikációs gráf, a Π ütemezéshalmaz és a τ megállási küszöb megválasztása hatással van az Általános-pletyka algoritmus speciális alkalmazásának futási idejére és üzenetszámára. Először tekintsük azt az esetet, amikor $G(n, f)$ egy olyan kommunikációs gráf, amely kielégíti a 19.27. definíció $\mathcal{R}(n, f)$ tulajdonságát, Π tartalmaz n véletlen permutációt, és $\tau = c \lg n$ elegendően nagy pozitív c konstans esetén. A 19.28. tételt alkalmazva a következő eredményt kapjuk.

19.30. tétel. *Minden* n *és* $f \leq c \cdot n$ *esetén bizonyos* $0 \leq c < 1$ *konstanshoz létezik olyan* $G(n, f)$ *gráf, hogy az általános pletyka séma implementációja a* $G(n, f)$*-fel mint kommunikációs gráffal és véletlenszerű permutációk egy* Π

halmazával $O(\lg^2 n)$ várható futási idő alatt és $O(n \lg^2 n)$ várható üzenetszámmal valósítja meg a szóbeszéd gyűjtését, ha legfeljebb f a megállások száma.

Tekintsük az ÁÁLTALÁNOS-PLETYKA algoritmus kis módosítását: a normál fázisban minden P_i processzor küld egy érdeklődő üzenetet az első Δ (egy helyett) processzornak, megfelelve a π_i permutációnak, ahol Δ az alkalmazott $G(n, f)$ kommunikációs gráf egy maximális foka. Figyeljük meg, hogy ez nincs hatással az üzenetszám nagyságrendjére, mivel az érdeklődő üzeneteken kívül minden P_i processzor minden egyes normál fázisban küld Δ gráfüzenetet.

19.31. tétel. *Minden n-hez létezik $f \leq n - 1$ és $\tau = O(\lg n)$ paraméter és létezik egy $G(n, f)$ gráf úgy, hogy a módosított* ÁLTALÁNOS-PLETYKA *algoritmus megvalósítása a $G(n, f)$-fel mint kommunikációs gráffal és véletlenszerű permutációk egy Π halmazával várt $O(\lg^2 n)$ időn belül és várt $O(n^{1.838})$ üzenetszámmal megvalósítja a pletykát, legyen a megállások száma bármennyi.*

Mivel a fenti tétel Π halmaza a számítás előtt lett kiválasztva, a következő determinisztikus létezési eredményhez jutunk.

19.32. tétel. *Minden n-hez létezik $f \leq n - 1$ és $\tau = O(\lg n)$ paraméter és létezik egy $G(n, f)$ gráf és ütemezések egy Π halmaza, hogy a módosított* ÁLTALÁNOS-PLETYKA *algoritmus megvalósítása a $G(n, f)$-fel mint kommunikációs gráffal és a Π ütemezésekkel $O(\lg^2 n)$ időn belül és $O(n^{1.838})$ üzenetszámmal valósítja meg a pletyka gyűjtését, legyen a megállások száma bármennyi.*

Gyakorlatok

19.7-1. Mutassuk meg, hogy a véletlenszerű $G(n, f)$ gráf – amelyben minden csúcs egymástól függetlenül véletlenszerűen választ ki $\frac{n}{n-f} \lg n$, önmagától más processzorokhoz vezető, élt – kielégíti a 19.27. definícióban szereplő $\mathcal{R}(n, f)$ tulajdonságot, és amelynek foka legalább $1 - O(1/n)$ valószínűséggel $O(\frac{n}{n-f} \lg n)$.

19.7-2. A vezetőválasztási probléma a következő: minden meghibásodásmentes processzornak választania kell ugyanabban a szinkron lépésben egy meghibásodásmentes processzort. Mutassuk meg, hogy a vezetőválasztás nem oldható meg gyorsabban, mint a pletyka probléma processzormegállással terhelt szinkron üzenetküldő rendszerekben.

19.8. Kölcsönös kizárás közös memóriában

Most bemutatjuk az osztott rendszerek leírására szolgáló második fő modellt, a ***közös memória*** modellt. Az ebben a modellben lévő algoritmus-

problémák bemutatásához a kölcsönös kizárás problematikájának megoldásait tárgyaljuk.

19.8.1. Közös memóriájú rendszerek

Feltesszük, hogy a rendszer n processzort $(P_0, \ldots, P_{n-1})$, valamint m regisztert $(R_0, \ldots, R_{m-1})$ tartalmaz. Minden processzor állapotgépként van modellezve. Minden regiszternek van egy típusa, amely meghatározza

1. az általa tárolható értékeket,

2. a rajta végrehajtható műveleteket,

3. a műveletek által visszaküldött értéket (ha van), valamint

4. a regiszternek az egyes műveletek által eredményezett új értékét.

Minden regiszternek lehet kezdőértéke.

Egy egész értékű olvasás/írás regiszter például felvehet bármilyen egész értéket, és rendelkezik az *olvasás*(R, v) és az *írás*(R, v) műveletével. Az olvasás művelete az utolsó, az olvasást megelőző írás v értékét adja vissza, változatlanul hagyva R-et. Az *írás*(R, v) műveletnek van egy v egész paramétere, nem ad vissza értéket, és R értékét v-re cseréli. Konfigurációnak nevezünk egy $C = (q_0, \ldots, q_{n-1}, r_0, \ldots, r_{m-1})$ vektort, ahol a Q_i a P_i processzor állapota, r_j pedig az R_j regiszter egy értéke. Az *események* a számítás processzoroknál történő azon lépései, amelyeknél a következők történnek automatikusan (láthatatlanul):

1. P_i választ egy osztott változót a P_i jelenlegi állapotán alapulva egy adott művelet elvégzéséhez,

2. a megadott művelet az osztott változón kerül végrehajtásra,

3. P_i állapota változik a saját átmenetén alapulóan, a saját jelenlegi állapota és a végrehajtott közös memóriájú művelet visszaküldött értéke alapján.

A konfigurációk és a kezdeti értékkel kezdődő események egy véges sorozatát ***végrehajtási sorozatnak*** hívjuk. Aszinkron közös memóriájú rendszerben egy végtelen végrehajtási sorozat elfogadható, ha végtelen számú számítási lépéssel rendelkezik.

19.8.2. A kölcsönös kizárás problémája

Ebben a problémában a processzorok egy csoportjának olyan osztott erőforrást kell elérnie, amelyet egyidejűleg legfeljebb egy processzor használhat. A megoldásnak a következő tulajdonságokkal kell rendelkeznie.

(1) ***Kölcsönös kizárás:*** Minden processzornak végre kell hajtania egy ***kritikus szakasznak*** nevezett kódszegmenst úgy, hogy bármely adott időpillanatban legfeljebb egy processzor hajthatja végre azt (azaz van a kritikus szakaszban).

(2) ***Holtpontmentesség:*** Ha egy vagy több processzor megkísérel belépni a kritikus szakaszba, valamikor végül egy sikerrel jár, feltéve, hogy egy processzor sem tartózkodik a kritikus szakaszban örökké. Ez a két tulajdonság semmilyen különleges biztosítékot nem nyújt egyik processzor számára sem.

(3) ***Kiéheztetésmentesség:*** A kritikus szakaszba belépni kívánó processzor valamikor végül sikerrel jár, feltéve, hogy egy processzor sem tartózkodik a kritikus szakaszban örökké. Ennek a problémának az eredeti megoldásai olyan speciális szinkronizációs eszközökre támaszkodnak, mint a szemafor vagy a monitor. Néhány olyan *osztott megoldást* mutatunk be, amelyek csak közönséges osztott változókat használnak.

Feltesszük, hogy egy processzor programja az alábbi szakaszokra bomlik:

- **Belépő/Próbálkozó:** a kritikus szakaszba való belépés előkészítéséhez végrehajtott kód.
- **Kritikus:** az egyidejű végrehajtástól megvédendő kód.
- **Kilépő:** a kritikus szakasz elhagyásakor végrehajtott kód.
- **Fennmaradó:** a kód többi része.

Egy processzor a következő sorrenden megy végig ciklikusan ezeken a szakaszokon: fennmaradó, belépő, kritikus és kilépő. A kritikus szakaszba belépni kívánó processzor először a belépő kódot hajtja végre. Ezt követően, amennyiben sikeres volt, belép a kritikus szakaszba. A processzor a kilépő szakasz futtatásával és a fennmaradó szakaszhoz való visszatéréssel feloldja a kritikus szakaszt. Feltesszük, hogy a processzor akárhányszor végrehajthatja az átmenetet a fennmaradó szakasztól a belépő szakaszig. Ezenkívül a belépő és a kilépő szakaszban elérhető osztott és lokális változók egyike sem érhető el sem a kritikus, sem a fennmaradó szakaszban. Végül is nincs olyan processzor, ami örökké a kritikus szakaszban tartózkodik. Egy közös memóriájú rendszerre vonatkozó algoritmus holtpont nélkül (vagy kizárás nélkül) megoldja a kölcsönös kizárás problémáját, ha teljesülnek a következők:

- **Kölcsönös kizárás:** Minden végrehajtási sorozat minden konfigurációja esetén legfeljebb egy processzor van a kritikus szakaszban.

- **Nincs holtpont:** Minden elfogadható végrehajtásnál, ha egy konfigurációban valamely processzor a belépő szakaszban van, akkor van egy későbbi konfiguráció, amely esetén *valamelyik* processzor a kritikus szakaszban van.
- **Nincs kiéheztetés:** Minden elfogadható végrehajtásnál, ha egy konfigurációban valamely processzor a belépő szakaszban van, akkor van egy későbbi konfiguráció, amely esetén *ugyanaz* a processzor van a kritikus szakaszban.

A kölcsönös kizárás összefüggésében egy végrehajtás ***elfogadható,*** ha minden P_i processzorra P_i vagy végtelen számú lépést tesz meg, vagy P_i a fennmaradó szakaszban fejeződik be. Sőt, nincs processzor a kilépő szakaszba örökre beragadva (akadálymentes kilépési feltétel).

19.8.3. Kölcsönös kizárás hatékony primitívek felhasználásával

Egyetlen bit elegendő a holtpont nélküli kölcsönös kizárás garantálásához egy hatékony tesztelés&beállítás regiszter használatával. A tesztelés&beállítás V változó egy bináris változó, amely két elemi műveletet, a tesztelés&beállítás és a visszaállítás műveletet támogatja, és amely a következő módon van definiálva:

tesztelés&beállítás(V : bináris változó) bináris érték visszaadása:
    $temp \leftarrow V$
    $V \leftarrow 1$
    visszatérés ($temp$)
visszaállítás(V : memóriacím):
    $V \leftarrow 0$

A tesztelés&beállítás művelet automatikusan olvassa és frissíti a változót. A visszaállítás művelete egész egyszerűen egy írás. Van egy olyan egyszerű holtpontmentes kölcsönös kizárás algoritmus, amely egy tesztelés&beállítás regisztert használ.

KÖLCSÖNÖS-KIZÁRÁS-TESZTELÉS&BEÁLLÍTÁS-REGISZTERREL

Kezdetben V értéke 0

⟨Belépés⟩:
1 várakozás, amíg tesztelés&beállítás$(V) = 0$
⟨Kritikus szakasz⟩
⟨Kilépés⟩:
2 visszaállítás(V)
⟨Fennmaradó⟩

Tegyük fel, hogy a V kezdőértéke 0. A belépő szakaszban a p_i processzor ismételten teszteli V-t mindaddig, amíg az 0-t nem ad vissza. Az utolsó ilyen teszt V-hez 1-et rendel, ami azt eredményezi, hogy minden más processzor által végzett teszt 1 értéket ad vissza – eltiltva ezáltal a többi processzort a kritikus szakaszba való lépéstől. A kilépő szakaszban P_i visszaállítja V értékét 0-ra; egy másik, a belépő szakaszban várakozó processzor most már beléphet a kritikus szakaszba.

19.33. tétel. *A tesztelő&beállító regisztert alkalmazó algoritmus holtpontmentes kölcsönös kizárást biztosít.*

19.8.4. Olvasás/írás regisztereket alkalmazó kölcsönös kizárás

Ha nem áll rendelkezésre egy olyan hatékony primitív, mint például a tesztelés&beállítás, akkor a kölcsönös kizárást az olvasás/írás műveletekkel kell megvalósítani.

A Pékség algoritmus

Lamport kölcsönös kizárásra vonatkozó Pékség (angolul Bakery) algoritmusa csak osztott olvasás/írás regisztereket használ fel. Az algoritmus garantálja a kölcsönös kizárást, és nincs kizárás $O(n)$ regisztert használó n processzor esetén (de a regisztereknek szüksége lehet olyan egész értékek tárolására, amelyekre előre nem adható meg felső korlát).

A kritikus szakaszba belépni kívánó processzorok úgy viselkednek, mint a vásárlók a pékségnél. Mindegyikük kap egy számot, és a legkisebb számot kezében tartó lesz a következő „kiszolgált” vásárló. A sorban nem álló regiszterek a 0 számot kapják, amit nem veszünk legkisebb számként számításba.

Az algoritmus a következő osztott adatszerkezeteket használja: a *Szám*$[0\,..\,n-1]$ egy n egész alkotta tömb, melynek i-edik tárolt bejegyzése a P_i processzor jelenlegi száma. A *Választás*$[0\,..\,n-1]$ n logikai értéknek egy olyan tömbje, amelyben a *Választás*$[i]$ IGAZ, amikor a P_i számára a szám megkapása éppen folyamatban van. Minden, a kritikus szakaszba belépni akaró P_i processzor próbál olyan számot választani, ami nagyobb minden egyéb pro-

cesszorénál, és beírja azt a *Szám*[*i*]-be. Ennek megtételéhez a processzorok olvassák a *Szám* tömböt és saját értékükként az olvasott legnagyobb számnál eggyel nagyobb számot választják. Mivel azonban egyidőben számos processzor olvashatja a tömböt, a szimmetria megtörik az i-edik jegyként a (*Szám*[*i*], i) választásával. Párok lexikografikus rendezését felhasználva egy rendezés van a jegyek sorrendbe rakására meghatározva. A P_i jegyének kiválasztása után addig vár, amíg az a legkisebb nem lesz: Minden más P_j esetén a P_i addig vár, amíg a P_j be nem fejezi a számválasztást, és ezután összehasonlítja a jegyeiket. Ha a P_j jegye kisebb, P_i addig vár, amíg a P_j végre nem hajtja a kritikus szakaszt, és ki nem lép abból.

PÉKSÉG

Kód a P_i, $0 \leq i \leq n-1$ processzorok esetén.
Kezdetben *Szám*[*i*]= 0 és
Választás[*i*] = HAMIS, $0 \leq i \leq n-1$ esetén

⟨Belépés⟩:
1 *Választás*[*i*] = IGAZ
2 *Szám*[*i*] = max(*Szám*[0],...,*Szám*[$n-1$]) + 1
3 *Választás*[*i*] = HAMIS
4 **for** $j = 1$ **to** n ($\neq i$) **do**
5 **wait until** *Választás*[j] = HAMIS
6 **wait until** *Szám*[j] = 0 vagy (*Szám*[j], j) > (*Szám*[*i*], i) nem lesz
⟨Kritikus szakasz⟩
⟨Kilépés⟩:
7 *Szám*[*i*] = 0
⟨Fennmaradó⟩

A következő tételek bizonyítását meghagyjuk gyakorlatnak.

19.34. tétel. *A* PÉKSÉG *algoritmus garantálja a kölcsönös kizárást.*

19.35. tétel. *A* PÉKSÉG *algoritmus garantálja a kiéheztetésmentességet.*

Egy korlátos kölcsönös kizárás algoritmus n processzorra

A PÉKSÉG algoritmus tetszőlegesen nagy értékek használatát kívánja meg. A következőkben bemutatunk egy olyan algoritmust, amely megszünteti ezt a követelményt. Ebben a kétprocesszoros algoritmusban a processzorok párosával, irányított fa elrendezésben versenyeznek. Minden párosával folytatott verseny egy teljes bináris fára van rendezve. Minden processzor a fa egy adott

leveléhez van hozzárendelve. Egy adott csúcs győztese minden szinten továbbhaladhat a következő magasabb szintre, ahol meg fog küzdeni az ezen csúcs egy másik gyerekéről feljövő győztessel (amennyiben létezik egy ilyen győztes). Annak a processzornak lesz lehetősége belépni a kritikus szakaszba, amelyik végül a gyökérben megnyeri a versenyt.

Legyen $k = \lceil \lg n \rceil - 1$. Tekintsünk egy 2^k levéllel és összesen $2^{k+1} - 1$ csúccsal rendelkező teljes bináris fát. A fa csúcsai indukciósan a következő módon sorszámozottak. A gyökér sorszáma 1; az m sorszámú csúcs bal oldali gyerekének sorszáma $2m$, míg a jobboldali gyerekének sorszáma $2m+1$. A fa leveleinek számozása tehát: 2^k, $2^k + 1$,..., $2^{k+1} - 1$.

Minden egyes m csúcshoz három bináris osztott változó tartozik: az $Igény^m[0]$, $Igény^m[1]$, és $Elsőbbség^m$. Minden változónak van egy 0 kezdőértéke. Az algoritmus rekurzív. Az algoritmus kódja egy *Csúcspont*$(m, oldal)$ eljárásból áll, ami akkor hajtódik végre, amikor a processzor elérte az m csúcsot, miközben az *oldal* processzor szerepét tölti be. Minden csúcsnak van egy kritikus szakasza. Ebbe beleszámít a szülő csúcstól a gyökérig vezető úton az összes csúcsnál lévő belépő szakasz, az eredeti kritikus szakasz, valamint a kilépő kód a gyökértől a szülő csúcsig vezető út minden csúcsán. Kezdésként a P_i processzor végrehajtja a $(2^k + \lfloor i/2 \rfloor, i \mod 2)$ csúcs kódját.

Irányított fa

```
    procedure Csúcs(m, oldal[0..1])
1   Igény^m[oldal] ← 0
2   wait until (Igény^m[1 − oldal] = 0 vagy Elsőbbség^m = oldal)
3   Igény^m[oldal] = 1
4   if Elsőbbség^m = 1−oldal
5      if Igény^m[1 − oldal] = 1)
                                  menjen az 1 sorra
6   else várjon addig, amíg Igény^m[1 − oldal] = 0 nem lesz
7   if v = 1
8            ⟨Kritikus szakasz⟩
9            else Csúcs(⌊m/2⌋, m mod 2)
10  Elsőbbség^m = 1 − oldal
11  Igény^m[oldal] = 0
12  eljárás vége
```

Az algoritmus korlátos értékeket használ, és amint azt a következő tételek megmutatják, kielégíti a kölcsönös kizárás, kizárásmentességi tulajdonságokat.

19.36. tétel. *Az* Irányított-fa *algoritmus garantálja a kölcsönös kizárást.*

Bizonyítás. Tekintsünk egy végrehajtást. A fa leveléhez legközelebb eső csúcstól indulunk ki. Egy processzor akkor lép be ennek a csúcsnak a kritikus szakaszába, ha eléri a 9. sort (elmegy a következő csúcsig). Tegyük fel, hogy annál az m csúcsnál vagyunk, amely kapcsolódik azokhoz a levelekhez, ahonnan P_i és P_j indul. Tegyük fel, hogy ez a két processzor valamely pontnál kritikus szakaszban van. A kódból következik, hogy ennél a pontnál ekkor $Igény^m[0] = Igény^m[1] = 1$. Az általánosság megszorítása nélkül feltehetjük, hogy P_i kritikus szakaszba történő belépése előtti utolsó írása $Igény^m[0]$-be követi a P_j kritikus szakaszba történő belépése előtti $Igény^m[1]$-be történő utolsó írását. Figyeljük meg, hogy P_i be tud lépni az (m-hez tartozó) kritikus szakaszba mind az 5., mind a 6. soron keresztül. P_i mindkét esetben $Igény^m[1] = 0$ értéket olvas. Annak ellenére, hogy P_i $Igény^m[1]$-et olvas, követi P_j $Igény^m[0]$-ba végzett írását. Tehát az, hogy P_i-nek $Igény^m[1]$ olvasásával 1-et kell visszaadnia, egy ellentmondás.

Az állítás a fa leveleire alkalmazott indukcióból következik. ∎

19.37. tétel. *Az* Irányított-fa *algoritmus garantálja a kiéheztetésmentességet.*

Bizonyítás. Tekintsünk egy elfogadható végrehajtást. Tegyük fel, hogy valamely P_i processzor megállt. Egy afdott időponttól kezdve P_i örökre a belépő szakaszban marad. Most azonban megmutatjuk, hogy P_i nem ragad be örökre az m csúcs belépő szakaszába. Az állítást indukcióval bizonyítjuk.

1. eset: Tételezzük fel, hogy P_j az $Elsőbbség^m$-nek a 0-ra állításával hajtja végre a 10. sort. Ezek után az $Elsőbbség^m$ már mindig 0 lesz. Tehát P_i áthalad a 2. soron, és az 5. sorra lép. Így tehát P_i-nek a 6. sorban várakoznia kell, arra kell várni, hogy $Igény^m[1]$ 0 legyen, ami sohasem fog bekövetkezni. P_j így mindig a 3. és a 11. sor közötti sorokat hajtja végre. Mivel azonban P_j nem marad örökké a kritikus szakaszban, ez azt jelentheti, hogy P_j örökre a belépő szakaszba ragad, ami viszont lehetetlen, mivel P_j végrehajtja az 5. sort, és visszaállítja $Igény^m[1]$-et 0-ra.

2. eset: Tételezzük fel, hogy valamely későbbi pontnál P_j sohasem hajtja végre a 10. sort. Így P_j-nek várakoznia kell a 6. sorban vagy a fennmaradó szakaszban. Ha ez a belépő szakaszban van, akkor P_j áthalad a 2. sorban lévő teszten ($Elsőbbség^m$ értéke 1). Így tehát P_i nem éri el a 6. sort. Ekkor P_i $Igny^m[0] = 0$ mellett a 2. sorban várakozik. Így P_j áthalad a 6. sorban lévő teszten. Tehát P_j nem maradhat örökre a belépő szakaszban. Ha P_j örökre a

fennmaradó szakaszban marad, a továbbiakban $Igény^m[1]$ egyenlő lesz 0-val. Tehát P_i nem ragadhat be az 2., 5. és 6. sorban, ami ellentmondás.

Az állítás a fa levelein vett indukció alapján következik. ■

Az írás/olvasás regiszterek számára adott alsó korlát

Eddig a bemutatott holtpontmentes kölcsönös kizárás algoritmusok megkövetelték legalább n osztott változó használatát, ahol n a processzorok száma. Mivel lehetséges volt olyan algoritmus kifejlesztése, amely csak korlátos értékeket használt, felmerül a kérdés, hogy van-e mód a használt osztott változók számának csökkentésére.

Burns és Lynch bebizonyították, hogy bármely csak osztott írás/olvasás regisztereket használó holtpontmentes kölcsönös kizárás algoritmusnak szüksége van legalább n osztott változóra, méretüktől függetlenül. Tételük bizonyítása lehetővé teszi azt, hogy a változók többes-író változók legyenek. Ez azt jelenti, hogy minden processzor írhat minden változóba. Figyeljük meg, hogy ha a változók egyszeres írók, a tétel nyilvánvaló, mivel minden processzornak írnia kell valamit egy (különálló) változóba, mielőtt a kritikus szakaszba lép. Különben a processzor anélkül léphetne be a kritikus szakaszba, hogy bármely másik processzor tudna erről, ami megengedné azt, hogy egyidejűleg tetszőleges másik processzor is beléphessen a kritikus szakaszba, ami ellentmondana a kölcsönös kizárás tulajdonságnak.

A bizonyítás bevezet egy új bizonyítástechnikát, az ***argumentum-lefedést.*** Adott egy tetszőleges A holtpontmentes kölcsönös kizárás algoritmus, ez azt mutatja meg, hogy van A-nak valamilyen olyan elérhető konfigurációja, amelyben az n processzor mindegyike azon van, hogy *egyedül* írjon egy osztott változóba. Ezt az osztott változók egy ***lefedésének*** nevezzük. Egy ilyen konfiguráció létezése megmutatható indukció alkalmazásával, és ez kihasználja azt a tényt, hogy a kritikus szakaszba történő belépés előtt minden processzornak írnia kell legalább egy osztott változóba. A bizonyítás megvalósítja az összes osztott változó egy lefedését. A processzor ezután belép a kritikus szakaszba. A lefedést követően az írások azonnal közzé vannak téve, ebből következően egyik processzor sem érzékeli a kritikus szakaszban lévő processzort. Így most egy másik processzor is beléphet egyidejűen a kritikus szakaszba, ami ellentmondás.

19.38. tétel. *Bármely írás/olvasás regisztert alkalmazó holtpontmentes kölcsönös kizárás algoritmusnak használnia kell legalább n osztott változót.*

19.8.5. Lamport gyors kölcsönös kizárás algoritmusa

Minden eddig bemutatott kölcsönös kizárás algoritmusban a processzorok által a kritikus szakaszba történő belépés előtt megtett lépések száma függött n-

től, a processzorok számától a verseny (amikor több processzor egyszerre akar belépni a kritikus szakaszba) hiánya esetén is, amikor csak egyetlen processzor van a belépő szakaszban. A legtöbb valós rendszerben azonban a várható küzdelmek száma rendszerint n-nél jóval kisebb.

Egy kölcsönös kizárás algoritmust ***gyorsnak*** nevezünk, ha egy processzor egy konstans számon belüli lépésszámmal lép be a kritikus szakaszba, amikor ez az egyetlen, a kritikus szakaszba belépni próbáló processzor. Figyeljük meg, hogy egy gyors algoritmus többes-író és többes-olvasó változók használatát igényli. Amennyiben csak egyszeres író változókat használna, egy processzornak legalább n számú változót kellene olvasnia.

Az alábbi GYORS-KÖLCSÖNÖS-KIZÁRÁS algoritmust Lamport javasolta.

GYORS-KÖLCSÖNÖS-KIZÁRÁS

Kód a P_i processzorra, $0 \leq i \leq n-1$.
Kezdetben a *Gyors-lezárás* és a *Lassú-lezárás* mindegyike 0,
és *Igény*[i] HAMIS minden $i, (0 \leq i \leq n-1)$ esetén

⟨ Belépés ⟩:
1 *Igény*[i] = IGAZ
2 *Gyors-lezárás* = i
3 **if** *Lassú-lezárás* $\neq 0$ **then**
4 *Igény*[i] = HAMIS
5 várjon addig, amíg *Lassú-lezárás* = 0 nem lesz
6 menj 1-re
7 *Lassú-lezárás* = i
8 **if** *Gyors-lezárás* $\neq i$
9 *Igény*[i] = HAMIS
10 minden j esetén, várjon addig, amíg *Igény*[j] = HAMIS nem lesz
11 **if** *Lassú-lezárás* $\neq i$
12 **wait until** *Lassú-lezárás* = 0
13 menj 1

⟨Kritikus szakasz⟩
⟨Kilépés⟩:
14 *Lassú lezárás* = 0
15 *Igény*[i] = HAMIS
⟨Fennmaradó⟩

Lamport algoritmusa két mechanizmus kifogástalan kombinációja, ezek közül az egyik a gyors belépés engedélyezése, amikor küzdelem nem érzékel-

hető, a másik pedig a holtpontmentesség biztosítása küzdelem esetén. A *Gyors-lezárás* és a *Lassú-lezárás* változókat használja a versenynélküli esetekben a hozzáférés vezérlésére. Ezen kívül minden P_i processzorhoz tartozik egy *Igény*$[i]$ logikai változó, amely értéke igaz, ha P_i érdekelt a kritikus szakaszba történő belépésben, egyébként hamis. A processzor akkor tud belépni a kritikus szakaszban, ha vagy *Gyors-lezárás* $= i$-t talál – ebben az esetben *gyors útvonalon* lép be a kritikus szakaszba –, vagy *Lassú-lezárás* $= i$-t talál, amely esetben *lassú útvonal* mentén lép be a kritikus szakaszba.

Tekintsük azt az esetet, amikor nincs processzor sem a kritikus, sem a belépő szakaszban. Ebben az esetben a *Lassú-lezárás* értéke 0, és minden *Igény* bejegyzés 0. Ha ekkor P_i belép a belépő szakaszba, az *Igény*$[i]$-t 1-re, a *Gyors-lezárást* pedig i-re állítja. Ekkor ellenőrzi a *Lassú-lezárást,* ami 0. Ekkor újra ellenőrzi *Gyors-lezárást,* és mivel nincs másik processzor a belépő szakaszban, beolvassa i-t és belép a kritikus szakaszba három írással és két olvasással rendelkező gyors útvonal mentén.

Ha *Gyors-lezárás* $\neq i$, akkor P_i vár addig, amíg az összes *Igény* jelző vissza nem állítódik. Miután valamely processzor végrehajtja a 10. sor for ciklusát, a *Lassú-lezárás* változatlan marad mindaddig, amíg valamely kritikus szakaszt elhagyó processzor vissza nem állítja azt. Tehát legfeljebb egy P_j processzor találja azt, hogy *Lassú lezárás* $= j$, és ez a processzor fog lassú útvonal mentén belépni a kritikus szakaszba. Figyeljük meg, hogy a GYORS-KÖLCSÖNÖS KIZÁRÁS algoritmus nem garantálja a kiéheztetésmentességet.

19.39. tétel. *A* GYORS-KÖLCSÖNÖS-KIZÁRÁS *algoritmus holtpontmentes kölcsönös kizárást garantál.*

Gyakorlatok

19.8-1. Egy algoritmus megoldja a ***2-kölcsönös kizárás*** problémáját, ha bármely időpillanatban legfeljebb két processzor van a kritikus szakaszban. Mutassunk be egy algoritmust a 2-kölcsönös kizárás megoldására a tesztelés&beállítás regiszterek használatával.

19.8-2. Bizonyítsuk be, hogy a PÉKSÉG algoritmus kielégíti a kölcsönös kizárás tulajdonságát.

19.8-3. Bizonyítsuk be, hogy a PÉKSÉG algoritmus kiéheztetésmentességét nyújt.

19.8-4. Különítsük el a két processzorra vonatkozó, kizárásmentes korlátos kölcsönös kizárás algoritmust és az irányított fa algoritmust. Mutassuk meg, hogy az előbbi algoritmus rendelkezik a kölcsönös kizárás tulajdonságával.

19.8-5. Bizonyítsuk be, hogy a GYORS-KÖLCSÖNÖS-KIZÁRÁS algoritmus rendelkezik a kölcsönös kizárás tulajdonsággal.

19.8-6. Bizonyítsuk be, hogy a GYORS-KÖLCSÖNÖS-KIZÁRÁS algoritmus rendelkezik a holtpontmentesség tulajdonsággal.
19.8-7. Mutassuk meg, hogy a GYORS-KÖLCSÖNÖS-KIZÁRÁS algoritmus nem elégíti ki kiéheztetésmentesség tulajdonságát, azaz alkossunk meg egy olyan végrehajtást, amelyben egy processzor ki van zárva a kritikus szakaszból.

Feladatok

19-1 Az* ELÁRASZTÁS *algoritmus üzeneteinek a száma
Adott egy G gráf n csúccsal és e éllel. Bizonyítsuk be, hogy bármely végrehajtási sorozat esetén az ELÁRASZTÁS algoritmus $O(e)$ üzenetet küld. Mi az üzenetek pontos száma a csúcsok és az élek számának függvényében?

19-2 Vezetőválasztás gyűrűben
Tegyük fel, hogy az üzeneteket csak az óramutató járásával egyező irányban lehet küldeni. Tervezzünk egy aszinkron, $O(n \lg n)$ üzenetszámú algoritmust a gyűrűbeli vezető megválasztására. *Útmutatás.* A processzorok dolgozzanak fázisokban. Minden processzor *aktív módban* kezdjen egy, a processzor azonosítójának megfelelő *értékkel,* s bizonyos feltételek teljesülése mellett lépjen *továbbító* módba, ahol csak továbbítja az üzeneteket. Egy aktív processzor két másik aktív processzortól vár üzenetet, megvizsgálja az azok által küldött értékeket, és eldönti, hogy vezető lesz-e, aktív marad-e és elfogadja az egyik *értéket,* vagy továbbító módba vált. Határozzuk meg, hogy mi alapján kell a döntéseket meghozni úgy, hogy ha három vagy több aktív processzor van, akkor legalább az egyik aktív marad, illetve, hogy függetlenül attól, hogy milyen értékekkel rendelkeznek az aktív processzorok egy fázisban, legalább a processzorok fele maradjon aktív a következő fázisban.

19-3 A* GYORS-KÖLCSÖNÖS-KIZÁRÁS *algoritmus elemzése
Alkossuk meg a GYORS-KÖLCSÖNÖS-KIZÁRÁS algoritmus egy olyan végrehajtási sorozatát, amelyben két processzor van a belépő szakaszban, és mindkettő $\Omega(n)$ változót olvas a kritikus szakaszba történő belépése előtt.

19-4 Egyetértés érvényessége aszinkron rendszerekben
Mutassuk meg, hogy az érvényességi feltétel ekvivalens azzal a követelménnyel, hogy minden hibamentes processzor döntése valamely processzor bemenete legyen.

19-5 Egyetlen forrású egyetértés

Az egyetértési probléma egy változata megköveteli, hogy egy kijelölt processzszor *(a tábornok)* bemenő értékét eljutassák az összes többi processzorhoz *(a hadnagyokhoz)*. Ez a probléma ***egyetlen forrású egyetértés*** néven is ismert. A következő feltételeknek kell teljesülniük:

- **Megállás**: Minden hibamentes hadnagy valamikor végül dönt.
- **Megegyezés**: Az összes hibamentes hadnagy ugyanarra a döntésre jut.
- **Érvényesség**: Ha a tábornok hibamentes, akkor a közös eredmény a tábornok bemenete lesz.

Tehát, ha a tábornok hibamentes, akkor a hibamentes processzoroknak nem kell dönteniük a tábornok bemenetéről, de még mindig egyetértésre kell jutniuk egymással. Tekintsünk egy bizánci hibákkal rendelkező szinkron üzenetküldő rendszert. Mutassuk meg, hogyan lehet a 19.4.5. pontban leírt egyetértési probléma egy megoldását a tábornok probléma egy megoldásává át- és visszaalakítani. Mik a transzformáció üzenet- és menetszámbeli többletköltségei?

19-6 Banki tranzakciók

K épzeljük el, hogy n bank összeköttetésben áll egymással. Az i-edik bank m_i összegű pénzzel rendelkezik. A bankok nem emlékeznek a kezdőösszegre. A bankok pénzösszegeket utalnak egymás között <10> alakú üzenetekben, melyek az átutalt összeget reprezentálják. Adott időpontban az egyik bank úgy dönt, hogy megállapítja a rendszerben lévő pénz összegét. Tervezzünk egy algoritmust, ami kiszámolja az $m_1 + \cdots + m_n$ összeget úgy, hogy közben a pénzügyi tranzakciók nem állnak le.

Megjegyzések a fejezethez

Az osztott rendszerek definícióját Attiya és Welch könyvéből [9] vettük át. Az üzenetküldő rendszerek meghibásodás nélküli modelljét Attiya, Dwork, Lynch és Stockmeyer cikke [8] alapján tárgyaljuk.

A rendszer egyes processzorait – Lynch és Fischer cikkét követve – egy automatával (amely véges és végtelen is lehet) modellezzük [189].

A végrehajtási sorozat fogalma Fischer, Gries, Lamport és Owicki cikkeiből [189, 226, 227] származik.

Az *aszinkron rendszer* definícióját Awerbuch [11], valamint Peterson és Fischer [234] cikkeiből vettük át.

A Feszítőfa-üzenetszóró algoritmust Segall cikke [263] alapján ismertettük.

A szinkron és aszinkron rendszerekben is alkalmazható Bully algoritmust Hector-Garcia Molina javasolta 1982-ben [95]. Az ismertetett aszinkron

vezetőválasztási algoritmus aszimptotikusan optimalitását Burns [39] bizonyította be.

A *két tábornok problémáját* Gray elemzése [106] alapján tárgyaltuk.

Az *egyetértési problémát* először Lamport, Pease és Shostak [171, 232] vizsgálta.

Az EGYETÉRTÉS-MEGÁLLÁSI-HIBÁKNÁL algoritmus Dolev és Strong [64] cikkén alapul. A 19.14. tételhez hasonló állítás bizáci hibákra Fischer és Lynch [80] eredménye. Az idézett, megállásos hibákra vonatkozó tétel is Dolev és Strong [64] cikkéből származik.

Az egyetértési problémát bizánci hibájú processzorok esetén megoldó algoritmust Berman és Garay [24] javasolta.

Az osztott algoritmusok elméletének egyik alapvető eredménye, hogy aszinkron rendszerekben – megbízható kommunikáció mellett – az egyetértési probléma már akkor sem oldható meg, ha egyetlen processzor meghibásodhat. Ezt az eredményt először Fischer, Lynch és Paterson [81] cikke mutatta be.

Leslie Lamportnak a kölcsönös kizárásra vonatkozó PÉKSÉG algoritmusa [169] korai, klasszikus példája az olyan algoritmusnak, amely csak osztott olvasás/írás regisztereket használ fel. A PÉKSÉG algoritmus tetszőlegesen nagy értékek használatát kívánja meg. Ezt a követelményt Peterson és Fischer [234] küszöbölte ki.

Először Burns és Lynch [40] mutatta meg, hogy bármely – csak osztott írás/olvasás regisztereket használó, holtpontmentes kölcsönös kizárást biztosító – algoritmusnak használnia kell legalább n osztott változót, méretüktől függetlenül.

A fejezetben tárgyalt GYORS-KÖLCSÖNÖS-KIZÁRÁS algoritmust Leslie Lamporttól [170] származik.

A *19-3.*, *19-4.* és a *19-5.* feladatok forrása Attiya és Welch könyve [9].

Az osztott algoritmusok témakörét összefoglaló mű Gerard Tel [301] és Nancy Ann Lynch [188] könyve – utóbbi magyarul is megjelent. Osztott algoritmusok gyakorlati alkalmazását mutatja be Fokkink [87].

Több osztott algoritmust (például a kölcsönös kizárás biztosítására) tárgyal Kozma László és Varga László könyve [161]. Sok hasznos ismeretet tartalmaz konkrét rendszerekről Tanenbaum és van Steen [295] könyve, amely szintén elérhető magyarul is.

20. Párhuzamos számítások

A folyamatok szintjén párhuzamos számítások[1] fő célja, hogy a feladatokat több processzor segítségével gyorsabban oldjuk meg, mint egy processzoron. Ezek a processzorok tartozhatnak egyetlen számítógéphez vagy különböző számítógépekhez, amelyek egy hálózaton keresztül kommunikálnak. A párhuzamos végrehajthatósághoz mindkét esetben szükség van arra, hogy a feladatot egyidejűleg megoldható alfeladatokra osszuk.

Bár a párhuzamos számítások története korábban kezdődött, az első párhuzamos számítógépnek a 64 processzort tartalmazó ILLIAC IV-et szokták tekinteni, amely 1972-ben kezdett működni. A párhuzamos számítógépek a nyolcvan évek végén indultak gyors fejlődésnek, amikor is több új céget alapítottak különböző párhuzamos számítógépek gyártására. Sajnos, abban az időben nehezen indult el a szoftver fejlesztése és a kifejlesztett szoftver nem volt hordozható. Ezért a párhuzamos számítógépeket csak a tudományos és műszaki élet leginkább számításigényes területein alkalmazták, mivel a piac túl kicsi volt ahhoz, hogy a jelentős fejlesztési költségeket meg tudja fizetni. Ezért számos cég abba is hagyta a fejlesztést.

A dolog pozitív oldala volt az a felismerés, hogy olcsó párhuzamos számítógépek hozhatók létre az elterjedt személyi számítógépek vagy munkaállomások összekapcsolásával. Ahogy a hálózatok gyorsabbá váltak, ezek az úgynevezett *klaszterek* rövidesen ugyanolyan nagyságrendű sebességet értek el, mint a speciális célú gépek. Jelenleg a világ 500 legnagyobb számítógépének folyamatosan frissített listáján a gépek 42 százaléka klaszter.

A párhuzamos számítások eszköztárát gyarapítják azok a multiprocesszoros gépek is, amelyeket ugyan a web és más területek kiszolgálására terveztek, de párhuzamos számítások elvégzésére is alkalmasak. Végül a szoftver hordozhatóságával kapcsolatos problémák is megoldódtak a párhuzamos programozás széles körben elterjesztett szabványai segítségével. A két legfontosabb szabvány az MPI és OpenMP – ezeket a 20.1. alfejezetben fogjuk ismertetni.

Összegezve, jelenleg elfogadható költségű hardverbázis áll rendelkezésre. A terület azonban még nem érte el fejlődési határait, mivel a párhuzamos szoftver fejlesztése komoly nehézségeket okoz. Míg egy soros program megírásához elegendő egy algoritmust találni, azaz elemi műveleteknek az

[1] Párhuzamos számítás több szinten valósítható meg, így utasítás-szinten, szál-szinten és folyamat-szinten. Ez a fejezet elsődlegesen a folyamat szintű párhuzamos feldolgozással foglalkozik. *A lektor.*

adott feladatot megoldó sorozatát, majd az algoritmust egy programozási nyelven leírni, addig a párhuzamos számítások további kihívásokat jelentenek:

- Az elemi műveleteket olyan taszkokká kell csoportosítani, melyek párhuzamosan megoldhatók.
- A taszkokat ütemezni kell a processzorokra.
- Az adatokat az architektúrától függően el kell osztani a memóriamodulok között.
- A folyamatokat és szálakat kezelni kell, azaz elindítani, megállítani és így tovább. A kommunikációt és a szinkronizációt meg kell szervezni.

Természetesen nem elég a műveletek csoportosítását, az ütemezést és így tovább megoldani – olyan megoldásokat kell találni, amelyek gyors programokhoz vezetnek. A teljesítménymértékeket és a teljesítmény optimalizálásának módszereit a **??**. alfejezetben tárgyaljuk, ahol a fent említett feladatok megoldását is vizsgáljuk. A különböző párhuzamos architektúrák és programozási modellek – a soros modellektől eltérően – különböző algoritmusokat részesítenek előnyben.

Ennek következtében a párhuzamos algoritmusok bonyolultabbak, mint a sorosak. A párhuzamos algoritmusok vizsgálatához gyakran alkalmaznak egyszerűsített modelleket. Például a népszerű PRAM (lásd a 20.4. alfejezetet) modell nem veszi figyelembe a kommunikációs és szinkronizálási költségeket.

Hasonlítsuk össze a párhuzamos számításokat az osztott és a konkurens számításokkal. Az ***osztott számítások*** összekapcsolt processzorokat használnak és a megoldandó problémákat kisebb részfeladatokra osztják, de a felosztás céljai különbözőek lehetnek. Míg a ***párhuzamos számításoknál*** a részfeladatok *egyidejűleg* oldódnak meg, az osztott számításoknál a részfeladatok *különböző helyeken* oldódnak meg, *különböző erőforrásokat felhasználva.* Ezek a tulajdonságok átfedhetik egymást, ezért számos alkalmazás egyszerre párhuzamos és osztott, a lényeg azonban különböző. A párhuzamos számításoknál a felépítés homogén, és a főcél a számítások gyorsabb elvégzése, az osztott számításoknál a felépítés heterogén és az alkalmazások számára gyakran kedvező a különböző erőforrások összekapcsolása. A párhuzamos rendszerek általában hosszabb ideig léteznek és kiszámíthatóak, míg az osztott alkalmazások olyan elemekből állnak, melyek a futás során kapcsolódtak össze.

A ***konkurens számítások*** nem feltétlenül igényelnek több processzort, és azt a körülményt emelik ki, hogy egyidejűleg több részszámítás van folyamatban. Ez a fontos tulajdonság garantálja, hogy az eredmény a részszámítások tetszőleges sorrendje és átfedése mellett is helyes lesz. Ezért a párhuzamosság és a konkurencia viszonya ahhoz hasonlítható, hogy egyidejűleg több könyvet

olvasunk (ami a gyakorlatban valószínűleg nem oldható meg). Így a konkurens számítás vagy párhuzamos, vagy nem párhuzamos, a párhuzamos számítás viszont mindig konkurens. Egyetlen kivétel az adatok párhuzamossága, melyben egyetlen program utasításait alkalmazzuk párhuzamosan különböző adatokra. Ezt a megközelítést követi a SIMD architektúra.

A nagy sebességnek köszönhetően a párhuzamos számítások tipikus alkalmazási területei a természettudományok és a műszaki feladatok megoldása, különösen a numerikus számítások és a szimuláció. Ezeknek az alkalmazásoknak egyre nagyobb a számítási igénye, mivel a nagyobb számítási kapacitás részletesebb modellek alkalmazását és így pontosabb eredmények elérését teszi lehetővé. A párhuzamos számítások másik előnye a nagyobb memóriakapacitás, ami lehetővé teszi, hogy több adatot tároljunk az olyan memóriaszinteken, mint a gyorsítótár.

A fejezet további részének felépítése a következő. A 20.1. alfejezetben röviden áttekintjük és osztályozzuk a jelenlegi párhuzamos architektúrákat. Azután a 20.2. alfejezetben olyan alapfogalmakat vezetünk be, mint a taszk és a folyamat, majd elemezzük a hatékonysági mértékeket és a hatékonyság javítására szolgáló általános módszereket. Ezután a 20.3. alfejezet a párhuzamos programozás modelljeit ismerteti, elsősorban a népszerű MPI és OpenMP szabványokra koncentrálva. Erre a gyakorlati alapra épül a fejezet hátralévő része, amely formalizált módon tárgyalja a legegyszerűbb párhuzamos algoritmusokat és tervezési módszereket. A soros algoritmusoktól eltérően a párhuzamos algoritmusok esetén nincs általánosan elfogadott tervezési és elemzési módszer, bár számos modellt javasoltak erre a célra. A modellek mindegyike bizonyos kompromisszumot valósít meg az egymásnak ellentmondó célok – nevezetesen a valódi architektúrák pontos tükrözése és a tervezés, valamint elemzés egyszerűsége – között. A 20.4. alfejezet áttekintést ad a modellekről, a 20.5. alfejezet pedig konkrét algoritmusokat mutat be – többek között a kiválasztás, összefésülés és rendezés feladatok megoldására.

Az algoritmusok elemzése során mindvégig nagy figyelmet fordítunk az adott feladat soros és párhuzamos megoldási idejének, valamint a kétféle megoldás során végzett munkának az összehasonlítására. Az algoritmusok elemzésének korszerű felfogását követve nem elégszünk meg a vizsgált hatékonysági mérték értékének egy felső korlátjával, hanem lehetőleg pontosan jellemezzük a legrosszabb esetben várható futási idő és a végzett munka nagyságrendjét.

20.1. Párhuzamos architektúrák

A párhuzamos architektúrák egyik egyszerű és közismert osztályozását Flynn javasolta. A számítógépeket négy osztályba sorolta: SISD, SIMD, MISD, és MIMD architektúrák.

- SI *egyszeres utasításáramot* (**S**imple **I**nstruction stream) jelent, azaz egyidejűleg egyetlen utasítás hajtódhat végre.
- MI *többszörös utasításáramot* (**M**ultiple **I**nstruction stream) jelent, azaz különböző processzorok egyidejűleg különböző utasításokat is végrehajthatnak.
- SD *egyszeres adatáramot* (**S**imple **D**ata stream) jelent, azaz egyidejűleg egyetlen adattal hajtódhatnak végre műveletek.
- MD *többszörös adatáramot* (**M**ultiple **D**ata stream) jelent, azaz egyidejűleg különböző adatokkal is végezhetők műveletek.

A SISM felépítés a Neumann-elvű számítógépeket jelenti. MISD típusú számítógépeket valószínűleg sohasem építettek. A korai számítógépek SIMD felépítésűek voltak, ma a legtöbb párhuzamos számítógép MIMD felépítésű. Bár ennek a sémának kicsi az osztályozási ereje, mégis széles körben alkalmazzák.

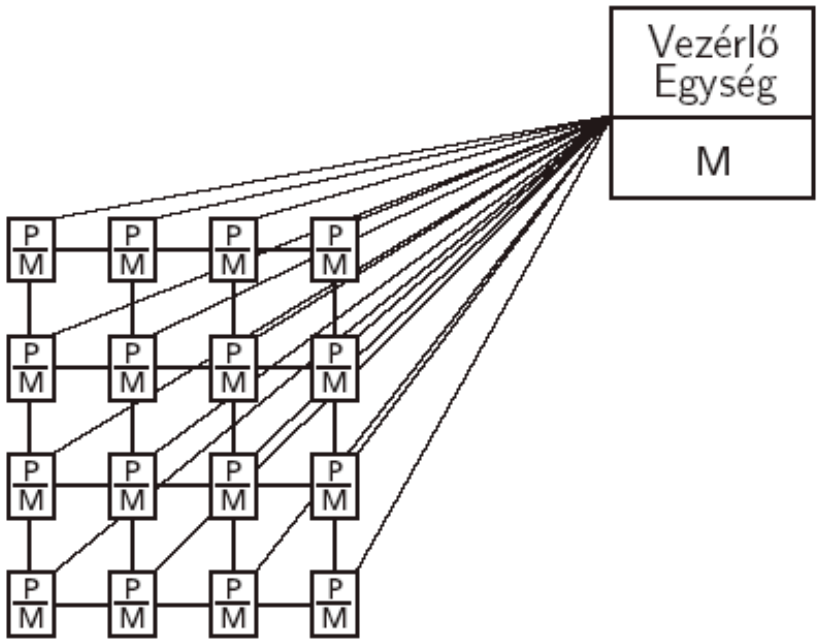
A következő osztályozás a párhuzamos gépeket a SIMD, SMP, ccNUMA, nccNUMA, NORMA, klaszterek és rács osztályokba sorolja.

SIMD architektúrák

Amint azt a 20.1. ábra mutatja, a SIMD számítógép egy nagyteljesítményű vezérlő processzorból és több, kisebb teljesítményű feldolgozó elemből áll. A feldolgozó egységeket gyakran rácsszerűen kötik össze úgy, hogy minden egység a közvetlen szomszédaival kommunikál. A vezérlő processzor – a soros gépek processzorához hasonlóan – egymás után olvassa és dekódolja az utasításokat. A soros esetben a processzor a beolvasott utasítást a saját memóriájában lévő adatokkal hajtja végre. A párhuzamos esetben a vezérlő processzor minden feldolgozó egységhez elküldi a beolvasott utasítást és azok egyidejűleg hajtják végre ezt az utasítást a saját memóriájukban tárolt adatokon. Példaként tekintsük az LD reg, 100 utasítást. Ennek hatására minden processzor betölti a 100-as memóriacím tartalmát a regiszterbe, de a 100-as cím processzoronként különböző memóriarekeszt jelent. Tehát a processzorok – a SIMD felépítésnek megfelelően – azonos utasítást hajtanak végre különböző adatokkal. Az

if *test* then *if_branch* else *else_branch*

utasítás hatására a processzorok egyidejűleg elvégzik a tesztelést, azután egy részük az *if_branch* ágon folytatja, míg mások tétlenek maradnak, és



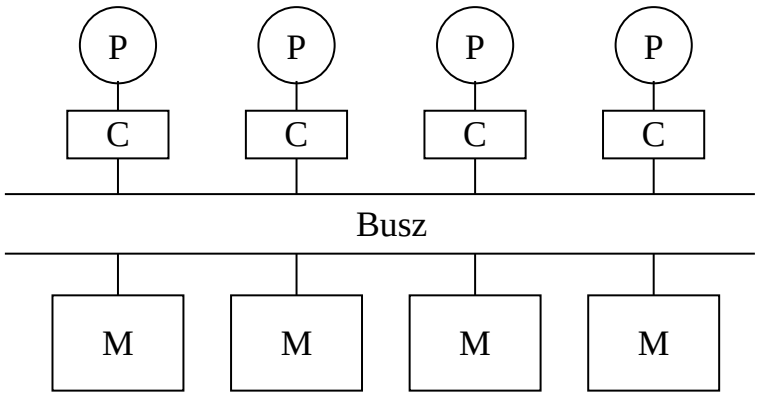
20.1. ábra. SIMD architektúra.

végül az előbbiek maradnak tétlenek és az utóbbiak végzik el az *else_branch* ágat. Emiatt a SIMD felépítésű számítógépek szabályos szerkezetű számítások elvégzésére alkalmasak. Ez a felépítés történetileg lényeges, de ma már nem alkalmazzák.

Szimmetrikus multiprocesszorok

A szimmetrikus multiprocesszorok (SMP) egy közös memóriához kapcsolódó processzorokat tartalmaznak. A hardver minden processzornak hozzáférést biztosít – a szokásos betölt/tárol műveletekkel – minden memóriarekeszhez. Ezért a programokat – beleértve az operációs rendszert is – elég egy példányban tárolni. A memória fizikailag modulokra bontható, de a hozzáférési idő azonos minden processzor-modul párra (ez indokolja a szimmetrikus jelzőt). A processzorokat egy adatátviteli vonal, egy mátrix-kapcsoló vagy egy kapcsolóhálózat köti össze a memóriával (lásd 20.2. ábra). A memóriához való hozzáférés mindegyik esetben késleltetéssel történik, ami részben a hálózati erőforrásokért folyó verseny következménye, és a processzorok számával nő.

Minden processzornak egy- vagy többszintes gyors hozzáférésű gyorsítótára is van. A fő memória és a gyorsítótár között az adatok a gyorsító adatátviteli vonalakon mozognak. Ha az egyes adatelemeket több gyorsí-



20.2. ábra. Busz-alapú SMP-architektúra.

tótárban is tároljuk, akkor koherenciaproblémák lépnek fel. Például ***rossz megosztásról*** beszélünk, ha több processzor is hozzáfér ugyanahhoz a gyorsítóvonalhoz, de annak különböző részét használják.

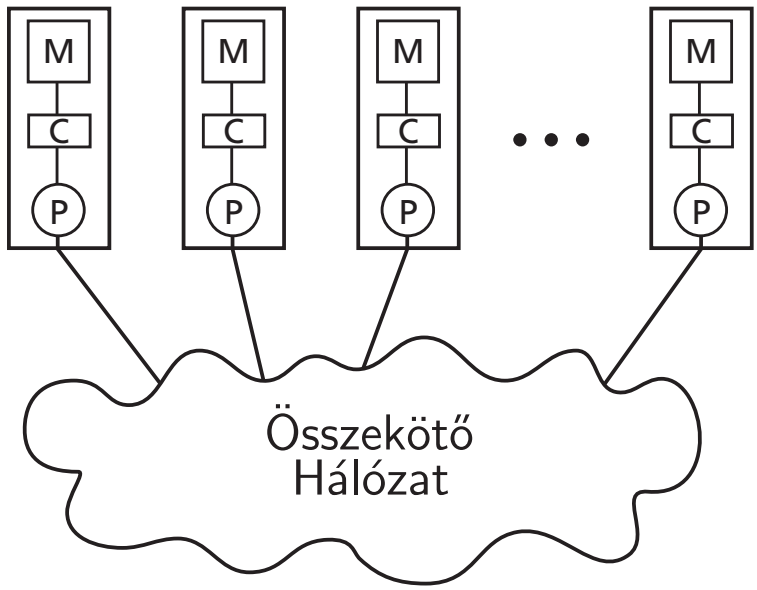
Gyorsítótáras NUMA architektúrák

A ccNUMA (gyorsítótárral koherens nem-egyenletes memóriahozzáférés) nem egységes memóriahozzáférést jelent, ellentétben az előző osztály szimmetrikus hozzáférésével. A 20.3. ábra mutatja ezeknek a számítógépeknek a felépítését.

Az ábra szerint minden processzorhoz tartozik egy *helyi memória* (gyorsítótár), amelyhez a hozzáférés gyorsabb, mint a memória további, *távoli memóriának* nevezett részéhez. A teljes memóriához szabványos betöltő/tároló műveletekkel férhetünk hozzá, ezért a programokat – az operációs rendszerek programjait is – csak egyszer kell tárolni. Az SMP-khez hasonlóan minden processzornak egy vagy többszintes gyorsítótára van; a gyorsítótár szintjeinek összhangját a hardver biztosítja.

Gyorsítótár nélküli NUMA architektúrák

Az nccNUMA architektúrák abban különböznek a ccNUMA architektúráktól, hogy a hardver csak a helyi memóriából származó adatokat tölti be a gyorsítótárba. A távoli memóriához való hozzáférés a szokásos beviteli/kiviteli műveletekkel megvalósítható, de először az operációs rendszernek kell a megfelelő lapot a helyi memóriába töltenie. Ez a különbség egyszerűsíti a hardver tervezését, és ezért az nccNUMA felépítésű gépek több processzort tartalmaznak. Hátrány viszont, hogy az operációs rendszer bonyolultabb, a távoli memóriához való hozzáférés ideje nagyobb. A 20.3. ábrán bemutatott felépítés az nccNUMA típusú számítógépekre is vonatkozik.



20.3. ábra. ccNUMA architektúra.

Távoli memóriához való gyors hozzáférés nélküli architektúrák

A NORMA (**NO R**emote **M**emory **A**ccess Architectures) architektúra abban különbözik az előző osztálytól, hogy a távoli memóriához lassú beviteli/kiviteli műveletekkel lehet hozzáférni, szemben az előző osztály betöltő/tároló műveleteivel. Amint azt a 20.3. ábra mutatja, minden csúcs egy processzorból, gyorsító memóriából és helyi memóriából áll, tartalmazza az operációs rendszer egy saját példányát, vagy legalább annak központi részét. Míg az SMP, ccNUMA és nccNUMA architektúrákat rendszerint a közös memóriájú gépekhez sorolják, a NORMA típusú gépeket, klasztereket és grideket az osztott memóriájúakhoz.

Klaszterek

A ***klaszter*** olyan párhuzamos vagy osztott számítógéprendszer, amely egységes felépítésű, egymással összekötött számítógépekből áll. A számítógép itt személyi számítógépet, munkaállomást vagy – egyre gyakrabban – szimmetrikus multiprocesszort jelent, azaz olyan hálózati elemet, amely processzorból (vagy processzorokból), memóriából, esetleg perifériákból és operációs rendszerből áll. Az egységes feldolgozó elemekből álló rendszert SSI tulajdonságúnak (**S**ingle **S**ystem **I**mage) mondjuk, ha csak a rendszerbe lehet bejelentkezni, az egyes elemekre nem – vagy ha egyetlen fájlrendszer van.

Természetesen az SSI tulajdonság fokozatos, ezért a határvonal az osztott rendszerek és klaszterek között nem éles. A NORMA rendszerek és klaszterek közötti határvonal is bizonytalan, mivel attól is függ, hogy eleve egységes rendszert terveztek vagy meglévő, eredetileg különálló komponenseket kapcsoltak össze.

A klaszterek osztályozhatók aszerint, hogy párhuzamos számításokra, nagyméretű számításokra vagy széles körű hozzáférhetőségre alkalmazzák-e őket. A párhuzamos számításokra használt klaszterek tovább oszthatók a ***kiemelt klaszterekre,*** melyeket eleve úgy terveztek, hogy párhuzamos gépekként üzemeljenek, vagy ***kampus-klaszterekre,*** amelyek az idő egy részében párhuzamos gépekként alkalmazott osztott rendszerek. A dedikált klaszterek elemei rendszerint nem rendelkeznek perifériákkal és nagy sebességű hálózati vonalakkal vannak összekötve. A kampusz-klaszterek viszont gyakran asztali személyi számítógépek, melyek között a belső kommunikáció közönséges hálózati vonalak segítségével valósul meg.

Gridek

Foster és Kesselman szerint „A grid egy hardver/szoftver infrastruktúra az erőforrások közös használatához és a problémák megoldásához". A gridek lehetővé teszik az olyan erőforrásokhoz való összehangolt hozzáférést, mint a processzorok, memóriák, adatok, perifériák stb. A gridek abban különböznek a párhuzamos számítási architektúrától, hogy nagyok, heterogének és dinamikusan változnak. A gridek vezérlése ezért bonyolult feladat.

20.2. Hatékonysági mértékek és optimalizálás

Ebben az alfejezetben először a gyakorlatban használt hatékonysági mértékeket és optimalizálási módszereket, azután pedig az algoritmusok aszimptotikus elemzésénél felhasználható fogalmakat tekintjük át.

20.2.1. Hatékonyság a gyakorlatban

Amint azt a bevezetésben leírtuk, a párhuzamos számítások során a feladatokat ***taszkokra*** bontjuk és a taszkokat egymástól függetlenül megoldjuk. A taszkokat ***folyamatokként*** vagy ***szálakként*** valósítjuk meg. Más szavakkal, a folyamatok végrehajtás alatt lévő programok. Minden folyamattal kapcsolatban tárolunk olyan erőforrásokkal kapcsolatos adatokat, mint a memóriaszegmensek, fájlok és jelek, míg a szálak a folyamatokon belül léteznek és a többszörös szálak megosztják egymás között az erőforrásokat. Adott folyamat szálai hozzáférnek a közös memóriához, míg a folyamatok rendszerint

üzenetekkel kommunikálnak. Minden szálnak van egy utasításszámlálója vagy más regiszterértéke, valamint egy verme a helyi változók tárolására. A folyamatokat tekinthetjük az erőforráshasználatot segítő egységnek, míg a szálakat a központi feldolgozó egységen való végrehajtást segítő egységnek. Mivel kevesebb információ tárolására van szükség, gyorsabban lehet szálakat létrehozni, megszüntetni és egyik szálról a másikra átkapcsolni, mint ugyanezeket a műveleteket folyamatokkal elvégezni.

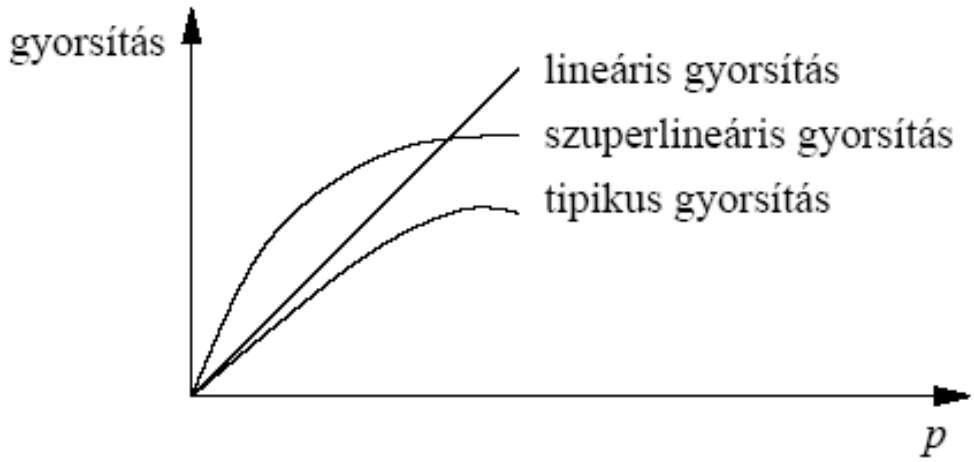
Az architektúrától függ, hogy szálakat vagy folyamatokat alkalmazunk-e. A közös memóriájú gépeken a szálak rendszerint gyorsabban futtathatók, ugyanakkor viszont a folyamatokat felhasználhatjuk a programok hordozhatóságának biztosítására. A szálak alkalmazhatók, ha van egy szoftver réteg (osztott közös memória), amely megvalósítja a közös memória absztrakcióját, de ezeknek a szálaknak nagyobbak a kommunikációs költségei.

Míg a taszkok fogalma a problémákhoz kapcsolódik, a folyamatok és szálak fogalma a megvalósításhoz kötődik. Amikor egy algoritmust tervezünk, rendszerint nagyszámú taszkot azonosítunk, amelyek potenciálisan futhatnak párhuzamosan, majd ezek közül többet ugyanarra a folyamatra vagy szálra képezünk le.

Párhuzamos programok két stílusban írhatók és ezek a stílusok keveredhetnek: az egyik stílusra ***adatpárhuzamosság*** jellemző, azaz ugyanazt a műveletet egyidejűleg különböző adatokra alkalmazzuk. Ez a művelet lehet egy gépi utasítás, mint a SIMD architektúrákban, vagy egy összetett művelet, például egy függvény alkalmazása. Utóbbi esetben különböző processzorok különböző műveleteket hajtanak végre. A másik stílus jellemzője a ***taszkpárhuzamosság,*** azaz a folyamatok/szálak különböző taszkokat hajtanak végre. Mivel egy függvény külső konstrukcióként tartalmazhat például **if** vagy **case** parancsot, az adatpárhuzamosság és taszkpárhuzamosság közti határvonal nem éles.

A folyamatok segítségével megvalósított párhuzamos programok tovább osztályozhatók: az SPMD (**S**ingle **P**rogram **M**ultiple **D**ata) programozási stílus azt jelenti, hogy minden folyamat ugyanazt a programot futtatja, míg az MPMD (**M**ultiple **P**rogram **M**ultiple **D**ata) stílus alkalmazásakor a folyamatok különböző programokat futtatnak. Az MPMD programok taszkpárhuzamosak, míg az SPMD programok mind taszkpárhuzamosak, mind pedig adatpárhuzamosak lehetnek. SPMD módban a taszkpárhuzamosságot feltételes utasítások segítségével fejezzük ki.

Mivel a párhuzamos számítások elsődleges célja a programok gyors futtatása, a teljesítménymértékek fontos szerepet játszanak ezen a területen. Természetes abszolút mérték a végrehajtási idő, de még gyakrabban használ-



20.4. ábra. Ideális, tipikus és szuperlineáris gyorsítási görbék.

nak egy relatív mértéket, a gyorsítást. Adott problémára a ***gyorsítást*** a

$$gyorsítás(p) = T_1/T_p$$

hányadossal definiáljuk, ahol T_1 a leggyorsabb ismert soros algoritmus futási ideje p processzoron. A környezettől függően a gyorsítás vonatkozhat p folyamat vagy p szál alkalmazására is. A gyorsítással kapcsolatos, de ritkábban használt mérték a ***hatékonyság,*** melyet a

$$hatékonyság(p) = gyorsítás(p)/p$$

összefüggéssel definiálunk.

Ettől a definíciótól függetlenül a *hatékonyságot* a jó teljesítmény szinonimájaként is használják.

A 20.4. ábra bemutatja az ideális, tipikus és szuperlineáris gyorsítási görbéket. Az ideális görbe azt a feltevést tükrözi, hogy ha a processzorok számát megkétszerezzük, akkor a végrehajtási idő felére csökken. Ezért az ideális gyorsítás egységnyi hatékonyságnak felel meg. A szuperlineáris gyorsítás a gyorsításnak köszönhetően fordulhat elő, azaz akkor, ha a több processzor alkalmazása megnöveli a gyorsítótár méretét, és így több adathozzáférés szolgálható ki a gyorsítótárból – ahelyett, hogy a lassú központi memóriához kellene fordulni.

A tipikus gyorsítás az ideális gyorsításnál kisebb és csak bizonyos processzorszámig nő. Emellett több processzor alkalmazása lassítja a programot.

A tipikus és ideális gyorsítás közötti különbségnek több oka van:

- ***Amdahl törvénye*** azt mondja, hogy minden programnak van egy s soros hányada, amely nem párhuzamosítható. Ezért $T_p > s$, és így $\mathit{gyorsítás}(p) < T_1/s$, azaz a gyorsítás mindig kisebb, mint egy konstans érték. Szerencsére egy másik gyakorlati megfigyelés, a ***Gustafson–Barsis-törvény*** csökkenti az Amdahl-törvény gyakorlati szerepét. A Gustafson-Barsis-törvény szerint a tipikus alkalmazásokban a párhuzamos algoritmus alkalmazása nem korlátozódik egy rögzített méretű probléma megoldására, hanem egyre nagyobb méretű feladatokat oldunk meg. Ebben az esetben s lassabban nőhet, mint T_1, és így T_1/s már nem konstans.
- A taszkkezelés, azaz a taszkok indítása, megállítása, megszakítása, valamint a folyamatok és szálak ütemezése bizonyos többletterhelést okoz. Továbbá az is rendszerint igaz, hogy az elvégzendő munkát nem lehet egyenletesen elosztani a folyamatok/szálak között.
- A kommunikáció és a szinkronizáció lassítja a programot. A kommunikáció az adatcserét jelenti, míg a szinkronizáció más típusú koordinációt jelent, például a kölcsönös kizárás biztosítását. A kommunikációs és szinkronizációs költségek még a nagy sebességű vonalakkal rendelkező hálózatokban is nagyságrendileg nagyobbak, mint a számítási költségek. Ennek oka a fizikai átviteli költségek mellett a protokoll számítási többletigénye és a hálózati erőforrásokért való versenyzés okozta késedelem.

A fenti tényezők hatásának minimalizálásával a teljesítmény javítható. Amdahl törvényét nehéz megkerülni, kivéve azt az esetet, ha új algoritmust tudunk tervezni, amelyre nézve s értéke kisebb – esetleg olyan áron, hogy T_1 értéke nagyobb lesz. Az algoritmikus fogalmakat később tekintjük át – most a gyakorlatban használt jellemzőket mutatjuk be.

Amint korábban láttuk, a taszkokat olyan folyamatokként vagy szálakként valósítjuk meg, amelyek rendszerint többszörös taszkokra vonatkoznak. A nagyobb teljesítmény érdekében a folyamatok/szálak szemcsézettségét az architektúrának megfelelően választjuk meg. Sok folyamat/szál szükségtelenül megnöveli a taszkkezelési költségeket, míg kevés folyamat/szál a számítógép alacsony kihasználtságát eredményezi. Ezért célszerű több folyamatot/szálat ugyanahhoz a processzorhoz rendelni, mivel így a processzorok átkapcsolhatnak, amikor bevitel/kivitel vagy más ok miatt várakozniuk kellene. A nagy szemcséjű folyamatok/szálak előnye a jobb kommunikáció/számítások arány, míg a finomabb szemcséjű folyamatok/szálak alkalmasabbak a terhelés egyenletes elosztására.

A terheléskiegyenlítés végezhető statikus vagy dinamikus módon. Ha a taszkok futási ideje előre megbecsülhető, akkor a statikus sémákat részesítjük

előnyben. Ezeknél a módszereknél a programozó minden folyamathoz/szálhoz hozzárendel bizonyos számú folyamatot/szálat úgy, hogy az összköltség minden processzoron hasonló nagyságú legyen. A dinamikus megoldásra példa a mester-szolga séma. Ebben a sémában először a mester folyamat hozzárendel minden processzorhoz egy folyamatot. Ezután valahányszor egy szolga befejez egy taszkot, értesíti erről a mestert és az új taszkot rendel a processzorhoz mindaddig, amíg minden taszk végre nem hajtódik. Ez a séma jó terheléskiegyenlítést biztosít, aminek az ára a taszkkezelés okozta többletterhelés.

A hatékonyság növelésének legjobb eszköze rendszerint a kommunikációs-szinkronizációs költségek csökkentése. Természetesen javulás érhető el az architektúra vagy a rendszerszoftver változtatásával, például a *késleltetésnek* – azaz a távolról érkező adatra várás idejének – csökkentésével, vagy a ***sávszélesség*** – azaz az időegység alatt átvitt adatmennyiség – növelésével.

Az algoritmustervező vagy felhasználói programozó csökkentheti a kommunikációs/szinkronizációs költségeket a beavatkozások számának csökkentésével. Az egyik fontos megközelítés ennek a minimalizálásnak az elvégzésére a helyi optimalizálás. A ***lokalitás*** a (soros vagy párhuzamos) programok olyan tulajdonsága, amely tükrözi az azonos adatokhoz időben vagy térben koncentrálva való hozzáférés mértékét. Az osztott memóriájú architektúrákban például az adatokat annál a processzornál célszerű tárolni, ahol felhasználjuk őket. A lokalitás javítható kódtranszformációval, adattranszformációval vagy a kettő kombinációjával. Például tekintsük a következő programrészlet végrehajtását három processzoron:

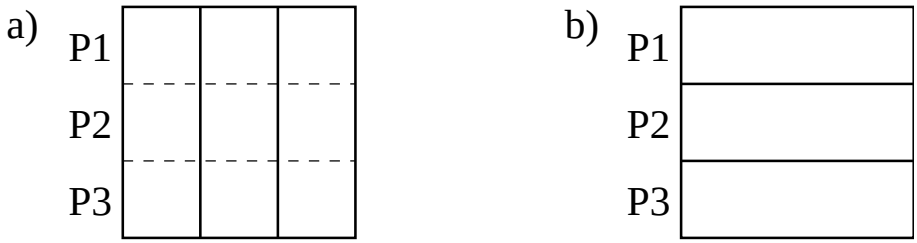
```
for (i=0; i<N; i++) in parallel
  for (j=0; j<N; j++)
    f(A[i][j]);
```

Itt az *in parallel* kifejezés azt jelenti, hogy az iterációk egyenletesen vannak elosztva a processzorok között úgy, hogy P_0 futtatja az $i = 0, \ldots, N/3$ iterációt, P_1 futtatja az $i = N/3 + 1, \ldots, 2N/3$ iterációkat és P_2 futtatja az $i = 2N/3 + 1, \ldots, N - 1$ iterációkat (az indexek szükség szerint kerekítve értendők). Feltesszük, hogy az f függvény mellékhatásoktól mentes.

A 20.5(a) ábrán lévő adatokkal a lokalitás kicsi, mivel sok hozzáférés szükséges a távoli memóriához. A lokalitás javítható úgy, hogy az adatok elosztását a 20.5(b) ábrán láthatóra változtatjuk, vagy a programot változtatjuk a következőre:

```
for (j=0; j<N; j++) in parallel
  for (i=0; i<N; i++)
    f(A[i][j]);
```

A második lehetőség, a kódtranszformáció előnye, hogy alkalmazható



20.5. ábra. Lokalitás optimalizálása az adatok transzformálásával.

szelektíven a kód egy részére, míg az adattranszformáció a program egészére hat és míg az egyik helyen javulást okoz, egy másik helyen hátrányos lehet. Az adatelosztások mindig helyesek, míg a kódtranszformációnak figyelembe kell vennie az *adatfüggőségeket,* melyek az utasítások sorrendjéből következnek. Például az

```
a = 3;          (1)
b = a;          (2)
```

kódrészletben az (1) és (2) utasítások között adatfüggőség van. A két utasítás sorrendjének megváltoztatása hibás programot eredményezne.

Közös memóriájú architektúrákon a programozó nem határozza meg az adatok eloszlását. A programok gyorsabban futnak, ha az együtt felhasznált adatok a gyorsítótárnak ugyanazon az adatátviteli vonalán vannak. A közös memóriájú architektúráknál az adatok elosztását a fordítóprogram végzi, például a C nyelvben sorfolytonosan. A programozónak csak közvetett befolyása van azzal, hogyan deklarálja az adatszerkezeteket.

A kommunikációs költségek csökkentésének egy másik módja az adatok másolása. Például kifizetődő a gyakran használt adatok több processzoron való tárolása, vagy rövid számítások megismétlése az eredmények elküldése helyett.

A szinkronizáció szükséges a helyességhez, de lassítja a program végrehajtását, egyrészt a saját végrehajtási ideje miatt, másrészt azzal, hogy a processzorokat arra kényszeríti, hogy egymásra várjanak. Ezért a szinkronizáció széles körű alkalmazását célszerű elkerülni. Például a kritikus szakaszok (melyekben a folyamatok/szálak ugyanahhoz az erőforráshoz igényelnek kizárólagos hozzáférést) számát célszerű minimális értéken tartani. *Sorossá tételnek* nevezzük azt, amikor legfeljebb egy folyamat aktív olyankor, amikor van várakozó folyamat.

Végül a hatékonyság javítható a késleltetés elrejtésével, azaz a kommunikáció és a számítás közötti párhuzamossággal. Például egy folyamat valamivel előbb elkezd egy távoli helyről való olvasást (előre való betöltés)

vagy a következő számítással párhuzamosan végzi a távoli memóriába való írást.

20.2.2. Hatékonyság az elemzésekben

Az algoritmusok elemzése során az igényelt erőforrások mennyiségét *abszolút* és *relatív* mennyiségekkel jellemezzük.

Ezeknek a mennyiségeknek a célszerű megválasztása attól is függ, hogy az algoritmus konkrét vagy absztrakt gépen (számítási modellen) fut.

Jelöljük $W(n, \pi, \mathrm{A})$-val, illetve $W(n, \pi, p, \mathrm{P})$-vel azoknak azt az időt, amelyekkel a π problémát az A soros, illetve a P párhuzamos algoritmus – (utóbbi p processzort felhasználva) – n méretű feladatokra legrosszabb esetben megoldja.

Hasonlóképpen jelöljük $B(n, \pi, \mathrm{A})$-val, illetve $B(n, \pi, p, \mathrm{P})$-vel azoknak a lépéseknek a számát, amelyekkel a π problémát az A soros, illetve a P párhuzamos algoritmus (utóbbi p processzort felhasználva) – n méretű feladatokra legjobb esetben megoldja.

Jelöljük $N(n, \pi)$-vel, illetve $N(n, \pi, p)$-vel azt az időt, amelyekre az n méretű π feladat megoldásához mindenképpen szüksége van bármely soros, illetve bármely párhuzamos algoritmusnak – utóbbinak akkor, ha legfeljebb p processzort vehet igénybe.

Tegyük fel, hogy minden n-re adott a π feladat n méretű konkrét előfordulásainak $D(n, \pi)$ eloszlásfüggvénye. Ekkor legyen $E(n, \pi, \mathrm{A})$, illetve $E(n, \pi, p, \mathrm{P})$ annak az időnek a várható értéke, amennyi alatt a π problémát n méretű feladatokra megoldja az A soros, illetve a P párhuzamos algoritmus – utóbbi p processzort felhasználva.

Az elemzések során gyakori az a feltételezés, hogy az azonos méretű bemenetek előfordulási valószínűsége azonos. Ilyenkor ***átlagos*** futási időről beszélünk, amit $A(n, \mathrm{A})$-val jelölünk.

A W, B, N, E és A jellemzők függnek attól a számítási modelltől is, amelynek alapul vételével az algoritmust megvalósítjuk. Az egyszerűség kedvéért feltesszük, az algoritmus egyúttal a számítási modellt is egyértelműen meghatározza.

Amennyiben a szövegkörnyezet alapján egyértelmű, hogy mely problémáról van szó, akkor a jelölésekből π-t elhagyjuk.

Ezek között a jellemzők között érvényesek az

$$N(n) \leq B(n, \mathrm{A}) \tag{20.1}$$
$$\leq E(n, \mathrm{A}) \tag{20.2}$$
$$\leq W(n, \mathrm{A}) \tag{20.3}$$

egyenlőtlenségek. Hasonlóképpen a párhuzamos algoritmusok jellemző

adataira az

$$\begin{aligned} N(n,p) &\leq B(n,\mathrm{P},p) && (20.4)\\ &\leq E(n,\mathrm{P},p) && (20.5)\\ &\leq W(n,\mathrm{P},p) && (20.6) \end{aligned}$$

egyenlőtlenségek teljesülnek. Az átlagos futási időre pedig a

$$\begin{aligned} B(n,\mathrm{A}) &\leq A(n,\mathrm{A}) && (20.7)\\ &\leq W(n,\mathrm{A})\ , && (20.8) \end{aligned}$$

illetve

$$\begin{aligned} B(n,\mathrm{P},p) &\leq A(n,\mathrm{P},p) && (20.9)\\ &\leq W(n,\mathrm{P},p) && (20.10) \end{aligned}$$

áll fenn.

Hangsúlyozzuk, hogy ezek a jelölések nem csak a futási időre, hanem az algoritmusok más jellemzőire – például memóriaigény, küldött üzenetek száma – is alkalmazhatók.

Ezután relatív jellemzőket, úgynevezett *hatékonysági mértékeket* definiálunk.

A gyorsítás azt mutatja, hányszor kisebb a párhuzamos algoritmus futási ideje a soros algoritmus futási idejénél.

A P párhuzamos algoritmusnak az A soros algoritmusra vonatkozó ***gyorsítása*** (vagy relatív lépésszáma)

$$g(n,\mathrm{A},\mathrm{P}) = \frac{W(n,\mathrm{A})}{W(n,p,\mathrm{P})}\ . \qquad (20.11)$$

Ha a gyorsítás egyenesen arányos a felhasznált processzorok számával, akkor lineáris gyorsításról beszélünk. Ha a P párhuzamos és az A soros algoritmusra

$$\frac{W(n,\mathrm{A})}{W(n,p,\mathrm{P})} = \Theta(p)\ , \qquad (20.12)$$

akkor P A-ra vonatkozó gyorsítása ***lineáris***.

Ha a P párhuzamos és az A soros algoritmusra

$$\frac{W(n,\mathrm{A})}{W(n,p,\mathrm{P})} = o(p)\ , \qquad (20.13)$$

akkor P A-ra vonatkozó gyorsítása ***szublineáris.***

Ha a P párhuzamos és az A soros algoritmusra

$$\frac{W(n,\mathrm{A})}{W(n,p,\mathrm{P})} = \omega(p) \ , \tag{20.14}$$

akkor P A-ra vonatkozó gyorsítása ***szuperlineáris.***

A párhuzamos algoritmusok esetében fontos jellemző az $m(n,p,\mathrm{P})$ ***munka***, amit a futási idő és a processzorszám szorzatával definiálunk. Akkor is ez a szokásos definíció, ha a processzorok egy része csak a futási idő egy részében dolgozik:

$$m(n,p,\mathrm{P}) = pW(n,p,\mathrm{P}) \ . \tag{20.15}$$

Érdemes hangsúlyozni, hogy – különösen az aszinkron algoritmusok esetében – a ténylegesen elvégzett lépések száma lényegesen kevesebb lehet, mint amit a fenti (20.15) képlet szerint kapunk.

Egy P párhuzamos algoritmusnak az ugyanazon problémát megoldó soros algoritmusra vonatkozó $h(n,p,\mathrm{P},\mathrm{A})$ ***hatékonysága*** a két algoritmus munkájának hányadosa:

$$h(n,p,\mathrm{P},\mathrm{A}) = \frac{W(n,A)}{pW(n,p,\mathrm{P})} \ . \tag{20.16}$$

Abban a természetes esetben, amikor a párhuzamos algoritmus munkája legalább akkora, mint a soros algoritmusé, a hatékonyság nulla és egy közötti érték – és a viszonylag nagy érték a kedvező.

Központi fogalom a párhuzamos algoritmusok elemzésével kapcsolatban a munkahatékonyság. Ha a P párhuzamos és A soros algoritmusra

$$pW(n,p,\mathrm{P}) = O(W(n,\mathrm{A})) \ , \tag{20.17}$$

akkor P A-ra nézve ***munkahatékony.***

Ez a meghatározás egyenértékű a

$$\frac{pW(n,p,\mathrm{P})}{W(n,\mathrm{A})} = O(1) \tag{20.18}$$

egyenlőség előírásával. Eszerint egy párhuzamos algoritmus csak akkor munkahatékony, ha összes munkája nagyságrendileg nem nagyobb, mint a soros algoritmus munkája.

Ha egy A soros – illetve P párhuzamos – algoritmus egy feladat megoldásához adott erőforrásból csak $O(N(n))$ – illetve $O(N(n,p))$ – mennyiséget használ fel, akkor az A algoritmust az adott erőforrásra (és számítási modellre) nézve ***aszimptotikusan optimálisnak*** nevezzük.

Ha egy A soros – illetve P párhuzamos – algoritmus egy feladat

megoldásához adott erőforrásból a bemenet minden lehetséges $n \geq 1$ mérete esetében csak az okvetlenül szükséges $N(n, \mathrm{A})$ – illetve $N(n, p, \mathrm{A})$ – mennyiséget használja fel, azaz

$$W(n, \mathrm{A}) = N(n, \mathrm{A}) \ , \tag{20.19}$$

illetve

$$W(n, p, \mathrm{P}) = N(n, p, \mathrm{P}) \ , \tag{20.20}$$

akkor az algoritmust az adott erőforrásra (és az adott számítási modellre) nézve ***abszolút optimálisnak*** nevezzük és azt mondjuk, hogy $W(n, p, \mathrm{P}) = N(n, p, \mathrm{P})$ a vizsgált feladat ***pontos bonyolultsága.***

Két algoritmus összehasonlításakor a

$$W(n, \mathrm{A}) = \Theta(W(n, \mathrm{B})) \tag{20.21}$$

esetben azt mondjuk, hogy a A és B algoritmus futási idejének növekedési sebessége aszimptotikusan ***azonos nagyságrendű.***

Amikor két algoritmus futási idejét (például a legrosszabb esetben) összehasonlítjuk, akkor gyakran találunk olyan váltási helyeket, melyeknél kisebb méretű bemenetekre az egyik, míg nagyobb méretű bemenetekre a másik algoritmus futási ideje kedvezőbb. A formális definíció a következő: ha a pozitív egész helyeken értelmezett $f(n)$ és $g(n)$ függvényekre, valamint a $v \geq 0$ pozitív egész számra teljesül, hogy

1. $f(v) = g(v)$;
2. $(f(v-1) - g(v-1))(f(v+1) - g(v+1)) < 0$,

akkor a v számot az $f(n)$ és $g(n)$ függvények ***váltási helyének*** nevezzük.

Például két mátrix szorzatának a definíció alapján történő és a Strassen-algoritmussal történő kiszámítását összehasonlítva (lásd például Cormen, Leiserson, Rivest és Stein többször idézett új könyvét) azt kapjuk, hogy kis mátrixok esetén a hagyományos módszer, míg nagy mátrixok esetén a Strassen-algoritmus az előnyösebb – egy váltási hely van, melynek értéke körülbelül 20.

Gyakorlatok

20.2-1. Határozzuk meg azokat a részfeladatokat, amelyek a szokásos mátrixszorzás során párhuzamosíthatók. Lehetőleg minél több részfeladatot adjunk meg. Ezután ajánljunk különböző lehetőségeket a részfeladatok (kisebb számú) szálakra való felbontására, majd hasonlítsuk össze ezeket a leképezéseket a közös memóriájú architektúrára való leképezés hatékonyságával.

20.2-2. Tekintsünk egy párhuzamos programot, melynek bemenete egy n egész szám és kimenete a $2 \leq p \leq n$ feltételnek eleget tevő prím számok sorozata. A T_i taszk meghatározza, hogy i prímszám-e. Ezt úgy dönti el, hogy az i számot rendre elosztja a potenciális osztókkal, azaz a $2, \ldots, \sqrt{i}$ számokkal. A programot rögzített számú folyamattal és szállal valósítsuk meg. Ajánljunk különböző lehetőségeket ennek megoldására, elemezzük előnyeiket és hátrányaikat. Vegyük figyelembe mind a statikus, mind pedig a dinamikus terhelés-kiegyensúlyozó sémákat.

20.2-3. Határozzuk meg a következő stencil kód adatösszefüggéseit:

```
for (t=0; t<tmax; t++)
  for (i=0; i<n; i++)
    for (j=0; j<n; j++)
      a[i][j] += a[i-1][j] + a[i][j-1];
```

Alakítsuk át úgy a kódot, hogy párhuzamosítható legyen.

20.3. Párhuzamos programozás

Részben a különböző architektúrák alkalmazása, részben pedig a terület újdonsága miatt a párhuzamos programozás számos modellje ismert. A legnépszerűbb modellek ma az üzenetküldés, amelyet az MPI (**M**essage **P**assing **I**nterface) szabvány és a strukturált közös memóriájú programozás, amelyet az OpenMP szabvány egységesít. Ezeket a programozási modelleket az 20.3.1., illetve a 20.3.2. pontokban tárgyaljuk. További fontos modelleket, mint a szálprogramozás, adatpárhuzamosság és automatikus párhuzamosítás a 20.3.3. pontban vázolunk.

20.3.1. MPI programozás

Amint a neve mutatja, az MPI az üzenetküldő programozási modellen alapul. Ebben a modellben több program fut párhuzamosan és kommunikál egymással üzeneteket küldve és kapva. A folyamatok nem férnek hozzá a közös memóriához, minden kommunikáció közvetlen üzenetcserével történik. A kommunikáció pontosan két folyamatot tételez fel: az egyik a *küld* műveletet, a másik pedig a *fogad* műveletet hajtja végre. Az üzenetküldés mellett az MPI csoportos műveleteket és kommunikációs eszközöket is tartalmaz.

Az üzenetküldés asszimetrikus abban az értelemben, hogy a küldőnek fel kell fednie kilétét, míg a fogadó vagy felfedi kilétét vagy kinyilvánítja készségét, hogy bármilyen forrásból adatokat kapjon.

Mivel mind a küldőnek, mind pedig a fogadónak aktívan részt kell vennie a kommunikációban, a programozónak előre meg kell terveznie, mikor fognak

egy adott folyamatpár elemei egymással kommunikálni. Üzenet cseréjére több célból is sor kerülhet:

- a programozó által előre megtervezett adatok olyan tulajdonságainak cseréje, mint az adatok mérete vagy típusa;
- olyan vezérlő információ cseréje, amely a későbbi üzenetcserére vonatkozik;
- szinkronizáció, melyet azzal érünk el, hogy a beérkező üzenet tájékoztatja a fogadót a küldő állapotáról. Amint később látni fogjuk, ez kiegészíthető azzal, hogy a küldő kap információt a fogadó állapotáról. Megjegyezzük, hogy a szinkronizáció a kommunikáció speciális esete.

Az MPI szabványt 1994-ben vezette be az MPI fórum, amely hardver- és szoftverkereskedők, kutatóintézetek és egyetemek egy csoportja. A lényegesen kiterjesztett MPI-2 változat 1997-ben jelent meg. Az MPI-2-nek ugyanazok az alaptulajdonságai, de kiegészítő függvényosztályokat vezet be.

Az MPI számos olyan könyvtári függvényt alkalmaz, amelyek összekötik a C, C++ és FORTRAN nyelveket. Az MPI legtöbb funkciója a folyamatok közötti kommunikációval kapcsolatos és nyitva hagy olyan folyamatkezelési problémákat, mint a folyamatok elindítása és megállítása – bár ez alól az MPI-2-ben számos kivétel van. Ezeket a lehetőségeket a szabványon kívül kell megoldani, és ezért a megoldások nem hordozhatók. Ezért és további okokból az MPI programok tipikusan rögzített folyamathalmazt alkalmaznak – és ezek a folyamatok egyszerre indulnak el akkor, amikor a program végrehajtása megkezdődik. A programok kódolhatók SPMD vagy MPMD stílusban. Párhuzamos programokat írhatunk úgy, hogy mindössze hat alapfüggvényt alkalmazunk:

- Az MPI_Init függvényt minden más MPI függvény előtt meg kell hívni.
- Az MPI_Finalize függvényt az utolsó MPI függvény után kell meghívni.
- Az MPI_Comm_size függvény megadja a programban lévő folyamatok számát.
- Az MPI_Comm_rank függvény megadja a hívó folyamat számát, a számozást nullától kezdve.
- Az MPI_Send függvény üzenetet küld. Ennek a függvénynek a következő paraméterei vannak:
  - az üzenet címe, mérete és adattípusa, valamint a fogadó sorszáma;
  - üzenet címkéje, ami egy olyan szám, amely az üzenetet ahhoz hasonló módon jellemzi, ahogyan az elektronikus levelet a tárgya;
  - kommunikátor, amely folyamatok egy csoportja, ahogy azt a továbbiakban elmondjuk.

- Az MPI_Recv függvény egy üzenetet kap. A függvénynek ugyanazok a paraméterei, mint az MPI_Send függvénynek, azzal az eltéréssel, hogy az üzenet méretére csak egy felső korlátot kell megadni, továbbá a küldőre egy szabad paraméter adható meg, és egy státusznak nevezett paraméter adja meg a fogadott üzenet küldőjét, méretét és címkét.

  A 20.6. ábra egy MPI programot mutat be.

Bár a fenti funkciók elegendők egy egyszerű program megírásához, számos további funkció segít az MPI programok hatékonyságának és/vagy szerkezetének javításában. Az MPI-1 többek között a következő függvényosztályokat támogatja:

- *Alternatív funkciók a páronkénti kommunikációhoz.* Az MPI_Send függvény, amelyet szabványos küldő függvénynek is neveznek, vagy a fogadó által elküldött üzenetet, vagy a rendszer által a bufferba tett üzenet ad vissza. A két lehetőség közül az MPI választ. Az MPI_Send függvény különböző változatai a következő lehetőségek valamelyikét választják: szinkronizált módban a függvény csak akkor küld üzenetet, ha először a fogadó kapott üzenetet – ezzel szinkronizál mindkét irányban. A buffer módban a rendszernek akkor kell az üzenetet tárolnia, ha a fogadó még nem alkalmazta az MPI_Recv függvényt.

  Mind a küldőnél, mind pedig a fogadónál a függvényeknek a szabványos, szinkronizált és buffer módban is van blokkolt és blokkolatlan változata. A blokkolt változatot korábban leírtuk. A blokkolatlan változatok közvetlenül hívás után válaszolnak, és hagyják, hogy a küldő/fogadó folytassa a program végrehajtását mindaddig, míg a rendszer a háttérben befejezi a kommunikációt. A blokkolatlan kommunikációnak az MPI_Wait vagy MPI_Test függvényekkel kell befejeződnie azért, hogy biztosak lehessünk abban, hogy a kommunikáció befejeződött és a buffer újra felhasználható. A befejezési függvények lehetővé teszik a többszörös igényekre való várakozást.

  z MPI programok holtpontra juthatnak, ha például a P_0 folyamat először az MPI_Send függvényt alkalmazza a P_1 folyamattal kapcsolatban, majd ugyanezt teszi a P_1 folyamat P_0-lal kapcsolatban, ezután pedig ugyanez történik fordított szereposztással. Az MPI támogatja a kombinált küld/fogad függvényt.

  Számos programban a folyamatpárok ismételten cserélnek adatokat ugyanazon buffer felhasználásával. A kommunikációs többletterhelés csökkentése érdekében címeket tartalmazó címkéket alkalmazhatunk – ezt ***állandó kommunikációnak*** nevezzük. Végül az MPI_Probe és MPI_Iprobe függvények lehetővé teszik, hogy az üzenet méretét és más

```
#include <stdio.h>
#include <string.h>
#include "mpi.h"

int main (int argc, char **argv) {
  char msg[20];
  int me, total, tag=99;
  MPI_Status status;

  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &me);
  MPI_Comm_size(MPI_COMM_WORLD, &total);

  if (me==0) {
    strcpy(msg, "Hello");
    MPI_Send(msg, strlen(msg)+1, MPI_CHAR, 1, tag,
             MPI_COMM_WORLD);
  }
  else if (me==1) {
    MPI_Recv(msg, 20, MPI_CHAR, 0, tag, MPI_COMM_WORLD,
             &status);
    printf("Received: %s \n", msg);
  }
  MPI_Finalize();
  return 0;
}
```

20.6. ábra. Egy egyszerű MPI program.

jellemzőit az üzenet fogadása előtt megvizsgálhassuk.

- *Adattípusokat kezelő függvények.* Az üzenetküldés egyszerű formájában azonos típusú (például float) adatokat továbbítunk. Az MPI modell megengedi, hogy egy üzenetben különböző típusú adatokat továbbítsunk, és azt is, hogy nem folyamatos bufferekből küldjünk adatokat, például egy tömb minden második elemét küldjük el. Ehhez az MPI két függvényosztályt definiál: a felhasználó által definiált adattípusok adatpozíciókat és típusokat írnak le, míg a pakolási függvények abban segítenek, hogy egy bufferbe több adat kerüljön. Az MPI modell azzal is támogatja a heterogenitást, hogy szükség esetén automatikusan konvertálja az adatokat.

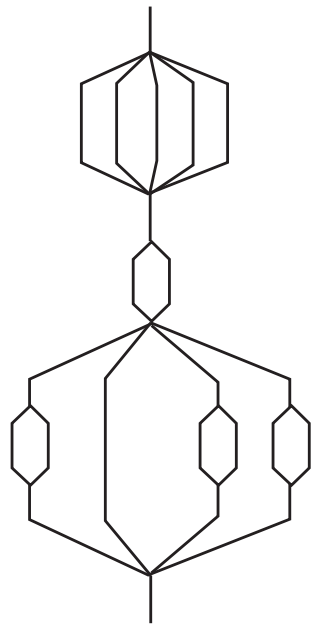
- *Kollektív kommunikációs függvények.* Ezek támogatják az olyan kom-

munikációs mintákat, mint az üzenetszórás. Bár minden kommunikációs minta megvalósítható küld/fogad függvények sorozatával, a csoportos függvények előnyösebbek, mivel javítják a program tömörségét és érthetőségét, és gyakran van optimizált megvalósításuk. Továbbá a megvalósítások kihasználhatják az architektúra sajátosságait, és így a másik gépre átvitt program az új gépen is hatékonyan futhat, felhasználva az adott gépre optimizált megvalósítást.

- *Csoport- és kommunikátor-vezérlő függvények.* Amint azt már említettük, a küld és fogad függvények tartalmaznak egy kommunikátor argumentumot, amely leírja a folyamatok egy csoportját. Technikai értelemben a kommunikátor egy osztott adatszerkezet, amely minden folyamatnak megmondja, hogyan érheti el a csoportjában lévő többi folyamatot, és ***attribútumoknak*** nevezett kiegészítő információt is tartalmaz. Ugyanaz a csoport különböző kommunikátorokkal is leírható. Az üzenetcsere csak akkor megy végbe, ha MPI_Send és MPI_Recv függvények kommunikátor argumentumai megfelelnek egymásnak. Ezért a kommunikátorok alkalmazása felbontja a programok üzeneteit diszjunkt halmazokra úgy, hogy azok a halmazok azután nem hatnak egymásra. Ily módon a kommunikátorok segítenek a programok strukturálásában és hozzájárulnak a programok helyességéhez. Az MPI segítségével megvalósított könyvtáraknál a kommunikátorok lehetővé teszik, hogy a könyvtári forgalmat és az alkalmazói programok forgalmát szétválasszuk. A csoportkommunikátorok a csoportos kommunikációhoz szükségesek. Az adatszerkezetben lévő attribútumok tartalmazhatnak olyan alkalmazás-specifikus információt, mint a hibakezelő. Az eddig leírt – csoporton belüli – kommunikátorok mellett az MPI támogatja a külső kommunikátorokat is.

  Az MPI-2 négy további függvénycsoportot alkalmaz:

- *Dinamikus folyamatkezelő függvények.* Ezekkel a függvényekkel a program futása során új MPI folyamatok indíthatók el. Továbbá, az egymástól függetlenül elindított MPI programok (mindegyik több folyamatból áll) ügyfél/kiszolgáló mechanizmus segítségével kapcsolatba kerülhetnek egymással.
- *Egyoldalú kommunikációs függvények.* Az egyoldalú kommunikáció a közös memória segítségével történő kommunikáció egyik típusa, melyben a folyamatok egy csoportja saját címterének egy részét közös memóriaként használja. A kommunikáció a közös memóriába való írással és az onnan való olvasással történik, Az egyoldalú kommunikáció abban különbözik a többi közös memóriás kommunikációtól – például az OpenMP-től – hogy a memóriához való hozzáféréshez explicit függvényekre van szükség.
- *Párhuzamos B/K függvények.* Számos függvény lehetővé teszi, hogy a



20.7. ábra. Egy OpenMP program szerkezete.

többszörös folyamatok csoportosan olvashassanak/írhassanak ugyanabból/ugyanabba a fájlba.

- *Kollektív kommunikációs függvények interkommunikátorokhoz.* Ezek a függvények általánosítják a csoportos kommunikáció fogalmát interkommunikátorokra. Például egy csoport egy folyamata üzenetet szórhat egy másik csoport minden folyamatához.

20.3.2. OpenMP programozás

Az OpenMP neve onnan származik, hogy nyitott szabvány a multiprogramozáshoz, azaz a közös memóriájú architektúrákhoz. A közös memória miatt ebben a pontban folyamatok helyett szálakról lesz szó.

A közös memóriával történő kommunikáció gyökeresen különbözik az üzenetküldéstől. Míg az üzenetküldés közvetlenül feltételez két folyamatot, a közös memória elválasztja a folyamatokat egy közbülső elem beiktatásával. Küld/fogad helyett olvas/ír kifejezéseket használunk, azaz egy szál ír a memóriába, egy másik szál pedig később olvas a memóriából. A szálaknak nem kell egymásról tudniuk, és egy beírt értéket több szál is olvashat. Az olvasás és az írás között tetszőleges hosszúságú idő telhet el. Ebben az esetben a szinkronizációt explicit módon kell szervezni: az olvasónak tudnia kell,

mikor fejeződött be az írás, és el kell kerülni azt, hogy ugyanazokat az adatokat egyidejűleg több szál is átalakíthassa.

Az OpenMP modell – az 20.3.1. pontban ismertetett MPI és a 20.3.3. alfejezetben ismertetett egyéb modellektől eltérően – nem tartalmaz egyoldalú kommunikációt. Az OpenMP abban különbözik a többi modelltől, hogy elágazásegyesítő szerkezete van, ahogyan azt a 20.7. ábra mutatja. A program végrehajtása egyetlen – ***mesterszálnak*** nevezett szál végrehajtásával kezdődik, majd a párhuzamos szakaszban több szál jön létre – ezek egyike a mesterszál. A párhuzamos szakaszok egymásba ágyazhatók, de a párhuzamos szálaknak egyszerre kell befejeződniük. Amint az ábra mutatja, a párhuzamos szakaszok sorozatot is alkothatnak úgy, hogy az egyes szakaszokban lévő szálak száma különböző.

Az OpenMP modell a könyvtári függvények helyett fordítóprogram parancsokat használ. Ezek a parancsok olyan útmutatásokat tartalmaznak, melyeket a fordítóprogram vagy figyelembe vesz, vagy nem. Például egy soros fordítóprogram figyelmen kívül hagyja ezeket a parancsokat. Az OpenMP modell támogatja a fokozatos párhuzamosítást, melyben egy soros programból indulunk ki, a hatékonyság szempontjából kritikus részeken fordítóprogram parancsokat alkalmazunk, majd később szükség esetén további parancsokat alkalmazunk.

Az OpenMP-t 1998-ban vezették be, 2.0 változata 2002-ben jelent meg. A fordítóprogramot vezérlő parancsok mellett az OpenMP néhány könyvtári funkciót és környezeti változót is alkalmaz. A szabvány C, C++ és FORTRAN nyelvekhez is rendelkezésre áll.

OpenMP-ben könnyebb programozni, mint az MPI szabvány szerint, mivel a fordítóprogram részekre bontja az elvégzendő munkát. Aki OpenMP-ben programoz, meghatározza a szálak számát, azután pedig a következő módok egyikén írja le a munka megosztását:

- *Explicit módon.* Egy szál az omp_get_thread_num paranccsal saját számot igényelhet. Egy egy feltételes utasítás értékeli ezt a számot és explicit módon taszkokat rendel a szálhoz – az MPI programok SPMD stílusához hasonló módon.

- *Párhuzamos ciklus.* A fordítóprogramot vezérlő #pragma omp parallel for parancs azt jelzi, hogy a következő for parancs végrehajtható párhuzamosan úgy, hogy minden szál több iterációt (taszkot) hajt végre. Erre egy példát mutat a 20.8. ábra. A programozó a munkát a schedule(static) vagy schedule(dynamic) parancsokkal befolyásolhatja. A statikus ütemezés azt jelenti, hogy minden szál az egymást követő iterációk hasonló méretű blokkját kapja meg. A dinamikus ütemezés azt jelenti, hogy először minden szálhoz egy iterációt rendelünk, azután pedig valahányszor egy szál

```
#include <omp.h>
#define N 100
double a[N][N], b[N], c[N];
int main() {
  int i, j;
  double h;
  /* a kezdőértékek beállítását elhagytuk */
  omp_set_num_threads(4);
  #pragma omp parallel for shared(a,b,c) private(j)
  for (i=0; i<N; i++)
    for (j=0; j<N; j++)
      c[i] += a[i][j] * b[j];
  /* az eredmények kivitelét elhagytuk */
}
```

20.8. ábra. Mátrix-vektor szorzás az OpenMP-ben párhuzamos ciklus alkalmazásával.

befejezi a kapott iterációt, kap egy újabbat, ahogyan azt korábban az MPI programozásnál a mester/szolga paradigmára leírtuk. Itt azonban – a mester/szolga megoldástól eltérően – a fordítóprogram dönti el, hogy melyik szál melyik taszkot hajtsa végre és a fordítóprogram határozza meg a szükséges kommunikációt is.

- *Taszkpárhuzamos szakaszok.* A #pragma omp parallel sections lehetővé teszi azon taszkok listájának leírását, amelyeket a rendelkezésre álló szálakhoz rendeltünk.

A szálak a közös memórián keresztül kommunikálnak, azaz a közös változókba írnak vagy azokat olvassák. A változóknak csak egy része közös, míg mások csak egy speciális szálhoz tartoznak. Azokat a szabályokat, amelyek meghatározzák, hogy egy változó közös vagy nem, a programozó felülírhatja.

Több OpenMP parancs azzal a szinkronizációval kapcsolatos, amely a kölcsönös kizáráshoz szükséges és lehetővé teszi a közös memória egységes kezelését. Bizonyos szinkronizációs parancsokat a fordítóprogram ad meg. Például a párhuzamos ciklus végén minden szál megvárja a többi szálat, mielőtt új ciklust kezdene.

20.3.3. Más programozási modellek

Bár az MPI és OpenMP a legnépszerűbb modellek, más megközelítéseknek is van gyakorlati jelentősége. Itt a szálprogramozást, a nagy teljesítményű FORTRAN-t és az automatikus párhuzamosítást mutatjuk be.

Az OpenMP-hez hasonlóan a ***szálprogramozásban*** – például a *phtreads*

könyvtárban vagy a Java nyelv szálainál – közös memóriát alkalmaznak. A szálak alacsonyabb absztrakciós szinten működnek, mint az OpenMP, amelyben a programozó a felelős a szálkezelés és a munkamegosztás minden részletéért. A szálakat egyesével konkrétan létre kell hozni, és hozzájuk kell rendelni az elvégzendő feladatot. A szálprogramozás a taszkpárhuzamosságra koncentrál, míg az OpenMP programozás az adatpárhuzamosságra. A szálprogramok strukturálatlanok lehetnek, azaz bármely szál létrehozhat új szálat vagy megállíthat egy másik szálat. Az OpenMP programokat gyakran alakítják át szálprogramokká.

Az adatpárhuzamosság más programozási stílushoz vezet, amelyet az olyan nyelvek támogatnak, mint például a a nagyteljesítményű FORTRAN (**H**igh **P**erformance **F**ortran). Míg az adatpárhuzamosság kifejezhető az MPI, OpenMP stb. eszközeivel, az adatpárhuzamos nyelvek másra helyezik a fő hangsúlyt. A HPF egyik fontos konstrukciója a párhuzamos ciklus, melynek iterációi egymástól függetlenül – azaz kommunikáció nélkül – végrehajthatók. Az adatpárhuzamos stílus könnyebben érthetővé teszi a programokat, mivel nem kell figyelmet fordítani a konkurens tevékenységekre. Hátránya, hogy nehézségeket okozhat az alkalmazások ilyen struktúrába kényszerítése. A HPF egyetlen osztott címtérrel rendelkezik, és a nyelv jelentős része az adatok mozgatásával foglalkozik. Míg az MPI programozók az adatokat explicit módon elküldik a megfelelő helyre, a HPF programozók az adatok szétosztását ahhoz hasonló absztrakt szinten írják le, mint ahogy az OpenMP programozók megadják a párhuzamos ciklusok ütemezését. A részletet a fordítóprogramra bízzák. Az OpenMP egyik fontos fogalma a tulajdonos-szabály, mely szerint egy értékadó utasítás bal oldali változójának tulajdonosa végzi el a műveletet. Ezért az adatok elosztásából adódik a számítások elosztása.

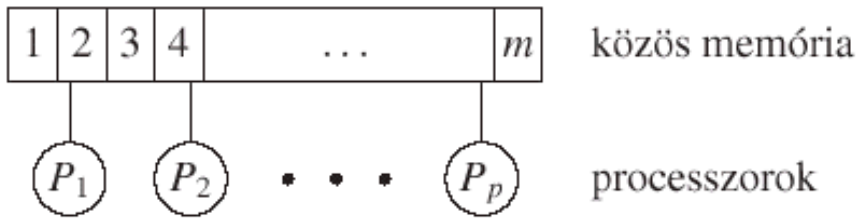
A ciklusok párhuzamosítása – különösen a tudományos számítások területén – jelentős teljesítmény-növekedést eredményez. Ez a párhuzamosítás gyakran elvégezhető automatikusan, párhuzamosító fordítóprogramok segítségével. Ezek a fordítóprogramok ellenőrzik az adatösszefüggéseket. Számos program szerkezete átalakítható a függőség csökkentése érdekében, a külső és a belső ciklusok felcserélével. A párhuzamosító fordítóprogramok fontos programosztályokra felismerik ezeket az átalakítási lehetőségeket.

Gyakorlatok

20.3-1. Tervezzünk MPI programot a 20.2-2 gyakorlatban szereplő prímfeladat megoldására. A program kövesse a „mester/szolga” paradigmát. A program az SPMD stílust vagy az MPMD stílust alkalmazza?

20.3-2. Módosítsuk az előző 20.3-1 gyakorlatban tervezett programot úgy, hogy kollektív kommunikációt alkalmazzon.

20.3-3. Hasonlítsuk össze az MPI-t és az OpenMP-t programozhatóság szem-



20.9. ábra. Párhuzamos közvetlen hozzáférésű gép.

pontjából, azaz soroljunk fel érveket amellett, hogy az egyik, illetve másik eszközzel könnyebb programot írni.

20.3-4. Tervezzünk egy OpenMP programot, amely megvalósítja a 20.2-3 gyakorlatban leírt stencil kódot.

20.4. Számítási modellek

PRAM

A legnépszerűbb számítási modell a párhuzamos közvetlen hozzáférésű gép (PRAM), amely a közvetlen hozzáférésű gép (RAM) természetes általánosítása.

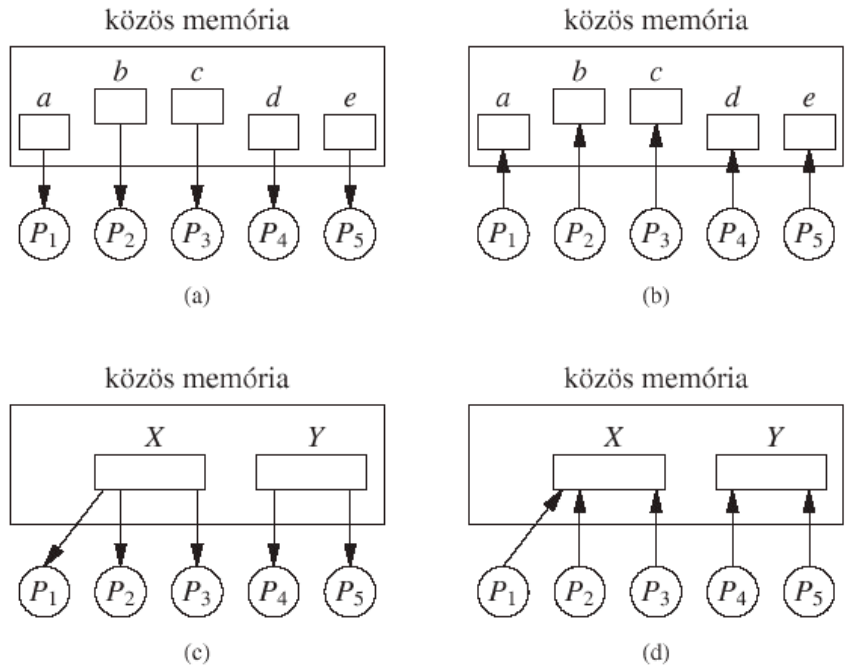
A PRAM modell p szinkronizált processzorból $(P_1, P_2, \ldots, P_p)$, az $M[1]$, $M[2]$, ..., $M[m]$ memóriarekeszeket tartalmazó közös memóriából és a processzorok saját memóriájából áll. A **??**. ábra mutatja a processzorokat és a közös memóriát.

Ennek a modellnek különböző változatai vannak. Ezek a modellek abban különböznek egymástól, hogy a különböző processzorok egyidejűleg hozzáférhetnek-e ugyanahhoz a memóriarekeszhez és hogyan oldódnak meg a konfliktusok. Többek között a következő változatokat különböztetjük meg:

Az ír/olvas műveletek tulajdonságai alapján a következő négy típusuk van:

- EREW (Exclusive-Read Exclusive-Write) PRAM
- ERCW (Exclusive-Read Concurrent-Write) PRAM
- CREW (Concurrent-Read Exclusive-Write) PRAM
- CRCW (Concurrent-Read Concurrent-Write) PRAM

A **??**(a) ábrán minden memóriarekeszhez legfeljebb egy processzor férhet hozzá (ER), a **??**(c) ábrán viszont több is. A **??**(b) ábrán minden memóri-



20.10. ábra. Párhuzamos közvetlen hozzáférésű gép típusai az írás és olvasás alapján.

arekeszbe legfeljebb egy processzor írhat (EW), a **??**(d) ábrán viszont több is (CW).

A párhuzamos írás lehetséges típusai: *közös, prioritásos, tetszőleges, kombinált.*

BSP, LogP és QSM

Ebben a pontban a BSP, LogP és QSM modelleket tárgyaljuk.

A szinkronizált párhuzamos modell (**B**ulk-synchronous **P**arallel **M**odel) a számítógépet olyan csomópontok együtteseként írja le, amelyek processzorból és memóriából állnak. A BSP feltételezi, hogy rendelkezésre áll egy útválasztó és egy akadályszinkronizáló képesség. Az útválasztó továbbítja az üzeneteket a csomópontok között, az akadály szinkronizálja a csomópontokat vagy azok egy részét. A BSP a taszkokat úgynevezett *szuperlépésekre* bontja. Egy szuperlépésben minden processzor számításokat végez a saját memóriájában tárolt adatokkal és kommunikációt kezdeményez a többi processzorral. A kommunikáció garantáltan befejeződik a következő szuperlépés megkezdéséig.

g úgy van definiálva, hogy gh az az idő, amelyet egy h-relációs útválasztás

igényel a szokásos forgalmi terhelés mellett. A h-reláció egy kommunikációs minta, melyben minden processzor legfeljebb h üzenetet küld és fogad.

Egy szuperlépés költsége $x + gh + l$, ahol x az egy processzor által kezdeményezett kommunikáció maximális száma. Egy program költsége az egyes szuperlépések költségeinek összege.

A BSP tartalmaz egy költség modellt, amely három paramétert tartalmaz: a processzorok számát (p), az akadály szinkronizáció költségét (l) és a rendelkezésre álló sávszélességet (g).

A LogP modell létrejöttét a BSP pontatlanságai és a szuperlépések alkalmazásának igénye motiválta.

Míg a LogP a BSP-nél pontosabb, a QSM egyszerűbb. A BSP-vel szemben a QSM közös-memóriájú modell. A BSP-hez hasonlóan a taszkok a QSM szerint is szuperlépésekre vannak felbontva, és minden processzornak van saját helyi memóriája. A processzor minden szuperlépésben számítási műveleteket végez a helyi memóriában tárolt adatokkal, és olvas/ír műveleteket végez a közös memóriával. A közös memóriához való hozzáférést igénylő műveletek befejeződnek a következő szuperlépés megkezdéséig. A QSM modell megengedi a párhuzamos olvasást és írást. Minden memóriarekeszhez szuperlépésenként legfeljebb k hozzáférés lehetséges. A QSM modellben a költségek értéke $\max(x, gh, k)$, ahol g, h és x a BSP modellnél definiált értékek.

Gyakorlatok

20.4-1. A P és Q párhuzamos algoritmusok megoldják a kiválasztási feladatot. A P algoritmus $n^{0.5}$ processzort igényel és a futási ideje $\Theta(n^{0.5})$. A Q algoritmus n processzort igényel és a futási ideje $\lg n$. Határozzuk meg az algoritmusok munkáját, gyorsítását és hatékonyságát. Munkahatékonyak-e ezek az algoritmusok?

20.5. PRAM algoritmusok

Ebben a részben egyszerű feladatokat (mint prefixszámítás, tömbelemek rangsorolása, összefésülés, kiválasztás és rendezés) megoldó párhuzamos algoritmusokat mutatunk. Ezek leírásához kiegészítjük az *Algoritmusok* című könyv új kiadásának második fejezetében leírt pszeudokódot.

P_i **in parallel for** $i \leftarrow 1$ **to** p
⟨1. *utasítás*⟩
⟨2. *utasítás*⟩
.
.
.
⟨*u. utasítás*⟩

Egy k dimenziós rács esetében a hasonló utasítás a

$P_{i_1,i_2,\ldots,i_k}$ **in parallel for** $i_1 \leftarrow 1$ **to** m_1, ..., $i_k \leftarrow 1$ **to** m_k
sorral kezdődik.

20.5.1. Prefixszámítás

Legyen Σ egy alaphalmaz, melyen definiáltuk a $\oplus$ *bináris asszociatív operátort*. Feltesszük, hogy a művelet egy lépéssel elvégezhető és a Σ halmaz zárt erre a műveletre nézve.

Egy $\oplus$ bináris operátor *asszociatív* a Σ alaphalmazon, ha minden $x, y, z \in \Sigma$ esetében teljesül

$$((x \oplus y) \oplus z) = (x \oplus (y \oplus z)) \; . \tag{20.22}$$

Legyenek az $X = x_1, x_2, \ldots, x_p$ sorozat elemei a Σ alaphalmaz elemei. Ekkor a prefixszámítás bemenő adatai az X sorozat elemei, a *prefixszámítási feladat* pedig az x_1, $x_1 \oplus x_2$, ..., $x_1 \oplus x_2 \oplus x_3 \oplus \ldots \oplus x_p$ elemek meghatározása. Ezeket a meghatározandó elemeket ***prefixeknek*** hívjuk.

Érdemes megjegyezni, hogy más területeken inkább az X sorozat $x_1, x_2, \ldots, x_k$ kezdősorozatait hívják prefixeknek.

20.1. példa. *Asszociatív műveletek.* Ha Σ az egészek halmaza, $\oplus$ az összeadás és a bemenő adatok sorozata a 3, -5, 8, 2, 5, 4, akkor a prefixek 3, -2, 6, 8, 13, 17. Ha az ábécé és a bemenő adatok ugyanazok, de a művelet a szorzás, akkor a kimenő adatok (prefixek) $3, -15, -120, -240, -1200, -4800$. Ha a művelet a minimum (ez is asszociatív), akkor a prefixek $3, -5, -5, -5, -5, -5$. Az utolsó prefix megegyezik a bemenő számok legkisebbikével.

A prefixszámítás sorosan megoldható $O(p)$ lépéssel. Bármely A soros algoritmusnak $N(p, \mathrm{A}) = \Omega(n)$ lépésre szüksége van. Vannak olyan párhuzamos algoritmusok, amelyek különböző párhuzamos modelleken megoldják a munkahatékony prefixszámítást.

Először a CREW-PREFIX CREW PRAM algoritmust mutatjuk be, amely p processzoron $\Theta(\lg p)$ lépéssel számítja ki a prefixeket.

Azután az EREW-PREFIX algoritmus következik, amelynek mennyi-

ségi jellemzői hasonlóak az előző algoritmuséhoz, azonban EREW PRAM számítási modellen is végrehajtható.

Ezekkel az algoritmusokkal a prefixszámítást a soros megoldás $\Theta(p)$ futási idejénél lényegesen gyorsabban meg tudjuk oldani, de sok az elvégzett munka.

Ezért is érdekes az Optimális-Prefix CREW PRAM algoritmus, amely $\lceil p/\lg p \rceil$ processzort használ és $O(\lg p)$ lépést tesz. Ennek elvégzett munkája csak $O(p)$, ezért a hatékonysága $\Theta(1)$ és az algoritmus munkahatékony. Ennek az algoritmusnak a gyorsítása $\Theta(n/\lg n)$.

A továbbiakban – a jelölések egyszerűsítése érdekében – a $\lceil p/\lg p \rceil$ típusú kifejezések helyett általában csak $(p/\lg p)$-t írunk.

Az algoritmusok tervezéséhez az oszd-meg-és-uralkodj elvet alkalmazzuk. A bemenő adatok legyenek $X = x_1,\ x_2,\ \ldots,\ x_p$. Az általánosság megszorítása nélkül feltehető, hogy p kettő hatvány.

CREW PRAM algoritmus

Először egy p CREW PRAM processzoros rekurzív algoritmust mutatunk be, melynek futási ideje $\Theta(\lg p)$.

A bemenő adatok a p processzorszám, az $X[1\,..\,p] = x_1,\ x_2, \ldots, x_p$ tömb, a kimenő adat pedig a prefixeket tartalmazó $Y[1\,..\,p] = \langle y_1,\ y_2,\ \ldots,\ y_p \rangle$ tömb.

CREW-prefix(p, X)

```
1  if p = 1
2     y_1 = x_1
3           return Y
4  if p > 1
5  P_i in parallel for i = 1 to p/2
6        az első p/2 processzor rekurzívan számítsa az x_1, x_2, ..., x_{p/2}-höz
         tartozó prefixeket – legyenek ezek y_1, y_2, ..., y_{p/2}
7        P_i in parallel for i = p/2 to p
               számítsa ki az x_{p/2+1}, x_{p/2+2}, ..., x_p-hez
               tartozó prefixeket – legyenek ezek y_{p/2+1}, y_{p/2+2}, ..., y_p
8        P_i in parallel for i = p/2 to p
               olvassák ki a globális memóriából y_{p/2}-t és az
               y_{p/2} ⊕ y_{p/2+1}, y_{p/2} ⊕ y_{p/2+2}, ..., y_{p/2} ⊕ y_p prefixeket
               számítva állítsák elő a végeredmény másik felét.
9        return Y
```

20.2. példa. *8 elem prefixeinek számítása 8 processzoron.* Legyen $n = 8$ és $p = 8$. A prefixszámítás bemenő adatai 12, 3, 6, 8, 11, 4, 5 és 7, az asszociatív művelet az

összeadás. Az első szakaszban az első 4 processzor 12, 3, 6, 8 bemenethez a 12, 15, 21, 29 prefixeket számolja ki. A másik 4 processzor pedig a 11, 4, 5, 7 bemenethez számolja ki a 11, 15, 20, 27 prefixeket.

A második szakaszban az első 4 processzor nem dolgozik, a második 4 pedig 29-et ad minden prefixhez és a 40, 44, 49, 56 eredményt kapja.

20.1. tétel. *A* CREW-PREFIX *algoritmus p* CREW PRAM *processzoron* $\Theta(\lg p)$ *lépésben számítja ki p elem prefixeit.*

Bizonyítás. Az első lépés $W(p/2)$ ideig, a második pedig $O(1)$ ideig tart. Ezért a következő rekurziót kapjuk:

$$W(p) = W\left(\frac{p}{2}\right) + O(1)\ , \tag{20.23}$$

$$W(1) = 1\ . \tag{20.24}$$

Ennek a rekurzív egyenletnek a megoldása $W(p) = O(\lg p)$. ■

Ez az algoritmus nem munkaoptimális, mivel $\Theta(p \lg p)$ munkát végez, és ismert olyan soros algoritmus, amely $O(p)$ lépést tesz - ugyanakkor munkahatékony. Munkahaoptimális algoritmust kaphatunk például úgy, ha a felhasznált processzorok számát lecsökkentjük $(p/\lg p)$-re, miközben a futási idő nagyságrendje ugyanaz marad. A processzorszámot úgy csökkentjük, hogy a bemenet méretét csökkentjük $(p/\lg p)$-re, alkalmazzuk az előző algoritmust, majd végül minden prefixet kiszámolunk.

EREW PRAM algoritmus

A következő algoritmusban a párhuzamos olvasás helyett elég a soros olvasás lehetősége, ezért EREW PRAM modellen is megvalósítható. Bemenő adatai a p processzorszám, az $X[0\,..\,p] = \langle x_0,\ x_1,\ \ldots,\ x_p\rangle$ tömb, kimenő adatai pedig a prefixeket tartalmazó $Y[1\,..\,p] = \langle y_1,\ y_2,\ \ldots,\ y_p\rangle$ tömb.

EREW-PREFIX(p, X)

1 P_i **in parallel for** $i = 1$ **to** p
2 $\quad Y[i] = X[i-1] \oplus X[i]$
3 $k = 2$

```
4  while k < p
5        P_i in parallel for i = 1 to p
6                                  Y[i] = Y[i − k] ⊕ Y[i]
7        k = k + k
8  return Y
```

20.2. tétel. *Az* EREW-PREFIX *algoritmus* p EREW PRAM *processzoron* $\Theta(\lg p)$ *lépésben számítja ki* p *elem prefixeit.*

Bizonyítás. Az 1–3. sorban lévő utasítások $O(1)$ idő alatt végrehajtódnak. A 4–6. sorok annyiszor hajtódnak végre, ahányszor a 7. sorban lévő értékadás, azaz $\Theta(p)$-szer. ■

Munkaoptimális algoritmus

Most egy rekurzív munkaoptimális eljárás következik, amely $p/\lg p$ CREW PRAM processzort használ. A bemenő adatok: p (a bemenő sorozat hossza) és az $X[1\,..\,p]$ tömb, kimenő adat pedig a kiszámított prefixeket tartalmazó $Y[1\,..\,p]$ tömb.

OPTIMÁLIS-PREFIX(p, X)

```
1  P_i in parallel for i = 1 to p/lg p)
2                    rekurzívan számolja a hozzárendelt lg p darab
                     x_{(i−1) lg p+1}, x_{(i−1) lg p+2}, ..., x_{i lg p} elem prefixeit
                     Legyen az eredmény z_{(i−1) lg p+1}, z_{(i−1) lg p+2}, ..., z_{i lg p}.
3  összesen p/lg p processzor alkalmazza együtt a CREW-PREFIX
   algoritmust a p/lg p darab elem, z_{lg p}, z_{2 lg p}, z_{3 lg p}, ..., , z_p prefixeinek
   számítására. Legyen az eredmény w_{lg p}, w_{2 lg p}, w_{3 lg p}, ..., w_p.
4  minden processzor aktualizálja az első lépésben kiszámolt értéket:
   a P_i processzor (i = 2, 3, ..., p/lg p) kiszámolja a
   w_{(i−1) lg p} ⊕ z_{(i−1) lg p+1}, w_{(i−1) lg p} ⊕ z_{(i−1) lg p+2},
   ..., w_{(i−1) lg p} ⊕ z_{i lg p} prefixeket, majd az első processzor
   változtatás nélkül kiadja a z_1, ..., z_{lg p} prefixeket
5  return Y
```

Az algoritmus futási ideje logaritmikus. Ennek belátását megkönnyíti a következő két képlet:

$$z_{(i-1)\lg p+k} = \sum_{j=(i-1)\lg p+1}^{i\lg p} x_j \quad (k = 1,\ 2,\ \ldots,\ \lg p) \tag{20.25}$$

és

$$w_{i\lg p} = \sum_{j=1}^{i} z_{j\lg p} \ (i = 1,\ 2,\ \ldots,) \ , \tag{20.26}$$

ahol az összegzés a megfelelő asszociatív művelet segítségével történik.

20.3. tétel. (párhuzamos prefixszámítás $\Theta(\lg p)$ lépéssel). *A* HATÉKONY-PREFIX *algoritmus* $p/\lg p$ CREW PRAM *processzoron* $\Theta(\lg p)$ *lépéssel számítja ki* p *elem prefixeit.*

Bizonyítás. Az algoritmus első szakasza $O(\lg p)$ ideig, a második szakasza a 20.1. tétel szerint $\Theta(\lg(p/\lg p)) = \Theta(\lg p)$ lépésig tart. Végül a harmadik szakasz ugyancsak $O(\lg p)$ lépést igényel. ■

A tételből következik, hogy az OPTIMÁLIS-PREFIX algoritmus munkahatékony.

20.3. példa. *16 elem összeadása.* Legyen 16 elemünk: 4, 10, 5, 2, 3, 9, 11, 12, 1, 5, 6, 7, 10, 4, 3, 5. Az asszociatív művelet az összeadás. Ekkor 16 elem $\lg 16 = 4$ processzort igényel. A processzorok először 4-4 elem prefixeit számolják – sorosan. A második lépésben a helyi összegekből a 4 processzor együtt globális összegeket számol, majd azokkal a harmadik lépésben a processzorok ismét külön sorosan frissítik a helyi eredményeket. A számítás menetét mutatja a 20.11. ábra.

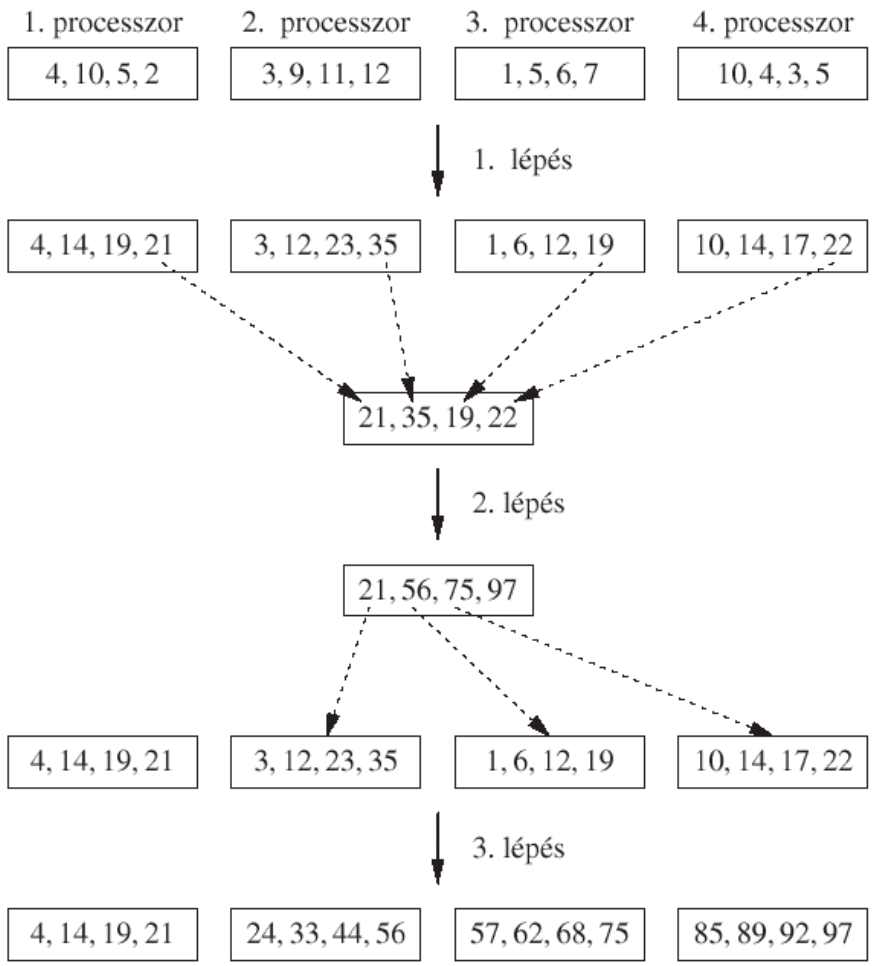
20.5.2. Tömb elemeinek rangsorolása

A *tömbrangsorolási probléma* bemenő adata egy p elemű tömbben ábrázolt lista: minden elem tartalmazza jobb oldali szomszédjának az indexét (és esetleges további adatokat). A feladat az elemek ***rangjának*** (jobb oldali szomszédai számának) meghatározása.

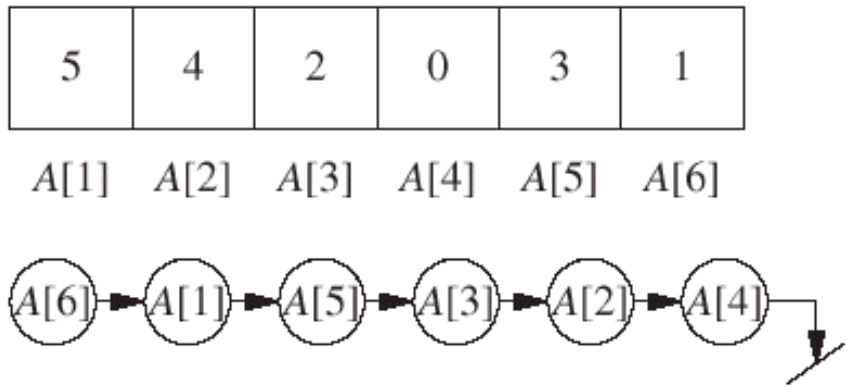
Mivel az adatokra nincs szükség a megoldáshoz, feltesszük, hogy az elemek csak a szomszéd indexét tartalmazzák. A jobb szélső elem index mezője nulla. Az indexet a továbbiakban mutatónak hívjuk.

20.4. példa. *Tömbrangsorolás bemenő adatai.* Legyen $A[1\,..\,6]$ a 20.12. ábra felső sorában bemutatott tömb. Ekkor az $A[1]$ elem jobb oldali szomszédja $A[5]$, $A[2]$ jobb oldali szomszédja $A[4]$. $A[4]$ az utolsó elem, ezért rangja 0. $A[2]$ rangja 1, mivel csak $A[4]$ van tőle jobbra. $A[5]$ rangja 3, mivel az $A[3], A[2]$ és $A[4]$ elemek vannak tőle jobbra. Az elemek sorrendjét (balról jobbra haladva) mutatja az ábra alsó része.

A tömbrangsorolás sorosan elvégezhető lineáris idő alatt. Először meghatározzuk a *tömb fejét* – az egyetlen olyan i értéket $(1 \le i \le p)$, melyre $A[j] \neq i$ teljesül minden $1 \le j \le n$ értékre. Legyen $A[i]$ a tömb feje. A fejtől kiindulva pásztázzuk a tömböt és az elemekhez rendre hozzárendeljük a



20.11. ábra. 16 elem prefixeinek számítása az Optimális-prefix algoritmussal.



20.12. ábra. Tömbrangsorolási probléma bemenő adatai és a megfelelő tömb.

$p, p-1, \ldots, 1, 0$ rangokat.

Ebben a részben a DET-RANGSOROL algoritmust mutatjuk be, amely egy p processzoros EREW PRAM algoritmus $\Theta(O(\lg p))$ futási idővel. Ennek gyorsítása $\Theta(p/\lg n)$, hatékonysága $\Theta(p)/\Theta(p \lg p) = 1/\lg p)$, azaz nem munkahatékony.

Determinisztikus tömbrangsorolás

Ebben az algoritmusban az egyik alapvető ötlet a *mutatóugrás*. DET-RANGSOROL szerint először mindegyik elem a jobb oldali szomszédjának indexét tartalmazza, és ennek megfelelően a rangja – a jobb oldali szomszédjához viszonyítva – 1 (kivétel a lista utolsó eleme, melynek rangja 0. Ezt a kezdeti állapotot mutatja a 20.12. ábra első sora.

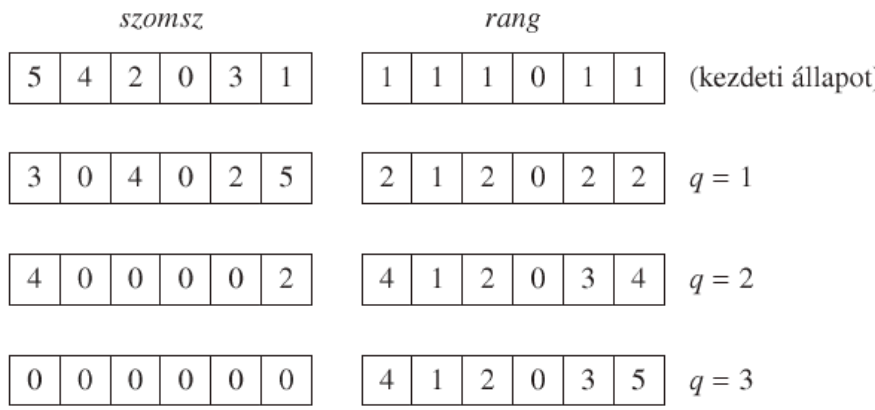
Ezután módosítjuk a csúcsokat úgy, hogy mindegyik a jobb oldali szomszédjának a jobb oldali szomszédjára mutasson (ha nincs, akkor a lista végére). Ezt tükrözi a 20.13. ábra második sora.

Ha p processzorunk van, akkor ez $O(1)$ lépéssel elvégezhető.

Most minden csúcs (kivéve az utolsót) olyan csúcsra mutat, amelyik eredetileg 2 távolságra volt. A mutatóugrás következő lépésében a csúcsok olyan csúcsra mutatnak, amelyek eredetileg 4 távolságra voltak tőlük (ha ilyen csúcs nincs, akkor a lista végére) – amint azt az ábra harmadik sora mutatja.

A következő lépésben a csúcsok (pontosabban a mutató részük) a 8 távolságú szomszédra mutatnak (ha van ilyen – ha nincs, akkor a lista végére), a 20.13. ábra utolsó sora szerint.

Minden csúcs minden lépésben információt gyűjt arról, hány csúcs van



20.13. ábra. A DET-RENDEZ algoritmus működése a 20.4 példa adataival.

közte és azon csúcs között, amelyre most mutat. Ehhez kezdetben legyen a csúcsok rang mezőjében 1 – kivéve a jobb oldali csúcsot, melyre ez az érték legyen 0. Legyen $R[i]$ és $N[i]$ az i csúcs rang, illetve szomszéd mezője. A mutatóugrás során $R[i]$-t általában $R[i] + R[N[i]]$-re módosítjuk – kivéve azokat a csúcsokat, melyekre $N[i] = 0$. Ezután $N[i]$-t úgy módosítjuk, hogy $N[N[i]]$-re mutasson.

Az algoritmus bemenő adatai a rangsorolandó elemek p száma, az elemek jobb oldali szomszédainak indexeit tartalmazó *szomsz*$[1\,..\,p]$ tömb, kimenő adatai pedig a meghatározott rangokat tartalmazó *rang*$[1\,..\,p]$ tömb.

DET-RANGSOROL pszeudokódja a következő.

DET-RANGSOROL(p, N, R)

```
1  P_i in parallel for i = 1 to n
2                          if  N[i] = 0
3                              R[i] = 0
4                          else R[i] = 1
5  for i = 1 to ⌈lg n⌉
6      P_i in parallel for  1 ≤ i ≤ n
7                              if N[i] ≠ 0
8                                 R[i] = R[i] + R[N[i]]
9                                 N[i] = N[N[i]]
10 return rang
```

A 20.13. ábra mutatja, hogyan működik DET-RANGSOROL a 20.4. példa

adataival.

Kezdetben minden csúcs rangja 1, kivéve a 4. csúcsot. Amikor $q = 1$, akkor például az 1. csúcs R mezőjét kettőre változtatjuk, mert jobb oldali szomszédjának (ez az 5. csúcs) rangja 1. Az 1. csúcs N mezőjét az 5. csúcs szomszédjának indexére, azaz 3-ra változtatjuk.

20.4. tétel. *A* Det-rangsorol *algoritmus egy p processzoros* EREW PRAM *modellen $\Theta(\lg p)$ lépésben határozza meg egy p elemű tömb elemeinek rangját.*

Mivel a A Det-rangsorol algoritmus $\Omega(p \lg p)$ munkát végez, ezért nem munkaoptimális – de munkahatékony.

A listarangsorolási probléma megfelel a lista prefix összege számításának, ahol minden csúcs súlya 1, kivéve a jobb oldalit, melynek súlya 0. A Det-rendez algoritmus könnyen módosítható úgy, hogy kiszámítsa egy lista prefixeit – a processzorszámra és a futási időre vonatkozó hasonló korlátokkal.

20.5.3. Összefésülés

Adott 2 csökkenőleg (vagy növekvőleg) rendezett sorozat, melyek együtt p elemet tartalmaznak. A feladat ennek a sorozatnak egy csökkenő (vagy növekvő) sorozattá való rendezése.

Ez a feladat egy soros processzoron $O(p)$ lépéssel megoldható. Mivel legrosszabb esetben minden elemet meg kell vizsgálni és a helyére kell tenni, ezért a feladat megoldásának időigénye $\Omega(p)$.

Összefésülés logaritmikus időben

Legyen $X_1 = \langle k_1, k_2, \ldots, k_m \rangle$ és $X_2 = \langle k_{m+1}, k_{m+2}, \ldots, k_{2m} \rangle$ a két bemenő sorozat. Az egyszerűség kedvéért legyen m kettő hatvány és a kulcsok különbözzenek.

Az összefésüléshez elég az összes kulcs rangjának kiszámítása. Ha a rangokat ismerjük, akkor $p = 2m$ processzoron egy lépésben beírhatjuk az i rangú kulcsot az i-edik memóriarekeszbe.

20.5. tétel. *A* Log-összefésül *algoritmus két m hosszúságú kulcssorozatot $O(\lg m)$ lépéssel összefésül $2m$* CREW PRAM *processzoron.*

Bizonyítás. A k kulcs rangja legyen r_k^1 (r_k^2) X_1-ben (X_2-ben). Ha $k = k_j \in X_1$, akkor legyen $r_k^1 = j$. Ha egy külön π processzort rendelünk k-hoz, akkor az bináris kiválasztással $O(\lg m)$ lépéssel meghatározza azon X_2-beli elemek q számát, amelyek kisebbek, mint k. Ha q ismert, akkor π kiszámíthatja k ($X_1 \cup X_2$)-beli rangját: ez $j+q$ lesz. Ha k X_2-höz tartozik, hasonlóképpen járhatunk

el.

Összegezve: ha elemenként egy, azaz összesen $2m$ processzorunk van, akkor két m hosszúságú rendezett sorozat $O(\lg m)$ lépéssel összefésülhető. Az ezt megoldó algoritmus neve LOG-ÖSSZEFÉSÜL. ■

Ez az algoritmus nem munkaoptimális, de munkahatékony.

Páratlan-páros összefésülő algoritmus

Ez a rekurzív algoritmus a klasszikus oszd-meg-és-uralkodj elvet alkalmazza.

Legyen $X_1 = \langle k_1, k_2, \ldots, k_m \rangle$ és $X_2 = \langle k_{m+1}, k_{m+2}, \ldots, k_{2m} \rangle$ a két bemenő tömb. Az egyszerűség kedvéért legyen m kettő hatvány és a kulcsok legyenek különbözőek. Az algoritmus $2m$ EREW PRAM processzort igényel, a kimenő adat az összefésült elemeket tartalmazó $Y[1\,..\,2m]$ tömb.

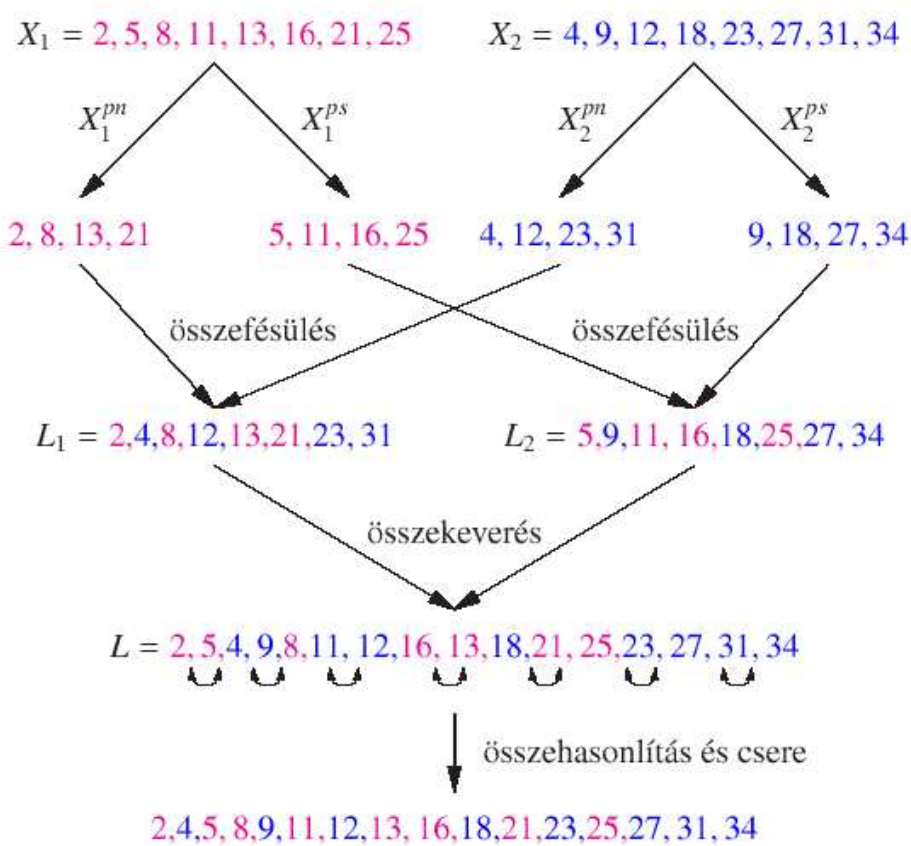
PÁRATLAN-PÁROS-ÖSSZEFÉSÜL(X_1, X_2)

```
1  if m = 1
2     fésüljük össze a sorozatokat egyetlen összehasonlítással
3     return Y
4  if m > 1
5     bontsuk fel X_1-et és X_2-t páros és páratlan részre, azaz legyen
      X_1^ptn = k_1, k_3, ..., k_{m-1} és X_1^prs = k_2, k_4, ..., k_m
6     hasonlóképpen bontsuk fel X_2-t is X_2^ptn és X_2^prs részekre
7     rekurzívan fésüljük össze X_1^ptn-t és X_2^ptn-t m processzoron
      legyen az eredmény L_1 = l_1, l_2, ..., l_m. Ugyanakkor fésüljük össze
      X_1^prs-t és X_2^prs-t másik m processzoron: az eredmény
      legyen L_2 = l_{m+1}, l_{m+2}, ..., l_{2m}.
8     keverjük össze az L_1 és L_2 sorozatokat, azaz legyen
      L = l_1, l_{m+1}, l_2, l_{m+2}, ..., l_m, l_{2m}.
9     hasonlítsuk össze az (l_{m+i}, l_{i+1}) (i = 1, 2, ..., m - 1) párokat és
      szükség esetén cseréljük fel őket. Az eredmény lesz a kimenő sorozat.
10 return Y
```

20.5. példa. *Kétszer nyolc szám összefésülése.* Legyen $X_1 = 2, 5, 8, 11, 13, 16, 21, 25$ és $X_2 = 4, 9, 12, 18, 23, 27, 31, 34$. A 16 szám rendezését mutatja a következő 20.14. ábra.

20.6. tétel. (összefésülés $O(\lg m)$ lépéssel). *A* PÁRATLAN-PÁROS-ÖSSZEFÉSÜL *algoritmus két m hosszúságú kulcssorozatot $2m$* EREW PRAM *processzoron $\Theta(\lg m)$ idő alatt fésül össze.*

Bizonyítás. Legyen az algoritmus futási ideje $W(n)$. Az 1. lépés $O(1)$ ideig



20.14. ábra. 16 szám rendezése a PÁRATLAN-PÁROS-ÖSSZEFÉSÜL algoritmussal.

tart. A 2. lépés $m/2$ ideig tart. Innen a

$$W(m) = W\left(\frac{m}{2}\right) + O(1) \tag{20.27}$$

rekurzív egyenlőséget kapjuk, melynek megoldása $W(m) = O(\lg m)$. ■

Ennek az algoritmusnak a helyességét a ***nulla-egy elv*** segítségével bizonyítjuk.

Egy összehasonlítás alapú rendező algoritmus ***egyszerű,*** ha az összehasonlítandó elemek sorozata előre meg van határozva (ekkor a következő összehasonlítás elemei nem függnek a mostani eredménytől).

Formálisan ez azt jelenti, hogy adott az összehasonlítandó elempárok indexeinek (i_1, j_1), (i_2, j_2), $\ldots$, (i_m, j_m) sorozata.

20.7. tétel. (nulla-egy elv). *Ha egy egyszerű összehasonlításos rendező algoritmus helyesen rendez egy n hosszúságú nulla-egy sorozatot, akkor tetszőleges kulcsokból álló n hosszúságú sorozatot is helyesen rendez.*

Bizonyítás. Legyen A egy egyszerű összehasonlításos (növekvőleg) rendező algoritmus és legyen S egy olyan kulcssorozat, melyet az adott algoritmus rosszul rendez. Ekkor a rosszul rendezett S sorozatban van olyan kulcs, amely az i-edik $(1 \leq i \leq n-1)$ helyen van annak ellenére, hogy S-ben legalább i nála kisebb elem van.

Legyen k S legelső (legkisebb indexű) ilyen kulcsa. A bemenő sorozatban írjunk a k-nál kisebb elemek helyére nullát, a többi elem helyére egyest. Ezt a módosított 0–1 sorozatot A helyesen rendezi, ezért a k helyére írt egyest a rendezett sorozatban legalább i darab nulla megelőzi.

Most kihasználjuk, hogy A egyszerű. A bemenő sorozatban színezzük pirosra a k-nál kisebb (nulla) elemeket, és kékre a többit (egyeseket). Indukcióval megmutatjuk, hogy az eredeti és a 0-1 sorozatnak megfelelő színes sorozatok minden összehasonlítás után azonosak. A színek szerint háromféle összehasonlítás van: kék, piros vagy különböző színű elemek összehasonlítása. Ha azonos színű elemeket hasonlítunk össze, akkor a színek sorozata egyik esetben sem változik. Ha viszont különböző színű elemeket hasonlítunk össze, akkor mindkét esetben a piros elem kerül a kisebb, és a kék elem a nagyobb indexű helyre. Eszerint k-t legalább i nála kisebb elem megelőzi a rendezett sorozatban. Az ellentmondás az állítás helyességét mutatja. ■

20.6. példa. *Egy nem összehasonlításos rendező algoritmus.* Legyen $k_1, k_2, \ldots, k_n$ egy bitsorozat. Rendezhetjük úgy, hogy megszámoljuk a nullák z számát, majd leírunk előbb z nullát, majd $n - z$ egyest. Erre az elv nem alkalmazható, mert ez nem összehasonlításos rendezés.

Az összefésülés viszont rendezés, és a páros-páratlan összefésülés egyszerű.

20.8. tétel. *A* Páratlan-páros-összefésül *algoritmus helyesen rendez tetszőleges számokból álló sorozatokat.*

Bizonyítás. Legyenek X_1 és X_2 rendezett 0-1 sorozatok, melyek közös hossza m. Legyen q_1 (q_2) az X_1 (X_2) elején álló nullák száma. Az X_1^{ptn}-ban lévő nullák száma $\lceil q_1/2 \rceil$, és az X_1^{prs}-ban lévő nullák száma $\lfloor q_1/2 \rfloor$. Így az L_1-beli nullák száma $z_1 = \lceil q_1/2 \rceil + \lceil q_2/2 \rceil$ és az L_2-beli nullák száma $z_2 = \lfloor q_1/2 \rfloor + \lfloor q_2/2 \rfloor$.

z_1 és z_2 különbsége legfeljebb 2. Ez a különbség pontosan akkor kettő, ha q_1 és q_2 is páratlan. Egyébként a különbség legfeljebb 1. Tegyük fel, hogy $|z_1 - z_2| = 2$ (a többi eset hasonló). Most L_1-ben kettővel több nulla van. Amikor ezeket a harmadik lépésben összekeverjük, akkor L elején nullák vannak, azután 1,0, majd egyesek. A rendezetlen (*piszkos*) rész csak az 1,0. Amikor a harmadik lépésben az utolsó összehasonlítás és csere megtörténik, az egész sorozat rendezetté válik. ∎

Munkaoptimális algoritmus

Most $\lceil 2m/\lg m \rceil$ processzoron $O(\lg m)$ lépéssel végezzük az összefésülést. Ez az Optimálisan-összefésül algoritmus az eredeti problémát $O(m/\lg m)$ részre osztja úgy, hogy mindegyikben $O(\lg m)$ hosszúságú rendezett sorozatokat kell összefésülni. Ezek a részfeladatok soros algoritmussal $O(\lg m)$ lépéssel megoldhatók.

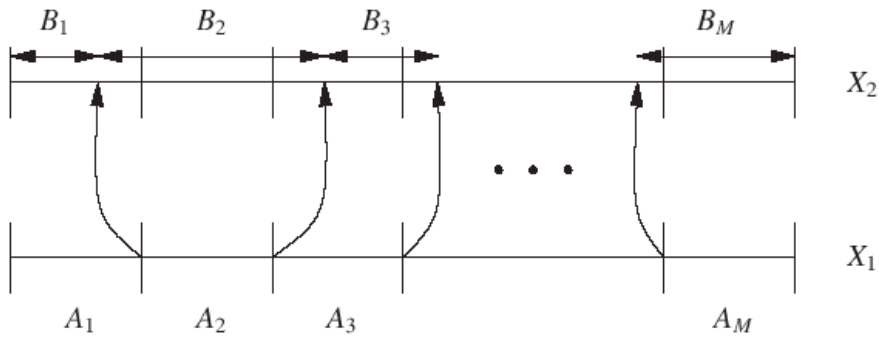
Legyen $X_1 = x_1, x_2, \ldots, x_m$ és $X_2 = x_{m+1}, x_{m+2}, \ldots, x_{m+m}$ a két bemenő sorozat. Osszuk X_1-et $\lceil m/\lg m \rceil$ részre: ekkor mindegyikben legfeljebb $\lceil \lg m \rceil$ kulcs lesz. A részek legyenek $A_1, \ldots, A_M$, ahol $M = m/\lg m$. Az A_1-beli legnagyobb kulcs legyen l_i ($i = 1, 2, \ldots, M$). Rendeljünk egy-egy processzort ezekhez az l_i elemekhez. Ezek a processzorok bináris kiválasztással meghatározzák l_i X_2-beli (rendezés szerinti) helyét. Ezek a helyek felbontják X_2-t M részre (ezek között üres részek is lehetnek – lásd a következő 16.3. ábrát). Jelöljük ezeket a részeket B_1, B_2, …, B_M-mel. B_i-t az A_1-nek X_2-ben megfelelő részhalmaznak nevezzük.

Ekkor X_1 és X_2 összefésülését megkaphatjuk úgy, hogy rendre összefésüljük A_1-et B_1-gyel, A_2-t B_2-vel és így tovább, majd ezeket a sorozatokat egyesítjük Lásd a **??**. ábrát).

20.9. tétel. *A* Hatékonyan-összefésül *két m hosszúságú rendezett kulcssorozatot $O(\lg m)$ lépésben összefésül $\lceil 2m/\lg m \rceil$* CREW PRAM *processzoron.*

Bizonyítás. Az előző algoritmust alkalmazzuk.

Az A_i részek hossza $\lg m$, a B_i részek hossza azonban nagy is lehet. Ezért még egyszer alkalmazzuk a felbontást. Legyen A_i, B_i tetszőleges pár. Ha



20.15. ábra. Munkaoptimális összefésülő algoritmus.

$|B_i| = O(\lg m)$, akkor A_i és B_i egy processzoron $O(\lg m)$ lépésben összefésülhető. Ha viszont $|B_i| = \omega(\lg m)$, akkor osszuk B_i-t $|B_i|/\lg m$ részre – ekkor minden rész legfeljebb $\lg m$ egymást követő kulcsot tartalmaz. Mindegyik részhez rendeljünk egy processzort, és az keresse meg az ennek a sorozatnak megfelelő részhalmazt A_i-ben: ehhez $O(\lg\lg m)$ lépés elegendő. Így A_i és B_i összefésülése $|B_i|/\lg m$ részproblémára redukálható, ahol minden részprobléma két $O(\lg m)$ hosszúságú sorozat összefésülése.

A felhasznált processzorok száma $\sum_{i=1}^{M} \lceil |B_i|/\lg m \rceil$, ami legfeljebb $m/\lg m + M$, és ez legfeljebb $2M$. ■

Összefésülés $O(\lg\lg m)$ idő alatt

Ha az előző algoritmust kiegészítjük az oszd-meg-és-uralkodj elvvel, akkor még gyorsabb algoritmust kapunk. A bemenő és kimenő adatok hasonlók az előző algoritmus adataihoz. Feltesszük, hogy m négyzetszám.

GYORSAN-ÖSSZEFÉSÜL(X_1, X_2)

1 bontsuk fel X_1-et $\sqrt{m} = b$ egyenlő részre – legyenek ezek $A_1, A_2, \ldots, A_b$.
2 legyen A_i-ben a legnagyobb kulcs l_i $(i = 1, 2, \ldots, b)$. Minden l_i-hez rendeljünk b processzort.
3 ezek a processzorok végezzenek b-áris keresést X_2-ben, hogy megtalálják l_i X_2-beli helyét.
4 ezzel X_2 b részre való felbontását kapjuk: legyenek ezek a részek $B_1, B_2, \ldots, B_b$ A B_i részhalmaz az A_i-nek X_2-ben megfelelő részhalmaz.

5 most X_1 és X_2 összefésüléséhez elegendő A_i és B_i $(i = 1, 2, \ldots, b)$ összefésülése. Az A_i-k mérete adott, viszont a B_i-k nagyon nagyok (és nagyon kicsik) is lehetnek. Ezért újra felbontunk.
6 **return** Y

Legyen A_i és B_i tetszőleges pár. Ha $|B_i| = O(b)$, akkor a két sorozat $O(1)$ lépésben összefésülhető m^ϵ-áris kereséssel (ahol ϵ tetszőleges pozitív szám). Ha viszont $|B_i| = \omega(b)$, akkor B_i-t $\lceil |B_i|/b \rceil$ részre osztjuk, ahol B_i-nek minden részben legfeljebb b egymást követő eleme van. Rendeljünk minden részhez b processzort, hogy megtalálják az ehhez a halmazhoz tartozó részhalmazt A_i-ben: ehhez $O(1)$ lépés elég. Így A_i és B_i összefésülésének problémája $\lceil |B_i|/b \rceil$ részproblémára redukálható, ahol minden részprobléma két $O(b)$ hosszúságú sorozat összefésülése.

A felhasznált processzorok száma $\sum_{i=1}^{b} \lceil b|B_i|/b \rceil$, ami legfeljebb $2m$.

20.10. tétel. (összefésülés $O(\lg \lg m)$ lépésben). *Két m hosszúságú rendezett kulcssorozat $O(\lg \lg m)$ lépésben összefésülhető $2m$ CREW PRAM processzoron.*

Bizonyítás. Legyen X_1 és X_2 a két adott sorozat. Legyenek a kulcsok különbözők és legyen $\sqrt{(m)} = b$. Az algoritmus a problémát $N \leq 2b$ részproblémára redukálja, melyek mindegyike két $O(b)$ hosszúságú rendezett sorozat összefésülése. A redukció m processzoron $O(1)$ lépésben elvégezhető. Ha az algoritmus futási ideje $2m$ processzoron $T(m)$, akkor $T(m)$ kielégíti a

$$T(m) = T(O(b)) + O(1)$$

rekurzív egyenletet, melynek megoldása $O(\lg \lg m)$. ∎

Ez az algoritmus munkahatékony, de nem munkaoptimális, bár $\Theta(m)/\Theta(\lg \lg m) = \Theta(m/\lg \lg m)$ relatív sebessége nagyon közel van m-hez. Hatékonysága csak $\Theta(1/\lg \lg m)$.

20.5.4. Munkaoptimális algoritmusok elemzése

Ebben az alfejezetben két munkaoptimális prefixszámító algoritmust mutatunk be.

1. algoritmus: Optimális-prefix

Tegyük fel, hogy $n/\lg n$ (továbbiakban jelölésben $n/\lg n$) processzorunk van. Az algoritmus három fő szakaszból áll:

1. szakasz. Az X bemenetet $n/\lg n$ részre osztjuk, majd $n/\lg n$ processzor kiszámolja a hozzárendelt $X_{(i-1)\lg n+1}, X_{(i-1)\lg n+2}, \ldots, X_{(i)\lg n}$ elem prefixeit rekurzívan. Az eredmény legyen $Z_{(i-1)\lg n+1}, Z_{(i-1)\lg n+2}, \ldots, Z_{i\lg n)}$. Ez $O(\lg n)$ lépés, gondoljunk csak a soros algoritmusra.

2. szakasz. $n/\lg n$ processzor együttesen alkalmazza a CREW-PREFIX-et segéd-algoritmusként az $n/\lg n$ elem $Z_{\lg n}, Z_{2\lg n}, \ldots, Z_n$ prefixeinek számítására, ami az alábbi:

Ez a szakasz két lépésből áll.

Első lépés: Az első $n/2\lg n$ processzor rekurzívan kiszámítja az első $(Z_{\lg n}, Z_{2\lg n}, \ldots, Z_{n/2})$-hez tartozó prefixet. Az eredmény $Y_{\lg n}, Y_{2\lg n}, \ldots, Y_{n/2}$ lesz. Továbbá a második $n/2\lg n$ processzor rekurzívan kiszámítja a $(Z_{n/2+\lg n}, Z_{(n/2)+2\lg n}, \ldots, Z_n)$-hez tartozó prefixet. Az eredmény $Y_{n/2+\lg n}, Y_{(n/2)+2\lg n}, \ldots, Y_n$ lesz.

Második lépés: A második $n/2\lg n$ processzor párhuzamosan kiolvassa a globális memóriából $Y_{n/2}$-t, és $Y_{n/2}+Y_{n/2+\lg n}, Y_{n/2}+Y_{(n/2)+2\lg n}, \ldots, Y_{n/2}+Y_n$ prefixeket számítva előállítja a végeredmény második felét.

A CREW-PREFIX algoritmus műveletigénye n bemenő adat esetén: $T(n) = O(\lg n)$. Mivel nekünk ebben a részben nem n, hanem csak $n/\lg n$ processzorunk van, és $n/\lg n$ elemünk, ezért a 2. lépésre is igaz az $O(\lg n)$ futási idő, ha behelyettesítjük n helyére $n/\lg n$-t:

$$O(\lg(n/\lg n)) = O(\lg(n(1/\lg n))) = O(\lg n + \lg(1/\lg n)) = O(\lg n)\ ,$$

és ennyi elég is a munkahatékonyság belátásához.

Ennek a lépésnek az eredménye legyen $Y_{\lg n}, Z_{2\lg n}, \ldots, Y_n$.

3. szakasz. Minden processzor aktualizálja az első lépésben kiszámolt értéket: a $P_i(i = 2, 3, \ldots, n/\lg n)$ processzor kiszámolja az $Y_{(i-1)\lg n} + Z_{(i-1)\lg n+1}, Y_{(i-1)\lg n} + Z_{(i-1)\lg n+2} \ldots, Y_{(i-1)\lg n} + Z_{(i)\lg n}$ prefixeket, majd az első processzor kiadja a $Z_1, Z_2, \ldots, Z_{\lg n}$ prefixeket.

OPTIMÁLIS-PREFIX(n, X) párhuzamos rekurzív eljárás, amely a CREW PRAM számítási modellt alkalmazza. Bemenete n (a bemenő sorozat hossza) és $X[1\,..\,n] = \langle x_1, x_2, \ldots, x_n\rangle$ (n hosszúságú sorozat), kimenete pedig $Y[1\,..\,n] = \langle y_1, y_2, \ldots, y_n\rangle$ (n hosszúságú sorozat, melynek elemei a prefixek).

OPTIMÁLIS-PREFIX(X)

```
1  P_i in parallel for i = 1 to n/lg n
2        Z[(i − 1) lg n + 1] = X[(i − 1) lg n + 1]
3  for j = (i − 1) lg n + 2 to i lg n
4       Z[j] = Z[j − 1]X[i]
```

```
5  P_i in parallel for i = 1 to n/lg n)
6        T[i] = Z[i lg n]
7  CREW-PREFIX(n/lg n, T, Y)
8  P_i in parallel for i = 2 to n/lg n)
9        for j ← (i − 1) lg n + 1 to i lg n
10           Z[j] ← Z[j]Y[(i − 1) lg n]
11 return Y
```

20.11. lemma. *Az* Optimális-prefix *algoritmus $n/\lg n$ CREW PRAM processzoron $O(\lg n)$ lépésben munkaoptimálisan számítja ki n elem prefixeit.*

Bizonyítás. Az algoritmus futási ideje: az első rész $O(\lg n)$ műveletigényű. a második rész $O(\lg n)$ műveletigényű. a harmadik rész $O(\lg n)$ műveletigényű. Összesen: $O(\lg n) + O(\lg n) + O(\lg n) = O(\lg n)$.

Mind a három rész $n/\lg n$ processzort használ. ∎

2. algoritmus: Optimális-prefix'

Ez az algoritmus hasonló az előzőhöz, de most csak $\lg n$ processzort használunk. A célunk az, hogy megmutassuk, felcserélhetők egymással az Optimális-prefix algoritmusban szereplő processzorszám és futási idő értékek.

1. szakasz. Az X bemenetet $\lg n$ részre osztjuk, majd $\lg n$ processzor rekurzívan kiszámolja a hozzárendelt $X_{(i-1)(n/\lg n)+1}$, $X_{(i-1)(n/\lg n)+2}, \ldots, X_{(i)(n/\lg n)}$ elem prefixeit rekurzívan. Az eredmény legyen $Z_{(i-1)(n/\lg n)+1}, Z_{(i-1)(n/\lg n)+2}, \ldots, Z_{(i)(n/\lg n)}$. Most $\lg n$ részre osztottuk a bemenetet, így egy rész hossza $n/\lg n$, tehát itt is a soros algoritmusra hivatkozva ennek a résznek a futási ideje: $O(n/\lg n)$.

2. szakasz. $\lg n$ processzor együttesen alkalmazza a CREW-prefix-et segédalgoritmusként a $\lg n$ elem $Z_{n/\lg n}, Z_{2(n/\lg n)}, \ldots, Z_n$ prefixeinek számítására, ami az alábbi:

Ez a szakasz két lépésből áll.

Első lépés: Az első $\lg n/2$ processzor rekurzívan kiszámítja az első $Z_{n/\lg n}, Z_{2(n/\lg n)}, \ldots, Z_{n/2}$-hez tartozó prefixet. Az eredmény $Y_{n/\lg n}, Y_{2(n/\lg n)}, \ldots, Y_{n/2}$ lesz. Továbbá a második $\lg n/2$ processzor rekurzívan kiszámítja a $Z_{n/2+n/\lg n}$, $Z_{(n/2)+2(n/\lg n)}, \ldots, Z_n$-hez tartozó prefixet. Az eredmény $Y_{n/2+n/\lg n}, Y_{(n/2)+2(n/\lg n)}, \ldots, Y_n$ lesz.

Második lépés: A második $(\lg n)/2$ processzor párhuzamosan kiolvassa a globális memóriából $Y_{n/2}$-t, és $Y_{n/2} + Y_{n/2+n/\lg n}, Y_{n/2} + Y_{(n/2)+2(n/\lg n)}, \ldots, Y_{n/2}+Y_n$ prefixeket számítva előállítja a végeredmény második felét.

A CREW-PREFIX eljárást itt is segédalgoritmusként használtuk $\lg n$ méretű bemenetre, így most az alábbiak szerint módosul a futási ideje:

A 2. lépés $O(\lg n)$ futási ideje az alábbiak alapján kapható meg: behelyettesítjük n helyére $\lg n$-t: $O(\lg(\lg n)) = O(\lg n)$. Ennek a lépésnek az eredménye legyen $Y \lg n, Z_{2 \lg n}, \ldots, Y_n$.

3. szakasz. Minden processzor aktualizálja az első lépésben kiszámolt értéket: a P_i $(i = 2, 3, \ldots, \lg n)$ processzor kiszámolja az $Y_{(i-1)(n/\lg n)} + Z_{(i-1)(n/\lg n)+1}, Y_{(i-1)(n/\lg n)} + Z_{(i-1)(n/\lg n)+2}, \ldots, Y_{(i-1)(n/\lg n)} + Z_{(i)(n/\lg n)}$ prefixeket, majd az első processzor kiadja a $Z_1, Z_2, \ldots, Z_{\lg n}$ prefixeket.

HATÉKONY-PREFIX2(n, X) párhuzamos rekurzív eljárás, amely CREW PRAM számítási modellen alapul. Bemenete n (a bemenő sorozat hossza) és $X[1\,..\,n] = \langle x_1, \ldots, x_n \rangle$ (n hosszúságú sorozat), kimenete pedig $Y[1\,..\,n] = \langle y_1, \ldots, y_n \rangle$ (n hosszúságú sorozat, melynek elemei a prefixek).

Az algoritmus pszeudokódja a következő.

OPTIMÁLIS-PREFIX'(X)

```
1  P_i in parallel for i ← 1 to lg n
2        Z[(i − 1)(n/ lg n) + 1] ← X[(i − 1)(n/ lg n) + 1]
3        for j ← (i − 1)(n/ lg n) + 2 to i lg n
4            Z[j] ← Z[j − 1]X[i]
5  P_i in parallel for i ← 1 to lg n
6        T[i] ← Z[i(n/ lg n)]
7        textscCREW-prefix(lg n, T, Y)
8  P_i in parallel for i ← 2 to lg n
9        for j ← (i − 1)(n/ lg n) + 1 to in/ lg n
10           Z[j] ← Z[j]Y[(i − 1)(n/ lg n)]
11 return Y
```

20.12. lemma. *Az* OPTIMÁLIS-PREFIX' *algoritmus* $\lg n$ CREW PRAM *processzoron* $O(n/\lg n)$ *lépésben munkaoptimálisan számítja ki* n *elem prefixeit.*

Bizonyítás. Az algoritmus futási ideje: az első rész $O(n/\lg n)$ műveletigényű, a második rész $O(\lg n)$ műveletigényű, a harmadik rész $O(n/\lg n)$ műveletigényű. Összesen: $O(n/\lg n) + O(\lg n) + O(n/\lg n) = O(n/\lg n)$.

Mind a három rész $\lg n$ processzort használ. ∎

A MUNKAOPTIMÁLIS-PREFIX algoritmus nemcsak munkaoptimális, hanem aszimptotikusan optimális is, ezt bizonyítja a következő tétel:

20.13. tétel. *Párhuzamos prefixszámításhoz* $\Omega(\lg n)$ *számú* $\oplus$ *műveletet kell végezni.*

Bizonyítás. Vegyünk egy egyszerűbb feladatot, az X elemeinek összeadását. Ez nyilván kevesebb lépést igényel, mivel prefixszámítás esetén is össze kell adni az összes elemet (legutolsó prefix), és még további prefixeket is meg kell határozni.

Megmutatjuk, hogy X elemeinek összeadásához a PRAM modellben legalább $\lg n$ időegységre van szükség. A PRAM számítási modellben egy lépésben egy processzor legfeljebb két elemet adhat össze. Az első lépésben tehát legfeljebb $(n/2)$-re, a másodikban $(n/4)$-re, ... csökkenthető az összeadandók száma, ezért valóban legalább szükség van legalább $\lg n$ lépésre. ■

További munkahatékony algoritmusok

Az előbbi két algoritmus egy korlátot ad a processzorszámra és futási időre vonatkozóan. Az előbbi munkahatékony algoritmusokat vizsgálva ennél többet is mondhatunk: tetszőleges $a \in (0,1)$ számhoz meg tudunk adni olyan munkaoptimális algoritmust, amely n^a részre osztja a sorozatot.

Ennek belátásához végig kell nézni az algoritmus mindhárom lépését, hogy azokra teljesül-e a munkaoptimalitás (az algoritmus felfogható három részalgoritmus egyesítéseként, ahol a futási idők összeadódnak). Az első lépésben a bemenetet n^a részre osztva, minden rész n^{1-a} elemet tartalmaz, így a soros algoritmust figyelembe véve ennek a résznek a futási ideje n^{1-a} lesz.

A harmadik lépésben szintén az n^a részt aktualizáljuk, ami n^{1-a} lépésből áll. A második lépés okozhat problémát, miszerint a $\lg n^a$ vajon aszimptotikusan nagyobb-e az (n^{1-a})-nál. A l'Hospital-szabály segítségével belátható, hogy ha $a \in (0,1)$, akkor

$$\lim_{n\to\infty} \frac{n^a}{\lg n} = \infty.$$

Innen adódik, hogy ha $a \in (0,1)$, akkor az Optimális-prefix algoritmust véve alapul a futási idő tetszőleges n^a alakú polinom lehet úgy, hogy minden esetben munkahatékony algoritmust kapjunk.

Ehhez elég azt észrevenni, hogy az algoritmus első részének műveletigénye $O(n^{1-a})$, a második részének műveletigénye $O(\lg n^a)$ és a harmadik rész műveletigénye ugyancsak $O(n^{1-a})$.

Így az összes műveletigény $O(n^{1-a}) + O(\lg n^a) + O(n^{1-a}) = O(n^{1-a})$.

20.5.5. Kiválasztás

Adott $n \geq 2$ kulcs és egy i $(1 \leq i \leq n)$ egész szám. A feladat az i-edik legkisebb kulcs álasztásálasztása. Mivel a kiválasztáshoz minden elemet meg kell vizsgálni, ezért $N(n) = \Omega(n)$. Erre a feladatra ismert olyan A soros algoritmus, amelyikre $W(n, \mathrm{A}) = O(n)$, tehát A aszimptotikusan optimális.

Ehhez hasonló a ***keresési feladat,*** amelyben azt kell eldönteni, hogy adott elem előfordul-e a vizsgált sorozatban – és ha igen, milyen indexszel. Ennél a feladatnál tagadó válasz is lehetséges és egy elemről tulajdonságai alapján eldönthető, megfelel-e a keresési feladatnak.

Először 3 speciális esetet vizsgálunk, majd egy munkahatékony véletlenített algoritmust ismertetünk.

Kiválasztás konstans időben n^2 processzoron

Legyen $i = n$, azaz a legnagyobb kulcsot keressük. Ez a feladat a következő NÉGYZETES-KIVÁLASZT algoritmussal n^2 CRCW processzoron $O(1)$ lépéssel elvégezhető.

A bemenő adatokat (n különböző kulcsot) a $K = k_1,\ k_2,\ \ldots,\ k_n$ tömb, a megtalált legnagyobb kulcsot pedig az y változó tartalmazza.

NÉGYZETES-KIVÁLASZT(K,Y)

1 **if** $n = 1$
2 $\quad y = x_1$
3 $\quad$ **return** y
4 P_{ij} **in parallel for** $i = 1$ **to** n, $j = 1$ **to** n
5 $\qquad\qquad\qquad$ számítsa ki a $x_{ij} = k_i < k_j$ értéket
6 osszuk az n^2 processzort n csoportba $(G_1, \ldots, G_n)$ úgy, hogy a G_i csoportba a $P_{i,1}, \ldots, P_{i,n}$ processzorok kerüljenek. Mindegyik csoport végezzen logikai VAGY műveletet az $x_{i1}, \ldots, x_{in}$ logikai változókkal
7 ha a G_i csoport számítási eredménye a 4. lépésben HAMIS, akkor a csoport P_{i1} processzora adja meg $(y = k_i)$-t kimenetként

Legyenek a bemenő adatok $k_1, \ldots, k_n$. Az összehasonlításokat párhuzamosan végezzük a P_{ij} $(1 \le i, j \le n)$ processzorokon úgy, hogy P_{ij} az $x_{ij} = (k_i < k_j)$ logikai értéket számítja ki. Feltehetjük, hogy a kulcsok különbözőek. Ha mégse, k_i helyett a (k_i, i) párt alkalmazva különbözővé tehetők: ehhez minden kulcshoz egy $(\lg n)$-bites számot kell hozzáadni. Ekkor egyetlen olyan kulcs van, amelyikre minden összehasonlítás eredménye HAMIS. Ez a kulcs egy logikai VAGY művelettel azonosítható.

20.14. tétel. (kiválasztás $O(1)$ lépéssel). *n kulcs közül a maximális $O(1)$ lépéssel meghatározható n^2 CRCW közös* PRAM *processzoron.*

Bizonyítás. Az első és a harmadik szakasz egységnyi ideig tart. A második szakasz $O(1)$ lépéssel elvégezhető. ∎

Ennek az algoritmusnak a relatív sebessége $\Theta(n)$. Az elvégzett munka

$\Theta(n^2)$. Ezért a hatékonyság $\Theta(n)/\Theta(n^2) = \Theta(1/n)$. Tehát az algoritmus nem munkaoptimális.

Kiválasztás logaritmikus időben n processzoron

Most megmutatjuk, hogy a maximális elem p közös CRCW processzoron $O(\lg\lg p)$ lépéssel meghatározható. A technika az oszd-meg-és-uralkodj. Az egyszerűség kedvéért feltesszük, hogy p négyzetszám.

Legyenek a bemenő adatok $X = k_1, k_2, \ldots, k_p$. Legyen az algoritmusunk futási ideje $T(p)$. A bemenő adatokat $\sqrt{p} = a$ csoportra osztjuk úgy, hogy minden csoportban a elem legyen. Minden csoporthoz rendeljünk q processzort – ekkor a csoportok maximális eleme párhuzamosan számítható. Mivel csoportonként q elem és ugyanannyi processzor van, a csoport maximális eleme $T(a)$ lépéssel meghatározható. Legyenek $M_1, M_2, \ldots, M_a$ a csoportok maximális elemei. Ezek maximuma lesz az algoritmus kimenete. Mivel most csak q elemünk van, az összes processzort alkalmazhatjuk (lásd a 20.16. ábrát).

A következő rekurzív CRCW algoritmus $O(\lg\lg p)$ lépést tesz. Bemenő és kimenő adatok az előző algoritmus adataihoz hasonlóak.

GYÖKÖS-KIVÁLASZT(X, K, y)

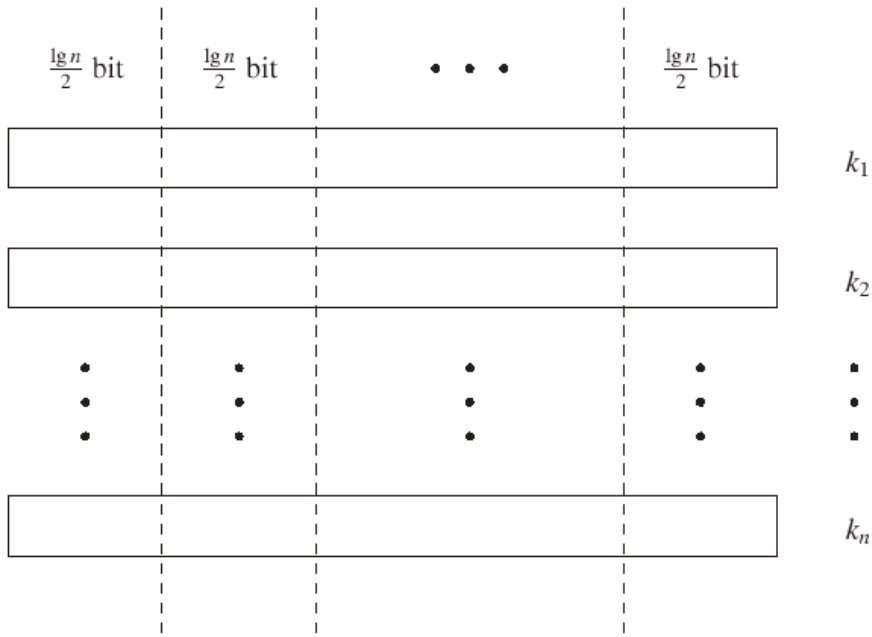
1 **if** $p = 1$
2 $\quad y = x_1$
3 $\quad$ **return** Y
4 osszuk a bemenetet a részre $(K_1, K_2, \ldots, K_a)$ úgy, hogy K_i a $k_{(i-1)a+1}, k_{(i-1)a+2}, \ldots, k_{ia}$ elemeket tartalmazza. Hasonlóképpen csoportosítsuk a processzorokat úgy, hogy a P_i $(1 \le i \le a)$ csoportba a $P_{(i-1)a+1}, P_{(i-1)a+2}, \ldots, P_{ia}$ processzorok tartozzanak. A P_i csoport határozza meg rekurzívan a K_i csoport maximális elemét.
5 legyenek a csoportok maximális elemei $M_1, M_2, \ldots, M_a$, és határozzuk meg ezek maximumát a NÉGYZETES-KIVÁLASZT algoritmussal
6 **return** Y

20.15. tétel. (kiválasztás $O(\lg\lg p)$ lépéssel). *A* GYÖKÖS-KERES *algoritmus p közös* CRCW PRAM *processzoron $O(\lg\lg p)$ lépéssel meghatározza p kulcs közül a legnagyobbat.*

Bizonyítás. Ennek az algoritmusnak az első lépése $W(\sqrt{p})$, második lépése $O(1)$ ideig tart. Ezért $W(p)$ kielégíti a

$$W(p) = W(\sqrt{p}) + O(1) \tag{20.28}$$

rekurzív egyenletet, melynek megoldása $O(\lg\lg p)$. ∎



20.16. ábra. Maximális egész szám kiválasztása

A GYÖKÖS-KERES algoritmus összes munkája $\Theta(p \lg\lg p)$, ezért hatékonysága $\Theta(p)/\Theta(p \lg\lg p) = \Theta(1/\lg\lg p)$, így ez az algoritmus sem munkahatékony.

Kiválasztás egész számok közül

Legyen a feladat ismét n kulcs maximumának meghatározása. Ha a kulcsok egyetlen bitből állnak, akkor a maximum keresése visszavezethető a logikai VAGY műveletre és ezért $O(1)$ lépéssel meghatározható. Ebből adódik a kérdés: mekkora intervallumban lehetnek a kulcsok ahhoz, hogy p processzoron konstans lépéssel meg tudjuk határozni a maximális elemet?

Legyen c adott konstans, a kulcsok pedig legyenek a $[0, n^c]$ intervallumban. Ekkor a kulcsok legfeljebb $c \lg n$ bites bináris számok. Az egyszerűség kedvéért feltesszük, hogy pontosan ennyi bitesek (a számok elejére szükség esetén nullákat írunk).

A következő CRCW algoritmus $O(1)$ lépést tesz.

Az alapötlet az, hogy a számok $b_1, b_2, \ldots, b_{2c}$ bitjeit $(\lg n)/2$ hosszúságú *részekre* bontjuk. Az i-edik rész a $b_{(i-1)+1}$, $b_{(i-1)+2}, \ldots, b_{(i-1)+b_{(i-1)+(\lg n)/2}}$ biteket tartalmazza, a részek száma $2c$.

Ezt a helyzetet mutatja a **??**. ábra: először az ábra első oszlopában lévő bitek alapján keressük a maximális kulcsot.

Az algoritmus futási idejét a következő állítással jellemezzük.

20.16. tétel. (kiválasztás egész számok közül). *Ha a kulcsok a* $[0, n^c]$ *intervallumból vett egész számok, akkor* p *kulcs közül a maximális* $O(1)$ *lépéssel meghatározható* p CRCW PRAM *processzoron tetszőleges* c *konstans esetében.*

Bizonyítás. Tegyük fel, hogy a kulcsok maximumát a $(\lg n)/2$ legfontosabb bit alapján határozzuk meg.

Legyen az első részben a maximum M. Ekkor azok a kulcsok, melyek legfontosabb bitjei nem M-et adnak, biztosan nem maximálisak. Ezt az alaplépést megismételjük $2c$-szer, azaz minden $(\lg p)/2$ bitre pontosan egyszer. Legalább egy kulcs megmarad az utolsó lépés után is – az lesz az eredmény. Az utolsó rész lehet rövidebb, mint $(\lg p)/2$ bit.

Ha egy kulcs legfeljebb $(\lg n)/2$-bites, akkor az értéke legfeljebb $\sqrt{n} - 1$. Ezért az EGÉSZET-KIVÁLASZT első lépésében a $[0, \sqrt{n} - 1]$ intervallumba eső egész kulcsok maximumát kell meghatározni. Rendeljünk minden kulcshoz egy processzort és használjunk $\sqrt{n}$ közös memóriarekeszt $(M_1, M_2, \ldots, M_{\sqrt{n}-1})$, melyek tartalma kezdetben $-\infty$. Egy párhuzamos lépésben a P_i processzor k_i-t ír az M_{k_i} memóriarekeszbe. Ezután az n kulcs maximuma a $\sqrt{n}$ memóriarekesz tartalmából n processzorral a 2.9. tétel alapján konstans idő alatt meghatározható. ∎

Az algoritmus pszeudokódja a következő.

Az EGÉSZ-KULCSOKAT-KIVÁLASZT algoritmus bemenő adatai a p processzorszám és a különböző kapcsolókat tartalmazó $X = k_1, \ldots, k_p$ tömb, kimenő adata pedig az y változó.

EGÉSZ-KULCSOKAT-KIVÁLASZT(p, X, y)

1 **for** $i = 1$ **to** $2c$
2 határozzuk meg a megmaradt kulcsok maximumát i-edik részük alapján legyen a maximum M
3 hagyjuk el azokat a kulcsokat, melyek i-edik része kisebb, mint M
4 y legyen a megmaradt kulcsok egyike
5 **return** y

Általános kiválasztási feladat

Tegyük fel, hogy az $X = \langle k_1, k_2, \ldots, k_n \rangle$ sorozat különböző kulcsokat tartalmaz és az i-edik legkisebb kulcsot akarjuk kiválasztani. Legyen most az x_i

kulcs *rangja* eggyel nagyobb, mint a nála kisebb kulcsok száma (ez a definíció eggyel nagyobb értéket ad, mint a korábban használt).

Ezt a rangot a 20.5-5 gyakorlat szerint $n/\lg n$ CREW PRAM processzoron bármely kulcsra $O(\lg n)$ lépésben meg tudjuk határozni.

Ha $n^2/\lg n$ processzorunk van, akkor azokat $C_1, \ldots, C_n$ csoportokba oszthatjuk úgy, hogy minden csoportban $n/\lg n$ processzor legyen. A C_j $(1 \leq j \leq n)$ csoport $O(\lg n)$ lépésben meghatározza a k_j kulcs rangját X-ben. Annak a csoportnak egyik processzora, amelyik az i rangot határozta meg, adja a kimenetet. Az így kapott algoritmus neve legyen Ált-kiválaszt.

20.17. tétel. (általános kiválasztás). *Az* Ált-kiválaszt *algoritmus $n^2/\lg n$ processzoron n különböző kulcs közül $\Theta(\lg n)$ lépésben meghatározza az i-edik legkisebbet.*

Az Ált-kiválaszt algoritmus munkája $\Theta(n^2)$, tehát ez az algoritmus sem munkaoptimális, sőt nem is munkahatékony.

20.5.6. Rendezés

Adott $n \geq 2$ kulcs. A feladat ezek csökkenő vagy növekvő sorrendbe való rendezése.

Ismert, hogy ha a megengedett művelet a szokásos összehasonlítás, akkor minden A soros algoritmusnak $N(n, \mathrm{A}) = \Omega(n \lg n)$ lépésre van szüksége, másrészt vannak $O(n \lg n)$ futási idejű összehasonlítás-alapú algoritmusok, amelyek tehát aszimptotikusan optimálisak. Más műveletek vagy a rendezendő kulcsok speciális tulajdonságai esetében a rendezés $O(n)$ lépéssel is megoldható. Ha legrosszabb esetben minden kulcsot meg kell vizsgálni, akkor természetesen a futási idő $N(n) = \Omega(n)$. Tehát mind az összehasonlítás alapú, mind pedig a speciális esetben ismert aszimptotikusan optimális soros algoritmus.

Vizsgáljuk meg a következő kérdéseket. Hány rendező algoritmus van? Ezek közül hány egyszerű, hány optimális (aszimptotikusan, szigorúan)? Ezekre a kérdésekre nem könnyű válaszolni – például először pontosan definiálnunk kell, mi is az a rendező algoritmus.

Szűkítsük a kérdést: hány összehasonlításra van szükség n elem rendezéséhez? Jelöljük ezt a számot $c(n)$-nel. Ismert, hogy

$$\left\lceil \sum_{i=1}^{n} \lg i \right\rceil \leq c(n) \leq \sum_{i=1}^{n} \lceil \lg i \rceil \tag{20.29}$$

és hogy

$$c(n) \leq n \lg n - (n-1) \; . \tag{20.30}$$

Az alsó becslés döntési fákkal vagy információelméleti eszközökkel igazolható, a felső becslések pedig a bináris beszúró, illetve az összefésüléses rendező jellemző adatai.

4 elemre az alsó és a felső becslések egyaránt ötöt adnak. 5 elemre $5! = 120$ miatt az alsó becslés 7, viszont az előbbi algoritmusoknak 8 összehasonlításra van szüksége.

Rendezés logaritmikus időben n^2 processzoron

n^2 processzoron a kulcsok rangja $O(\lg n)$ lépéssel meghatározható. Ha a rangokat ismerjük, akkor a rendezés egy párhuzamos olvasással megoldható.

Tehát igaz a következő tétel.

20.18. tétel. (rendezés $O(\lg n)$ lépésben). *n kulcs n^2 CREW PRAM processzoron rendezhető $O(\lg n)$ lépéssel.*

Mivel a kulcsok meghatározása $\Omega(\lg n)$ ideig tart, ez a módszer $\Theta(n^2 \lg n)$ munkát igényel, azaz nem munkahatékony.

Páratlan-páros algoritmus – $O(\lg^2 n)$ futási idővel

Ez az algoritmus a klasszikus oszd-meg-és-uralkodj elvet alkalmazza. Az egyszerűség kedvéért legyen n kettő hatvány és a kulcsok legyenek különbözők.

A következő EREW PRAM algoritmus $O(\lg^2 n)$ lépést tesz. Bemenete a kulcsokat tartalmazó $X = \langle x_1, \ldots, x_p \rangle$ tömb, kimenete pedig a rendezett kulcsokat tartalmazó $Y = \langle y_1, \ldots, y_p \rangle$ tömb.

PÁRATLAN-PÁROS-RENDEZ(n, X)

```
1  if n = 1
2     Y = X
3     return Y
4  osszuk az X bemenetet két részre: ezek legyenek
          X'_1 = ⟨k_1, k_2, ..., k_{n/2}⟩ és ⟨X'_2 = k_{n/2+1}, k_{n/2+2}, ..., k_{2n}⟩.
5         rendezze n/2 processzor rekurzívan X'_1-t. Az eredmény legyen
          X_1. Ugyanakkor rendezze n/2 processzor rekurzívan X'_2-t.
          Az eredmény legyen X_2.
6         fésüljük össze X_1-et és X_2-t 2m processzoron
          a PÁRATLAN-PÁROS-ÖSSZEFÉSÜL algoritmussal Y-ná
7  return Y
```

Ez az algoritmus hasonlít a soros összefésüléses algoritmusra. Ott azonban a sorozatok legkisebb elemeit hasonlítjuk össze és a kisebb elem az összehasonlítás eredménye.

20.19. tétel. (rendezés $O(\lg^2 n)$ lépéssel). *n kulcs n EREW PRAM processzoron rendezhető $O(\lg^2 n)$ lépéssel.*

Bizonyítás. Legyen $W(n)$ az algoritmus futási ideje. A 4. lépés $O(1)$ ideig tart, az 5. lépés $W(n/2)$ ideig, a 6. lépés pedig $O(\lg n)$ ideig. Ezért $W(n)$ kielégíti a

$$W(n) = O(1) + W\left(\frac{n}{2}\right) + O(\lg n) \tag{20.31}$$

rekurzív egyenlőséget, melynek megoldása $W(n) = O(\lg^2 n)$. ■ **20.7.**

példa. Rendezés 16 processzorral. Rendezzük 16 processzoron a következő számokat: 25, 21, 8, 5, 2, 13, 11, 16, 23, 31, 9, 4, 18, 12, 27, 34. Az első lépésben a páros és páratlan részeket kapjuk meg, majd a másodikban az első 8 processzor a páratlan részből kapja az $X_1 = 2, 5, 8, 11, 13, 16, 21, 25$-öt, a második 8 processzor pedig az $X_2 = 4, 9, 12, 18, 23, 27, 31, 34$-et. A harmadik lépésben kapjuk az eredményt.

Ez az algoritmus $\Theta(n \lg^2 n)$ munkát végez. Hatékonysága $\Theta(1/\lg n)$, gyorsítása $\Theta(n/\lg n)$.

Preparata algoritmusa – $O(\lg n)$ futási idővel

Több processzorral a futási idő csökkenthető: Preparata rekurzív algoritmusa $n \lg n$ CREW PRAM processzoron $\lg n$ párhuzamos lépést végez. A bemenő $X[1..p]$ tömb a rendezendő, a kimenő $Y[1..p]$ tömb pedig a rendezett kulcsokat tartalmazza.

PREPARATA(X)

1 **if** $n \leq 20$
2 rendezzük az X bemenetet például a beszúró algoritmussal
3 **return** Y
4 osszuk az adott n kulcsot $\lg n$ részre úgy, hogy mindegyikben $n/\lg n$ kulcs legyen. Rendezzük a részeket külön, rekurzívan, mindegyik részhez n processzort rendelve. A rendezett sorozatok legyenek $S_1, S_2, \ldots, S_{\lg n}$.
5 fésüljük össze S_i-t és S_j-t $(1 \leq i, j \leq \lg n)$ párhuzamosan
6 rendeljünk $\lg n$ processzort a kulcsok eredeti sorozatra vonatkozó rangjának meghatározásához
7 legyen Y olyan tömb, amely a kulcsokat tartalmazza a rangok sorrendjében
8 **return** Y

Ez az algoritmus oszd-meg-és-uralkodj elvű. A kulcssorozatot $\lg n$ részre osztjuk, majd a részeket páronként összefésülve minden kulcsnak minden részre nézve meghatározzuk a rangját. Ezután a kulcsok tényleges rangja

az előbbi rangok összege.

Ha a harmadik lépésben minden (i, j) párhoz $n/\lg n$ processzort rendelünk, akkor az összefésülés $O(\lg\lg n)$ lépéssel elvégezhető.

A negyedik lépésben a rang számítása párhuzamosan végezhető a harmadik lépésben kapott $\lg n$ rang összeadásával: ez a prefixet számító algoritmussal $O(\lg\lg n)$ lépéssel megoldható.

20.20. tétel. (rendezés $O(\lg n)$ lépéssel). *A* Preparata *algoritmus n elemet $n\lg n$* CREW PRAM *processzoron $O(\lg n)$ lépéssel rendez.*

Bizonyítás. Legyen a Preparata-algoritmus futási ideje $W(n)$. A negyedik lépés lépésigénye $W(n/\lg n)$, az ötödik és hatodik lépésé együtt $O(\lg\lg n)$. Ezért

$$W(n) = W\left(\frac{n}{\lg n}\right) + O(\lg\lg n), \tag{20.32}$$

melynek megoldása (helyettesítéses módszerrel) $W(n) = O(\lg n)$. ∎

A lassulásra vonatkozó tétel segítségével kapjuk a következő állítást.

20.21. következmény. (rendezés $(n\lg n)/t$ processzoron). *Tetszőleges $t \geq 1$ egész számra n tetszőleges kulcs rendezhető $O(t\lg n)$ lépésben $(n\lg n)/t$* CREW PRAM *processzoron.*

A Preparata-algoritmus munkája ugyanannyi, mint a páros-páratlan rendező algoritmusé, viszont a gyorsítása jobb: $\Theta(n)$. Mindkét algoritmus hatékonysága $\Theta(1/\lg n)$.

Gyakorlatok

20.5-1. A globális memória M_1 rekeszében van bizonyos adat. Másoljuk át ezt az adatot az $M_2, M_3, \ldots, M_n$ rekeszekbe. Mutassuk meg, hogyan lehet ezt megvalósítani $O(\lg n)$ lépéssel n EREW PRAM processzor felhasználásával.

20.5-2. Adjunk meg egy olyan algoritmust, amely $n/\lg n$ PRAM processzor felhasználásával $O(\lg n)$ lépéssel megoldja az előző gyakorlatot.

20.5-3. Legyen $f(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$. Adjunk $O(1)$ idejű CREW PRAM algoritmust a polinom értékének adott x helyen való kiszámítására. Mennyi processzort igényel a javasolt algoritmus?

20.5-4. Adjunk meg egy $O(\lg\lg n)$ idejű algoritmust, amely $n/\lg\lg n$ közös CRCW PRAM processzoron $O(\lg\lg n)$ lépésben megadja n tetszőleges szám maximumát.

20.5-5. Legyen A egy n kulcsot tartalmazó tömb. Mutassuk meg, hogy $n/\lg n$ CREW PRAM processzoron $O(\lg n)$ lépésben meghatározható tetszőleges

$k \in A$ kulcs rangja.

20.5-6. Tervezzünk egy $O(1)$ futási idejű algoritmust, amely n közös CRCW PRAM processzoron eldönti, hogy adott $A[1\,..\,n]$ tömb elemei között előfordul-e az 5, és ha igen, megadja a legnagyobb olyan i indexet, amelyre $A[i] = 5$.

20.5-7. Tervezzünk algoritmust, amely n^2 CREW PRAM processzoron $O(1)$ lépésben összefésül két n hosszúságú rendezett sorozatot.

20.5-8. Határozzuk meg a fejezetben tárgyalt algoritmusok gyorsítását, munkáját és hatékonyságát.

Feladatok

20-1 Közös elem

Tervezzünk n^2 CRCW PRAM processzoron $O(1)$ futási idejű algoritmust annak eldöntésére, hogy adott $A[1\,..\,n]$ és $B[1\,..\,n]$ tömböknek van-e közös eleme.

20-2 Minimális feszítőfa

Párhuzamosítsuk a minimális feszítőfák meghatározására szolgáló Kruskal-algoritmust és Prim-algoritmust. Tervezzünk algoritmust arra a speciális esetre, amikor az élek súlya csak 0 vagy egy lehet.

20-3 Összes csúcspár távolsága

Párhuzamosítsuk a gráfok összes csúcspárjának távolságát meghatározó Bellman-Ford algoritmust.

20-4 Körmentesség

Tervezzünk egy P párhuzamos algoritmust annak eldöntésére, hogy adott irányítatlan gráf tartalmaz-e kört. Elemezzük a különböző nagyságrendű processzorszám esetében elérhető $W(n, p, \mathrm{P})$ futási időket.

Megjegyzések a fejezethez

A számítógépek felépítését elsősorban Claudia Leopold [181], Leighton [175, 176], valamint Sima, Fountain és Kacsuk [269] monográfiája alapján tárgyaljuk. A párhuzamos programozásról szóló alfejezetek alapja Grama, Gupta, Karypys és Kumar [105], Leopold [181], valamint Roosta [252] könyve.

Az 500 legnagyobb teljesítményű számítógép adatait [305] tartalmazza.

A tárgyalt párhuzamos algoritmusok többsége Berman és Paul [25], Cormen, Leiserson és Rivest [53], Horowitz, Sahni és Rajasekakaran [120], Iványi Antal [131], valamint Jaja [135] tankönyvéből származik.

A számítógépek ismertetett osztályozási rendszereit Flynn [85], illetve

Leopold [181] javasolta. Ugyancsak Leopold említett könyvéből származnak a **??**., a **??**., a 20.3., a **??**. és a 20.7. ábrák.

A klasztereket Pfister [236], a grideket pedig Foster és Kesselman könyve [90], valamint hálózaton elérhető anyaga [90] alapján tárgyaljuk. A feladatok kisebb feladatokra való bontásával részletesen foglalkozik például Tanenbaum [292].

A közös memória használatával foglalkozik Hwang és Xu [124], Kleiman és társai [155], valamint Tanenbaum és van Steen [294], Részletesen tárgyalják az üzenetküldést a Culler és társai [58], valamint a Snir és társszerzői [273] által írt könyvek.

Amdahl, Brent és Gustafson eredményei a névadók cikkeiből származnak [3, 35, 111]. A lokalitás javításának módszereit elemzi például Kandemir, Ramanujam és Choudhary [145]. A kódtranszformáció és az adatok kapcsolatával részletesen foglalkozik Wolfe [315]. Sok hasznos ismeretet tartalmaz a kódoptimalizációról Kennedy és Allen könyve [151].

Az MPI programozási modellt Gropp [107], az OpenMP modellt pedig Chandra és társainak műve [43], valamint egy hálózaton elérhető anyag [223] alapján ismertetjük. További programozási modellek – mint a csontvázak, párhuzamos funkcionális programozás, koordinációs nyelvek és párhuzamos mobil ágensek – részletes ismertetése megtalálható Leopold [181] könyvében.

A P-szálakkal például Lewis és Berg [182], a Java-szálakkal pedig Oaks és Wong [220] foglalkoztak részletesen. A nagy teljesítményű FORTRAN leírása megtalálható Koelbel könyvében [157]. A párhuzamosító fordítóprogramokkal például Wolfe [315] foglalkozott.

A PRAM számítási modellt 1978-ban vezette be Fortune és Willye [89].

A BSP modellt Valiant [307], a LogP modellt Culler és társai [58], míg a QSM modellt Gibbons, Matias és Ramachandran javasolták [99].

A munkahatékony prefixszámítási algoritmusok elemzése Iványi cikkén [130] alapul. A feladatokban szereplő algoritmusok többsége megtalálható az *Új algoritmusok* [53] című tankönyvben.

21. Petri-hálók és elosztott programok

A Petri-hálók szemléletes gráf-modellek, amelyek párhuzamos folyamatok leírására, modellezésére, elemzésére alkalmasak. A modellezett folyamatok lehetnek kémiai, fizikai, gyártási, irányítási, társadalmi folyamatok vagy éppen párhuzamos, elosztott algoritmusok is. Petri-hálókkal különböző absztrakciós szinten fogalmazhatunk meg modelleket, részletekben gazdagabb leíráshoz általában nagyobb háló tartozik.

A fejezet első részében a Petri-hálókkal kapcsolatos alapfogalmakat vezetjük be. Ezt követően a Petri-hálókkal kifejezetten azzal a céllal foglalkozunk, hogy elosztott algoritmusok, programok tulajdonságait igazoljuk. Ezen cél eléréséhez a fejezet második részében a Petri-háló általánosan használt definícióját kiegészítjük, bevezetjük a Petri-dobozok fogalmát.

A modellezett rendszerek, például elosztott programok tulajdonságait oly módon is igazolhatjuk, hogy a nekik megfelelő Petri-hálót vizsgáljuk. Petri-hálók segítségével elsősorban a program egyes folyamatai vagy folyamatrészei közötti sorrendi, függőségi és szinkronizációs kapcsolatokat, az információ áramlását, a kommunikációt, a konfliktus-, illetve a konkurenciahelyzeteket elemezhetjük. Egyidejűleg az egyes értékadások, függvényhívások jelentésétől elvonatkoztatunk. Ha két különböző rendszert modellező Petri-háló lényegében azonos, akkor a két rendszer kommunikációs, szinkronizációs struktúrája is megegyezik az adott absztrakciós szinten. Ebben az esetben hasonló dinamikus viselkedésre számíthatunk még akkor is, ha az egyik társadalmi folyamatokat modellez a másik pedig egy számítógépes hálózat protokolljának felépítését írja le.

Egy program szövegében könnyen azonosíthatók a szinkronizációs és kommunikációs elemek, így az algoritmus működését modellező háló akár programmal is előállítható és az elemzés is automatizálható. Állapot- vagy adatfolyam-diagramból is származtathatunk Petri-hálót. A háló vizsgálata alapján kapott eredmények felhasználásához azonban azt is tudnunk kell, hogy a háló egyes részei mely programrésznek vagy adatelemnek feleltek meg eredetileg. Ennek érdekében a hálók egyes elemeit a programszövegre utaló címkékkel láthatjuk el. Programok tervezésére, tulajdonságaik vizsgálatára készített modellek egyik legfontosabb jellemzője, hogy tudunk-e programrészek tulajdonságaiból az összetett rendszer tulajdonságaira következtetni. Ahhoz, hogy elemi programoknak megfelelő hálókból összetett algoritmusokat

leíró hálókat állítsunk elő, a programkonstrukcióknak megfelelő kompozíciós műveleteket is definiálnunk kell a Petri-hálók felett.

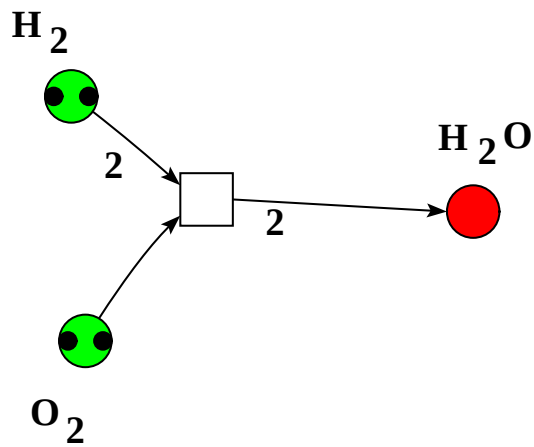
21.1. Alapfogalmak

A Petri-háló fogalmát páros gráfok segítségével definiálhatjuk. Páros gráfnak nevezzük azt a gráfot, amelyben a csúcsok két diszjunkt halmazba sorolhatóak oly módon, hogy az azonos halmazba tartozó csúcsok között nem vezet él. A Petri-háló egy olyan rendezett pár, amelynek első eleme magát a hálót definiáló irányított gráfot adja meg, míg második eleme a kezdősúlyozást. A gráfban kétféle csúcspont van: hely, illetve átmenet. A helyek feltételeket jelölnek, az átmenetek eseményeket, például értékadásokat, összetett utasításokat, eljárásokat vagy függvényhívásokat. Az élek az átmenetekkel reprezentált események előfeltételeit ábrázoló helyeket kötik össze az átmenetekkel, illetve az átmeneteket az utófeltételeknek megfelelő helyekkel. Minden élhez tartozik egy súlyérték is, ez szükségtelenné teszi a többszörös élek használatát. Ha például egy él súlya kettő, akkor az él két egyszeres élnek felel meg. A helyekhez is rendelünk súlyokat, amelyekkel az elő-, illetve utófeltételek teljesülését reprezentáljuk. A helyekhez tartozó súlyértékek a Petri-háló működése során az egyes átmenetek hatására dinamikusan változnak. A Petri-háló állapotát azzal írjuk le, hogy megadjuk az egyes helyeken lévő súlyok számát.

21.1. definíció. (Petri-háló). *A* ***Petri-háló*** *egy* (N, M_0) *rendezett pár, ahol:*

- $N = (P, T, R, v)$ *a tartógráf, irányított, páros gráf, amelynek az élei súlyozottak,*
- P, T *: a csúcsok véges halmazai.* P *a helyek halmaza,* T *az átmenetek halmaza,*
- $P \cup T \neq \emptyset$,
- $P \cap T = \emptyset$,
- R *– az éleket megadó reláció:* $R \subseteq (P \times T) \cup (T \times P)$,
- $v : R \mapsto \mathcal{N}_0$ *- az élek súlyait megadó függvény,*
- *A helyek kezdeti súlyozását (kezdőállapot)* M_0 *adja meg:* $M_0 : P \to \mathcal{N}_0$. *A helyek súlyozását (állapot) általában rendezett n-esként adjuk meg, például* $M_0 = (1, 0, \ldots, 2)$.

Jelölések:



21.1. ábra. Víz szintézise.

Petri-hálók tulajdonságainak tömör leírásához az alábbi jelöléseket vezetjük be:

- a p helyet megelőző átmenetek halmaza:
  ${}^\bullet p ::= R^{(-1)}(p)$, (a p hely R élreláció szerinti inverz képe),
- a t átmenetet megelőző, bemenő helyek halmaza:
  ${}^\bullet t ::= R^{(-1)}(t)$, (a t átmenet R élreláció szerinti inverz képe),
- adott csúcs utóda(i): $p^\bullet ::= R(p), t^\bullet ::= R(t)$.

A helyeket körrel, az átmeneteket négyzettel jelöljük. A 21.1. ábrán egy egyszerű Petri-hálót láthatunk, amely a víz oxigén- és hidrogénmolekulákból történő szintézisének folyamatát modellezi. A háló három helyet és egy átmenetet tartalmaz. Az átmenet két vízmolekula ($2 * H_2O$) szintézisét reprezentálja, melynek előfeltétele, hogy legalább két hidrogénmolekula ($2 * H_2$) és legalább egy oxigénmolekula (O_2) álljon rendelkezésre. Az ábrán látható élsúlyozás ezt fejezik ki, az átmenethez tartozó H_2 címkéjű helytől kettő súlyú él vezet az átmenethez. Az esemény bekövetkezésének egyik előfeltétele, hogy ezen a bemenő helyen legalább két súly (két hidrogénmolekula) jelenléte szükséges. Ha egy él súlyát nem jelöljük, akkor az megállapodás szerint egy. Ilyen él vezet az O_2 címkéjű helytől az átmenethez. Ez azt jelenti, hogy az átmenet másik előfeltétele szerint legalább egy súly szükséges az alsó bemenő helyen is. Az ábrán látható esetben az átmenet előfeltételei teljesülnek, az átmenet tüzelhet. Az átmenet tüzelésének hatására a bemenő

helyekről az onnan vezető élek súlyának megfelelő számú súly lekerül, a kimenő helyekre pedig az oda vezető élek súlyával egyenlő számú súly hozzáadódik.

Egy átmenethez tartozó esemény előfeltétele akkor teljesül, ha a gráfban az átmenetet megelőző helyeken kellő számú súly helyezkedik el. A Petri-háló működési szabálya adja meg, hogy egy átmenet mikor tüzelhet és hogyan változik a háló állapota. Többféle működési szabály is megadható. Jelöljük M-mel a Petri-háló aktuális állapotát leíró, illetve M'-vel a tüzelés után előálló állapotot megadó súlyvektort. $M(p)$ a p helyen lévő súlyok száma. $v(p,t)$ a p helyről t átmenethez, $v(t,p)$ pedig a t átmenettől a p helyhez vezető él súlya. Az él t tüzelése esetén éppen ennyi súlyt távolít el, illetve ad hozzá a p helyhez.

21.2. definíció. (működési szabály (alapértelmezés)).

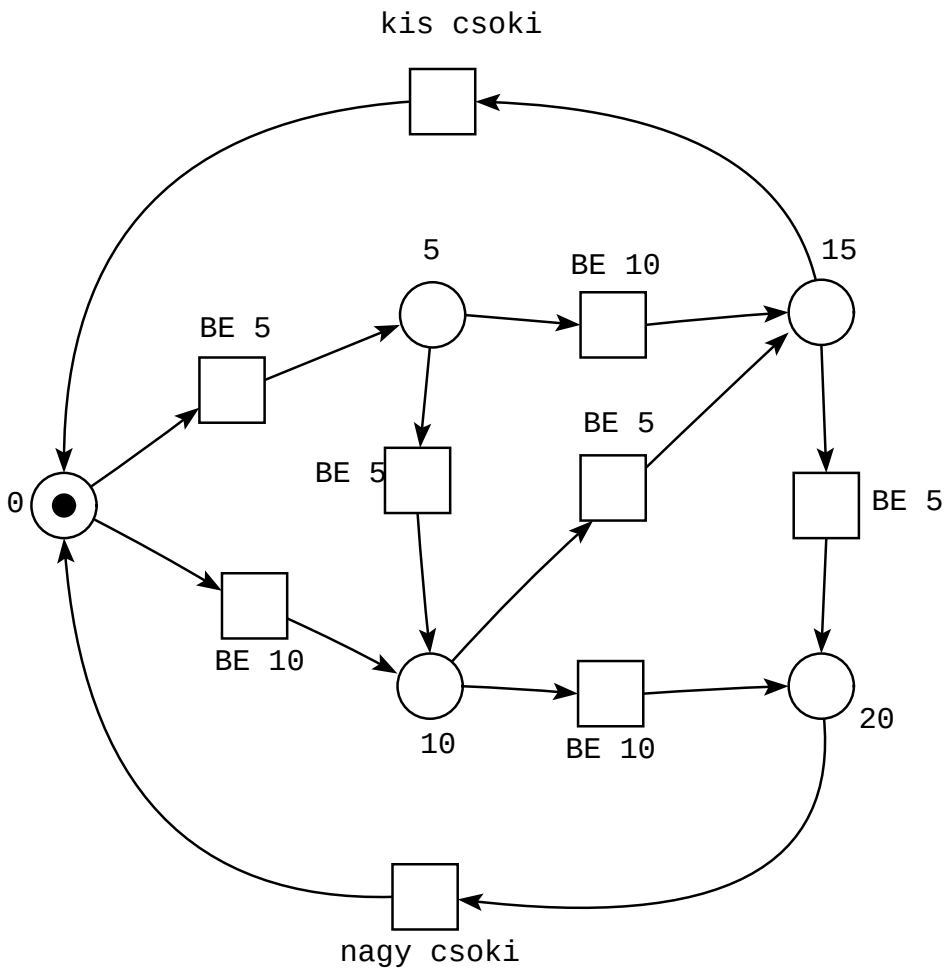
1. *t átmenet **tüzelhet** (aktivizálható, megengedett), ha* $\forall p \in {}^\bullet t : M(p) \geq v(p,t)$.

2. *t átmenet tüzelése után az új állapotot leíró M' súlyozás:* $\forall p \in P : M'(p) = M(p) + v(t,p) - v(p,t)$.

Általában nem teljesül, hogy a tüzelés során a háló különböző helyein lévő súlyok számának összege megmarad. Előfordulhat, hogy az átmenet több vagy kevesebb súlyt távolít el a bemenő helyekről, mint amennyit a kimenő helyeken elhelyez. Ha egy átmenetnek nincs bemenő helye, akkor az átmenet előfeltétele folyamatosan teljesül, tetszés szerinti időpontban tüzelhet, ú.n. ***forrás átmenet***: ${}^\bullet t = \emptyset$. Ha egy átmenetnek nincs kimenő helye, akkor a tüzelés során csak a bemenő helyeken csökken a súlyok száma, sehol nem jelenik meg azonban új súly. Az ilyen átmeneteket ***nyelő átmenetnek*** nevezzük: $t^\bullet = \emptyset$. Ugyanaz a hely lehet egyszerre bemenő és kimenő helye is ugyanazon átmenetnek, ilyenkor ***hurokról*** beszélünk: $(p,t) : p \in {}^\bullet t \wedge p \in t^\bullet$. Egy háló ***átlátszó***, ha hurokmentes. ***Normális*** hálónak nevezzük azokat a Petri-hálókat, amelyekben minden behúzott él súlya pontosan egy: $\forall r \in R : v(r) \leq 1$.

Akciósorozatnak nevezzük a háló által végrehajtott tüzelésekben szereplő átmenetek azonosítóinak sorozatát. Például: $t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, \ldots$.

A 21.2. ábrán egy csoki automata működésének modelljét mutatjuk be. Kezdőállapotban az automata nyílásában 0 Ft értékű érme van, ezt a helyzetet mutatja az az állapot, amikor csak a 0-val jelölt helyen van súly. Két esemény előfeltétele is teljesül, ennek megfelelően a 0 helyet követő „BE 10”, illetve



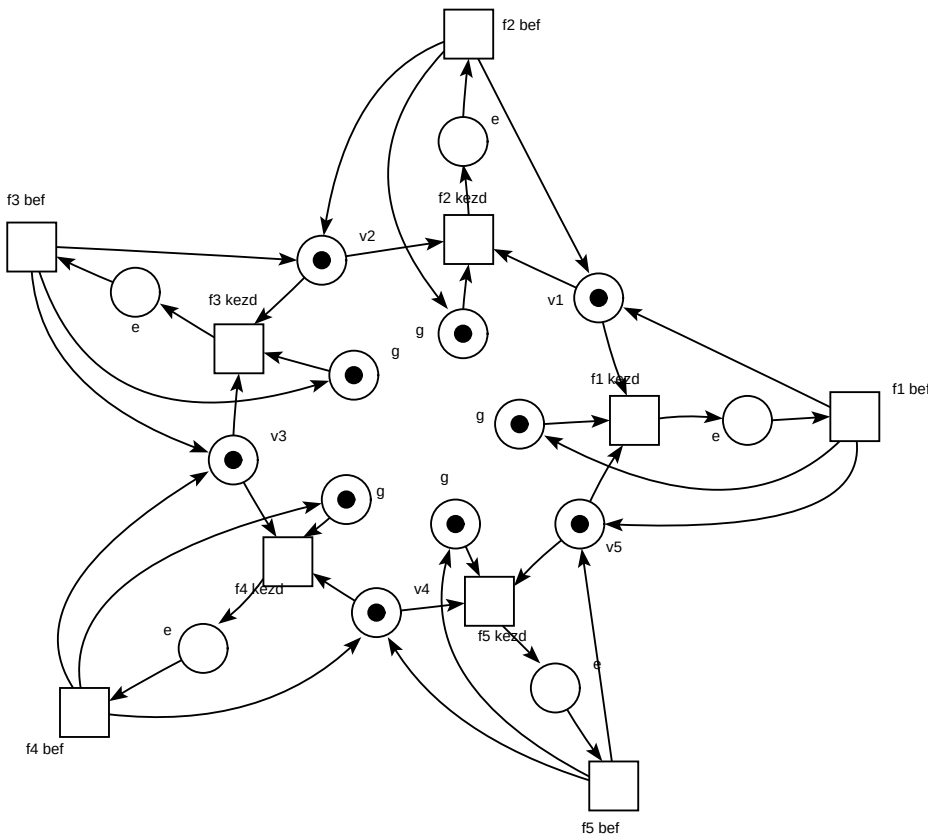
21.2. ábra. Csoki automata.

a „BE 5” eseménnyel címkézett átmenet is tüzelhet. Egy átmenet azonban nem feltétlenül tüzel akkor, ha az előfeltétele teljesül, külső feltételektől is függhet, hogy a tüzelésre sor kerül-e és mikor. Ha a fogyasztó egy 5 Ft-os érmét dob be, akkor a „BE 5” jelű átmenet tüzel, ha 10 Ft-os érme kerül a gépbe, akkor a „BE 10” jelű átmenet megy végbe. A Petri-háló működése nem-determinisztikus, a modell nem írja le, hogy a döntés mitől függ.[1] Az első esetben a súly a 0-val jelölt helyről az 5-tel, a másik esetben pedig a 10-zel jelölt helyre kerül át. Figyeljük meg, hogy ugyanazon esemény (pl. „BE 10”) egyes előfordulásait meg tudjuk különböztetni egymástól oly módon, hogy különböző átmenet tartozik hozzá. Ha a rendszer eljut a 15-tel jelölt állapotba, akkor a fogyasztó ismét választhat, hogy egy kis csokit kér, vagy egy újabb 5 Ft-os érme bedobása után egy nagy csokit ehet meg. Végül a gép visszatér alapállapotába.

Következő példánk E. W. Dijkstrától származik, aki az operációs rendszerek tárgy tanítása során a folyamatok közötti kölcsönös kizáráson alapuló erőforrás-megosztás elvét szemléltette vele. Ez a példa öt, egymással együttműködő párhuzamos folyamatból álló rendszert modellez. A történet szerint egy asztal körül öt filozófus ül, akik egy nagy közös tálból vehetnek maguknak makarónit. Ahhoz, hogy a makaróni a tányérba kerüljön, két villára van szükség. A tálat többen is használhatják egyszerre, de az étkezéshez szükséges villák mindegyikére külön-külön teljesülnie kell, hogy egyszerre csak egy filozófus használatában lehetnek. Minden filozófus számára két villa érhető el, ezek azonban olyan villák, amelyeket a szomszéd filozófusok is szeretnének használni. Ha egy filozófus felemeli az asztalról bal és jobb oldali villáját és eszik, akkor egyik szomszédja sem rendelkezhet az étkezéshez szükséges két villával.

A csillag alakú 21.3. ábrán az egyes filozófusokat egy-egy ág reprezentálja. Az egyes sorszámú filozófust például a jobb oldali vízszintes ággal modellezzük. Kezdőállapotban minden filozófus gondolkodik, a filozófus állapotát leíró súly a „g” jelű helyen van. Kezdetben az összes villa az asztalon fekszik, a „v1”, ..., „v5” jelű helyek mindegyike tartalmaz egy-egy súlyt. Az „f1 kezd” jelű átmenet előfeltételei teljesülnek, az első filozófus elkezdhet enni oly módon, hogy mindkét szomszédos villát felveszi. Ha az „f1 kezd” átmenet tüzel, akkor mindhárom bemenő helyéről eltűnnek a súlyok és az egyes sorszámú filozófus „e” jelű, étkezését jelző helyén jelenik meg egy súly. Ebben az állapotban az „f2”-vel és „f5”-tel jelölt átmenetek nem tüzelhetnek, az előfeltételeik közül egy-egy hamis, egy-egy villa hiányzik az átmenet végrehajtásához. Az első filozófus étkezésének befejeztekor az „f1 bef” jelű átmenet tüzel és visszateszi a súlyokat a kiinduló helyzetnek megfelelő helyekre. Figyeljük meg, hogy

[1] Egyes kiterjesztett modellek megengedik tiltó feltételek, valószínűségi értékek, illetve prioritások megadását is a nem-determinisztikus választás befolyásolására.



21.3. ábra. Étkező filozófusok.

csak egymással nem szomszédos filozófusok étkezhetnek egyidejűleg. Ez az egyszerű modell kizárja holtpont kialakulását azzal, hogy a két villa felvételét egyetlen atomi eseménybe sűríti. Nem alakulhat ki holtpont oly módon, hogy minden filozófus felemeli a bal oldali villáját és a végtelenségig arra várakozik, hogy a jobb oldali villa is megszerezhető legyen. Egyes filozófusok kiéheztetése azonban előfordulhat, hiszen semmi sem akadályozza meg azt, hogy egy filozófust szomszédai konfliktus helyzetben mindig megelőzzenek a villák megszerzésében.

Figyeljük meg, hogy az étkező filozófusokat bemutató hálóban a súlyok elhelyezkedése egyrészt az egyes filozófusok lokális állapotait is és egyúttal a teljes rendszer globális állapotát is meghatározza!

Gyakorlatok

21.1-1. Egy juhász át szeretne kelni a folyón egy kecskével, egy farkassal és egy fej káposztával. Egyszerre csak egy dolgot vihet magával a csónakjában a túlsó partra. Egyik parton sem maradhat kettesben a kecske és a farkas, illetve a kecske és a káposzta. Készítsük el az átkelés modelljét Petri-háló segítségével.

21.1-2. Módosítsuk a csoki automata működését leíró hálót oly módon, hogy kis vagy nagy csoki választása mellett a pénz visszakérésének lehetőségét is választhassa a fogyasztó.

21.1-3. Készítsünk az étkező filozófusok feladatához egy finomabb modellt, amelyben a filozófus a két villát nem egyszerre, hanem egymás után veheti fel.

21.2. Kapacitáskorlát

A csoki automata és az étkező filozófusok Petri-hálójára egyaránt igaz volt, hogy legfeljebb egy súly jelent meg egy-egy helyen. Elosztott programok és protokollok Petri-hálókkal történő modellezése során gyakran teljesül az, hogy egy-egy helyen egynél több súly egyszerre sohasem lehet. Ezen feltétel azonban nem minden hálóra teljesülne automatikusan. Ha biztosítani akarjuk, hogy a háló adott p helyén a súlyok száma sohasem haladhat meg egy előre megadott $k(p)$ kapacitásértéket, akkor ezt a működési szabály megváltoztatásával könnyen elérhetjük.

A szigorú működési szabály a kimenő helyeket is figyelembe veszi a tüzelés előfeltételének meghatározásakor. Ha a tüzelés hatására a kimenő helyek valamelyikén a súlyok száma meghaladja az arra a helyre érvényes kapacitáskorlátot, akkor a tüzelés nem engedélyezett.

21.3. definíció. (szigorú működési szabály). *Legyen $k : P \mapsto \mathcal{N}_0 \cup \{\infty\}$ egy kapacitáskorlátot definiáló függvény.*

- *1/a. t átmenet tüzelhet (aktivizálható, megengedett), ha $\forall p \in {}^\bullet t : M(p) \geq v(p,t)$ és $\forall p \in t^\bullet : M''(p) \leq k(p)$, ahol $M''(p) = M(p) + v(t,p) - v(p,t)$.*

A tüzelés után az új állapot megegyezik a 21.2. definíció 2. pontjában meghatározottal.

Egy ettől eltérő módszer szerint az átmenet mindenképp tüzelhet, de a kimenő hely kapacitáskorlátját meghaladó súlyok elvesznek.

21.4. definíció. (gyenge működési szabály).

- *2/a. t átmenet tüzelése után az új M' súlyozás: $\forall p \in P$: $M'(p) = \min(k(p), M(p) + v(t,p) - v(p,t))$.*

Az átmenetek pontosan akkor tüzelhetnek, amikor azt a 21.2. definíció 1. pontja megengedi.

21.2.1. Korlátos kapacitású helyek kiküszöbölése

Számunkra a szigorú működési szabály érdekes elsősorban, mert a számítógépek erőforrásainak végessége miatt szükséges lehet algoritmusok modellezésekor egyes helyekre kapacitáskorlátot meghatározni. Megmutatható azonban, hogy minden kapacitáskorláttal rendelkező és a szigorú működési szabályt használó háló helyettesíthető egy, az eredeti működési szabályt alkalmazó hálóval. A két háló abban az értelemben ekvivalens, hogy pontosan ugyanazok az akciósorozatok lehetségesek mindkettőben.

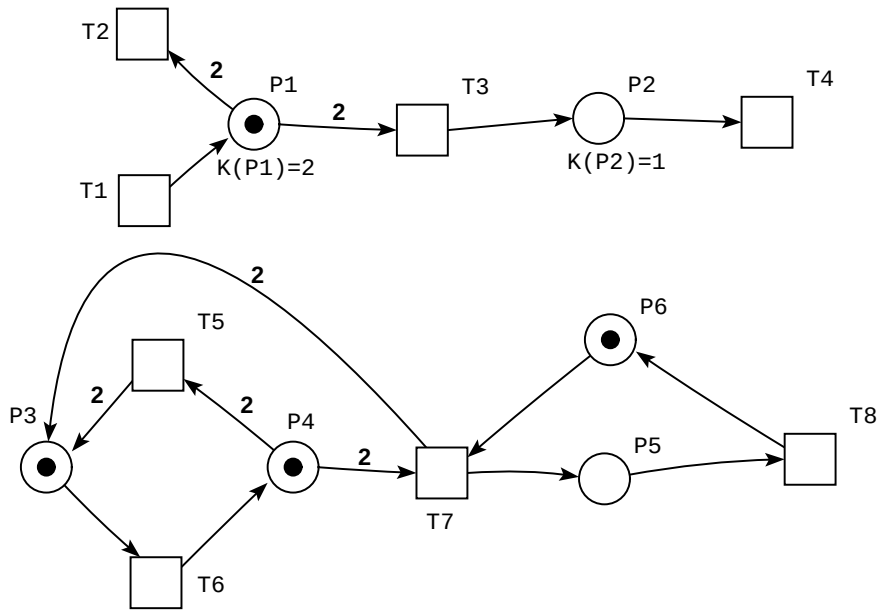
21.5. definíció. (komplemens hely transzformáció).

1. *$\forall p \in P : k(p) < \infty$ helyhez bevezetünk egy komplemens p' helyet úgy, hogy $M'_0(p') = k(p) - M_0(p)$. Jelöljük a komplemens helyek halmazát P'-vel.*

2. *$\forall p \in P', t \in T$ párhoz bevezetünk egy új élet úgy, hogy $v(t,p') = v(p,t)$ és $v(p',t) = v(t,p)$.*

A 21.4. ábrán egy példát láthatunk komplemens helyek bevezetésére.

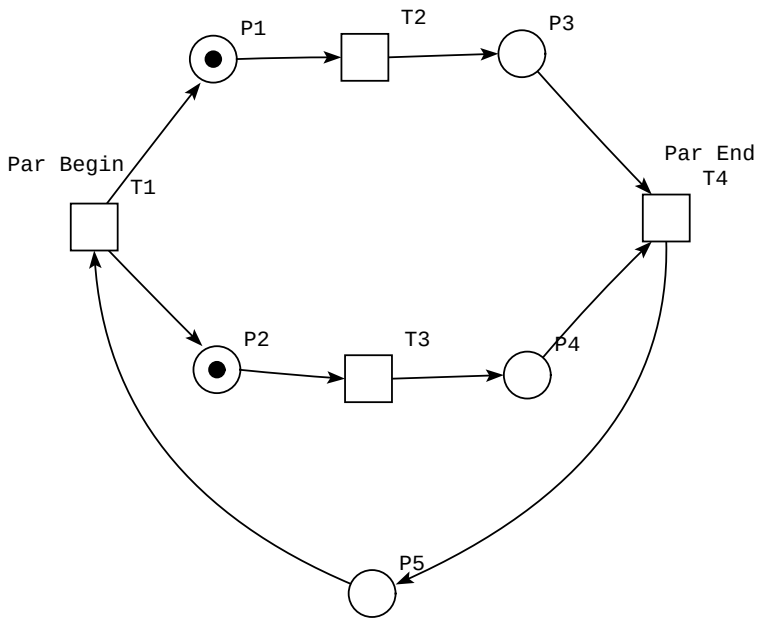
A komplemens helyeket az átmenetekkel összekötő új éleket éppen úgy definiáltuk, hogy azok az eredeti tüzelési szabály mellett pontosan annyi súlyt visznek egy átmenet tüzelésekor a komplemens helyre, amennyit az átmenet



21.4. ábra. Korlátos kapacitású helyek.

elvesz az eredeti helyről, illetve pontosan annyi súlyt vesznek el a komplemens helyről, amennyit az átmenet az eredetire tesz. A komplemens helyek bevezetése transzformáció tehát azt eredményezi, hogy a hely és a hozzá tartozó új komplemens hely súlyösszege a háló működése során állandó, a komplemens hely a szabad kapacitást tartja nyilván. A háló invariáns tulajdonsága, hogy a hely és a komplemens hely összege éppen a megadott kapacitáskorlát (helyinvariáns). Mivel helyek súlyértéke mindig nemnegatív, ezért sem az eredeti helyen, sem a komplemens helyen nem lehet sohasem több súly, mint a kapacitáskorlát. Az eredeti tüzelési szabály alkalmazásával működő háló a kimenő helyeknek megfelelő komplemens bemenő helyek használatával akadályozza meg a kapacitáskorlát túllépését. Nem aktivizálható egy átmenet a transzformációval kapott hálóban, ha nincs a bemenő komplemens helyen súly, azaz az eredeti kimenő helyen a súlyok száma elérte a kapacitáskorlátot. Belátható a következő tétel:

21.6. tétel. (korlátos kapacitású helyek kiküszöbölése). *(N, M_0) átlátszó, korlátos kapacitású háló, szigorú működési szabállyal. (N', M_0') az (N, M_0)-ból komplemens hely transzformációval kapott háló. Ekkor (N, M_0) és (N', M_0') hálók ekvivalensek, a lehetséges akciósorozataik megegyeznek.*



21.5. ábra. Konkurencia és szinkronizáció.

Gyakorlatok

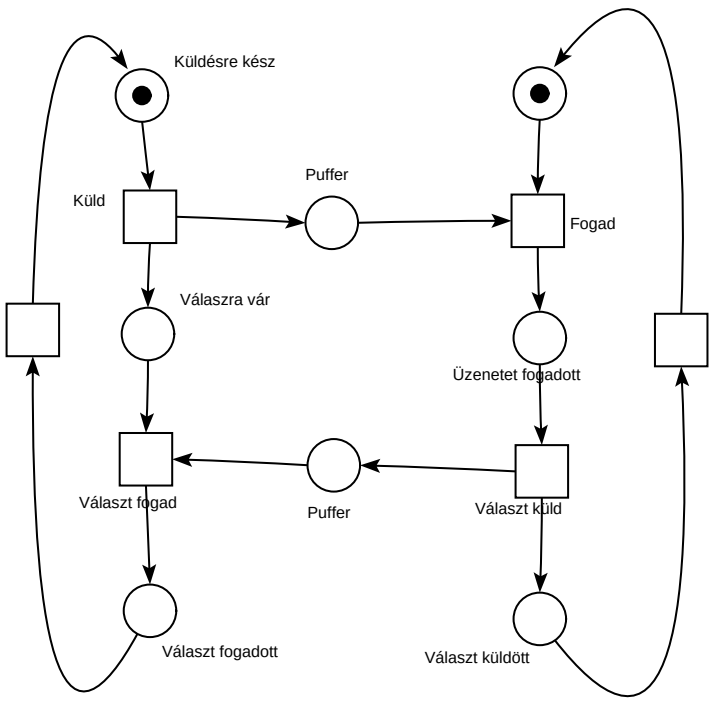
21.2-1. Mutassunk példát három olyan átmenetsorozatra, amely eltér egymástól abban az esetben, ha ugyanarra a hálóra az eredeti, a szigorú, illetve a gyenge működési szabályt alkalmazzuk.

21.2-2. Mutassuk meg, hogy a komplemens hely transzformációval készített háló mindig tüzelhet az alapértelmezés szerint érvényes működési szabálynak megfelelően, ha az eredeti háló tüzelhet a szigorú szabály szerint.

21.3. Párhuzamos folyamatok együttműködése

Párhuzamos és elosztott programok esetén a folyamatok közötti kapcsolatok leírásakor gyakran találkozunk néhány jellegzetes struktúrával. Az egyik ilyen példa a `Par Begin Par End` programszerkezet (21.5. ábra), amelyet Dijkstra vezetett be és széles körben használnak párhuzamos algoritmusok pszeudokóddal történő leírásakor.

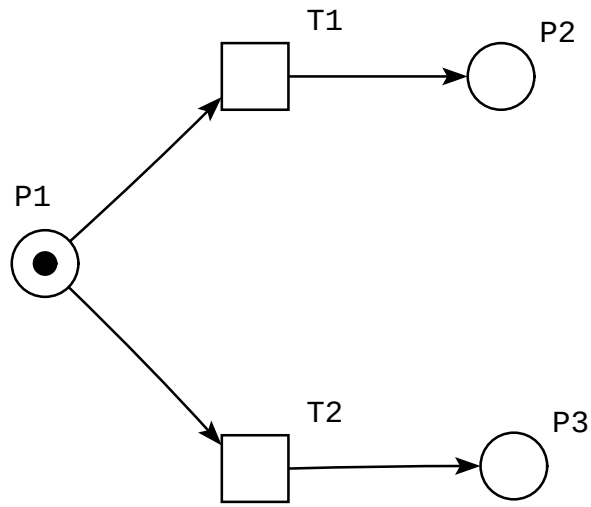
Ebben az esetben két alapelem együttes megjelenését figyelhetjük meg. A T_2-vel és T_3-mal jelölt átmenetek előfeltételei egyaránt a „Par Begin” címkéjű átmenettől függenek, a feltételek egyidejűleg teljesülnek. A T_2 és



21.6. ábra. Egyszerű protokoll.

T_3 átmenetek egymástól függetlenül, aszinkron módon futó folyamatokat modelleznek, melyek viszonya a konkurencia szóval jellemezhető, azaz egymás működését nem befolyásolhatják. Ezzel szemben a T_4 átmenet csak a két folyamat szinkronizációja után tüzelhet, a P_3 és a P_4 helyen is szüksége van egy-egy súlyra.

A 21.6. ábra egy egyszerű kommunikációs protokollt mutat be. A bal oldali irányított kör a szinkron kommunikációt kezdeményező folyamatot modellezi. A *Küld* átmenet hatására egy üzenet kerül a pufferbe, amelyet a jobb oldali irányított körrel reprezentált folyamat kiolvas, majd az üzenet fogadását nyugtázza. A küldő folyamat bevárja a nyugtát és ezután folytatja tevékenységét.

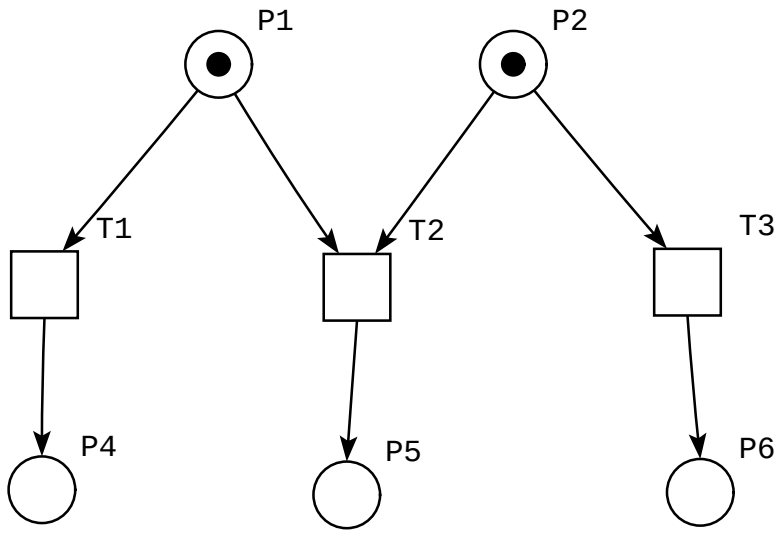


21.7. ábra. Konfliktus.

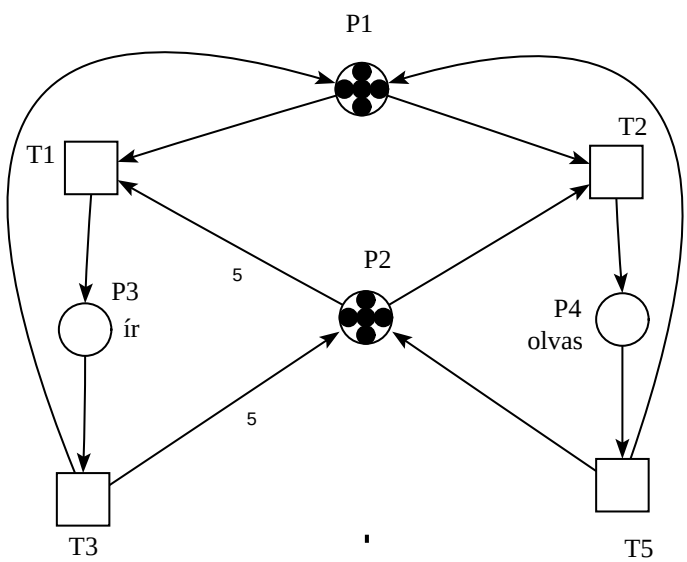
A 21.7. ábrán két folyamat, T_1 és T_2 konfliktushelyzetét figyelhetjük meg. Az előfeltételüket reprezentáló P_1 helyen csak egy súly található, amelyre azonban mindkettőnek szüksége lenne a tüzeléshez. Ezt a helyzetet már megfigyelhettük az étkező filozófusok esetében is, ahol a villa használatakor jelentkezett konfliktus.

Bonyolultabb esetekben a konfliktus és a konkurencia egyszerre figyelhető meg. A 21.8. ábrán a T_1 és T_2 átmenetek konkurensek, de a T_2 és T_3 átmenetek konfliktusban vannak. Ezt a helyzetet ***zavarnak*** hívjuk. Megosztva használt adatok esetén egyidejűleg többen is olvashatják az adatokat, de egyidejű írás és olvasás, vagy egyidejűleg több folyamat által végzett írási műveletek nem megengedettek. Ezt a helyzetet modellezi a 21.9. ábrán látható Petri-háló. A modell egyidejűleg öt olvasó folyamatnak vagy egyetlen egy író folyamatnak ad lehetőséget a kritikus szakaszba történő belépésre, az adatokhoz való hozzáférésre. A P_1 hely a várakozó folyamatok számát, a P_5 pedig a szemafor értékét mutatja. Írási művelethez a T_1 átmenet tüzelésére van szükség, ennek előfeltétele, hogy a P_2 helyen mind az öt súly jelen legyen. Olvasási műveletet a T_2 átmenet tüzelése reprezentál, amelyhez a P_2 helyről egyetlen súly is elegendő. P_4 mutatja a kritikus szakaszban tartózkodó olvasó folyamatok számát.

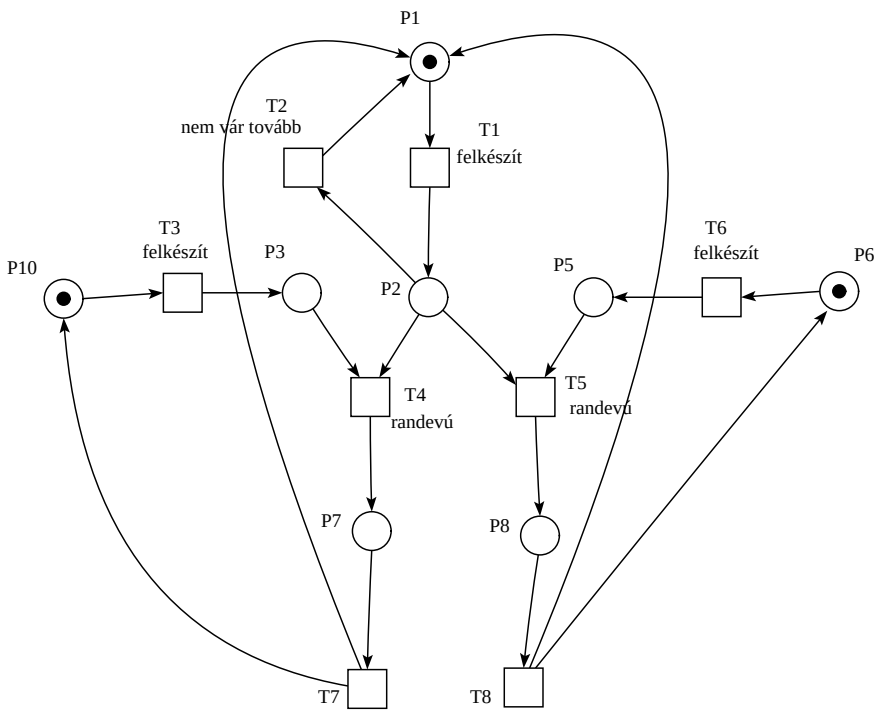
A 21.10. ábrán az Ada programozási nyelvből ismert randevú időhöz kötött szelektív várakozásának modelljét figyelhetjük meg. A modell három folyamat együttműködését mutatja be, a T_1 és T_2 átmeneteket tartalmazó kör-



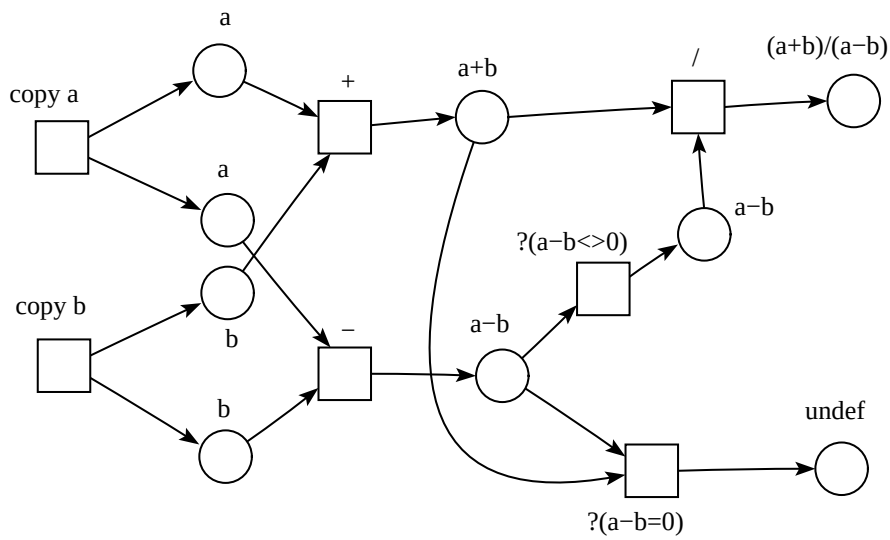
21.8. ábra. Zavar.



21.9. ábra. Író-olvasó szinkronizáció kölcsönös kizárással.



21.10. ábra. Szelektív, időhöz kötött várakozás.



21.11. ábra. Adatfolyamszámítás.

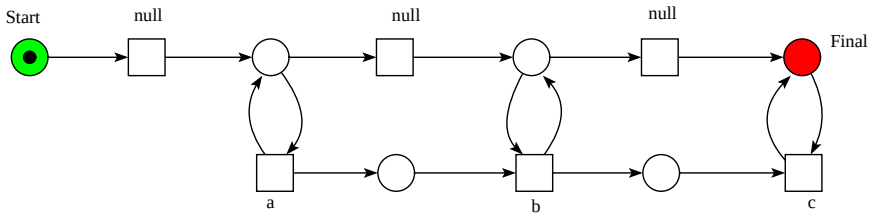
rel modellezett folyamat randevúzik T_3, T_4, T_7 vagy a T_6, T_5, T_8 átmeneteket tartalmazó folyamatok egyikével. Randevú akkor jöhet létre, ha a T_4, illetve a T_5 átmenetek bemenő helyein egyidejűleg súly jelenik meg. Ha mindhárom partner kész a randevúra, akkor konfliktushelyzet áll elő, mert a P_2 helyen lévő egyetlen súlyra mindkét randevúhoz szükség lenne, de egyszerre csak az egyikhez használható fel.

A 21.11. ábra egy adatfolyam segítségével megfogalmazott számítás Petri-hálós modelljét mutatja be. Adatfolyam modell esetén a számítási folyamat adatvezérelt, azaz ha az átmenetekhez tartozó függvények argumentumai a bemenő helyeken megjelennek, akkor az átmenet tüzelésének eredményeként a függvény értéke a kimenő helyre kerül. Az ábrán az $(a+b)/(a-b)$ kifejezés értékének kiszámítását követhetjük nyomon.

Minden véges állapotú gép leírható Petri-hálóval, így Petri-hálókkal is modellezhetjük formális nyelvek felismerési és generálási szabályait oly módon, hogy egyes átmenetekhez szimbólumokat rendelünk hozzá és azok az akciósorozatok felelnek meg a nyelvhez tartozó szavaknak, amelyek végén a háló nem tartalmaz súlyt. Példaként bemutatjuk az $a^n b^n c^n$, $n \in N$ szavakat elfogadó Petri-hálót.

Gyakorlatok

21.3-1. Terjesszük ki a Petri-hálókat a tiltó élek bevezetésével. A tiltó éleket tartalmazó Petri-hálók tüzelési szabálya a következőképpen módosul: ha egy



21.12. ábra. Nyelvelfogadás.

helytől egy átmenethez tiltó él vezet, akkor az átmenet csak akkor tüzelhet, ha a hely súlyozatlan. (A tiltó élek növelik a Petri-hálók kifejezőerejét. A tiltó éleket is tartalmazó modell Turing-teljes.) Tiltó élek berajzolásával egészítsük ki az Ada randevút modellező Petri-hálót oly módon, hogy a T_1, T_2 körrel leírt folyamat csak akkor kerülhesse el a randevút, ha egyik partnere sem kész arra.

21.3-2. Készítsük el annak a rendszernek a modelljét, amelyben két termelő-fogyasztó folyamatpár működik és a második fogyasztó csak akkor fogadhat üzenetet, ha az első fogyasztó puffere üres (prioritásos termelő-fogyasztók).

21.4. Viselkedési tulajdonságok

Ebben az alfejezetben a Petri-hálók dinamikus viselkedésének olyan tulajdonságait vizsgáljuk, melyek szerkezetük mellett függnek kezdősúlyozásuktól is.

A tulajdonságok vizsgálatához az akciósorozat fogalmát használjuk. Párhuzamos programok tulajdonságainak vizsgálatához általában önmagában nem elegendő sem a program által érintett állapotok sorozatának ismerete, sem csak az egymás után tüzelő átmeneteket tartalmazó akciósorozat ismerete. Ezért az akciósorozatot gyakran kiterjesztjük oly módon, hogy abban az átmeneteken kívül az állapotokat leíró súlyvektorokat is feltüntetjük.

21.4.1. Jelölések

Tegyük fel, hogy az M_0 súlyozásból a ς akciósorozattal elérhető az M_n súlyozás: $M_0\,[\varsigma>\,M_n$, ahol $\varsigma = t_1, t_2, \ldots, t_n$ akciósorozat.

Az összes érintett állapotot és átmenetet tartalmazó kiterjesztett akciósorozat jelölése: $M_0\,[t_1>\,M_1\,[t_2>\,M_2 \ldots M_{n-1}\,[t_n>\,M_n$.

$L(N, M)$ – azon akciósorozatok halmaza, amely az N-ben elérhető az M súlyozásból.

$R(N, M)$ az N hálóban az M súlyozásból elérhető súlyozások halmaza, az M ún. továbbsúlyozó osztálya.. $R(N, M) = \{M' | \exists \varsigma \in L(N, M) : M [\varsigma > M'\}$. $R(N, M)$-et röviden $R(M)$-mel vagy $\overrightarrow{M}$-mel jelöljük, ha N egyértelműen adott.

Jelölje $\#(\varsigma, t)$ a t átmenet előfordulásainak számát ς-ban.

Az a kérdés eldönthető, hogy egy adott súlyozás elérhető-e, (azaz $M \in R(N, M_0)$ teljesül-e). Az azonban, hogy két adott háló ekvivalens-e az akciósorozatokra nézve, $(L(N_1, M_{1,0}) = L(N_2, M_{2,0}))$ általában **nem** eldönthető.

21.7. definíció. (k-korlátosság, biztonságosság). *Az* $k \in \mathcal{N}$. *Az* (N, M_0) *Petri-háló* ***k-korlátos,*** *ha* $\forall M \in R(N, M_0) : \forall p \in P : M(p) \leq k$. *Egy Petri-háló* ***biztonságos,*** *ha 1-korlátos.*

k-korlátos Petri-hálók esetében a korlát univerzális, azaz minden helyre egységesen ugyanaz a felső korlát érvényes, eltérően a kapacitáskorláttal rendelkező hálóktól, ahol a kapacitáskorlát a hely függvényében változhat.

21.8. definíció. (elevenség). *Legyen* $t \in T$. M_0 *kezdősúlyozástól függően az* N *Petri-hálóban a* t *átmenet*

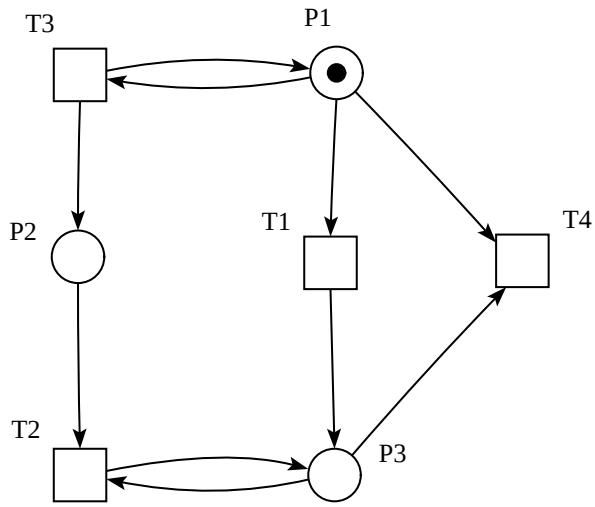
- L_0 ***szinten eleven*** *(holt), ha* $\forall \varsigma \in L(N, M_0) : t \notin \varsigma$,
- L_1 ***szinten eleven*** *(aktivizálható), ha* $\exists \varsigma \in L(N, M_0) : t \in \varsigma$,
- L_2 ***szinten eleven,*** *ha* $\forall k \in \mathcal{N} : \exists \varsigma \in L(N, M_0) : \#(\varsigma, t) \geq k$,ű
- L_3 ***szinten eleven,*** *ha* $\exists \varsigma \in L(N, M_0) : \#(\varsigma, t) = \infty$,
- L_4 ***szinten eleven,*** *ha* $\forall M \in R(M_0)$*-ra a* t L_1 *szinten eleven.*

21.9. tétel. *Ha egy* t *átmenet* L_4 *szinten eleven, akkor* L_3 *szinten is az.*

A négyes szintű elevenség szerint minden elérhető állapot után is létezik olyan akciósorozat, hogy abban újra előfordul az átmenet. Ez azt jelenti, hogy meg tudunk konstruálni egy olyan akciósorozatot, amelyben végtelen sokszor fordul elő. A tétel megfordítására ellenpélda a 21.13. ábrán látható Petri-háló T_3 átmenete.

21.10. tétel. *Ha egy* t *átmenet* L_3 *szinten eleven, akkor* L_2 *szinten is az.*

Ha létezik egy olyan akciósorozat, amelyben az átmenet végtelen sokszor fordul elő, akkor tetszőleges k természetes számhoz tudunk mutatni olyan sorozatot, amelyben legalább k-szor fordul elő az átmenet. A tétel megfordítására ellenpélda a 21.13. ábrán látható Petri-háló T_2 átmenete.



21.13. ábra. L_1, L_2 és L_3 eleven átmenetekkel rendelkező háló. (Lehetséges akciósorozatok: t_1; t_3! (végtelen sokszor t_3); $t_3(k)$,$t_2(k)$ (ahol k tetszőleges egész szám).

21.11. tétel. *Ha egy t átmenet L_2 szinten eleven, akkor L_1 szinten is az.*

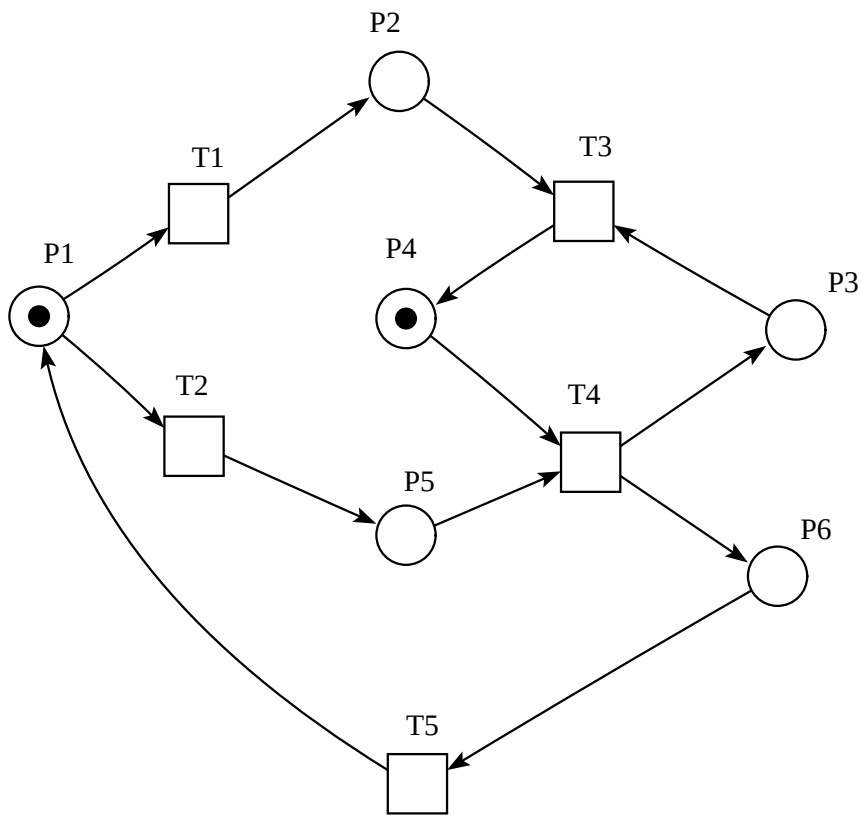
Ha tetszőleges k természetes számhoz tudunk mutatni olyan sorozatot, amelyben legalább k-szor fordul elő az átmenet, akkor $(k = 1)$-re is létezik ilyen. A tétel megfordítására ellenpélda a 21.14. ábrán látható Petri-háló T_1 átmenete.

21.12. tétel. *Egy átmenet L_1 szinten akkor és csak akkor eleven, ha nem holt.*

21.13. definíció. *Azt mondjuk, hogy egy átmenet szigorúan L_k eleven, ha L_k szinten eleven, de nem eleven L_{k+1} szinten. Egy (N, M_0) Petri-hálót L_k elevennek mondunk, ha $\forall t \in T : t$ L_k eleven.*

21.14. definíció. (visszatérési képesség, otthonállapot). *Azt mondjuk, hogy egy Petri-háló rendelkezik a* ***visszatérési képességgel,*** *ha minden elérhető állapotból létezik olyan akciósorozat, amellyel a kezdőállapot elérhető. Formálisan: (N,M_0) Petri-háló esetén $\forall M \in R(M_0) : M_0 \in R(M)$.*
Otthonállapotról *beszélünk, ha van egy olyan, nem feltétlenül a kezdőállapottal azonos állapot, amelybe a háló tetszőleges elérhető állapotból el tud jutni. Formálisan: M' otthonállapot, ha $\forall M \in R(M_0) : M' \in R(M)$.*

Megvizsgálhatjuk, hogy egy tetszőlegesen megválasztott súlyozás



21.14. ábra. Biztonságos, szigorúan L_1 eleven háló. (Lehetséges akciósorozatok: t_1; t_2,t_4,t_5,t_2; t_2,t_4,t_5,t_1,t_3.)

lefedhető-e, azaz található-e olyan elérhető állapot, amelyben minden helyen legalább ekkora súly van.

21.15. definíció. (lefedhetőség). *(N, M_0) Petri-háló esetén az M súlyozás* ***lefedhető,*** *ha $\exists M' \in R(M_0) : \forall p \in P : M'(p) \geq M(p)$.*

Tetszőleges $t \in T$ átmenetre legyen M az a minimális súlyozás, amely mellett t végrehajtható. Ekkor t L_1 szinten akkor és csak akkor eleven, ha M lefedhető, illetve akkor és csak akkor holt, ha M nem lefedhető.

21.16. definíció. (perzisztencia). *Az (N,M_0) Petri-hálót* ***perzisztensnek*** *nevezzük, ha tetszőleges $t_1, t_2 \in T$ állapotpárra teljesül, hogy ha t_1 és t_2 is engedélyezett (aktivizálható), akkor (közvetlenül) t_1 aktivizálása után t_2 továbbra is engedélyezett (aktivizálható) marad.*

Ha egy perzisztens hálóban egy átmenet aktivizálhatóvá válik, akkor mindaddig az marad, amíg végre nem hajtódik.

A 21.15., 21.16., ..., 21.22. ábrák segítségével megmutatjuk, hogy az elevenség, k-korlátosság, visszatérési képesség egymástól független tulajdonságok.

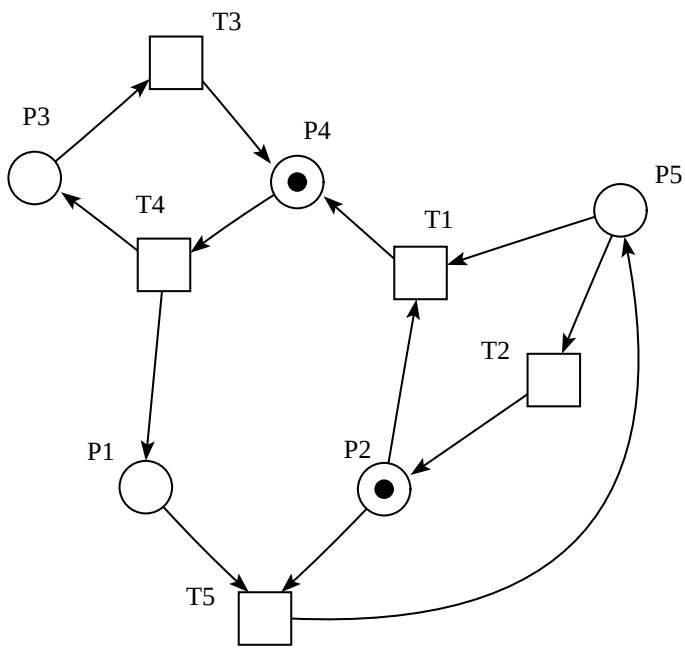
Petri-hálók átmeneteinek ok-okozati függetlenségének mértékéről ad tájékoztatást a szinkronizációs távolság. Meghatározásakor azt vizsgáljuk, hogy a két átmenethalmaz elemeinek előfordulásainak különbsége legfeljebb mekkora lehet a hálóhoz tartozó (véges) akciósorozatokban. Az előfordulások számának legnagyobb eltérése nem feltétlenül a kezdőállapotból induló akciósorozatok esetén figyelhető meg, így a szinkronizációs távolság kiszámításakor az összes elérhető állapotból induló akciósorozatot figyelembe kell venni.

Szinkronizációs távolságot speciális esetben két átmenetre is megadhatunk.

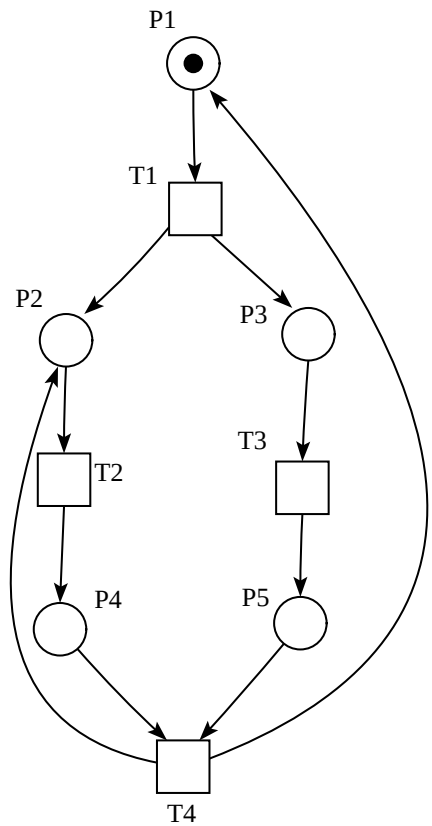
21.17. definíció. (szinkronizációs távolság). *$(N, M_0) : t_1, t_2 \in T$ esetén* $d_{1,2} := \max_{\varsigma \in L(N,M), M \in R(M_0)} |\#(\varsigma, t_1) - \#(\varsigma, t_2)|$.

21.18. definíció. (szinkronizációs távolság (2)). *$(N, M_0) : E_1, E_2 \subseteq T$ esetén*
$d_{E_1,E_2} := \max_{\varsigma \in L(N,M), M \in R(M_0)} |\#(\varsigma, E_1) - \#(\varsigma, E_2)|$.

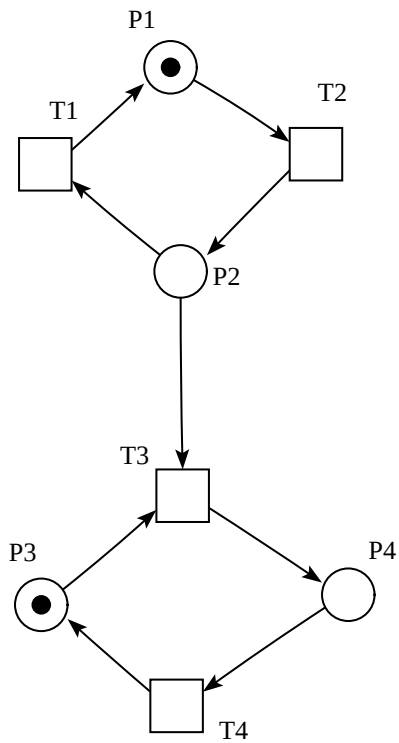
A szinkronizációs távolságot megjeleníthetjük grafikusan új helyek bevezetésével is. Ezek a helyek csak illusztrációként szolgálnak, szervesen nem kapcsolódnak a háló működéséhez. Az illusztrációs hely rákövetkező átmenetei aktivizálhatóak lehetnek abban az esetben is, ha a helyen nincs



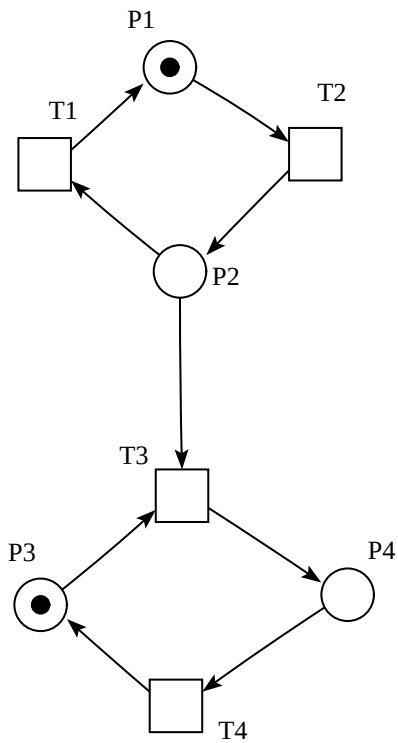
21.15. ábra. Példa nem korlátos (P_1 hely), nem eleven (T_1 átmenet), visszatérési képességgel rendelkező hálóra.



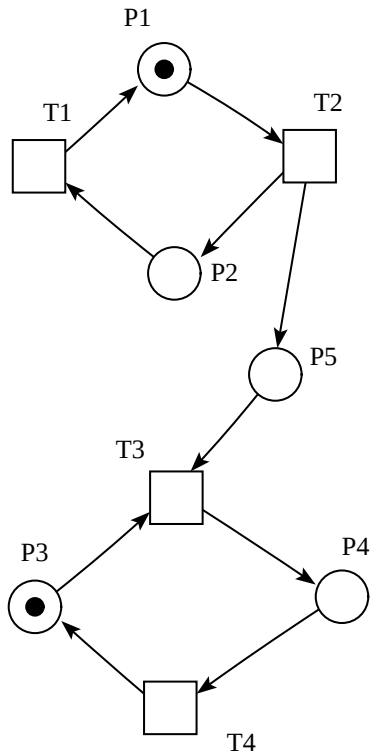
21.16. ábra. Példa nem korlátos (P_2 hely), eleven, perzisztens, visszatérési képességgel nem rendelkező hálóra.



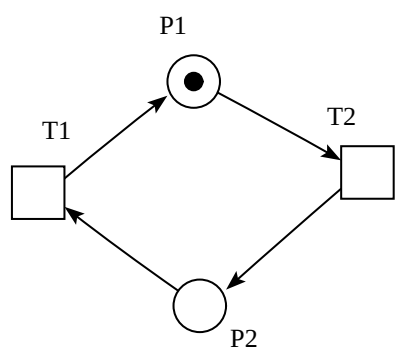
21.17. ábra. Példa 1-korlátos (biztonságos), nem eleven, visszatérési képességgel nem rendelkező *AC* hálóra. (Lehetséges akciósorozatok: $(t_1, t_2)(k),t_3,t_4.$)



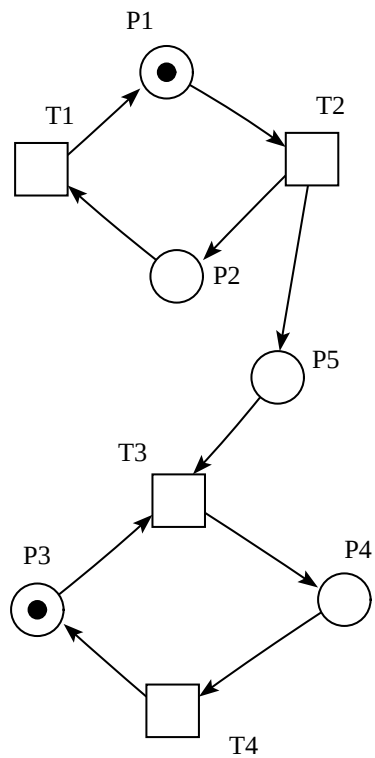
21.18. ábra. Példa 1-korlátos (biztonságos), nem eleven, visszatérési képességgel nem rendelkező *AC* hálóra. (Lehetséges akciósorozatok: (t_1, t_2)(k),t_3,t_4.)



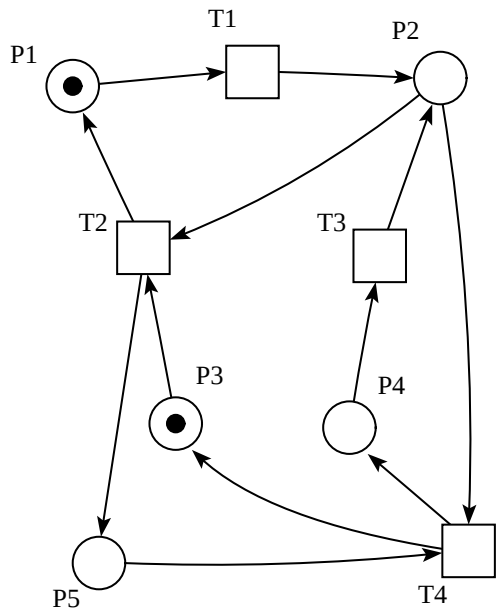
21.19. ábra. Példa nem korlátos (P_5), eleven, visszatérési képességgel rendelkező MG hálóra.



21.20. ábra. Példa 1-korlátos (biztonságos), eleven, visszatérési képességgel nem rendelkező AC hálóra. (Lehetséges akciósorozatok: t_1, t_2,(t_1, t_4,t_3,t_2)!.)



21.21. ábra. Példa nem korlátos (P_5), eleven, visszatérési képességgel rendelkező MG hálóra.



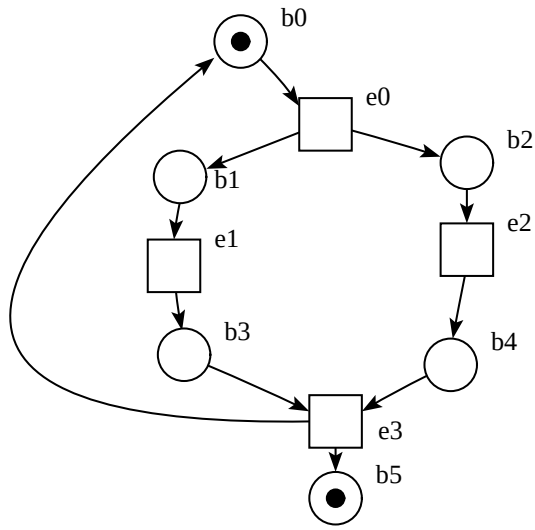
21.22. ábra. Példa nem korlátos (P_3), nem eleven (T_4), visszatérési képességgel nem rendelkező $\overline{AC}(p_2, p_4)$ hálóra. (Lehetséges akciósorozatok: (t_1,t_3,t_2)!.)

súly. Egy ilyen hely az egyik átmenethalmazhoz tartozó tüzelések számának a másik átmenethalmazhoz tartozó tüzelések számához viszonyított többletét jeleníti meg. Az egyik halmaz átmenetei tüzeléskor elhelyeznek egy súlyt az új helyen, a másik halmaz átmenetei pedig elvesznek egy súlyt erről a helyről (ha van rajta súly). Mivel egy-egy ilyen hely csak az egyik irányú többletet jeleníti meg, két adott átmenethalmaz szinkronizációs távolságának meghatározásához két új helyet kell bevezetnünk. A két helyen megjelenő legnagyobb súlyértékek maximuma adja a szinkronizációs távolságot.

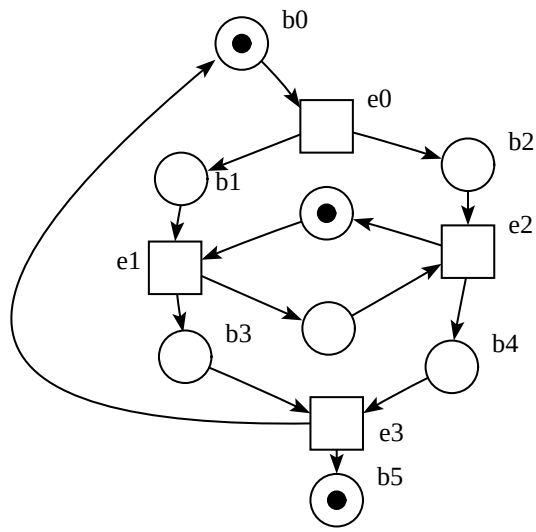
Például az E_1 és E_2 átmenethalmazok szinkronizációs távolságának mérésére bevezethetjük az $s_{1,2}$ és $s_{2,1}$ helyeket, ahol ${}^\bullet s_{1,2} = E_1, s_{1,2}^\bullet = E_2$, illetve ${}^\bullet s_{2,1} = E_2, s_{2,1}^\bullet = E_1$.

A pártatlanság fogalma általában az események ütemezéséhez kapcsolódik és csak végtelen működés esetén van értelme vizsgálni. A Petri-háló a működési szabállyal együtt egy absztrakt végrehajtási modellt határoz meg, így a szokásos értelemben nem beszélhetünk pártatlanságról.

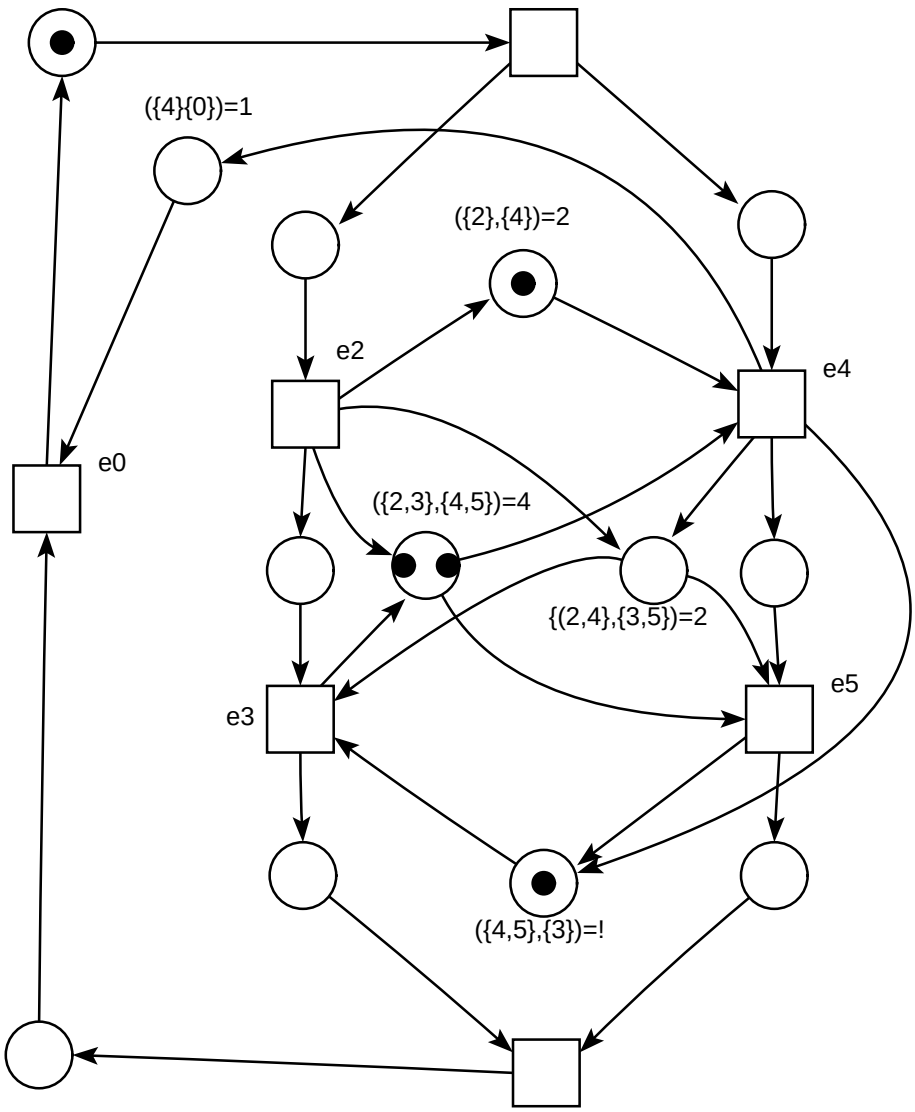
Megvizsgálhatjuk azonban, hogy a Petri-háló által megengedett végtelen akciósorozatokban két átmenet egymáshoz való viszonya milyen. Előfordulhat-e az valamely akciósorozatban, hogy az egyik átmenet akár végtelen sokszor is előfordul, míg a másik egyszer sem?



21.23. ábra. Az e_1 és e_2 átmenetek szinkronizációs távolsága 2 ($d(1,2) = 2$). (Pl. e_1,e_2,e_3,e_0,e_2 akciósorozat.)



21.24. ábra. Az e_1 és e_2 átmenetek szinkronizációs távolsága 1 ($d(1,2) = 1$).



21.25. ábra. Példa különböző szinkronizációs távolságokra.

21.19. definíció. (pártatlanság).

1. *Egy háló* ***korlátosan pártatlan*** t_i, t_j*-re nézve, ha bármely akciósorozatban* t_i *előfordulásainak száma felülről korlátos* t_j *következő előfordulásáig és viszont.*

2. *A háló* ***korlátosan pártatlan****, ha tetszőleges* $t_i, t_j \in T$ *átmenetpárra nézve a háló korlátosan pártatlan.*

Megfogalmazhatunk pártatlansági követelményt az egyes akciósorozatokra is. Egy akciósorozat szigorúan pártatlan, ha véges vagy minden átmenet végtelen sokszor fordul elő benne.

21.20. definíció. (szigorú pártatlanság).

1. $\forall M \in R(M_0) : \forall \varsigma \in L(N, M)$ *esetén* ς ***feltétlenül szigorúan pártatlan****, ha* $\forall t_j \in T : \#(\varsigma, t_j) = \infty$ *vagy* ς *véges.*

2. (N, M_0) *Petri-háló* ***feltétlenül szigorúan pártatlan****, ha* $\forall M \in R(M_0) : \forall \varsigma \in L(N, M) : \varsigma$ *feltétlenül szigorúan pártatlan.*

Petri-hálók pártatlanságát tehát kétféleképpen is definiáltuk. A 21.21. ábrán látható hálóra mindkét pártatlansági követelmény teljesül, a 21.22. ábra hálójára egyik sem. A két követelmény azonban csak akkor ekvivalens, ha a háló korlátos (21.7. def.).

21.21. tétel. *Egy korlátosan pártatlan háló szigorúan pártatlan is.*

Ha bármely két átmenetre teljesül, hogy az egyik csak véges sokszor fordulhat elő úgy, hogy a másik nem fordul elő, akkor egy végtelen átmenetsorozatban minden átmenet végtelen sokszor fordul elő. A tétel megfordítása nem igaz. Előfordulhat két átmenet egy végtelen sorozatban végtelen sokszor úgy, hogy az egyik mindig eggyel többször fordul elő a másik következő előfordulásáig. Ezt az esetet mutatja be a 21.16. ábra jobb oldalán látható Petri-háló. Ebben a hálóban T_2 véges, de nem korlátos sokszor hajtható végre a T_3 következő előfordulásáig, attól függően, hogy hány súly halmozódott fel már a P_2 helyen.

21.22. tétel. *Ha egy háló korlátos és szigorúan pártatlan, akkor korlátosan pártatlan is.*

Gyakorlatok

21.4-1. Létezik-e olyan Petri-háló, amelyben t_1 és t_2 egy korlátosan pártatlan átmenetpár, szinkronizációs távolságuk mégsem véges szám?

21.5. Petri-hálók vizsgálata

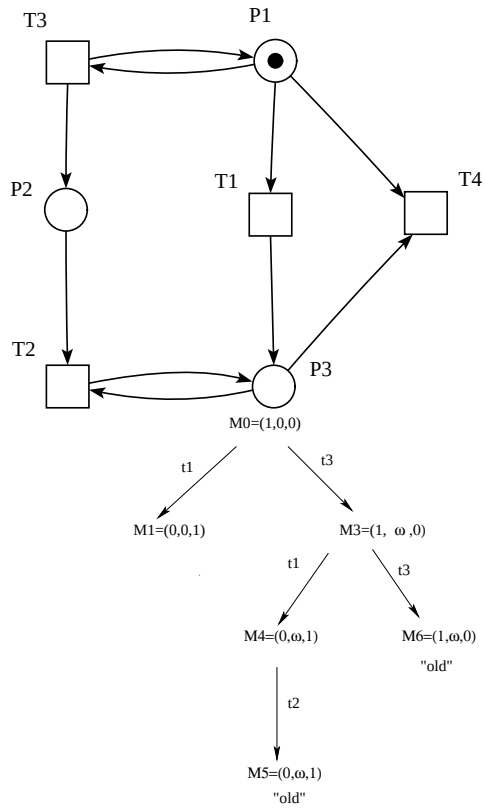
Többféleképpen is vizsgálhatjuk Petri-hálók tulajdonságait. Az egyik lehetőség, hogy előállítjuk a háló elérhető állapotait leíró gráfot. Elkészíthetjük a háló fedési fáját, illetve gráfját, amelyben a csúcspontokat állapotokkal, az éleket átmenetekkel címkézzük. A gyökér címkéje a kezdőállapot. A csúcspontokat összekötő irányított élek a megengedett átmeneteknek felelnek meg.

Elindulunk a gyökérből és a megengedett átmenetek segítségével rendre új élekkel és csúcspontokkal bővítjük a fát. Az összes elérhető állapotot tartalmazó gráf azonban végtelen nagy is lehet, ha a háló nem korlátos (21.7. definíció). Elérhetőségi fa helyett ezért a fedési fát készítjük el, amelyben a súlyok ciklikus növekedését kizárjuk. Ha egy olyan csúcspontot találunk, amelyhez tartozó állapot már szerepel a gyökérből hozzá vezető úton, akkor ezt a csúcspontot már nem terjesztjük ki. Ha egy új csúcspont címkéjében szereplő súlyozás fedi a gyökérből hozzá vezető út egyik csúcspontjának súlyozását, de különbözik attól, akkor az új csúcs súlyvektorát módosítjuk. Azokra pozíciókra, ahol az új csúcshoz tartozó súlyérték nagyobb, a valódi súlyérték helyett a végtelen nagy súlyt jelző ω jelet tesszük. Az elérhető állapotok közül nem jelenik meg mindegyik a fában, de minden elérhető állapothoz található azt fedő súlyvektor.

21.5.1. Elérhetőségi és fedési fa

21.23. definíció. (elérhetőségi fa). *Az (N, M_0) Petri-háló elérhetőségi (nem korlátos háló esetén fedési) fája olyan gráf, amelyben a csúcsok súlyozásokkal vannak címkézve, az élek pedig átmenetekkel és ezt a gráfot az alábbi módon állítjuk elő:*

1. *új* $:= \{M_0\}$
2. *ciklus, amíg új* $\neq \emptyset$
   - (a) $M :\in$ *új*, *új* $:=$ *új* $\setminus \{M\}$
   - (b) *ha M-ig a gyökértől létezik már M címkéjű csúcs* $\Rightarrow$ *M régi.*
   - (c) *ha M-ben nincs megengedett átmenet* $\Rightarrow$ *M zsákutca.*
   - (d) *M-ben $\forall t$ megengedett átmenetre kiszámoljuk $M\,[t\!>\,M'$-t.*
     - i. *Ha a gyökértől M-ig $\exists M'' : M''$-t lefedi M'-t és $M' \neq M''$, akkor minden $p \in M' : M'(p) > M''(p)$ helyre: $M'(p) := \omega$.*
     - ii. *M' új csúcs: új* $:=$ *új* $\cup \{M'\}$, *az él címkéje t lesz.*



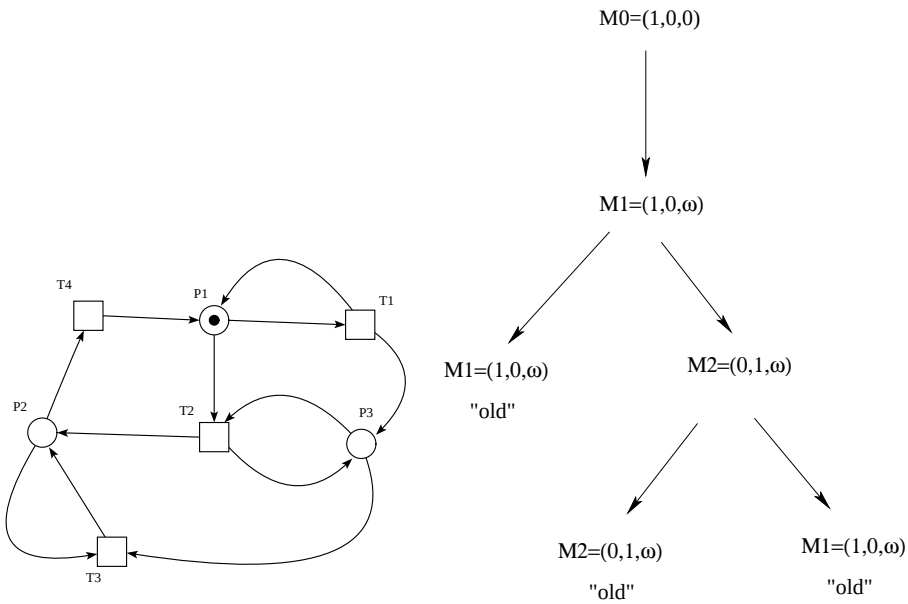
21.26. ábra. I. példa fedési fára.

A fedési fa alkalmas arra, hogy a háló korlátosságát vizsgáljuk és arra is, hogy eldöntsük, hogy egy átmenet holt-e.

21.24. tétel. *Legyen G a (N, M_0) Petri-háló elérhetőségi fája.*
Korlátos a Petri-háló akkor és csak akkor, ha nincs G-ben ω címkéjű csúcs.
1-korlátos (biztonságos) a Petri-háló akkor és csak akkor, ha kizárólag 0 és 1 súlyérték szerepel a címkékben.
A háló t átmenete holt (L_0 eleven) akkor és csak akkor, ha nincs t élcímke a G fedési fában.

21.25. tétel. *Ha $M \in R(M_0)$, akkor $\exists M'$-vel címkézett csúcs úgy, hogy M' lefedi M-et.*

Az azonos címkéjű helyek összevonásával a fedési fából fedési gráf készíthető. A fedési fa, illetve a fedési gráf azonban nem tartalmaz minden elérhető



21.27. ábra. II.a. példa fedési fára.

állapotot, így eltérő tulajdonságú hálók fedési fája is lehet azonos. A 21.27. háló eleven, a 21.28. háló nem eleven, fedési fájuk mégsem különbözik.

21.5.2. Elérhetőség szükséges feltételének meghatározása

Átlátszó hálók esetén lineáris algebrai eszközök is használhatók annak vizsgálatára, hogy egyes állapotok elérhetőek-e. Az alábbiakban egy szükséges feltételt adunk elérhetőségre.

21.26. definíció. *Legyen n az átmenetek, m a helyek száma. Legyen $A \in \mathbb{Z}^{n\times m}$ mátrix olyan, hogy*

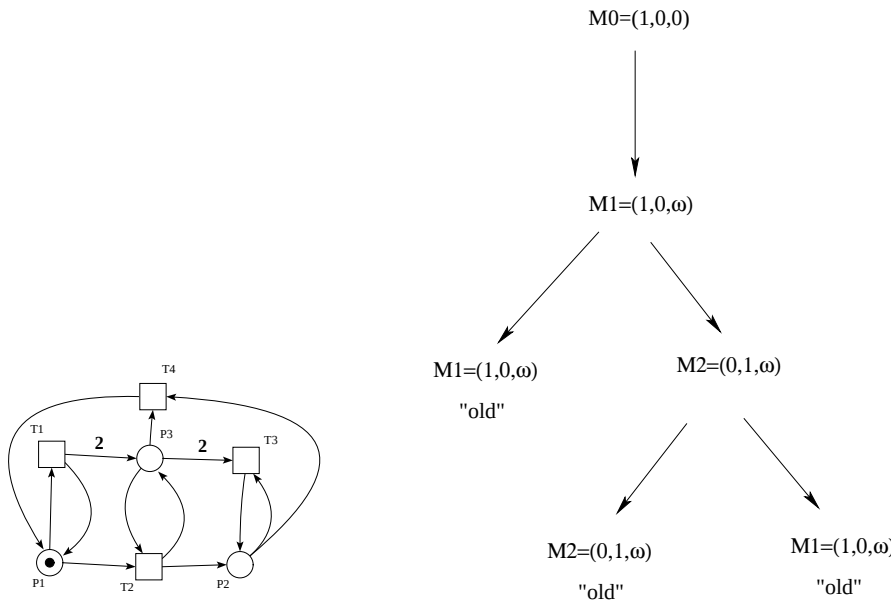
$$a_{ij} = a_{ij}^{+} - a_{ij}^{-} ,$$

ahol

$$a_{ij}^{+} = v(t_i, p_j) ,$$

$$a_{ij}^{-} = v(p_j, t_i) .$$

A ς akciósorozathoz tartozó $M_k \in \mathbb{N}^{m\times 1}$ $(k = 0, \ldots, |\varsigma|)$ vektorok tartalmazzák a k-adik átmenet aktivizálása utáni súlyozásokat. Legyenek az $u_k \in$



21.28. ábra. II.b. példa fedési fára.

$\mathbb{N}^{n\times 1}$ $(k = 1\,..\,|\varsigma|)$ *kontroll-vektorok olyanok, hogy*

$$(u_k)_i = \begin{cases} 1, & ha\ \varsigma_k{=}t_i\ , \\ 0 & különben\ . \end{cases}$$

Ekkor a következő összefüggéseket lehet felírni:

$$M_k = M_{k-1} + A^T u_k\ ,$$

$$M_d = M_0 + A^T \sum_{k=1}^{d} u_k\ .$$

Legyen $\Delta M = M_d - M_0 = A^T x$, ahol $x = \sum_{k=1}^{d} u_k$. Ekkor ha M_d elérhető M_0-ból, akkor $\exists x \in \mathbb{N}^{n\times 1} : \Delta M = A^T x$.

21.27. tétel. $\exists x \in \mathbb{R}^{n\times 1} : \Delta M = A^T x \Leftrightarrow \forall y \in \mathbb{R}^{m\times 1} : Ay = 0 :< \Delta M, y >= 0$ *(ahol* $< \Delta M, y >$ *a skaláris szorzatot jelöli).*

Bizonyítás. $\Rightarrow$ (Elégségesség)
Feltehető, hogy $\Delta M \neq 0$. Legyen $y \in \mathbb{R}^m : Ay = 0$. Mivel alkalmas x-szel $\Delta M = A^T x$, ezért $y^T \Delta M =< \Delta M, y >= y^T A^T x = (Ay)^T x = 0$.

$\Leftarrow$ (Szükségesség)
Legyenek az $a_1, \ldots, a_n \in \mathbb{Z}^{1\times m}$ vektorok az A mátrix sorvektorai. Azt kell

belátni, hogy $\exists x : \Delta M = A^T x$, azaz $\Delta M^T \in \mathcal{L}(a_1, \ldots, a_n)$ (ahol $\mathcal{L}$ a lineáris kombinációt jelöli).

Definiáljuk a következő halmazokat:

$$Y_0 := \{y \in \mathbb{R}^{m\times 1} \,|a_i y =< y^T, a_i >= 0 \quad (i = 1 \ldots n)\} ,$$

$$Y_1 := \mathcal{L}(a_1, \ldots, a_n) ,$$

$$Y_2 := \{y \in \mathbb{R}^{m\times 1} \,|\forall z \in Y_1 :< z, y >= 0\} .$$

Ekkor $Y_0 = Y_2$, ui. $y \in Y_0 \Leftrightarrow Ay = 0 \Leftrightarrow \forall \alpha_1, \ldots, \alpha_n \in R :< y^T, \alpha_1 a_1 + \ldots + \alpha_n a_n >= 0 \Leftrightarrow \forall z \in Y_1 :< z, y >= 0 \Leftrightarrow y \in Y_2$.

A feltételek szerint $\forall y \in Y_2 :< \Delta M, y >= 0$. Már csak azt kell belátni, hogy $\forall x \in \mathbb{R}^m \forall y \in Y_2 :< x, y >= 0 \Rightarrow x \in Y_1$. Ehhez indirekt módon tegyük fel, hogy $\exists x \in \mathbb{R}^m \setminus Y_1 \forall y \in Y_2 :< x, y >= 0$. Mivel $\mathbb{R}^m = Y_1 \oplus Y_2$, ezért $\exists a \in Y_1, b \in Y_2 \setminus \{0\} : x = a + b$. Ezt felhasználva az indirekt feltevés szerint $< a + b, y >=< a, y > + < b, y >=< b, y >= 0$. Mivel y választása tetszőleges volt, legyen most y egyenlő b-vel. Ekkor viszont $\|b\|^2 = 0$, ami (a normált tér axiómái miatt) azzal ekvivalens, hogy $b = 0$. Ezzel pedig ellentmondásra jutottunk indirekt feltevésünkkel.

A tétel feltételét felhasználva, valamint az előző állítást $x = \Delta M$-re alkalmazva a bizonyítandó állítást kapjuk. ∎

Legyen M_d elérhető M_0-ból (ekkor $\exists x \in \mathbb{N}^n : \Delta M = A^T x$). Legyen $r = \rho(A)$ az A rangja. Sor és oszlopcserékkel az A mátrix a következő alakra transzformálható:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} ,$$

ahol $A_{11} \in \mathbb{Z}^{r\times(m-r)}, A_{12} \in \mathbb{Z}^{r\times r}, A_{21} \in \mathbb{Z}^{(n-r)\times(m-r)}, A_{22} \in \mathbb{Z}^{(n-r)\times r}$ és $|A_{12}| \neq 0$. Az A mátrix mellett hasonló módon ΔM-et átalakítva az eredeti egyenletrendszerrel ekvivalens egyenletrendszert kapunk. Legyen $\Delta M = \begin{bmatrix} \Delta M_1 \\ \Delta M_2 \end{bmatrix}$, ahol $\Delta M_1 \in \mathbb{N}^{m-r}$ és $\Delta M_2 \in \mathbb{N}^r$. Transzponáljuk az átalakított egyenletrendszert:

$$\left[\Delta M_1^T, \Delta M_2^T\right] = \left[x_1^T A_{11} + x_2^T A_{21}, x_1^T A_{12} + x_2^T A_{22}\right] ,$$

ahol $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ és $x_1 \in \mathbb{N}^r, x_2 \in \mathbb{N}^{n-r}$. Mivel $\rho([A_{11}, A_{12}]) = r$, így $[A_{11}, A_{12}]$ sorai lineáris kombinációként előállítják $[A_{21}, A_{22}]$ sorait. Ebből következik, hogy $\forall x_2 \exists \widehat{x_2} : x_2^T [A_{21}, A_{22}] = \widehat{x_2}^T [A_{11}, A_{12}]$. Ezt felhasználva:

$$\left[\Delta M_1^T, \Delta M_2^T\right] = x_1^T [A_{11}, A_{12}] + \widehat{x_2}^T [A_{11}, A_{12}] = \widehat{x}^T [A_{11}, A_{12}] ,$$

ahol $\widehat{x}^T = x_1^T + \widehat{x_2}^T$. Az egyenletrendszert szétbontva:

$$\Delta M_1^T = \widehat{x}^T A_{11},$$

$$\Delta M_2^T = \widehat{x}^T A_{12} \Leftrightarrow \Delta M_2^T A_{12}^{-1} = \widehat{x}^T.$$

Az első egyenletrendszerben $\widehat{x}^T$ helyébe $\Delta M_2^T A_{12}^{-1}$-et helyettesítve kapjuk:

$$\Delta M_1^T = \Delta M_2^T A_{12}^{-1} A_{11} \Rightarrow \Delta M_1 = A_{11}^T (A_{12}^T)^{-1} \Delta M_2 \ .$$

Ekkor

$$\left[I_{m-r}, -A_{11}^T (A_{12}^T)^{-1}\right] \begin{bmatrix} \Delta M_1 \\ \Delta M_2 \end{bmatrix} = 0 \ .$$

21.28. tétel. *Legyen* $B_f = \left[I_{m-r}, -A_{11}^T (A_{12}^T)^{-1}\right]$. *Ha* M_d *elérhető* M_0*-ból, akkor* $B_f \Delta M = 0$.

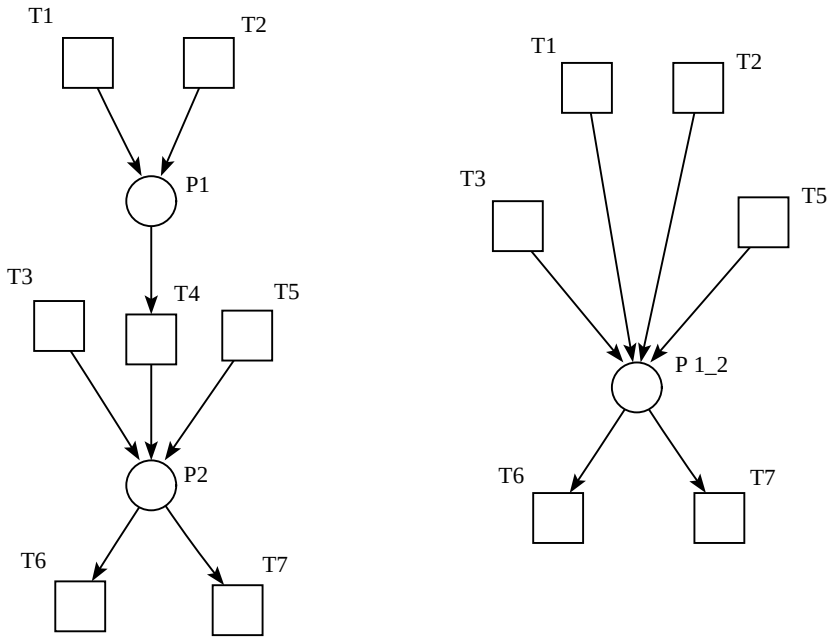
Gyakorlatok

21.5-1. Rajzoljuk fel az evő filozófusok viselkedését modellező Petri-háló fedési fáját és fedési gráfját.

21.5-2. Határozzuk meg a 21.13. és 21.14. ábrák két Petri-hálójának fedési fáját és fedési gráfját.

21.6. Elevenséget, biztonságosságot és korlátosságot megőrző transzformációk

Minél kevesebb csúcspontot és élet tartalmaz egy Petri-háló gráfja, annál könnyebb a háló tulajdonságait megállapítani. Az alábbi, lokális transzformációk alkalmazásával lépésenként egyszerűsíthetjük a vizsgált Petri-hálót oly módon, hogy az eredményül kapott hálók és az eredeti háló elevenségét és korlátosságát leíró tulajdonságok nem változnak. Egy (N, M) Petri-hálóból kapott (N', M') Petri-háló akkor és csak akkor eleven, korlátos, illetve biztonságos, ha (N, M) rendelkezik a megfelelő tulajdonsággal. Ezen transzformációk alkalmazása emeli az absztrakció szintjét. A transzformációk külső kommunikációs és szinkronizációs kapcsolatokkal nem rendelkező szekvenciális részfolyamatok belső struktúrájának részleteit leíró részgráfokat töröl-nek a Petri-hálóból. Részgráfok törlése egyben azt eredményezi, hogy a folyamat működésének kevésbé részletgazdagabb modelljéhez jutunk. A transzformációk megfordításának alkalmazásával pedig úgy finomíthatjuk egy folyamat modelljét, hogy az elevenség, korlátosság, biztonságosság megmarad. A transzformációkat bemutató ábrákon látható gráfok nem feltétlenül teljes



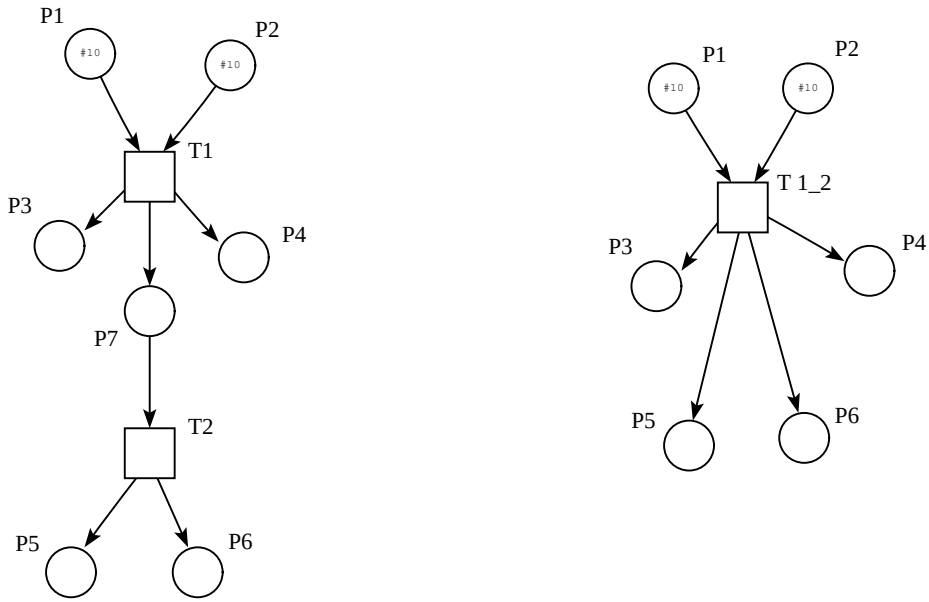
21.29. ábra. Helyek sorozatának összevonása.

Petri-hálók, hanem azok részgráfjai is lehetnek.

A helyek sorozatának összevonása transzformáció (21.29. ábra) segítségével két helyet (P_1, P_2) és az őket összekötő átmenetet (T_4) helyettesíthetjük egyetlen hellyel (P_{1_2}). Indirekt módon belátható, hogy ha mindkét régi hely k-korlátos volt, akkor és csak akkor az új hely is k-korlátos. Ha ugyanis a P_{1_2} helyen több, mint k súly jelenne meg, akkor ezek a súlyok megoszthatóak lettek volna az eredeti hálóban a T_4 átmenet működésének alkalmas megválasztásával úgy, hogy mindegyik a P_1 helyen, illetve a P_2 helyen van. A törölt T_4 átmenet attól függően eleven vagy sem, hogy a P_1 helyet megelőző átmenetek között van-e eleven. Az átmenet eltűnésével tehát a háló elevensége sem változik.

A helyek sorozatának összevonása transzformáció duálisa az átmenetek sorozatának összevonása (21.30. ábra) transzformáció. A P_7 hely akkor és csak akkor k-korlátos, ha a rákövetkező átmenet kimenő helyei (P_5 és P_6) is k-korlátosak. A hely törlésével tehát a háló korlátossága nem változik. A T_2 átmenet, illetve a két átmenet összevonásával keletkező átmenet is pontosan akkor eleven, ha a T_1 átmenet eleven. Az átmenetek összevonásával tehát a háló elevensége sem változik.

A párhuzamos helyek összevonása transzformációt mutatja be a 21.31.



21.30. ábra. Átmenetek sorozatának összevonása.

ábra. A helyek (P_5,P_6) az átmenet különböző előfeltételeit modellezik.

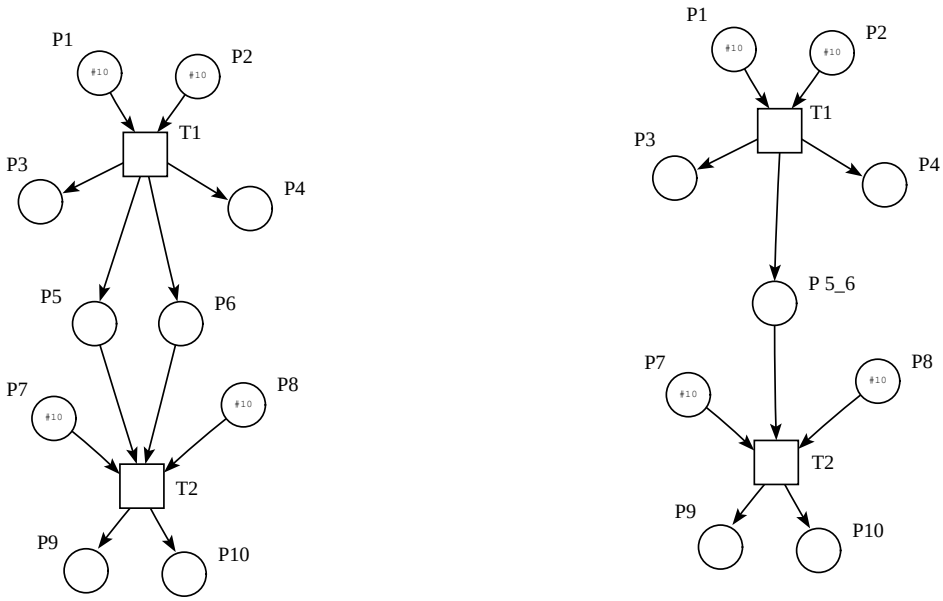
Ezek a feltételek azonban teljesen egyenértékűek a háló viselkedése szempontjából, így egyetlen átmenettel helyettesíthetőek.

A párhuzamos átmenetek összevonása transzformációt mutatja be a 21.32. ábra. Ha az átmenetek bármelyike tüzel, akkor a P_1 helyről a P_2-re helyez át súlyt. A modellezett folyamat szempontjából a két átmenet különböző eseményeknek felel meg, de a Petri-háló viselkedése szempontjából egyenértékűek és egyetlen átmenettel helyettesíthetőek.

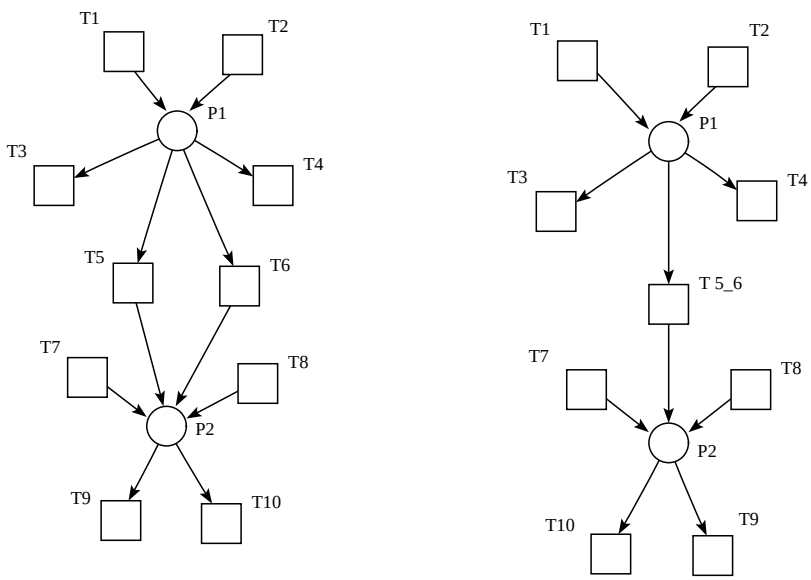
A 21.33. és a 21.34. ábrákon látható transzformációk hurkok elhagyására adnak lehetőséget. Ha egy hely súlyozott, egyetlen átmenethez kapcsolódik, annak egyszerre bemenő és kimenő helye, akkor az átmenet viselkedését nem befolyásolja. Ha egy átmenet egyetlen helyhez kapcsolódik és az onnan elvett súlyokat a tüzelése eredményeként visszahelyezi, akkor a háló többi részének viselkedésére nem hat ki a működése.

Gyakorlatok

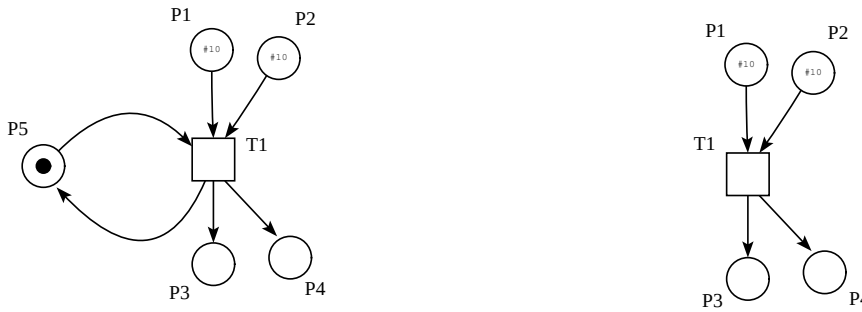
21.6-1. Állapítsuk meg a 21.17. ábra bal oldali hálójának elevenségi és biztonságossági tulajdonságait az alfejezetben ismertetett transzformációkkal.



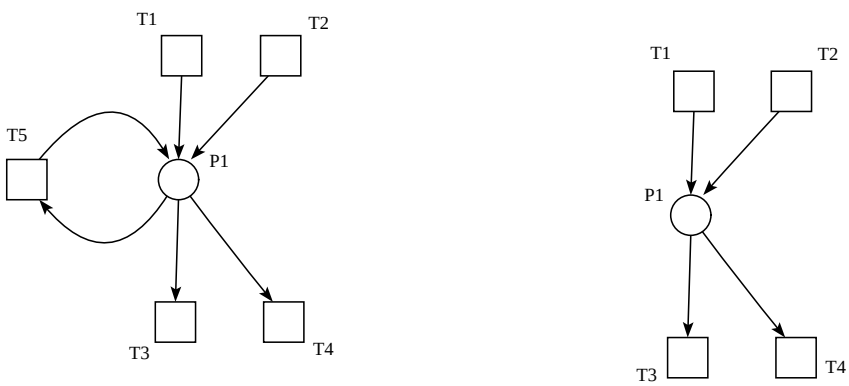
21.31. ábra. Párhuzamos helyek összevonása.



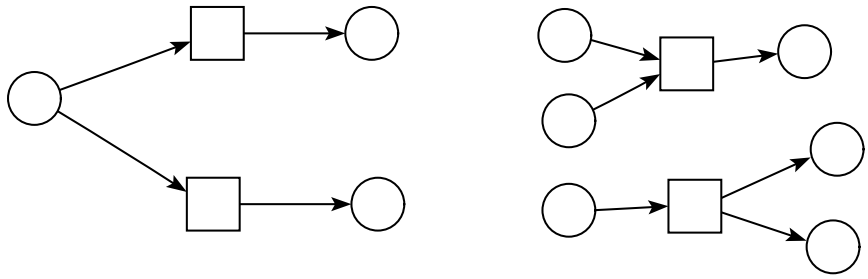
21.32. ábra. Párhuzamos átmenetek összevonása.



21.33. ábra. Hurok helyek elhagyása.



21.34. ábra. Hurok átmenetek elhagyása.



21.35. ábra. Példa állapotgépre, illetve állapotgépben nem megengedett részgráfok.

21.7. Petri-hálók osztályozása

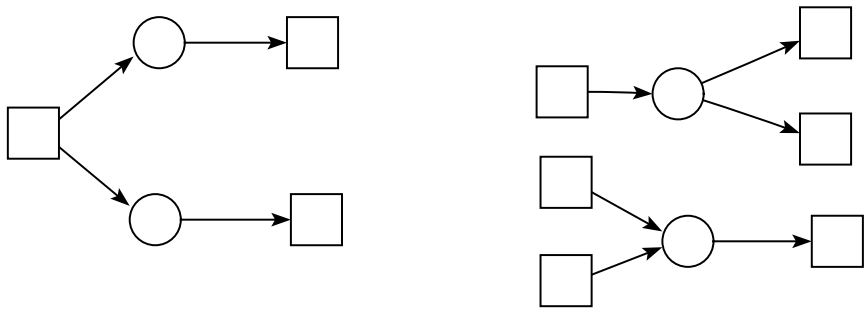
A Petri-hálók szerkezeti felépítése nagyon sokféle lehet, a háló viselkedése a struktúrája alapján általában nehezen határozható meg. Egyes részosztályokra azonban megadhatóak olyan könnyen ellenőrizhető feltételek, amelyek alapján könnyen eldönthető, hogy az adott osztályba tartozó hálók elevenek-e, illetve biztonságosak-e. Részosztályokat a normális hálók részhalmazán belül definiálunk, ahol minden él súlya egy. A részosztályok meghatározása alapvető programozási fogalmakhoz kapcsolódik: a konfliktushoz, illetve a szinkronizációhoz.

21.29. definíció. (állapotgép). *Az **állapotgép** olyan Petri-háló, melyben minden átmenetnek pontosan egy megelőző és egy rákövetkező helye van* $(\forall t \in T : |{}^\bullet t| = |t^\bullet| = 1)$.

Egy állapotgépben nem fordulhat elő a 21.35. ábra jobb oldalán látható részgráfok egyike sem. Állapotgépben minden átmenetnek egyetlen hellyel reprezentált előfeltétele lehet csak, ily módon az állapotgép szinkronizációt nem enged meg. Ha csak egyetlen súly van a hálóban, akkor működése egy szekvenciális véges állapotú gépnek, a gép állapotai pedig egy-egy súlyozott helynek felelnek meg.

21.30. definíció. (jelzett gráf). *A **jelzett gráf** olyan Petri-háló, melyben minden helynek pontosan egy megelőző és egy rákövetkező átmenete van* $(\forall p \in P : |{}^\bullet p| = |p^\bullet| = 1)$.

Ha a Petri-háló egy-egy helyéből és a beléje mutató, illetve a belőle induló éléből egyetlen összetett élet készítünk, akkor a jelzett gráf reprezentálható olyan gráffal is, amelyben az egymást követő átmeneteket kötjük össze közvetlenül. Ebben az esetben a Petri-háló helyein megjelenő súlyokat



21.36. ábra. Példa jelzett gráfra, illetve jelzett gráfban nem megengedett részgráfok.

a gráf megfelelő élein jelezzük, innen származik a részosztály elnevezése. Egy jelzett gráfban nem fordulhat elő a 21.36. ábra jobb oldalán látható részgráfok egyike sem, tehát nem alakulhat ki konfliktus. Minden jelzett gráf perzisztens kezdősúlyozásától függetlenül. Megjegyezzük, hogy nem kizárólag jelzett gráf lehet konfliktusmentes, de perzisztens és biztonságos háló jelzett gráffá konvertálható.

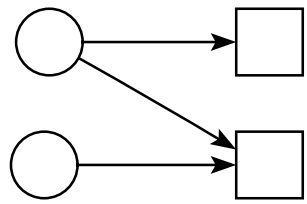
Az állapotgépek és a jelzett gráfok osztályának metszete nem üres, van olyan Petri-háló, amelyik mindkettőnek eleme (21.22. ábra hálója). A szabad választású hálók az állapotgépek és a jelzett gráfok általánosításai, konfliktust és szinkronizációt is megengednek, de ezek lokálisan együttes megjelenését, a zavart (21.7 ábra) nem.

21.31. definíció. (szabad választású háló (FC)). *Szabad választású (FC) a háló, ha* $\forall p \in P : |p^\bullet| \leq 1$ *vagy* $^\bullet(p^\bullet) = \{p\}$. *Másképpen:* $\forall p_1, p_2 : p_1^\bullet \cap p_2^\bullet \neq \emptyset \Rightarrow |p_1^\bullet| = |p_2^\bullet| = 1$.

Ehhez a részosztályhoz tartozó hálók esetén az egy helyről kiinduló minden él vagy a hely egyetlen kimenő éle, vagy egy átmenet egyetlen bemenő éle. A szabad választású és a kiterjesztett szabad választású hálók elnevezése azt a tulajdonságukat tükrözi, hogy két, közös bemenő hellyel rendelkező átmenet esetén nincs olyan súlyozás, amelyiknél csak az egyik megengedett, tehát ezen átmenetek közül szabadon választhatunk, hogy melyik tüzeljen. A 21.37. ábrán bemutatott struktúra szabad választású hálóban nem fordulhat elő, így zavar sem jöhet létre.

21.32. definíció. (kiterjesztett szabad választású (EFC) háló). *Egy háló kiterjesztett szabad választású (EFC) háló, ha* $\forall p_1, p_2 : p_1^\bullet \cap p_2^\bullet \neq \emptyset \Rightarrow p_1^\bullet = p_2^\bullet$.

Az aszimmetrikusan választó (vagy más néven egyszerű) hálók esetén még tovább gyengítjük a feltételeket és csak annyit követelünk, hogy két olyan



21.37. ábra. Szabad választású hálóban nem megengedett részgráfok.

átmenet esetén, ahol a bemenő helyek halmazainak metszete nem üres, az egyik halmaz tartalmazza a másikat. A 21.38. ábrán láthatjuk az egyes osztályok között fennálló kapcsolatokat.

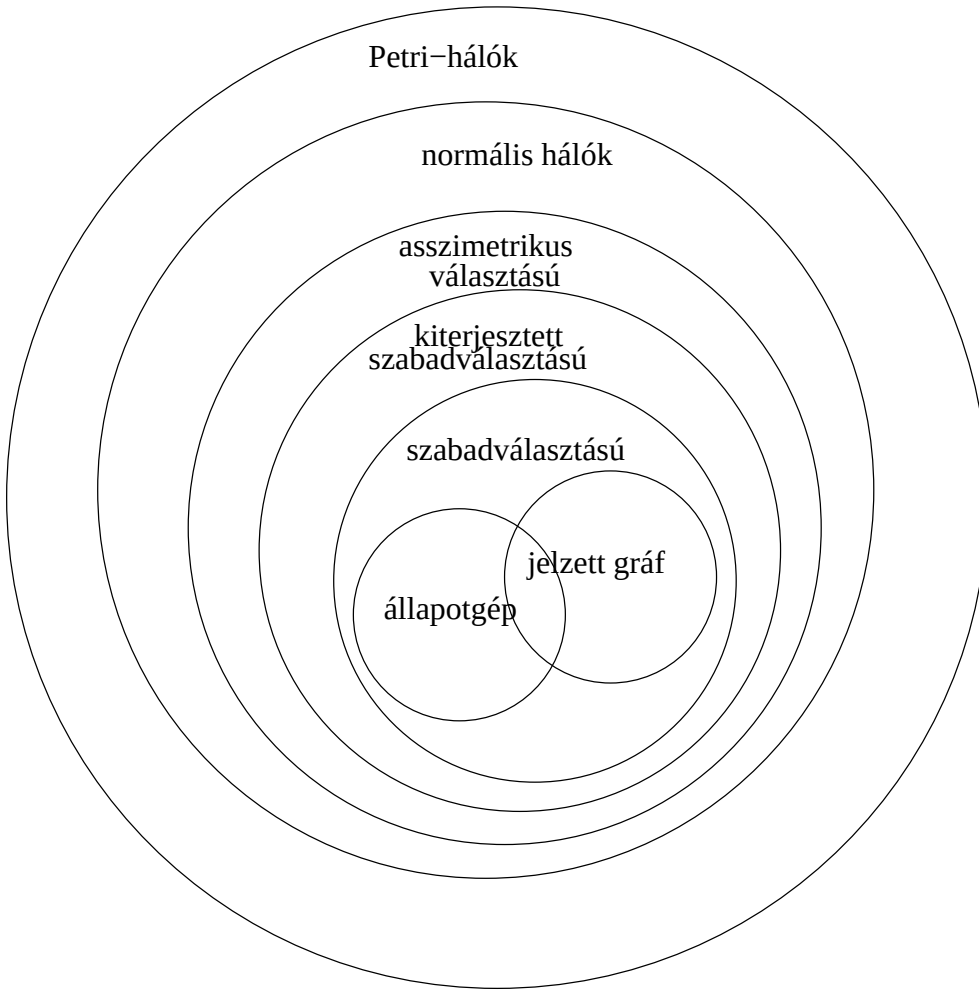
21.33. definíció. (aszimmetrikusan választható (AC) háló). *Egy háló aszimmetrikusan választható (AC) háló, ha* $\forall p_1, p_2 : p_1^\bullet \cap p_2^\bullet \neq \emptyset \Rightarrow p_1^\bullet \subseteq p_2^\bullet \vee p_2^\bullet \subseteq p_1^\bullet$

Petri-hálók tulajdonságainak elemzéséhez hasznos alapfogalom a szifon, a csapda és a helyinvariáns. Mindhárom helyek részhalmazára vonatkozik. A szifon olyan helyhalmaz, amelyikre teljesül, hogy minden olyan átmenetnek, amelyiknek van a halmazhoz tartozó kimenő helye, annak van a halmazhoz tartozó bemenő helye is. Ez azt jelenti, hogy a szifonba súlyt helyező átmenetek a szifonból el is vesznek súlyt. Ha egy szifon sima, azaz nem tartalmaz súlyt, akkor sima is marad. Nem lehetnek elevenek azok az átmenetek, amelyeknek van olyan bemenő helye, amely sima vagy simává váló szifonhoz tartozik. A 21.47. lemma ad magyarázatot arra, hogy miért szokták a szifont holtpontnak is nevezni. Egy helyinvariáns egy olyan helyhalmaz, amelyen a súlyösszeg állandó. Helyinvariáns alkalmazására mutatott példát a 21.6. tétel bizonyítása.

21.34. definíció. (szifon/holtpont). *Helyek* S *halmaza* ***holtpont/szifon****, ha* ${}^\bullet S \subseteq S^\bullet$.

A csapda olyan helyhalmaz, amelyikre teljesül, hogy minden olyan átmenetnek, amelynek van a halmazhoz tartozó bemenő helye, annak van a halmazhoz tartozó kimenő helye is. Ez azt jelenti, hogy a csapdából súlyt eltávolító átmenetek a csapdába helyeznek is súlyt. Ha egy csapda súlyozott, akkor súlyozott is marad. Ez az csapdához csatlakozó átmenetek elevensége szempontjából kedvező helyzet.

21.35. definíció. (csapda). *Helyek* S *halmaza* ***csapda****, ha* $S^\bullet \subseteq {}^\bullet S$.



21.38. ábra. Petri-hálók osztályai.

A definíciókból következik, hogy csapdák, szifonok uniója csapda, illetve szifon. Egy szifon vagy csapda alapszifon, illetve alapcsapda, ha nem más szifonok, illetve csapdák uniója. Minimális egy szifon vagy csapda, ha semmilyen valódi része nem szifon, illetve csapda.

A Petri-hálók részosztályai között tár fel kapcsolatokat a duális és a fordított háló fogalma. Egy háló fordítottja az a háló, amelyet úgy kapunk, hogy az összes irányított élet megfordítjuk.[2] Egy háló duálisának nevezzük azt a hálót, amelyet úgy kapunk, hogy a gráfban az átmeneteket helyekkel, a helyeket átmenetekkel helyettesítjük. Jelzett gráf duálisa az állapotgép. Egy szabadválasztású Petri-háló fordított-duálisa is szabadválasztású Petri-háló. A szifon a csapda fordítottja.

Szifonok meghatározása

Legyen $S \subseteq P$ az N Petri-háló helyeinek egy halmaza. Jelölje az s_i logikai változó a következő állítást: $p_i \in S$ $(i = 1 \ldots |P|)$. Legyen az A_i $(i = 1 \ldots |P|)$ predikátum a következő:

$$s_i \to ((s_{(j_1)_1} \vee \ldots \vee s_{(j_1)_{m_1}}) \wedge \ldots \wedge (s_{(j_n)_1} \vee \ldots \vee s_{(j_n)m_n})) \ ,$$

ahol $\{t_{j_1}, \ldots, t_{j_n}\} = {}^\bullet p_i$, $m_k = |{}^\bullet t_{j_k}|$ $(k = 1 \ldots n)$ és $\forall k = 1 \ldots n$: $\{p_{(j_k)_1}, \ldots, p_{(j_k)m_k}\} = {}^\bullet t_{j_k}$.

21.36. tétel. $S = \{p_{l_1}, \ldots, p_{l_{|S|}}\}$ *holtpont* $\Leftrightarrow A_1 \wedge \ldots \wedge A_{|P|} =$ IGAZ *az* $s_{l_1} = s_{l_2} = \ldots = s_{l_{|S|}} =$ IGAZ, $s_{l_{|S|+1}} = s_{l_{|S|+2}} = \ldots = s_{l_{|P|}} =$ HAMIS *igazságértékelés mellett.*

Bizonyítás. Tegyük fel, hogy $\exists i = 1 \ldots |P| : A_i =$ HAMIS. Feltehető, hogy $i \notin \{l_{|S|+1}, \ldots, l_{|P|}\}$ ui. ekkor $A_i = \neg s_i \vee (\ldots) =$ IGAZ. Ha $i \in \{l_1, \ldots, l_{|S|}\}$ (azaz $p_i \in S$), akkor $\neg s_i =$ HAMIS miatt a

$$(s_{(j_1)_1} \vee \ldots \vee s_{(j_1)_{m_1}}), \ldots, (s_{(j_n)_1} \vee \ldots \vee s_{(j_n)m_n})$$

klózok közül legalább az egyik hamis. Legyen ez pl. az

$$(s_{(j_1)_1} \vee \ldots \vee s_{(j_1)_{m_1}})$$

klóz. Ekkor $t_{j_1} \in {}^\bullet p_i \subseteq {}^\bullet S$ (azaz $(t_{j_1}, p_i) \in T \times S$) és $\forall q \in {}^\bullet t_{j_1} : q \notin S$ azaz ${}^\bullet t_{j_1} \bigcap S = \oslash$, amivel ellentmondásra jutottunk.

Tegyük fel, hogy S nem szifon. Ekkor $\exists (t, p_{l_1}) \in T \times S : {}^\bullet t \bigcap S = \oslash$. Legyen $\{p_{j_1}, \ldots, p_{j_m}\} = {}^\bullet t$. Ekkor $s_{j_1} = s_{j_2} = \ldots = s_{j_m} =$ HAMIS és $s_{l_1} =$

[2] A háló fordítottjának (vagy duálisának) nevezzük azt a hálót is, amelyet a megfelelő transzformáció és átcímkézés kompozíciójával kapunk.

IGAZ, ezért

$$s_{l_1} \to ((s_{j_1} \vee s_{j_2} \vee \ldots \vee s_{j_m}) \wedge \ldots) = \text{HAMIS} ,$$

amivel ellentmondásra jutottunk. ■

Tehát az összes olyan igazságértékelés, amely kielégíti az $A_1 \wedge \ldots \wedge A_{|P|}$-t, meghatároz egy szifont. A feladat ezek után ezen igazságértékelések meghatározása. Ezzel ekvivalens feladat azon igazságértékelések megkeresése, melyek nem elégítik ki a $\neg(A_1 \wedge \ldots \wedge A_{|P|})$-t. Ezen igazságértékelések könnyen meghatározhatók úgy, hogy a $\neg(A_1 \wedge \ldots \wedge A_{|P|})$ konjunktív normálformájában (KNF) szereplő összes klózra megkeressük az őt hamissá tevő igazságértékeléseket.

21.1. példa.

$$s_1 \to s_2, s_2 \to s_3 \vee s_4, s_3 \to s_1 \wedge s_2, s_4 \to s_1 .$$

Az ebből kapott KNF:

$$(\neg s_1 \vee s_2) \wedge (\neg s_2 \vee s_3 \vee s_4) \wedge (\neg s_3 \vee s_1) \wedge (\neg s_3 \vee s_2) \wedge (\neg s_4 \vee s_1) .$$

A fenti KNF negáltjának KNF-je:

$$(\neg s_1 \vee \neg s_2 \vee \neg s_3) \wedge (\neg s_1 \vee \neg s_2 \vee \neg s_3 \vee s_4) \wedge (\neg s_1 \vee \neg s_2 \vee \neg s_4)$$

$$\wedge(\neg s_1 \vee \neg s_2 \vee s_3 \vee \neg s_4) \wedge (s_1 \vee s_2 \vee s_3 \vee s_4) .$$

Ebből a megoldások:

s_1	s_2	s_3	s_4
T	T	T	*
T	T	T	F
T	T	*	T
T	T	F	T
F	F	F	F

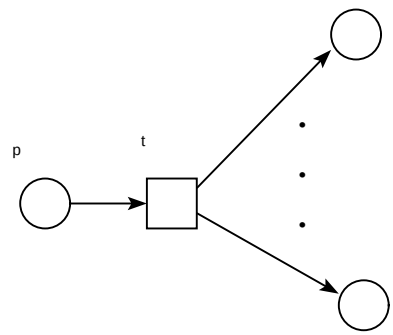
azaz az adott gráf holtpontjai: $\oslash, \{p_1, p_2, p_3, p_4\}, \{p_1, p_2, p_3\}, \{p_1, p_2, p_4\}$.

21.37. megjegyzés. *A tartalmazott klózok a negált KNF-ből elhagyhatók.*

Gyakorlatok

21.7-1. Állapítsuk meg, hogy a 21.15., ..., 21.22. ábrák egyes hálói melyik Petri-háló osztályhoz tartoznak.

21.7-2. Keressünk csapdákat és szifonokat a 21.15. és 21.16. ábrák hálóin.



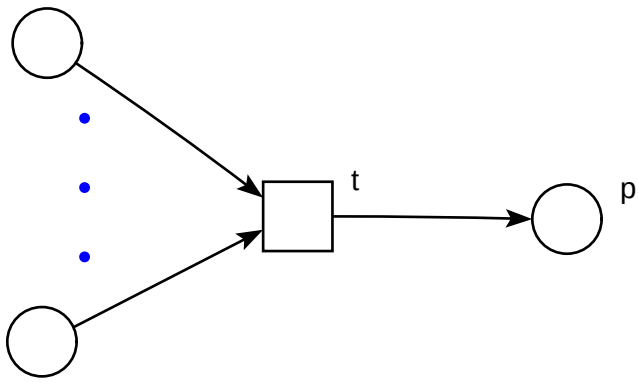
21.39. ábra. Példa.

21.8. Eleven és biztonságos Petri-hálók

Arra a kérdésre szeretnénk választ kapni, hogy adott struktúrájú Petri-hálóhoz létezik-e olyan kezdősúlyozás, hogy a háló eleven és biztonságos egyszerre. Először néhány egyszerűbb szükséges feltételt határozunk meg, majd a részletesebb választ a Petri-hálók egyes osztályaira külön-külön adjuk meg. Nem lehet egy Petri-háló biztonságos is és eleven is, ha forrás vagy nyelő átmenetet vagy olyan helyet tartalmaz, amelyiknek nincs legalább egy-egy bemenő és kimenő átmenete.

A 25.11. ábra bal oldali részén egy olyan helyet látunk, amelyiknek nincs bemenő éle: ${}^\bullet p = \emptyset$. A helyet követő t átmenet nem lehet eleven. Ha a kezdősúlyozás biztonságos és legfeljebb 1 súlyt helyez a P helyre, akkor a t tüzelése után a súly 0-ra csökken. További súly nem kerülhet a p helyre, így a t átmenet legfeljebb L_1 szinten eleven. A 25.11. ábra jobb oldali részén látható Petri-háló p helyének nincs kimenő átmenete: $p^\bullet = \emptyset$. Ha t eleven, akkor p nem lehet biztonságos. Az eleven átmenet újra és újra súlyokat helyez el a p helyen, így a hely semmilyen k természetes számra sem k-korlátos.

A 25.13. ábra bal oldali részén t forrás átmenet: ${}^\bullet t = \emptyset$. A forrás átmenet bármikor tüzelhet, így semmi sem korlátozza azt, hogy a p helyen felhalmozzon súlyokat. A p hely tehát nem biztonságos. A 25.13. ábra jobb oldalán t nyelő átmenet: $t^\bullet = \emptyset$. Ha p biztonságos, akkor a p helyen t lustasága, inaktivitása esetén sem halmozódhatnak fel súlyok. Ez azt jelenti, hogy t nem lehet eleven, nem áll rendelkezésre elegendő súly ahhoz, hogy újra és újra tüzeljen.



21.40. ábra. Példa.

Beláttuk, ha (N, M_0) eleven és biztonságos, akkor $\forall x \in P \cup T : x^\bullet \neq \emptyset$ és ${}^\bullet x \neq \emptyset$. Erősen összefüggőnek nevezünk egy irányított gráfot, ha minden csúcsából minden csúcsába vezet irányított út. Teljesül a következő általánosabb tétel is:

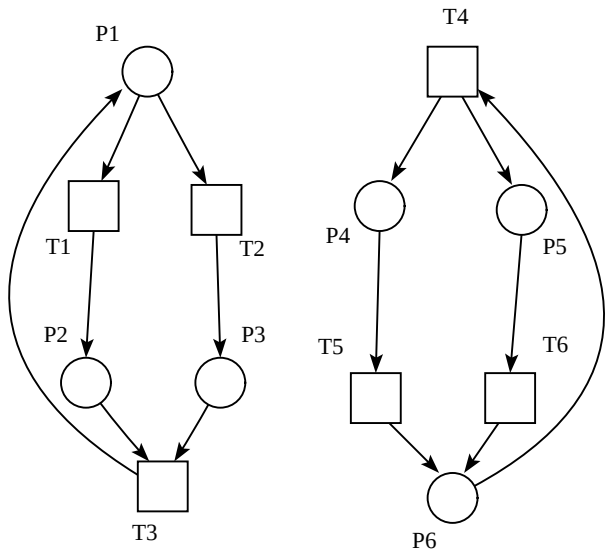
21.38. tétel. *Összefüggő, eleven és biztonságos háló erősen összefüggő.*

A tétel általában nem megfordítható. Nem minden erősen összefüggő hálóhoz találunk eleven és biztonságos kezdősúlyozást. Két ellenpéldát mutat a 21.41. ábra. Az első hálónak nincs olyan kezdősúlyozása, amely mellett eleven, a második hálónak nincs olyan nem teljesen üres kezdősúlyozása, amely mellett biztonságos, pedig mindkettő erősen összefüggő. Állapotgépek és jelzett gráfok esetén azonban belátható, hogy a tétel megfordítható, ha a gráf erősen összefüggő, akkor ezekben az esetekben létezik eleven és biztonságos súlyozás.

21.8.1. Állapotgép eleven és biztonságos súlyozása

Az állapotgép egyszerű szerkezete könnyen megadhatóvá teszi az eleven súlyozás szükséges és elégséges feltételét. Állapotgépben a súlyok összege nem változhat a működés során, minden átmenet egy súlyt vesz el a bemenő helyéről és egy súlyt helyez el a kimenő helyére. Ha a gráf erősen összefüggő és legalább egy súlyt tartalmaz, akkor a súlyt tartalmazó bármelyik helytől tetszőleges átmenet előtti helyre odavihető a súly az út mentén elhelyezkedő átmenetek aktivizálásával. Így bármelyik átmenet aktivizálható újra és újra. Ha több, mint egy súly is van a gráfban, akkor ezek a súlyok utolérhetik egymást és a háló nem lesz biztonságos. Kimondhatjuk tehát az alábbi tételeket:

21.39. tétel. *Egy összefüggő állapotgép*



21.41. ábra. Ellenpélda a 21.38. tétel megfordítására.

- *akkor és csak akkor eleven, ha erősen összefüggő és van benne legalább egy súly.*
- *akkor és csak akkor biztonságos, ha legfeljebb egy súly van benne.*
- *akkor és csak akkor eleven és biztonságos, ha pontosan 1 súly van benne és erősen összefüggő.*

21.8.2. Jelzett gráf eleven és biztonságos súlyozása

Jelzett gráfok esetén a helyeket „beleolvasztjuk" az átmeneteket összekötő élekbe. Az esetleges súlyt jelezni kell az él mentén. Tekintettel arra, hogy a normális hálók osztályán belül definiáljuk a jelzett gráfok osztályát, ezért minden átmenet pontosan egy-egy súlyt vesz le a bemenő élekről és egy-egy súlyt helyez el a kimenő élein, ha tüzel. Ha kiválasztunk egy tetszőleges irányított kört a gráfban, akkor a kör két szomszédos élének súlyösszege nem változik akkor, amikor a találkozásuknál elhelyezkedő átmenet tüzel. Ez azt jelenti, hogy tetszőleges irányított körhöz tartozó éleken elhelyezkedő súlyok összege állandó:

21.40. tétel. *Jelzett gráf irányított körén a súlyösszeg invariáns (helyinvariáns), azaz* $\forall M \in R(M_0) : \forall C : M_0(C) = M(C)$, *ahol a* C *az irányított kör élhalmaza.*

Súlymentes irányított körök tehát súlymentesek maradnak a háló működése során. Egy erősen összefüggő jelzett gráfban egy átmenet holt, ha rajta van egy súlymentes irányított körön. Indirekt úton belátható, ha egy átmenet egyetlen súlymentes irányított körnek sem eleme, akkor a vele azonos irányított körökön lévő csúcspontokra is igaz ugyanez. A körök mentén tehát újra és újra eljutnak a bemenő éleire a súlyok.

21.41. tétel. *Erősen összefüggő jelzett gráf akkor és csak akkor eleven, ha minden irányított körén legalább egy súly található.*

Az eleven és biztonságos súlyozás megtalálásához el kell távolítanunk a hálóból a felesleges súlyokat. Ha egy irányított körben egynél több súly is van, akkor ezek a súlyok utolérhetik egymást, a háló nem biztonságos. Megmutatjuk, hogy egy eleven jelzett gráf súlyozása lépésenként csökkenthető oly módon, hogy az elevenség, mint invariáns tulajdonság mindig megmarad és a végén a háló biztonságos lesz. Az eljárás véges lépésben véget ér, eleven jelzett gráf súlyainak száma ugyanis mindig egy pozitív érték. A súlyok száma ily módon az eljárás variáns függvénye.

21.42. tétel. (mini-max tétel). *Jelzett gráfban egy él maximális súlya megegyezik annak a rajta áthaladó irányított körnek a súlyösszegével, amelyiknek a legkisebb a súlyösszege.*

Bizonyítás. Jelöljük $C_1, \ldots, C_m$-mel az $e = (x, y)$ élen átmenő köröket, $e_1, \ldots, e_m$-mel az e előtti éleket. Mindegyik e előtti élre mozgassunk a lehető legtöbb súlyt, a rajta áthaladó irányított kör összes súlyát. Tüzeljünk x átmenettel minél többször úgy, hogy közben y átmenet ne tüzeljen. Ez legfeljebb annyiszor lehetséges, amíg valamelyik e előtti élről elfogynak a súlyok, tehát az irányított körök súlyösszegeinek minimuma a felső korlát. ■

A mini-max tétel szerint akkor biztonságos egy jelzett gráf, ha minden él legalább egy olyan irányított körhöz tartozik, amelynek súlyösszege egy. Eleven jelzett gráf tehát akkor és csak akkor biztonságos, ha minden élhez létezik egy olyan C irányított kör, amin rajta van és aminek kezdősúlyozása, $M_0(C) = 1$. Konstruáljuk meg egy erősen összefüggő jelzett gráf eleven és biztonságos súlyozását a következőképpen. Az elevenség érdekében helyezzünk el minden irányított körre legalább egy súlyt, majd vizsgáljuk meg, hogy van-e olyan elérhető súlyozás, amikor a súlyok száma egy adott élen több, mint egy. A felesleges súlyok az elevenség veszélyeztetése nélkül eltávolíthatók erről az élről, hiszen az élt tartalmazó irányított körön továbbra is marad súly. Folytassuk ezt az eljárást addig, amíg marad olyan állapot, amelyikben a súlyozás nem biztonságos. Beláttuk a következő tételt:

21.43. tétel. *Jelzett gráfnak akkor és csak akkor létezik eleven és biztonságos súlyozása, ha erősen összefüggő.*

Jelzett gráfok eleven és biztonságos súlyozásának feltételei megfogalmazhatóak a visszacsatoló élhalmaz fogalmának segítségével is. Visszacsatoló élhalmaznak nevezzük egy irányított gráf éleinek halmazát akkor, ha ezen élek eltávolítása után a gráf irányított kört nem tartalmaz. Egy adott gráf visszacsatoló élhalmaza többféleképpen is kiválasztható, nem feltétlenül egyértelmű.

21.44. definíció. (visszacsatoló élhalmaz). *E' **visszacsatoló élhalmaz** (VéH) a $G(V,E)$ gráfban, ha $G' = (V, E - E')$ körmentes. Egy VéH minimális, ha egyetlen valódi részhalmaza sem VéH.*

A 21.41. tétel következményeként eleven jelzett gráf nem nulla súlyú éleinek halmaza VéH-t alkot, valamint ha egy jelzett gráf valamely VéH-ának minden eleme súlyozott, akkor a jelzett gráf eleven.

21.45. tétel. *Egy erősen összefüggő eleven jelzett gráf akkor és csak akkor biztonságos, ha $\forall M \in R(M_0)$: a súlyozott élek halmaza minimális VéH.*

A 21.43. tétel kimondásához vezető gondolatmenet szerint minimális visszacsatoló élhalmazok súlyozása esetén már nincs több eltávolítható súly az elevenség megőrzése mellett.

21.8.3. Elevenség és biztonságosság szabad választású és aszimmetrikus választású hálókban

Szabad választású Petri-hálók eleven és biztonságos súlyozásának feltételeit szifonok és csapdák vizsgálata alapján határozhatjuk meg.

Először megmutatjuk, hogy ha egy Petri-háló W átmenethalmazát megelőző sima helyek bemenő átmenetei is mind W-beliek, akkor vagy van megengedett t átmenet W elemei között, vagy egy sima szifon utódhalmaza tartalmazza W-t. Formálisan:

21.46. lemma. *Egy (N,M) Petri-hálóban jelölje M^0 a sima, illetve M^+ a súlyozott helyek halmazát, $P = M^0 \cup M^+$. Legyen $W \subseteq T$ átmenetek egy részhalmaza. Ha ${}^\bullet({}^\bullet W \cap M^0) \subseteq W$, akkor vagy $\exists t \in W : {}^\bullet t \subseteq M^+$ vagy létezik olyan sima szifon D, hogy $W \subseteq D^\bullet$.*

Bizonyítás. Tegyük fel, hogy nincs W-ben megengedett t. Ekkor $\forall t \in W$: ${}^\bullet t \cup M^0 \neq \emptyset$ és $t \in ({}^\bullet t \cap M^0)^\bullet$, így $W \subseteq ({}^\bullet W \cap M^0)^\bullet$, azaz $D = ({}^\bullet W \cap M^0)$ egy sima szifon (${}^\bullet D \subseteq D^\bullet$). ■

Az alábbi lemma szerint szabad választású Petri-hálókban a holt

átmenetek halmaza jellemezhető egy szifonnal (más néven holtponttal). Megmutatható, hogy létezik olyan elérhető állapot, amelyben egy sima szifon kimenő átmeneteinek halmaza tartalmazza a holt átmenetek halmazát.

21.47. lemma. *Egy (N, M) szabad választású Petri-hálóban legyen $W \subseteq T$ a holt átmenetek részhalmaza (egyetlen M-ből induló akciósorozatban sem szerepel egyetlen W-beli átmenet sem). Ekkor létezik egy olyan M-ből elérhető M' súlyozás, amelyben van egy olyan D sima szifon, hogy $W \subseteq D^\bullet$.*

Bizonyítás. A bizonyítást a $T \setminus W$ halmaz számossága szerinti indukcióval végezzük el. Alapeset: tegyük fel, hogy $|T \setminus W| = 0$, azaz minden átmenet holt. Ekkor $W = T$, így ${}^\bullet({}^\bullet W \cap M^0) \subseteq W$. Ha nincs W-ben egyetlen megengedett átmenet sem, akkor a 21.46. lemma alkalmazásával kapjuk, hogy létezik olyan D szifon, amelyre: $W \subseteq D^\bullet$.

Indukciós lépés: legyen $|T \setminus W| > 0$. Legyen a kezdő súlyozás $M_0 = M$. Konstruálunk egy olyan akciósorozatot, amely a $M_1, \ldots, M_i, \ldots, M'$ súlyozásokhoz vezet és létezik egy $D \subseteq M'^\bullet$ sima holtpont és $W \subseteq D^\bullet$.

Először megmutatjuk, hogy egyetlen akciósorozat sem aktivizál $({}^\bullet W)^\bullet$-beli átmenetet. Tegyük fel indirekt módon, hogy létezik olyan $t_0 \in W$ és $p_0 \in {}^\bullet t_0$, hogy valamely $t_1 \in p_0{}^\bullet$ aktivizálható. t_0 holt, így $t_1 \in p_0{}^\bullet \setminus W$. Ez azonban ellentmond a szabad választású háló definíciójának, ha t_1 aktivizálható, akkor t_0 is az.

Legyen az aktuális súlyozás M_i. Ha ${}^\bullet({}^\bullet W \cap M_i^0) \subseteq W$, akkor a 21.46. lemma szerint $D = {}^\bullet W \cap M_i^0$ sima holtpont és $W \subseteq D^\bullet$. Ha $M_i = M'$, akkor az állítást kapjuk. Ha ${}^\bullet({}^\bullet W \cap M_i^0) \not\subseteq W$, akkor létezik olyan $t \in {}^\bullet({}^\bullet W \cap M_i^0) \setminus W$, amelyre

a) egyetlen akciósorozatban sem szerepel t, vagy

b) létezik olyan ς akciósorozat, amelyikben t előfordul.

Az a) esetben jelöljük a $W \cup \{t\}$ halmazt W'-vel. Egyetlen akciósorozat sem aktivizál W'-beli átmenetet. $|T \setminus W'| = |T \setminus W| - 1$, ezért az indukciós feltevés szerint van egy olyan ς akciósorozat, amely M'-höz vezet M-ből és van egy $D \subseteq M'^0$ sima holtpont, hogy $W' \subseteq D^\bullet$. Mivel $W \subseteq W'$, ezért a lemmát ebben az esetben bizonyítottuk.

A b) esetben jelöljük M_{i+1}-gyel azt a súlyozást, amelyiket M_i-ből kapunk a ς akciósorozattal. Megmutattuk, hogy egyetlen akciósorozat sem aktiválhat $({}^\bullet W)^\bullet$-beli átmenetet, így ς sem, azaz: $({}^\bullet W \cap M_i^+) \subseteq ({}^\bullet W \cap M_{i+1}^+)$. (Ami súlyozott volt ${}^\bullet W$-ben, az súlyozott marad.) Mivel t ebben az esetben ${}^\bullet W \cap M_i^0$-ba mutat és $({}^\bullet W)^\bullet$-ben ς nem aktivizál, $|{}^\bullet W \cap M_{i+1}^0| < {}^\bullet W \cap M_i^0$. Ismételjük meg az állítást M_{i+1}-re. Valahányszor b) esetet alkalmazzuk, a ${}^\bullet W \cap M_{i+1}^0$

számossága csökken, így végül az a) esethez jutunk. ■

A 21.47. lemmából következik, hogy ha a Petri-hálóban egyetlen szifon sem válik sima szifonná, akkor bármely adott átmenethez van olyan akciósorozat, amelyben az átmenet szerepel. (Legyen $W = \{t\}$.) Ha a szifon súlyozott csapdát tartalmaz, akkor nem válik simává. Egy szabad választású háló eleven, ha minden szifon tartalmaz súlyozott csapdát. A tétel megfordítása is teljesül, a bizonyítást a további tételek bizonyításával együtt az érdeklődő Olvasó megtalálhatja az idézett szakirodalomban.

21.48. tétel. (szabad választású háló elevensége). *Egy szabad választású háló akkor és csak akkor eleven, ha minden szifon tartalmaz súlyozott csapdát.*

21.49. tétel. (szabad választású háló elevensége és biztonságossága). *Egy szabad választású háló akkor és csak akkor eleven és biztonságos, ha lefedhető olyan erősen összefüggő állapotgépekkel, amelyeknek mindegyikében pontosan 1 súly van, és minden minimális szifon egy súlyozott erősen összefüggő állapotgép.*

21.50. tétel. (aszimmetrikusan választó háló elevensége). *Aszimmetrikusan választó háló eleven akkor (de nem csak akkor), ha minden szifon tartalmaz súlyozott csapdát.*

Gyakorlatok

21.8-1. Mutassunk példát biztonságos és eleven jelzett gráfra, illetve állapotgépre.

21.8-2. Mutassunk példát biztonságos és eleven szabad választású hálóra.

21.9. Petri-dobozok

A fejezet eddigi részében a Petri-háló általános fogalmát vizsgáltuk. A továbbiakban kifejezetten párhuzamos programok, illetve elosztott algoritmusok vizsgálatához használható hálókkal, Petri-dobozokkal fogunk foglalkozni.

A Petri-doboz speciális tulajdonságokkal rendelkező címkézett Petri-háló. A címkézett Petri-háló egy speciális címkefüggvénnyel kiterjesztett Petri-háló. A címkefüggvény a háló helyeihez e, i és x címkéket rendel. Az e címke jelöli a háló belépési helyeit, az i a belső helyeket, míg x a kilépési helyeket.

Az átmenetekhez ennél jóval bonyolultabb címkéket, úgynevezett átcímkézéseket rendelünk. A Petri-dobozoknak két alapvető típusa van. A sima doboz, amelyet egy algoritmus vagy program leírására használhatunk, illetve az operátordoboz, amely programok közötti műveletek (programkonstrukciók, átnevezés, szinkronizáció) leírására szolgál. A sima doboz esetén egy

átmenet végrehajtása a háló által szimulált program, algoritmus egy adott eseményének vagy több szinkronizált eseményének a végrehajtását jelenti. Így sima dobozok átmeneteihez a címkézés egy eseményzsákot rendel. Ezzel szemben operátordoboz esetén egy átmenet egy teljes programot (az ahhoz tartozó sima dobozt) reprezentál és az átmenethez tartozó címke fontos szerepet játszik az operátordoboz által meghatározott műveletben (lásd a 21.10 alfejezetet).

Ennek megfelelően az átcímkézésnek is két alapvetően különböző fajtája van, a konstans átcímkézés és a nem konstans átcímkézés. A konstans átcímkézés megfeleltethető egy eseményzsáknak, míg a nem konstans átcímkézés olyan zsákokhoz, amelyek eseményzsákokat tartalmazó halmazokból állnak, rendel hozzá eseményzsákokat tartalmazó halmazt.

Jelöljük $mult(A)$-val az A-beli eseményekből képzett zsákok halmazát. Egy $mult(A)$-beli eseményzsákot jellemezhetünk egy zsákfüggvénnyel, amely megadja, hogy melyik eseményből hány darabot tartalmaz.

21.51. definíció. (esemény előfordulási száma – zsákfüggvény).
$\mu : A \longrightarrow \mathcal{N}_0$ *zsákfüggvény.* $mult(A) = \{\mu | \mu : A \to \mathbf{N}_0\ \}$.
Például μ *jelölje az* $\{aabccc\}$ *zsákot, ekkor* $\mu(a) = 2, \mu(b) = 1, \mu(c) = 3$.

Nézzük meg az átcímkézés definícióját formálisan. Legyen Act a primitív események egy előre megadott halmaza (ez a háló által szimulált program vagy algoritmus eseményeinek a halmaza).

21.52. definíció. (átcímkézés).
$Lab = mult(Act)$
ρ *átcímkézés egy reláció:*
$\rho \subseteq (mult(Lab)) \times Lab$, *úgy, hogy* $(\emptyset, \alpha) \in \rho$ *akkor és csak akkor, ha* $\rho = \{(\emptyset, \alpha)\}$.

Speciális átcímkézések:
konstans átcímkézés: $\rho_\alpha = \{(\emptyset, \alpha)\}$,
megszorítás: $\rho_{Lab'} = \{(\{\alpha\}, \alpha) | \alpha \in Lab'\}$, csak a $Lab' \subset Lab$ beli eseményeket tartja meg,
identitás: $\rho_{id} = \{(\{\alpha\}, \alpha) | \alpha \in Lab\}$.

Mint látható, a konstans átcímkézést is $(mult(Lab)) \times Lab$ alakban adjuk meg, ám egy $\rho_\alpha = \{(\emptyset, \alpha)\}$ átcímkézés egyértelműen megfeleltethető α-nak. Ennek megfelelően a leírt példákban ρ_α helyett mindig csak α-t fogunk használni.

Ezek után nézzük a címkézett Petri-háló formális definícióját. Ez az

előzőekben tárgyalt címkefüggvényt nem számítva csak jelölésekben különbözik az általános Petri-háló definíciójától.

21.53. definíció. (címkézett Petri-háló).
A $\Sigma = (S, T, W, \lambda, M)$ *ötös egy* ***címkézett Petri-háló****, ahol* S *a helyek,* T *az átmenetek halmaza,* W *az éleket leíró reláció,* λ *a címkefüggvény,* M *pedig a súlyozás.*
$S \cap T = \emptyset$,
$W : ((S \times T) \cup (T \times S)) \to \mathbf{N}_0$,
$\forall s \in S : \lambda(s) \in \{e, i, x\}$,
$\forall t \in T : \lambda(t)$ *egy* $\rho \subseteq (mult(Lab)) \times Lab$ *átcímkézés,*
$M \subseteq S \times \mathbf{N}_0$.

Az események között párokat definiálhatunk, melyek egy külső környezet felől nézve kioltják egymás hatását. Például ilyen lehet egy fájl megnyitására irányuló kérelem és a fájl megnyitása, vagy elosztott esetben egy üzenet elküldése és fogadása.

21.54. definíció. (párokat definiáló függvény). $\hat{} : A \to A$ *függvény párokat definiál* A *felett, bijekció,* $a \neq \hat{a}$ *és* $\hat{\hat{}} = \hat{}^{(-1)}$.

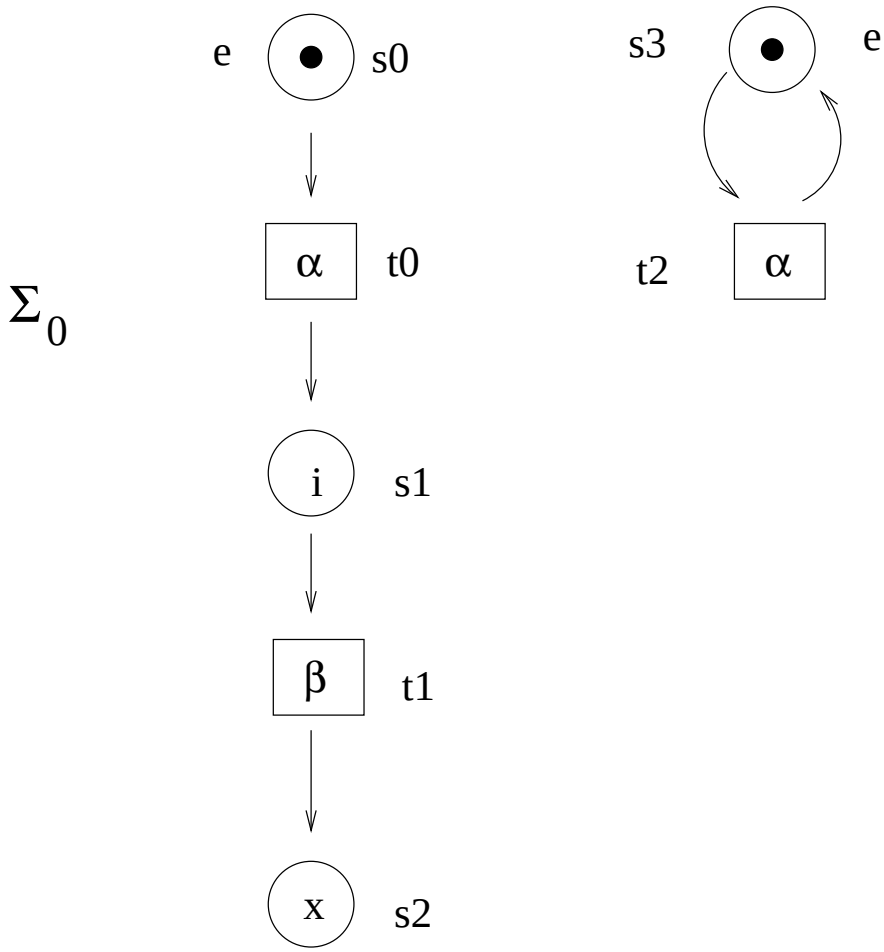
Zsákok esetén az alábbi jelölést használjuk:
Legyen adott egy A eseményhalmaz és egy $\hat{}$ párokat definiáló függvény A felett. $\hat{\mu}(a) ::= \mu(\hat{a})$.

Mint említettük, címkézett Petri-hálók esetén a címkézés alapján a helyeket három különböző halmazba soroljuk, a belépési helyek (${}^\bullet\Sigma$), kilépési helyek ($\Sigma^\bullet$) és a belső helyek ($\ddot{\Sigma}$) halmazába.

Azaz formálisan: $\Sigma = (S, T, W, \lambda, M)$. Legyen $s \in S$. Ha $\lambda(s) = e$ (entry), akkor$s$ belépési hely, $\lambda(s) = x$ (exit), akkor s kilépési hely, $\lambda(s) = i$ (internal), akkor s belső hely, ${}^\bullet\Sigma = \{s \in S | \lambda(s) = e\}$ a belépési helyek halmaza, $\Sigma^\bullet = \{s \in S | \lambda(s) = x\}$ a kilépési helyek halmaza, $\ddot{\Sigma} = \{s \in S | \lambda(s) = i\}$ a belső helyek halmaza.

Valós alkalmazás szimulációja esetén általában az alkalmazás indításakor a belépési helyekre helyezünk súlyt, és az alkalmazás végrehajtásának megfelelő szimuláció során a súlyok a belső helyeken keresztül átkerülnek a kilépési helyekre, ha az összes súly kilépési helyen van, akkor a szimuláció befejeződött. Azonban a címkézett Petri-háló definíciója ennél jóval általánosabb, semmilyen megkötés nincs arra, hogy melyik helyhez milyen címkét rendeljünk. Megkötéseket majd csak a Petri-doboz definíciójánál vezetünk be.

Nézzünk meg most példaként egy címkézett Petri-hálót. A példa négy



21.42. ábra. A 21.2 példában leírt Σ_0 címkézett Petri-háló.

helyet ($\{s_0, s_1, s_2, s_3\}$) és három átmenetet ($\{t_0, t_1, t_2\}$) tartalmaz. A helyek közül s_0 és s_3 belépési, s_1 belső, s_2 pedig kilépési hely, kezdetben s_0 és s_3 tartalmaz súlyt. A t_0 átmenet α konstans címkével van címkézve, egy megelőző helye (s_0) és egy rákövetkező helye (s_1) van. A t_1 átmenet β konstans címkével van címkézve, egy megelőző helye (s_1) és egy rákövetkező helye (s_2) van. Végezetül pedig a t_2 átmenet egy hurok átmenet, amely α-val van címkézve és a megelőző és a rákövetkező helye is s_3.

21.2. példa. [címkézett Petri-háló]
$\Sigma_0 = (S_0, T_0, W_0, \lambda_0, M_0)$

$S_0 = \{s_0, s_1, s_2, s_3\}$
$T_0 = \{t_0, t_1, t_2\}$
$W_0 = ((TS \cup ST) \times \{1\}) \cup (((S \times T) \setminus ST \cup (T \times S) \setminus TS) \times \{0\})$
$\lambda_0 = \{(s_0, e), (s_1, i), (s_2, x), (s_3, e), (t_0, \alpha), (t_1, \beta), (t_2, \alpha)\}$
$M_0 = \{(s_0, 1), (s_1, 0), (s_2, 0), (s_3, 1)\}$,
ahol
$TS = \{(t_0, s_1), (t_1, s_2), (t_2, s_3)\}$ és $ST = \{(s_0, t_0), (s_1, t_1), (s_3, t_2)\}$ (lásd 21.42. ábra).

21.9.1. Működési szabály címkézett Petri-hálón

Címkézett Petri-hálók esetén kiterjesztjük az eddigiekben megismert működési szabályt. Általános Petri-hálók esetén egyidejűleg csak egy átmenet hajtódhat végre. Ezzel szemben itt megengedjük, hogy egyidejűleg egyszerre több átmenet, sőt egy átment többször is végrehajtódjon. Ennek megfelelően címkézett Petri-hálók esetén egy lépés nem egy átmenet, hanem egy átmenetzsák végrehajtását írja le. Így itt egy lépés akkor engedélyezett, ha minden helyen van elég súly ahhoz, hogy szinkron végre tudjuk hajtani a zsákban szereplő összes lépést (amelyik többször szerepel, azt annyiszor, ahányszor szerepel a zsákban).

21.55. definíció. (engedélyezett lépés).
$\Sigma = (S, T, W, \lambda, M)$
Legyen $U \in mult(T)$ *átmenetek egy véges zsákja, az* U *által meghatározott lépés engedélyezett* Σ*-ban, ha minden* $s \in S$ *helyre:*
$M(s) \geq \sum_{t \in U} U(t) * W(s, t)$.

A lépés végrehajtása során a zsákbeli összes átmenet levesz a megelőző helyeiről az átmenetbe vezető éleknek megfelelő számú súlyt és rárak a rákövetkező helyeire a kivezető éleknek megfelelő számú súlyt (természetesen, ha egy átmenet többször is szerepel a zsákban, akkor ez az előfordulási számnak megfelelően, többszörösen megtörténik).

21.56. definíció. (lépés végrehajtása).
Jelölje M' *az* $U \in mult(T)$ *által meghatározott lépés végrehajtása utáni súlyozást.*
$\forall s \in S : M'(s) = M(s) + \sum_{t \in U} U(t) * (W(t, s) - W(s, t))$
Jelölés: $M\,[U > M'$*, vagy* $\Sigma\,[U > \Theta$*, ahol* $\Theta = (S, T, W, \lambda, M')$.

A háló működése során egy adott pillanatban bármelyik engedélyezett lépés végrehajtható. A különböző tulajdonságok vizsgálata szempontjából

számunkra igazából nem is az egyes lépések, hanem az egymás után végrehajtható lépések sorozata, a lépéssorozat a fontos. Ha egy M_0 kezdősúlyozásból kiindulva van olyan végrehajtható lépéssorozat, amely egy adott M súlyozáshoz vezet, akkor azt mondjuk, hogy M elérhető M_0-ból. Hasonlóan, ha van olyan lépéssorozat, amely egy Σ_0 címkézett hálóból egy Σ címkézett hálóba vezet, akkor azt mondjuk, hogy Σ származtatható Σ_0-ból.

21.57. definíció. (véges lépéssorozat).
$\sigma = U_1 \dots U_k$ egy véges lépéssorozat Σ-ban, ha létezik $\Sigma_0, \dots, \Sigma_k$ címkézett háló, úgy, hogy $\Sigma = \Sigma_0$ és $\forall i \in [1 \dots k] : \Sigma_{i-1} [U_i > \Sigma_i$.
Jelölés: $\Sigma [\sigma > \Sigma_k$, vagy $M_\Sigma [\sigma > M_{\Sigma_k}$.
Elnevezés: M_{Σ_k} súlyozás elérhető M_Σ-ból ($M_{\Sigma_k} \in [M_\Sigma >$), és Σ_k származtatható Σ-ból ($\Sigma_k \in [\Sigma >$).

Egy hálóról akkor mondjuk, hogy önkonkurenciát tartalmaz, ha megenged olyan lépéssorozatot, amely nem átmenethalmazok sorozata (azaz a sorozat lépéseinek valamelyike valódi zsák).

Nézzünk egy példát végrehajtható lépéssorozatra.

21.3. példa. A 21.2. példabeli Σ_0-ban t_0 és t_2 konkurensen engedélyezett, így $\{t_0, t_2\}$ egy engedélyezett lépés. Miután ezt a lépést végrehajtjuk, t_1 és t_2 konkurensen végrehajthatóvá válik, így a $\{t_0, t_2\}\{t_1, t_2\}$ egy lépéssorozata Σ_0-nak.
Megjegyzés. Egy lépésnek nem kell maximálisnak lennie, például $\{t_0\}$ is egy lépése, illetve $\{t_0\}\{t_1\}\{t_2\}\{t_2\}$ és $\{t_0, t_2\}\{t_1\}\{t_2\}$ is egy lépéssorozata Σ_0-nak.

21.9.2. Címkézett Petri-hálók tulajdonságai

Ebben a részben definiálunk néhány, a címkézett Petri-hálókon értelmezett tulajdonságot. Az első, az úgynevezett *T-megszorítás* azt fogalmazza meg, hogy a háló minden átmenetének van megelőző és rákövetkező helye is, azaz el akarjuk kerülni a forrás és nyelő átmeneteket.

21.58. definíció. (T-megszorítás). $\forall t \in T : {}^\bullet t \neq \emptyset \neq t^\bullet$
Ezt a tulajdonságot a továbbiakban minden hálóra feltesszük.

A következő tulajdonság az *ex-megszorítás*, amely akkor teljesül, ha a hálónak létezik legalább egy belépési és egy kilépési helye.

21.59. definíció. (ex-megszorítás:). ${}^\bullet\Sigma \neq \emptyset$ *és* $\Sigma^\bullet \neq \emptyset$

Valós alkalmazások esetén a belépési helyekre csak az alkalmazás indításakor akarunk súlyokat helyezni, azaz nem akarjuk, hogy a belépési

helyekre vezessen él (*e-iranyitott@e-irányított* háló). Illetve a kilépési helyekről nem akarunk súlyt elvenni, hiszen ha kerül súly egy kilépési helyre, akkor az az adott rész befejeződését jelenti. Tehát azt akarjuk, hogy egyik kilépési helyről se vezessen ki él (*x-irányított* háló).

21.60. definíció. (e-irányított háló). Σ ***e-irányított,*** *ha* $\forall s \in {}^\bullet\Sigma : \forall t : W(t,s) = 0$.

21.61. definíció. (x-irányított háló). Σ ***x-irányított,*** *ha* $\forall s \in \Sigma^\bullet : \forall t : W(s,t) = 0$.

21.62. definíció. (ex-irányított háló). Σ ***ex-irányított,*** *ha e-irányított és x-irányított.*

A másik viszonylag természetes elvárás lehet egy hálóval szemben, hogy ne kerüljön egyszerre súly egy belépési és egy kilépési helyére. Ezt fogalmazza meg az *ex-kizárólagosság*, azzal a megszorítással, hogy mindezt ne csak a kezdősúlyozásból ne lehessen elérni, de azokból a súlyozásokból sem, amelyek csak a belépési ($M_{\bullet\Sigma}$ súlyozás), illetve kilépési helyekre ($M_{\Sigma\bullet}$ súlyozás) helyeznek súlyt.

21.63. definíció. (ex-kizárólagos). *Legyen*

$$\forall s \in S : M_{\bullet\Sigma}(s) = \begin{cases} 1, & \text{ha } s \in {}^\bullet\Sigma \text{ ,} \\ 0, & \text{különben .} \end{cases}$$

$$\forall s \in S : M_{\Sigma\bullet}(s) = \begin{cases} 1, & \text{ha } s \in \Sigma^\bullet, \\ 0, & \text{különben .} \end{cases}$$

Σ ***ex-kizárólagos****, ha minden* M_Σ*-ból, vagy* $M_{\bullet\Sigma}$*-ból, vagy* $M_{\Sigma\bullet}$*-ból elérhető* M *súlyozásra* $M \cap M_{\bullet\Sigma} = \emptyset$ *vagy* $M \cap M_{\Sigma\bullet} = \emptyset$.

21.4. példa. A 21.2 példabeli Σ_0 háló ex-megszorított és x-irányított, de nem e-irányított és nem ex-kizárólagos.

A fenti tulajdonságok vizsgálata szempontjából érdekes lehet, hogy a háló tartalmaz-e olyan átmenetet, amelynek vagy a megelőző helyei között vagy a rákövetkező helyei között van belépési és kilépési pont is. Ha van ilyen átmenet egy címkézett Petri-hálóban, akkor biztosan tudjuk, hogy az a háló ex-megszorított, de nem ex-irányított. Abban az esetben pedig, ha az adott átmenet végrehajtható, akkor a háló nem ex-kizárólagos, mivel kell

legyen olyan, a kezdősúlyozásból elérhető súlyozás, amely az átmenet minden megelőző helyére helyez súlyt és olyan is, amely az átmenet minden rákövetkező helyére rak súlyt, így a definíció szerint valamelyik súlyozás a kettő közül súlyt fog helyezni belépési és kilépési pontra is. A leírt tulajdonságú átmenetet *ex-aszimmetrikusnak* nevezzük.

21.64. definíció. (ex-aszimmetrikus átmenet). *Egy* $t \in T$ *átmenet* ***ex-aszimmetrikus****, ha*
${}^\bullet t \cap {}^\bullet\Sigma \neq \emptyset \neq {}^\bullet t \cap \Sigma^\bullet$ *vagy* $t^\bullet \cap {}^\bullet\Sigma \neq \emptyset \neq t^\bullet \cap \Sigma^\bullet$.

Végül a lehetséges lépéssorozatok vizsgálata szempontjából fontos lehet, hogy melyek azok az átmenetek, melyek teljesen függetlenek egymástól. A függetlenséget fogalmazza meg formálisan az alábbi definíció.

21.65. definíció. (függetlenség reláció).
$ind_\Sigma = \{(t,u) \in T \times T | ({}^\bullet t \cup t^\bullet) \cap ({}^\bullet u \cup u^\bullet) = \emptyset\}$.

Ha egy Σ címkézett Petri-háló biztonságos, akkor bármely két átmenet, amely megjelenik egy lépésben, *független* a fenti definíció szerint.

A címkézett Petri-hálókon definiálunk néhány műveletet, amelyek a hálók súlyozását módosítják (ezen kívül semmilyen más változtatást nem végeznek). Ezek közül az első elveszi az összes súlyt a hálóból, a második csak a háló belépési helyeire helyez súlyt, míg a harmadik a háló összes kilépési helyére rak súlyt (és sehova máshova nem).

21.66. definíció. (súlyozást módosító műveletek).
Legyen $\Sigma = (S, T, W, \lambda, M)$*, ekkor*

$\lfloor\Sigma\rfloor = (S, T, W, \lambda, \emptyset)$

$\overline{\Sigma} = (S, T, W, \lambda, M_{\bullet\Sigma})$

$\underline{\Sigma} = (S, T, W, \lambda, M_{\Sigma\bullet})$.

21.9.3. Petri-doboz definíciója

A Petri-doboz egy címkézett Petri-háló, amely megadott tulajdonságokkal rendelkezik.

21.67. definíció. (Petri-doboz). Σ *címkézett Petri-háló* ***Petri-doboz****, ha ex-megszorított, valamint ex-irányított (és* T *megszorított).*

Azaz egy Petri-doboztól megköveteljük, hogy legyen legalább egy belépési és egy kilépési pontja, valamint hogy ne vezessen él a belépési pontjaiba és ne vezessen ki él a kilépési pontjaiból. Ezen felül természetesen megköveteljük, hogy minden átmenetének legyen megelőzője és rákövetkezője (mint említettük ezt gyakorlatilag minden címkézett Petri-hálóra feltesszük).

A Petri-dobozok legegyszerűbb fajtája a *sima doboz*, amelynél minden átmenethez konstans átcímkézés van rendelve.

21.68. definíció. (sima doboz). *Σ Petri-doboz sima doboz, ha minden $t \in T_\Sigma$ átmenetre a $\lambda_\Sigma(t)$ egy konstans átcímkézés.*

A sima dobozokat egyszerű párhuzamos folyamatok, algoritmusok szimulációjához használhatjuk. Az átmenetekhez rendelt konstans címkék az összetartozó eseményeket jelölik, amelyeket egyszerre hajtunk végre.

Egy valódi folyamat szimulációja esetén elvárhatjuk, hogy ha az összes belépési vagy kilépési helyén van súly, akkor máshol ne legyen. Ha egy súlyozás teljesíti ezt, akkor *tiszta súlyozásnak* nevezzük.

21.69. definíció. (tiszta súlyozás). *Egy M súlyozás tiszta, ha nem valódi szuper-zsákja $M_{\bullet\Sigma}$-nak és $M_{\Sigma\bullet}$-nak, azaz, ha $M_{\bullet\Sigma} \subseteq M$, akkor $M_{\bullet\Sigma} = M$, valamint ha $M_{\Sigma\bullet} \subseteq M$, akkor $M_{\Sigma\bullet} = M$.*

A továbbiakban elsősorban olyan sima dobozokkal fogunk foglalkozni, melyeknél az összes elérhető súlyozás biztonságos és tiszta. Az ilyen dobozokon belül is két alapvető típust különböztetünk meg, az elsőnél gyakorlatilag csak a doboz szerkezete a fontos, azaz azok a dobozok tartoznak bele, amelyek nem tartalmaznak súlyt, míg a második típusba azon dobozok tartoznak, melyekhez nem üres súlyozás van rendelve.

21.70. definíció. (statikus doboz). *Egy Σ súlyozatlan sima doboz statikus doboz, ha minden $M_{\bullet\Sigma}$-ból és $M_{\Sigma\bullet}$-ból elérhető súlyozás biztonságos és tiszta.*

21.71. definíció. (dinamikus doboz). *Egy Σ súlyozott sima doboz dinamikus doboz, ha minden M_Σ-ból, $M_{\bullet\Sigma}$-ból és $M_{\Sigma\bullet}$-ból elérhető súlyozás biztonságos és tiszta.*

Ha Σ egy statikus doboz és Θ származtatható $\overline{\Sigma}$-ból (azaz abból a dobozból, amelyet Σ-ból úgy kapunk meg, hogy minden belépési helyre helyezünk súlyt), akkor Θ egy dinamikus doboz (azaz speciálisan $\overline{\Sigma}$ maga is egy dinamikus doboz). Ugyanakkor a 21.71. definíció nem követeli meg, hogy M_Σ elérhető legyen $M_{\bullet\Sigma}$-ból vagy $M_{\Sigma\bullet}$-ból, ha mégis az, akkor a definícióban az M_Σ-ra vonatkozó rész elhagyható.

A fentieken kívül a sima dobozoknak még két speciális osztályát különböztetjük meg, ezek a *belépési* és *kilépési* dobozok, olyan dinamikus dobozok, amelyeknél a súlyozás megegyezik $M{\bullet}_\Sigma$-val, illetve $M_\Sigma{\bullet}$-val. (A dinamikus dobozok halmaza tartalmazza a *belépési* és a *kilépési* dobozok halmazát.)

Az említett osztályokra a következő jelöléseket vezetjük be:
Statikus dobozok halmaza: Box^s,
Dinamikus dobozok halmaza: Box^d,
Belépési dobozok halmaza: Box^e,
Kilépési dobozok halmaza: Box^x,
Sima dobozok halmaza: Box

Dinamikus dobozokra megfogalmazhatjuk az alábbi egyszerű tételt:

21.72. tétel. *Legyen Σ egy dinamikus doboz és U egy engedélyezett lépés Σ-ban. Ekkor*

1. *minden Σ-ból elérhető címkézett háló egy dinamikus doboz,*
2. *U kölcsönösen független átmenetek egy halmaza:*
   $U \times U \subseteq ind_\Sigma \cup id_{T_\Sigma}$, ahol $id_X = \{(x,x)|x \in X\}$ minden X halmazra,
3. *és minden U-hoz kapcsolódó él egyszeres:*
   $W_\Sigma(U \times S_\Sigma) \cup W_\Sigma(S_\Sigma \times U) \subseteq \{0,1\}$.

A tétel bizonyítását az Olvasóra hagyjuk.

21.9.4. Operátordoboz

A Petri-dobozok másik alapvető fajtája az operátordoboz. Az ilyen dobozokat elsősorban programrészletek közötti műveletek (például programkonstrukciók, szinkronizáció, átnevezés) leírására használhatjuk. Ennek megfelelően az átmeneteihez nem konstans átcímkézéseket rendelünk. A nem konstans átcímkézéseket transzformációs átcímkézésnek nevezzük. Az átcímkézésen kívül az operátordoboz minden átmenetéhez hozzárendelhetünk egy sima dobozt is. Az operátordoboz által leírt műveletet az átmeneteihez rendelt sima dobozokon fogjuk elvégezni.

21.73. definíció. (operátordoboz). *Egy Ω **operátordoboz** olyan doboz, melynek minden átmenetéhez transzformációs (azaz nem konstans) átcímkézés van rendelve. Az operátordobozhoz hozzárendelhetünk egy $\Sigma : T_\Omega \to Box$, az Ω átmeneteiről a sima dobozok halmazára képező függvényt. Σ-t rendezett $\Omega - n -$es-nek fogjuk nevezni. $\forall v \in T_\Omega$-ra $\Sigma(v)$-t Σ_v-vel jelöljük.*

Ha Ω átmenethalmaza véges, akkor feltesszük, hogy $T_\Omega = \{v_1, \ldots, v_n\}$ egy tetszőleges, de fix rendezés az átmenethalmazon és ekkor a $\Sigma = \{\Sigma_{v_1}, \ldots, \Sigma_{v_n}\}$ vagy a $\Sigma = \{\Sigma_1, \ldots, \Sigma_n\}$ jelölést használjuk.

Mint említettük, operátordobozok esetén egy-egy átmenet egy teljes sima doboznak, azaz egy teljes programrészletnek, programnak felel meg. Ennek megfelelően korántsem biztos, hogy egy átmenet egy időpillanat alatt végrehajtódik, sőt lehet olyan átmenet, amelynek végrehajtása akár végtelen időt is igénybe vehet (például holtpont, végtelen ciklus). Ezért szükség van annak nyilvántartására, hogy van-e olyan átmenetünk, amely aktivizálódott, de még nem fejeződött be (pillanatnyilag aktív). Ebből a célból vezetjük be a *komplex súlyozás* definícióját.

21.74. definíció. (komplex súlyozás). *Egy Ω operátordoboz **komplex súlyozása** egy $\mathcal{M} = (M, Q)$ pár, ahol M egy szabványos súlyozás, Q pedig az Ω-beli aktív átmenetek véges zsákja. M-et az $\mathcal{M}$ súlyozás valós részének, míg Q-t a képzetes részének nevezzük. Egy szabványos M súlyozást az $(M, \emptyset)$ komplex súlyozással lehet reprezentálni.*

Az eddig megismert működési szabályt ki kell terjesztenünk komplex súlyozás esetére. Egy lépés teljes végrehajtása gyakorlatilag megegyezik az eddig megismert végrehajtással. A változás abban áll, hogy a kiterjesztett definíció mellett megengedett, hogy egy lépés csak aktívvá váljon (azaz elkezdődjön a végrehajtása), de ne fejeződjön be, illetve egy éppen aktív lépés bármikor befejeződjön.

21.75. definíció. (kiterjesztett működési szabályok). *Egy U lépés engedélyezett az $\mathcal{M} = (M, Q)$ súlyozás mellett, ha M mellett engedélyezett. Ezt $\mathcal{M}[U>$-val jelöljük.*

*Egy engedélyezett U lépés (teljesen) **végrehajtottá** válhat:*
$\mathcal{M}[U> \mathcal{M}'$, ahol $\mathcal{M}' = (M', Q)$, úgy, hogy

$$\forall s \in S: \; M'(s) = M(s) + \sum_{t \in U} U(t) * (W(t,s) - W(s,t)).$$

*Egy engedélyezett U lépés **aktívvá** válhat:*
$\mathcal{M}[U^+> \mathcal{M}'$, ahol $\mathcal{M}' = (M', Q + U)$, úgy, hogy
$\forall s \in S: \; M'(s) = M(s) - \sum_{t \in U} U(t) * W(s,t).$

*Valamint egy $U \subseteq Q$ aktív lépés bármikor **befejezetté** válhat:*
$\mathcal{M}[U^-> \mathcal{M}'$, ahol $\mathcal{M}' = (M', Q - U)$, úgy, hogy

$$\forall s \in S: \; M'(s) = M(s) + \sum_{t \in U} U(t) * W(t,s).$$

általánosan a fenti három esetet együtt a következőképpen jelöljük:
$\mathcal{M}[U : V^+ : Y^- > (M', Q')$, *ha* $Y \subseteq Q$, $Q' = Q + V - Y$

és $\forall s \in S : M(s) \geq \sum_{t \in U+V} (U(t) + V(t)) * W(s,t)$

$$M'(s) = M(s) + \sum_{t \in U+Y} (U(t) + Y(t)) * W(t,s) - \sum_{t \in U+V} (U(t) + V(t)) * W(s,t)$$

Az $\mathcal{M}$*-ből elérhető komplex súlyozásokat* $[\mathcal{M}>$*-el jelöljük.*

Komplex súlyozások esetére viszonylag egyszerűen kiterjeszthetjük a *biztonságosság*, *korlátosság* és *tisztaság* definícióit.

21.76. definíció. (kiterjesztett definíciók). *Egy* $\mathcal{M} = (M, Q)$ *komplex súlyozás biztonságos, k-korlátos és tiszta, ha megfelelően* M *biztonságos, k-korlátos és tiszta.*

Ha Ω egy $\mathcal{M} = (M, Q)$ komplex súlyozású operátordoboz, akkor a doboz által definiált művelet sima dobozok bármely Σ rendezett $\Omega - n$-esére alkalmazható feltéve, hogy Σ_v akkor és csak akkor súlyozott, ha $v \in Q$.

Gyakorlatok

21.9-1. Rajzoljunk fel olyan Petri-dobozt, amely ex-kizárólagos.

21.9-2. Bizonyítsuk be a 21.72. tételt.

21.10. Az operátordoboz által definiált művelet, hálófinomítás

Mint említettük, egy operátordoboz mindig valamilyen programrészek közötti műveletet ír le. Egy Ω operátordoboz esetén egy adott Σ rendezett $\Omega - n$-es határozza meg, hogy a doboz által definiált műveletet milyen sima dobozokra kell alkalmazni. A művelet maga két részből áll. Először a doboz átmeneteihez rendelt átcímkézések által meghatározott *interfész váltást* kell elvégezni az adott átmenethez a Σ által hozzárendelt sima dobozon. Ez a műveletrész a sima dobozok átmenetein végez átalakítást, ezáltal megváltoztatható azok szerkezete.

21.77. definíció. (interfész váltás). *Legyen Ω egy operátordoboz és Σ egy rendezett $\Omega - n$-es. A Σ-ra vonatkozó Ω szerinti* ***interfész váltás*** *azt jelenti, hogy minden Σ-beli Σ_v-re végrehajtjuk a megfelelő v átmenet $\lambda_\Omega(v)$ átcímkézése által meghatározott interfész váltást.*

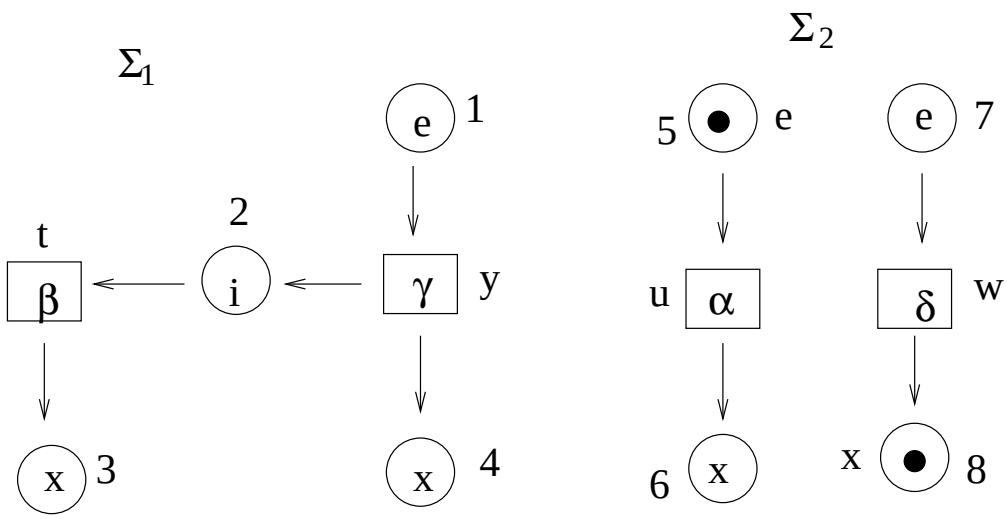
A második műveletrészben az így átalakított sima dobozokat kapcsoljuk össze az operátordoboz szerkezete alapján. Ezt a műveletrészt *átmenet finomításnak* nevezzük, mivel az operátordoboz átmeneteit finomítjuk azáltal, hogy az átmenetek helyére a hozzájuk rendelt sima dobozokat illesztjük. A két műveletrészt együtt *hálófinomításnak* nevezzük

21.78. definíció. (hálófinomítás). *Legyen Ω egy operátordoboz, Σ pedig egy rendezett $\Omega - n$-es. Σ Ω szerinti hálófinomítását, $\Omega(\Sigma)$-t úgy kapjuk meg, hogy minden Σ-beli Σ_v-re vesszük a megfelelő $\lambda_\Omega(v)$ szerinti interfész váltást, majd az így kapott új hálókat felhasználva elvégezzük az Ω szerinti átmenetfinomítást.*

Nézzük meg kicsit részletesebben a két műveletrészt. Először vizsgáljuk meg egy adott Σ_v-re vonatkozó $\rho_v = \lambda_\Omega(v)$ szerinti interfész váltás algoritmusát. Az eljárás csak Σ_v átmeneteit változtatja meg, a helyeket változatlanul hagyja (a súlyozásukkal együtt). Az algoritmus veszi a Σ_v-beli átmenetek által képzett összes lehetséges nem üres halmazt, mivel egy címkét több átmenethez is rendelhetünk, így ezek a halmazok címkezsákokat fognak meghatározni. A kapott címkezsákok közül ki kell választani azokat, amelyek szerepelnek a ρ_v átcímkézés értelmezési tartományában. Ha van ilyen címkezsák, akkor venni kell a hozzá tartozó átmenethalmazt, az ebbe tartozó átmeneteket össze kell vonni egyetlen átmenetté, és a ρ_v által a vizsgált címkezsákhoz rendelt címkét kell rendelni ehhez az összevont átmenethez.

INTERFÉSZVÁLTÁS(Σ_v, ρ_v)

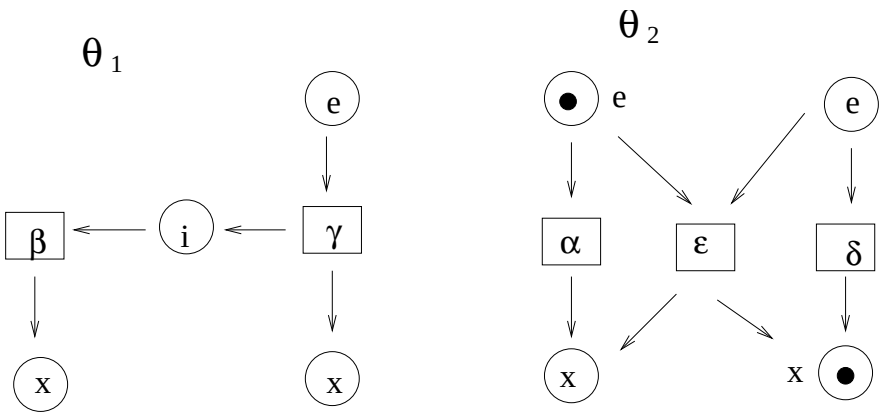
```
1  legyen T a Σ_v átmeneteinek halmaza
2  HT = 2^T \ ∅
3  Θ_v = üres doboz
4  Θ_v = BEILLESZT-HELYEK(Θ_v, Σ_v)
5  minden α ∈ HT halmazra
6          c = CÍMKEZSÁK(α)
7          if c ∈ D_ρv
8              újátmenet = ÖSSZEVONT-ÁTMENETET-KÉSZÍT(α)
9              Θ_v = BEILLESZT-ÁTMENET(Θ_v, újátmenet)
10 return Θ_v
```



21.43. ábra. Σ_1 és Σ_2.

A fenti algoritmusban szereplő BEILLESZT-HELYEK(Θ_v, Σ_v) függvény a Θ_v Petri-dobozba beilleszti a Σ_v doboz minden helyét, annak súlyozásával együtt. A CIMKEZSÁK(α) függvény meghatározza a paraméterül kapott α halmazbeli átmenetekhez rendelt címkék zsákját. Az ÖSSZEVONT-ÁTMENETET-KÉSZÍT(α) függvény összevonja az α átmenethalmaz elemeit egyetlen átmenetté, mely átmenetnek minden olyan él bemenő éle lesz, amely bemenő éle volt valamelyik α halmazbeli átmenetnek, és hasonlóan minden olyan él kimenő éle lesz, amely él valamely α halmazbeli átmenetnek kimenő éle volt. Végül a BEILLESZT-ÁTMENET$(\Theta_v, \textit{újátmenet})$ függvény beilleszti a paraméterként kapott átmenetet (az összes bemenő és kimenő élével együtt) a Θ_v Petri-dobozba.

21.5. példa. Példaként tekintsük a 21.43. ábrabeli Σ_1 és Σ_2 sima dobozokat, valamint egy olyan operátordobozt (például a 21.45. ábrán látható dobozt), amely két átmenetéhez rendelt átcímkézés:$\rho_1 = \rho_{id}$ (ezt használjuk majd Σ_1-hez), és $\rho_2 = \rho_{id} \cup \{(\{\alpha, \delta\}, \epsilon)\}$ (ezt használjuk majd Σ_2-höz). Itt mivel mindkét címkézés tartalmazza ρ_{id}-et, ezért az eredményül kapott Petri-dobozok tartalmazni fogják az eredeti hálók minden átmenetét. Ezen felül, mivel $(\{\alpha, \delta\}, \epsilon) \in \rho_2$ és a Σ_2 dobozbeli u és w átmenetekhez rendelt címke α és δ, ezért ezen átmenetek összevonásával képezni kell egy új átmenetet (az $\{u, w\}$ halmazhoz tartozó $\{\alpha, \delta\}$ címkezsák eleme lesz D_{ρ_2}-nek), melynek címkéje ϵ (azaz $\rho_2(\{\alpha, \delta\})$) lesz. Az interfész váltás után kapott két háló a 21.44. ábrán látható.



21.44. ábra. Az interfész váltás után Σ_1-ből kapott Θ_1, illetve Σ_2-ből kapott Θ_2.

Vizsgáljuk meg most a második műveletrészt, melyben az interfész váltás eredményeként kapott dobozokat kapcsoljuk össze a belépési és kilépési pontjaikon keresztül. Tegyük fel, hogy Ω egy operátordoboz, Σ pedig egy rendezett $\Omega - n$-es (amely a már átalakított dobozokat rendeli Ω megfelelő átmeneteihez). Az átmenetfinomítás során Ω minden helyére meg kell vizsgálni a megelőző és rákövetkező átmeneteket, pontosabban az átmenetekhez rendelt sima dobozokat. Majd az összes lehetséges módon venni kell a megelőző átmenetekhez rendelt dobozok egy-egy kilépési helyét, illetve a rákövetkező átmenetekhez rendelt dobozok egy-egy belépési helyét és egyesíteni kell ezeket a helyeket. Az összekapcsolt dobozok átmenetei, belső helyei, illetve az átmenetek és a belső helyek közötti élek változtatás nélkül átkerülnek az új Petri-dobozba.

ÁTMENETFINOMÍTÁS(Σ, Ω)

1 legyen S az Ω helyeinek halmaza
2 legyen T az Ω átmeneteinek halmaza
3 Θ = üres doboz
4 minden $t \in T$ átmenetre
5 $\quad\quad \Theta =$ BEILLESZT-BELSŐ-HELYEK-ÁTMENETEK$(\Theta, \Sigma(t))$
6 minden $s \in S$ helyre
7 $\quad\quad$ *Helyek-direktszorzata* $= \emptyset$
8 $\quad\quad$ minden $t \in {}^\bullet s$ átmenetre
9 $\quad\quad\quad$ **if** *Helyek-direktszorzata* $= \emptyset$

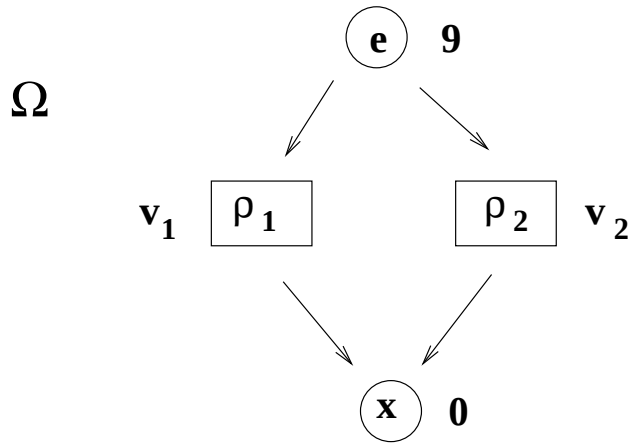
```
10              Helyek-direktszorzata ← •Σ(t)
11              else Helyek-direktszorzata =
                        Helyek-direktszorzata × (•Σ(t))
12      minden t ∈ s• átmenetre
13              if Helyek-direktszorzata = ∅
14                 Helyek-direktszorzata ← Σ(t)•
15                 else Helyek-direktszorzata ←
                           Helyek-direktszorzata × (Σ(t)•)
16      minden α ∈ Helyek-direktszorzata rendezett n-esre
17              újhely = Összevont-helyet-készít(α)
18              Θ = Beilleszt(Θ, újhely)
19  return Θ
```

Az algoritmusban szereplő Beilleszt-belső-helyek-átmenetek$(\Theta, \Sigma(t))$ függvény változtatás nélkül beilleszti a $\Sigma(t)$ Petri-doboz minden átmenetét, belső helyét, illetve az átmenetek és a belső helyek közötti éleket a Θ dobozba. Az Összevont-helyet-készít(α) függvény paramétere egy rendezett n-es, melynek minden komponense egy hely, a függvény összevonja ezeket a helyeket oly módon, hogy minden él, amely bemenő éle valamely komponens helynek, az bemenő éle lesz az új helynek, illetve hasonlóan minden él, amely kimenő éle volt valamely komponens helynek, az kimenő éle lesz az új helynek. A Beilleszt$(\Theta, \textit{újhely})$ függvény pedig beilleszti a paraméterül adott helyet az összes bemenő és kimenő élével együtt a Θ Petri-dobozba.

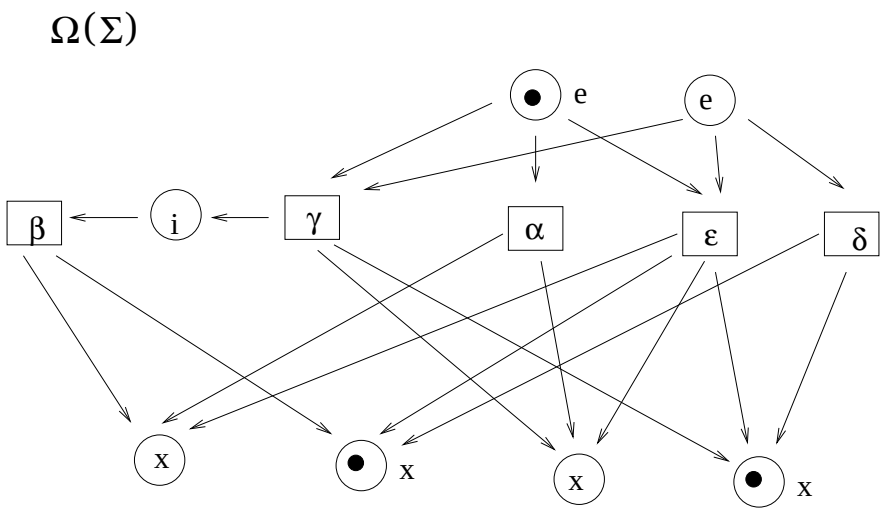
21.6. példa. Példaként tekintsük a 21.44. ábrán szereplő, a 21.5. példabeli interfész váltás után kapott Θ_1 és Θ_2 Petri-dobozokat és a 21.45. ábrán szereplő Ω operátordobozt. Az átmenet finomítás során Θ_1 és Θ_2 bemenő helyeit kell az összes lehetséges módon összekapcsolni (mivel az operátordobozban v_1 és v_2 is rákövetkezője a 9-es címkéjű helynek), illetve hasonlóan Θ_1 és Θ_2 kimenő helyeit is össze kell kapcsolni az összes lehetséges módon (mivel v_1 és v_2 is megelőzője a 0-ás címkéjű helynek). Az eredményül kapott háló a 21.46. ábrán látható.

21.10.1. Speciális operátordobozok

Az operátordobozok elsődleges célja, hogy segítségükkel leírható legyen, hogy egy adott programkonstrukciós műveletnek milyen konstrukciós művelet felel meg a Petri-dobozok szintjén. Ezáltal könnyebbé (sőt bizonyos mértékig automatikussá) válik egy bonyolult programhoz rendelt Petri-háló előállítása. Elegendő megadni a program alapvető alkotó részeihez tartozó sima Petri-



21.45. ábra. Ω operátordoboz az előbbi ρ_1 és ρ_2 átcímkézéssel.



21.46. ábra. Ω(Σ).

dobozokat továbbá azt, hogy ezen programrészekből milyen konstrukciós műveletekkel áll elő a teljes program. Ha minden konstrukciós művelethez adott a megfelelő operátordoboz, akkor a teljes programhoz tartozó Petri-doboz megkapható az alkotórészekhez tartozó dobozokból, elvégezve rajtuk az operátordobozok által definiált hálófinomításokat.

Egy konkrét esetben az *Act* halmaz adja meg a program, algoritmus legalapvetőbb alkotórészeit, eseményeit. Mivel párhuzamos folyamatokról van szó, ezért ezek közül az események közül egyszerre, egy időben több is végrehajtódhat (bizonyos esetekben akár némelyik esemény többször is), ezért ezen programok elemi lépéseit *Act*-beli elemek zsákjával modellezhetjük.

Ebből adódik, hogy az alapvető programrészekhez rendelt Petri-doboz a legtöbbször a 21.47. ábrán látható N_α szerkezetű, úgynevezett *alap doboz*.

21.79. definíció. *Legyen $\alpha \in Lab$ egy esemény, ekkor N_α (lásd 21.47. ábra) egy alap doboz. Az átmenethez rendelt címkézés a $\rho_\alpha = \{(\emptyset, \alpha)\}$ konstans címkézés.*

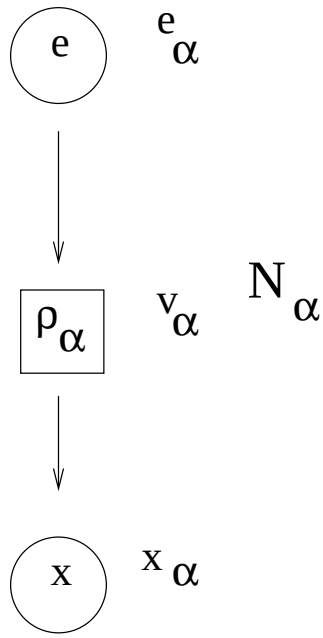
Érdemes megjegyezni, hogy N_α természetesen egy statikus doboz (azaz nem operátordoboz).

Az alapvető egységeknek megfelelő Petri-dobozok definiálása után a továbbiakban az általánosan használt konstrukciókhoz tartozó operátordobozokat definiáljuk. Ezek közül az első a párhuzamos kompozíciót leíró Ω_{II} doboz (lásd 21.48. ábra). Ez az operátordoboz két teljesen különálló másolatot készít a paraméterként adott két sima dobozról (abban az esetben is, ha a két doboz megegyezik), és beilleszti őket egyetlen hálóba.

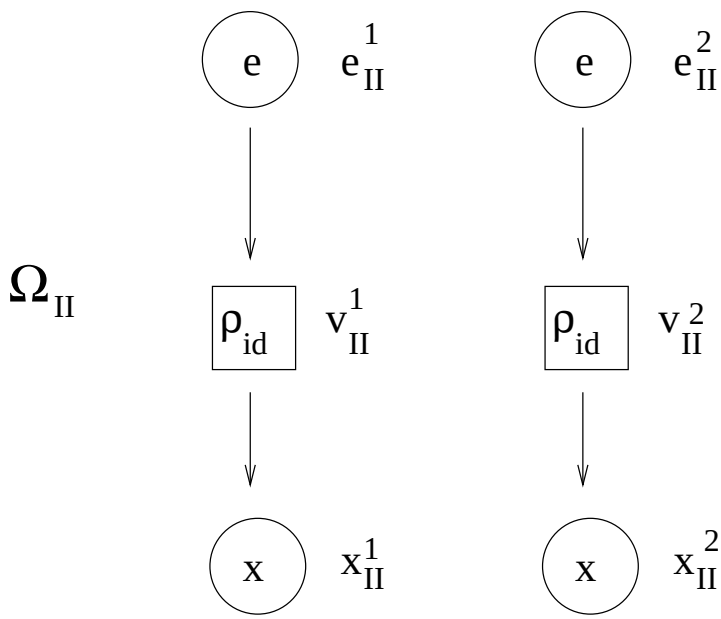
A második operátordoboz, amit definiálunk, az elágazást leíró $\Omega_\square$ doboz (lásd 21.51. ábra). Ennek az operátordoboznak az előbbihez hasonlóan két paramétere van, melyek belépési, illetve kilépési helyeit kapcsolja össze. Ez a doboz hasonlít a 21.6 példabeli operátordobozhoz, azzal a különbséggel, hogy itt minden átmenethez identikus átcímkézést rendelünk, azaz a paraméterként megadott dobozok átmenetein nem változtatunk.

A következő vizsgált operátor az $\Omega_;$ szekvenciális kompozíció (lásd 21.50. ábra), melynek szintén két paramétere van. A művelet az első paraméter kilépési helyeit kapcsolja össze a második paraméter belépési helyeivel, így modellezve a két alkotórész végrehajtásának szekvenciáját.

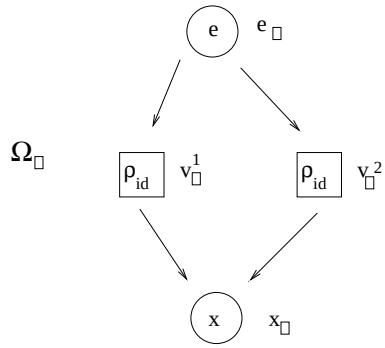
A vezérlési szerkezet operátorok közül utolsóként nézzük meg a ciklus $\Omega_{[*]}$ operátordobozát (lásd 21.52. ábra). Ez egy olyan ciklust modellez, melyben három különböző utasításunk van. Az első a ciklus inicializáló utasítása, a második a ciklusmag, a harmadik pedig egy lezáró utasítás. Megjegyezzük, hogy ez az operátor nem minden esetben ad biztonságos hálót eredményül,



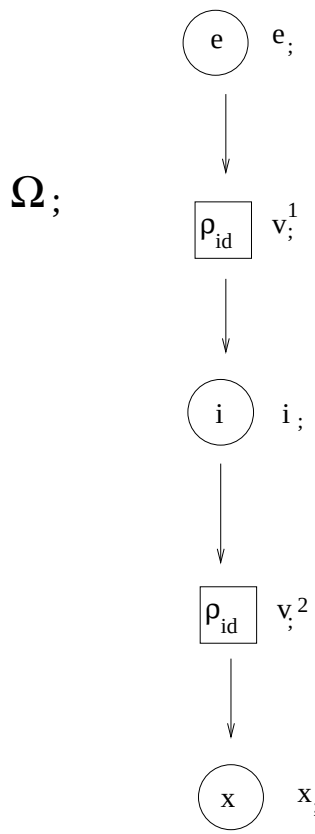
21.47. ábra. Az N_α sima doboz.



21.48. ábra. A párhuzamos kompozíciót leíró Ω_{II} operátordoboz.



21.49. ábra. Az elágazást leíró $\Omega_{\square}$ operátordoboz.



21.50. ábra. A szekvenciális kompozíciót leíró $\Omega_{;}$ operátordoboz.

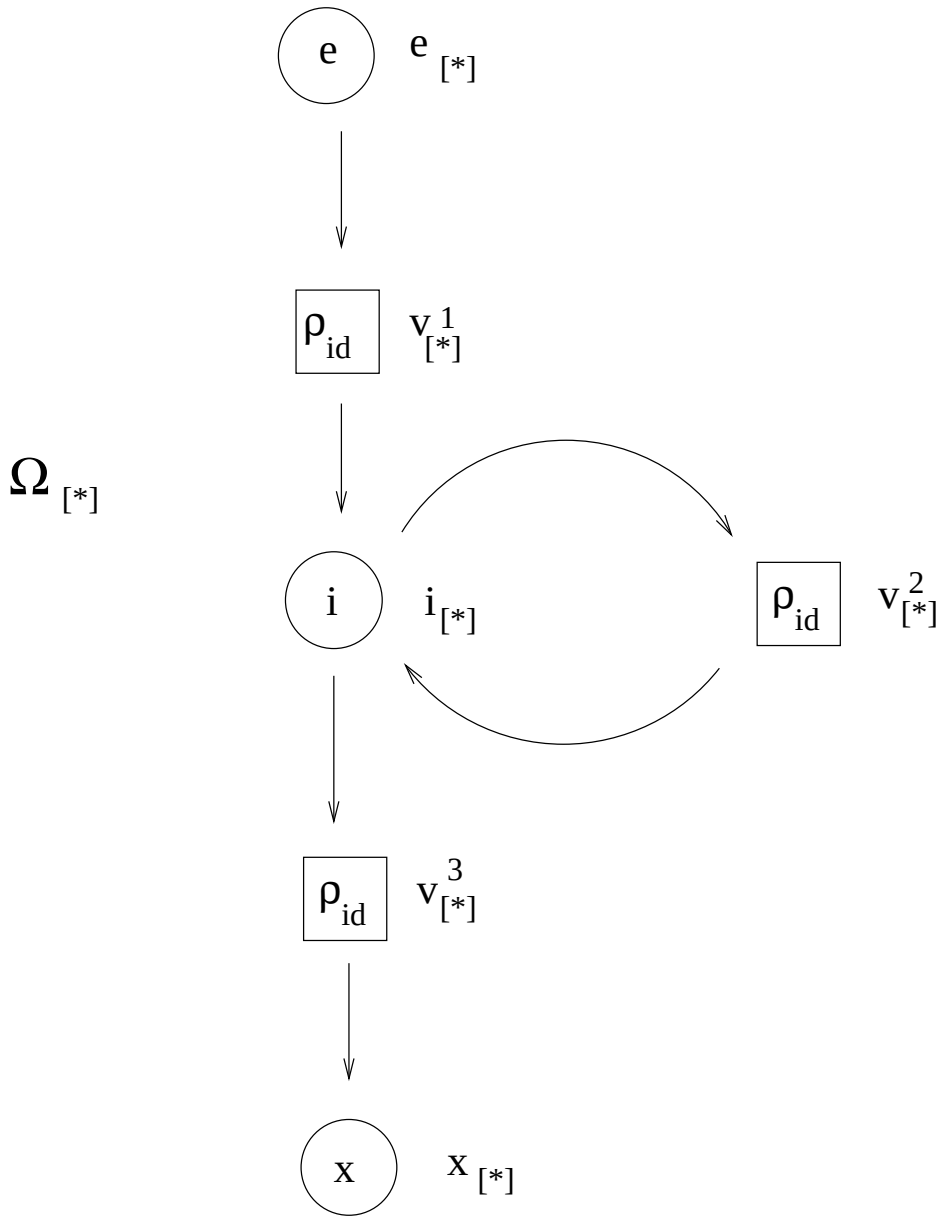
csak az biztosítható, hogy a kapott háló mindig 2-korlátos legyen. Megadható olyan (a bemutatottnál jóval bonyolultabb szerkezetű) ciklus operátor is, amely biztonságos hálót ad, de ennek tárgyalásától most eltekintünk.

Ezek után vizsgáljunk meg néhány olyan operátort, amelynek csak egyetlen paramétere van. Az ilyen dobozok az eddigiekkel ellentétben nem különböző programrészek valamely programkonstrukció szerinti összekapcsolását modellezik, hanem csak egy adott programrészt leíró Petri-dobozon végeznek változtatásokat. A legegyszerűbb ilyen változtatás az $\Omega_{[f]}$ átnevezés (lásd 21.52. ábra), amely csak az átmenetek címkéit változtatja meg. Az operátordobozban szereplő átcímkézés a $\rho_{[f]} = \{(\{\alpha\}, f(\alpha)) | \alpha \in Lab\}$ átcímkézés, ahol $f : Act \to Act$ egy függvény, melyet kiterjesztünk a $Lab = mult(Act)$ halmazra $(\forall \alpha \in Lab : f(\alpha)(f(a)) = \alpha(a))$.

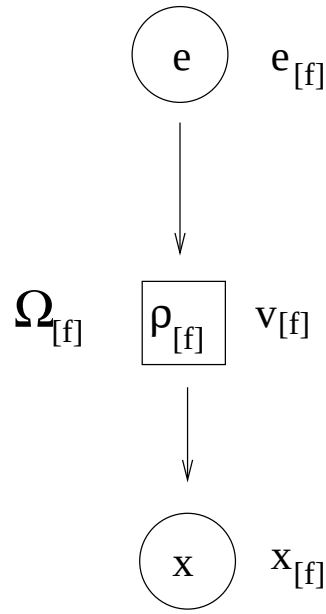
Némileg bonyolultabb művelet a szinkronizáció, amelyet mindig egy adott $a \in Act$ esemény szerint végzünk el. Ez a művelet új, összevont átmenetet készít bármely két olyan átmenethez, amelyre teljesül, hogy közülük az egyikhez olyan címke van rendelve, amely tartalmazza a-t, míg a másikhoz olyan, amelyik tartalmazza $\hat{a}$-t (ahol $\hat{}$ a 21.54. definícióban szereplő, párokat definiáló függvény). Ha az egyik átmenethez rendelt címke $\alpha + \{a\}$, a másikhoz rendelt pedig $\beta + \{\hat{a}\}$, akkor az új átmenethez az $\alpha + \beta$ címkét rendeljük (itt $+$ a zsákokra vonatkozó összeadást jelöli). A művelet az eredeti átmeneteket változatlanul hagyja, addig működik, amíg van két megfelelő átmenet, melyekből még nem készítettünk újat. Rekurzívan működik, egy már újonnan létrehozott átmenethez is készít új összevont átmenetet, ha létezik hozzá megfelelő pár. Egy adott a esemény szerinti szinkronizációt ír le a 21.53. ábrán látható $\Omega_{sy\ a}$ operátordoboz. Itt $\rho_{sy\ a}$ a legszűkebb olyan átcímkézés, amely tartalmazza ρ_{id}-et, és ha $(\Gamma, \alpha + \{a\}) \in \rho_{sy\ a}$ és $(\Delta, \beta + \{\hat{a}\}) \in \rho_{sy\ a}$, akkor $(\Gamma + \Delta, \alpha + \beta) \in \rho_{sy\ a}$.

A következő művelet nagyon hasonló, az egyetlen különbség az, hogy itt egy átmenetet önmagával is szinkronizálhatunk, azaz ha létezik olyan átmenet, melyhez rendelt címke $\alpha + \{a, \hat{a}\}$ alakú, akkor az átmenet alapján létrehozunk egy új átmenetet α címkével. A műveletet önszinkronizációnak nevezzük (lásd 21.54. ábra, $\Omega_{sy'\ a}$ háló). A műveletben szereplő $\rho_{sy'\ a}$ átcímkézés a legszűkebb olyan átcímkézés, amely tartalmazza $\rho_{sy\ a}$-t, és ha $(\Gamma, \alpha + \{a, \hat{a}\}) \in \rho_{sy'\ a}$, akkor $(\Gamma, \alpha) \in \rho_{sy'\ a}$.

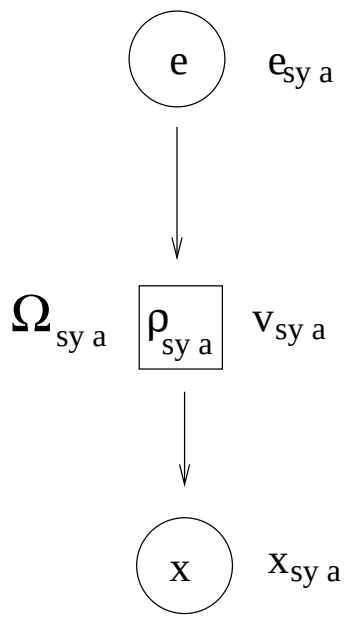
Vizsgáljuk meg ezek után a megszorítás műveletét (lásd 21.55. ábra), melyet szintén egy adott $a \in Act$ esemény szerint végzünk el. A művelet eltávolítja a paraméteréül adott sima dobozból az összes olyan átmenetet, melynek címkéjében szerepel a vagy $\hat{a}$. Itt $\rho_{rs\ a} = \rho_{mult(A \setminus \{a, \hat{a}\})}$. Megjegyezzük, hogy a megszorítás során a hálóból akár az összes átmenet törlődhet (például lásd 21.55. ábrán látható Stop doboz).



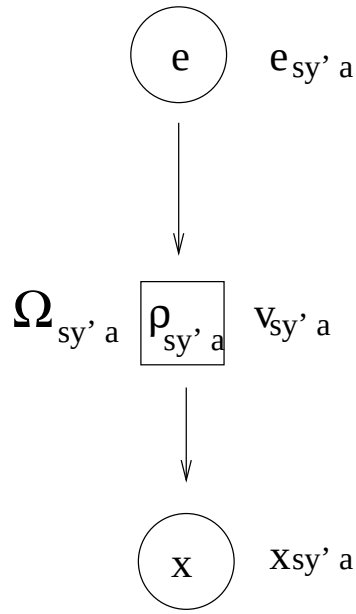
21.51. ábra. Az elágazást leíró $\Omega_{\square}$ operátordoboz.



21.52. ábra. a) a ciklust leíró $\Omega_{[*]}$ operátordoboz. **b)** az átnevezést leíró $\Omega_{[f]}$ operátordoboz.



21.53. ábra. Az a esemény szerinti szinkronizációt leíró Ω_{sya} operátordoboz.



21.54. ábra. Az a esemény szerinti önszinkronizációt leíró $\Omega_{sy'\ a}$ operátordoboz.

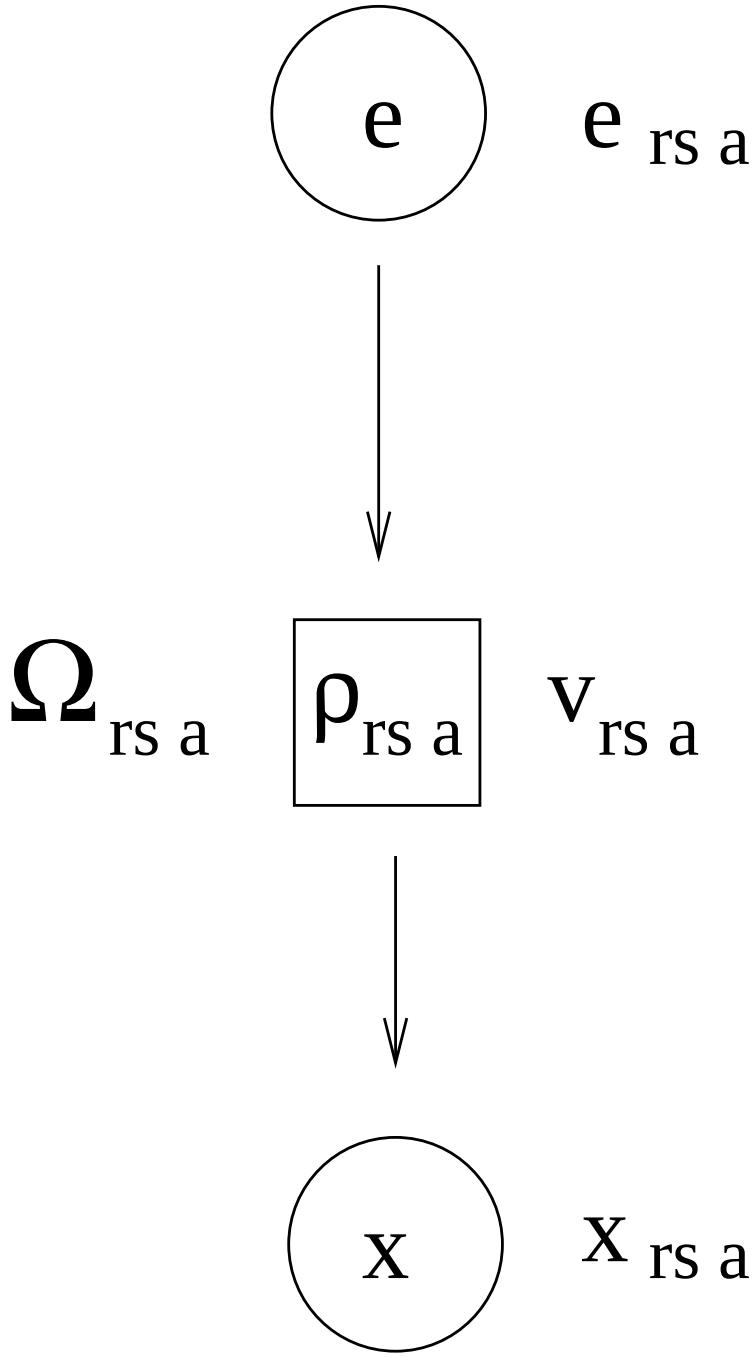
Végezetül vizsgáljuk meg a hatókör operátort (lásd 21.56. ábra), amelyet ugyancsak egy adott $a \in Act$ esemény szerint végzünk el, és hatását tekintve megegyezik a szinkronizáció és a megszorítás műveletének kompozíciójával. Az operátordobozban szereplő átcímkézés a $\rho_{[a:]} = \{(\Gamma, \alpha) \mid \alpha(a) = \alpha(\hat{a}) = 0 \wedge (\Gamma, \alpha) \in \rho_{sy\ a}\}$ átcímkézés.

A definiált operátordobozok alapján létrehozhatunk egy olyan absztrakt nyelvet, melyben a dobozok által modellezett operátorok a megengedettek. Így, ha egy programot vagy algoritmust le tudunk írni ezen az absztrakt nyelven, akkor a hozzá tartozó Petri-háló automatikusan generálható az operátordobozok által leírt műveletek segítségével. Nézzük meg ennek az absztrakt nyelvnek a pontos definícióját.

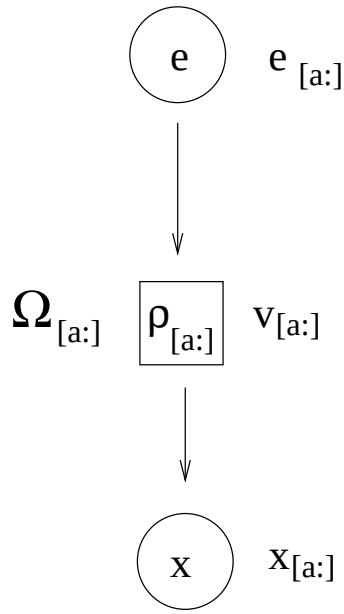
21.80. definíció. (Petri-doboz kifejezések szintaxisa).

$$E ::= \alpha \mid E||E \mid E\Box E \mid E;E \mid [E * E * E] \mid$$
$$X \mid E\ \texttt{sy}\ a \mid E[f] \mid E\ \texttt{rs}\ a \mid [a:E] \quad .$$

A definícióban $\alpha \in Lab$ egy alap esemény, amelyhez egy alap doboz rendelhető. Az operátorokat két részre bonthatjuk a három bináris operátor (a || párhuzamos operátor, az $\Box$ elágazás, a ; szekvencia) és a 3-áris [**] ciklus



21.55. ábra. Az a esemény szerinti megszorítás műveletet leíró $\Omega_{rs\ a}$ operátordoboz. **b)** Átmenet nélküli Stop doboz.



21.56. ábra. Az a esemény szerinti hatókör műveletet leíró $\Omega_{[a:]}$ operátordoboz.

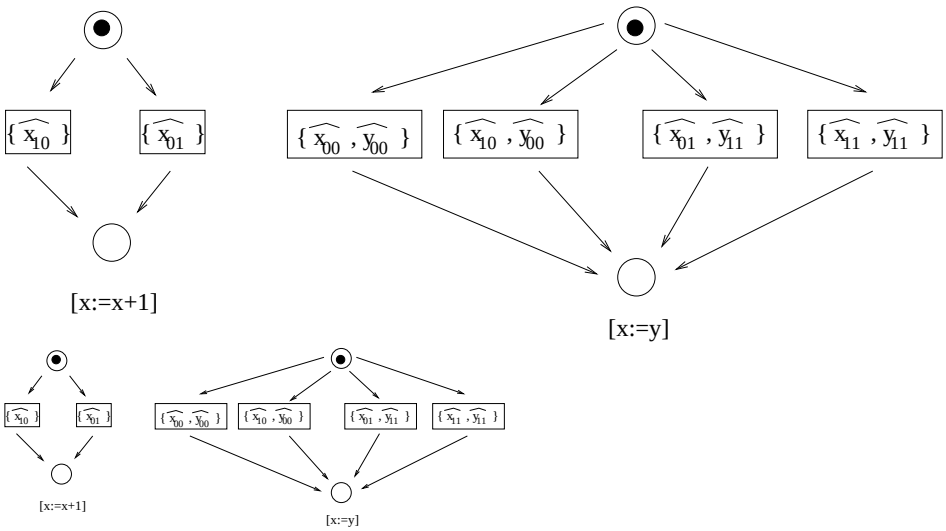
operátor *vezérlő szerkezet operátor*. Míg az unáris operátorok, azaz az $[f]$ alap átnevezés, a $sy\ a$ szinkronizáció, a $rs\ a$ megszorítás és a $[a:]$ hatókör operátor *kommunikációs interfész operátorok*. Ezekhez értelemszerűen az előzőleg leírt operátordobozokat rendelhetjük hozzá. Végezetül az X egy változó az előre definiált változóhalmazból, melyhez egyértelműen hozzárendelődik egy E Petri-doboz kifejezés. A változók a különböző szintű absztrakciót segítik elő.

21.10.2. Programok modellezése

Ebben a részben megvizsgáljuk, hogyan lehet Petri-hálók segítségével modellezni konkrét programokat. Tekintsük először az alábbi egyszerű párhuzamos programot.

21.7. példa.

```
...
 begin var x:{0,1};
  [x:=x+1] || [x:=y]
 end
...
```



21.57. ábra. [x:=x+1] || [x:=y] ábrázolása.

A programban az egyszerűség kedvéért olyan változókat használunk, melyek a {0,1} értékeket vehetik fel és feltételezzük, hogy y egy korábban deklarált (szintén $\{0,1\}$ halmazbeli) változó. A változók típusából adódóan az összeadás modulo 2 értendő. A [] jel az atomi, felbonthatatlan akciókat jelöli.

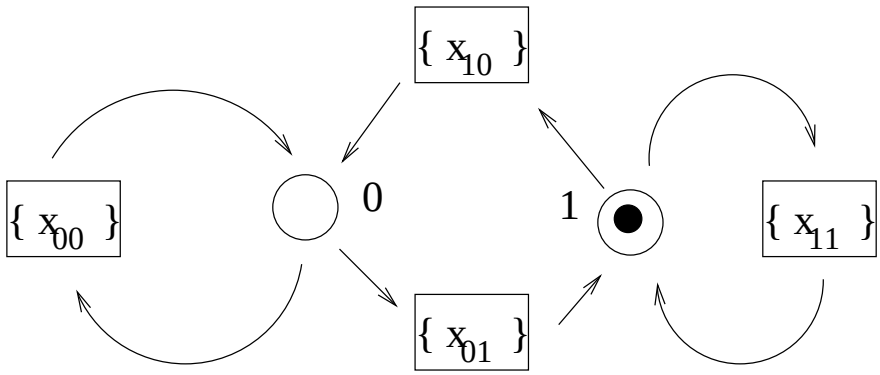
A programot legegyszerűbben úgy modellezhetjük, hogy bevezetjük az $x_{vw}, \hat{x_{vw}}$ akciókat, ahol $v, w \in \{0,1\}$. Egy ilyen akció átállítja az x értékét v-ről w-re, illetve, ha $v = w$, akkor csak lekérdezi, hogy valóban v a változó értéke. A $\hat{}$-vel megjelölt akciókat elképzelhetjük úgy, mint egy kérelmet (például az operációs rendszer felé) a leírt akció valós, fizikai végrehajtására, a jelöletleneket pedig, mint egy ilyen valós végrehajtást.

Ezek után a vizsgált program atomi akcióit a következőképpen írhatjuk át általunk már könnyen kezelhető akciókká:
$[x := x + 1] \rightsquigarrow \{\hat{x_{01}}\}\square\{\hat{x_{10}}\}$,

$[x := y] \rightsquigarrow \{\hat{x_{00}}, \hat{y_{00}}\}\square\{\hat{x_{10}}, \hat{y_{00}}\}\square\{\hat{x_{01}}, \hat{y_{11}}\}\square\{\hat{x_{11}}, \hat{y_{11}}\}$.

Az x_{vw} és $\hat{x_{vw}}$ akciókhoz alap dobozokat rendelünk. A leírt operátordobozok műveleteinek végrehajtása után megkapjuk az $[x := x + 1]||[x := y]$ programhoz tartozó Petri-hálót (lásd 21.57. ábra).

Ahhoz azonban, hogy egy program valós, fizikai működését modellezni tudjuk, vizsgálni kell az egyes változók viselkedését is. Azaz azt is modelleznünk kell, hogy egy változó milyen értékeket vehet fel, és milyen vál-



21.58. ábra. A bináris x változó ábrázolása.

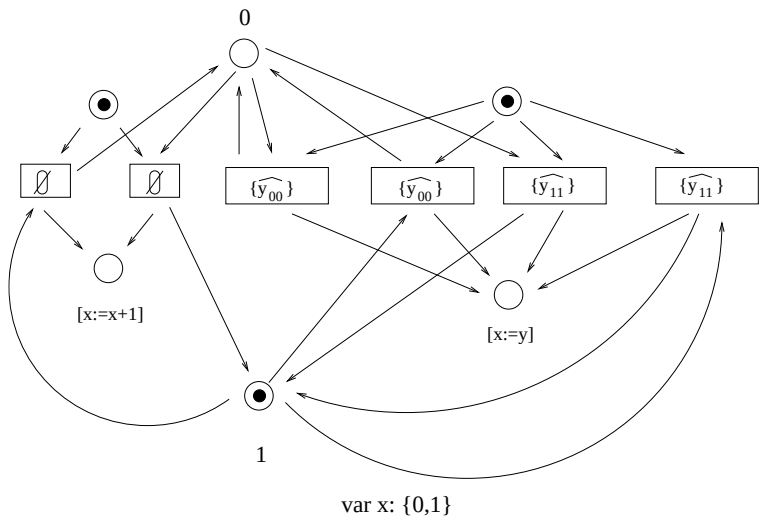
toztatásokat végezhetünk rajta (hogyan változtathatjuk meg az értékét). Az $x \in \{0,1\}$ esetén ezt írja le a 21.58. ábra, amelynél az egyszerűség kedvéért azt feltételezzük, hogy a változó kezdeti értéke 1.

A valódi működést modellező hálót úgy kaphatjuk meg a 21.57. ábrán és a 21.58. ábrán látható hálóból, hogy vesszük a két háló párhuzamos kompozícióját és az így kapott hálóra elvégezzük mindegyik x_{vw} atomi akció szerint a hatókör operátort. A végeredményként kapott hálót mutatja be a 21.59. ábra.

A bemutatott programhoz nem valós programozási nyelvet használtunk, de a leírt kód könnyen implementálható. Most definiálunk egy olyan absztrakt programozási nyelvet, melyben leírt utasításokra teljesül egyrészt, hogy könnyen implementálhatóak valós konkurens programozási nyelvekkel, másrészt hogy szemantikájuk könnyen megadható Petri-doboz kifejezésekkel. Ezt az absztrakt nyelvet *Razor* nyelvnek fogjuk nevezni.

21.81. definíció. (*Razor* konkurens programozási nyelv szintaxisa).

1. $Program ::= Block$
2. $Block ::= \texttt{begin}\ Body\ \texttt{end}$
3. $Body ::= Decl; Body \mid Com$
4. $Decl ::= \texttt{var}\ id : Set \mid \texttt{var}\ chanid : \texttt{chan}\ Capacity\ \texttt{of}\ Set$
5. $Com ::= Block \mid [Act] \mid Com_1; Com_2 \mid Com_1 || Com_2 \mid$
   $\texttt{if}\ GC_1 \square \ldots \square GC_m\ \texttt{fi} \mid \texttt{do}\ GC_1 \square \ldots \square GC_m\ \texttt{od}$
6. $Act ::= id := Expr \mid chanid?x \mid chanid!Expr$
7. $GC ::= GC; Com \mid [Bexpr] \mid [Bexpr; Act]$



21.59. ábra. A program működését leíró háló.

8. $Expr ::= id \mid Const \mid Expr\ Binop\ Expr \mid Unop\ Expr \mid Bexpr$

9. $Bexpr ::= boolid \mid \texttt{false} \mid \texttt{true} \mid Bexpr\ Boolop\ Bexpr \mid \neg Bexpr \mid Expr\ Relop\ Expr$

10. $Binop ::= + \mid - \mid * \mid \texttt{mod} \mid \texttt{div}$

11. $Unop ::= + \mid -$

12. $Boolop ::= \wedge \mid \vee \mid \rightarrow$

13. $Relop ::= = \mid \neq \mid < \mid \leq$

Mint a definícióban látható, a *Razor* nyelv blokkokból áll. Egy blokk tartalmazhat deklarációkat, beágyazott blokkokat, atomi utasításokat, szekvenciát, párhuzamos utasítást, elágazást és ciklust. Az elágazás és a ciklus őrfeltételes utasításokat tartalmaz (GC), amelyek mindenképpen egy (nulladrendű) logikai kifejezéssel kezdődnek (vagy egyszerűen csak egyetlen logikai kifejezésből állnak) és csak akkor hajthatóak végre, ha a megadott logikai feltétel teljesül. A nyelvben egy speciális típust vezetünk be a párhuzamos folyamatok közötti kommunikáció leírására. Ez a típus a csatorna. A csatornán kétféle műveletet végezhetünk, elküldhetünk rá egy kifejezés által definiált értéket, vagy levehetünk róla egy értéket egy változóba. Ezek a műveletek atomiak. Ezen kívül csak egyetlen atomi műveletet vezetünk be, az értékadást,

amelynek során egy kifejezés által definiált értéket teszünk egy változóba. Egy kifejezésben csak az összeadás, kivonás, szorzás, osztás, modulo képzés megengedett.

Egy *Razor* nyelvbeli kifejezést könnyedén át lehet írni Petri-doboz kifejezéssé. A pontos szabályokra jelen fejezetben nem térünk ki, de megjegyezzük, hogy valójában csak az őrfeltételes utasítás és a csatornaműveletek esetén kell átírási szabályokat megfogalmaznunk, hiszen a többi elem (részben az előzőekben tekintett, 21.57. ábrán bemutatott példa alapján) automatikusan megfeleltethető egy-egy Petri-doboz kifejezésbeli elemnek.

Látható, hogy a definiált nyelv még mindig igen leegyszerűsített. Mégis ezen a nyelven már nagyon sok algoritmus formalizálható. Vizsgáljuk meg példaként a kölcsönös kizárást, ahol két folyamat ugyanazt a közös erőforrást próbálja meg használni a *KritikusSzakasz* műveletrészben (például ugyanarra a nyomtatóra próbálnak írni, vagy ugyanazt a fájlt megnyitni). Ezért garantálnunk kell, hogy egyszerre csak az egyikük léphessen be ebbe a végrehajtási szakaszába. Erre a problémára ad megoldást Peterson jól ismert algoritmusa. Az algoritmust a következőképpen lehet leírni *Razor* nyelven.

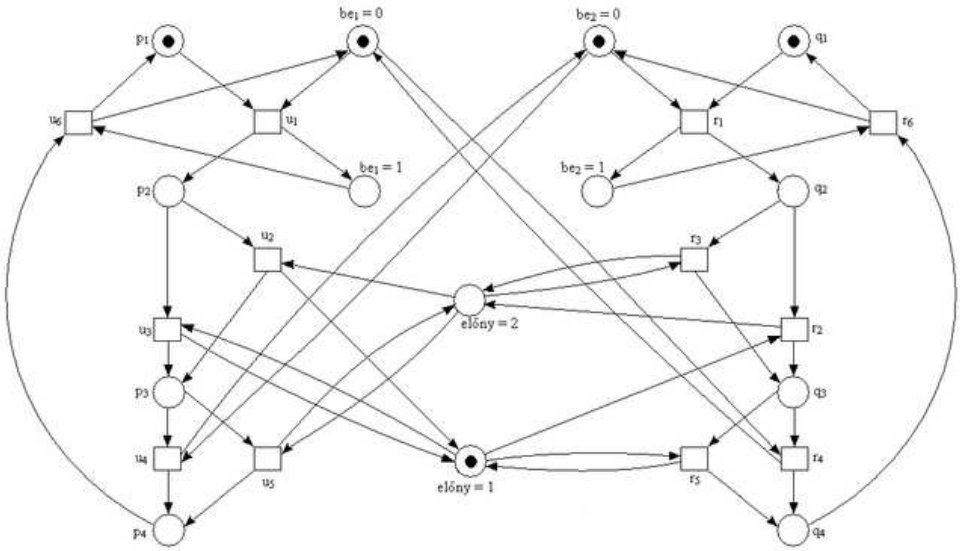
begin
var $be_1,\ be_2 : \{0,1\}$ (**init** 0); $előny : \{1,\ 2\}$ (**init** 1);

do		**do**	
$[be_1 := 1];$	a_1	$[be_2 := 1];$	b_1
$[előny := 1];$	a_2	$[előny := 2];$	b_2
if $[be_2 = 0]$		**if** $[be_1 = 0]$	
$\square\ [(előny \neq 1)]$ **fi**;	a_3	$\square\ [(előny \neq 2)]$ **fi**;	b_3
$KritikusSzakasz_1;$		$KritikusSzakasz_2;$	
$[be_1 := 0];$	a_4	$[be_2 := 0];$	b_4
od		**od**	

end

Az algoritmusnak megfelelő Petri-háló automatikusan generálható. Az eredményül kapott hálót mutatja be egyszerűsített formában a 21.60. ábra. Az ábrán az u_1 átmenet felel meg az algoritmusbeli a_1 műveletnek, az u_2 és u_3 átmenet az a_2 műveletnek, az u_4 és u_5 átmenet az a_3 műveletnek, és az u_6 átmenet az a_4 műveletnek. A $KritikusSzakasz_1$ programrészbe való belépést pedig a p_4 hely modellezi. Szimmetrikusan az r_j átmenetek felelnek meg a megfelelő b_i műveletnek és a $KritikusSzakasz_2$ programrészbe való belépést a q_4 hely modellezi.

Az így kapott háló segítségével megvizsgálhatjuk az algoritmus különböző tulajdonságait. Példaként bebizonyítjuk, hogy az algoritmus valóban teljesíti azon feltételt, hogy a két párhuzamos folyamat nem léphet be egyszerre a kritikus szakaszába. Ez a Petri-háló szintjén azt jelenti, hogy nem lehet egyszerre



21.60. ábra. A Peterson-algoritmust reprezentáló Petri-háló.

súly a p_4 és q_4 helyek mindegyikén. Ennek bizonyításához először tekintsük a következő helyhalmazokat.
$\theta_1 = \{p_1, p_2, p_3, p_4\}$,
$\theta_2 = \{q_1, q_2, q_3, q_4\}$,
$\theta_3 = \{előny = 1, előny = 2\}$,
$\theta_4 = \{be_1 = 0, be_1 = 1\}$,
$\theta_5 = \{be_2 = 0, be_2 = 1\}$,
$\theta_6 = \{be_1 = 0, p_2, p_3, p_4\}$,
$\theta_7 = \{be_2 = 0, q_2, q_3, q_4\}$.

Könnyen belátható, hogy ezek a halmazok helyinvariánsokat alkotnak, méghozzá a kezdeti súlyozás miatt oly módon, hogy az egyes halmazokban lévő helyek közül pontosan az egyiken van súly. Emellett az is könnyen belátható, hogy a következő két helyhalmaz egy-egy csapdát alkot, melyekben kezdetben egy-egy súly van.

$F_1 = \{be_1 = 0, p_2, előny = 1, q_3\}$,
$F_1 = \{be_2 = 0, q_2, előny = 2, p_3\}$.

Ezek után tegyük fel, hogy valamely, a kezdeti súlyozásból elérhető M súlyozás a p_4 és q_4 helyek mindegyikére helyez súlyt. Felhasználva, hogy θ_6 és

θ_7 helyinvariáns, ebből az következik, hogy a többi hely, amely a θ_6 vagy θ_7 halmazban van, nem tartalmaz súlyt, azaz

$$M(be_1 = 0) = M(p_2) = M(p_3) = M(be_2 = 0) = M(q_2) = M(q_3) = 0 \ .$$

Mivel θ_3 is helyinvariáns, ezért az *előny* $= 1$, illetve *előny* $= 2$ helyek közül csak az egyiken lehet súly. Ebből viszont az következik, hogy az F_1, F_2 csapdák valamelyike üres, ami ellentmond annak, hogy egy olyan helyhalmaz, ami csapdát alkot és kezdetben legalább egy súlyt tartalmaz, nem válhat súlyozatlanná egy Petri-háló működése során. Tehát végeredményképpen azt kaptuk, hogy nem létezik olyan elérhető súlyozás, amely a p_4 és q_4 helyek mindegyikére helyez súlyt, azaz Peterson-algoritmusa valóban teljesíti a kölcsönös kizárás követelményét.

Petri-hálók viselkedését a PEP eszköz segítségével automatikusan is vizsgálhatjuk. Ez az eszköz képes egy Petri-doboz kifejezés alapján elkészíteni egy Petri-hálót, tudja szimulálni annak működését, és bizonyos tulajdonságok (például elérhetőség, holtpontmentesség) vizsgálatára is használható. Ezen felül lineáris, illetve elágazó idejű temporális logikai kifejezések teljesülését is képes ellenőrizni. Például a Peterson-algoritmus esetén a PEP eszköz segítségével automatikusan megvizsgálhatjuk, hogy teljesülhet-e a $\neg(\Diamond(p_4 \wedge q_4))$ temporális logikai kifejezés. A $\Diamond$ temporális logikai operátor jelentése informálisan: „valaha a jövőben teljesül”, így az előbbi formula pontosan az általunk vizsgált tulajdonságot írja le, hogy sohasem lehet a p_4 és q_4 helyeken egyszerre súly.

Gyakorlatok

21.10-1. Rajzoljuk fel, hogy a párhuzamos, illetve a szekvenciális kompozíciót alkalmazva a 21.43. ábrán látható hálókra milyen Petri-hálót kapunk. Ehhez használjuk fel a 21.48. ábra, illetve a 21.50. ábra jobb oldalán található operátordobozokat.

21.10-2. Írjuk fel a *Razor* nyelven megadott Peterson-algoritmust Petri-doboz kifejezésekkel.

Megjegyzések a fejezethez

A Petri-háló fogalmát C. A. Petri vezette be 1962-ben [235]. Az eredeti modellt számos irányban továbbfejlesztették, például a színezett Petri-hálók, vagy a többszintű hálók bevezetésével. Ezen kiterjesztések közül többet is ismertet [229].

Az alapfogalmak bemutatásánál elsősorban Bagyinszkiné Orosz Anna jegyzetére [15] és T. Murata összefoglaló cikkére [209] támaszkodtunk. Az

érdeklődő Olvasó megtalálja a 21.48. és 21.49. tételek bizonyítását [15]-ben. A Petri-hálók és a formális nyelvek kapcsolatát részletesen tárgyalja [16].

Az elosztott programok tulajdonságainak vizsgálatára bevezetett Petri-dobozok modelljét az azt kidolgozó E. Best és munkatársainak cikkei [28] és könyve [27], illetve az ezek alapján készült Peptool szimulációs és elemző rendszer alapján ismertettük. A fejezetben bemutatott algoritmusokat modellező Petri-hálókat is Peptool [233] segítségével készítettük. A hálókat leíró Peptool fájlokat a hálózaton keresztül hozzáférhetővé tesszük az érdeklődő Olvasó számára [241], aki ily módon az algoritmusok működését szimulációs környezetben maga is könnyen kipróbálhatja és elemezheti.

Elosztott algoritmusok Petri-hálókkal történő modellezése során gyakran teljesül az, hogy egy-egy helyen egynél több súly egyszerre sohasem lehet [247]. A kapacitáskorláttal rendelkező hálók azonban mindig helyettesíthetőek ekvivalens hálóval (21.6. tétel).

Helyinvariánsok (P-invariáns) és T-invariánsok lineáris algebrai eszközökkel történő meghatározását írja le [229].

Ez a fejezet a T037742 sz. OTKA pályázat és az MTA Bolyai János Kutatási Ösztöndíj keretében készült.

22. Szisztolikus rendszerek

A szisztolikus rács – a speciális feladatot ellátó számítógépek legtökéletesebb formája – legegyszerűbb esetben csupán egyetlen számítási művelet ismételt végrehajtására alkalmas. Ennek ellenére számos, gyakorlati szempontból is jelentős alkalmazási területe van, főként az iteratív módszereket alkalmazó numerikus matematika, kombinatorikus optimizálás, lineáris algebra, algoritmikus gráfelmélet, kép- és jelfeldolgozás, nyelv- és szövegfeldolgozás stb. területén.

Egy szisztolikus rács felépítése egészen pontosan megfeleltethető egy végrehajtásra váró algoritmus szerkezetének úgy, hogy minden egyes számítás helye és időpontja egyszer és mindenkorra meghatározott, az egymással kommunikáló cellák közvetlenül és permanens módon egymáshoz vannak kapcsolva, így kommunikációs csatornák létrehozása feleslegessé válik. Az algoritmust közvetlenül hardver valósítja meg. Emiatt a szisztolikus algoritmusokat ebben az összefüggésben „hardver-algoritmusként” is szokták emlegetni.

A „szisztolikus algoritmusok” kifejezés tehát nem egy bizonyos, jól körülhatárolt számítási feladatra adott megoldások sokaságát jelenti (mint például a rendezési algoritmusok esetében), hanem sokkal inkább egy sajátos feladatmeghatározási-, programozási-, illetve számítási stílust határoz meg. Igen sokféle, különböző felhasználási területhez tartozó algoritmus lehet „szisztolikus jellegű”, ami nem jelenti feltétlenül azt, hogy ezen területek összes ismert algoritmusa szisztolikus feldolgozásra alkalmas alakra hozható lenne.

Ennek a fejezetnek nem célja, hogy „a szisztolikus algoritmusokat”, vagy akár csak a „legfontosabb szisztolikus algoritmusokat” bemutassa. A cél az, hogy néhány egyszerű, de tipikus példa segítségével megalapozzuk a szisztolikus algoritmusok általános megértését.

A fejezet öt alfejezetből áll. Az alapfogalmak (22.1. alfejezet) után a téridő leképezést (22.2. alfejezet), majd a be/kiviteli sémát (22.3. alfejezet) mutatjuk be. A 22.4. alfejezetben a vezérlési szempontokat, a 22.5. alfejezetben pedig a lineáris szisztolikus rácsokat tárgyaljuk.

22.1. A szisztolika alapfogalmai

Maga a ***szisztolikus*** elnevezés a szisztolikus architektúrák működési elvéből származik. Szisztolikus munkamódon a futószalag-elv, illetve a térbeli párhuzamosság intenzív együttes alkalmazását kell értenünk, melyet egy globális, és teljes mértékben szinkronizált órajel irányít. Ez eredményezi az adatfolyamoknak az összekapcsolt cellák hálózatán keresztül történő ritmikus áramlását, az emberi szervezetben keringő vérhez hasonlóan, melyet a szív a vérereken keresztül a test különböző pontjai felé küld. A futószalag-elv nem csupán egyetlen irányban érvényesül, hanem a különböző irányba haladó, a szisztolikus rács celláiban egymást keresztező adatfolyamok mindegyikére vonatkozik.

Egy ***szisztolikus rendszer*** általában egy *gazdagépből*, illetve a tulajdonképpeni szisztolikus rácsból áll. A gazdagépnek elvileg alárendelt szerepe van; csupán a szisztolikus rács működésbe hozatalára, továbbá adatokkal való ellátására szolgál. A ***szisztolikus rács*** egy sajátos, cellákból álló hálózatként is felfogható, mely a masszív párhuzamosságnak köszönhetően az adatorientált műveleteket igen nagy sebességgel hajtja végre. Egy szisztolikus rács celláinak egységes működése révén végrehajtott program adja magát a ***szisztolikus algoritmust.***

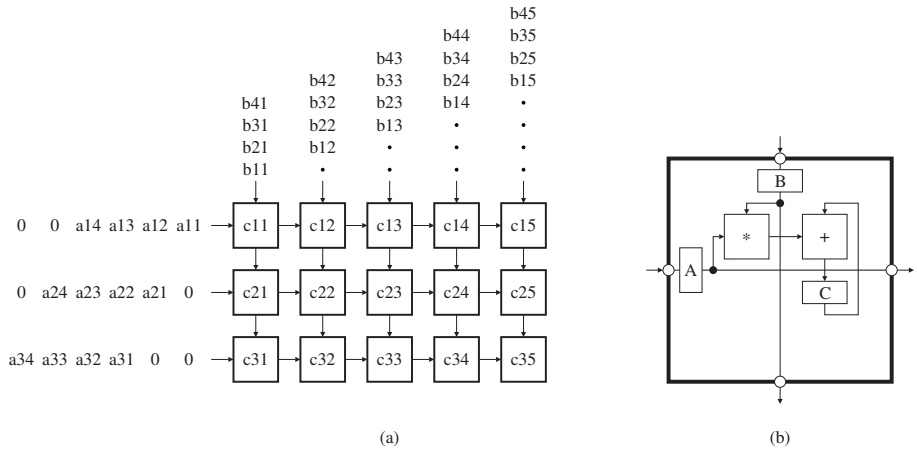
Bármennyire is különböznek egymástól a szisztolikus rácsok, mégis jó néhány közös tulajdonsággal rendelkeznek: ezek a diszkrét időséma, a szinkron munkamód, a szabályos (általában két dimenziós) mértani felépítés, csak közvetlenül szomszédos cellák között fennálló kommunikáció, valamint a legegyszerűbb vezérlési mechanizmusok alkalmazása.

A fejezet további részében a szisztolikus rácsokkal kapcsolatos alapvető jelenségeket fogjuk egy jellemző példán keresztül bemutatni és megvilágítani. Egy számítási feladatra gyakran többféle megoldás kínálkozik, azaz különböző szisztolikus rácsokat találhatunk, melyek közül a legjobbak általában igen bonyolultak. A továbbiakban ezért nem a legjobb változat bemutatását tűztük ki célul, hanem a legfontosabb elvek tömör és intuitív módon való bemutatását.

22.1.1. Bevezető példa: mátrixok szorzása

A 22.1. ábrán egy 15 *cellából* álló, téglalap alakú *szisztolikus rács* látható, amely egy $3 \times N$ méretű A mátrixnak egy $N \times 5$ méretű B mátrixszal való összeszorzására alkalmas. Az N paraméter a 22.1. ábrán látható szisztolikus rács *felépítése* szempontjából semmiféle szerepet nem játszik, viszont lényeges a *beviteli séma,* valamint az algoritmus *futási ideje* szempontjából.

A beviteli séma az $N = 4$ értéknek felel meg. A 22.1. ábra a következő



22.1. ábra. Téglalap alakú szisztolikus rács mátrixszorzáshoz: **(a)** A rács felépítése és a beviteli séma; **(b)** cellaszerkezet.

konkrét feladat megoldását mutatja:

$$A \cdot B = C \ ,$$

ahol

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{pmatrix} \ ,$$

$$B = \begin{pmatrix} b_{11} & b_{12} & b_{13} & b_{14} & b_{15} \\ b_{21} & b_{22} & b_{23} & b_{24} & b_{25} \\ b_{31} & b_{32} & b_{33} & b_{34} & b_{35} \\ b_{41} & b_{42} & b_{43} & b_{44} & b_{45} \end{pmatrix} \ ,$$

$$C = \begin{pmatrix} c_{11} & c_{12} & c_{13} & c_{14} & c_{15} \\ c_{21} & c_{22} & c_{23} & c_{24} & c_{25} \\ c_{31} & c_{32} & c_{33} & c_{34} & c_{35} \end{pmatrix} \ ,$$

továbbá

$$c_{ij} = \sum_{k=1}^{4} a_{ik} \cdot b_{kj} \qquad (1 \le i \le 3, 1 \le j \le 5) \ .$$

A szisztolikus rács cellái adatokat cserélhetnek egymás között az őket összekötő ***kapcsolatokon*** keresztül, melyeket a 22.1(a) ábrán a cellákat

összekötő nyilakkal jelöltünk. A szisztolikus rács a *peremcelláin* keresztül kommunikál a *külvilággal*. A szisztolikus rács minden egyes cellája ugyanazt a kapcsolódási mintát követi a környezettel való kommunikáció során. A szisztolikus rács teljesen szabályos felépítése (a cellák elhelyezése, illetve a kapcsolatok szerkezete) szabályos adatfolyamokat eredményez a különböző kapcsolódási tengelyek mentén.

A 22.1(b) ábra *egy cella belső szerkezetét* szemlélteti. Egy cella egy ***szorzóból,*** egy ***összeadóból,*** három ***regiszterből*** és négy ***csatornából,*** illetve az ezeket összekapcsoló vezetékekből áll. Minden cellának ugyanolyan a felépítése.

Az A, B és C regiszterek mindegyike egy szám tárolására képes. A regiszterek jelölését itt értelem szerint választottuk meg, de tulajdonképpen tetszés szerint jelölhetjük őket. Az A, illetve B regiszterek az ún. ***bemeneti csatornáktól*** kapják értékeiket. A 22.1(b) ábrán a cella bal, illetve felső szélén található apró körökkel jelöltük ezeket. Egy ilyen csatorna csatlakozási felületet képez a cellához kívülről illeszkedő kapcsolathoz.

Az A, illetve B regiszterek aktuális értékét a szorzó operandusként fogja felhasználni, ezzel egyidőben az értékek továbbadódnak a cella ***kimeneti csatornáin*** keresztül (lásd az ábra jobb, illetve alsó szélén elhelyezkedő köröket). A szorzás eredményét az összeadó veszi át, mely második paraméterét a C regisztertől kapja. Az összeadás eredménye felül fogja írni C korábbi értékét.

22.1.2. A feladat és a rács paraméterei

A 22.1. ábrán bemutatott szisztolikus rács 15 cellája 3 sorból és 5 oszlopból álló téglalap alakú mintát alkot (akárcsak a C mátrix). Ennek méretei megegyeznek az A mátrix sorainak, illetve a B mátrix oszlopainak számával. Tehát esetünkben a *szisztolikus rács mérete* a megoldásra váró feladatban szereplő *adatszerkezetek méretéhez* igazodik. Általános esetben, ha egy $N_1 \times N_3$ méretű A mátrixot egy $N_3 \times N_2$ méretű B mátrixszal kellene összeszoroznunk, egy N_1 sorral, illetve N_2 oszloppal rendelkező szisztolikus rácsra lenne szükségünk.

A 22.1. ábrán látható felépítés természetesen azt is megengedi, hogy egy több, mint N_1 sorral vagy több, mint N_2 oszloppal rendelkező szisztolikus rácsot használjunk. Ez abban az esetben lényeges, amikor egy *rögzített méretű* szisztolikus rácsot szeretnénk használni különböző méretű mátrixok összeszorzására. Ekkor minden esetben csupán azt az N_1 sorból, illetve N_2 oszlopból álló téglalap alakú részt használnánk, mely a példában a rács bal felső sarkában található. Jóllehet a többi cella is ugyanúgy fog működni, ám a feladat megoldásához nem járulnak hozzá (de hibát sem okoznak).

Az N_1, N_2, N_3 méretek a megoldásra váró feladat paramétereit képezik,

mivel a megoldás során végrehajtott műveletek száma mind a három értéktől függ; ezek tehát *a feladat paraméterei*. Ellenben a szisztolikus rács mérete, illetve felépítése csak az N_1 és N_2 méretektől függ, ezek tehát egyúttal a *rács paraméterei* is lesznek.

Megjegyzés. A 22.2. alfejezetben a mátrixok szorzását megvalósító újabb szisztolikus ráccsal ismerkedhetünk meg, melynek méretei mindhárom (N_1, N_2, N_3) paramétertől függnek.

22.1.3. Térbeli koordináták

A továbbiakban a szisztolikus rács minden egyes cellájához szeretnénk egy egyértelmű ***térbeli koordinátát*** hozzárendelni, ennek segítségével jellemezve a cella mértani elhelyezkedését a rács egészéhez képest. Egy téglalap alakú szisztolikus rács esetén legegyszerűbb, ha erre a célra a megfelelő sor-, illetve oszlopszámokat használjuk. A 22.1. ábrán c_{11}-gyel jelölt cella tehát az $(1,1)$ koordinátákat kapja, a c_{21}-gyel jelölt pedig a $(2,1)$ koordinátákat, és így tovább. Az így meghatározott térbeli koordinátákat a fejezet hátralevő részében adottnak tekintjük.

Elvileg lényegtelen, hogy hol helyezkedik el a koordinátarendszer kezdőpontja, milyen irányban helyezkednek el a koordinátatengelyek, melyik irány felel meg az első, és melyik a második koordinátának. A példában a mátrix komponenseinek jelölésére a megadott számozást választottuk, ennek alapján az első koordináta a fentről lefelé, 1-gyel kezdődően számozott soroknak felel meg, míg a második komponens a balról jobbra, szintén 1-gyel kezdődően számozott oszlopoknak.

Természetesen másképp is megválaszthattuk volna a koordinátarendszert. A fenti séma azonban kitűnően megfelel az alábbi szisztolikus rácsnak: az egy adott cellában kiszámított c_{ij} mátrixkomponens indexei pontosan megegyeznek a cella koordinátáival. Az A mátrix beviteli adatként megadott sorainak száma pontosan ugyanaz, mint azon cellák első koordinátája, amelyeken ezek az adatok keresztülhaladnak; ugyanez érvényes a második koordinátára, a B mátrix oszlopaival kapcsolatban. Az összes kapcsolat (és ezzel együtt a rajtuk keresztülhaladó összes adatfolyam) tengelypárhuzamos és a koordináták növekvő irányába mutat.

Nem mindig ennyire egyértelmű, hogyan rendelhetünk megfelelő térbeli koordinátákat a cellákhoz; példaként álljon itt a 22.3(a) ábrán látható szisztolikus rács. A koordinátarendszer megválasztásától függetlenül fontos, hogy a cellák koordinátái szembeötlő módon tükrözzék a szisztolikus rács szabályos felépítését. Emiatt használunk tulajdonképpen mindig egész koordinátákat. Ezért jó, ha az egymástól minimális euklideszi távolságra fekvő cellák koordinátái csupán egyetlen komponensben különböznek, és a különbség értéke

1.

22.1.4. Generikus operátorok sorba fejtése

Minden, a 22.1. ábrán látható aktív (i, j) cella pontosan a C eredménymátrix c_{ij} elemét számolja ki. Tehát a cellának az alábbi *skalárszorzatot* kell kiértékelnie:

$$\sum_{k=1}^{4} a_{ik} \cdot b_{kj} \ .$$

Ez iteratív módon történik: minden lépésben egy újabb $a_{ik} \cdot b_{kj}$ szorzatot számítunk ki, ami hozzáadódik az addigi részösszeghez. A részösszeget a számítások elején természetesen nulláznunk kell (vagy egy tetszés szerinti kezdőértéket határozhatunk meg). Az imperatív programozási nyelvek klasszikus jelölésmódját követve a következőképpen írhatjuk le, hogy mi történik.

MÁTRIXSZORZÁS(N_1,N_2,N_3)

```
1  for i = w1 to N_1
2      for j = 1 to N_2
3          c(i,j) = 0
4          for k = 1 to N_3
5              c(i,j) = c(i,j) + a(i,k) · b(k,j)
6  return C
```

Ha $N_1 = N_2 = N_3$, akkor ennek az algoritmusnak a végrehajtása során $\Theta(N^3)$ összeadást, $\Theta(N^3)$ szorzást és $\Theta(N^3)$ értékadást kell végrehajtani, ezért egy soros processzoron a futási ideje $\Theta(N^3)$.

A $\sum$ összegző operátor az úgynevezett ***generikus operátorok*** közé tartozik, melyekhez tetszőleges számú operandus rendelhető. A 22.1. ábrán látható szisztolikus rács esetén minden egyes összeadás, mely ugyanannak az összegnek a kiszámításához tartozik, ugyanabban a cellában hajtódik végre. Ugyanakkor sok olyan példa is van, melyben egy generikus operátor egyes műveletei különböző cellák közt oszlanak meg (lásd például a 22.3. ábrán bemutatott szisztolikus rácsot).

Megjegyzés. Néhány további példa generikus operátorokra: szorzat, minimum, maximum, továbbá az ÉS, VAGY, illetve KIZÁRÓ-VAGY logikai műveletek.

Szükség van tehát a generikus operátorok ***sorba fejtésére,*** mielőtt a végrehajtandó műveleteket hozzárendelhetnénk a szisztolikus rács celláihoz. Ezeket az operátorokat azonban másképp kell kezelnünk, mint a megszokott, rögzített számú operandussal rendelkező operátorokat – ilyen például a két

operandusú összeadás – mivel ezek beosztásánál bizonyos fokú szabadsággal rendelkezünk.

22.1.5. Értékadásmentes jelölés

A szisztolikus programok leírására az imperatív forma helyett, akárcsak a MÁTRIXSZORZÁS algoritmus esetében, inkább egy ***értékadásmentes jelölést*** használunk, mely egy *egyenlet megoldásán* alapszik.

Ily módon elkerüljük a mellékhatásokat és közvetlen módon ki tudjuk fejezni a párhuzamosságot. Az alábbi esetben zavaró a $c(i,j)$ változó értékének felülírása. Ezért bevezetjük helyette a $c(i,j,k)$ példányok sorozatát, mely a $c(i,j)$ változónak a MÁTRIXSZORZÁS algoritmus lépéseinek végrehajtása során felvett értékeit jelöli. Ezáltal egy úgynevezett ***rekurzív egyenlet*** jön létre. A MÁTRIXSZORZÁS algoritmusban bemutatott mátrixszorzást például a következő módon lehet értékadásmentes alakban kifejezni:

$$
\begin{aligned}
&\textit{Bemeneti műveletek}\\
&c(i,j,0) = 0 \qquad\qquad 1 \le i \le N_1, 1 \le j \le N_2\ .\\
&\textit{Számítások}\\
&c(i,j,k) = c(i,j,k-1) + a(i,k)\cdot b(k,j) \quad 1 \le i \le N_1, 1 \le j \le N_2, 1 \le k \le N_3\ .\\
&\textit{Kimeneti műveletek}\\
&c_{ij} = c(i,j,N_3) \qquad\qquad 1 \le i \le N_1, 1 \le j \le N_2\ .
\end{aligned}
\tag{22.1}
$$

A (22.1) rendszer leírja a végrehajtott szisztolikus algoritmus pontos felépítését. A legfelső egyenlet az összes *bemeneti adatot* jellemzi, a legalsó pedig az összes *kimeneti adatot;* a szisztolikus rácsban ezek az egyenletek nem számítási, hanem ki/bemeneti műveleteknek felelnek meg. A középső egyenlet írja le a tulajdonképpeni számításokat.

A rendszer minden egyes egyenletéhez hozzátartozik egy, az egyenlet jobb oldalán látható ***kvantifikálás,*** mely az i és j iterációs változók által felvett értékek halmazát határozza meg (a középső egyenlet esetében k változóét is). Minden ilyen halmazt ***értéktartománynak*** nevezünk. A középső egyenlet i,j,k iterációs változói egy (i,j,k) ***iterációs vektort*** alkotnak. A ki/bemenet esetében az iterációs vektoroknak csupán i és j komponense van. Ahhoz, hogy egy zárt jelölésmódot kapjunk, kiegészíthetjük ezeket a vektorokat egy k komponenssel, melynek rögzített az értéke. A bemeneti adatokat $k = 0$-val jelöljük, a kimeneti adatokat pedig $k = N_3$-mal. A következőket kapjuk:

$$
\begin{aligned}
&\textit{Bemeneti műveletek}\\
&c(i,j,k) = 0 \qquad\qquad 1 \le i \le N_1, 1 \le j \le N_2, k = 0\ .\\
&\textit{Számítások}\\
&c(i,j,k) = c(i,j,k-1) + a(i,k)\cdot b(k,j) \qquad 1 \le i \le N_1, 1 \le j \le N_2, 1 \le k \le N_3\ .\\
&\textit{Kimeneti műveletek}\\
&c_{ij} = c(i,j,k) \qquad\qquad 1 \le i \le N_1, 1 \le j \le N_2, k = N_3\ .
\end{aligned}
\tag{22.2}
$$

Figyeljük meg, hogy a be/kimeneti adatokhoz tartozó értéktartományok formálisan szintén háromdimenzióssá váltak ugyan, a megszokott értelemben azonban csak kétdimenziósak.

22.1.6. Elemi számítások

A (22.2) rendszer egyenleteiből közvetlen módon leolvashatók azok az elemi, azaz tovább már nem osztható műveletek, melyeket a szisztolikus rács cellái fognak elvégezni. Ezek pontosan azok – a rendszer valamely egyenletében leírt – műveletek, melyek az egyenlethez rendelt kvantifikálás egy rögzített pontjára vonatkoznak. Amennyiben egy egyenletben több részművelet fordul elő, ezeket együvé tartozónak, azaz egyetlen ***összetett műveletnek*** tekintjük, és ugyanabban a lépésben, ugyanaz a cella fogja őket együttesen végrehajtani.

A (22.2) rendszer középső egyenletében két művelet jelenik meg: az $a(i,k)\cdot b(k,j)$ szorzás, illetve a hozzá tartozó $c(i,j,k) = c(i,j,k-1) + \cdots$ összeadás. Az együvé tartozó egyes műveleteket, vagyis az összeadást és szorzást a 22.1(b) ábrán látható szisztolikus rács cellája egy összetett „szorzás-összeadás” művelet során fogja elvégezni.

Minden egyes ilyen elemi számításhoz hozzárendelhetünk egy jelet. Nyugodtan nevezhetjük ezeket is koordinátáknak. Hogy megfelelő koordinátákat kapjunk, kézenfekvő megoldásnak kínálkozik az adott értéktartományból vett (i,j,k) iterációs vektorok használata.

A fentieket alkalmazva a (22.1) rendszerre, a $c(i,j,k) = c(i,j,k-1) + a(i,k)\cdot b(k,j)$ számításhoz például az (i,j,k) koordinátahármast rendelhetjük hozzá. Ugyanígy a $c(i,j,k) = 0$ bemeneti művelethez is ugyanazt az (i,j,k) koordinátahármast rendelhetjük hozzá; persze itt minden esetre érvényes, hogy $k = 0$. Egyébként a példában szereplő értéktartományokat úgy választottuk meg, hogy azok diszjunkt halmazok legyenek.

Mivel a számítások, illetve a ki/bemeneti műveletek jelölésére is ugyanúgy iterációs vektorokat használunk, nincs többé szükség arra, hogy ezt a két fo-

galmat megkülönböztessük. Ez egyúttal azt is jelenti, hogy az értelmezési tartomány valamely pontjához tartozó műveletek ismét egy összetett műveletet képeznek – akkor is, ha ezek különböző egyenletekből származnak, és lehet, hogy semmi közük egymáshoz. A továbbiakban az egyszerűség kedvéért koordinátaként mindig iterációs vektorokat fogunk használni.

22.1.7. Diszkrét időszeletek

A szisztolikus cellák által végzett egyes elemi számítási folyamatok mindig ***diszkrét időszeletekben*** mennek végbe. Egy szisztolikus rács esetében minden ilyen időszelet ugyanolyan hosszú. Ezen felül a szisztolikus rács minden egyes cellája teljes mértékben ***szinkron*** módon működik, azaz mindegyikük egyidőben fejezi be az adott lépéshez tartozó kommunikációs, illetve a számítási műveleteket. Az egyes cellák egyes időszeletei megszakítás nélkül követik egymást.

Megjegyzés. Természetesen Albert Einstein felismerései óta tudjuk, hogy fizikailag nem tudunk igazi egyidejűséget létrehozni. A valóságban itt arról van szó, hogy a szisztolikus rács cellái időben egymáshoz igazodva dolgoznak. Technikailag ez úgy valósítható meg, hogy a cellákat egy közös ütemjel vézérli, mely a rács összes regiszterét összeköti. Az ilyen módon elért egyidejűség keretében a cellák közti kommunikáció eléggé egyidőben megy végbe ahhoz, hogy az egymáshoz kapcsolódó írási, illetve olvasási folyamat során ne vesszenek el adatok. Ennélfogva mindenképpen az a helyénvaló, ha az elméleti fejtegetések során elvi egyidejűséget tételezünk fel.

Emiatt a fizikai időt diszkrét időszeletekre oszthatjuk, folytatólagosan megszámozva az időszeleteket. Nem lényeges, hogy az időtengely kezdőpontja hol helyezkedik el, hiszen az idő múlása minden egyes cella számára szinkron módon történik. Kézenfekvő a $t = 0$ időt úgy megválasztani, hogy ez éppen annak az időszeletnek feleljen meg, melynek során a legelső bemeneti adat elérkezik valamely cellához. Ezt az egyezményt használva, a (22.1) rendszer (i, j, k)-val jelölt összetett műveletének elvégzése az $i + j + k - 3$ időpontban történik. Másrészt ugyanúgy megfelelne az is, ha az (i, j, k) koordinátákat az $i + j + k$ időponthoz rendelnénk hozzá; a különbség az eddigiekhez képest csupán annyi, hogy ekkor az idő globálisan három egységgel lenne eltolódva.

Tételezzük tehát fel a továbbiakban, hogy egy bizonyos (i, j, k) számítás az $i + j + k$ időpontban megy végbe. A legelső számításra ekkor a $t = 3$ időpontban kerül sor, a legutolsó pedig a $t = N_1 + N_2 + N_3$ időpontban. A ***teljes végrehajtási idő*** tehát $N_1 + N_2 + N_3 - 2$ időszeletet tesz ki.

22.1.8. Külső és belső kommunikáció

Szisztolikus rácsok esetén a számításokhoz szükséges adatok a számítás kezdetén általában még nincsenek a rács celláiban. Ezeket a rácsba a ***külvilágból*** kell betölteni. A külvilág egy központi *memóriához* való hozzáféréssel rendelkező *gazdagépből* áll, melyet egy vezérlőegység vezérel. A vezérlőegység a megfelelő időpontban kiolvassa a memóriából a szükséges adatokat, megfelelő módon továbbítja azokat a szisztolikus rácsnak, illetve a kiszámolt eredményeket visszaírja a memóriába.

A k-nak megfelelő időszeletben az a_{ik} és b_{kj} operandusoknak minden egyes (i, j) cella rendelkezésére kell állniuk. De a 22.1. ábrán látható szisztolikus rács esetén mindössze az első oszlop cellái kapják az A mátrix elemeit közvetlenül a külvilágtól bemeneti adatként. Az összes többi cella arra van utalva, hogy a szükséges a_{ik} értékeket egy szomszédos cellától kapja meg. Ez, ahogy a 22.1(a) ábrán is látható, a szomszédos cellák között levő vízszintes *kapcsolatokon* keresztül történik. Az a_{ik} érték rendre keresztülhalad az $(i, 1)$, $(i, 2)$, ..., (i, N_2) cellákon. Ennek megfelelően a b_{kj} érték az $(1, j)$ cellán keresztül kerül a rácsba, onnan pedig a függőleges összeköttetéseken keresztül halad tovább a $(2, j)$, $(3, j)$, ... cellákon át, egészen az (N_1, j) celláig. Az ábrán látható nyilak hegye azt mutatja, hogy egy ilyen kapcsolat milyen *irányban* működik.

Az osztott/párhuzamos architektúrák esetén gyakorta gondot okoz, hogy egy értéket egy időegység alatt nagy távolságra továbbítsunk. Ebből következik, hogy a mi esetünkben az a_{ik} érték továbbítása az (i, j) cellától az $(i, j + 1)$ cella felé nem ugyanazon az időszeleten belül történik, melyben az (i, j) cella ezt az értéket az $(i, j - 1)$ cellától vagy a külvilágtól átvette, hanem egy időszelettel később. Ugyanez érvényes a b_{kj} érték továbbítására is. Ez a ***késleltetés*** a részletes cellaszerkezetet bemutató 22.1(b) ábrán világosan látható: minden bemeneti adattól kiinduló útvonal, mely egy cellán vezet keresztül, áthalad egy regiszteren, és minden egyes regiszter pontosan egy időszeletnyi késleltetést eredményez.

Megjegyzés. Szisztolikus architektúrák esetében általában elő van írva, hogy a különböző cellákat összekötő útvonalak mindig legalább egy regiszteren keresztül vezessenek – akkor is, ha csupán szomszédos cellák közti adatátvitelről van szó. A szisztolikus rács globális ütemjele összekapcsolja a cellák regisztereit. Mindez a szisztolikus rács kapcsolatain keresztüláramló, jellemzően ritmikus adatforgalomhoz vezet. A „szisztolé” (magyarul szívösszehúzódás) orvosi kifejezés pontosan emiatt, a lüktető vérerekkel szemben fennálló képi rokonság okán vált e fogalom névadójává.

Hogy az adatoknak ezt a késleltetett továbbadását szemléltessük, tovább bővítjük a (22.1) rendszert úgy, hogy a többszörösen olvasott értékek – mint például az a_{ik} – számára különböző példányokat vezetünk be (ez egy, a szisz-

tolikus algoritmusok tervezésére alkalmas tipikus eljárásmód):

Bemeneti műveletek

$$\begin{aligned} &a(i,j,k) = a_{ik} && 1 \le i \le N_1, j = 0, 1 \le k \le N_3 \ , \\ &b(i,j,k) = b_{kj} && i = 0, 1 \le j \le N_2, 1 \le k \le N_3 \ , \\ &c(i,j,k) = 0 && 1 \le i \le N_1, 1 \le j \le N_2, k = 0 \ . \end{aligned}$$

Számítások

$$\begin{aligned} &a(i,j,k) = a(i,j-1,k) && 1 \le i \le N_1, 1 \le j \le N_2, 1 \le k \le N_3 \ , \\ &b(i,j,k) = b(i-1,j,k) && 1 \le i \le N_1, 1 \le j \le N_2, 1 \le k \le N_3 \ , \\ &c(i,j,k) = c(i,j,k-1) && 1 \le i \le N_1, 1 \le j \le N_2, 1 \le k \le N_3 \\ &\quad + a(i,j-1,k) \cdot b(i-1,j,k) \ . \end{aligned} \tag{22.3}$$

Kimeneti műveletek

$$c_{ij} = c(i,j,k) \qquad 1 \le i \le N_1, 1 \le j \le N_2, k = N_3 \ .$$

A c_{ij} kiszámítását célzó minden egyes $c(i,j,k)$ részösszeg egy bizonyos időszelet során számítódik ki, és azt csupán egyetlen alkalommal, ti. a következő időszeletben, használjuk. Az (i,j) cellának tehát szüksége van egy regiszterre (a 22.1(b) ábrán ez a C), amely megőrzi a $c(i,j,k)$ értéket egy időszelet erejéig. A $c(i,j,k)$ értékre a továbbiakban nem lesz szükség, ezért a megfelelő regiszter tartalma felülírható a $c(i,j,k+1)$ értékkel. A skalárszorzat kiszámításának végeztével a regiszterben a $c(i,j,N_3)$ érték, azaz a kiszámításra váró c_{ij} eredmény marad meg. A számítások kezdetén a regisztert nulláznunk kell (vagy egy tetszőleges kezdőértéket adhatunk neki).

Az a_{ik} és b_{kj} értékeket ezzel szemben nem szükséges az (i,j) cellában tárolnunk. Amint a 22.1(a) ábrán vázolt adatbeviteli sémán látható, az A mátrix minden sorának bevitele egy időegységgel késleltetve van az előzőhöz képest. Ugyanúgy, a B mátrix minden egyes oszlopa egy időegységgel késleltetve van az előzőhöz képest. Így az $a(i,j-1,k)$ és $b(i-1,j,k)$ értékek pontosan abban az időszeletben érkeznek az (i,j) cellához, amikor ott a $c(i,j,k)$ érték kiszámítása történik, értékük az A, illetve B regiszterbe íródik, innen viszont azonnal felhasználjuk őket a szorzás elvégzésére, illetve értékük továbbadódik a szomszédos celláknak. Amint az (i,j) cellában megtörtént az a_{ik} és b_{kj} értékek összeszorzása, ebben a cellában már nincs szükség az értékükre. Emiatt nem fontos az értéküket a cellában tovább őrizni, így az A és B a következő időszelet során új értéket fog kapni.

Ezekből a fejtegetésekből máris kiderül, hogy ahhoz, hogy egy cella tárolókapacitását a minimálisra csökkentsük, ügyelnünk kell arra, hogy a számítási, illetve kommunikációs folyamatokat úgy irányítsuk térben és időben, hogy az azonnali felhasználás, illetve továbbadás révén az adatokat a

lehető legrövidebb ideig kelljen csupán tárolni. A szisztolikus rács általános felépítésén túl ezt lényegében azzal érhetjük el, hogy egy megfelelő ki/beviteli sémát választunk, illetve megfelelő módon állapítjuk meg a cellákon belüli késleltetési időket.

A példában látható ***ki/beviteli séma*** geometriai elrendezése az A és B mátrixok *szétdarabolásának* eredményeként született. A bemeneti adatfolyamokban ezáltal szabaddá vált helyeket null-értékekkel töltjük fel, különben elrontanánk a c_{ij} értékek kiszámítását. A bemeneti adatfolyamok hossza a feladat N_3 paraméterétől függ.

A C mátrix elemeinek kiszámítása a 22.1. ábra alapján ***stacionárius módon*** történik, ami azt jelenti, hogy valamely mátrixelem végleges értéke kiszámításának lépései ugyanabban a cellában mennek végbe. A ***stacionárius változók*** egyáltalán nem mozdulnak a számítások során a szisztolikus rácson belül. Az eredményeket végül továbbítanunk kell a rács peremén levő cellákhoz, mivel csak ezeken keresztül tudjuk átadni őket a külvilágnak. Ráadásul figyelembe kell vennünk azt is, hogy a c_{ij} kiszámítására szolgáló regisztereket kezdőértékkel kell ellátnunk. E két különleges kiegészítő feladat megoldása elég nagy időbeli, illetve anyagi ráfordítást igényel, de ezt a 22.4. alfejezetben még pontosabban megvizsgáljuk majd.

22.1.9. Futószalag-elvű feldolgozás

A jellemző módon diszkrét, egyforma hosszúságú, globálisan szinkronizált időszeletekkel való munkamódnak, illetve a cellák egymástól regiszterek segítségével történő szigorú időbeli elszigeteltségének köszönhetően, a szisztolikus rácsokat a *futószalag-elvű feldolgozás* egy sajátos alkalmazásának tekinthetjük. A cellák regiszterei ez esetben a jól ismert ***futószalag-regiszteréknek*** felelnek meg. A klasszikus értelemben vett futószalag kétségkívül mindig lineáris felépítésű, míg a szisztolikus rácsok szerkezete (amint az a példából is látszik) gyakorta kétdimenziós. Egy ***többdimenziós szisztolikus rácsot*** felfoghatunk úgy, mint ami több, egymásba fonódó futószalag szövedékéből áll. Természetesen egyértelmű, hogy az egydimenziós futószalagelvű feldolgozásra érvényes alapvető sajátosságok a többdimenziós szisztolikus rácsoknál ugyanúgy fellelhetőek.

A futószalagelvű feldolgozás egyik tipikus velejárója, hogy a ***hatékonyság*** kisebb a számítások kezdetén, illetve végén. Eleinte a futószalag üres, egyetlen fokozata sem dolgozik. Ezt követően csupán az első fokozat rendelkezik adatokkal és ez végez számításokat; az összes többi fokozatnak továbbra sincs semmi tennivalója. A következő időszeletben az első fokozat adatokat ad át a következőnek, ugyanakkor maga is újabb adatokat kap; csak ez a két fokozat dolgozik. Ez így megy tovább mindaddig, amíg végül minden fokozat

megtelik adatokkal, és a futószalag dolgozza fel ezeket, vagyis első ízben állíthatjuk a futószalagról, hogy teljes hatékonysággal dolgozik. Néhány ilyen teljes telítettségű lépés után, melynek időtartama az adatfolyamok méretétől függ, egyszer csak elfogy a bemeneti adat, a futószalag első fokozata tehát abbahagyja a működést. A következő időszeletben a második fokozat is abbahagyja a munkát, majd ugyanígy tovább, míg legvégül egyetlen fokozat sem dolgozik már. A kezdő, illetve végső munkaszakasz csökkenti a teljes futószalag átlagteljesítményét, és ennek a teljesítménycsökkenésnek a viszonylagos értéke annál nagyobb, minél több fokozata van a futószalagnak az adatfolyamok hosszához képest.

Ugyanezt a jelenséget a maga sajátos mivoltában kitűnően vizsgálhatjuk a 22.1. ábrán bemutatott szisztolikus rács esetében. Itt is találunk a számítások kezdetén, illetve végén szinte tétlenül álló cellákat. Az első időszeletben csupán az $(1,1)$ cella végez hasznos munkát; az összes többi cella is dolgozik ugyan, de csak üresjáratban. A második lépésben ehhez hozzáadódnak még az $(1,2)$ és $(2,1)$ cellák; ezt az időszeletet szemlélteti a 22.2(a) ábra. Mindez ugyanígy folytatódik, míg végül az (N_1, N_2) cella is hozzá nem járul a számításokhoz. Amint a legutolsó tényleges adat elhagyta az $(1,1)$ cellát, ez többé nem járul hozzá a számításokhoz, csupán a már kiszámolt c_{11} értéket fogja újra és újra reprodukálni. Lépésről lépésre egyre több cella hagyja abba az aktív tevékenységet, míg legvégül már csak az (N_1, N_2) cella végzi el az utolsó fontos számítást; a 22.2(b) ábra szemlélteti ezt a végső időszeletet.

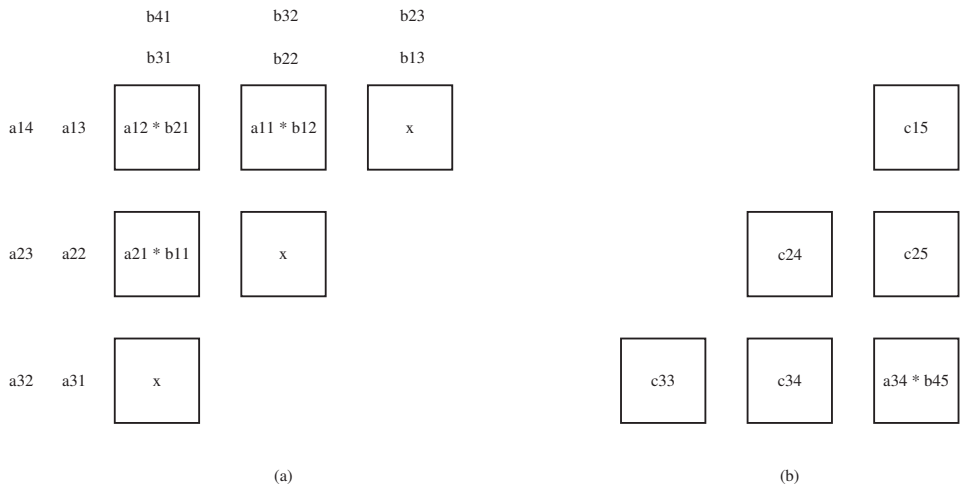
Megjegyezzük, hogy a képletekben a szorzás jelölésére pontot használunk, az ábrákon azonban csillagot.

Gyakorlatok

22.1-1. Hogyan kellene módosítani a 22.1(a) ábrán bemutatott beviteli adatsémát, ha ugyanazon szisztolikus rács segítségével egy (2×6)-os és egy (6×3)-as mátrixot szeretnénk összeszorozni? Átszervezhetők-e a számítások oly módon, hogy az eredménymátrix a szisztolikus rács jobb alsó sarkába kerüljön?

22.1-2. Miért fontos a 22.1. ábra szerinti mátrixszorzás szempontjából, hogy a beviteli folyamokban a nem használt helyekre 0 értékeket helyezzünk? Miért nem lényeges ugyanez a B mátrix esetében?

22.1-3. Ha a 22.1. ábrán látható szisztolikus rácsot futószalagként kellene értelmezzük, hány fokozatra lenne szükségünk ahhoz, hogy megfelelőképpen tudjuk leírni ennek viselkedését?



22.2. ábra. Két kiragadott helyzetkép a 22.1. ábrához (részletek).

22.2. Tér-idő-leképezés és szisztolikus rács

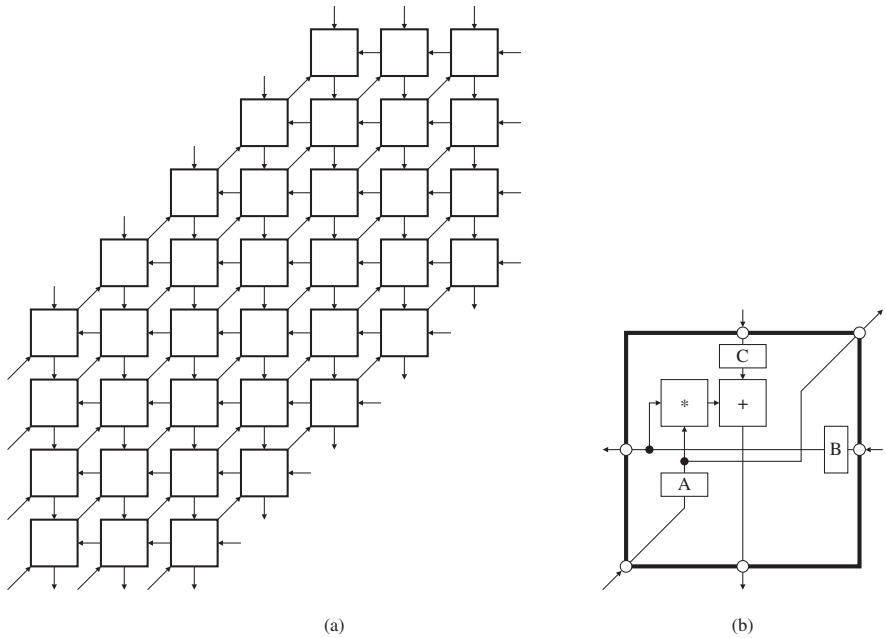
Az előző alfejezetben leírt bevezető elegendő ugyan egy egyszerű fogalom megalkotásához, de akkor már nem elég, ha a szisztolikus rács tulajdonságait mennyiségileg pontosan szeretnénk megragadni és értékelni. Különösen a *paraméteres feladat-meghatározás* bevezetése igényel matematikai segédeszközöket. Ezért ez az alfejezet az *egyenletes* szisztolikus algoritmusok *lineáris leképezésekre* alapozó elméletének központi kérdéseivel foglalkozik.

22.2.1. Példa: mátrixszorzás stacionárius változók nélkül

A (22.3) rendszer nem csak a 22.1. ábrán bemutatott szisztolikus rács segítségével számolható ki, hanem sok más szisztolikus ráccsal is. A 22.3. ábra példa egy ilyen szisztolikus rácsra. Bár ugyanazt a függvényt értékeli ki, mint a 22.1. ábrán látható rendszer, a képi megjelenése teljesen más:

- a cellák száma itt lényegesen nagyobb, 15 helyett összesen 36;
- a rács körvonala hexagonális, téglalap alakú helyett;
- minden cellának három bemenete és három kimenete van;
- az adatok bevitele lényegesen másképp történik, mint a 22.1(a). ábrán;
- végül: itt a C mátrix elemei is végighaladnak a rácson.

A 22.3(b) ábrán látható cellaszerkezet első látásra nem tűnik lényegesen különbözőnek a 22.1(b) ábrához képest. A különbségek azért mégis jelentősek:



22.3. ábra. Hexagonális szisztolikus rács mátrixszorzáshoz: **(a)** a rács felépítése és az adatok bevitelének/kivitelének elve; **(b)** cellaszerkezet.

az új cellában nincs *ciklikus út*, így *stacionárius változók* itt nem jelennek meg. Ehelyett a cellának három bemenő, illetve három kimenő csatornája van, melyeken keresztül a három mátrix elemei haladnak. A csatornákon keresztül történő kommunikáció iránya a cella jobb és bal oldalán megváltozott, a mátrixok csatornákhoz való hozzárendelése úgyszintén.

22.2.2. A tér-idő leképezés, mint globális szemléletmód

Hogyan függ tehát össze a (22.3) rendszer és a 22.3. ábra? A 22.1. alfejezetben bemutatott szisztolikus rács működését minden segítség nélkül jól tudtuk követni. Az alábbi példa esetén ez azonban lényegesen nehezebb – így sokkal inkább indíttatást érzünk egy matematikai segédeszköz használatára.

Ahhoz, hogy le tudjuk írni a szisztolikus rácson belül végzett műveleteket, minden ilyen művelethez két alapvető mértéket rendelhetünk hozzá: az időszeletet, melyben a művelet végrehajtásra kerül, illetve a cellát, amely a műveletet végrehajtja. Amint ez a későbbiekben még inkább nyilvánvalóvá lesz, a ***tér-idő-leképezés*** megválasztásával tulajdonképpen ki is merítettük a tervezéssel kapcsolatos szabadságunkat; szinte minden további elem kényszerű

módon következik a tér-idő-leképezésből.

Akárcsak a 22.1. ábra szisztolikus rácsa esetében, a 22.3. ábrán látható szisztolikus rácsban is a $t = i + j + k$ időpontban történik az (i, j, k) elemhez kapcsolódó számítások elvégzése. Ugyanezt egy

$$\pi = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix} \tag{22.4}$$

idővektor és egy $v = (i, j, k)$ iterációs vektor skalárszorzataként is leírhatjuk,

$$t = \pi \cdot v \ , \tag{22.5}$$

esetünkben tehát

$$t = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} i \\ j \\ k \end{pmatrix} = i + j + k \ . \tag{22.6}$$

A 22.1. ábrán látható példában végrehajtott műveletek $z = (x, y)$ térkoordinátáit, a 22.1.3. pontban leírt egyezményünk alapján a $v = (i, j, k)$ iterációs vektorokból következtethetjük ki. Az itt megválasztott leképezés az $\mathbb{R}^3$ tér k tengely menti *projekcióját* végzi el. Ez a lineáris leképezés egy

$$P = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \tag{22.7}$$

projekciós mátrix segítségével írható le. A térkoordinátákat úgy kapjuk, hogy a projekciós mátrixot megszorozzuk a v iterációs vektorral, amit az alábbi módon írhatunk le:

$$z = P \cdot v \ . \tag{22.8}$$

A ***projekció irányát*** az a vektor adja, amely ortogonális a projekciós mátrix minden sorára nézve:

$$P \cdot u = \vec{0}. \tag{22.9}$$

A (22.7)-ben szereplő P projekciós mátrix számára például $u = (0, 0, 1)$ egy lehetséges ***projekciós vektor.***

Szisztolikus rácsok tervezésénél gyakran alkalmaznak projekciókat a térkoordináták meghatározására. A 22.3(a) ábrán látható példánkban is az iterációs vektorokra alkalmazott projekció útján kapjuk a térkoordinátákat. Tekintsük adottnak az alábbi projekciós mátrixot:

$$P = \begin{pmatrix} 0 & -1 & 1 \\ -1 & 1 & 0 \end{pmatrix} \ . \tag{22.10}$$

Egy hozzá tartozó lehetséges projekciós vektor $u = (1, 1, 1)$.

A projekciós mátrixot és az idővektort összegezhetjük egy T mátrix segítségével, mely magát az úgynevezett *tér-idő-leképezést* írja le:

$$\begin{pmatrix} z \\ t \end{pmatrix} = \begin{pmatrix} P \\ \pi \end{pmatrix} \cdot v = T \cdot v \; . \tag{22.11}$$

T első két sorát a P projekciós mátrix adja, a harmadikat pedig a π idővektor.

A 22.1. ábrán szereplő példa esetén a tér-idő-leképezés T mátrixa a következő:

$$T = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix} \; , \tag{22.12}$$

a 22.3. ábrán szereplő példa esetén pedig

$$T = \begin{pmatrix} 0 & -1 & 1 \\ -1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix} \; . \tag{22.13}$$

A tér-idő-leképezést a szisztolikus rácsra vonatkozó globális szemléletmódként is felfoghatjuk. Amennyiben egy (esetünkben lineáris, T-vel jelölt) tér-idő-leképezést alkalmazunk egy rekurzív egyenletrendszerre, máris láthatóvá válnak a szisztolikus rács külső ismérvei, azaz a felépítése (térkoordináták, kapcsolatrendszer, cellaszerkezet).

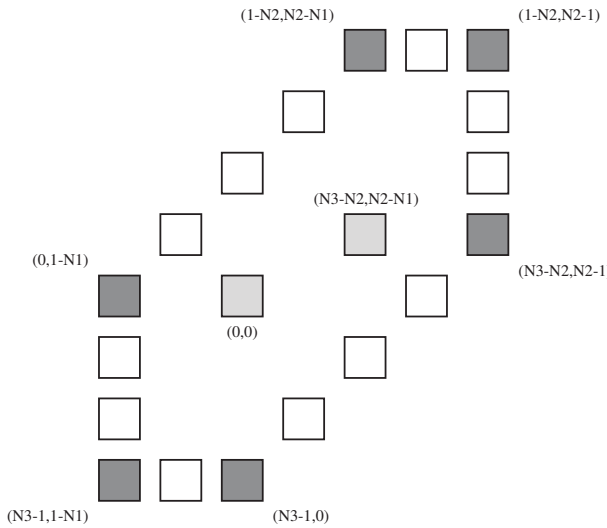
Megjegyzés. Tisztán lineáris leképezések helyett affin leképezések is szóba jöhetnek, melyeknek egy vektor-komponensük is van, például $T \cdot v + h$. Affin leképezésekre viszont nincs feltétlenül szükségünk, amíg minden iterációs vektort ugyanazzal a tér-idő-leképezéssel kezelünk.

22.2.3. A térkoordináták szimbolikus meghatározása

Amikor az értéktartományok számszerűen adottak, és eléggé kicsik, a (22.11) összefüggés segítségével könnyűszerrel kiszámíthatjuk a térkoordináták konkrét halmazát. Amennyiben az értéktartományok paraméteresek, mint a (22.3) rendszer esetében, a cellák helyzetét *szimbolikusan* kell meghatároznunk. Az alábbiakban leírtak e feladat megoldását célozzák.

Minden egyes cellát egy $z = (x, y)$ térkoordinátájú pontnak tekintünk az $\mathbb{R}^2$ kétdimenziós térben. Az S értéktartomány minden egyes iterációs vektorából a (22.8) kifejezés segítségével egy-egy processzor (cella) z térkoordinátáit kapjuk, $z = P \cdot v$, így a v-vel jelölt művelet a z cellára lesz rávetítve. Az így kapott térkoordináták halmaza $P(S) = \{P \cdot v : v \in S\}$ adja meg a szisztolikus rács működése szempontjából fontos összes cella helyzetét.

Általában olyan értéktartományok fordulnak elő, melyek egy konvex



22.4. ábra. Egy téglalap alakú értéktartomány projekciójának eredménye.

terület (itt $\mathbb{R}^3$-ból) összes egész koordinátájú pontjának halmazaként jeleníthetők meg ***(sűrű konvex értéktartományok).*** Egy ilyen (véges számosságú) értéktartomány konvex burka egy *politóp*, melynek csúcsai az értéktartomány pontjai. A politópokat tetszőleges lineáris leképzés újabb politóppá alakítja. Kihasználhatjuk tehát, hogy minden projekció lineáris leképezés. Az újonnan keletkező politóp csúcsai az eredeti politóp csúcsainak projekciói.

Megjegyzés. Nem szükséges, hogy a projekció során az eredeti politóp minden csúcsa az új politópnak is csúcsa legyen. Lásd például a 22.4. ábrát.

A $\mathbb{Z}^3$ rács projekciója egy egész számú P projekciós mátrix által a $\mathbb{Z}^2$ rácshoz vezet, amennyiben a P-t egy π egész számú idővektor megválasztásával egy *unimoduláris T mátrixra* egészítjük ki. Ez egy sűrű konvex értéktartományt (néhány, az alkalmazás szempontjából lényegtelen kivételtől eltekintve) a koordináták sűrű konvex halmazává alakítja, melyet az ezt burkoló politóp csúcsai teljes mértékben jellemeznek.

Megjegyzés. Egy mátrixot ***unimodulárisnak*** nevezünk, amennyiben négyzetes, csak egész számú bemenetei vannak és a determinánsa ± 1. Az unimoduláris mátrixok invertálhatók, és inverzük szintén unimoduláris.

Alkalmazzuk ezt a módszert a (22.3) rendszer

$$S = [1, N_1] \times [1, N_2] \times [1, N_3] \tag{22.14}$$

egész számokat tartalmazó értéktartományára. A konvex burok csúcsai esetünkben

$$\begin{aligned}&(1,1,1),(N_1,1,1),(1,N_2,1),(1,1,N_3)\ ,\\&(1,N_2,N_3),(N_1,1,N_3),(N_1,N_2,1),(N_1,N_2,N_3)\ .\end{aligned}\tag{22.15}$$

A (22.10)-beli P projekciós mátrix esetén a projekció csúcsainak helyzete

$$\begin{aligned}&(N_3-1,0),(N_3-1,1-N_1),(0,1-N_1)\ ,\\&(1-N_2,N_2-N_1),(1-N_2,N_2-1),(N_3-N_2,N_2-N_1)\ .\end{aligned}\tag{22.16}$$

Mivel S-nek nyolc csúcsa van, a $P(S)$ képének viszont csupán hat, világos, hogy az S két csúcsa a terület belső részébe fog vetítődni, tehát a mértékek meghatározásában nem játszik szerepet; ezek az $(1,1,1)$ és az (N_1,N_2,N_3) csúcsok.

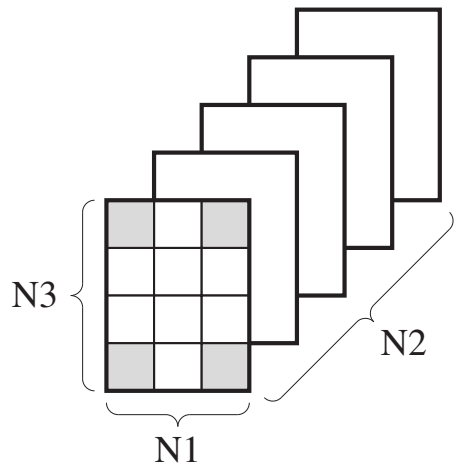
Konkrétan $N_1=3$, $N_2=5$ és $N_3=4$ esetében a $(3,0)$, $(3,-2)$, $(0,-2)$, $(-4,2)$, $(-4,4)$ és $(-1,4)$ csúcsok adódnak. Láthatjuk, hogy nem kell minden térkoordinátának feltétlenül pozitívnak lennie. A koordinátarendszer kezdőpontjának megválasztása, mely esetünkben a politóp belsejében helyezkedik el, szintén nem nyilvánvaló.

A projekció eredményeként egy hatszöget kapunk, melynek szembefekvő oldalai párhuzamosak. Ennek peremén mindig N_1, N_2 vagy N_3 egész számú pont helyezkedik el. Az első példával ellentétben itt tehát a feladat minden paramétere egyúttal a rács paramétere is. Ennek a szisztolikus rácsnak a körvonalát ***hexagonálisnak*** nevezzük.

Ennek a tartománynak az F felülete $\Theta(N_1N_2+N_1N_3+N_2N_3)$, mely egyformán függ mindhárom mátrixmérettől. Itt tehát egészen más helyzettel találkozunk, mint a 22.1(a) ábrán, ahol (ugyanennek a feladatnak!) a bonyolultsága csupán $\Theta(N_1N_2)$ volt.

Ezután a hozzávetőleges számolás után most megszámoljuk egész pontosan a cellákat. Ehhez a számoláshoz célszerű az egész tartományt résztartományokra bontani, melyekben a cellák száma könnyűszerrel meghatározható (lásd 22.5. ábra). A $(0,0)$, $(N_3-1,0)$, $(N_3-1,1-N_1)$ és $(0,1-N_1)$ pontok egy N_1N_3 cellát tartalmazó téglalapot határolnak körül. Ha eltoljuk ezt a ponthalmazt N_2-1 cellával feljebb és N_2-1 cellával jobbra, végigjárjuk ezzel a teljes tartományt. Minden alkalommal azonban, amikor a téglalapot egy cellával feljebb és jobbra toljuk, csupán N_1+N_3-1 új cella jön hozzá az eddigiekhez. Ez összesen $N_1N_3+(N_2-1)(N_1+N_3-1)=N_1N_2+N_1N_3+N_2N_3-(N_1+N_2+N_3)+1$ cellát tesz ki.

Például $N_1=3$, $N_2=5$ és $N_3=4$ esetében 36 cellát kapunk, amint az a 22.3(a) ábrán látszik is.



22.5. ábra. A térkoordináták halmazának részekre bontása.

22.2.4. A teljes végrehajtási idő szimbolikus kiszámítása

Egy szisztolikus algoritmus ***teljes végrehajtási idejének*** szimbolikus kiszámítása a 22.2.3. pontban leírtakhoz hasonló módon történik. Az időleképezés a (22.5) összefüggés alapján szintén lineáris leképezés. Az első, illetve utolsó számításnak megfelelő időszeletet a számítási időpontok $\pi(S) = \{\pi \cdot v : v \in S\}$ halmazának minimuma, illetve maximuma adja. A fentebb leírtak alapján elegendő, ha az S konvex burkának v csúcsait vesszük figyelembe.

A *teljes végrehajtási időt* a következő képlet alapján számíthatjuk ki:

$$t_\Sigma = 1 + \max P(S) - \min P(S) \; . \tag{22.17}$$

Az 1 hozzáadása mindenképpen fontos, mert a legelső és a legutolsó számítás időpontja is számít.

A 22.3. ábrán látható példa esetében a (22.6) összefüggést alkalmazva a (22.15)-ben kiszámolt politópcsúcsokra, a következő képek halmazát kapjuk: $\{3, 2+N_1, 2+N_2, 2+N_3, 1+N_1+N_2, 1+N_1+N_3, 1+N_2+N_3, N_1+N_2+N_3\}$. Az alapfeltevésből, mely szerint $N_1, N_2, N_3 \geq 1$ következik, hogy a minimum 3, a maximum pedig $N_1+N_2+N_3$, a teljes végrehajtási idő pedig $N_1+N_2+N_3-2$ időegységet tesz ki (akárcsak a 22.1. ábrán látható szisztolikus rács esetén – elvégre az értéktartomány és az idővektor megegyezik a két példában).

A feladat paramétereinek az $N_1 = 3$, $N_2 = 5$ és $N_3 = 4$ konkrét értékeket adva a kiszámolt teljes végrehajtási idő $12 - 3 + 1 = 10$ időszeletet tesz ki.

Ha $N_1 = N_2 = N_3$, akkor a szisztolikus algoritmus egy $\Theta(N^2)$ cellát tartalmazó rendszerrel $\Theta(N)$ idő alatt határozza meg két, $N \times N$ méretű mátrix szorzatát.

22.2.5. A kapcsolatszerkezet levezetése

A szisztolikus rács ***kapcsolatszerkezetét*** úgy kapjuk, hogy a tér-idő-leképezést alkalmazzuk a feladat *adatfüggőségeire*. Minden egyes adatfüggőség abból adódik, hogy egy változó bizonyos példányát közvetlen módon felhasználjuk ugyanazon vagy egy másik változó egy példányának kiszámolására.

Megjegyzés. A szisztolikus rácsoknál – az imperatív programnyelven megírt programok párhuzamos párhuzamos végrehajtásánál szükséges adatfüggőség vizsgálatok helyett, melyeket vagy a párhuzamosan optimalizáló fordító és/vagy a processzor végez el – csupán folyamfüggőségekről van szó. Ez az általunk használt hozzárendelésmentes jelölés következménye.

Az adatfüggőségeket úgy tudjuk leolvasni, hogy az értékadásmentes jelölés kvantifikált egyenletének egyszerre szemléljük a jobb és bal oldalát. Mindenekelőtt a (22.3) rendszer $c(i,j,k) = c(i,j,k-1) + a(i,j-1,k) * b(i-1,j,k)$ egyenletét vizsgáljuk.

A $c(i,j,k)$ érték kiszámítása a $c(i,j,k-1)$, $a(i,j-1,k)$ és $b(i-1,j,k)$ értékek segítségével történik. Van tehát egy ***adatfolyamunk*** $c(i,j,k-1)$-től $c(i,j,k)$ irányában, egy adatfolyam $a(i,j-1,k)$-tól $c(i,j,k)$ irányában és egy adatfolyam $b(i-1,j,k)$-tól $c(i,j,k)$ felé.

Egy ilyen adatfolyam számunkra fontos sajátosságait egy ***függőségi vektor*** segítségével írhatjuk le. Ezt a kiszámolt változópéldány iterációs vektora, illetve az ennek kiszámításához éppen használt változópéldány iterációs vektora közti különbségvektor adja.

A $c(i,j,k)$ iterációs vektora (i,j,k), a $c(i,j,k-1)$-é pedig $(i,j,k-1)$. Az ebből kapott különbségvektor pedig

$$d_C = \begin{pmatrix} i \\ j \\ k \end{pmatrix} - \begin{pmatrix} i \\ j \\ k-1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} . \tag{22.18}$$

Ugyanúgy, megfelelő módon kapjuk:

$$d_A = \begin{pmatrix} i \\ j \\ k \end{pmatrix} - \begin{pmatrix} i \\ j-1 \\ k \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \tag{22.19}$$

és

$$d_B = \begin{pmatrix} i \\ j \\ k \end{pmatrix} - \begin{pmatrix} i-1 \\ j \\ k \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} . \tag{22.20}$$

A (22.3) rendszer $a(i,j,k) = a(i,j-1,k)$ egyenletében közvetlenül felismerhető, hogy melyik a kiszámolt változópéldány, és melyik a kiszámításához szükséges változópéldány. Itt láthatjuk tehát a különbséget egyenlet és

értékadás között. Tekintve, hogy $a(i,j,k)$-t $a(i,j-1,k)$-ból másolás révén kapjuk, ugyanazt a függőségi vektort kapjuk, mint a (22.19) kifejezésben. Ugyanez érvényes a $b(i,j,k) = b(i-1,j,k)$ egyenletre is.

Amennyiben egy változópéldány a v iterációs vektorral rendelkezik, azt a $P \cdot v$ cellában számítjuk ki. Ha ennek kiszámításához egy másik, v' iterációs vektorral rendelkező változópéldányra is szükség van, akkor ez a $P \cdot v'$ cellában számítódott ki és $d = v - v'$ függőségi vektor jellemzi a köztük fennálló adatfüggőséget. Ez a $z' = P \cdot v'$ cellától a $z = P \cdot v$ cella felé történő kommunikációt feltételez. Szisztolikus rácsok esetében ezt a kommunikációt a kommunikáló cellák közti közvetlen, statikus kapcsolat biztosításával oldják meg. A (22.8) leképzés lineáris voltából következik, hogy $z - z' = P \cdot v - P \cdot v' = P \cdot (v - v') = P \cdot d$.

Amennyiben $P \cdot d = \vec{0}$, a kommunikáció a számítást végző cellán belül történik, ami azt jelenti, hogy az csak időben, de nem térben zajlik. Az értékek továbbadása időben a számítást végző cella regiszterein keresztül valósul meg.

Ha viszont $P \cdot d \neq \vec{0}$, akkor két különböző cella közötti kommunikációra lesz szükség. Ekkor a szisztolikus rács összes cellájához hozzá kell rendelni egy $P \cdot d$ irányú kapcsolatot. Meghúzzuk a $P \cdot d$ vektort a számítást végző cella z térkoordinátájától a vektor irányításával ellentétesen haladva, és a z-t ellátó, z' térkoordinátájú cellához jutunk.

Ha több ilyen d függőségi vektorunk van, mindegyikhez tartozik egy-egy neki megfelelő kapcsolat. Tekintsük például a (22.18), (22.19), (22.20) és (22.10) összefüggéseket. A következők állnak fenn: $P \cdot d_A = (-1,1)$, $P \cdot d_B = (0,-1)$ és $P \cdot d_C = (1,0)$. A három $P \cdot d$ vektornak megfelelő kapcsolatokat a 22.3(a) ábrán minden egyes cellánál megfigyelhetjük. Ez egy hexagonális kapcsolatszerkezetet eredményez (az eddig ismert ortogonális helyett).

22.2.6. A cellaszerkezet meghatározása

Most átvisszük a 22.2.5. pontban tárgyalt, térrel kapcsolatos fejtegetéseket az időbeli összefüggésekre. Egy v iterációs vektorral rendelkező változópéldány kiszámítására a $\pi \cdot v$ időszeletben kerül sor. Amennyiben ennek kiszámításához egy másik, v' iterációs vektorral rendelkező változópéldány szükséges, akkor ez utóbbi értéke a $\pi \cdot v'$ időszeletben került kiszámításra. A $d = v - v'$ függőségi vektornak megfelelő kommunikáció tehát pontosan $\pi \cdot v - \pi \cdot v'$ időszeletet vesz igénybe.

Mivel a (22.6) leképezés lineáris, fennáll $\pi \cdot v - \pi \cdot v' = \pi \cdot (v - v') = \pi \cdot d$. Mivel a szisztolikus alapelv szerint minden egyes kommunikáció legalább egy regiszteren vezet keresztül, a d függőségi vektorok pedig rögzítettek, az

idővektor megválasztását a

$$\pi \cdot d \geq 1 \tag{22.21}$$

feltétel korlátozza.

$P \cdot d = \vec{0}$ esetén az éppen szükséges érték tárolása céljából egy ***regiszterre*** van szükség minden egyes cellában. Mivel egy ilyen regiszter tartalma a következő időszeletben máris felülíródik egy újabb értékkel, a régi értéket egy további regiszterbe kell átmentenünk, amennyiben $\pi \cdot d \geq 2$ fennáll. Mivel mindez $\pi \cdot d$ időszeleten belül megismétlődik minden egyes tárolt érték esetén, a cellának pontosan $\pi \cdot d$ regiszterre van szüksége, melyeken az értékek rendre keresztülhaladnak, mielőtt értékük a következő cellának adódik át. Az előbb vázolt helyzetnek megfelelően, $P \cdot d \neq \vec{0}$ esetében az adatátvitel ugyancsak $\pi \cdot d$ regiszteren keresztül történik, itt azonban nem fontos, hogy ezek mind a számítást végző cellában legyenek elhelyezve.

Minden egyes d függőségi vektor esetén szükség van megfelelő regiszterekre. A 22.3(b) ábrán a cellához rendelt három bemenetet láthatjuk, melyek a d_A, d_B és d_C függőségi vektoroknak felelnek meg. Mind a három vektor esetében fennáll, hogy a $P \cdot d \neq \vec{0}$, illetve (22.6) és (22.4) összefüggések következtében $\pi \cdot d = 1$. Tehát minden egyes d függőségi vektorhoz egyetlen regiszterre van szükség. A (22.3) rendszer szabályos volta miatt a cella három bemenetéhez ugyanakkor három kimenet tartozik, a cella középpontjához képest egymással ellentétes pozícióban.

Mivel a d függőségi vektorok száma egy (22.3)-hoz hasonló rendszer által statikusan korlátozott, és a nekik megfelelő $\pi \cdot d$ érték rögzített és többnyire kicsi, egy cellának általában kevés regiszterre van szüksége.

A cella három be-, illetve kimenete három dinamikus mátrix használatát teszi lehetővé. A 22.1. ábrával ellentétben egy $\sum_{k=1}^{4} a_{ik} \cdot b_{kj}$ skalárszorzat kiszámítása itt nem egyetlen cellában történik, hanem a szisztolikus rács cellái közt felosztva. Itt tehát feltétlenül szükséges, hogy az összeget részösszegek sorozatára bontsuk. Ez tehát példa egy szétosztott generikus operátorra.

A három bemeneten, a hozzátartozó regisztereken és a három kimeneten kívül a 22.3(b) ábrán egy szorzót láthatunk egy sorosan hozzákapcsolt összeadóval. A (22.8) leképezés alkalmazása a (22.3) rendszer $c(i,j,k) = c(i,j,k-1) + a(i,j-1,k) * b(i-1,j,k)$ egyenletének S értéktartományára a fent említett két egység összes cellában való jelenlétét eredményezi. Mivel az egyenlet alapján az összeg felépítése csakis a szorzat sikeres végrehajtása után lehetséges, ebből következik a két operátor 22.3(b) ábrán látható sorrendje.

Hogy a megfelelő operandusok honnan származnak, az a hozzájuk tartozó függőségi vektorok projekciójából olvasható le. Így például $a(i,j-1,k)$-hoz a $d_A = (0,1,0)$ függőségi vektor tartozik. Ennek projekciója, $P \cdot d_A = (-1,1)$, az A mátrix haladási irányát mutatja. A beolvasandó adatot ezért,

a számítást végző processzor szemszögéből nézve az ezzel ellentétes $(1, -1)$ irányból kell várni, vagyis a cella bal alsó sarkához kapcsolódó csatornából (de az A regiszteren keresztül). Ugyanígy $b(i-1, j, k)$ jobb oldalról jön (a B regiszteren keresztül), $c(i, j, k-1)$ pedig fentről (a C regiszteren keresztül). A megfelelő $a(i, j, k)$, $b(i, j, k)$ és $c(i, j, k)$ értékek a szemközti irányba csatornákon keresztül továbbítódnak: jobbra, felfelé, bal alsó irányba.

Másrészt, a (22.7)-beli P projekciós mátrixot alkalmazva a d_C-re a $(0, 0)$ projekciót kapjuk. Mivel ugyanakkor $\pi \cdot d_C = 1$, következik, hogy pontosan egy C regiszterre van szükség a C mátrix minden egyes eleme számára. Ez a regiszter szolgáltatja a $c(i, j, k)$ érték kiszámítására a $c(i, j, k-1)$ értéket, majd a számítást követően felveszi a $c(i, j, k)$ értékét. Mindez a 22.1(b) ábrán jól követhető. A 22.1(a) ábra ennek megfelelően azt mutatja, hogy a C mátrix számára nincs szükség cellák közti kapcsolatra: a mátrix *stacionárius*.

Gyakorlatok

22.2-1. Egy u projekciós irányhoz mindig több különböző P projekciós mátrixot találunk.

a. Mutassuk meg, hogy a

$$P = \begin{pmatrix} 0 & 1 & -1 \\ -1 & 0 & 1 \end{pmatrix}$$

projekciós mátrix is megfelel az $u = (1, 1, 1)$ projekciós iránynak.

b. Számítsuk ki ennek a projekciós mátrixnak a segítségével a (22.3) rendszer értéktartományát.

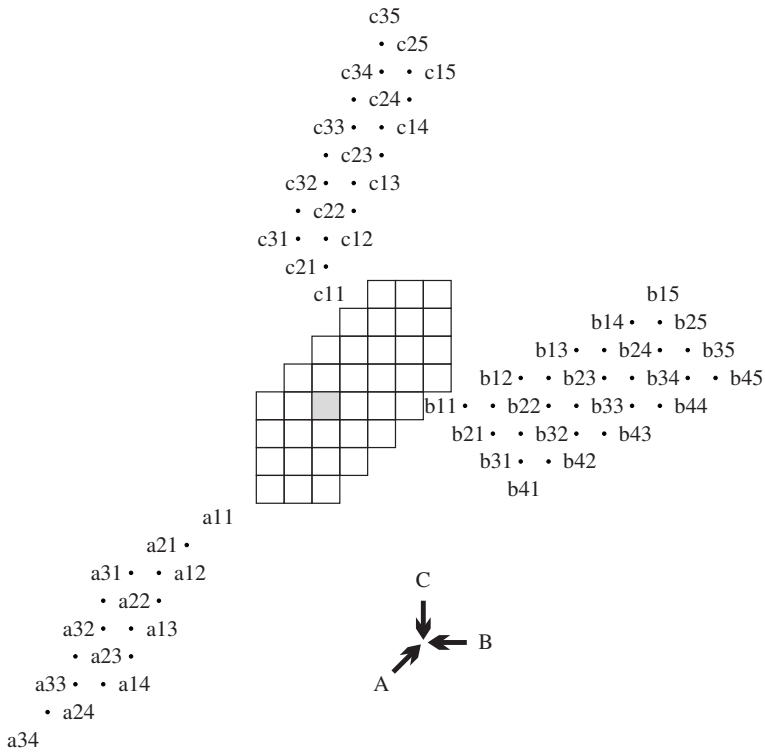
c. Az így kapott térkoordináták különböznek a (22.2.3) pontban adottaktól. Miért mondhatjuk mégis, hogy a két esetben kapott ponthalmazok topológiai szempontból ekvivalensek?

d. Tanulmányozzuk a cellák elhelyezkedését a két esetben, hasonlóságokat és különbségeket keresve.

22.2-2. Végezzük el a 22.2. alfejezetben leírt számításokat a (22.3) rendszerrel és a

$$T = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

tér-idő-mátrixszal kapcsolatban.



22.6. ábra. Részletes be/kiviteli séma a 22.3(a). ábrához.

22.3. A be/kiviteli séma levezetése

A 22.3(a) ábrán a ***be/kiviteli séma*** az A, B, C mátrixokhoz tartozó *adatfolyamok irányával* van csupán megadva. Az adatok be/kivitelével kapcsolatos folyamatok megértéséhez szükséges részleteket a 22.6. ábra tartalmazza.

A 22.6. ábrán látható be/kiviteli séma a 22.1(a) ábrához képest egy sor új mozzanatot tartalmaz. Itt is egy bemeneti, illetve kimeneti adatfolyam áramlik keresztül mindegyik közönséges peremcellán, a szisztolikus rács sarkaihoz pedig két-két adatfolyam tartozik. Az egy mátrixhoz tartozó beviteli cellák persze itt már nem egy egyenes vonal mentén helyezkednek el, hanem két, egymással határos perem mentén.

A 22.6. ábrán látható adatszerkezeteknek ugyanakkor más a dőlésszögük, mint a 22.1(a) ábrán. Ezen kívül a 22.6. ábrán az A és B mátrix a 22.1(a) ábrához viszonyítva adatfolyamonként kétharmaddal csökkentett sebességgel érkezik.

Kevés fáradozással alapjában véve itt is lehetséges az, hogy az adott szisz-

tolikus rácshoz elemi szinten próbáljunk találó ki/beviteli sémát építeni.

Sokkal biztosabb azonban egy formális levezetésen keresztül vezető út. A következő pontokat az eljárás egyes lépéseinek bemutatásának szenteljük.

22.3.1. Az adatszerkezetek indexeitől az iterációs vektorokig

Mindenekelőtt az absztrakt adatszerkezetek és az értékadásmentes jelölésmód konkrét változópéldányai közti összefüggést kell megvilágítanunk.

Az A mátrix minden eleme egy i sorindexszel és egy k oszlopindexszel van ellátva. Ezt a két indexet egy $w = (i, k)$ ***adatszerkezet-vektorral*** foghatjuk össze. Az a_{ik} elem a (22.3) rendszer $a(i, j, k)$ példányának felel meg, tetszőleges j-vel. E példány koordinátái $\mathbb{R}^3$-nek elemei, és mind egy egyenesen helyezkednek el a $q = (0, 1, 0)$ irány mentén. Az (i, k) adatszerkezet vektortól az (i, j, k) koordinátákra való átmenetet az alábbi leképezés írja le:

$$\begin{pmatrix} i \\ j \\ k \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} i \\ k \end{pmatrix} + j \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} . \tag{22.22}$$

A (22.3) rendszerben használt alak esetén minden egyes változópéldány (i, j, k) koordináta vektora pontosan megegyezik az értéktartomány azon iterációs vektorával, melynek kapcsán az illető változópéldány kiszámítása történik. Ezért a (22.22) képletet az adatszerkezet vektorok és az iterációs vektorok között fennálló összefüggésként is felfoghatjuk. Absztrakt módon kifejezve, a megfelelő v iterációs vektorokat megkaphatjuk a

$$v = H \cdot w + \lambda \cdot q + p \tag{22.23}$$

képlet segítségével a w adatszerkezet vektorból. A p affin vektor a mi példánk esetén mindig a nullvektor, általános esetben viszont szükség lesz rá.

Mivel $b(i, j, k) = b_{kj}$, a B mátrixnak megfelelő megjelenítés az alábbi:

$$\begin{pmatrix} i \\ j \\ k \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} k \\ j \end{pmatrix} + i \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} . \tag{22.24}$$

Ami a C mátrixot illeti, minden $c(i, j, k)$ változópéldány szükséges egy másik érték kiszámításához. Viszont az összes rögzített (i, j) indexpárral rendelkező $c(i, j, k)$ példányt tekinthetjük úgy, mint ami a c_{ij} mátrixelemhez tartozik, mivel ezek közvetlenül a c_{ij} kiszámítására használt összegző operátor sorba fejtéséből származnak. A (22.23) képletnek megfelelően tehát C-re a következőket kapjuk:

$$\begin{pmatrix} i \\ j \\ k \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} i \\ j \end{pmatrix} + k \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} . \tag{22.25}$$

22.3.2. Adatszerkezetekről készült helyzetképek

Az A, B és C mátrixok mindegyike az adatszerkezet két indexének iránya mentén jött létre: egy sor, illetve egy oszlop mentén. A $(0,1)$ különbségvektor írja le az átmenetet egyik elemről a másikra ugyanazon a soron belül, azaz a következő oszlopbeli elemet adja: $(0,1) = (x, y+1) - (x, y)$. Ugyanígy az $(1,0)$ különbségvektor az ugyanabban az oszlopban, és a következő sorban levő elemhez vezető átmenetet írja le: $(1,0) = (x+1, y) - (x, y)$.

A 22.1(a) és 22.6. ábrákon látható be/kiviteli sémák különböző helyzetképeket mutatnak be: az adatok szisztolikus rácshoz viszonyított helyzete ugyanarra az időpontra vonatkozik.

Amint a 22.6. ábrán látható, az absztrakt adatszerkezetek téglalap alakú mátrixformái ebben a helyzetképben parallelogrammává alakulnak. Ez az alkalmazott tér-idő-leképezés lineáris voltának tulajdonítható. Ezeket a paralelogrammákat is ki lehet fejezni a peremek mentén kiszámolt különbségvektorok segítségével.

Most az adatszerkezetek Δw különbségvektorait Δz térbeli különbségvektorokká szeretnénk alakítani. Ehhez olyan konkrét v, v' iterációs vektor párokat kell meghatároznunk, a (22.23) összefüggésbeli λ paraméterek megválasztása révén, melyeket a megválasztott tér-idő-leképezés ugyanarra az időpontra képez le. Hogy pontosan melyik időpontról is van szó, az itt most nem lényeges. Tehát a $\pi \cdot v = \pi \cdot v'$ egyenlőséget összevetjük azzal, hogy

$$v = H \cdot w + \lambda \cdot q + p \quad \text{és} \quad v' = H \cdot w' + \lambda' \cdot q + p \; . \tag{22.26}$$

Ez azt eredményezi, hogy

$$\pi \cdot H \cdot (w - w') + (\lambda - \lambda') \cdot \pi \cdot q = 0 \; , \tag{22.27}$$

azaz

$$\Delta\lambda = (\lambda - \lambda') = \frac{-\pi \cdot H \cdot (w - w')}{\pi \cdot q} \; . \tag{22.28}$$

A keresett Δz térbeli különbségvektor tehát a használt leképezések lineáris voltából adódóan a következőképpen számítható ki az adatszerkezet $\Delta w = w - w'$ különbségvektorából:

$$\Delta z = P \cdot \Delta v = P \cdot H \cdot \Delta w + \Delta\lambda \cdot P \cdot q \; , \tag{22.29}$$

tehát

$$\Delta z = P \cdot H \cdot \Delta w - \frac{\pi \cdot H \cdot \Delta w}{\pi \cdot q} \cdot P \cdot q \; . \tag{22.30}$$

Most meghatározzuk a (22.30) képletből a Δz térbeli különbségvektorokat az A mátrix számára. A fentiek alapján érvényes a következő:

$$H = \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix}, \; q = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \; P = \begin{pmatrix} 0 & -1 & 1 \\ -1 & 1 & 0 \end{pmatrix}, \; \pi = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}.$$

Mivel $\pi \cdot q = 1$, következik

$$\Delta z = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \cdot \Delta w + \Delta\lambda \cdot \begin{pmatrix} -1 \\ 1 \end{pmatrix} \quad \text{ahol} \quad \Delta\lambda = -\begin{pmatrix} 1 & 1 \end{pmatrix} \cdot \Delta w \, .$$

A sorok esetében a $\Delta w = (0,1)$ különbségvektorunk van, melyből a $\Delta z = (2,-1)$ térbeli különbségvektor következik. Ugyanígy az oszlopok esetében $\Delta w = (1,0)$-ból következik a $\Delta z = (1,-2)$. Egyeztessük ezt most a 22.6. ábrával, és láthatjuk, hogy az A sorai valóban a $(2,-1)$ vektor mentén helyezkednek el, az oszlopok pedig a $(1,-2)$ vektor mentén.

Ugyanígy kapjuk a B sorai esetében, hogy $\Delta z = (-1,2)$, illetve $\Delta z = (1,1)$ a B oszlopaira. Megfelelőképpen $\Delta z = (-2,1)$ a C sorai esetén, és $\Delta z = (-1,-1)$ a C oszlopai esetében.

Ezzel tehát most már meg tudjuk szerkeszteni a be/kiviteli sémát minden egyes mátrixra külön-külön.

22.3.3. A be/kiviteli séma megszerkesztése

Az A, B, C mátrixok megjelenési formája a helyzetkép számára adott ugyan, de még meg kell határozzuk a szisztolikus rácshoz (és egyúttal egymáshoz) viszonyított elhelyezkedésüket. Ennek megvalósítására létezik egy egyszerű grafikus módszer.

Kiválasztunk egy tetszőleges iterációs vektort, legyen ez $v = (1,1,1)$. A P projekciós mátrix segítségével levetítjük ezt arra a cellára, melyben a neki megfelelő számítások történnek.

$$z = \begin{pmatrix} 0 & -1 & 1 \\ -1 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} .$$

Az $(1,1,1)$ iterációs vektorhoz az $a(1,1,1)$, $b(1,1,1)$ és $c(1,1,1)$ számítások vannak hozzárendelve, melyek viszont a a_{11}, b_{11} és c_{11} mátrixelemeknek felelnek meg. Helyezzük az A, B, C mátrixok esetén kapott be/kiviteli sémát a szisztolikus rácsra úgy, hogy a megfelelő a_{11}, b_{11} és c_{11} bemenetek mind a $z = (0,0)$ cellában legyenek.

Ezzel tulajdonképpen már készen is volnánk. Ez a be/kiviteli séma azonban átfedi a szisztolikus rács celláit, és emiatt nem elég világosan felismerhető.

Ezért minden mátrix be/kiviteli sémáját egy-egy pozícióval tovább toljuk az adatfolyamok haladásával ellentétes irányba mindaddig, amíg nem áll fenn többé átfedés. Ezzel pontosan a 22.6. ábrán bemutatott be/kiviteli sémát kapjuk.

Ezen elegáns grafikus módszer mellett itt is megvan a lehetőségünk arra is, hogy az átfedésmentes elhelyezkedést formálisan számoljuk ki.

Csak a be/kiviteli séma megadása után tudjuk a szükséges időszeletek számát meghatározni. Az első lényeges időszelet a legelső adat bevitelével kezdődik. Az utolsó lényeges időszelet az utolsó adatkivitellel végződik. A 22.6. ábrán látható példa esetén a számítások kezdetét a b_{11} elem 0 időszeletben történő bevitelétől számítjuk, a számítások befejeztének pedig a 14. időszeletet tekintjük, miután a c_{35} eredmény kiszámítása megtörtént. Ez összesen 15 időszeletet tesz ki – ami öttel több a tulajdonképpeni számítások elvégzéséhez szükséges időszeletek számánál.

22.3.4. Tér-idő-leképezés által előidézett adatsebesség

Az A és B mátrix 22.1(a) ábrán látható beviteli sémája kompakt felépítésű: ha megrajzoljuk az ábrán a mátrix széleit, ennek belsejében nem találunk kihasználatlan helyet.

Más a helyzet a 22.6. ábrán. Bármelyik adatfolyamot tekintjük, mindegyik elemet két kihasználatlan hely követ. A beviteli mátrixok esetében ez azt jelenti, hogy: a szisztolikus rács peremcellái csak minden harmadik időszeletben kapnak valódi adatot.

Ez a tulajdonság az éppen alkalmazott tér-idő-leképezés egyenes következménye. Maguk az absztrakt adatszerkezetek mind a két esetben kompaktak. Azonban, hogy milyen sűrűn helyezkednek el az adatok a be/kiviteli sémában, az a T transzformációs mátrix determinánsának abszolút értékétől függ: minden egyes be/kiviteli adatfolyamban a tényleges értékek pontosan $|\det(T)|$ pozíciónyi távolságra követik egymást. Míg a 22.1. ábra esetében $|\det(T)| = 1$ érvényes, addig a 22.6. ábra esetében a $|\det(T)| = 3$ értéket állapíthatjuk meg, melynek most már gyakorlati jelentése is ismert.

Mi történik vajon a ki nem használt helyekkel, mint amilyeneket a 22.6. ábrán is láthatunk? Annak ellenére, hogy a 22.3. ábra szisztolikus rácsának minden cellája tulajdonképpen csak minden harmadik időszeletben végez értelmes munkát, nincs sok értelme annak, hogy minden munkát végző lépés után két időszelet erejéig szüneteltessük őket. Ha pontosabban megfigyeljük, megállapíthatjuk, hogy a 22.6. ábrán pontokkal jelölt helyeken lévő értékeknek semmiféle befolyásuk nincs a c_{ij} elemek kiszámítására, mivel egy $c(i,j,k)$ változó kiszámításának időpontjában soha nincsenek ott a számításnak megfelelő cellában. A nem használt helyeket tehát egyszerűen tetszőleges, akár vélet-

lenszerű értékekkel is feltölthetjük anélkül, hogy a végeredményt elrontanánk ezzel. Ráadásul az is lehetséges, hogy a 22.3. ábra szisztolikus rácsának segítségével egyidőben három különböző mátrixszorzást elvégezzünk, anélkül, hogy ezek zavarnák egymást. A 22.3.7. pontban ezt még pontosabban kifejtjük.

22.3.5. Be/kivitel kiterjesztése és a bővített be/kiviteli séma

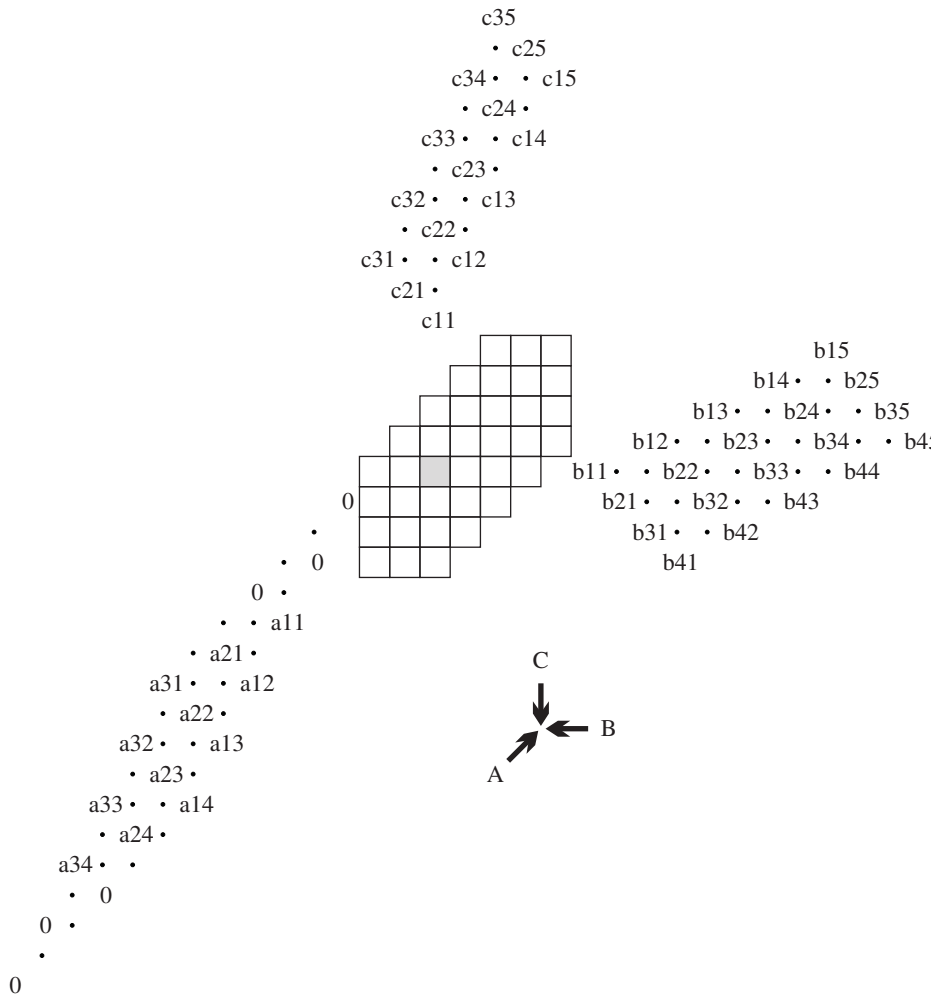
A 22.6. ábrát alaposabban tanulmányozva egy újabb kérdés merül fel. Tekintsük példaként a c_{22} útját a szisztolikus rács celláin keresztül. A tér-idő-leképezés alapján a számítások a $(-1,0)$, $(0,0)$, $(1,0)$ és $(2,0)$ cellákban mennek végbe. A 22.6. ábrán látható be/kiviteli séma szerint természetesen előbb a $(-2,0)$ cellán, illetve legvégül a $(3,0)$ cellán is keresztülhalad a c_{22}.

Ezt értelmezhetjük úgy, hogy a választott tér-idő-leképezés révén a (22.3) rendszerbe fiktív számítások kerülnek be, itt például az értéktartomány új $(2,2,0)$ és $(2,2,5)$ pontjai kapcsán. Ennek a jelenségnek az alapja az a tény, hogy a be/kimeneti műveletek értéktartományai nem a megválasztott u projekciós iránnyal párhuzamosan helyezkednek el. Ennek következtében egyes be/kimeneti műveletek olyan cellákra vetítődnek, amelyek nem a szisztolikus rács peremén vannak. Itt viszont a be/kimeneti műveletre közvetlen módon nem kerülhet sor. Az útvonalat az adatfolyamok iránya mentén meghosszabbítva, illetve ezekkel ellentétes irányban ezektől a belső celláktól a szisztolikus rács pereméig, kiküszöbölhetjük ezt a problémát. Ezzel viszont új számítások jönnek be, esetleg újabb pontokkal bővül az értéktartomány ***(be/kivitel kiterjesztése).***

Vigyáznunk kell arra, nehogy a $(-2,0)$ és $(3,0)$ cellákban végbemenő, mellékes számítások elrontsák a c_{22} tulajdonképpeni értékét. A mátrixszorzás esetében ezt elég könnyű elérni (az általános esetben viszont sokkal nehezebb). A generikus összegzőoperátornak van egy semleges eleme, éspedig a nulla. Tehát ha elérjük, hogy a kiegészítés révén bekerült számításokban mindig csak a nullát adjuk hozzá az eddigiekhez, ez semmiféle kárt nem okoz.

A 22.7. ábra egy alkalmas bővített be/kiviteli sémára ad példát. Az A mátrix elé és mögé hozzá vannak fűzve a szükséges null-elemek. Mivel a bevitt null értékeket adatnak kell tekintenünk, a 22.6. ábra be/kiviteli sémája még egy pozícióval visszatolódik.

A számítások tehát már a -1. időszeletben megkezdődnek, viszont ugyanúgy a 14. időszelettel végződnek, mint korábban. A teljes végrehajtási idő 16 időszelet lesz.



22.7. ábra. Bővített be/kiviteli séma a 22.6. ábrához.

22.3.6. A stacionárius változók kezelése

Térjünk vissza a 22.1(a) ábra példájához. Az A és B elemeinek beviteléhez nincs szükség semmiféle kiterjesztésre, mivel ezekre mindig a peremcellákban van legelőször szükség. Más a helyzet viszont a C mátrixszal. Ennek elemeit stacionárius változókban számítjuk ki, vagyis mindvégig ugyanabban a cellában. A c_{ij} eredmények tehát a szisztolikus rács belsejére esnek, ahonnan a kiszámításukat követően egy további folyamatban a szisztolikus rács peremcelláihoz kell továbbítanunk őket, mert csak ezeken keresztül férhetünk hozzájuk.

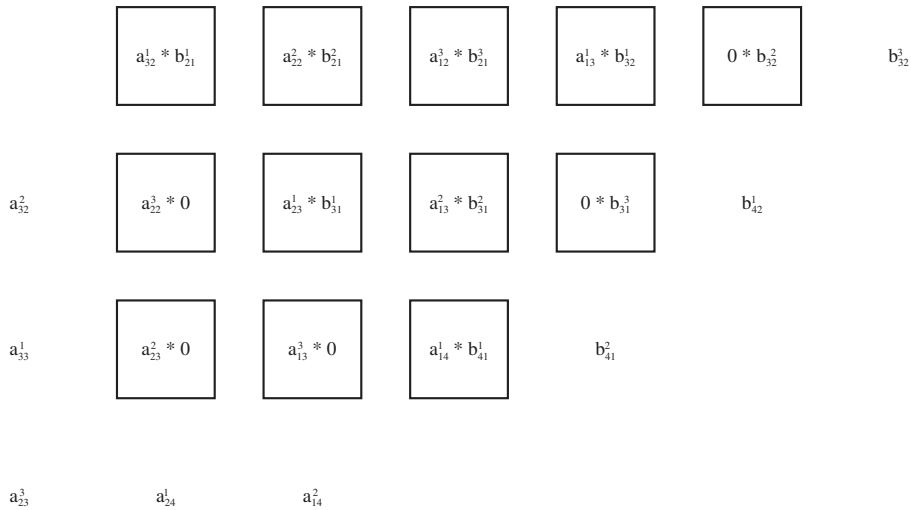
Annak ellenére, hogy a megoldandó feladatot felületesen szemlélve úgy tűnik, mintha az a 22.3.5. pontban leírthoz nagyon hasonló lenne, és emiatt igen egyszerűnek látszik, mégis egy teljesen más helyzetről van itt szó. Nem arról van szó ugyanis, hogy a meglévő adatfolyamokat előre vagy visszafelé a szisztolikus tömb pereméig meghosszabbítsuk. Hiszen stacionárius változók esetében a függőségi vektor a nullvektor. Így ez semmiféle térbeli adatfolyamot nem eredményezhet. Egy ilyen adatfolyamot nekünk kell előbb felépítenünk, amiben igen nagy a szabadságunk. Alapjában véve egy vezérlő egységre is szükségünk van a cellákban. Ezt a kérdést a 22.4. alfejezetben tárgyaljuk tovább.

22.3.7. Számítások összekapcsolása

Amint az könnyen belátható, a 22.3. ábra szisztolikus rácsának *kihasználtsága* a 22.7. ábra be/kiviteli sémájával meglehetősen csekély. Anélkül, hogy a betöltési vagy a végső szakaszt közelebbről tanulmányoznánk, máris észrevesszük, hogy a rács átlagos kihasználtsága kevesebb, mint egyharmadot tesz ki – hiszen valódi számítást minden cella legfeljebb minden harmadik időszeletben végez.

Egy egyszerű eszköz ennek a viselkedésnek a javítására a számítások ***összekapcsolása.*** Amennyiben három egymástól független, ugyanazokkal a paraméterekkel rendelkező mátrixszorzást kell elvégeznünk, ezek adatait bevihetjük egymástól csupán egy időszeletnyi távolságban, anélkül, hogy a szisztolikus rácson vagy annak celláin bármit is változtatnánk. A 22.8. ábra helyzetképet mutat a szisztolikus rács egy részletéről, a be/kiviteli séma megfelelő részeivel.

Szeretnénk egyúttal valamilyen formális levezetésen keresztül meggyőződni arról, hogy ez az ötlet valóban működik. Ennek érdekében úgy módosítjuk a (22.3) rendszert, hogy a változókat és az értéktartományt egy negyedik dimenzióval bővítjük ki, mely csupán a három mátrixszorzás



22.8. ábra. Három mátrixszorzás összekapcsolt kiszámítása a 22.3. ábra szisztolikus rácsával (részlet).

megkülönböztetésére szolgál:

Bemeneti műveletek

$$
\begin{aligned}
a(i,j,k,n) &= a^n_{ik} && 1 \le i \le N_1, j = 0, 1 \le k \le N_3, 1 \le n \le 3\ , \\
b(i,j,k,n) &= b^n_{kj} && i = 0, 1 \le j \le N_2, 1 \le k \le N_3, 1 \le n \le 3\ , \\
c(i,j,k,n) &= 0 && 1 \le i \le N_1, 1 \le j \le N_2, k = 0, 1 \le n \le 3\ .
\end{aligned}
$$

Számítások

$$
\begin{aligned}
a(i,j,k,n) &= a(i,j-1,k,n) && 1 \le i \le N_1, 1 \le j \le N_2, 1 \le k \le N_3, 1 \le n \le 3\ , \\
b(i,j,k,n) &= b(i-1,j,k,n) && 1 \le i \le N_1, 1 \le j \le N_2, 1 \le k \le N_3, 1 \le n \le 3\ , \\
c(i,j,k,n) &= c(i,j,k-1,n) && 1 \le i \le N_1, 1 \le j \le N_2, 1 \le k \le N_3, 1 \le n \le 3\ . \\
&\quad + a(i,j-1,k,n) \cdot b(i-1,j,k,n)
\end{aligned}
$$

Kimeneti műveletek

$$
c^n_{ij} = c(i,j,k,n) \qquad 1 \le i \le N_1, 1 \le j \le N_2, k = N_3, 1 \le n \le 3\ . \tag{22.31}
$$

Nyilvánvaló, hogy a (22.31) rendszerben a különböző n értékekhez tartozó feladatoknak semmi közük egymáshoz. Ennek tehát a szisztolikus rácsban is így kell lennie. Egy erre vonatkozó, az ábrának megfelelő tér-idő-mátrix a

következő

$$T = \begin{pmatrix} 0 & -1 & 1 & 0 \\ -1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix} . \tag{22.32}$$

A T mátrix itt már nem négyzetes, de ez nem tesz semmit. A térkoordináták kiszámításának szempontjából a negyedik dimenzió teljesen jelentéktelen; ezért a T első két sorának utolsó oszlopában levő null-értékek segítségével egyszerűen eltüntethető.

T utolsó sora újraépíti a π idővektort. A π megfelelő megválasztásával a három megoldásra váró feladat átfedésmentesen ágyazható be a tér-idő-szerkezetbe. A három feladat egymásnak megfelelő iterációs vektorainak példányai egymástól egy időegységnyi távolságban ugyanarra a cellára lesznek vetítődnek, mivel π negyedik bemenete 1.

Kiszámítjuk most az átlagos ***kihasználtságot*** összekapcsolással, illetve e nélkül az $N_1 = 3$, $N_2 = 5$ és $N_3 = 4$ konkrét feladat paraméterei esetében. Egyetlen mátrixszorzás esetén $N_1 \cdot N_2 \cdot N_3 = 60$ számítást kell elvégezni. Ilyenkor egy szorzás és a hozzá tartozó összeadás összetett műveletnek számít, azaz együtt csupán egyetlen számítást jelent; a be/kimeneti műveleteket nem számítjuk hozzá. A szisztolikus rács 36 cellával rendelkezik.

Összekapcsolás nélkül a szisztolikus rácsunknak 16 időszeletre van szüksége egy mátrixszorzás elvégzéséhez. Ez a cellák következő átlagos kihasználtságát eredményezi:

$$60/(16 \cdot 36) \sim \ 0.104 \text{ számítás időszeletenként és cellánként.}$$

A leírt összekapcsolási technika alkalmazása esetén mindhárom mátrix kiszámítása csupán két időszelettel igényel többet, tehát 18 időszeletet tesz ki. Ám a végrehajtott számítások száma megháromszorozódott, a cellák átlagos kihasználtsága tehát a következő:

$$3 \cdot 60/(18 \cdot 36) \sim \ 0.278 \text{ számítás időszeletenként és cellánként.}$$

Az összekapcsolással tehát körülbelül 167 százalékkal sikerült növelnünk a kihasználtságot.

Gyakorlatok

22.3-1. Vezessük le formálisan a B és C mátrix térbeli különbségvektorait a 22.6. ábrán látható be/kiviteli séma esetén a (22.30) összefüggés alapján.

22.3-2. Vázoljuk a bővített be/kiviteli sémát a 22.6. ábrához, arra az esetre, amikor a többlet szorzásműveletek mindkét operandusát nullára kell állítanunk.

22.3-3. Alkalmazzuk a 22.3. alfejezetben bemutatott módszereket a 22.1.

ábra szisztolikus rácsára.

22.3-4. * Igazoljuk a (22.32) sajátos tér-idő-leképezés 22.3.7. pontban leírt tulajdonságait a (22.31) rendszerre vonatkozóan.

22.4. Vezérlési szempontok

Mindeddig abból indultunk ki, hogy egy szisztolikus rács cellái teljesen egyformán viselkednek minden időszeletben. Érdekes példákat találhatunk ilyen szisztolikus rácsokra. Általában azonban vezérlés segítségével a cellákat különböző *működési módokba* lehet állítani. A következőkben néhány tipikus helyzetet fogunk tanulmányozni.

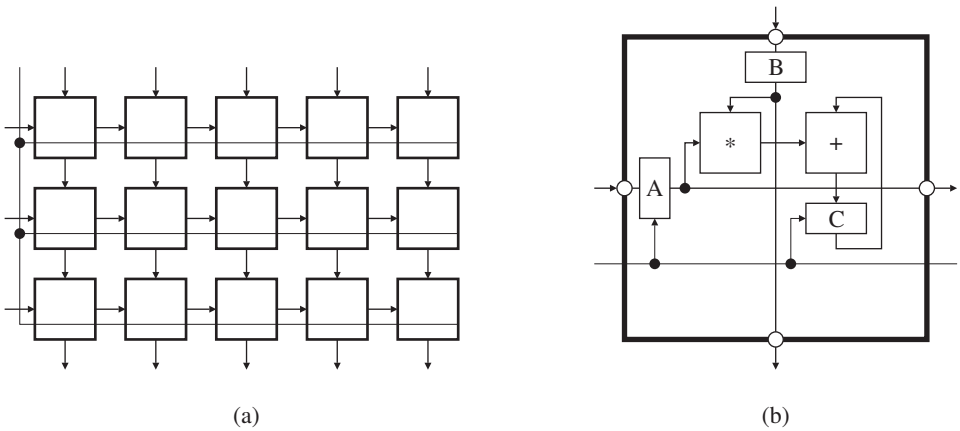
22.4.1. Vezérlésmentes cellák

A 22.3(b) ábrán látható cella az A, B, C regisztereket tartalmazza, amelyek egy globális órajel aktivál annak érdekében, hogy átvegyék a mindenkori bemenetükön levő jeleket, majd a kimenetükre továbbítsák ezeket. Ettől a globális aktiválástól eltekintve azonban, a sejtek működése minden időszeletben ugyanaz: a három, A, B, C bemenő operandusra egy *szorzás-összeadás* művelet hajtódik végre, melynek az eredménye egy szomszédos cellába kerül; ezzel párhuzamosan az A és B operandusok két másik szomszédos cellának továbbítódnak. Következésképpen ez a cella semmiféle vezérléssel nem rendelkezik.

A generikus összegző operátor végrehajtásához szükséges $c(i, j, 0)$ kezdőértékek, melyeknek itt nem kell feltétlenül nullértékeknek lenniük, a 22.6. ábra alapján bemeneti adatfolyamként kerülnek a szisztolikus rácsba, a $c(i, j, N_3)$ eredmények pedig a rács peremén keresztül, ugyanabban az irányban áramlanak kifelé. A 22.3(b) ábra szerint tehát a be/kimenet implicit módon része a cellák működésének. Ennek a nagyon egyszerű, vezérlésmentes sejtműködésnek az ára mindhárom mátrixméret korlátozása: $(M_1 \times M_3)$-as A mátrix csak akkor szorozható össze a 22.3. ábra szisztolikus rácsán keresztül egy $(M_3 \times M_2)$-es B mátrixszal, ha a rács meghatározott N_1, N_2, N_3 paramétereire a következő feltételek érvényesülnek: $M_1 \leq N_1$, $M_2 \leq N_2$ és $M_3 \leq N_3$.

22.4.2. Globális vezérlésű cellák

A 22.1. ábrán látható szisztolikus rács esetén az előírások erre vonatkozóan kevésbé szigorúak: bár M_1 és M_2 itt is korlátozva van $M_1 \leq N_1$ és $M_2 \leq N_2$ által, az M_3-ra viszont nincs semmi korlátozás. A feladat azon paraméterei, melyeket a rács előre meghatározott paraméterei nem korlátoznak, csak



22.9. ábra. Regiszterek visszaállítása globális vezérléssel: **(a)** rácsszerkezet; **(b)** cellaszerkezet.

időben tudnak megnyilvánulni, térben azonban nem. Ezáltal *stacionárius változók* használatára kényszerülünk.

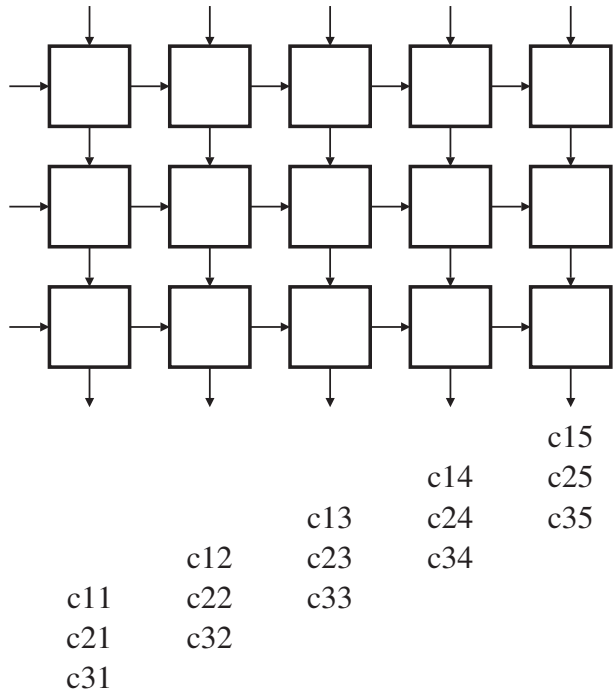
Egy új számítás kezdetén minden egyes stacionárius változóhoz rendelt regisztert a megelőző számításoktól független alapállapotba kell hozni. A 22.3(b) ábrán látható szisztolikus cella esetében ez a regiszter a C. Egy, az órajelhez hasonlítható globális jel segítségével az összes cella C regisztere egyszerre törölhető, azaz lenullázható. A 22.9. ábra egy ezen az ötleten alapuló rácsszerkezetet, illetve cellaszerkezetet mutat be.

22.4.3. Helyi vezérlés

Sajnos a mátrixszorzáshoz nem elegendő csupán a globális vezérlés elve. A 22.1. ábrán bemutatott szisztolikus rácsból ugyanis két lényeges tulajdonság is hiányzik: egyrészt a $c(i,j,0)$ változóknak nincs kezdőértékük, másrészt a c_{ij} eredmények sem a rács peremén jelennek meg.

Először az eredmények perem felé vezérlése valóban egyszerű feladatnak tűnik: egy c_{ij} elem kiszámításának befejeztével már nincs szükség arra, hogy az (i,j) cellának a szomszédos $(i,j+1)$ és $(i+1,j)$ cellák felé vezető kapcsolatait az A és B mátrix elemeinek továbbadására használjuk. Használhatjuk tehát más célokra őket. Például a C összes elemét a lefelé vezető kapcsolatokon keresztül a szisztolikus rács alsó pereméhez vezethetjük.

Sajnos kiderül azonban, hogy a rács alsó részén még be nem fejezett számítások akadályozzák az eredmények átvezetését a fenti részből. Ha az $i+j+N_3$ időszeletben kiszámított c_{ij} eredményt a következő időszeletben az $(i+1,j)$ cellának továbbítanánk, ez értékütközéshez vezetne: mivel az



22.10. ábra. Be/kiviteli séma késleltetett eredménymegadás esetén.

$(i+1,j)$ cella időszeletenként csak egy értéket bocsáthat ki az alsó csatornán keresztül, vagy a c_{ij}-nek kellene várnia, vagy az $(i+1,j)$ cellában kiszámolt eredménynek. Ugyanez a probléma lefelé az összes további cellát érintené.

Megoldásképpen késleltethetjük a c_{ij} továbbadását. Ha c_{ij}-nek egy cellán való keresztülhaladáshoz nem egy, hanem két időszeletre van szükség, akkor az ütközések elmaradnak. Az eredmények így egymástól egy időszeletnyi eltéréssel haladnak egymást követve, ugyanazon a kapcsolaton keresztül. Egy oszlop alsó peremcelláján legelőször az illető oszlop utolsó sorának eleme jelenik meg, majd az utolsó előtti, legvégül az első. Tehát a 22.10. ábrán látható be/kimeneti sémát kapjuk.

Honnan tudja egy cella, mikor kell az alsó csatornán keresztül továbbítania, ahelyett hogy a B mátrix elemeit a C mátrix elemeivel együtt adná tovább? Ezt a feladatot a cella globális és lokális vezérlésének kombinált alkalmazásával, véges áállapotú automaták segítségével oldhatjuk meg.

Amikor az A és B utolsó értékeit is bevisszük az $(1,1)$ cellába, egy globális jelet küldünk az összes cellának, ami jelzi, hogy minden cellában elindíthatunk egy számlálót, amely a még elvégzendő számítási lépések számát adja meg. Az (i,j) cellában még $i+j-1$ további lépést kell elvégezni, mielőtt a sima

továbbvezetésre kapcsolnánk át. A korábban említett globális visszaállító jel később ismét visszakapcsol majd a számítási üzemmódba.

Egy ilyen elv alapján működő szisztolikus rács látható a 22.11. ábrán. A rács felépítése, valamint a kapcsolatszerkezet lényegében változatlanok maradtak. Viszont minden cella kétféle üzemmódban működtethető, melyek között az átkapcsolást egy vezérlőlogika végzi:

1. *Számítási üzemmódban* az összeadás eredménye (akárcsak eddig) az C regiszterbe íródik. Ezzel egy időben a szorzásra használt operandus az B regiszterből a cella alsó csatornáján keresztül átadódik.

2. *Adattovábbító üzemmódban* az B és C regiszterek sorba lesznek kapcsolva. Ebben az üzemmódban a cellák egyetlen feladata az, hogy minden, a felső csatornán kiolvasott értéket két időszeletnyi késleltetéssel továbbítson az alsó csatorna felé.

Adattovábbító üzemmódban az (i, j) cellából kilépő első érték a legutoljára kiszámított, majd az C regiszterben elhelyezett érték lesz, azaz a c_{ij} eredmény. A továbbiakban kilépő összes érték a fentebb elhelyezkedő cellákban kiszámolt, majd innen továbbvezetett eredmény. A 22.11 ábrán megvalósított algoritmus formális leírása a (22.33) értékadásmentes rendszert eredményezi.

Bemeneti műveletek

$$
\begin{aligned}
&a(i,j,k) = a_{ik} && 1 \le i \le N_1, j = 0, 1 \le k \le N_3 \ , \\
&b(i,j,k) = b_{kj} && i = 0, 1 \le j \le N_2, 1 \le k \le N_3 \ , \\
&c(i,j,k) = 0 && 1 \le i \le N_1, 1 \le j \le N_2, k = 0 \ .
\end{aligned}
$$

Számítások

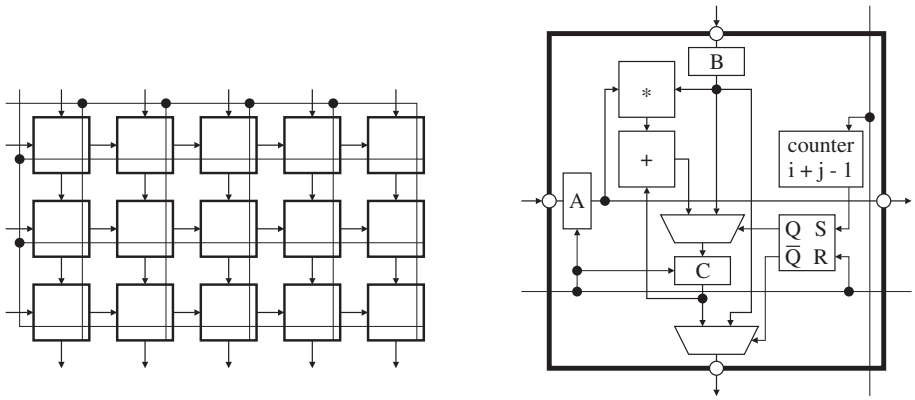
$$
\begin{aligned}
&a(i,j,k) = a(i,j-1,k) && 1 \le i \le N_1, 1 \le j \le N_2, 1 \le k \le N_3 \ , \\
&b(i,j,k) = b(i-1,j,k) && 1 \le i \le N_1, 1 \le j \le N_2, 1 \le k \le N_3 \ , \\
&c(i,j,k) = c(i,j,k-1) && 1 \le i \le N_1, 1 \le j \le N_2, 1 \le k \le N_3 \\
&\quad + a(i,j-1,k) \cdot b(i-1,j,k) \ .
\end{aligned}
$$

Átvezetés

$$
\begin{aligned}
&b(i,j,k) = c(i,j,k-1) && 1 \le i \le N_1, 1 \le j \le N_2, 1 + N_3 \le k \le i + N_3 \ , \\
&c(i,j,k) = b(i-1,j,k) && 1 \le i \le N_1, 1 \le j \le N_2, 1 + N_3 \le k \le i - 1 + N_3 \ ,
\end{aligned}
$$

Kimeneti műveletek

$$
c_{1+N_1+N_3-k,j} = b(i,j,k) \qquad i = N_1, 1 \le j \le N_2, 1 + N_3 \le k \le N_1 + N_3 \ . \tag{22.33}
$$



22.11. ábra. A helyi és a globális vezérlés kombinált alkalmazása: **(a)** a rács felépítése; **(b)** cellaszerkezet.

Tisztáznunk kell még, hogy hogyan hozzuk létre egy cella vezérlőjeleit ebben a modellben. A cellának mindenekelőtt egy flipflop állapotkapcsolóval kell rendelkeznie, mely az éppen bekapcsolt üzemmódot adja meg. Ezen flipflop kimenete hozzákapcsolódik a 22.11(b) ábrán látható mindkét multiplexer vezérlőbemenetéhez. A globális visszaállító jel nem csak a cella C regiszterét állítja vissza, hanem a flipflop állapotkapcsolót is: a cella tehát számítási üzemmódban fog dolgozni.

Amikor a globális végjel megérkezik, a cellában egy visszaszámláló indul el, amely minden időszeletben eggyel csökken. A számláló kezdőértéke – cellától függően – $i+j-1$ értékre lesz állítva. Amikor a számláló eléri a nulla értéket, a flipflop ismét átáll: a cella adattovábbító üzemmódba megy át.

A visszaállítás előtti utolsó, egy cella C regiszterébe továbbított érték felhasználható a cellában kiszámítandó következő skalárszorzat szabadon választható kezdőértékeként, ha az C regiszter közvetlen visszaállításáról lemondunk. Ezután, akárcsak a 22.3. ábrán látható szisztolikus rács esetében, az általános

$$C = A \cdot B + D \ , \tag{22.34}$$

feladatot a következő képletek segítségével oldjuk meg:

Bemeneti műveletek

$$\begin{aligned} a(i,j,k) &= a_{ik} && 1 \le i \le N_1, j = 0, 1 \le k \le N_3 \ , \\ b(i,j,k) &= b_{kj} && i = 0, 1 \le j \le N_2, 1 \le k \le N_3 \ , \\ c(i,j,k) &= d_{ij} && 1 \le i \le N_1, 1 \le j \le N_2, k = 0 \ . \end{aligned}$$

Számítások (22.35)

$$\begin{aligned} a(i,j,k) &= a(i,j-1,k) && 1 \le i \le N_1, 1 \le j \le N_2, 1 \le k \le N_3 \ , \\ b(i,j,k) &= b(i-1,j,k) && 1 \le i \le N_1, 1 \le j \le N_2, 1 \le k \le N_3 \ , \\ c(i,j,k) &= c(i,j,k-1) && 1 \le i \le N_1, 1 \le j \le N_2, 1 \le k \le N_3 \\ &\quad + a(i,j-1,k) \cdot b(i-1,j,k). \end{aligned}$$

Átvezetés

$$\begin{aligned} b(i,j,k) &= c(i,j,k-1) && 1 \le i \le N_1, 1 \le j \le N_2, 1 + N_3 \le k \le i + N_3 \ , \\ c(i,j,k) &= b(i-1,j,k) && 1 \le i \le N_1, 1 \le j \le N_2, 1 + N_3 \le k \le i - 1 + N_3 \ . \end{aligned}$$

Kimeneti műveletek

$$c_{1+N_1+N_3-k,j} = b(i,j,k) \quad i = N_1, 1 \le j \le N_2, 1 + N_3 \le k \le N_1 + N_3 \ . \tag{22.36}$$

22.4.4. Osztott vezérlés

A 22.11. ábrán bemutatott módszernek a következő hátrányai vannak:

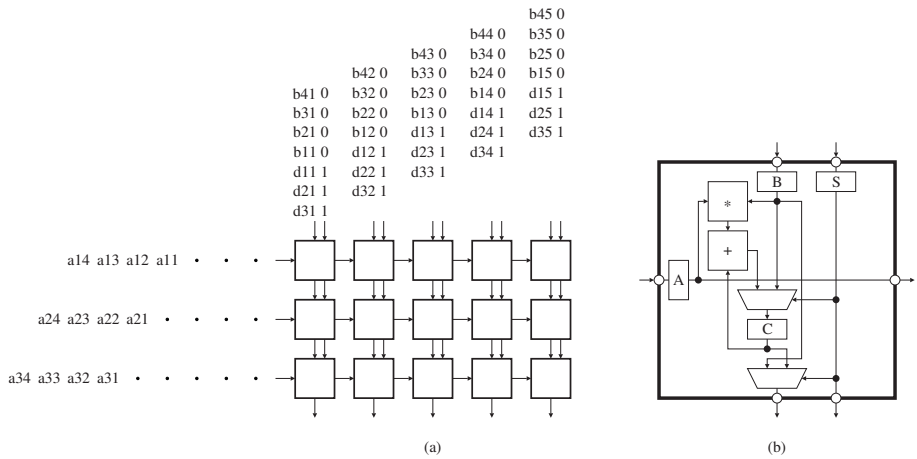
1. Globális vezérlőjeleket használunk. Ez magas fokú technikai pontosságot igényel.

2. Minden cellának egy számlálóra van szüksége, mely igen nagy ráfordítást igényel.

3. A számlálók kezdőértéke nem azonos minden cella esetében. Ezért minden cellát egyénileg kell megtervezni és megvalósítani.

4. Egy új feladat bevitelével meg kell várni, amíg az utolsó eredmény elhagyja a szisztolikus rácsot.

Ezek a hátrányok eltűnnek, ha a vezérlőjeleket az adatokhoz hasonlóan továbbítjuk, tehát osztott vezérlést alkalmazunk. Megtartjuk ugyan a 22.11(b) ábrán látható módon az B és C regiszterek multiplexerekhez való kapcsolását, nem hozunk viszont létre vezérlőjeleket a cellákon belül; a globális visszaállító



22.12. ábra. Mátrixszorzás téglalap alakú szisztolikus ráccsal, az eredmények kivitelével és osztott vezérléssel: **(a)** a rács felépítése; **(b)** cellaszerkezet

jeltől is eltekintünk. Ehelyett kívülről vezetjük be a cellákba a szükséges vezérlőjeleket, egy (csupán 1 bit szélességű) S pótregiszterben tároljuk, majd onnan a szomszédos cellákba megfelelő módon továbbítjuk. A tulajdonképpeni vezérlőjel létrehozását a gazdagép veszi át, a betáplálás kizárólag a peremcellákon keresztül történik. A 22.12(a) ábra az ehhez szükséges rácsfelépítést, a 22.12(b) ábra pedig a módosított cellaszerkezetet mutatja be.

Az adattovábbító üzemmódba való átkapcsolás egy oszlop cellái esetén lefele haladva egy-egy időszeletnyi különbséggel következik be. Ezért elegendő csupán az S regiszter okozta késleltetés.

A számítási üzemmódba való visszaállítás ugyanazon vezérfonal mentén következik be, tehát ugyanúgy cellánként egy időszeletnyi késleltetéssel történik. Mivel a c_{ij} eredmények csak fele akkora sebességgel mozognak lefelé, a cellák visszaállításával megfelelő ideig várni kell: ha egy cella a t időszeletben lett számítási üzemmódba állítva, akkor a $t + N_3$ időszeletben kapcsol át adattovábbító üzemmódba, és a $t + N_1 + N_3$ időszeletben ismét visszakapcsol számítási üzemmódba.

Amint látjuk, a szisztolikus rácsok osztott vezérlése a lokális/globális vezérléstől makroszkopikus időtényezőben különbözik. Míg a 22.12. ábrán látható szisztolikus rács minden $N_1 + N_3$ időszeletben egy új (22.34) feladat megoldásába kezdhet, ugyanez a 22.11. ábrán látható szisztolikus rács esetében csak minden $2N_1 + N_2 + N_3 - 2$ időszeletben lehetséges. Az $N_1 + N_3$, illetve $2N_1 + N_2 + N_3 - 2$ időkülönbséget ***periódusnak*** nevezzük, inverzét pedig ***teljesítménynek***.

A (22.37) rendszer az osztott vezérlés és számítás közötti formális összefüggéseket írja le. Egy tetszőleges hosszúságú, lehető legszorosabban egymást követő mátrixszorzásokból álló sorozatból indulunk ki, a bevezetett n iterációs változó ezért korlátlan.

Vezérlés

$$
\begin{aligned}
&s(i,j,k,n) = 0 && i = 0, 1 \le j \le N_2, 1 \le k \le N_3 \,. \\
&s(i,j,k,n) = 1 && i = 0, 1 \le j \le N_2, 1 + N_3 \le k \le N_1 + N_3 \\
&s(i,j,k,n) = s(i-1,j,k,n) && 1 \le i \le N_1, 1 \le j \le N_2, 1 \le k \le N_1 + N_3
\end{aligned}
$$

Adatbevitel

$$
\begin{aligned}
&a(i,j,k,n) = a^n_{ik} && 1 \le i \le N_1, j = 0, 1 \le k \le N_3 \,, \\
&b(i,j,k,n) = b^n_{kj} && i = 0, 1 \le j \le N_2, 1 \le k \le N_3 \,, \\
&b(i,j,k,n) = d^{n+1}_{N_1+N_3+1-k,j} && i = 0, 1 \le j \le N_2, 1 + N_3 \le k \le N_1 + N_3
\end{aligned}
$$

Fedőnévvel rendelkező változók

$$
c(i,j,k,n) = c(i,j,N_1+N_3,n-1) \qquad 1 \le i \le N_1, 1 \le j \le N_2, k = 0 \,.
$$

Adatokkal végzett műveletek

$$
\begin{aligned}
&a(i,j,k,n) = a(i,j-1,k,n) && 1 \le i \le N_1, 1 \le j \le N_2, 1 \le k \le N_1 + N_3 \\
&b(i,j,k,n) = \begin{cases} b(i-1,j,k,n) & : s(i-1,j,k,n) = 0 \\ c(i,j,k-1,n) & : s(i-1,j,k,n) = 1 \end{cases} && 1 \le i \le N_1, 1 \le j \le N_2, 1 \le k \le N_1 + N_3 \\
&c(i,j,k,n) = \begin{cases} c(i,j,k-1,n) \\ \quad + a(i,j-1,k,n) \\ \quad \cdot b(i-1,j,k,n) & : s(i-1,j,k,n) = 0 \\ b(i-1,j,k,n) & : s(i-1,j,k,n) = 1 \end{cases} && 1 \le i \le N_1, 1 \le j \le N_2, 1 \le k \le N_1 + N_3
\end{aligned}
$$

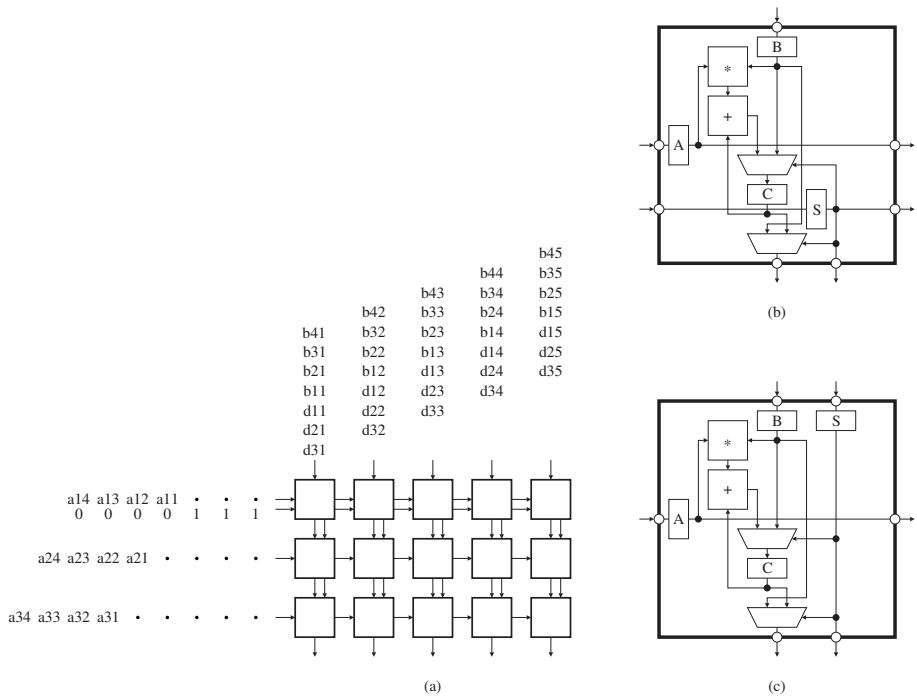
Adatkivitel

$$
c^n_{1+N_1+N_3-k,j} = b(i,j,k,n) \qquad i = N_1, 1 \le j \le N_2, 1 + N_3 \le k \le N_1 + N_3 \tag{22.37}
$$

A (22.38) képlet a hozzátartozó tér-idő-mátrixot mutatja, amelyben az egyik elem nem konstans, hanem a feladat paramétereitől függ.

$$
T = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & N_1 + N_3 \end{pmatrix} \,. \tag{22.38}
$$

Feltűnik, hogy az egy sorhoz tartozó cellák esetében jobbra haladva szintén egy időszeletnyi különbséggel kell átkapcsolni ezeket. A szisztolikus rács teljes szabályosságának követelményét figyelembe véve, ez a körülmény felhasználható arra, hogy a vezérlőjeleket csak az $(1,1)$ cellán keresztül tápláljuk be, felszabadítva ezáltal a gazdagépet. Ekkor a vezérlést a következőképpen



22.13. ábra. Mátrixszorzás téglalap alapú szisztolikus ráccsal, eredménytovábbítással és osztott vezérléssel: **(a)** A rács felépítése; **(b)** a felső perem cellái; **(c)** a fennmaradó területek cellái.

módosítanánk:

Vezérlés

$$
\begin{aligned}
&s(i,j,k,n) = 0 && i = 1, j = 0, 1 \le k \le N_3 \ , \\
&s(i,j,k,n) = 1 && i = 1, j = 0, 1 + N_3 \le k \le N_1 + N_3 \ , \\
&s(i,j,k,n) = s(i-1,j,k,n) && 1 \le i \le N_1, 1 \le j \le N_2, 1 \le k \le N_1 + N_3 \ , \\
&\ldots
\end{aligned}
$$

Fedőnévvel rendelkező változók

$$
\begin{aligned}
&s(i,j,k,n) = s(i+1,j-1,k,n) && i = 0, 1 \le j \le N_2, 1 \le k \le N_1 + N_3 \ . \\
&\ldots
\end{aligned}
\tag{22.39}
$$

A 22.13. ábra e módosítás eredményét mutatja. Két cellatípus létezik tehát: a szisztolikus rács felső szélén található cellák (22.13(b) ábra), és az összes többi cella (22.13(c) ábra). A kapcsolatszerkezet is csupán igen csekély

eltérést mutat s szisztolikus rács felső szélén, a szabályos területhez képest.

22.4.5. A cellaprogram, mint lokális szemléletmód

Egy cella működését egy ***cellaprogrammal*** is kifejezhetjük. Ez különösen akkor érdekes, ha rendelkezésünkre áll egy programozható szisztolikus rács, melynek celláit valójában egy ismételt módon végrehajtott program vezérli.

Akárcsak a globális nézetet, azaz a szisztolikus rács architektúráját, a lokális nézetet, azaz a cellaprogramot is a *tér-idő-leképezés* határozza meg. Ez azonban csak implicit alakban adódik, ezért először matematikai transzformációval explicit alakra kell átalakítani, amely aztán alkalmas lesz cellaprogramnak.

A programváltozók példányait általános alakban ***indexkifejezések*** írják le, melyek az iterációs változókra hivatkoznak. Tekintsük például a következő egyenletet a (22.3) rendszerből:

$$c(i,j,k) = c(i,j,k-1)+a(i,j-1,k)\cdot b(i-1,j,k) \quad 1 \le i \le N_1, 1 \le j \le N_2, 1 \le k \le N_3 \; .$$

A c programváltozók $c(i,j,k-1)$ példánya az i, j és $k-1$ indexkifejezésekkel rendelkezik, melyek az i,j,k iterációs változók függvényeiként is felfoghatók.

Amint láthattuk, a (22.11) tér-idő-leképezését alkalmazva a (22.13)-beli T transzformációs mátrixszal, az értéktartomány (i,j,k) iterációs vektorainak halmaza az (x,y,t) tér-idő-koordináták halmazába megy át:

$$\begin{pmatrix} x \\ y \\ t \end{pmatrix} = T \cdot \begin{pmatrix} i \\ j \\ k \end{pmatrix} = \begin{pmatrix} 0 & -1 & 1 \\ -1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} i \\ j \\ k \end{pmatrix} \; . \tag{22.40}$$

Mivel minden cellát az (x,y) térbeli koordinátája jellemez, és a hozzá tartozó cellaprogramnak a múló t időre is hivatkoznia kell, a programváltozók indexkifejezéseiben előforduló i,j,k iterációs változók nem használhatók többé, és át kell írni őket az új, x,y,t koordinátákra. Ehhez az i,j,k iterációs változókat a (22.40)-beli tér-idő-leképezés inverzének segítségével fejezzük ki, az (x,y,t) tér-idő-koordináták függvényében.

$$\begin{pmatrix} i \\ j \\ k \end{pmatrix} = T^{-1} \cdot \begin{pmatrix} x \\ y \\ t \end{pmatrix} = \frac{1}{3} \cdot \begin{pmatrix} -1 & -2 & 1 \\ -1 & 1 & 1 \\ 2 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ t \end{pmatrix} \; . \tag{22.41}$$

Egy ilyen inverz leképezés akkor létezik, ha a tér-idő-leképezés injektív az értéktartományon – és ennek mindig igaznak kell lennie, mert különben ugyanabban az időszeletben egyetlen cellában egyszerre több számítást kellene végrehajtani. A példában az invertálhatóságot a négyzetes, nem szinguláris T mátrix az értéktartományra való hivatkozás nélkül is biztosítja. A π idővektorra és u projekciós irányra vonatkozóan elegendő a következő tulajdonság: $\pi \cdot u \neq 0$.

Az iterációs változók tér-idő-koordinátákkal való lecserélésével, mely az ***értéktartományok transzformációjaként*** is értelmezhető, általában kevésbé szép, új

index-kifejezéseket kapunk. A $c(i,j,k-1)$ tehát így alakul:

$$c((-x-2\cdot y+t)/3,(-x+y+t)/3,(2\cdot x+y+t)/3)\ .$$

Az ***indexhalmazok*** egy utólagos ***transzformációjával*** viszont átnevezhetjük a programváltozók példányait úgy, hogy a cellára illetve időre történő hivatkozások tisztábban kivehetők legyenek. Különösen ajánlatos az egyenletrendszert ismét ***kimeneti normál formába*** hozni, azaz a t időszeletben az (x,y) cellában kiszámított eredményeket a programváltozók (x,y,t) példányaiként jelölni.

Ennek az eljárásnak a megértését leginkább egy absztrakt matematikai formalizmussal érhetjük el, melyet végül a mi speciális helyzetünkre alkalmazunk. Legyen egy kvantifikált egyenlet r és s programváltozókkal, valamint S értéktartománnyal a következőképpen adott:

$$r(\psi_r(v))=\mathcal{F}(\ldots,s(\psi_s(v)),\ldots),\qquad v\in S\ . \tag{22.42}$$

A ψ_r, valamint ψ_s indexfüggvények a programváltozók példányait indexkifejezéspárokként állítják elő.

Az értéktartomány egy S-en injektív φ függvény segítségével történő transzformációja által a (22.42) a következőképpen alakul át:

$$r(\psi_r(\varphi^{-1}(e)))=\mathcal{F}(\ldots,s(\psi_s(\varphi^{-1}(e))),\ldots),\qquad e\in\varphi(S)\ , \tag{22.43}$$

ahol φ^{-1} egy függvény, amely $\varphi(S)$-en a φ inverz függvényét képezi. Az új indexfüggvények $\psi_r\circ\varphi^{-1}$, valamint $\psi_s\circ\varphi^{-1}$.

Az indexhalmazok transzformációinak nincs közük az értéktartományokhoz és minden egyes programváltozóra külön végrehajthatók, mivel csak ennek az egy programváltozónak a példányait nevezik át következetes módon. A ϑ_r és ϑ_s átnevezésekkel (22.43) a következőképpen alakul:

$$r(\vartheta_r(\psi_r(\varphi^{-1}(e))))=\mathcal{F}(\ldots,s(\vartheta_s(\psi_s(\varphi^{-1}(e)))),\ldots),\qquad e\in\varphi(S)\ . \tag{22.44}$$

Amennyiben a kimeneti normál formát szeretnénk megkapni, a $\vartheta_r\circ\psi_r\circ\varphi^{-1}$-nek az identitásfüggvénynek kell lennie.

A (példában látható) legegyszerűbb esetben ψ_r mindig az identitásfüggvény és ψ_s egy $\psi_s(v)=v-d$ alakú affin leképezés, konstans d-vel, a már ismert függőségi vektorral. ψ_r ugyanúgy fejezhető ki $d=\vec{0}$-ral. Az értéktartományok transzformációja a $\varphi(v)=T\cdot v$ tér-idő-leképezéssel történik, ahol T invertálható mátrix. Minden indextranszformáció esetében egyértelműen $(\vartheta=\varphi)$-t választjuk. Tehát a (22.44) a következőképpen alakul:

$$r(e)=\mathcal{F}(\ldots,s(e-T\cdot d),\ldots),\qquad e\in T(S)\ . \tag{22.45}$$

Egy *cellaprogram* előállításakor a végrehajtandó műveleteknek, az adatok forrásának, valamint az eredmények rendeltetési helyének (az assembly programokból ismert `opc, src, dst`) minden egyes időszeletben pontosan adottnak kell lennie.

Az éppen végrehajtandó művelet (`opc`) közvetlenül az $\mathcal{F}$ függvényből adódik. A vezérléssel ellátott cellák esetében meg kell még állapítani azt az időtartamot is,

mely alatt ez a speciális $\mathcal{F}$ végrehajtódik. Ez meghatározható a térkoordináták függvényében, a $T(S)$ időtengelyre való levetítéssel, egy általános poliéderes S esetében például egy „Fourier-Motzkin elimináció” útján.

A (22.45) rendszerből adódik az új $T{\cdot}d$ függőségi vektor, amely két komponensből áll, egy térbeli (vektoriális), és egy időbeli (skaláris) részből. A Δz térbeli rész, mint különbségvektor leírja, hogy a szomszédos cellák melyikében számítottuk ki az operandust. Ezt az operandusoknak a z cellába történő bevitelével kapcsolatos információt közvetlenül átalakíthatjuk egy $-\Delta z$ pozíciójú csatorna-jelöléssé (`src`). Ennek megfelelően az operandusokat kiszámító $z - \Delta z$ cella ezt az értéket egy Δz pozíciójú csatornán keresztül adja át (`dst`).

A $T \cdot d$ időbeli része a Δt időkülönbség által megadja, hogy mikor történt az operandusok kiszámítása. Ez az információ az olvasó z cella számára jelentéktelen, mivel a szomszédos cellákból mindig a közvetlenül megelőző időszelet kimenetét olvassa ki. Azonban ha $\Delta t > 1$, akkor az operandust abban a $z - \Delta z$ cellában, amelyben kiszámították, $\Delta t - 1$ időegyszeleten keresztül tárolni kell. Ez megoldható például úgy, hogy a $z - \Delta z$ cella programja $\Delta t - 1$ másolási műveletet hajt végre, melynek során az operandusok értékei $\Delta t - 1$ regiszteren haladnak keresztül, míg a cella tényleges kimenetéhez érnek.

Ezt a módszert alkalmazva a (22.37)-re, a (22.38)-beli T-t használva a következőket kapjuk:

$$
\begin{aligned}
s(x,y,t) &= s(x-1,y,t-1)\ . \\
a(x,y,t) &= a(x,y-1,t-1)\ . \\
b(x,y,t) &= \begin{cases} b(x-1,y,t-1) & : s(x-1,y,t-1) = 0 \\ c(x,y,t-1) & : s(x-1,y,t-1) = 1\ , \end{cases} \\
c(x,y,t) &= \begin{cases} c(x,y,t-1) \\ \quad + a(x,y-1,t-1) \\ \quad \cdot b(x-1,y,t-1) & : s(x-1,y,t-1) = 0 \\ b(x-1,y,t-1) & : s(x-1,y,t-1) = 1\ . \end{cases}
\end{aligned}
\tag{22.46}
$$

Az n iterációs változónak itt csupán a be/kiviteli séma szempontjából van jelentősége és a transzformáció számára egy rögzített értékre állíthatjuk. A hozzá tartozó cellaprogram, mely minden időszeletben lefut, a következő.

Cellaprogram

1 $\texttt{S} = \texttt{C}(-1,0)(0)$
2 $\texttt{A} = \texttt{C}(0,-1)$
3 $\texttt{B} = \texttt{C}(-1,0)(1:N)$
4 $\texttt{C}(1,0)(0) = \texttt{S}$
5 $\texttt{C}(0,1) = \texttt{A}$

```
6  if S = 1
7     C(1,0)(1 : N) = C
8     C = B
9     else C(1,0)(1 : N) = B
10          C = C + A · B
```

A csatornajelölések a cella lokális be/kimeneteire vonatkoznak. Alakjukat a csatornának a cella középpontjához viszonyított helyzetéből kapják. $\mathtt{C}(0,-1)$ a cella bal szélén helyezkedik el, $\mathtt{C}(0,1)$ a jobb szélen, $\mathtt{C}(-1,0)$ fent van, $\mathtt{C}(1,0)$ pedig lent. A csatornajelölés után megadható még egy bit-tartomány: $\mathtt{C}(-1,0)(0)$ kizárólag a csatorna 0. bitjét jelenti, $\mathtt{C}(-1,0)(1:N)$ ugyanannak a csatornának az 1-től N-ig terjedő bitjeit. Az A, B, ... jelölések a cella regisztereit jelentik.

A (22.12)-beli T-t (22.35)-re megfelelően alkalmazva a következőket kapjuk:

$$\begin{aligned}
&a(x,y,t) = a(x,y-1,t-1) && 1+x+y \le t \le x+y+N_3\ ,\\
&b(x,y,t) = b(x-1,y,t-1) && 1+x+y \le t \le x+y+N_3\ ,\\
&c(x,y,t) = c(x,y,t-1) && 1+x+y \le t \le x+y+N_3\\
&\quad + a(x,y-1,t-1)\cdot b(x-1,y,t-1)\ . &&\\
&b(x,y,t) = c(x,y,t-1) && x+y+1+N_3 \le t \le 2*x+y+N_3\ ,\\
&c(x,y,t) = b(x-1,y,t-1) && x+y+1+N_3 \le t \le 2*x+y\\
& && \qquad -1+N_3\ .
\end{aligned} \tag{22.47}$$

Szépen kidomborodnak tehát az osztott vezérlés előnyei. A (22.46)-ben leírt cellaprogram egy tetszőleges t időszeletre vonatkozik, nem kell tehát globális vezérlőjelekre reagáljon, nincs szüksége számlálóregiszterre, nincsenek számlálóműveletek és a helyi cellakoordinátákat sem kell kódolni.

Gyakorlatok

22.4-1. Adjuk meg a be/kimeneti sémákat két, lehető legszorosabban egymást követő számítás végrehajtására a (22.35) rendszer alapján, a 22.11., illetve 22.12. ábrán bemutatott szisztolikus rácsokra.

22.4-2. Hogyan kellene a 22.12. ábrán látható szisztolikus rácsot módosítani ahhoz, hogy hatékonyan támogassa a mátrixszorzatok kiszámítását $M_1 < N_1$ vagy $M_2 < N_2$ paraméterek esetén?

22.4-3. Hogy néz ki a cellaprogram a 22.3. ábrán látható szisztolikus rács esetében?

22.4-4. * Mekkora teljesítményt ér el a 22.3. ábrán látható szisztolikus rács N_1, N_2, N_3 konkrét értékeire? Hát általános N_1, N_2, N_3-ra?

22.4-5. * Módosítsuk a 22.1. ábrán látható szisztolikus rácsot úgy, hogy a stacionárius változók a számítás befejezése után jobb-alsó irányba (azaz a (i,j) cellától a $(i+1,j+1)$ cella felé) vezető járulékos kapcsolatokon keresztül legyenek kivezetve. Adjunk meg egy, a (22.35)-nek megfelelő értékadásmentes rendszert, mely leírást ad a hozzá tartozó viselkedésről. Hogyan néz ki a be/kimeneti séma? Milyen periódus érhető el?

22.5. Lineáris szisztolikus rácsok

A fenti alfejezetekben leírt megfontolások kétdimenziós szisztolikus rácsokra vonatkoznak. Azonban egydimenziós szisztolikus rácsokra is átültethetjük őket.

A két forma közötti lényeges különbség a szisztolikus rács *peremére* vonatkozik. Az egydimenziós szisztolikus rácsokat egyrészt felfoghatjuk úgy, mint amelyek kizárólag peremcellákból állnak, az adatbevitel a gazdagépről, illetve a kimenetek továbbítása a gazdagépnek minden további intézkedés nélkül lehetséges. Másrészt rendelkeznek egy teljes dimenzióval és egy csupán formális dimenzióval. A lineáris szisztolikus rács működési iránya menti kommunikációjában adott esetben hasonló kérdések merülnek fel, mint a 22.3.5. pontban. Végül, a lineáris szisztolikus rács pereme teljesen másként is definiálható, mégpedig úgy, mint ami csak a két végen elhelyezkedő cellából áll.

22.5.1. Mátrix és vektor szorzása

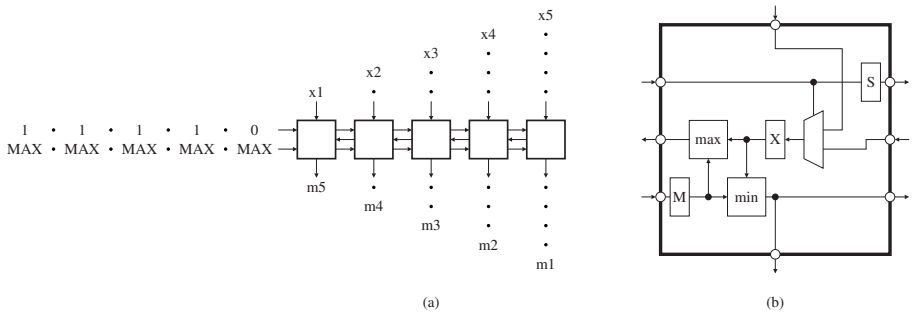
Ha a 22.1. ábrán az N_1 vagy N_2 feladat-paraméterek egyikének értéke 1, a mátrixszorzás átalakul mátrix és vektor közötti szorzássá: balról $N_1 = 1$-re, illetve jobbról $N_2 = 1$-re. A kétdimenziós szisztolikus rács ekkor egydimenzióssá ***fajul el.*** A szorzandó vektort, mint bemeneti adatfolyamot, a lineáris szisztolikus rács egyik végcelláján keresztül vezetjük be. Ezzel egy időben a mátrix elemei a rács hosszanti oldalán kerülnek be.

Akárcsak a teljes mátrixszorzás esetében, az eredmények stacionárius módon jönnek létre. Ezeket viszont kivezethetjük a rács hosszában az egyik végcelláján keresztül, vagy átadhatjuk közvetlenül a számítást végző cellákból a gazdagépnek. Ez különböző vezérlőmechanizmusokat, idősémákat és teljes futási időket eredményez.

Lehetséges lett volna-e az összes bemeneti adatot a végcellákon keresztül bevezetni? Semmiképp sem, ha a teljes futási időnek $\Theta(N)$-nek kell lennie. A beviteli mátrixnak $\Theta(N^2)$ eleme van, tehát $\Theta(N)$ elemet kell időszeletenként bevinni. Egy időszeleten belül a végcellákba bemenő adatok száma azonban korlátozott. Az esetünkben $\Theta(N)$ nagyságrendű ***be/kimeneti adatsebesség*** előre kizár bizonyos döntéseket.

22.5.2. Rendezés

Rendezéskor a feladat az, hogy egy teljesen rendezett G alaphalmazból vett $\{x_1, \ldots, x_N\}$ elemekből álló halmazt $\{m_i\}_{i=1,\ldots,N}$ növekvő sorrendbe állítsunk, vagyis hogy $m_i \leq m_k$ érvényes legyen minden $(i < k)$-ra. A feladatot a következőképpen fogalmazhatjuk meg értékadásmentes jelölést használva, ahol $MAX\ G$ maximumát jelöli:



22.14. ábra. „Buborékos rendezés" lineáris szisztolikus rács segítségével: **(a)** a rács felépítése be/kiviteli sémával; **(b)** Cellaszerkezet.

Bemeneti műveletek

$$\begin{aligned} x(i,j) &= x_i && 1 \le i \le N, j = 0\ , \\ m(i,j) &= MAX && 1 \le j \le N, i = j-1\ . \end{aligned}$$

Számítások

$$\begin{aligned} m(i,j) &= \min\{x(i,j-1), m(i-1,j)\} && 1 \le i \le N, 1 \le j \le i\ , \\ x(i,j) &= \max\{x(i,j-1), m(i-1,j)\} && 1 \le i \le N, 1 \le j \le i\ . \end{aligned} \tag{22.48}$$

Kimeneti műveletek

$$m(i,j) = m_j \qquad 1 \le j \le N, i = N\ .$$

Egy $u = (1,1)$ irány menti projekció segítségével a tér-idő-leképezésre

$$\begin{pmatrix} x \\ t \end{pmatrix} = \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} i \\ j \end{pmatrix} \tag{22.49}$$

a 22.14. ábrán látható egydimenziós szisztolikus rácsot kapjuk, mely a buborékrendezést valósítja meg.

Hasonlóképpen, az alábbi tér-idő-mátrix

$$T = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \tag{22.50}$$

a beszúró rendezést megvalósító lineáris szisztolikus rácshoz vezetne, míg végül a

$$T = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \tag{22.51}$$

tér-idő mátrix a kiválasztó rendezést valósítaná meg.

A rendezési feladatnál $\Theta(N)$ bemeneti adatunk, $\Theta(N)$ kimeneti adatunk, és $\Theta(N)$ időszeletünk van. Ez $\Theta(1)$ be/kimeneti adatsebességet eredményez. Ellentétben a 22.5.1. pontban bemutatott mátrix-vektor szorzással, a be/kimeneti adatsebesség a kommunikációt elvileg itt még kizárólag csak a lineáris szisztolikus rács végpontjain levő cellákon keresztül engedi meg.

Mindazonáltal a rendezés mindhárom leírt változatában a bemenetek az összes cellán keresztül történnek: a buborékos rendezésnél csak a rendezni való elemek, a kiválasztásos rendezésnél ehhez hozzájönnek még a konstans MAX értékek, a beszúrásos rendezésnél csak a konstans értékek. Az utóbbiakat ugyan nem kötelező bemeneti adatként megadni, hanem létrehozhatók közvetlenül a cellákban, vagy kiolvashatók egy csak olvasható (ROM) memóriából is.

Mindhárom változat cellavezérlést igényel: a beszúró-, illetve kiválasztásos rendezés azért, mert stacionárius változói vannak, a buborékos rendezés azért, mert a bemeneti adatok és a kiszámított értékek feldolgozása között átkapcsolásra van szükség.

22.5.3. Lineáris egyenletrendszer alsó háromszögmátrixszal

A (22.52)-beli képletek egy lokalizált algoritmust írnak le az $A \cdot x = b$ lineáris egyenletrendszer megoldására, ahol az $(N \times N)$-es A mátrix egy alsó háromszögmátrix.

Bemeneti műveletek

$$\begin{aligned} a(i,j) &= a_{i,j+1} && 1 \le i \le N, 0 \le j \le i-1\ , \\ u(i,j) &= b_i && 1 \le i \le N, j = 0\ . \end{aligned}$$

Számítások

$$\begin{aligned} u(i,j) &= u(i,j-1) - a(i,j-1) \cdot x(i-1,j) && 2 \le i \le N, 1 \le j \le i-1\ , \\ x(i,j) &= u(i,j-1)/a(i,j-1) && 1 \le i \le N, j = i\ , \\ x(i,j) &= x(i-1,j) && 2 \le i \le N-1, 1 \le j \le i-1\ . \end{aligned}$$

Kimeneti műveletek

$$x_i = x(i,j) \qquad 1 \le i \le N, j = i\ . \tag{22.52}$$

Az összes eddig tanulmányozott példában, a másolási műveletektől eltekintve, az értéktartomány egy hordozópontjára ugyanazok a számítási műveletek vonatkoztak: szorzás és összeadás egymás utáni végrehajtása a szorzásalgoritmusok esetében, valamint minimum és maximum párhuzamos végrehajtása a rendezési algoritmusok esetében. A (22.52) rendszerben ezzel el-

lentétben egyes hordozópontokra szorzás és kivonás, míg más hordozópontok esetében csak osztás történik.

A (22.52) lineáris szisztolikus rácsra való levetítése során, a választott projekció-iránytól függően ugyanolyan, illetve különböző cellaműködést kapunk. Az $u = (1, 1)$ projekció-iránnyal (és csakis ezzel) egyetlenegy osztóval rendelkező cellát kapunk, az összes többi szorzó- és kivonó egységgel rendelkezik. Az $u = (1, 0)$ vagy $u = (0, 1)$ mentén történő projekció esetén csupa egyforma cellát kapunk, melyek osztóval, szorzóval és kivonóval is rendelkeznek. Az $u = (1, -1)$ projekció-irány egy, három különböző cellatípussal rendelkező, lineáris szisztolikus rácsot eredményez: mindkét végen levő cellában csak osztóra van szükség. Az összes többi cella kap egy szorzó- és kivonó egységet, mindamellett egy osztóval rendelkező és egy osztó nélküli cella váltja egymást. A projekció egy bizonyos módja egy szisztolikus rácsban ***inhomogenitáshoz*** vezethet (ami lehet hasznos – vagy sem).

Gyakorlatok

22.5-1. Adjunk meg a mátrix és vektor szorzásának a 22.5.1. pontban leírt változataira (az eredmények megadása egy végcellán keresztül, illetve az összes cellán keresztül) egy alkalmas rácsfelépítést be/kiviteli sémával, cellaszerkezettel, valamint vezérlési mechanizmussal együtt.

22.5-2. Tanulmányozzuk további projekciós irányok hatását a (22.52) rendszerre.

22.5-3. Adjuk meg a 22.5.2. pontban leírt beszúró, illetve kiválasztásos rendező eljárásokhoz a hozzájuk tartozó szisztolikus rácsokat – beleértve a cellaszerkezetet is.

22.5-4. * Hogyan működtethető a 22.14. ábrán látható, buborékrendezésre használt szisztolikus rács akár vezérlés nélkül is, a bemeneti adatfolyam észszerű megformálásával?

22.5-5. * Mi a szerepe a MAX érték használatának a (22.48) rendszerben? Hogyan lehetne (22.48)-et kifejezni a e konstans érték használata nélkül? Milyen következményei lennének ennek a leírt szisztolikus rácsokra nézve.

Feladatok

22-1 Szalagmátrix algoritmusok

A 22.1. és 22.2. alfejezetekben, valamint a 22.5.1. és 22.5.3. pontokban mindig ***sűrű*** mátrixokból indultunk ki, azaz minden a_{ij} mátrixelem nullától különböző érték (az alsó háromszögmátrix esetében a főátló feletti elemek értéke ugyan mind nulla, de ezek nem képeznek bemenetet az említett algoritmus számára).

Ezzel szemben a gyakorlatban gyakran találkozunk ***szalagmátrixokkal.*** Ezeknél a főátló körüli keskeny sávot kivéve, a legtöbb átló nulla értékekkel van feltöltve. Formálisan fennáll tehát, hogy $a_{ij} = 0$ minden i, j-re, melyre $i - j \geq K$ vagy $j - i \geq L$, ahol K és L pozitív egész számok. Tehát a ***sávszélesség,*** vagyis azon átlók száma, amelyekben a nullától különböző elemek megengedettek, $K + L - 1$.

Feltevődik tehát a kérdés, hogy egy vagy több bemeneti mátrix sávszerkezetét ki lehet-e használni a szisztolikus számítások optimalizálására. Egyrészt fennáll annak a lehetősége, hogy elhagyjuk azokat a cellákat, melyek soha nem végeznek hasznos munkát. Egy másik optimalizálási lehetőség lehetne a be/kimeneti adatfolyam lerövidítése, a teljes futási idő lerövidítése vagy a teljesítmény növelése.

Tanulmányozzuk, hogyan optimalizálhatók a fejezetben bemutatott szisztolikus rácsok ebből a szempontból.

Megjegyzések a fejezethez

A szisztolikus rács fogalmát Kung és Leiserson vezette be, e területen úttörőnek számító cikkükben [164].

Karp, Miller és Winograd az egyenletes rekurzív egyenletek területén végeztek úttörő munkát [150].

Rao doktori értekezése [246] és Quinton munkái [245] is lényeges ötletekkel járultak hozzá a szisztolikus rácsok módszeres tervezéséhez.

Teich és Thiele [300] cikkükben rámutatnak, hogy a cellavezérlés formálisan hasonló módszerekkel vezethető le, mint az általános rácsfelépítés és a normális cellafunkcionalitás.

Darte, Robert és Vivien modern könyve [61] a fordítóprogramok kifinomult módszereit kapcsolja össze a szisztolikus rácsokkal, és alaposan foglalkozik az adatfüggőségek vizsgálatával.

A szalagmátrixokra vonatkozó feladat Kung és Leiserson cikkéből [164] származik.

A szisztolikus rendszerek területén a legátfogóbb áttekintést mind a mai napig a [320] monográfia nyújtja.

Minden szisztolikus rács modellezhető sejtautomataként. Egy cella regiszterei képezik a cella állapotát. Szükség van tehát egy faktorizált állapottér használatára. Amennyiben a szisztolikus rács különböző típusú cellákkal rendelkezik (például változó cella-funkcionalitás vagy pozíciófüggő cellavezérlés révén), ez az állapottér egy újabb komponensével írható le.

Minden szisztolikus algoritmus felfogható olyan PRAM-algoritmusként, melynek viselkedése időben állandó. Ekkor egy szisztolikus cella minden reg-

iszterét egy PRAM-memóriacella valósítja meg, mely kizárólag ezt a célt szolgálja. Mivel minden időszeletben pontosan egy szisztolikus cella olvas ebből a regiszterből, és ugyanakkor pontosan egy szisztolikus cella ír ebbe a regiszterbe, megfelelő az EREW-PRAM-modell.

A szisztolikus rendszerek a Lynch [187] által leírt szinkronizált hálózatok speciális esetei. A futási idő azonos a két rendszerben. A szisztolikus rendszerekben az üzenetek számát nem szokás vizsgálni. A szisztolikus rendszerekben gyakran megkívánt – peremen át történő be/kivitel szinkronizált hálózatokkal jól modellezhető. A hibák fogalma a szisztolikus rendszerek vizsgálatához nem szükséges.

Részletesen foglalkozik a szisztolikus rendszerekkel Sima, Fountain és Kacsuk könyve, amely magyarul is megjelent [269].

23. Versenyképességi elemzés

A gyakorlatban gyakran fordulnak elő olyan optimalizálási feladatok, ahol a bemenetet csak részenként ismerjük meg és a döntéseinket a már megkapott információk alapján, a további adatok ismerete nélkül kell meghoznunk.

Ilyen esetekben ***on-line feladatról*** beszélünk. Az on-line algoritmusok elméletének igen sok alkalmazása van a számítástudomány, a közgazdaságtan és az operációkutatás különböző területein.

Az on-line algoritmusok elméletéhez kapcsolódó első eredmények az 1970-es évekből származnak, majd a 90-es évek elejétől kezdve egyre több kutató kezdett el az on-line algoritmusok területéhez kapcsolódó feladatokkal foglalkozni. Számos részterület alakult ki és a legfontosabb, algoritmusokkal foglalkozó konferenciákon napjainkban is rendszeresen ismertetnek új eredményeket ezen témakörből. Jelen fejezetnek nem célja a témakör részletes áttekintése, ezen keretek között terjedelmi okokból ez nem is lenne lehetséges. Célunk néhány részterület részletesebb ismertetésén keresztül a legfontosabb algoritmustervezési technikák és bizonyítási módszerek bemutatása.

A következő alfejezetben az on-line algoritmusok elemzésénél használt alapvető fogalmakat definiáljuk. A legfontosabb definíciók ismertetését követően az egyik legismertebb on-line feladatot – a k-szerver feladatot – és néhány kapcsolódó eredményt ismertetünk. Ezt követően egy új területet mutatunk be, a számítógépes hálózatok vizsgálatához kapcsolódó on-line algoritmusok területét. Majd a ládapakolási feladatot és annak többdimenziós változatait ismertetjük. Végül a fejezet utolsó részében az ütemezéssel foglalkozunk és a területre vonatkozó alapvető eredményeket tekintjük át.

23.1. Fogalmak, definíciók

Mivel egy on-line algoritmusnak részenként kell meghozni a döntéseit a teljes bemenet ismerete nélkül, ezért egy ilyen algoritmustól nem várhatjuk el, hogy a teljes információval rendelkező algoritmusok által előállítható optimális megoldást szolgáltassa. Azon algoritmusokat, amelyek ismerik a teljes bemenetet, ***off-line algoritmusoknak*** nevezzük.

A többi algoritmushoz hasonlóan az on-line algoritmusok hatékonyságának vizsgálatára két alapvető módszert használnak. Az egyik lehetőség az

átlagos eset elemzése.

Ezen megközelítés hátránya, hogy általában nincs információnk arról, hogy a lehetséges bemenetek milyen valószínűségi eloszlást követnek. Mi ebben a fejezetben az átlagos eset elemzésével nem foglalkozunk.

A másik megközelítés egy legrosszabb eset korlát elemzés, amelyet ***versenyképességi elemzésnek*** nevezünk. Ebben az esetben az on-line algoritmus által kapott megoldás célfüggvényértékét hasonlítjuk össze az optimális off-line célfüggvényértékkel. A továbbiakban feltételezzük, hogy a célfüggvény pozitív. Mivel ebben a fejezetben ezt az elemzést használjuk, ezért az alábbiakban pontosan definiáljuk az alapvető fogalmakat.

Egy on-line minimalizálási feladat esetén egy on-line algoritmust ***C-versenyképesnek*** nevezünk, ha tetszőleges bemenetre teljesül, hogy az algoritmus által kapott megoldás költsége nem nagyobb, mint C-szer az optimális off-line költség. Egy ***algoritmus versenyképességi hányadosa*** a legkisebb olyan C szám, amelyre az algoritmus C-versenyképes.

A továbbiakban egy tetszőleges ALG on-line algoritmusra az I bemeneten felvett célfüggvényértéket $\mathrm{ALG}(I)$-vel jelöljük. Az I bemeneten felvett optimális off-line célfüggvényértéket $\mathrm{OPT}(I)$-vel jelöljük. Ezt a jelölésrendszert használva a versenyképességet minimalizálási feladatokra a következőképpen definiálhatjuk.

Az ALG algoritmus C-versenyképes, ha $\mathrm{ALG}(I) \leq C \cdot \mathrm{OPT}(I)$ teljesül minden I bemenet esetén.

Szokás használni a versenyképesség további két változatát. Egy minimalizálási feladat esetén az ALG algoritmus ***enyhén C-versenyképes,*** ha van olyan B konstans, hogy $\mathrm{ALG}(I) \leq C \cdot \mathrm{OPT}(I) + B$ teljesül minden I bemenet esetén.

Egy ***algoritmus enyhe versenyképességi hányadosa*** a legkisebb olyan C szám, amelyre az algoritmus enyhén C-versenyképes.

A versenyképességi hányados egy további változata az aszimptotikus versenyképességi hányados. Minimalizálási feladatok esetén az ALG algoritmus ***aszimptotikus versenyképességi hányadosa*** ($R^{\infty}_{\mathrm{ALG}}$) a következő képletekkel definiálható:

$$R^{n}_{\mathrm{ALG}} = \sup\Big\{ \frac{\mathrm{ALG}(I)}{\mathrm{OPT}(I)} \;\Big|\; \mathrm{OPT}(I) = n \Big\} \,,$$

$$R^{\infty}_{\mathrm{ALG}} = \limsup_{n \to \infty} R^{n}_{\mathrm{ALG}} \,.$$

Egy algoritmust ***aszimptotikusan C-versenyképesnek*** nevezünk, ha az aszimptotikus versenyképességi hányadosa nem nagyobb, mint C.

Megjegyezzük, hogy R^{n}_{ALG} és $R^{\infty}_{\mathrm{ALG}}$ a harmadik kötet 34.3. alfejezetében

bevezetett *hibafüggvény,* illetve *aszimptotikus hiba* speciális esetei (A = ALG, B = OPT). hányados fő tulajdonsága az, hogy azt vizsgálja, miként viselkedik az algoritmus akkor, ha a bemenet mérete végtelenhez tart. Ez azt jelenti, hogy az algoritmus – kis méretű bemenetekre mutatott – viselkedése nem befolyásolja az aszimptotikus hányadost.

A fentiekben a minimalizálási feladatokra definiáltuk a versenyképességi elemzés fogalmait. A definíciók hasonlóan értelmezhetők maximalizálási feladatok esetén is. Ekkor az ALG algoritmus C-versenyképes, ha $\mathrm{ALG}(I) \geq C \cdot \mathrm{OPT}(I)$ teljesül minden I bemenet esetén, illetve enyhén C-versenyképes, ha valamely B konstans mellett $\mathrm{ALG}(I) \geq C \cdot \mathrm{OPT}(I) + B$ teljesül minden I bemenetre. Az aszimptotikus hányadost maximalizálási feladatokra a következőképpen definiálhatjuk:

$$R^n_{\mathrm{ALG}} = \inf\Big\{\frac{\mathrm{ALG}(I)}{\mathrm{OPT}(I)} \;\Big|\; \mathrm{OPT}(I) = n\Big\}\,,$$

$$R^\infty_{\mathrm{ALG}} = \liminf_{n\to\infty} R^n_{\mathrm{ALG}}\,.$$

Ekkor egy algoritmust ***aszimptotikusan C-versenyképesnek*** nevezünk, ha az aszimptotikus versenyképességi hányadosa nem kisebb, mint C.

Számos tudományos dolgozat vizsgál ***véletlenített on-line algoritmusokat,*** ebben az esetben az algoritmus által kapott célfüggvényérték egy valószínűségi változó és a versenyképességi hányados definíciójában ezen valószínűségi változó várható értéke szerepel. Mivel a fejezetben csak determinisztikus on-line algoritmusokkal fogunk foglalkozni, ezért a véletlenített algoritmusokra vonatkozó fogalmakat nem részletezzük.

23.2. A k-szerver feladat

Az egyik legismertebb on-line feladat a k-szerver feladat. A feladat általános definíciójának megadásához szükség van a metrikus tér fogalmára. Egy (M, d) párost, ahol M a metrikus tér pontjait tartalmazza, d pedig az $M \times M$ halmazon értelmezett távolságfüggvény, metrikus térnek nevezünk, ha a távolságfüggvényre teljesülnek az alábbi tulajdonságok:

- $d(x, y) \geq 0$ minden $x, y \in M$ esetén,
- $d(x, y) = d(y, x)$ minden $x, y \in M$ esetén,
- $d(x, y) + d(y, z) \geq d(x, z)$ minden $x, y, z \in M$ esetén,
- $d(x, y) = 0$ akkor és csak akkor teljesül, ha $x = y$.

A k-szerver feladatban adott egy metrikus tér és van k darab szerverünk, amelyek a térben mozoghatnak. A tér pontjaiból álló kérések egy listáját

kell kiszolgálni azáltal, hogy a megfelelő kérések helyére odaküldünk egy-egy szervert.

A feladat on-line, ami azt jelenti, hogy a kéréseket egyenként kapjuk meg és az egyes kéréseket a további kérések ismerete nélkül azok érkezése előtt kell kiszolgálnunk. A cél a szerverek által megtett össztávolság minimalizálása. Ezen modellnek és speciális eseteinek számos alkalmazása van. A továbbiakban azt a metrikus tér pontjaiból álló multihalmazt, amely megadja mely pontokban helyezkednek el a szerverek (azért kell multihalmazokat használnunk, mert egy pontban több szerver is lehet) a ***szerverek konfigurációjának*** nevezzük.

Az első fontos eredményeket a k-szerver feladatra Manasse McGeoch és Sleator érték el. Az általuk javasolt első eljárás a következő EGYENSÚLY algoritmus, amelyet a továbbiakban ES-sel jelölünk. Az eljárás során a szerverek mindig különböző pontokban helyezkednek el. Az algoritmus futása során minden szerverre számon tartja, hogy az aktuális időpontig összesen mekkora távolságot tett meg. Jelölje rendre $s_1, \ldots, s_k$ a szervereket és a pontokat is, ahol a szerverek elhelyezkednek. Továbbá jelölje $D_1, \ldots, D_k$ rendre a szerverek által az adott időpontig megtett összutat. Ekkor, amennyiben egy P pontban megjelenik egy kérés, akkor az ES algoritmus azt az i szervert választja a kérés kiszolgálására, ahol a $D_i + d(s_i, P)$ érték minimális, azaz az algoritmus igyekszik az egyes szerverek által megtett utakat egyensúlyban tartani. Tehát az algoritmus számon tartja a szerverek $S = \{s_1, \ldots, s_k\}$ szerver konfigurációját és a szerverekhez rendelt távolságokat, amelyek kezdeti értékei $D_1 = \cdots = D_k = 0$. Ezt követően az algoritmus egy $I = P_1, \ldots, P_n$ pontsorozatra a következő pszeudokóddal írható le:

ES(I)

```
1  for j = 1 to n
2      i = argmin{D_i + d(s_i, P_j)}
3      szolgáljuk ki a kérést az i-edik szerverrel
4      D_i = D_i + d(s_i, P_j)
5      s_i = P_j
```

23.1. példa. Tekintsük a kétdimenziós euklideszi teret, mint metrikus teret. A pontok (x, y) valós számpárokból állnak, két (a, b) és (c, d) pontnak a távolsága $\sqrt{(a-c)^2 + (b-d)^2}$. Legyen két szerverünk, kezdetben a $(0, 0)$ és $(1, 1)$ pontokban. Tehát a kiindulási állapotban $D_1 = D_2 = 0$, $s_1 = (0, 0)$, $s_2 = (1, 1)$. Az első kérés legyen az $(1, 4)$ pontban. Ekkor $D_1 + d((0, 0), (1, 4)) = \sqrt{17} > D_2 + d((1, 1), (1, 4)) = 3$, így a második szervert használjuk és a kérés kiszolgálása után $D_1 = 0, D_2 = 3$, $s_1 = (0, 0)$, $s_2 = (1, 4)$ teljesül. Legyen a második kérés $(2, 4)$, ekkor $D_1 + d((0, 0), (2, 4)) =$

$\sqrt{20} > D_2 + d((1,4),(2,4)) = 3+1 = 4$, így ismét a második szervert használjuk, és a kérés kiszolgálása után $D_1 = 0, D_2 = 4$, $s_1 = (0,0)$, $s_2 = (2,4)$ teljesül. A harmadik kérés legyen ismét az $(1,4)$ pontban, ekkor $D_1 + d((0,0),(1,4)) = \sqrt{17} < D_2 + d((2,4),(1,4)) = 4+1 = 5$, így az első szervert használjuk és a kérés kiszolgálása után $D_1 = \sqrt{17}, D_2 = 4$, $s_1 = (1,4)$, $s_2 = (2,4)$ teljesül.

Az algoritmus hatékony speciális terek esetén, miként ezt a következő állítás mutatja. A tétel bizonyítására vonatkozó hivatkozások megtalálhatók a fejezet végén szereplő megjegyzésekben.

23.1. tétel. *Amennyiben a metrikus tér $k+1$ pontot tartalmaz, akkor az* Egyensúly *algoritmus enyhén k-versenyképes.*

Az alábbi állítás mutatja, hogy a k-szerver feladatra általában nem adható meg k-versenyképesnél jobb algoritmus.

23.2. tétel. *Nincs olyan – legalább $k+1$ pontból álló – metrikus tér, ahol megadható olyan on-line algoritmus, amelynek kisebb a versenyképességi hányadosa, mint k.*

Bizonyítás. Tekintsünk egy tetszőleges – legalább $k+1$ pontból álló – teret, és egy tetszőleges on-line algoritmust. Jelölje az algoritmust ONL, a pontokat, ahol kezdetben ONL szerverei állnak, $P_1, P_2, \ldots, P_k$, a térnek egy további pontját jelölje P_{k+1}. Vegyük kéréseknek egy hosszú $I = Q_1, \ldots, Q_n$ sorozatát, amelyet úgy kapunk, hogy a következő kérés mindig a $P_1, P_2, \ldots, P_{k+1}$ pontok közül abban a pontban keletkezik, ahol az ONL algoritmusnak nem tartózkodik szervere.

Vizsgáljuk elsőként az $\mathrm{ONL}(I)$ költséget. Mivel a Q_j pont kiszolgálása után a Q_{j+1} pont lesz szabad, ezért a Q_j pontot mindig a Q_{j+1} pontban álló szerver szolgálja ki, így a kiszolgálás költsége $d(Q_j, Q_{j+1})$. Következésképpen

$$\mathrm{ONL}(I) = \sum_{j=1}^{n} d(Q_j, Q_{j+1}) \ ,$$

ahol Q_{n+1} azt a pontot jelöli, ahol az $(n+1)$-edik kérés lenne, azaz azt a pontot, amelyről kiszolgáltuk az n-edik kérést.

Most vizsgáljuk az $\mathrm{OPT}(I)$ költséget. Az optimális off-line algoritmus meghatározása helyett definiálunk k darab off-line algoritmust, és ezek költségeinek az átlagát használjuk. Mivel mindegyik algoritmus költsége legalább akkora, mint a minimális költség, ezért a költségek átlaga is felső korlátja lesz az optimális költségnek.

Definiáljunk k darab off-line algoritmust, jelölje őket $\mathrm{OFF}_1, \ldots, \mathrm{OFF}_k$. Tegyük fel, hogy a kiindulási állapotban az OFF_j algoritmus szerverei a

$P_1, P_2, \ldots, P_{k+1}$ pontok közül a P_j pontot szabadon hagyják, és a halmaz további pontjainak mindegyikén egy szerver helyezkedik el. Ez a kiindulási állapot elérhető egy C_j konstans extra költség felhasználásával.

A kérések kiszolgálása a következőképpen történik. Ha a Q_i ponton van az OFF_j algoritmusnak szervere, akkor nem mozgat egyetlen szervert sem, ha nincs, akkor a Q_{i-1} ponton levő szervert használja. Az algoritmusok jól definiáltak, hiszen ha nincs szerver a Q_i ponton, akkor ezen pont kivételével a $P_1, P_2, \ldots, P_{k+1}$ pontok mindegyikén, így Q_{i-1}-en is van szerver. Továbbá a $Q_1 = P_{k+1}$ ponton az OFF_j algoritmusok mindegyikének áll szervere a kezdeti konfigurációban.

Vegyük észre, hogy az $\mathrm{OFF}_1, \ldots, \mathrm{OFF}_k$ algoritmusok szerverei rendre különböző konfigurációkban vannak. Kezdetben ez az állítás a definíció alapján igaz. Utána a tulajdonság azért marad fenn, mert amely algoritmusok nem mozdítanak szervert a kérés kiszolgálására, azoknak a szerver konfigurációja nem változik. Amely algoritmusok mozgatnak szervert, azok a Q_{i-1} pontról viszik el a szervert, amely pont az előző kérés volt, így ott minden algoritmusnak van szervere. Következésképp ezen algoritmusok szerver konfigurációja nem állítódhat azonos pozícióba olyan algoritmus szerver konfigurációjával, amely nem mozdított szervert. Másrészt, ha több algoritmus is mozgatná Q_{i-1}-ről Q_i-be a szervert, azok szerver konfigurációja se válhat azonossá, hisz a mozgatást megelőzőleg különböző volt.

Következésképp egy Q_i kérés esetén minden OFF_j algoritmusra más a szerver konfiguráció. Továbbá minden konfigurációnak tartalmaznia kell Q_{i-1}-et, tehát pontosan egy olyan OFF_j algoritmus van, amelynek nincsen szervere a Q_i ponton. Tehát a Q_i kérés kiszolgálásának a költsége az OFF_j algoritmusok egyikénél $d(Q_{i-1}, Q_i)$, a többi algoritmus esetén pedig 0.

Következésképpen

$$\sum_{j=1}^{k} \mathrm{OFF}_j(I) = C + \sum_{i=2}^{n} d(Q_i, Q_{i-1}) \ ,$$

ahol $C = \sum_{j=1}^{k} C_j$ egy a bemeneti sorozattól független konstans, amely az off-line algoritmusok kezdeti konfigurációinak beállításának költsége.

Másrészt az off-line optimális algoritmus költsége nem lehet nagyobb semelyik off-line algoritmus költségénél sem, így $k \cdot \mathrm{OPT}(I) \leq \sum_{j=1}^{k} \mathrm{OFF}_j(I)$. Következésképpen

$$k \cdot \mathrm{OPT}(I) \leq C + \sum_{i=2}^{n} d(Q_i, Q_{i-1}) \leq C + \mathrm{ONL}(I) \ ,$$

amely egyenlőtlenségből következik, hogy az ONL algoritmus versenyképességi hányadosa nem lehet kisebb, mint k, hiszen a bemeneti sorozat

hosszúságának növelésével a $\mathrm{OPT}(I)$ érték tetszőlegesen nagy lehet. ■

A kérdés felvetése nagy érdeklődést keltett, az elkövetkező néhány évben számos eredmény született. Az általános esetre konstans-versenyképes algoritmust ($O(2^k)$-versenyképes algoritmust) elsőként Fiat, Rabani és Ravid fejlesztettek ki. Ezt követően hosszú időn át nem sikerült lényegesen csökkenteni a felső és az alsó korlát közötti rést. Az áttörést Koutsoupias és Papadimitriou eredménye hozta meg, sikerült a feladat megoldására a Chrobak és Larmore által javasolt munkafüggvényen alapuló algoritmust elemezniük és igazolniuk, hogy az algoritmus $(2k-1)$-versenyképes. Nem sikerült meghatározniuk az algoritmus pontos versenyképességi hányadosát, bár általánosan elfogadott sejtés, hogy az algoritmus valójában k-versenyképes. Ezen versenyképességi hányados pontos meghatározása, illetve egy k-versenyképes algoritmus kifejlesztése azóta is az on-line algoritmusok elméletének legismertebb és sokak által legfontosabbnak tartott nyílt problémája. Az alábbiakban ismertetjük a munkafüggvény algoritmust alapötletét.

Legyen A_0 az on-line szerverek kezdeti konfigurációja. Ekkor a t-edik kérés utáni, X multihalmazra vonatkozó ***munkafüggvény*** $w_t(X)$ az a minimális költség, amellyel kiszolgálható az első t kérés az A_0 konfigurációból kiindulva úgy, hogy a szerverek az X konfigurációba kerüljenek a kiszolgálás végén. A munkafüggvény értékeinek meghatározása egy off-line optimalizálási feladat, amely dinamikus programozási módszerrel megoldható. A MUNKAFÜGGVÉNY algoritmus a munkafüggvényt használja. Legyen A_{t-1} a szervereknek a konfigurációja közvetlenül a t-edik kérés érkezése előtt. Ekkor a MUNKAFÜGGVÉNY algoritmus azzal az s szerverrel szolgálja ki az R_t pontban megjelent kérést, amelynek a P helyére a $w_{t-1}(A_{t-1} \setminus \{P\} \cup \{R_t\}) + d(P, R_t)$ érték a minimális.

Az algoritmusra a bemenet pontoknak egy $I = P_1, \ldots, P_k$ sorozata, amely a kérések listája, és a szerverek kezdeti S konfigurációja, ami szintén pontoknak egy halmaza, az algoritmus azt munkafüggvényt használja, amelyre a kiindulási A_0 halmaz S kezdeti értéke. Az algoritmust a következő pszeudokóddal adhatjuk meg.

MUNKAFÜGGVÉNY(I, S)

```
1  for j = 1 to n
2      i = argmin_s w_{j-1}(S \ {s} ∪ {P_j}) + d(s, P_j)
3      a kérést az i-edik szerverrel szolgáljuk ki
4      s_i = P_j
```

23.2. példa. Tekintsük azt a metrikus teret, amelyben három pont van, A, B és C, a távolságok $d(A,B) = 1$, $d(B,C) = 2$, $d(A,C) = 3$. A kezdeti szerver konfiguráció legyen $\{A, B\}$. Ekkor a kezdeti munkafüggvények $w_0(\{A,A\}) = 1$, $w_0(\{A,B\}) = 0$,

$w_0(\{A,C\}) = 2$, $w_0(\{B,B\}) = 1$, $w_0(\{B,C\}) = 3$, $w_0(\{C,C\}) = 4$, Legyen az első kérés a C pontban. Ekkor $w_0(\{A,B\} \setminus \{A\} \cup \{C\}) + d(A,C) = 3+3 = 6$ és $w_0(\{A,B\} \setminus \{B\} \cup \{C\}) + d(B,C) = 2+2 = 4$, így a MUNKAFÜGGVÉNY a B pontban levő szervert küldi a kérés kiszolgálására.

Az algoritmusra teljesül a következő állítás.

23.3. tétel. *A* MUNKAFÜGGVÉNY *algoritmus enyhén* $(2k-1)$*-versenyképes.*

Az általános feladat vizsgálata mellett a feladat speciális eseteit számos dolgozatban vizsgálták. Amennyiben bármely két pont távolsága 1, akkor a lapozási feladat on-line változatához jutunk, amely a számítógépek memóriakezelését modellezi. Egy másik vizsgált speciális tér az egyenes. Az egyenes pontjait a valós számoknak feleltetjük meg, és két pontnak, a-nak és b-nek a távolsága $|a-b|$. Erre az esetre, ahol a metrikus tér egy egyenes, Chrobak és Larmore kifejlesztett egy k-versenyképes algoritmust, amelyet DUPLA-LEFEDŐ algoritmusnak nevezünk. Az algoritmus a P kérést a P-hez legközelebb eső s szerverrel szolgálja ki, és amennyiben vannak szerverek P-nek az s-sel átellenes oldalán is, akkor azon szerverek közül a P-hez legközelebbit $d(s,P)$ egységnyit mozdítja P felé. A továbbiakban a DUPLA-LEFEDŐ algoritmust DL-lel jelöljük. Az algoritmusra a bemenet pontok (valós számok) egy $I = P_1, \ldots, P_k$ sorozata, amely a kérések listája, és a szerverek kezdeti $S = (s_1, \ldots, s_k)$ konfigurációja, ami szintén pontok (valós számok) egy halmaza. Az algoritmust a következő pszeudokóddal adhatjuk meg.

DL(I,S)

```
1  for j = 1 to n
2      i = argmin_l d(P_j, s_l)
3  if s_i == min_l s_l vagy s_i == max_l s_l
4     a kérést az i-edik szerverrel szolgáljuk ki
5     s_i = P_j
6  else if s_i ≤ P_j
7          m == argmin_{l:s_l>P_j} d(s_l, P_j)
8          a kérést az i-edik szerverrel szolgáljuk ki
9          s_m = s_m − d(s_i, P_j)
10         s_i = P_j
```

11 **else if** $s_i > P_j$
12 $n = \text{argmin}_{l:s_l<P_j} d(s_l, P_j)$
13 a kérést az i-edik szerverrel szolgáljuk ki
14 $s_n = s_n + d(s_i, P_j)$
15 $s_i = P_j$

23.3. példa. Tegyük fel, hogy három szerverünk van (s_1, s_2, s_3), amelyek az egyenes 0, 1, 2 pontjaiban helyezkednek el. Amennyiben a következő kérés a 4 pontban jelenik meg, akkor DL a legközelebbi s_3 szervert küldi a kérés kiszolgálására. A többi szerver helye nem változik, a költség 2 és a kérés kiszolgálása után a szerverek a $0, 1, 4$ pontokban lesznek. Amennyiben ezt követően a következő kérés a 2 pontban jelenik meg, akkor DL a legközelebbi s_2 szervert küldi a kérés kiszolgálására, de mivel a kérés másik oldalán is van szerver, ezért s_3 is megtesz egy egységnyi utat a kérés felé, így a költség 2 és a kérés kiszolgálása után a szerverek a 0, 2, 3 pontokban lesznek.

A DL algoritmusra teljesül a következő állítás, amelynek a bizonyításához az on-line algoritmusok elemzése során gyakran használt potenciálfüggvény technikát alkalmazzuk.

23.4. tétel. *Ha a metrikus tér egy egyenes, akkor a* Dupla-lefedő *algoritmus enyhén k-versenyképes.*

Bizonyítás. Vegyük kérések egy tetszőleges sorozatát és jelölje ezt az bemenetet I. Az eljárás elemzése során feltételezzük, hogy párhuzamosan fut a DL algoritmus és egy optimális off-line algoritmus. Szintén feltesszük, hogy minden kérést elsőként az off-line algoritmus szolgál ki, utána pedig az on-line algoritmus. Az on-line algoritmus szervereit és egyben a szerverek pozícióit (amelyek valós számok az egyenesen) $s_1, \ldots, s_k$ jelöli, az optimális off-line algoritmus szervereit és egyben a szerverek pozícióit $x_1, \ldots, x_k$ jelöli. Mivel a szerverek rendszeres átjelölésével ez elérhető, feltételezzük, hogy $s_1 \leq s_2 \leq \cdots \leq s_k$ és $x_1 \leq x_2 \leq \cdots \leq x_k$ mindig teljesül.

A tétel állítását a potenciálfüggvény technikájával igazoljuk. A potenciálfüggvény a szerverek aktuális pozíciójához rendel egy értéket, az on-line és az off-line költségeket a potenciálfüggvény változásainak alapján hasonlítjuk össze. Legyen a potenciálfüggvény

$$\Phi = k \sum_{i=1}^{k} |x_i - s_i| + \sum_{i<j} (s_j - s_i) \ .$$

Az alábbiakban igazoljuk, hogy a potenciálfüggvényre teljesülnek a következő állítások.

- Amíg OPT szolgálja ki a kérést, addig a potenciálfüggvény növekedése legfeljebb k-szor az OPT szerverei által megtett távolság.
- Amíg DL szolgálja ki a kérést, addig Φ legalább annyival csökken, mint amennyi a kérés kiszolgálásának költsége.

Amennyiben a fenti tulajdonságok teljesülnek, akkor a tétel állítása következik, hiszen ebben az esetben adódik, hogy $\Phi_v - \Phi_0 \leq k \cdot \mathrm{OPT}(I) - \mathrm{DL}(I)$, ahol Φ_v és Φ_0 a potenciálfüggvény kezdeti és végső értékei. Mivel a potenciálfüggvény nemnegatív, ezért adódik, hogy $\mathrm{DL}(I) \leq k\mathrm{OPT}(I) + \Phi_0$, azaz azt kapjuk, hogy a DL algoritmus enyhén k-versenyképes.

Most igazoljuk a potenciálfüggvény tulajdonságait.

Elsőként vizsgáljuk azt az esetet, amikor OPT valamely szervere d távolságot mozog. Ekkor a potenciálfüggvényben szereplő első részben az összegnek egyetlen tagja változhat, annak az értéke legfeljebb d-vel növekszik, így ez a rész legfeljebb kd-vel növekszik. A második rész nem változik, tehát a potenciálfüggvény első tulajdonsága valóban fennáll.

Most vizsgáljuk DL szervereit. Legyen P a kérés, amelyet ki kell szolgálni. Mivel elsőként OPT szolgálta ki a kérést, ezért $x_j = P$ valamely szerverre. Most különböztessünk meg két esetet DL szervereinek elhelyezkedésétől függően.

Elsőként tegyük fel, hogy minden szerver P-nek ugyanarra az oldalára esik. Feltehetjük, hogy minden szerver pozíciója nagyobb P-nél, a másik eset teljesen hasonló. Ekkor s_1 a legközelebbi szerver P-hez és DL s_1-et küldi P-be, más szervert nem mozgat. Tehát DL költsége $d(s_1, P)$. A potenciálfüggvényben szereplő első összegben csak az $|x_1 - s_1|$ tag változik és ez csökken $d(s_1, P)$ egységgel, tehát az első rész csökken $kd(s_1, P)$ egységgel. A második tag növekszik $(k-1)d(s_1, P)$ egységgel, így Φ értéke csökken $d(s_1, P)$ egységgel.

Most tekintsük a másik esetet. Ekkor P-nek mindkét oldalára esik szerver, legyenek ezek a szerverek s_i és s_{i+1}. Tegyük fel, hogy s_i esik közelebb P-hez, a másik eset teljesen hasonló. Tehát DL költsége $2d(s_i, P)$. Vizsgáljuk a potenciálfüggvény első részének változásait. Az i-edik és az $i+1$-edik tag változik. Az egyik tag növekszik, a másik csökken $d(s_i, P)$ egységgel, tehát az első rész összességében nem változik. A Φ függvény második részének változása

$$d(s_i, P)\big(-(k-i) + (i-1) - (i) + (k-(i+1))\big) = -2d(s_i, P)\ .$$

Tehát ebben az esetben is fennáll a potenciálfüggvény második tulajdonsága. Mivel több eset nem lehetséges, ezért igazoltuk a potenciálfüggvény tulajdonságainak fennállását, amivel a tétel állítását is bebizonyítottuk. ∎

Gyakorlatok

23.2-1. Legyen (M, d) egy metrikus tér. Igazoljuk, hogy (M, q) is egy

metrikus tér, ahol $q(x, y) = \min\{1, d(x, y)\}$.

23.2-2. Vegyük azt az algoritmust, amely minden kérést a hozzá legközelebbi szerverrel szolgál ki. Igazoljuk, hogy ha a metrikus tér egy egyenes, akkor az algoritmus nem konstans versenyképes.

23.2-3. Igazoljuk, hogy tetszőleges X és Z k-elemű multihalmazokra, továbbá minden t-re teljesül a $w_t(Z) \leq w_t(X) + d(X, Z)$ egyenlőtlenség, ahol $d(X, Z)$ az X és Z multihalmazok közötti minimális párosítás költsége, azaz a minimális költség, amellyel a szerverek az X konfigurációból a Z konfigurációba átvihetők.

23.2-4. Vegyük az egyenest, mint metrikus teret. Legyenek az on-line algoritmus szerverei a 2, 4, 5, 7 pontokban, az off-line algoritmus szerverei az 1, 3, 6, 9 pontokban. Határozzuk meg a 23.4. tétel bizonyításában használt potenciálfüggvény értékét. Hogyan változik ez a függvényérték, ha a 7 pontban levő on-line szerver a 8 pontba mozdul?

23.3. Számítógépes hálózatokhoz kapcsolódó modellek

A számítógépes hálózatok elmélete az alkalmazások rendkívüli fontosságából adódóan az informatika egyik legjelentősebb kutatási területévé vált. A hálózatok megtervezésénél és a hálózatok működése során számos optimalizálási feladat merül fel, amely feladatok többsége lényegében on-line, hiszen sem a forgalom, sem pedig a hálózati topológia esetleges változásai nem láthatók előre. A 90-es évek végén az on-line algoritmusok területén kutató szakemberek on-line matematikai modelleket dolgoztak ki a számítógépes hálózatok témakörébe tartozó on-line feladatokra. Ebben a részben ezzel a területtel fogunk foglalkozni, három kérdést vizsgálunk meg, bemutatjuk az alapvető eredményeket. Elsőként a nyugtázás feladatára kidolgozott modellt ismertetjük, majd a lapletöltés feladatával, végül az on-line forgalomirányítás területével foglalkozunk.

23.3.1. A nyugtázási feladat modellje

A számítógépes hálózatok kommunikációja során a küldő a címzettnek adatcsomagokat küld. Amennyiben a kommunikációs csatorna nem teljesen megbízható (például vezeték nélküli), akkor lényeges a csomagok megérkezéséről a címzettnek nyugtát küldeni, így lehetővé tehető az elveszett adatcsomagok újraküldése. A nyugtázási feladat során felmerülő kérdések egyike, hogy mikor nyugtázzuk a csomag megérkezését. Egy nyugtával több csomag megérkezését igazolhatjuk. Minden csomagra külön nyugta küldése nagy

mértékben növelné a kommunikációs csatornák telítettségét. Másrészt, a nyugta küldésével hosszan várakozni sem lehet, hiszen az igazolás késedelme a csomag újraküldéséhez vezethet, ami ismét a csatorna túltelítettségét eredményezheti. A nyugtaküldések idejének megállapítására az első optimalizálási modellt Dooly, Goldman és Scott fejlesztették ki 1998-ban. Az alábbiakban ismertetjük az általuk kifejlesztett modell lényegét és az alapvető eredményeket.

A nyugtázási feladat matematikai modelljében a bemenetet a csomagok $a_1, \ldots, a_n$ érkezési idejei adják. Az algoritmusnak meg kell határoznia, mikor küld nyugtákat, ezeket az időpontokat $t_1, \ldots, t_k$ jelöli. A modellben a költségfüggvény:

$$k + \sum_{j=1}^{k} \nu_j \ ,$$

ahol k a nyugták száma és $\nu_j = \sum_{t_{j-1} < a_i \leq t_j} (t_j - a_i)$ a j-edik nyugta által összegyűjtött teljes késedelem. A feladat on-line, azaz egy adott t időpontban csak a t-ig megérkezett csomagok érkezési idejét ismerjük és nincs semmi információnk a további csomagokról. Az a_i csomag érkezése után σ_i jelöli a nyugtázatlan csomagok halmazát.

A feladat megoldására az ébresztő beállításán alapuló algoritmusokat dolgoztak ki. Egy ***ébresztő algoritmus*** a következőképpen működik. Az a_j csomag érkezésekor beállítunk egy ébresztőt valamilyen $a_j + e_j$ időpontra. Ha az $a_j + e_j$ időpontig nem érkezik új csomag, akkor az $a_j + e_j$ időpontban nyugtát küldünk, egyébként a következő csomag érkezésekor az a_{j+1} időpontban átállítjuk az ébresztőt egy $a_{j+1} + e_{j+1}$ időpontra. Az alábbiakban egy ébresztő algoritmust elemzünk részletesebben, azt az algoritmust, amely úgy állítja be az ébresztőt, hogy az első nyugtázatlan csomagtól legyen a teljes késedelem költsége 1. Ezt az algoritmust ÉBRESZT algoritmusnak nevezzük. A fenti szabály azt jelenti, hogy az általános definícióban az e_j érték a következő egyenlet megoldása:

$$1 = |\sigma_j| e_j + \sum_{a_i \in \sigma_j} (a_j - a_i) \ .$$

23.4. példa. Tekintsük a következő példát. Az első csomag a 0 időpontban érkezik, azaz $a_1 = 0$, ekkor ÉBRESZT beállítja az ébresztőkezdeti állapotát az $e_1 = 1$ értékkel. Legyen a következő csomag érkezési ideje $a_2 = 1/2$, ekkor még nem járt le az ébresztő, így nem küldtünk nyugtát, tehát átállítjuk az ébresztőt az $e_2 = (1 - 1/2)/2 = 1/4$ értékkel az $1/2 + 1/4$ időpontra. Legyen a következő csomag érkezési ideje $a_3 = 5/8$. Ekkor még nem járt le az ébresztő és nem küldtünk nyugtát, így újra állítjuk az ébresztőt az $e_3 = (1 - 5/8 - 1/8)/3 = 1/12$ értékkel az $5/8 + 1/12$ időpontra. Legyen a következő csomag érkezési ideje $a_4 = 1$, mivel az $5/8 + 1/12$ időpontig nem érkezett újabb csomag, ezért abban az időpontban nyugtát küldtünk, és most az ébresztőt az $e_4 = 1$ értékkel a 2 időpontra állítjuk be.

Az algoritmus versenyképességére teljesül a következő állítás.

23.5. tétel. *Az* ÉBRESZT *algoritmus 2-versenyképes.*

Bizonyítás. Tegyük fel, hogy az ÉBRESZT algoritmus k darab nyugtát küld. Ezek a nyugták k darab intervallumot határoznak meg. Az algoritmus költsége legfeljebb $2k$, hiszen k a nyugtákból keletkező költség, és az algoritmus úgy állítja be az ébresztőt, hogy a teljes késedelemből adódó költség minden nyugtára pontosan 1 legyen.

Legyen k^* az optimális off-line algoritmus által küldött nyugták száma. Ha $k^* \geq k$, akkor $\mathrm{OPT}(I) \geq k = \textsc{Ébreszt}(I)/2$ nyilvánvalóan teljesül és adódik, hogy az algoritmus valóban 2-versenyképes. Amennyiben $k^* < k$, akkor az ÉBRESZT algoritmus nyugtái által meghatározott k intervallum közül legalább $k - k^*$ intervallumban OPT-nak nincs nyugtája ez legalább $k - k^*$ késedelemből származó költséget jelent OPT számára, így ismét $\mathrm{OPT}(I) \geq k$ adódik, amely egyenlőtlenségből következik, hogy az algoritmus 2-versenyképes. ■

A fentiekben ismertetett ÉBRESZT algoritmus a lehetséges legjobb algoritmus a versenyképességi analízis szempontjából, hiszen miként a következő állítás mutatja, nem létezik olyan algoritmus, amelynek kisebb lenne a versenyképességi hányadosa.

23.6. tétel. *Nem létezik olyan on-line algoritmus az on-line nyugtázási feladatra, amelynek kisebb a versenyképességi hányadosa, mint 2.*

Bizonyítás. Vegyünk egy tetszőleges on-line algoritmust, jelölje ONL. Tekintsük a következő bemenetet. Vegyük csomagok egy hosszú sorozatát, amelyet úgy kapunk, hogy minden esetben, amikor ONL egy nyugtát küld, azonnal egy új csomag érkezik. Ekkor egy $2n$ csomagból álló sorozat esetén az on-line algoritmus költsége $\mathrm{ONL}(I_{2n}) = 2n + t_{2n}$, hiszen a nyugtákból adódó költsége $2n$, és az i-edik nyugtánál fellépő késedelem $t_i - t_{i-1}$, ahol a $t_0 = 0$ értéket használjuk.

Vizsgáljuk meg a következő két algoritmust. ODD a páros sorszámú csomagok után küld nyugtát, EV pedig a páratlan sorszámú csomagok és közvetlenül az utolsó, $2n$-edik csomag után.

Ekkor ezen algoritmusok költségei

$$\mathrm{EV}(I_{2n}) = n + \sum_{i=0}^{n-1} (t_{2i+1} - t_{2i}) + 1 \ ,$$

és

$$\mathrm{ODD} = n + \sum_{i=1}^{n} (t_{2i} - t_{2i-1}) \ .$$

Következésképpen $\mathrm{EV}(I_{2n})+\mathrm{ODD}(I_{2n}) = \mathrm{ONL}(I_{2n})+1$. Másrészt az optimális off-line algoritmusra $\mathrm{OPT}(I_{2n}) \leq \min\{\mathrm{EV}(I_{2n}), \mathrm{ODD}(I_{2n})\}$, így azt kapjuk, hogy $\mathrm{ONL}(I_{2n})/\mathrm{OPT}(I_{2n}) \geq 2 - 1/\mathrm{OPT}(I_{2n})$. Ezen egyenlőtlenség alapján adódik, hogy ONL nem lehet jobb, mint 2-versenyképes, hiszen a csomagok elegendően hosszú sorozatát véve az $\mathrm{OPT}(I_{2n})$ érték tetszőlegesen nagy lehet. ■

23.3.2. A lapletöltési feladat

A 34.2. alfejezetben tárgyalt lapcserélési feladat általánosítása a lapletöltési feladat. A világhálón a böngészők is használnak egy memóriát, amelyben a letöltött lapokat tárolják, hogy amennyiben a felhasználó egy oldalt rövid időn belül többször meg akar nézni, ne kelljen minden alkalommal letölteni. Amennyiben a memória megtelik és az új lap nem helyezhető el benne, akkor valamilyen stratégia alapján ki kell rakni bizonyos lapokat a memóriából. A lapletöltési feladat a megfelelő cserélési stratégiák megkeresése. A különbség a lapozás feladatához képest, hogy nem minden lap mérete egyforma, továbbá az egyes lapok letöltési költsége is különböző. Tehát a modellt a következőképpen fogalmazhatjuk meg.

Adott egy k méretű memória. A bemenet a letöltendő lapok sorozata. Minden p lapnak van egy $s(p)$ ***mérete*** és egy $c(p)$ ***letöltési költsége.*** A letöltendő lapot a memóriába kell tennünk. Ha nem fér el, akkor fel kell szabadítani elegendő helyet a memóriában szereplő lapok kihelyezésével. Ha a kért lap a memóriában van, akkor a letöltés költsége 0, egyébként $c(p)$. Célunk az összköltség minimalizálása. A feladat on-line, ami azt jelenti, hogy a döntéseket (telített memória esetén mely lapokat dobjuk ki) csak az adott kérésig megtörtént kérések ismerete alapján kell meghozni, a további kérésekre vonatkozó információk nélkül. A továbbiakban feltesszük, hogy mind a memória mérete, mind pedig a lapok méretei pozitív egész számok.

A feladatnak és speciális eseteinek megoldására több eljárást is javasoltak. Az alábbiakban a Young által kifejlesztett enyhén k-versenyképes HÁZIÚR algoritmust és annak elemzését ismertetjük.

Az algoritmus minden lapra, amely az algoritmus memóriájában van, számon tart egy $0 \leq cr(f) \leq c(f)$ értéket. Az algoritmus futása során a HÁZIÚR algoritmus memóriájában aktuálisan szereplő lapok halmazát HA jelöli. Amennyiben egy g fájlt kell letöltenünk, a következő lépéseket hajtjuk végre:

HÁZIÚR(HA, g)

```
1  if g nincs a memóriában
2     then while nincs elég hely
3                Δ ← min_{f∈HA} cr(f)/s(f)
4                minden f ∈ HA-ra cr(f) ← cr(f) − Δ · s(f)
5                tegyünk ki olyan lapokat, amelyekre cr(f) = 0
6                tegyük be g-t a HA memóriába, cr(g) ← c(g)
7     else  állítsuk át cr(g) értékét valamelyik cr(g) és c(g) közötti értékre
```

23.5. példa. Tegyük fel, hogy $k = 10$ és az adott pillanatban a HA memória három lapot tartalmaz: g_1-re $s(g_1) = 2, cr(g_1) = 1$, g_2-re $s(g_2) = 4, cr(g_2) = 3$, g_3-ra $s(g_3) = 3, cr(g_3) = 3$. Legyen a következő letöltendő lap g_4, amelyre $s(g_4) = 4, c(g_4) = 4$. Ekkor ez a lap nem fér el a HA memóriában, ki kell tennünk lapokat. A HÁZIÚR algoritmus által számolt érték $\Delta = 1/2$ és a megváltoztatott cr értékek: $cr(g_1) = 0, cr(g_2) = 1, cr(g_3) = 3/2$, így g_1-et eltávolítjuk a HA memóriából. Ekkor a g_4 lap még mindig nem fér el a HA memóriában. Az új Δ érték $\Delta = 1/4$ és a megváltoztatott cr értékek: $cr(g_2) = 0, cr(g_3) = 3/4$, így a g_2 lapot is eltávolítjuk a HA memóriából. Ekkor már van hely g_4 számára, elhelyezzük a memóriában a $cr(g_4) = 4$ értékkel.

Az algoritmus enyhén k-versenyképes, de ennél erősebb állítás is igaz. A lapletöltési feladatra egy ALG on-line algoritmust enyhén (C, k, h)-versenyképesnek nevezünk, ha van olyan B konstans, hogy $\mathrm{ALG}_k(I) \leq C \cdot \mathrm{OPT}_h(I) + B$ minden bemenetre teljesül, ahol $\mathrm{ALG}_k(I)$ az algoritmus által elvégzett összes letöltés költsége k méretű memória mellett, $\mathrm{OPT}_h(I)$ pedig a minimális elérhető teljes letöltési összeg h méretű memória mellett. A HÁZIÚR algoritmusra teljesül a következő állítás.

23.7. tétel. *Ha $h \leq k$, akkor a* HÁZIÚR *algoritmus $(k/(k-h+1), k, h)$-versenyképes.*

Bizonyítás. Tekintsük kérések egy tetszőleges sorozatát, jelölje ezt a bemenetet I. Miként a 23.4. tétel bizonyítása során, itt is a potenciálfüggvény technikát alkalmazzuk. Párhuzamosan hajtjuk végre az OPT és a HÁZIÚR algoritmusokat, minden kérésnél először az OPT és utána a HÁZIÚR algoritmus lépését hajtjuk végre.

Jelölje az optimális off-line algoritmusnak a memóriájában aktuálisan szereplő lapjainak halmazát OPT. Tekintsük a következő potenciálfüggvényt.

$$\Phi = (h-1) \sum_{f \in HA} cr(f) + k \sum_{f \in OPT} (c(f) - cr(f)) \; .$$

Vizsgáljuk a potenciálfüggvény változásait egy g lap letöltése során.

- OPT elhelyez egy g lapot a memóriájában.

  Ekkor OPT költsége $c(g)$, a potenciálfüggvénynek csak a második része változhat, mivel $cr(g) \geq 0$, ezért a potenciálfüggvény növekedése legfeljebb $k \cdot c(g)$.

- HÁZIÚR csökkenti minden $f \in HA$-ra a $cr(f)$ értéket.

  Ekkor minden $f \in HA$ esetén a $cr(f)$ érték csökkenése $\Delta \cdot s(f)$, így összességében Φ a

  $$\Delta((h-1)s(HA) - ks(OPT \cap HA))$$

  értékkel csökken, ahol $s(HA)$ és $s(OPT \cap HA)$ rendre a HA illetve az $OPT \cap HA$ halmazokban levő lapoknak a méreteinek az összege. Abban az időpontban, amikor ez a lépés bekövetkezik, OPT már elhelyezte a g lapot OPT-ban, de a lap még nincs a HA halmazban. Következésképp $s(OPT \cap HA) \leq h - s(g)$. Másrészt ez a lépés azért hajtódik végre, mert nem fért el a g lap a HA memóriában, tehát $s(HA) > k - s(g)$, és így, mivel a lapok méretei egészek, ezért $s(HA) \geq k - s(g) + 1$. Következésképpen azt kapjuk, hogy a Φ potenciálfüggvény csökkenése legalább

  $$\Delta\big((h-1)(k-s(g)+1) - k(h-s(g))\big) \ .$$

  Mivel $s(g) \geq 1$ és $k \geq h$, ezért a fenti érték legalább $\Delta((h-1)(k-1+1) - k(h-1)) = 0$.

- HÁZIÚR kirak egy f lapot a HA memóriából.

  Mivel HÁZIÚR csak akkor rak ki egy f lapot a memóriából, ha arra $cr(f) = 0$, ezért ebben a részben nem változik Φ.

- HÁZIÚR elhelyezi a g lapot a HA memóriában és beállítja a $cr(g) = c(g)$ értéket.

  Ekkor HÁZIÚR költsége $c(g)$. Másrészt g nem volt eddig benne a HA memóriában így $cr(g) = 0$ teljesült. Továbbá elsőként OPT helyezte el a lapot, így $g \in OPT$ teljesül. Következésképpen a Φ függvény értékének csökkenése ebben a lépésben $-(h-1)c(g) + kc(g) = (k-h+1)c(g)$.

- HÁZIÚR átállítja a $g \in HA$ lapra a $cr(g)$ értéket, valamely $cr(g)$ és $c(g)$ közötti értékre.

  Ebben az esetben is fennáll $g \in OPT$, hisz OPT már hamarabb elhelyezte g-t a memóriájába. Mivel $cr(g)$ nem csökkenhet, és $k > h-1$, ezért ebben az esetben Φ nem növekedhet.

Végignéztük az algoritmusok lehetséges lépéseit és a Φ függvény következő tulajdonságai adódtak.

- Ha OPT elhelyez egy lapot a memóriájában, akkor a potenciálfüggvény növekedése legfeljebb k-szor az OPT algoritmusnál fellépő költség.
- Ha HÁZIÚR elhelyez egy lapot a memóriájában, akkor a Φ függvény $(k-h+1)$-szer annyival csökken, mint amennyi az algoritmusnál fellépő költség.
- A többi esetben Φ nem növekszik.

A fentiek alapján azt kapjuk, hogy $\Phi_v - \Phi_0 \leq k \cdot \mathrm{OPT}_h(I) - (k-h+1) \cdot \mathrm{HÁZIÚR}_k(I)$, ahol Φ_0 és Φ_v a potenciálfüggvény kezdeti és végső értékei. Mivel a potenciálfüggvény nemnegatív, ezért adódik, hogy $(k-h+1) \cdot \mathrm{HÁZIÚR}_k(I) \leq k \cdot \mathrm{OPT}_h(I) + \Phi_0$, azaz azt kapjuk, hogy a HÁZIÚR algoritmus enyhén $(k/(k-h+1), k, h)$-versenyképes. ■

23.3.3. Forgalomirányítási algoritmusok

A számítógépes hálózatok esetén az egyes kommunikációs csatornák túltelítettsége a számítógépes forgalom nagymértékű lassulásához és információk elvesztéséhez vezethet. Ezért a számítógépes hálózatok elméletének egyik alapvető feladata a forgalom szabályozása. A forgalom szabályozásának témakörébe tartozik a forgalomirányítás is, melynek során azt kell meghatároznunk, hogy az egyes üzenetek az üzenet küldőjétől a címzetthez milyen útvonalon jussanak el, mely közbenső állomásokon keresztül. A számítógépes hálózatok esetén nincs teljes információnk az összes jelenlegi és jövőbeli üzenetről, amely a rendszerbe kerül, így valóban egy on-line feladatról van szó. Az alábbiakban a forgalomirányítás feladatának két on-line modelljét mutatjuk be, amely modellek nem csak a számítógépes hálózatok modellezésére alkalmasak, hanem általánosabb forgalomirányítási feladatok tanulmányozására is.

A matematikai modell

A hálózatot egy gráf ábrázolja és minden e élnek van egy $u(e)$ maximális felhasználható sávszélessége, az élek számát m-el jelöljük. A feladat az, hogy sorban kérések érkeznek, a j-edik kérés egy $(s_j, t_j, r_j, d_j, b_j)$ vektor, a feladat pedig az s_j pontból a t_j pontba egy kiválasztott úton r_j sávszélességet lefoglalni d_j időtartamra a kérés megjelenésétől kezdve. Amennyiben a kérést elfogadjuk, a nyereség b_j. A továbbiakban mindig feltesszük, hogy $d_j = \infty$ minden j kérés esetén. A feladat on-line, ami azt jelenti, hogy a kérés időpontjában nem tudunk semmit a későbbi kérésekről. Két különböző modellt ismertetünk.

Terhelést kiegyensúlyozó modell. Ebben a modellben minden kérést el kell fogadni és a célfüggvény az élek túltelítettségének, ami a hozzájuk ren-

delt teljes sávszélességnek és a megengedett maximális felhasználható sávszélességnek a hányadosa, a maximumának a minimalizálása.

Nyereség maximalizáló modell. Ebben a modellben visszautasíthatunk kéréseket, a teljes sávszélesség egyetlen élen sem lehet nagyobb, mint a megengedett maximális sávszélesség, a cél az elfogadott kérésekhez rendelt nyereségek összegének maximalizálása. Mivel a továbbiakban ezzel a modellel foglalkozunk ezért fontosnak tartjuk kiemelni, hogy az eddigi részekkel ellentétben itt maximalizálási feladatról van szó, így a versenyképesség fogalmát a maximalizálási feladatokra megadott formában fogjuk használni.

Az alábbiakban ismertetjük az exponenciális algoritmust. Az algoritmus pontos megfogalmazásához és elemzéséhez szükségünk lesz az alábbi jelölésekre. Minden elfogadott i kérésre a kéréshez lefoglalt utat P_i jelöli. Legyen A az algoritmus által elfogadott kérések halmaza. Ekkor az $l_e(j) = (\sum_{i\in A, i<j, e\in P_i} r_i)/u(e)$ érték azt adja meg, hogy a j-edik kérés előtt az e-n keresztül lefoglalt sávszélesség a megengedett sávszélességnek mekkora része.

Az exponenciális algoritmusok alapötlete, hogy minden élhez egy $l_e(j)$-ben exponenciális költséget rendelnek, és minimális költségű utat választanak. Az alábbiakban részletesen ismertetjük és elemezzük az exponenciális algoritmust a nyereségmaximalizáló modellre. Ehhez legyen μ egy későbbiekben definiált, a feladat paramétereitől függő konstans és legyen $c_e(j) = \mu^{l_e(j)}$, minden j kérés és e él esetén. Ekkor az algoritmus egy (s_j, t_j, r_j, b_j) kérést a következőképpen bírál el.

EXP(s_j, t_j, r_j, b_j)

1 U_j legyen az (s_j, t_j) utak halmaza
2 $P_j = \operatorname{argmin}_{P\in U_j}\{\sum_{e\in P} \frac{r_j}{u(e)} c_e(j)\}$
3 **if** $C(P_j) = \sum_{e\in P_j} \frac{r_j}{u(e)} c_e(j) \le 2mb_j$
4 lefoglaljuk a P_j úton az r_j sávszélességet
5 **else** visszautasítjuk a kérést

Megjegyzés. Amennyiben a fenti algoritmusban minden kérést elfogadunk, akkor a terhelést kiegyensúlyozó modellre kapjuk meg az exponenciális algoritmust.

23.6. példa. Tekintsük azt a hálózatot, amely az A, B, C és D pontokból, valamint az (A, B), (B, D), (A, C) és (C, D) élekből áll, ahol az élek maximális sávszélességére $u(A, B) = 1$, $u(B, D) = 3/2$, $u(A, C) = 2$,$u(C, D) = 3/2$. Tegyük fel, hogy $\mu = 10$ és a j-edik kérés előtt az eddig lefoglalt sávszélességek 3/4 az A, B, D úton, 5/4 az A, C, D úton, 1/2 a (B, D) élen, 1/2 az (A, C) élen. Ekkor az $l_e(j)$ értékek: $l_{(A,B)}(j) = (3/4) : 1 = 3/4$, $l_{(B,D)}(j) = (3/4 + 1/2) : (3/2) = 5/6$, $l_{(A,C)}(j) = (5/4+1/2) : 2 = 7/8$, $l_{(C,D)}(j) = (5/4) : (3/2) = 5/6$. Legyen a következő

kérés 1/8 sávszélesség lefoglalása A és D között. A két lehetséges útra a $C(P)$ értékek

$$C(A,B,D) = 1/8 \cdot 10^{3/4} + 1/12 \cdot 10^{5/6} = 1.269\ ,$$

$$C(A,C,D) = 1/16 \cdot 10^{7/8} + 1/12 \cdot 10^{5/6} = 1.035\ .$$

A minimális érték a (A,C,D) úton van. Így, amennyiben a tárgyra $2mb_j = 8b_j \geq 1,035$ a kérést az (A,C,D) úton elfogadjuk.

Az algoritmus elemzéséhez tekintsük kérések egy tetszőleges I bemeneti sorozatát. Jelölje A az EXP algoritmus által elfogadott kérések halmazát, A^* az OPT algoritmus által elfogadott, de az EXP algoritmus által elutasított kérések halmazát. Továbbá minden olyan j kérésre, amelyet az OPT algoritmus elfogad, jelölje az OPT által hozzárendelt utat $P_j{}^*$. Az $l_e(j)$ értékekhez hasonlóan definiáljuk az $l_e(v) = (\sum_{i\in A, e\in P_i} r_i)/u(e)$ értéket, amely azt adja meg, hogy az e-n keresztül lefoglalt sávszélesség a megengedett sávszélességnek mekkora része az eljárás végén.

Most tegyük fel, hogy $\mu = 4mPB$, ahol B felső korlátja a nyereségeknek, továbbá minden kérésre és élre teljesül

$$\frac{1}{P} \leq \frac{r(j)}{u(e)} \leq \frac{1}{\lg \mu}\ .$$

Ekkor az algoritmus versenyképességére vonatkozó állítás kimondása előtt igazoljuk a következő lemmákat.

23.8. lemma. *Az* EXP *algoritmus által kapott megoldás lehetséges, azaz egyetlen élen sem lépjük túl a megengedett sávszélességet.*

Bizonyítás. Az állítást indirekt igazoljuk. Tegyük fel, hogy a megengedett sávszélességet az f élen túllépjük. Legyen j az első olyan kérés, amelynek az elfogadásával túlléptük a megengedett sávszélességet.

Mivel minden élre és kérésre, így j-re és f-re is teljesül, hogy $r_j/u(f) \leq 1/\lg\mu$ és a j kérés elfogadása után az f élen túllépjük a megengedett sávszélességet, ezért adódik, hogy $l_f(j) > 1 - 1/\lg\mu$. Másrészt ekkor az EXP algoritmus második lépésében számolt $C(P_j)$ értékre

$$C(P_j) = \sum_{e\in P_j} \frac{r_j}{u(e)} c_e(j) \geq \frac{r_j}{u(f)} c_f(j) > \frac{r_j}{u(f)} \mu^{1-1/\lg\mu}\ .$$

Szintén a tétel feltételei alapján $\frac{r_j}{u(e)} \geq \frac{1}{P}$, továbbá $\mu^{1-1/\lg m} = \mu/2$, így a fenti egyenlőtlenség alapján azt kapjuk, hogy

$$C(P) > \frac{1}{P}\frac{\mu}{2} = 2mB\ .$$

Másrészt ezzel az egyenlőtlenséggel ellentmondáshoz jutottunk, hiszen amennyiben a fenti egyenlőtlenség fennáll, akkor EXP visszautasítja a kérést. Mivel ellentmondáshoz jutottunk, ezért a kiinduló feltevésünk hamis kell legyen, azaz a lemma állítását igazoltuk. ■

23.9. lemma. *Az* OPT *algoritmus által kapott megoldásra teljesül*

$$\sum_{j\in A^*} b_j \le \frac{1}{2m}\sum_{e\in E} c_e(v)\ .$$

Bizonyítás. Mivel EXP minden $j \in A^*$ kérést visszautasított, ezért minden $j \in A^*$-ra $b_j < \frac{1}{2m}\sum_{e\in {P_j}^*}\frac{r_j}{u(e)}c_e(j)$, hiszen a fenti egyenlőtlenség minden (s_j, t_j) útra teljesül, így az optimális megoldás által választott útra is. Tehát

$$\sum_{j\in A^*} b_j < \frac{1}{2m}\sum_{j\in A^*}\sum_{e\in {P_j}^*}\frac{r_j}{u(e)}c_e(j)\ .$$

Másrészt minden e élre $c_e(j) \le c_e(v)$, így a fenti egyenlőtlenség alapján adódik, hogy

$$\sum_{j\in A^*} b_j < \frac{1}{2m}\sum_{e\in E} c_e(v)\Big(\sum_{j\in A^*:e\in {P_j}^*}\frac{r_j}{u(e)}\Big)\ .$$

Mivel minden e élre az OPT által elfogadott teljes sávszélesség legfeljebb $u(e)$, ezért minden e-re $\sum_{j\in A^*:e\in P^*(j)}\frac{r_j}{u(e)} \le 1$. Következésképpen

$$\sum_{j\in A^*} b_j \le \frac{1}{2m}\sum_{e\in E} c_e(v)$$

adódik, amely pontosan a lemma állítása. ■

23.10. lemma. *Az* EXP *algoritmus által kapott megoldásra teljesül*

$$\frac{1}{2m}\sum_{e\in E} c_e(v) \le (1+\lg\mu)\sum_{j\in A} b_j\ .$$

Bizonyítás. A lemma igazolásához elegendő belátnunk, hogy minden $j \in A$ kérésre teljesül $\sum_{e\in P_j}(c_e(j+1) - c_e(j)) \le 2mb_j\log_2\mu$. Másrészt

$$c_e(j+1) - c_e(j) = \mu^{l_e(j)+\frac{r_j}{u(e)}} - \mu^{l_e(j)}\mu^{l_e(j)}(2^{\log_2\mu\frac{r_j}{u(e)}} - 1)\ .$$

Mivel $2^x - 1 < x$, ha $0 \le x \le 1$, és a feltételeink alapján $0 \le \log_2\mu\frac{r_j}{u(e)} \le 1$, ezért azt kapjuk, hogy

$$c_e(j+1) - c_e(j) \le \mu^{l_e(j)}\log_2\mu\frac{r_j}{u(e)}.$$

A fentiekben kapott felső becsléseket összegezve adódik, hogy

$$\sum_{e\in P_j}(c_e(j+1)-c_e(j)) \le \log_2\mu \sum_{e\in P_j}\mu^{l_e(j)}\frac{r_j}{u(e)} = \log_2\mu\cdot C(P_j)\ .$$

Mivel EXP azokat a kéréseket fogadja el, amelyekre $C(P_j)\le 2mb_j$, ezért a fenti egyenlőtlenség alapján adódik a lemma állítása. ∎

A fenti lemmák alapján igazoljuk a következő tételt.

23.11. tétel. *Az* EXP *algoritmus* $1/O(\lg\mu)$*-versenyképes, ha* $\mu = 4mPB$*, ahol* B *felső korlátja a nyereségeknek, továbbá minden kérésre és élre teljesül*

$$\frac{1}{P} \le \frac{r(j)}{u(e)} \le \frac{1}{\lg\mu}\ .$$

Bizonyítás. A 23.8. lemma alapján adódik, hogy az eljárás korrekt nem lépjük át a megengedett sávszélességet. A fentiekben bevezetett jelöléseket használva a EXP algoritmus nyeresége az I inputon $\mathrm{EXP}(I)=\sum_{j\in A} b_j$, az OPT algoritmus nyeresége pedig legfeljebb $\sum_{j\in A\cup A^*} b_j$. Következésképpen a 23.9 és 23.10 lemmák alapján azt kapjuk, hogy

$$\mathrm{OPT}(I) \le \sum_{j\in A\cup A^*} b_j \le (2+\log_2\mu)\sum_{j\in A} b_j \le (2+\log_2\mu)\mathrm{EXP}(I)\ ,$$

amely egyenlőtlenség igazolja a tétel állítását. ∎

Gyakorlatok

23.3-1. Tekintsük a nyugtázási feladat azon változatát, amelyben a célfüggvény, $k+\sum_{j=1}^{k}\mu_j$, ahol k a nyugták száma és $\mu_j = \max_{t_{j-1}<a_i\le t_j}\{t_j-a_i\}$ a j-edik nyugta által összegyűjtött maximális késedelem. Igazoljuk, hogy az ÉBRESZT algoritmus ezen célfüggvény esetén is 2-versenyképes.

23.3-2. Reprezentáljuk a lapletöltési feladat azon speciális esetét, amelyben minden lapra $s(g)=c(g)=1$, a k-szerver feladat speciális eseteként. Milyen metrikus teret használhatunk?

23.3-3. Legyen a lapletöltési feladatban a 8 méretű HA memóriában három lap a,b,c, amelyeknek a mérete és az aktuális kreditértékei $s(a)=3, s(b)=2, s(c)=3, cr(a)=2, cr(b)=1/2, cr(c)=2$. Egy d lapot akarunk letölteni, amelynek a mérete 3, és a letöltési költsége 4. A 6 méretű memóriával rendelkező OPT algoritmus már letöltötte a lapot, a memóriájában d és c szerepel. Adjuk meg, miként helyezi el a HÁZIÚR algoritmus a d lapot és adjuk meg a 23.7. tétel bizonyítása során használt potenciálfüggvény változásait.

23.3-4. Igazoljuk, hogy amennyiben a nyereségmaximalizáló forgalomirányítási modellben az $r(j)/u(e)$ hányadosokra semmiféle kikötést nem teszünk, akkor nem adható meg konstans-versenyképes algoritmus.

23.4. On-line ládapakolási modellek

A ládapakolási feladatnak és különböző változatainak számos gyakorlati alkalmazása van. Ebben az alfejezetben ezzel a feladattal foglalkozunk. Az első részben ismertetjük a klasszikus on-line ládapakolásra vonatkozó alapvető eredményeket. Az alfejezet második részében megadjuk a lehetséges többdimenziós általánosításokat és részletesebben vizsgáljuk a sávpakolási feladatot.

23.4.1. On-line ládapakolás

A ládapakolási feladatban bemenetként tárgyak egy sorozatát kapjuk meg, ahol az i-edik tárgyat a mérete határozza meg, ami egy $a_i \in (0,1]$ érték. Célunk a tárgyak elhelyezése a lehető legkevesebb egység méretű ládába. Formálisabban megfogalmazva a tárgyakat olyan csoportokba akarjuk osztani, hogy minden csoportra a benne levő tárgyakra a $\sum a_i \leq 1$ feltétel teljesüljön. A feladat modellezi a memóriakezelés esetén az állományok optimális elhelyezésének feladatát is, amely kérdéskört ezen könyv első kötetében a memóriagazdálkodásról szóló fejezetben is tárgyaljuk.

Ebben a részben az on-line ládapakolási feladatot vizsgáljuk, amelyben az algoritmusnak az i-edik tárgyat az $a_1, \ldots, a_i$ értékek alapján kell elhelyezni a további tárgyakra vonatkozó információk ismerete nélkül.

Az NF algoritmus, helykorlátos algoritmusok

Érdemes külön megemlítenünk azt a modellt, amelyben az egyidőben nyitott ládák száma korlátozott. Ez azt jelenti, hogy amennyiben a nyitott ládák száma elér egy k-korlátot, akkor ezt követően csak akkor nyithatunk új ládát, ha az eddig nyitott ládák valamelyikét bezárjuk, ami azt jelenti, hogy többet nem használhatjuk. Ha a nyitott ládák száma csak egy lehet, akkor természetesen adódik az algoritmus, amely az aktuálisan nyitott ládába teszi a tárgyat ha az elfér, ellenkező esetben bezárja a nyitott ládát és új ládát nyit. Ezt az algoritmust NF algoritmusnak nevezzük. Az algoritmus pszeudokódját nem ismertetjük, a memóriagazdálkodás fejezetben megtalálható. Az NF algoritmus aszimptotikus versenyképességi hányadosára teljesül a következő állítás.

23.12. tétel. *Az* NF *algoritmus aszimptotikus versenyképességi hányadosa 2.*

Bizonyítás. Vegyünk egy tetszőleges σ tárgysorozatot. Jelölje n az OPT algoritmus által használt ládák számát, jelölje m az NF algoritmus által használt ládák számát. Továbbá legyen S_i, $i = 1, \ldots, m$ az i-edik ládában levő tárgyak méreteinek összege.

Ekkor $S_i+S_{i+1} > 1$, hiszen ellenkező esetben az $(i+1)$-edik láda első eleme elfért volna az i-edik ládában, ami ellentmond az algoritmus definíciójának. Következésképp a tárgyak méreteinek összege több, mint $\lfloor m/2 \rfloor$.

Másrészt az optimális off-line algoritmus sem rakhat 1-nél több összméretű tárgyakat egy ládába, így azt kapjuk, hogy $n > \lfloor m/2 \rfloor$. Ez pedig azt jelenti, hogy $m \leq 2n-1$, így

$$\frac{\mathrm{NF}(\sigma)}{\mathrm{OPT}(\sigma)} \leq \frac{2n-1}{n} = 2 - 1/n \; .$$

Következésképpen igazoltuk, hogy az algoritmus aszimptotikusan 2-versenyképes.

Most megmutatjuk, hogy a fenti korlát éles. Ehhez tekintsük minden n-re a következő σ_n sorozatot. A sorozat $4n-2$ tárgyból áll, a $(2i-1)$-edik tárgy mérete $1/2$, a $2i$-edik tárgy mérete $1/4n$, ahol $i = 1, \ldots, 2n$. Ekkor az NF algoritmus az i-edik ládába a $(2i-1)$-edik és a $2i$-edik tárgyat teszi, és $\mathrm{NF}(\sigma_n) = 2n-1$. Az optimális algoritmus az első $n-1$ ládába $1/2$ méretű tárgyakat párosít, és az n-edik ládába egy $1/2$ méretűt és a kis tárgyakat teszi, így $\mathrm{OPT}(\sigma_n) = n$. Mivel $\mathrm{NF}(\sigma_n)/\mathrm{OPT}(\sigma_n) = 2 - 1/n$ a 2 értékhez konvergál, ha n tart végtelenbe, ezért igazoltuk, hogy az algoritmus aszimptotikus versenyképességi hányadosa legalább 2. ■

Itt érdemes megemlítenünk, hogy amennyiben egynél több (de korlátozott számú) láda lehet nyitva, az NF algoritmusnál jobb algoritmusok is ismertek. A jelenlegi legjobb algoritmusok a ***harmonikusalgoritmusok*** családjába tartoznak, ahol az alapötlet az, hogy a $(0, 1]$ intervallumot részintervallumokra osztjuk, és minden tárgynak az az intervallum lesz a típusa, amely intervallumba a mérete esik. A különböző típusú tárgyakat különböző ládákba pakoljuk, az algoritmus párhuzamosan alkalmaz egy-egy NF algoritmust az egyes típusokhoz tartozó tárgyakra.

Az FF algoritmus, a súlyfüggvény technika

Ebben a részben egy olyan módszert mutatunk be, amelyet gyakran használunk a ládapakolási algoritmusok elemzése során. A módszert az FF (First Fit) algoritmus bemutatásával ismertetjük.

Az FF algoritmus az NF algoritmus továbbfejlesztett változata, arra az esetre, amelyben nincs korlátozva a nyitott ládák száma. Az algoritmus az aktuálisan megérkező tárgyat mindig a legkorábban kinyitott ládába teszi, ahol elfér. Ha nem fér el egyik ládában sem, kinyit egy új ládát és abba rakja. Az algoritmus pszeudokódja megtalálható a memóriagazdálkodásról szóló fejezetben. Az algoritmus versenyképességi hányadosára ad felső korlátot a következő állítás.

23.13. tétel. *Az* FF *algoritmus aszimptotikusan 1.7-versenyképes.*

Bizonyítás. A bizonyítás alapötlete a súlyfüggvény technika, amely azt jelenti, hogy minden tárgyhoz egy súlyt rendelünk, amely azt adja meg valamilyen értelemben, hogy mennyire sok helyet foglalhat el a tárgy egy pakolásban. A tárgyaknak vesszük az összsúlyát, és ezen érték segítségével becsüljük meg az off-line és az on-line célfüggvények értékét. Definiáljuk a következő súlyfüggvényt:

$$w(x) = \begin{cases} 6x/5, & \text{ha } 0 \le x \le 1/6 \\ 9x/5 - 1/10, & \text{ha } 1/6 \le x \le 1/3 \\ 6x/5 + 1/10, & \text{ha } 1/3 \le x \le 1/2 \\ 6x/5 + 2/5, & \text{ha } 1/2 < x\ . \end{cases}$$

A tárgyak egy tetszőleges H halmazára legyen $w(H) = \sum_{i \in H} w(a_i)$. Ekkor a súlyfüggvényre teljesülnek az alábbi állítások. Mivel mindkét lemma bizonyítása azon alapul, hogy eseteket különböztetünk meg a tárgyak méretétől függően, továbbá a bizonyítások hosszúak és sok technikai részletet tartalmaznak, ezért a bemutatásuktól itt eltekintünk. ■

23.14. lemma. *Amennyiben tárgyak egy H halmazára teljesül, hogy $\sum_{i \in H} a_i \le 1$, akkor ezen tárgyakra $w(H) \le 17/10$.*

23.15. lemma. *Tárgyak tetszőleges L listájára $w(L) > \mathrm{FF}(L) - 2$.*

Bizonyítás. A lemmák alapján könnyen igazolható, hogy az algoritmus aszimptotikusan 1.7-versenyképes. Tekintsük tárgyak tetszőleges L listáját. Mivel az optimális off-line algoritmus el tudja pakolni a lista elemeit $\mathrm{OPT}(L)$ ládába úgy, hogy minden ládában a tárgyak méreteinek összege legfeljebb 1, ezért az első lemma alapján $w(L) \le 1.7\mathrm{OPT}(L)$. Másrészt a második lemma alapján $\mathrm{FF}(L) - 2 \le w(L)$, így azt kapjuk, hogy $FF(L) \le 1.7\mathrm{OPT}(L) + 2$, amiből következik, hogy az algoritmus aszimptotikusan 1.7-versenyképes.

Fontosnak tartjuk megjegyezni, hogy a fentiekben igazolt felső korlát éles, azaz a FF algoritmusra az is igaz, hogy az aszimptotikus versenyképességi hányadosa 17/10. ■

Érdemes megjegyeznünk, hogy számos az FF algoritmusnál kisebb versenyképességi hányadossal rendelkező algoritmus került kifejlesztésre. A jelenleg ismert legjobb algoritmus aszimptotikus versenyképességi hányadosa 1,5888.

Alsó korlátok

Ebben a részben azt vizsgáljuk, miként találhatunk általános alsó korlátokat a lehetséges versenyképességi hányadosokra. Elsőként egy egyszerű alsó korlátot tekintünk. Ezt követően megmutatjuk, miként általánosítható a bizonyítás alapgondolata egy általános módszerré.

23.16. tétel. *Nincs olyan on-line algoritmus a ládapakolási feladatra, amelynek az aszimptotikus versenyképességi hányadosa kisebb, mint 4/3.*

Bizonyítás. Legyen A egy tetszőleges on-line algoritmus. Tekintsük tárgyak következő sorozatát. Legyen $\varepsilon < 1/12$ és L_1 egy n darab $1/3+\varepsilon$ méretű tárgyból álló sorozat, L_2 pedig n darab $1/2+\varepsilon$ méretű tárgyból álló sorozat. Elsőként az algoritmus megkapja az L_1 listát. Ekkor az algoritmus bizonyos ládákba két tárgyat tesz, bizonyos ládákba egyet. Jelölje k azon ládák számát, amelyek két tárgyat tartalmaznak. Ekkor az algoritmus költsége $A(L_1) = k + n - 2k = n - k$. Másrészt az optimális off-line algoritmus minden ládába két tárgyat tesz, így a költség $OPT(L_1) = n/2$.

Amennyiben ugyanez az algoritmus az L_1L_2 összetett listát kapja, akkor az első részben szintén k ládát használ két tárgynak. (Az on-line algoritmus nem tudja, hogy az L_1 vagy az L_1L_2 lista alapján kapja a tárgyakat.) Következésképp az $1/2+\varepsilon$ méretű tárgyak közül csak $n-2k$ darabot párosíthat az előző tárgyakhoz, az összes többihez új ládát kell nyitnia. Tehát $A(L_1L_2) \geq n - k + (n - (n - 2k)) = n + k$. Másrészt az optimális off-line algoritmus minden ládába egy kisebb, $1/3+\varepsilon$ méretű és egy nagyobb, $1/2+\varepsilon$ méretű tárgyat tesz, így $OPT(L_1L_2) = n$.

Következésképpen azt kapjuk, hogy az A on-line algoritmusra van olyan L lista, amelyre

$$A(L)/OPT(L) \geq \max\left\{\frac{n-k}{n/2}, \frac{n+k}{n}\right\} \geq 4/3\ .$$

Másrészt a fenti hányadosokban az $OPT(L)$ érték legalább $n/2$, ami tetszőlegesen nagynak választható. Így a fenti egyenlőtlenségből adódik, hogy az A algoritmus aszimptotikus versenyképességi hányadosa legalább 4/3, amivel a tétel állítását igazoltuk. ∎

A fenti bizonyítás alapötlete, hogy egy hosszabb tárgysorozatot (a fentiekben L_1L_2) veszünk és az algoritmus viselkedésétől függően választjuk ki azt a kezdőszeletét a sorozatnak, amelyre a költségek hányadosa maximális. Természetes gondolat a bizonyításban használt sorozatnál bonyolultabb sorozatot használni. Több alsó korlát született különböző sorozatok felhasználásával. Másrészt a sorozatok elemzéséhez szükséges számítások egyre bonyolultabbak lettek. Az alábbiakban megmutatjuk, miként írható fel a sorozat elemzése vegyes egészértékű programozási feladatként, amely lehetővé teszi, hogy az alsó korlátot számítógép segítségével határozzuk meg.

Tekintsük a következő tárgysorozatot. Legyen $L = L_1L_2\ldots L_k$, ahol L_i $n_i = \alpha_i n$ egyforma méretű tárgyat tartalmaz, amelyek mérete a_i. Amennyiben egy A algoritmus aszimptotikusan C-versenyképes akkor minden j-re

teljesülnie kell a

$$C \geq \limsup_{n\to\infty} \frac{\mathrm{A}(L_1 \dots L_j)}{\mathrm{OPT}(L_1 \dots L_j)}$$

feltételnek. A fentiekben tekinthetjük azt az algoritmust, amelyre az általunk adható alsó korlát minimális, így célunk az

$$R = \min_{\mathrm{A}} \max_{j=1,\dots,k} \limsup_{n\to\infty} \frac{\mathrm{A}(L_1 \dots L_j)}{\mathrm{OPT}(L_1 \dots L_j)}$$

érték meghatározása, amely érték egy alsó korlát lesz a versenyképességi hányadosra. Ezen érték meghatározható egy vegyes egészértékű programozási feladat optimumaként. A feladat megadásához szükségünk van a következő fogalmakra.

Egy tetszőleges ládára a láda tartalma leírható a láda pakolási mintájával, amely azt adja meg, hogy az egyes észlistákból hány elemet tartalmaz a láda. A ***pakolási minta*** egy k-dimenziós $(p_1, \dots, p_k)$ vektor, amelynek a p_j koordinátája azt adja meg, hány elemet tartalmaz a láda az L_j részlistából. Pakolási minta olyan nemnegatív egész koordinátájú vektor lehet, amelyre a $\sum_{j=1}^{k} a_j p_j \leq 1$ feltétel teljesül. (Ez a feltétel azt írja le, hogy a minta által leírt tárgyak valóban elférnek egy ládában.)

Osztályozzuk a lehetséges minták T halmazát a következőképpen. Minden j-re legyen T_j azon minták halmaza, amelyeknek az első pozitív együtthatója a j-edik. (Egy p minta a T_j halmazba kerül, ha $p_i = 0$ minden $i < j$ esetén, és $p_j > 0$.)

Most tekintsük az A algoritmus által kapott pakolást. Az algoritmus minden ládát valamely pakolási minta alapján töltött meg, így az algoritmus által kapott pakolás leírható a pakolási minták segítségével. Jelölje $n(p)$ minden $p \in T$ esetén azon ládák számát, amely ládákat a p mintának megfelelően pakolt az algoritmus.

Vegyük észre, hogy egy láda, amely egy a T_j osztályba eső mintának megfelelően lett megtöltve, az első elemét az L_j részlistából kapja. Következésképpen azt kapjuk, hogy az algoritmus által az $L_1 \dots L_j$ részlista pakolása során kinyitott ládák száma a következőképpen adható meg az $n(p)$ értékekkel:

$$A(L_1, \dots, L_j \sum_{i=1}^{j} \sum p \in T_i n(p) \ .$$

Tehát egy adott n-re a keresett A a keresett A értéket a következő vegyes egészértékű programozási feladat megoldásával számíthatjuk ki.

Min R

$\sum_{p\in T} p_j n(p) = n_j, \qquad 1 \leq j \leq k$

$$\sum_{i=1}^{j} \sum_{p \in T_i} n_p \leq R \cdot OPT(L_1 \dots L_j), \qquad 1 \leq j \leq k$$
$$n(p) \in \{0, 1, \dots\}, \qquad p \in T \quad .$$

Az első k feltétel azt írja le, hogy az összes tárgyat el kell helyeznünk a ládákban. A második k feltetel az írja le, hogy az R érték valóban nem kisebb, mint az algoritmus költségének és az optimális költségnek a hányadosa a vizsgált részlistákra.

Az $L_1 L_2 \dots L_k$ lista alapján a pakolási minták T halmaza és az optimális $OPT(L_1 \dots L_j)$ értékek meghatározhatóak.

A feladatban a változók nagy értékeket vehetnek fel és a változók száma is nagy lehet, ezért a feladat helyett a lineáris programozási relaxációt szokás tekinteni. Továbbá a megoldást azon feltétel mellett kell meghatároznunk, hogy n tart a végtelenbe, és ezen feltétel mellett az egészértékű feladat és a relaxáció ugyanazokat a korlátokat adják.

Az eljárást megfelelően választott listákra alkalmazva igazolni lehet, hogy nincs olyan on-line algoritmus, amelynek kisebb az aszimptotikus versenyképességi hányadosa, mint 1.,5401. Hosszan, közel 20 éven át ez volt a legjobb ismert alsó korlát, de nemrég javították az eredményt, az alsó korlát értékét 1,54037-re növelték.

23.4.2. Többdimenziós modellek

A ládapakolási feladatnak három különböző többdimenziós általánosítása van, a vektorpakolási, a dobozpakolási és a sávpakolási modellek. Ezen általánosítások közül csak a sávpakolási feladattal foglalkozunk részletesen, a többi általínosításnak csak a modelljét ismertetjük. A ***vektorpakolási feladatban*** a bemenet d dimenziós vektorokból áll, és ezeket a tárgyakat kell minimális számú egységnyi kapacitású ládában szabályosan elhelyezni. Szabályosnak mondunk egy elhelyezést, ha az egy ládába tett vektorokra minden komponensben az értékek összege legfeljebb egy. Az on-line vektorpakolási feladatban a vektorok egyenként jönnek, és az egyes vektorokat a további vektorokra vonatkozó információ ismerete nélkül kell ládákhoz rendelnünk. A ***dobozpakolási feladatban*** a bemenet d dimenziós téglatestekből áll, és ezeket kell minimális számú d-dimenziós egységkockában elhelyeznünk. Az on-line dobozpakolási feladatban a téglatestek egyenként jönnek, és az egyes téglatesteket a további téglatestekre vonatkozó információk ismerete nélkül kell ládához rendelnünk.

On-line sávpakolás

A ***sávpakolási feladatban*** téglalapok egy halmaza adott a szélességükkel és magasságukkal, és a célunk az, hogy ezeket a téglalapokat elhelyezzük forgatások nélkül egy függőleges w szélességű sávba úgy, hogy minimalizáljuk

a felhasznált rész magasságát. A továbbiakban feltételezzük, hogy a tárgyak magassága legfeljebb 1. Általában az ütemezés megosztott erőforrásokkal két dimenziót eredményez, az erőforrást és az időt. Ebben az esetben tekinthetjük a szélességet a felhasznált erőforrás nagyságának, a magasságot pedig a felhasznált időnek, így célunk a felhasznált idő minimalizálása. A feladat on-line változatát vizsgáljuk, ahol a téglalapok egy listáról érkeznek, és a megérkezett téglalapot el kell helyeznünk a függőleges sávban a további téglalapokra vonatkozó ismeretek nélkül. Az on-line sávpakolási feladatra kidolgozott algoritmusok többsége a polc algoritmusok családjába tartozik. az alábbiakban ezt az algoritmuscsaládot ismertetjük.

Polc algoritmusok

Egy alapvető módszer a téglalapok pakolására az, hogy polcokat definiálunk és a téglalapokat ezekre a polcokra helyezzük el. ***Polcon*** a feltöltendő sávnak egy vízszintes részét értjük. A Polc algoritmus minden téglalapot egy polcra helyez. Miután az algoritmus kiválasztotta azt a polcot, amely a téglalapot tartalmazni fogja, az algoritmus a téglalapot elhelyezi a polcon annyira balra, amennyire lehetséges a már a polcon levő egyéb téglalapok átfedése nélkül. Tehát a téglalap érkezése után az eljárásnak két döntést kell hoznia. Az első döntés az, hogy az eljárás kialakít-e egy új polcot vagy sem. Ha új polcot alakítunk ki, meg kell határoznunk a polc magasságát is. Az újonnan kialakított polcokat mindig az előző polc tetejére helyezzük, az első polc a sáv legalján van. A második döntés, hogy az algoritmusnak ki kell választania azt a polcot, amelyre a téglalapot helyezi. A továbbiakban akkor mondjuk, hogy egy *téglalap elhelyezhető* egy polcon, ha a polc magassága nem kisebb a téglalap magasságánál és a polcon elég hely van ahhoz, hogy a téglalapot elhelyezzék rajta.

Egy eljárást vizsgálunk részletesen a fenti feladat megoldására. Ez az algoritmus a Baker és Schwarz által 1983-ban kifejlesztett NFS_r algoritmus. Az algoritmus egy $r < 1$ paramétertől függ. Az algoritmus minden j-re legfeljebb egy r^j magasságú aktív polcot tart fent és egy tárgy érkezése után a következő szabállyal definiálhatjuk.

A $p_i = (w_i, h_i)$ téglalap érkezése után válasszunk egy olyan k értéket, amelyre teljesül, hogy $r^{k+1} < h_i \leq r^k$. Amennyiben van r^k magasságú aktív polc, és a téglalap elhelyezhető ezen a polcon, akkor helyezzük el. Ellenkező esetben alakítsunk ki egy új r^k magasságú polcot, helyezzük el a téglalapot rajta, és a továbbiakban legyen ez a polc az r^k magasságú aktív polc (ha volt korábbi aktív polc, azt lezárjuk).

23.7. példa. Legyen $r = 1/2$. Legyen az első tárgy mérete $(w/2, 3/4)$. Ez a tárgy 1 magasságú polcra kerül. Ekkor létrehozunk egy 1 magasságú polcot a sáv legalján,

ez lesz az 1-magasságú aktív polc, és ennek a polcnak a bal sarkára helyezzük el a tárgyat. Legyen a következő tárgy mérete $(3w/4, 1/4)$. Ez a tárgy $1/4$ magasságú polcra kerül. Mivel nincs ilyen aktív polc, ezért létrehozunk egy $1/4$ magasságú polcot az előző 1 magasságú polc tetején, ez lesz az $1/4$ magasságú aktív polc, és ennek a polcnak a bal sarkára helyezzük el a tárgyat. Legyen a következő tárgy mérete $(3w/4, 5/8)$. Ez a tárgy ismét 1 magasságú polcra kerül. Mivel az 1 magasságú aktív polcon nem fér el, ezért azt lezárjuk, és létrehozunk egy új 1 magasságú polcot az előző $1/4$ magasságú polc tetején, ez lesz az 1 magasságú aktív polc, és ennek a polcnak a bal sarkára helyezzük el a tárgyat. Legyen a következő tárgy mérete $(w/8, 3/16)$. Ez a tárgy $1/4$ magasságú polcra kerül. Az $1/4$ magasságú aktív polcon még elfér a tárgy, ezért arra polcra rakjuk, annyira balra, amennyire lehetséges a második tárgy mellé.

NFS_r versenyképességére igazak az alábbi állítások.

23.17. tétel. *Az* NFS_r *algoritmus* $\left(\frac{2}{r} + \frac{1}{r(1-r)}\right)$*-versenyképes. Az* NFS_r *algoritmus aszimptotikusan* $2/r$*-versenyképes.*

Bizonyítás. Tekintsük téglalapoknak egy tetszőleges L listáját, jelölje H a legmagasabb téglalap magasságát. Mivel ekkor $\mathrm{OPT}(L) \geq H$ teljesül, ezért a tétel állításának igazolásához elegendő belátnunk, hogy erre a sorozatra

$$\mathrm{NFS}_r(L) \leq 2/r\mathrm{OPT}(L) + H/r(1-r) \; .$$

Jelölje H_A az L lista végén az aktív polcok összmagasságát, H_Z pedig a többi lezárt polc összmagasságát. Elsőként vizsgáljuk az aktív polcokat. Jelölje h a legmagasabb aktív polc magasságát. Ekkor az aktív polcok magasságai a hr^i értékeket vehetik fel és minden i-re legfeljebb egy aktív polc van hr^i magassággal. Tehát az aktív polcok összmagasságára

$$H_A \leq h \sum_{i=0}^{\infty} r^i = \frac{h}{1-r} \; .$$

Másrészt a $H > rh$ egyenlőtlenségnek is teljesülnie kell, hiszen különben a legmagasabb téglalap is elfért volna a legfeljebb rh magasságú polcokon és nem nyitottuk volna meg a h magasságú polcot. Következésképpen $H_A \leq H/r(r-1)$.

Most vizsgáljuk a lezárt polcokat. Vegyük egy tetszőleges i-re a hr^i magasságú polcokat, jelölje ezeknek a számát n_i. Minden ilyen lezárt S polcra a következő S' polc elsőként egy olyan elemet tartalmaz, amely már nem volt elhelyezhető, így a két egymást követő polcra az elhelyezett téglalapok teljes szélessége legalább w. Másrészt a hr^i magasságú polcokon minden tárgy magassága legalább hr^{i+1}, hiszen egyébként a tárgyat egy kisebb magasságú

polcra helyeznénk. Tehát párosítva a lezárt polcokat és használva az aktív hr^i magasságú polcot is, ha a lezárt polcok száma páratlan, azt kapjuk, hogy az ilyen polcokon elhelyezett tárgyak összterülete legalább $wn_ihr^{i+1}/2$. Következésképpen az összes téglalap összterülete legalább $\sum_{i=0}^{\infty} wn_ihr^{i+1}/2$, így $\mathrm{OPT}(L) \geq \sum_{i=0}^{\infty} n_ihr^{i+1}/2$. Másrészt a lezárt polcok összmagassága $H_Z = \sum_{i=0}^{\infty} n_ihr^i$, így azt kaptuk, hogy $H_Z \leq 2\mathrm{OPT}(L)/r$. Mivel $\mathrm{NFS}_r(L) = H_A + H_Z$, ezért a fentiek alapján adódik a kívánt egyenlőtlenség. ■

A fenti algoritmuson kívül további polc algoritmusokat is vizsgáltak a feladat megoldására. A fenti algoritmus alapgondolata, hogy az egyes polctípusokat ládaként fogjuk fel, és az adott polctípushoz rendelt tárgyakat az NF ládapakolási algoritmussal helyezzük el. Természetes gondolat más ládapakolási algoritmusok használata. A jelenlegi legjobb polc algoritmust Csirik és Woeginger fejlesztették ki 1997-ben, amely algoritmus a harmonikus ládapakolási algoritmust használja az adott polctípusokhoz rendelt tárgyak elhelyezésére.

Gyakorlatok

23.4-1. Tegyük fel, hogy egyetlen tárgy mérete sem lehet nagyobb, mint 1/3. Igazoljuk, hogy ezen feltétel mellett NF aszimptotikus versenyképességi hányadosa 3/2.

23.4-2. Igazoljuk, hogy amennyiben egy ládában minden elem mérete legfeljebb 1/3, akkor teljesül a 23.15. lemma.

23.4-3. Tegyük fel, hogy a tárgyak listája az $L_1L_2L_3$ lista, ahol L_1 n darab 1/2 méretű tárgyat, L_2 n darab 1/3 méretű tárgyat, L_3 pedig n darab 1/3 méretű tárgyat tartalmaz. Milyen pakolási minták alapján lehet feltölteni a ládákat? Melyik minták esnek a T_2 osztályba?

23.4-4. Tekintsük a sávpakolási feladat azon változatát, amelyben a tárgyak megnyújthatóak oly módon, hogy a tárgyak területe ne változzon meg. Igazoljuk, hogy ebben az esetben NFS_r azon változata, amely a téglalap polcra helyezése előtt a téglalapot megnyújtja úgy, hogy a polccal megegyező magasságú legyen, $2 + 1/(r(1-r))$-versenyképes.

23.5. On-line ütemezés

Az ütemezési feladatok elméletének igen nagy irodalma van. Az első on-line ütemezési eredmény tulajdonképpen Graham nevéhez kapcsolódik, aki 1966-ban elemezte a Listás ütemezés algoritmust. Mondhatjuk ezt annak ellenére, hogy Graham nem használta az on-line algoritmusok körében később elterjedt fogalomrendszert és az általa vizsgált algoritmust nem on-line algoritmusként vizsgálta, hanem heurisztikus algoritmusnak tekintette.

Speciálisan on-line ütemezési algoritmusokkal az 1990-es években kezdtek

el foglalkozni és azóta számos eredmény született. Ebben a részben csak olyan modellekkel foglalkozunk, ahol a cél a maximális befejezési időpont minimalizálása. Az ütemezési feladatokkal kapcsolatos fogalmakat és eredményeket ezen könyv első kötetében egy fejezet ismerteti, ezért itt csak az általunk az alábbiakban használt fogalmak definícióit elevenítjük fel.

Ütemezési feladatokban munkák végrehajtásait kell megterveznünk. Az általános modellben a munkadarabok több tevékenységből állhatnak és a tevékenységekhez kell meghatároznunk a gépeket, amelyeken és az időintervallumokat, amelyekben az egyes műveletek végrehajtandóak. A továbbiakban azt az egyszerűbb modellt vizsgáljuk, amelyben minden munka egyetlen műveletből áll. Tehát a feladatunk az, hogy a munkákhoz, amelyeknek ismerjük a megmunkálási idejét hozzárendeljük a gépet, amelyen a munkát végrehajtjuk és a megmunkálás kezdési és befejezési időpontját, amely időpontok különbsége szükségképpen a megmunkálási idő.

A gépek tekintetében három különböző modellel foglalkozunk. Amennyiben a munka megmunkálási ideje minden gépen ugyanannyi, akkor azonos párhuzamos gépekről beszélünk. Amennyiben a gépekhez hozzá van rendelve egy s_i sebesség, a munkáknak van egy p_j megmunkálási súlya és a j-edik munka megmunkálási ideje az i gépen p_j/s_i, akkor hasonló párhuzamos gépekről beszélünk. Végül, ha a j-edik munka megmunkálási ideje tetszőleges $P_j = (p_{j1}, \ldots, p_{jm})$ pozitív vektor lehet, ahol a munka megmunkálási ideje az i-edik gépen p_{ji}, akkor általános párhuzamos gépekről beszélünk.

Ütemezési feladatok esetén számos célfüggvényt szokás vizsgálni, ebben a fejezetben azt a modellt vizsgáljuk, amelyben a cél a maximális befejezési idő minimalizálása.

Az alfejezet első részében ismertetjük a két legelterjedtebb on-line ütemezési modellt, és a következő két fejezetben ezen modellekkel foglalkozunk.

23.5.1. On-line ütemezési modellek

Az alábbiakban definiáljuk a két legfontosabb on-line ütemezési modellt.

LISTA modell

Ebben a modellben a munkák egy listáról érkeznek. Amikor egy munkát megkapunk a listáról, akkor ismerjük meg a szükséges megmunkálási időt, ezt követően ütemeznünk kell a munkát, hozzárendelve a kezdési és a befejezési időt, amelyeket később már nem változtathatunk meg, és csak ezt követően kapjuk meg a listáról a következő munkát.

Vegyük észre, hogy amennyiben a célfüggvény a maximális befejezési idő, akkor (miként az off-line esetben is) elegendő olyan algoritmusokkal foglalkoznunk, amelyek nem hagynak üres részeket a gépeken, azaz amelyekben az

egyes gépeken a munkák szünet nélkül követik egymást. Ebben az esetben minden gépre a maximális befejezési idő megegyezik a géphez rendelt megmunkálási idők összegével. Minden gépre a gépen levő megmunkálási idők összegét a gépen levő ***töltésnek*** hívjuk.

23.8. példa. Tekintsük a LISTA modellben az azonos párhuzamos gépek esetét, legyen két gép és vegyük a következő munkasorozatot, ahol a munkákat a megmunkálási idők által adjuk meg: $I = (4, 3, 2, 5)$. Ekkor egy on-line algoritmus elsőként a 4 megmunkálási idővel rendelkező munkát kapja csak meg, hozzárendeli valamelyik géphez, tegyük fel, hogy az M_1 géphez. Ezt követően az algoritmus a 3 megmunkálási idővel rendelkező munkát kapja csak meg, és ezt hozzá kell rendelnie valamelyik géphez, tegyük fel, hogy az M_2 géphez. Majd az algoritmus a 2 megmunkálási idővel rendelkező munkát kapja csak meg, és ezt hozzá kell rendelnie valamelyik géphez, tegyük fel, hogy ismét az M_2 géphez. Végül az algoritmus az 5 megmunkálási idővel rendelkező munkát kapja csak meg, és ezt hozzá kell rendelnie valamelyik géphez, tegyük fel, hogy az M_1 géphez. Ekkor a gépeken a töltés $4 + 5$ illetve $3+2$ és valóban elérhető, hogy a töltések legyenek a maximális befejezési idők, hiszen az első gépen a munkákat végrehajthatjuk a $(0, 4)$ és $(4, 9)$ időintervallumokban, a második gépen a $(0, 3)$ és $(3, 5)$ időintervallumokban.

IDŐ modell

A második modellben a munkáknak egy r_j ***érkezési ideje*** is van, a munkáról semmit sem tudunk az érkezési ideje előtt, de a munka érkezése után bármikor megkezdhetjük a munka végrehajtását, ha van olyan gépünk, amely nem dolgozik. Amennyiben egy munkát elkezdünk végrehajtani, akkor nem szakíthatjuk félbe.

23.9. példa. Az IDŐ modellre tekintsünk egy két darab hasonló párhuzamos gépekből álló példát. Legyen az M_1 gép sebessége 1, az M_2 gép sebessége 2. Vegyük a következő $I = ((1, 0), (1, 1), (1, 1), (1, 1))$ munkasorozatot, ahol a munkák a (megmunkálási idő, érkezési idő) párokkal vannak megadva. Tehát a 0 időpontban egyetlen munka érkezik 1 megmunkálási idővel, az algoritmus elkezdheti végrehajtani valamely gépen, de várhat is, arra számítva, hogy később nagyobb megmunkálási idővel rendelkező munkák érkeznek. Tegyük fel, hogy az algoritmus az $1/2$ időpontig vár és akkor kezdi el végrehajtani a munkát az M_1 gépen. Az 1 időpontban megérkezik három újabb munka, ekkor csak az M_2 gép szabad. Kezdjük el végrehajtani az egyik $(1, 1)$ munkát a gépen. A $3/2$ időpontban válik szabaddá mindkét gép, ekkor mindkét géphez hozzárendelhetünk egy-egy munkát, amelyek az M_1 gépen az $5/2$ az M_2 gépen a 2 időpontban fejeződnek be, így az algoritmus költsége $5/2$. Látszik, hogy amennyiben egyből elkezdjük a 0 időpontban az első munka végrehajtását, akkor az algoritmus kisebb, 2 költséggel is ütemezhette volna a munkákat. Fontos megjegyeznünk, hogy ennek ellenére bizonyos esetekben hasznos lehet várakoztatni munkákat, ezzel helyett hagyva az esetleg később érkező nagyobb megmunkálási idővel rendelkező munkák számára.

23.5.2. A LISTA modell

Elsőként vegyük észre, hogy az ütemezésről szóló fejezetben tárgyalt listás ütemezés algoritmus valójában egy on-line algoritmus. Ez a LISTA algoritmus az aktuális tárgyat mindig ahhoz a géphez rendeli hozzá, ahol az aktuális töltés minimális. Az első kötetben igazolást nyert az az állítás, mely szerint az ütemezési feladat tetszőleges I bemenete esetén $\text{LISTA}(I) \leq (2-1/m)\text{OPT}(I)$,, amiből következik, hogy ezen algoritmus versenyképességi hányadosa $2-1/m$. Első ránézésre nehéz elképzelni más algoritmust az on-line esetre, de több algoritmust fejlesztettek ki, amelyek versenyképességi hányadosa 2-nél kisebb számhoz konvergál, amennyiben a gépek száma tart a végtelenhez. Ezen algoritmusok többsége azon az ötleten alapszik, hogy a gépek többségén igyekszik egyenletesen elosztani a munkákat, de a LISTA algoritmussal ellentétben a gépek egy részén alacsonyan tartja a töltést, biztonsági tartalékként fenntartva ezeket a gépeket az esetleges nagy megmunkálási idővel rendelkező munkáknak.

A továbbiakban az általánosabb eseteket – amelyekben a gépek nem azonosak – vizsgáljuk. Az általános modellben nyilvánvalóan nem elegendő azzal foglalkoznunk, melyik gépen minimális az aktuális töltés, hiszen azon a gépen nagyon nagy lehet a munka megmunkálási ideje. A LISTA algoritmus, amely mohó módon ütemezi a munkákat, a következőképpen általánosítható. A munkák helyét a következő szabály alapján határozzuk meg. Ütemezzük a munkát azon a gépen, ahol a töltés a munka ütemezése után minimális lesz. Ha több ilyen gép is van, akkor ezek közül azt választjuk, ahol a munka megmunkálási ideje a legkisebb és ha ilyen gépből is több van, akkor ezek közül a legkisebb indexű gépet választjuk. Ezt az algoritmust MOHÓ algoritmusnak nevezzük.

23.10. példa. Tekintsük a hasonló párhuzamos gépek esetét, ahol 3 darab gép van és a sebességek $s_1 = s_2 = 1$, $s_2 = 3$. Vegyük a következő $I = (2, 1, 1, 3, 2)$ munkasorozatot, ahol a munkák a megmunkálási idők által vannak megadva. Ekkor az első munka $2/3$-kor fejezhető be az M_3 gépen és az 1 időpontban a többi gépen, így M_3-hoz rendeljük. A következő munka az 1 időpontra fejezhető be az összes gépen, így M_3 kapja, mivel ott a legkisebb a megmunkálási idő. A következő munka az 1 időpontra fejezhető be M_1-en és M_2-n, és a $4/3$ időpontban az M_3 gépen, így M_1 kapja. A negyedik munka M_1-en a 4, M_2-n a 3 időpontban fejeződne be, az M_3-on a 2 időpontban, így M_3-ra kerül. Végül az utolsó munka M_1-en 3, M_2-n a 2 időpontban fejeződne be, az M_3-on $8/3$, így M_2-re kerül.

23.11. példa. Tekintsük az általános párhuzamos gépek esetét, ahol 2 darab gép van. Vegyük a következő $I = ((1, 2), (1, 2), (1, 3), (1, 3))$ munkasorozatot, ahol a munkák a megmunkálási idővektorok által vannak megadva. Ekkor az első munka

az 1 időpontban fejezhető be az M_1 gépen és a 2 időpontban a második gépen, így M_1-hez rendeljük. A következő munka a 2 időpontra fejezhető be mindkét gépen, így M_1 kapja. A következő munka a 3 időpontra fejezhető be mindkét gépen, így ismét M_1 kapja. Végül az utolsó munka a 4 időpontra fejezhető be az M_1 gépen és a 3 időpontra fejezhető be az M_2 gépen, így az M_2 géphez rendeljük.

Az algoritmus versenyképességi hányadosát határozzák meg az alábbi tételek.

23.18. tétel. *A* Mohó *algoritmus versenyképességi hányadosa m az általános párhuzamos gépek esetében*

Bizonyítás. Elsőként megmutatjuk, hogy az algoritmus versenyképességi hányadosa nem lehet kisebb, mint m. Tekintsük a következő munkasorozatot. Legyen $\varepsilon > 0$ egy kicsi szám. A sorozat m darab munkát fog tartalmazni. Az első munkára a megmunkálási idő az első gépen 1, az m-edik gépen $1+\varepsilon$, a többi gépen ∞, $(p_1(1) = 1, p_1(i) = \infty, i = 2, \ldots, m-1, p_1(m) = 1+\varepsilon)$, ezt követően a j-edik munkára a megmunkálási idő j a j-edik gépen, $1+\varepsilon$ a $j-1$-edik gépen, és ∞ a többi gépen $(p_j(j-1) = 1+\varepsilon, p_j(j) = j, p_j(i) = \infty$, ha $i \neq j-1$ és $i \neq j)$.

Ezen munkasorozatra a Mohó algoritmus a j-edik munkát a j-edik gépen ütemezi, és a maximális befejezési idő m. Másrészt az optimális off-line algoritmus az első munkát az m-edik gépen, utána a j-edik munkát a $(j-1)$-edik gépen ütemezi, így az optimális maximális befejezési idő $1+\varepsilon$. A hányados $m/(1+\varepsilon)$. Ez az érték m-hez tart, ha az ε érték 0-hoz tart, amivel az állítást igazoltuk.

Most megmutatjuk, hogy az algoritmus m-versenyképes. Tekintsünk egy tetszőleges munkasorozatot, legyen az optimális maximális befejezési idő L^*, továbbá legyen $L(k)$ az első k munka Mohó algoritmus általi ütemezésében a maximális befejezési idő. Mivel az i-edik munka ütemezéséhez valamely gépen legalább $\min_j p_i(j)$ idő szükséges, és egyik gépen sem használhatunk L^*-nál több időt, ezért $mL^* \geq \sum_{i=1}^{n} \min_j p_i(j)$.

Most igazoljuk teljes indukcióval, hogy $L(k) \leq \sum_{i=1}^{k} \min_j p_i(j)$. Mivel az első munkát ahhoz a géphez rendeljük, ahol a leghamarabb végrehajtható, ezért az állítás $k = 1$-re igaz. Most legyen $1 \leq k < n$ és tegyük fel, hogy az állítás k-ra igaz. Tekintsük a $k+1$-edik munkát. Legyen az l-edik gép, amelyre a munka megmunkálási ideje minimális. Ezen a gépen végrehajtva a munkát a megmunkálási idő legfeljebb $L(k) + p_{k+1}(l) \leq \sum_{i=1}^{k+1} \min_j p_i(j)$ (az indukciós feltevés alapján).

Mivel a Mohó algoritmus által kapott maximális befejezési idő legfeljebb annyi, mint abban az esetben, ha a $k+1$-edik munkát az l-edik géphez rendeljük, ezért $L(k+1) \leq \sum_{i=1}^{k+1} \min_j p_i(j)$, azaz az állítást igazoltuk $k+1$-re,

így tetszőleges n-nél nem nagyobb egészre.

Következésképpen azt kapjuk, hogy $mL^* \geq \sum_{i=1}^{n} \min_j p_i(j) \geq L(n)$, amivel igazoltuk, hogy az algoritmus m-versenyképes. ■

A hasonló párhuzamos gépek esetének vizsgálatához tekintsünk egy tetszőleges bemenetet. Legyen L az algoritmus által kapott maximális befejezési idő, L^* pedig az optimális off-line algoritmus által kapott maximális befejezési idő. Az elemzés az alábbi lemmákon alapul, amelyek az egyes gépeken levő töltés mennyiségére adnak alsó korlátot.

23.19. lemma. *A töltés a leggyorsabb gépen legalább $L - L^*$.*

Bizonyítás. Vegyük azt a munkát, amelynek a befejezési ideje megegyezik a maximális befejezési idővel. Ha a munka a leggyorsabb gépre van ütemezve, akkor az állítás nyilvánvalóan teljesül. Tegyük fel, hogy nem a leggyorsabb gépen ütemeztük. Mivel az optimális ütemezés költsége L^*, ezért ezen munkának a leggyorsabb gépen történő végrehajtása legfeljebb L^* ideig tarthat. Másrészt ezt a munkát úgy ütemeztük, hogy a befejezési ideje L lett, ami azt jelenti, hogy a munka ütemezésekor a töltésnek a leggyorsabb gépen legalább $(L - L^*)$-nak kellett lennie, hiszen különben a munkát ott ütemeztük volna. ■

23.20. lemma. *Amennyiben a töltés minden legalább v sebességű gépen legalább l, akkor a töltés legalább $l - 4L^*$ minden legalább $v/2$ sebességű gépen.*

Bizonyítás. Amennyiben $l < 4L^*$, az állítás nyilvánvalóan teljesül. Tegyük fel, hogy $l \geq 4L^*$. Tekintsük azon munkák halmazát, amelyeket a legalább v sebességű gépeken ütemezünk az $[l - 2L^*, l]$ intervallumban. Ezen munkák összsúlya nem lehet kevesebb, mint $2L^*$-szor a legalább v sebességű gépek sebességeinek az összege, következésképpen van olyan munka közöttük, amelyet az optimális off-line algoritmus lassabb gépen ütemez (hiszen ellenkező esetben nem lehetne az off-line maximális befejezési idő L^*).

Legyen egy ilyen munka j. Mivel v-nél lassabb gépen ütemezi az off-line algoritmus, ezért a megmunkálási súlya legfeljebb vL^*. Következésképpen ezen munka megmunkálási ideje a legalább $v/2$ sebességű gépeken legfeljebb $2L^*$. Mivel ezen munka befejezés iideje a Mohó algoritmus mellett legalább $l - 2L^*$, ezért a munka ütemezésekor minden legalább $v/2$ sebességű gépen a töltés legalább $l - 4L^*$ volt, hiszen ellenkező esetben egy ilyen gépen ütemeztük volna a munkát. ■

A fenti lemmák alapján igazolhatjuk a következő állítást.

23.21. tétel. *Hasonló párhuzamos gépek esetén a* Mohó *algoritmus versenyképességi hányadosa* $\Theta(\lg m)$.

Bizonyítás. Elsőként megmutatjuk, hogy az algoritmus $O(\lg m)$-versenyképes. Ehhez vegyünk egy tetszőleges bemenetet. Legyen L az algoritmus által kapott maximális befejezési idő, L^* pedig az optimális off-line algoritmus által kapott maximális befejezési idő.

Legyen $v_{\max}$ a leggyorsabb gép sebessége, ekkor ezen a gépen a 23.19. lemma alapján a töltés legalább $L - L^*$. Ezt követően többször alkalmazva a 23.20. lemmát azt kapjuk, hogy a legalább $v_{\max}2^{-i}$ sebességű gépeken a töltés legalább $L - L^* - 4iL^*$. Következésképp a legalább $v_{\max}/m$ sebességű gépeken a töltés legalább $L - (1 + 4\lceil \lg m \rceil)L^*$. Jelölje I a legfeljebb $v_{\max}/m$ sebességű gépek halmazát.

Vizsgáljuk most meg a munkák megmunkálási súlyainak W összegét. Mivel az off-line algoritmus úgy osztja el a munkákat, ogy a maximális töltés L^*, és mivel legfeljebb m gép van, amelyeknek a sebessége kisebb, mint $v_{\max}/m$ ezért

$$W \leq L^* \sum_{i=1}^{m} v_i \leq mL^* v_{\max}/m + L^* \sum_{i \notin I} v_i \leq 2L^* \sum_{i \notin I} v_i \ .$$

Másrészt az on-line algoritmus ezeket a munkákat osztja szét, ezért nem lehetséges, hogy minden I-n kívül eső gépen a töltés $2L^*$-nál nagyobb legyen, hiszen ez a fenti felső korlátnál nagyobb W értéket eredményezne.

Következésképp azt kapjuk, hogy

$$L - (1 + 4\lceil \lg m \rceil)L^* \leq 2L^* \ ,$$

amiből adódik, hogy $L \leq 3 + 4\lceil \lg m \rceil)L^*$, azaz igazoltuk, hogy az algoritmus $O(\lg m)$-versenyképes.

Most megmutatjuk, hogy az algoritmus versenyképességi hányadosa nem lehet kisebb, mint $\Omega(\lg m)$. Tekintsük a következő halmazát a gépeknek. A G_0 halmazban van egy gépünk, amelynek a sebessége 1, a G_1 halmazban van 2 gépünk, amelyek sebessége $1/2$, így folytatva a G_i halmazban olyan gépeink vannak, amelyek sebessége 2^{-i}. A halmaz elemszámát úgy definiáljuk, hogy a benne levő gépek annyian legyenek, ahány 2^{-i} súlyú munka végrehajtható a $G_0, G_1, \ldots, G_{i-1}$ halmazban levő gépeken összesen 1 egységnyi idő alatt. Formálisan $|G_i| = \sum_{j=0}^{i-1} |G_j| 2^{i-j}$. Definiáljunk $k+1$ ilyen halmazt. Egyszerű számolás alapján adódik, hogy ekkor $|G_i| = 2^{2i-1}$, ha $i \geq 1$, így a gépek száma $1 + (2/3)(4^k - 1)$.

Tekintsük a következő munkasorozatot. Elsőként az első fázisban érkezzen $|G_k|$ darab munka 2^{-k} súllyal, majd a második fázisban $|G_{k-1}|$ munka $2^{-(k-1)}$ súllyal, és így folytatva az i-edik fázisban $|G_i|$ munka 2^{-i} súllyal, egészen az utolsó, $k+1$-edik fázisig, ahol egy munka érkezik 1 súllyal. Egy off-line algoritmus ütemezheti az i-edik fázis munkáit a G_{k+1-i} géphalmaz gépein, ekkor a maximális befejezési idő 1, így az optimális off-line költség legfeljebb

1.

Vizsgáljuk meg a MOHÓ algoritmus viselkedését ezen a bemeneten. A G_i halmaz elemszámának definíciója alapján az első fázisban érkező munkák végrehajthatóak a $G_0, \ldots, G_{k-1}$ halmazba eső gépeken 1 idő alatt, és a G_k halmaz gépein is 1 ideig tart végrehajtani a fázisban érkező munkákat, ezért ezeket a munkákat az algoritmus a $G_0, \ldots, G_{k-1}$ halmaz gépein hajtja végre, azokon a gépeken utána 1 lesz a töltés, a G_k halmaz gépein 0. Ezt követően a második fázis munkáit az algoritmus a $G_0, \ldots, G_{k-2}$ halmazok gépein hajtja végre, a harmadik fázis munkáit a $G_0, \ldots, G_{k-3}$ halmazok gépein, végül az utolsó előtti és az utolsó fázis munkáját a G_0 halmazba eső gépen. Tehát a MOHÓ algoritmus költsége $k+1$, (ez a maximális befejezési idő a G_0-ba eső gépen), és mivel $k = \Omega(\lg m)$, ezért az állítást igazoltuk. ■

23.5.3. Az IDŐ modell

Ebben a modellben egyetlen algoritmust elemzünk, amelynek alapötlete, hogy a munkákat az érkezési idők alapján szétosztja és az egyes munkahalmazokat optimálisan a munkák ismeretében off-line módon ütemezi. Ezt az algoritmust ***intervallumonkénti ütemező algoritmusnak*** nevezzük és INTV algoritmusként jelöljük. Legyen t_0 az első munka érkezési ideje, $i = 0$ és ettől az időponttól kezdve az algoritmus a következő pszeudokóddal írható le:

INTV(I)

```
1  while a munkasorozatnak nincs vége
2        legyen H_i a t_i időpontig megérkezett és ütemezetlen munkák
         halmaza
3        legyen OFF_i ezen munkákra egy optimális off-line ütemezés
4        rendeljük hozzá a munkákat a gépekhez a kapott ütemezésnek
         megfelelően
5        legyen q_i a kapott maximális befejezési idő
6        if érkezett a (t_i, q_i] intervallumban új munka
            vagy vége van a munkasorozatnak
7           t_{i+1} = q
8        else legyen t_{i+1} a következő munka érkezési ideje
9        i = i + 1
```

23.12. példa. Tekintsük két párhuzamos gép esetét. Legyen a munkák sorozata a következő $I = (1,0), (1/2,0), (1/2,0), (1,3/2), (1,3/2), (2,2)$, ahol a munkákat a (megmunkálási idő, érkezési idő) párral adtuk meg. Az első iterációs részben az első három munkát ütemezzük, egy optimális ütemezés az első munkát az első géphez,

a második és harmadik munkákat a második géphez rendeli, és az 1 időpontban befejezi a munkák végrehajtását. Utána, mivel nem érkezett új munka és nincs vége a sorozatnak, várunk egészen a 3/2 időpontig. Ekkor egy új iterációs rész kezdődik, az algoritmus ütemezi a negyedik és ötödik munkákat az első és második gépen, a munkákat az 5/2 időpontra fejezi be. Ekkor elkezdődik a harmadik iterációs rész, és az algoritmus ütemezi a hatodik munkát valamely gépen, az $[5/2, 9/2]$ intervallumra.

A INTV algoritmus versenyképességére vonatkozik a következő állítás.

23.22. tétel. *Az IDŐ modellben az* INTV *algoritmus 2-versenyképes.*

Bizonyítás. Vegyük az algoritmus által kapott ütemezését a munkáknak. Jelölje i az iterációk számát. Legyen $T_3 = t_{i+1} - t_i$, $T_2 = t_i - t_{i-1}$, $T_1 = t_{i-1}$ és T_{OPT} az optimális off-line költség. Ekkor $T_2 \leq T_{OPT}$. Ez az észrevétel a $t_{i+1} \neq q_{i+1}$ esetben nyilvánvaló. Ha $t_{i+1} = q_{i+1}$, akkor azért teljesül, mert az i-edik iterációban ütemezett munkákat az optimális algoritmusnak is ütemezni kell. Másrészt $T_1 + T_3 \leq T_{OPT}$. Ezen észrevétel igazolásához vegyük észre, hogy az i-edik iterációban ütemezett munkák mindegyikének legalább $T_1 = t_{i-1}$ az érkezési ideje, (ellenkező esetben már az $i-1$-edik lépésben ütemeztük volna őket). Következésképp az optimális algoritmus nem ütemezheti ezeket a munkákat a T_1 időpont előtt. Másrészt a munkák végrehajtásához legalább további T_3 idő kell. Mivel az algoritmus által kapott ütemezés költsége $T_1 + T_2 + T_3$, ezért a tétel állítása következik. ∎

Később az algoritmusnál kisebb versenyképességi hányadossal rendelkező algoritmusokat is sikerült kifejleszteni, Vestjens igazolta, hogy az ***on-line* LPT** algoritmus, amely minden időpontban, amikor szabad használható gép van, a már megérkezett de még nem ütemezett munkák közül a legnagyobb megmunkálási idővel rendelkező munkát kezdi el végrehajtani a gépen, (3/2)-versenyképes. Szintén Vestjens igazolta az alábbi állítást.

23.23. tétel. *Nincs olyan on-line algoritmus az on-line ütemezés IDŐ modelljében a maximális befejezési idő minimalizálására, amelynek a versenyképességi hányadosa kisebb, mint 4/3.*

Bizonyítás. Legyen $\alpha = 0.3473$ az $\alpha^3 - 3\alpha + 1 = 0$ egyenlet $[1/3, 1/2]$ intervallumba eső megoldása. Igazoljuk, hogy egyetlen on-line algoritmusnak se lehet kisebb a versenyképességi hányadosa, mint $1 + \alpha$. Vegyünk egy tetszőleges on-line algoritmust, jelölje ALG. Tekintsük a következő munkasorozatot.

A 0 időpontban egyetlen munka érkezik, amelynek a megmunkálási ideje 1. Legyen S_1 az az időpont, amelyben az algoritmus elkezdi a munkát végrehajtani valamely gépen. Ha $S_1 > \alpha$, akkor erre az egyetlen munkából álló bemenetre $\mathrm{ALG}(I)/\mathrm{OPT}(I) > 1 + \alpha$, amivel az állítást igazoljuk. Tehát feltehetjük, hogy $S_1 \leq \alpha$.

A következő munka érkezzen az S_1 időpontban, a megmunkálási ideje legyen $\alpha/(1-\alpha)$. Legyen a kezdési ideje S_2. Amennyiben $S_2 \leq S_1 + 1 - \alpha/(1-\alpha)$, akkor a munkasorozatot $m-1$ darab munkával fejezzük be, amelyek mindegyikének az érkezési ideje S_2, a megmunkálási ideje $1+\alpha/(1-\alpha)-S_2$. Ekkor egy optimális off-line algoritmus az első két munkát ugyanazon a gépen ütemezi, a maradék $m-1$ munka mindegyikét pedig egy-egy gépen az S_2 időpontban elkezdve, így az optimális maximális befejezési idő $1+\alpha/(1-\alpha)$. Másrészt az on-line algoritmus esetében az utolsó $m+1$ munkából legalább egyet csak az első vagy a második munka befejezése után kezdhetünk el, így erre a bemenetre $\mathrm{ALG}(I) \geq 1 + 2\alpha/(1-\alpha)$, amiből adódik, hogy ebben az esetben sem lehet az algoritmus versenyképességi hányadosa kisebb, mint $1+\alpha$. Következésképpen feltehetjük, hogy $S_2 > S_1 + 1 - \alpha/(1-\alpha)$.

Ekkor az $S_1 + 1 - \alpha/(1-\alpha)$ időpontban $m-2$ darab munka érkezik, amelyeknek a megmunkálási ideje $\alpha/(1-\alpha)$ és egy további munka, amelynek a megmunkálási ideje $1-\alpha/(1-\alpha)$. Az optimális ütemezésben a második és utolsó munka kivételével, minden munkát külön gépen hajtunk végre, a második és az utolsó munkát ugyanazon a gépen és így egy olyan ütemezést kapunk, amelyben a maximális befejezési idő $1+S_1$. Mivel az $S_1+1-\alpha/(1-\alpha)$ időpont előtt az utolsó m munka egyikét sem kezdte el az on-line algoritmus, ezért van olyan gép, amelyen ezután az időpont után legalább két munkát kell végrehajtania, és ezen a gépen a maximális befejezési idő legalább $S_1 + 2 - \alpha/(1-\alpha)$ lesz. Mivel $S_1 \leq \alpha$, ezért az $\mathrm{OPT}(I)/\mathrm{ALG}(I)$ hányados az $S_1\alpha$ érték mellett minimális, és ebben az esetben a hányados $1+\alpha$, amivel az állítást igazoltuk. ■

Gyakorlatok

23.5-1. Igazoljuk, hogy nincs olyan on-line algoritmus, amelynek két azonos párhuzamos gép esetén kisebb a versenyképességi hányadosa, mint $3/2$.

23.5-2. Igazoljuk, hogy a LISTA ütemezési algoritmus nem konstans-versenyképes az általános párhuzamos gépek esetében.

23.5-3. Igazoljuk, hogy amennyiben az INTV algoritmusnál nem határozzuk meg minden lépésben az optimális off-line ütemezést, hanem helyette csak egy c-közelítő megoldást használunk, amelynek a költsége legfeljebb c-szer az optimális költség, akkor az így kapott változat $2c$-versenyképes.

Feladatok

23-1 Lapozási feladat

Vegyük a k-szerver feladat azon speciális esetét, ahol bármely két pont távolsága 1. (Ez a feladat ekvivalens az on-line lapozási feladattal.) Tekintsük

azt az algoritmust, amely minden esetben, ha a kérés helyén nincs szerver, azt a szervert küldi a kérés kiszolgálására, amely szerver a legrégebben volt használva. (Ez az algoritmus a lapozásnál használt LRU algoritmusnak felel meg.) Igazoljuk, hogy az algoritmus k-versenyképes.

23-2 Az* ÉBRESZT2 *algoritmus
Vizsgáljuk meg a nyugtázási feladatra a következő ÉBRESZT2 algoritmust, ahol az általános algoritmusban szereplő e_j értékekre $e_j = 1/|\sigma_j|$. Igazoljuk, hogy ez az algoritmus nem konstans-versenyképes.

23-3 Ládapakolási alsó korlát
Igazoljuk, hogy nincs olyan on-line ládapakolási algoritmus, amelynek kisebb az aszimptotikus hányadosa, mint 3/2 egy olyan listát használva, amely $1/7+\varepsilon$, $1/3+\varepsilon$, $1/2+\varepsilon$ méretű tárgyakat tartalmaz, ahol ε egy kicsi pozitív szám.

23-4 Sávpakolás nyújtható téglalapokkal
Tekintsük a sávpakolási feladat azon változatát, amelyben a téglalapok megnyújthatóak a terület megváltoztatása nélkül. Adjunk 4-versenyképes algoritmust a feladat megoldására.

23-5 On-line* LPT *algoritmus
Tekintsük az IDŐ modellben azt az algoritmust, amely minden időpontban, amikor egy gép szabaddá válik, azt a munkát ütemezi a megérkezett munkák közül, amelynek a legnagyobb a megmunkálási ideje. Ezt az algoritmust on-line LPT algoritmusnak nevezzük.Igazoljuk, hogy az algoritmus 3/2-versenyképes.

Megjegyzések a fejezethez

Az on-line algoritmusok területéhez kapcsolódó eredmények részletes összefoglalásai találhatóak a [34, 79] könyvekben.

A k-szerver feladatra vonatkozó első eredményeket, a 23.1. és 23.2. tételeket Manasse, McGeoch és Sleator publikálták a [191] cikkben. Az egyenesre, mint speciális metrikus térre vonatkozó eredmény a 23.3. tétel Chrobak, Karloff, Payne és Viswanathan [48] cikkéből származik, később Chrobak és Larmore általánosították az algoritmust fák esetére is [46]. Az első konstans-versenyképes algoritmust az általános feladatra Fiat, Rabani és Ravid [78] fejlesztették ki, a legjobb jelenleg ismert eljárás a munkafüggvényen alapuló $(2k-1)$-versenyképes algoritmus, amelyet Chrobak és Larmore fejlesztett ki az [47] cikkben és Koutsoupias és Papadimitriou elemeztek a [159] cikkben. Bittner, Imreh és Nagy-György a munkafüggvény algoritmus viselkedését vizsgálta arra az általánosabb modellre, amelyben az igényeket vissza lehet utasítani [31].

A nyugtázás modelljét Dooly, Goldman, és Scott vetették fel a [65] cikkben innen származnak a 23.5. és 23.6. tételek. A feladat egy további modelljét vizsgálja Albers és Bals [2], a feladatra kidolgozott véletlenített algoritmusokat vizsgálja Karlin Kenyon és Randall [148]. A lapletöltési feladat megoldására a HÁZIÚR algoritmust Young [319] dolgozta ki. Az on-line forgalomirányítás területének részletes tárgyalása megtalálható Leonardi [180] összefoglaló cikkében. Az exponenciális algoritmust töltéselosztási modellre Aspnes, Azar, Fiat, Plotkin és Waarts [7] vizsgálja, az itt ismertetett exponenciális algoritmust Awerbuch, Azar és Plotkin alkalmazták a nyereségmaximalizáló modellre a [12] cikkben.

A ládapakolás elméletét tekinti át Csirik és Woeginger [56] cikke. Az NF és FF algoritmusokat a versenyképességi elemzés alapján Johnson, Demers, Ullman, Garey és Graham elemezték a [139, 140] cikkekben, részletesebb elemzés található Johnson [138] doktori értekezésében. Az on-line ládapakolási algoritmusok lehetséges versenyképességi hányadosára vonatkozó alsó korlátokat meghatározó pakolási minták módszerét van Vliet [309, 308] dolgozta ki. A legjobb ismert alsó korlátot Balogh, Békési és Galambos igazolták [19]. Az on-line sávpakolási feladatra az NFS_r algoritmust Baker és Schwarz [17] fejlesztették ki és elemezték. Később egyéb polcok meghatározásán alapuló algoritmusok is kifejlesztésre kerültek, a legkisebb versenyképességi hányadossal rendelkező polcokon alapuló algoritmust az on-line sávpakolási feladatra Csirik és Woeginger [57] fejlesztették ki.

Az on-line ütemezés területén elért eredmények részletes összefoglalója található Sgall [264] cikkében. Az első on-line eredmény a LISTA algoritmus elemzése Graham [101] cikkében szerepel, azóta számos eredmény született az azonos gépek esetére, jelenleg a legkisebb versenyképességi hányadossal a Fleischer és Wahl által kifejlesztett, [84] cikkben publikált algoritmus rendelkezik, melynek versenyképességi hányadosa az 1.9201 számhoz tart, ahogy a gépek száma tart a végtelenbe. A hasonló párhuzamos gépek esetén a MOHÓ algoritmus versenyképességi hányadosára alsó korlátot Cho és Sahni [45] adott, a felső korlátot, az általános párhuzamos gépek elemzését és a forgalomirányításnál is használt exponenciális függvényt használó algoritmusok elemzését tartalmazza Aspnes, Azar, Fiat, Plotkin és Waarts [7] cikke. Később további fejlesztésekkel sikerült javítani az exponenciális függvényt használó algoritmusok versenyképességi hányadosán, a kapcsolódó eredmények megtalálhatók Azar [13] cikkében. Az IDŐ modellben ismertetett 23.22. tétel Shmoys, Wein és Williamson [267] cikkén alapul. Az LPT algoritmus elemzése, az IDŐ modellre vonatkozó alsó korlátok és a modellre vonatkozó eredmények részletes tárgyalása megtalálhatóak Vestjens [324] doktori értekezésében. Ebben a fejezetben csak egy on-line ütemezési modellt ismertettünk; ezen modell

érdekes általánosítása, amelyben a gépek száma nem adott, hanem meg kell őket vásárolni – ezt a modellt a [126] és [66] cikkekben vizsgálták.

A *23-1*. feladat a [271], a *23-2*. feladat a [65], a *23-3*. feladat a [316], a *23-4*. feladat a [125] és végül a *23-5*. feladat a [324] cikken alapul.

A fejezet megírását támogatta az OTKA F048587 számú pályázata.

24. Belsőpontos algoritmusok

A lineáris programozás – napjainkban is – számos területen a legjobb modellezési eszközt biztosítja. Gazdasági, ipari, logisztikai és tudományos kérdések sokaságát lehet pontosan (vagy közelítőleg) lineáris egyenletekkel és egyenlőtlenségekkel modellezni.

A lineáris programozás hosszú múltra tekint vissza. Az egyik alapvető és napjainkban legtöbbször hivatkozott eredménye Farkas Gyula 1894-ben közölt lemmája. Farkas Gyula dolgozata a Fourier-féle mechanikai elv alkalmazásairól szólt. A lineáris programozás fejlődése során később is előfordult, hogy gyakorlati feladatok megoldása késztette a kutatókat újabb és újabb algoritmusok bevezetésére, illetve elméleti eredmények igazolására. Dantzig első – lineáris programozással kapcsolatos – közleménye csak 1948-ban jelent meg, annak ellenére, hogy a ***szimplex algoritmust*** már 1947-ben megfogalmazta és gyakorlati feladatok megoldására is kipróbálták. Egyes tudósok visszaemlékezései alapján tudjuk, hogy mind a brit, mind pedig az amerikai hadseregben használtak operációkutatási (lineáris programozási) módszereket a II. világháború alatt szállítási, logisztikai és más hasonló feladatok megoldására.

Hacsián[1] 1979-ben publikálta az ún. ***ellipszoid módszerét,*** amely akkoriban az elméletileg leghatékonyabb (polinomiális) módszer volt. Átlagos futási ideje azonban a szimplex algoritmusénál lényegesen nagyobb. Karmarkar 1984-ben közölte ***projektív skálázású primál algoritmusát,*** amellyel elindította a belsőpontos algoritmusok fejlődésének az évtizedét. Számos, az 50-es és 60-as években bevezetett és akkoriban a gyakorlat szempontjából használhatatlan, lineáris programozási algoritmust (logaritmikus barrier módszer, affin skálázású algoritmus) fedeztek fel újra és tökéletesítettek. A 90-es évek közepére, a kor számítástechnikai lehetőségeit maximálisan kihasználva, a belsőpontos algoritmusok hatékony és numerikusan stabil számítógépes megvalósításait fejlesztették ki, amelyek napjaink optimalizálási szoftvereinek (CPLEX, XPRESS-MP stb.) nélkülözhetetlen részét képezik.

Ebben a fejezetben az a célunk, hogy megfogalmazzuk a lineáris programozás néhány primál-duál belsőpontos algoritmusát és elemezzük azok elméleti hatékonyságát. Ezt megelőzően röviden összefoglaljuk az ehhez szük-

[1] Hacsián nevének többféle írásmódjával találkozhatunk a szakirodalomban. A fejezetben a magyar megfelelőjét használjuk, ám az irodalomjegyzékben a Khacijan átírással találkozik az Olvasó, mivel az angol szakirodalomban ez az elfogadott.

séges lineáris programozási ismereteket. A fejezetben bemutatásra kerülő ismeretek megértéséhez elemi lineáris algebrai és analízisbeli ismeretekre lesz szükség. Természetesen az algoritmusok egy olyan variánsát is ismertetjük (nagy lépéses primál-duál logaritmikus barrier módszer), amelyet leggyakrabban valósítottak meg az elmúlt években.

A fejezet olvasásához a következő útmutatást nyújtjuk. Feltételezzük, hogy az Olvasó egyetemi szintű ismeretekkel rendelkezik lineáris algebrából és valós analízisből. A konkrét belsőpontos algoritmusokról szóló és a megvalósítást segítő megjegyzéseket tartalmazó 24.3.1.–24.3.4. alfejezetek önmagukban is érthető egységet alkotnak. Így azon Olvasók, akik a belsőpontos algoritmusok hatékony változatát szeretnék a lehető leggyorsabban megismerni, ezen alfejezetek olvasására szorítkozhatnak. Akik szeretnének az elméleti alapokkal is, a dualitás elmélettel és a centrális út elméletével megismerkedni, a 24.1. alfejezet alapján betekintést kapnak ebbe. Azon Olvasók, akik a belsőpontos algoritmusok egy egyszerű változatával, teljes lépésszám elemzéssel, az algoritmusok lépésszámára adott elméleti felső korlát levezetésével kívánnak megismerkedni, valamint a kerekítési eljárás iránt érdeklődőknek, ami leírja, hogy közelítő megoldásból miként kaphatunk pontos optimális megoldást, a Dikin-algoritmussal foglalkozó 24.2. alfejezetet ajánljuk. Végül az utolsó, 24.3.5. alfejezet napjaink egyik fő kutatási irányát ismerteti.

24.1. A lineáris programozás alapvető tételei

Ebben az alfejezetben megfogalmazzuk az általános primál és duál lineáris programozási feladatot, és a gyenge, illetve az erős dualitás tételeket bizonyítjuk. Ezután a Goldman-Tucker-modellt és a speciális önduális lineáris programozási feladatot vezetjük be, majd a centrális út legfontosabb tulajdonságait tárgyaljuk. A Goldman-Tucker- és a Sonnevend-tétel bizonyítása után kitérünk a lineáris programozási feladatokhoz tartozó Newton-rendszer megoldására is. Ezekkel az eredményekkel építjük fel a belsőpontos algoritmusok elméletét. Végezetül a több mint 100 éves Farkas-lemmára a Goldman-Tucker- modell felhasználásával adunk elemi bizonyítást.

Gyenge dualitás tétel

Az ***általános primál*** (P) és ***duál*** (D) ***lineáris programozási feladatot*** az alábbi kanonikus alakban tekintjük:

$$\begin{aligned} (P) \qquad & \min\left\{\mathbf{c}^T\mathbf{x} \;:\; A\mathbf{x} \geq \mathbf{b},\;\; \mathbf{x} \geq \mathbf{0}\right\}, \\ (D) \qquad & \max\left\{\mathbf{b}^T\mathbf{y} \;:\; A^T\mathbf{y} \leq \mathbf{c},\;\; \mathbf{y} \geq \mathbf{0}\right\}, \end{aligned}$$

ahol $A \in \mathbb{R}^{m\times k}$ mátrix, $\mathbf{b},\mathbf{y} \in \mathbb{R}^m$ és $\mathbf{c},\mathbf{x} \in \mathbb{R}^k$. Legyen a ***primál,*** illetve a ***duál megengedett megoldások*** halmaza rendre

$$\mathcal{P} = \{\mathbf{x} \in \mathbb{R}^k_\oplus \ : \ A\mathbf{x} \geq \mathbf{b}\} \quad \text{és} \quad \mathcal{D} = \{\mathbf{y} \in \mathbb{R}^m_\oplus : \ A^T\mathbf{y} \leq \mathbf{c}\} \ ,$$

ahol $\mathbb{R}^k_\oplus$ a nemnegatív k-dimenziós vektorok halmazát, míg a későbbiekben előforduló $\mathbb{R}^k_+$ a pozitív k-dimenziós vektorok halmazát jelöli. Továbbá a ***primál,*** illetve a ***duál optimális megoldások halmazát*** következőképpen jelöljük:

$$\begin{aligned} \mathcal{P}^* &= \{\mathbf{x}^* \in \mathcal{P} \ : \ \mathbf{c}^T\mathbf{x}^* \leq \mathbf{c}^T\mathbf{x}, \ \ \forall \mathbf{x} \in \mathcal{P}\} \\ \text{és} \quad \mathcal{D}^* &= \{\mathbf{y}^* \in \mathcal{D} \ : \ \mathbf{b}^T\mathbf{y}^* \geq \mathbf{b}^T\mathbf{y}, \ \ \forall \mathbf{y} \in \mathcal{D}\} \ . \end{aligned}$$

A gyenge dualitás tétel egyszerűen bizonyítható a kanonikus lineáris programozási feladatra.

24.1. tétel. (gyenge dualitás tétel). *Tegyük fel, hogy $\mathbf{x} \in \mathbb{R}^k$ és $\mathbf{y} \in \mathbb{R}^m$ a primál (P) és duál (D) feladatok megengedett megoldásai. Ekkor*

$$\mathbf{c}^T\mathbf{x} \geq \mathbf{b}^T\mathbf{y} \ ,$$

ahol egyenlőség akkor és csak akkor áll fenn, ha
(i) $x_i(\mathbf{c} - A^T\mathbf{y})_i = 0$ *minden* $i = 1,\ldots,k$ *és*
(ii) $y_j(A\mathbf{x} - \mathbf{b})_j = 0$ *minden* $j = 1,\ldots,m$.

Bizonyítás. Az $\mathbf{x}$ és $\mathbf{y}$ vektorok primál és duál megengedettségét használva kapjuk, hogy

$$(\mathbf{c} - A^T\mathbf{y})^T\mathbf{x} \geq 0 \qquad \text{és} \qquad \mathbf{y}^T(A\mathbf{x} - \mathbf{b}) \geq 0 \ ,$$

ahol egyenlőség akkor és csak akkor áll fenn, ha (i) és (ii) teljesül (lásd 24.1-1 gyakorlat). Ezen két egyenlőtlenséget összeadva a kívánt

$$0 \leq (\mathbf{c} - A^T\mathbf{y})^T\mathbf{x} + \mathbf{y}^T(A\mathbf{x} - \mathbf{b}) = \mathbf{c}^T\mathbf{x} - \mathbf{b}^T\mathbf{y}$$

egyenlőtlenséget kapjuk. Tételünket bebizonyítottuk. ■

Az (i) és (ii) feltételeket ***komplementaritási feltételeknek*** nevezzük. Tehát a gyenge dualitás tétel szerint, komplementaritás és megengedettség a megoldások optimalitását garantálja.

Vektorok ***koordinátánkénti (Hadamard) szorzatát***[2] bevezetve, ahol

[2] Hasonlóan, koordinátánkénti műveletként definiáljuk a vektorok hányadosát és tetszőleges hatványát is.

uv szorzat $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$ esetén azt az n-dimenziós vektort jelöli, melynek koordinátái az $u_i v_i$, $i = 1, \ldots, n$ értékek, a komplementaritási feltétel

$$\mathbf{x}(\mathbf{c} - A^T\mathbf{y}) = \mathbf{0} \quad \text{és} \quad \mathbf{y}(A\mathbf{x} - \mathbf{b}) = \mathbf{0}$$

alakban is írható. A továbbiakban a $\mathbf{c}^T\mathbf{x} - \mathbf{b}^T\mathbf{y}$ különbség értékét $\mathbf{x} \in \mathcal{P}$, $\mathbf{y} \in \mathcal{D}$ esetén ***dualitásrésnek*** nevezzük.

Könnyen bizonyítható az alábbi elégséges optimalitási feltétel (lásd 24.1-2 gyakorlat).

24.2. következmény. (gyenge egyensúlyi tétel). *Legyenek* $\mathbf{x} \in \mathbb{R}^k$ *és* $\mathbf{y} \in \mathbb{R}^m$ *olyan primál és duál megengedett megoldások, amelyekre a dualitásrés nulla, azaz* $\mathbf{c}^T\mathbf{x} = \mathbf{b}^T\mathbf{y}$. *Ekkor* $\mathbf{x}$ *a* (P) *primál feladat egy optimális megoldása és* $\mathbf{y}$ *a* (D) *duál feladat egy optimális megoldása.*

Goldman-Tucker-modell

Célunk a lineáris programozási feladat olyan megoldásainak előállítása, melyekre a dualitásrés nulla, így azt az egyenlőtlenségrendszert kell megoldanunk, amely a primál és a duál feltételekből áll, valamint előírjuk, hogy a duál célfüggvény értéke legalább akkora legyen, mint a primál célfüggvény értéke ($\mathbf{b}^T\mathbf{y} \geq \mathbf{c}^T\mathbf{x}$). Ekkor ugyanis a gyenge dualitás tétel miatt ezen rendszer minden megengedett megoldása primál és duál megengedett megoldást ad, amelyre a célfüggvényértékek szükségképpen egyenlők, így a 24.2. következmény szerint optimális megoldások.

A szükséges eltérésváltozók $(\mathbf{t}, \mathbf{s}, \zeta)$ bevezetésével az alábbi egyenlőtlenségrendszerre jutunk:

$$\begin{aligned} A\mathbf{x} - \mathbf{t} &= \mathbf{b}, \quad \mathbf{x} \geq \mathbf{0}, \quad \mathbf{t} \geq \mathbf{0}\,, \\ A^T\mathbf{y} + \mathbf{s} &= \mathbf{c}, \quad \mathbf{y} \geq \mathbf{0}, \quad \mathbf{s} \geq \mathbf{0}\,, \\ \mathbf{b}^T\mathbf{y} - \mathbf{c}^T\mathbf{x} - \zeta &= 0, \quad \zeta \geq 0\,. \end{aligned}$$

Homogenizálva az egyenleteket a ***Goldman-Tucker-feladatot*** kapjuk:

$$\left.\begin{array}{rrrrrl} A\mathbf{x} & -\xi\mathbf{b} & -\mathbf{t} & & = \mathbf{0}, & \mathbf{x} \geq \mathbf{0}, \quad \mathbf{t} \geq \mathbf{0}\,, \\ -A^T\mathbf{y} & +\xi\mathbf{c} & -\mathbf{s} & & = \mathbf{0}, & \mathbf{y} \geq \mathbf{0}, \quad \mathbf{s} \geq \mathbf{0}\,, \\ \mathbf{b}^T\mathbf{y} & -\mathbf{c}^T\mathbf{x} & & -\zeta & = 0, & \xi \geq 0, \quad \zeta \geq 0\,. \end{array}\right\} \quad (GT)$$

A triviális, azonosan nulla megoldás kielégíti ezt a homogén rendszert, de ez a céljaink szempontjából érdektelen. A Goldman-Tucker-rendszer bizonyos nemtriviális, speciális megoldását szeretnénk előállítani, és ennek segítségével nyerjük a (P) és (D) feladatok optimális megoldásait, illetve következtetünk a (P) és (D) feladatok megoldhatóságára az alábbi tétel alapján.

24.3. tétel. *Adott egy primál* (P) *és duál* (D) *lineáris programozási feladatpár. Az alábbi állítások igazak:*

1. *A (P) és (D) feladatok tetszőleges $(\mathbf{x},\mathbf{y})$ optimális megoldás párja, melyre a dualitásrés nulla, a megfelelő Goldman-Tucker-rendszer egy megoldását adja $\xi=1$ és $\zeta=0$ választással.*

2. *Ha $(\mathbf{y},\mathbf{x},\xi,\mathbf{t},\mathbf{s},\zeta)$ a Goldman-Tucker-rendszer egy megoldása, akkor vagy $\xi=0$ vagy $\zeta=0$, azaz $\xi\zeta>0$ nem teljesülhet.*

3. *Ha a Goldman-Tucker-rendszer egy $(\mathbf{y},\mathbf{x},\xi,\mathbf{t},\mathbf{s},\zeta)$ megoldására $\xi>0$ és $\zeta=0$, akkor a $(\frac{\mathbf{x}}{\xi},\frac{\mathbf{y}}{\xi})$ vektor a primál (P) és a duál (D) feladatok egy optimális megoldás párja.*

4. *Ha a Goldman-Tucker-rendszernek van olyan $(\bar{\mathbf{y}},\bar{\mathbf{x}},\bar{\xi},\bar{\mathbf{t}},\bar{\mathbf{s}},\bar{\zeta})$ megoldása, amelyre $\bar{\xi}=0$ és $\bar{\zeta}>0$, akkor megállapíthatjuk, hogy vagy a (P), vagy a (D) feladat, vagy mindkettő nem megengedett.*

Bizonyítás. Az első és a harmadik állítás behelyettesítéssel könnyen ellenőrizhető (lásd 24.1-3 gyakorlat).

A második állítást indirekt módon bizonyítjuk. Ha $\xi\zeta$ pozitív lenne, akkor

$$0<\xi\zeta=\xi\mathbf{b}^T\mathbf{y}-\xi\mathbf{c}^T\mathbf{x}=\mathbf{x}^TA\mathbf{y}-\mathbf{t}^T\mathbf{y}-\mathbf{x}^TA^T\mathbf{y}-\mathbf{s}^T\mathbf{x}=-\mathbf{t}^T\mathbf{y}-\mathbf{s}^T\mathbf{x}\le 0$$

egyenlőtlenséget kapnánk, ami nyilvánvaló ellentmondás.

Az utolsó állítás igazolásánál a $\bar{\xi}=0$ feltételből következik, hogy $A\bar{\mathbf{x}}\ge\mathbf{0}$ és $A^T\bar{\mathbf{y}}\le\mathbf{0}$. Továbbá, ha $\bar{\zeta}>0$, akkor vagy $\mathbf{b}^T\bar{\mathbf{y}}>0$, vagy $\mathbf{c}^T\bar{\mathbf{x}}<0$, vagy mindkettő fennáll. Ha $\mathbf{b}^T\bar{\mathbf{y}}>0$, akkor feltételezve, hogy a (P) feladatnak van egy $\mathbf{x}\ge\mathbf{0}$ megoldása a

$$0<\mathbf{b}^T\bar{\mathbf{y}}\le\mathbf{x}^TA^T\bar{\mathbf{y}}\le 0$$

ellentmondáshoz jutunk. Tehát ha $\mathbf{b}^T\bar{\mathbf{y}}>0$, akkor (P) nem megengedett. Hasonlóan, ha $\mathbf{c}^T\bar{\mathbf{x}}<0$, akkor a duál feladat nem megengedettségét kapjuk (lásd 24.1-4 gyakorlat). ∎

Vegyük észre, hogy a Goldman-Tucker-rendszer az alábbi kompakt alakban írható:

$$M\mathbf{u}\ge\mathbf{0},\qquad \mathbf{u}\ge\mathbf{0},\qquad \mathbf{z}=M\,\mathbf{u}, \tag{24.1}$$

ahol

$$\mathbf{u}=\begin{pmatrix}\mathbf{y}\\ \mathbf{x}\\ \xi\end{pmatrix},\qquad \mathbf{z}=\begin{pmatrix}\mathbf{t}\\ \mathbf{s}\\ \zeta\end{pmatrix}\quad\text{és}\quad M=\begin{pmatrix}0 & A & -\mathbf{b}\\ -A^T & 0 & \mathbf{c}\\ \mathbf{b}^T & -\mathbf{c}^T & 0\end{pmatrix}$$

egy ***ferdén szimmetrikus mátrix*** ($M^T=-M$, lásd 24.1-5 gyakorlat), azaz tetszőleges (P) és (D) feladatpárhoz tartozó Goldman-Tucker-rendszer

felírható, mint egy speciális struktúrájú, kanonikus lineáris programozási feladat, azaz

$$(\overline{SP}) \qquad \min\left\{\mathbf{0}^T\mathbf{u} \;:\; M\mathbf{u}\geq\mathbf{0},\;\; \mathbf{u}\geq\mathbf{0}\right\}\;.$$

24.1.1. A ferdén szimmetrikus önduális feladat alaptulajdonságai

Az $(\overline{SP})$ feladatnál tekintsünk egy kicsit általánosabb feladatot. Ez az $(\overline{SP})$ feladattól abban különbözik, hogy nem homogén, azaz a jobb oldala nem feltétlenül a nulla vektor, és célfüggvény együttható vektora a jobb oldali vektor negatívja. A következő (speciális) lineáris programozási feladattal ***(önduális feladat)*** foglalkozunk a továbbiakban

$$\left.\begin{array}{rl} \min \mathbf{q}^T\mathbf{u} & \\ M\mathbf{u} & \geq -\mathbf{q} \\ \mathbf{u} & \geq \;\;\mathbf{0} \end{array}\right\} \quad (SP)\;,$$

ahol $M\in\mathbb{R}^{n\times n}$ ferdén szimmetrikus mátrix és $\mathbf{q}\in\mathbb{R}^n_\oplus$ vektor. Jelölje

$$\mathcal{F}=\{\mathbf{u}\in\mathbb{R}^n_\oplus \;:\; M\mathbf{u}\geq-\mathbf{q}\,\}$$

az (SP) feladat ***megengedett megoldásainak halmazát***. Felhasználva, hogy az M mátrix ferdén szimmetrikus és hogy a jobb oldalon álló vektor $(-\mathbf{q})$ a célfüggvényvektor negatívja, az Olvasó könnyen ellenőrizheti, hogy (SP) duálja ekvivalens az (SP) feladattal, azaz az (SP) ***önduális*** feladat (lásd 24.1-6 feladat). Az önduális tulajdonság miatt a következő eredmény triviális (lásd 24.1-7 feladat).

24.4. lemma. *Az (SP) feladat optimum értéke nulla, továbbá az azonosan nulla vektor,* $\mathbf{u}=\mathbf{0}$, *megengedett és egyben optimális megoldása az (SP) feladatnak.*

Ha $\mathbf{u}$ megoldása az (SP) feladatnak, és $z(\mathbf{u})=M\mathbf{u}+\mathbf{q}$ az $\mathbf{u}$ vektorhoz tartozó ***eltérésvektor***, akkor M ferdén szimmetrikussága miatt $\mathbf{u}^TM\mathbf{u}=0$, így

$$\mathbf{q}^T\mathbf{u}=\mathbf{u}^T(z(\mathbf{u})-M\mathbf{u})=\mathbf{u}^Tz(\mathbf{u})=\mathbf{e}^T(\mathbf{u}\,z(\mathbf{u}))\;,$$

ahol $\mathbf{e}\in\mathbb{R}^n$ a csupa egyes vektor. A fenti egyenlőség miatt tetszőleges optimális megoldásra $\mathbf{e}^T(\mathbf{u}\,z(\mathbf{u}))=0$, azaz $\mathbf{u}\,z(\mathbf{u})=0$, amiből az is következik, hogy az $\mathbf{u}$ és $z(\mathbf{u})$ vektorok komplementárisak. Jelölje

$$\begin{aligned}\mathcal{F}^* &= \{\mathbf{u}^*\in\mathcal{F} \;:\; \mathbf{q}^T\mathbf{u}^*\leq\mathbf{q}^T\mathbf{u},\;\forall\mathbf{u}\in\mathcal{F}\}\\ &= \{\mathbf{u}^*\in\mathcal{F} \;:\; \mathbf{q}^T\mathbf{u}^*=0\}=\{\mathbf{u}^*\in\mathcal{F} \;:\; \mathbf{u}^*\,z(\mathbf{u}^*)=\mathbf{0}\}\end{aligned}$$

az ***(SP) feladat optimális megoldásainak a halmazát.*** Adott $\mathbf{u} \in \mathcal{F}$ esetén az $\mathbf{u}^T z(\mathbf{u}) = \mathbf{q}^T \mathbf{u}$ értéket *dualitásrésnek* nevezzük. [3] Az $\mathcal{F}^*$ halmaz definíciója alapján világos, hogy optimális megoldás esetén a dualitásrés nulla.

Az optimális megoldások egy, a továbbiakban gyakran használt tulajdonságát fogalmazzuk meg az alábbi lemmában.

24.5. lemma. *Legyen* $\mathbf{u}$ *és* $\hat{\mathbf{u}}$ *az (SP) feladat megengedett megoldása. Az* $\mathbf{u}$ *és* $\hat{\mathbf{u}}$ *vektorok akkor és csak akkor optimálisak, ha*

$$\mathbf{u}\, z(\hat{\mathbf{u}}) = \hat{\mathbf{u}}\, z(\mathbf{u}) = \mathbf{u}\, z(\mathbf{u}) = \hat{\mathbf{u}}\, z(\hat{\mathbf{u}}) = \mathbf{0}\ .$$

Bizonyítás. Mivel M ferdén szimmetrikus, így $(\mathbf{u}-\hat{\mathbf{u}})^T M(\mathbf{u}-\hat{\mathbf{u}}) = 0$, amiből következik, hogy $(\mathbf{u}-\hat{\mathbf{u}})^T(z(\mathbf{u})-z(\hat{\mathbf{u}})) = 0$. Ekkor $\mathbf{u}^T z(\hat{\mathbf{u}})+\hat{\mathbf{u}}^T z(\mathbf{u}) = \mathbf{u}^T z(\mathbf{u}) + \hat{\mathbf{u}}^T z(\hat{\mathbf{u}})$ és ez akkor és csak akkor nulla, ha $\mathbf{u}$ és $\hat{\mathbf{u}}$ is optimális, de ekkor

$$\mathbf{u}^T z(\hat{\mathbf{u}}) + \hat{\mathbf{u}}^T z(\mathbf{u}) = 0\ . \tag{24.2}$$

Figyelembe véve az $\mathbf{u}, \hat{\mathbf{u}} \in \mathcal{F}$ feltételt $\mathbf{u}^T z(\hat{\mathbf{u}}) \geq 0$ és $\hat{\mathbf{u}}^T z(\mathbf{u}) \geq 0$ teljesül, amelyből, az (24.2) összefüggés alapján

$$\mathbf{u}^T z(\hat{\mathbf{u}}) = \mathbf{e}^T(\mathbf{u}\, z(\hat{\mathbf{u}})) = 0 \qquad \text{és} \qquad \hat{\mathbf{u}}^T z(\mathbf{u}) = \mathbf{e}^T(\hat{\mathbf{u}}\, z(\mathbf{u})) = 0$$

következik. Tehát $\mathbf{u}\, z(\hat{\mathbf{u}}) = \hat{\mathbf{u}}\, z(\mathbf{u}) = \mathbf{0}$ adódik. ∎

Megállapíthatjuk, hogy az optimális megoldások általános értelemben is komplementárisak, azaz nem csak saját eltérésvektorukkal, hanem bármelyik más optimális megoldás eltérésvektorával is komplementáris párt alkotnak.

Az (SP) feladat ***optimalitási kritériumát*** a következő alakban is megadhatjuk

$$\left.\begin{aligned} -M\mathbf{u} + \mathbf{z} &= \mathbf{q} \\ \mathbf{u} \geq \mathbf{0}, \mathbf{z} &\geq \mathbf{0} \\ \mathbf{u}\,\mathbf{z} &= \mathbf{0} \end{aligned}\right\} (SP_{OPT})\ .$$

Az összes eddigi eredmény, egy triviális optimális megoldás létezését is beleértve, szinte magától értetődő volt az (SP) feladatra. Ebből talán arra következtethetnénk, hogy az (SP) feladat a lineáris programozási feladatok érdektelen, nagyon speciális esete. Tekintettel arra, hogy az (SP) feladat homogén változatát a kanonikus primál-és duál feladatokból vezettük le, fel sem merül annak a lehetősége, hogy egy érdektelen feladattal foglalkoznánk.

Az első felvetődő kérdés az, hogy létezik-e a triviálison kívül másmilyen optimális megoldása a feladatnak, és hogyan tudjuk azt előállítani. Ennek a kérdésnek a megválaszolása elvezet az optimális megoldások bizonyos

[3] Tesszük ezt annak ellenére, hogy az önduális feladat esetén az $\mathbf{u}^T z(\mathbf{u}) = \mathbf{q}^T \mathbf{u}$ nyilván csak a fele a klasszikus értelemben vett dualitásrésnek (lásd **??**. oldal).

komplementaritási tulajdonságainak kérdéséhez, azaz a lineáris programozás elméletének egyik alapvető jelentőségű tételéhez, a Goldman–Tucker-tételhez.

Szükségünk lesz a következő definícióra.

24.6. definíció. *Legyen* $\mathbf{u} \in \mathcal{F}^*$. *Az* $\mathbf{u}$ *és* $z(\mathbf{u})$ *vektorokat* ***szigorúan komplementárisnak*** *nevezzük, ha a* $\mathbf{u} + z(\mathbf{u}) > \mathbf{0}$ *feltétel teljesül.*

Az $\mathbf{u}$ és $z(\mathbf{u})$ vektorok szigorú komplementaritásának egyszerű következménye az, hogy minden $i = 1, 2, \ldots, n$ index esetén az $u_i = 0$ és $z_i(u) = 0$ feltételek közül pontosan az egyik teljesül. A 24.3. tétel 3. és 4. állításai bizonyítják, hogy amennyiben az(SP) feladat a (GT) rendszer alapján egy lineáris programozási feladatpárból származik, bármely szigorúan komplementáris megoldása vagy egy optimális megoldás párt ad az eredeti (P) és (D) lineáris programozási feladatokra, vagy azok valamelyikének megoldhatatlanságát bizonyítja. Az (SP) feladat egy szigorúan komplementáris megoldásának az előállítása a lineáris programozás belsőpontos módszereinek a bevezetését igényeli.

Vezessük be a ***belsőpontos megoldások halmazát***

$$\mathcal{F}^0 = \{\mathbf{u} \in \mathcal{F} \ : \ (\mathbf{u},\, z(\mathbf{u})) > \mathbf{0}\} \ .$$

Ekkor az ún. ***belső pont feltételt*** az alábbi módon fogalmazhatjuk meg:

$$\mathcal{F}^0 \neq \emptyset \ ,$$

amelyet másként úgy is kimondhatunk, hogy létezik olyan $\bar{\mathbf{u}} \in \mathcal{F}$ vektor, amelyre

$$(\bar{\mathbf{u}},\ z(\bar{\mathbf{u}})) > \mathbf{0}$$

teljesül. (A 24.1-11 gyakorlat ad példát olyan lineáris programozási feladatra, amelyikből elkészített Goldman-Tucker-feladatnak van belső pontja.)

Newton-lépés és tulajdonságai

Legyen adott $(\mathbf{u},\, \mathbf{z}) > \mathbf{0}$, melyre $\mathbf{z} = M\,\mathbf{u} + \mathbf{q}$. Adott $\mathbf{w} \in \mathbb{R}^n$, $\mathbf{w} > \mathbf{0}$ vektor esetén szeretnénk[4] olyan $(\triangle\mathbf{u},\, \triangle\mathbf{z})$ elmozdulás vektort („lépést") meghatározni, melyre

$$\begin{aligned} M\,(\mathbf{u} + \triangle\mathbf{u}) + \mathbf{q} &= \mathbf{z} + \triangle\mathbf{z} \ , \\ (\mathbf{u} + \triangle\mathbf{u})\,(\mathbf{z} + \triangle\mathbf{z}) &= \mathbf{w} \end{aligned}$$

[4] Célunk a

$$\begin{aligned} -M\bar{\mathbf{u}} + \bar{\mathbf{z}} &= \mathbf{q} \\ \bar{\mathbf{u}}\,\bar{\mathbf{z}} &= \mathbf{0} \end{aligned}$$

rendszer megoldása, ahol a megoldást $\bar{\mathbf{u}} = \mathbf{u} + \Delta\mathbf{u}$ és $\bar{\mathbf{u}} = \mathbf{u} + \Delta\mathbf{u}$ alakban keressük.

egyenletrendszer teljesül. Ez nemlineáris egyenletrendszer, így direkt megoldása nem lehetséges. A $(\Delta\mathbf{u}, \Delta\mathbf{z})$ ismeretlenekre vonatkozóan átrendezve a

$$\begin{aligned} M\,\Delta\mathbf{u} - \Delta\mathbf{z} &= \mathbf{0}\,, \\ \mathbf{u}\,\Delta\mathbf{z} + \mathbf{z}\,\Delta\mathbf{u} + \Delta\mathbf{u}\,\Delta\mathbf{z} &= \mathbf{w} - \mathbf{u}\,\mathbf{z} \end{aligned}$$

egyenletrendszert kapjuk, amely még mindig nemlineáris. A másodrendű $\Delta\mathbf{u}\,\Delta\mathbf{z}$ tag elhagyásával az

$$\begin{aligned} M\,\Delta\mathbf{u} - \quad \Delta\mathbf{z} &= \mathbf{0}\,, \\ Z\,\Delta\mathbf{u} + U\,\Delta\mathbf{z} &= \mathbf{w} - \mathbf{u}\,\mathbf{z} \end{aligned} \tag{24.3}$$

lineáris egyenletrendszer adódik, ahol $U = \mathrm{diag}(\mathbf{u})$ és $Z = \mathrm{diag}(\mathbf{z})$ pozitív diagonális mátrixok. A (24.3) egyenletrendszert ***Newton-rendszernek***[5] nevezzük, amelynek az együtthatómátrixa

$$\bar{M} = \begin{pmatrix} M & -I \\ Z & U \end{pmatrix} \in \mathbb{R}^{2n\times 2n} \tag{24.4}$$

alakú. A $\Delta\mathbf{z} = M\,\Delta\mathbf{u}$ behelyettesítése után a Newton rendszer

$$(Z + U\,M)\,\Delta\mathbf{u} = \mathbf{w} - \mathbf{u}\,\mathbf{z} \tag{24.5}$$

alakra egyszerűsödik, amelyet ***normál egyenletrendszernek*** nevezünk.

24.7. állítás. *Legyenek U, Z, I, $M \in \mathbb{R}^{n\times n}$ mátrixok. U és Z pozitív diagonális, I egység, míg M ferdén szimmetrikus mátrix. Ekkor a (24.4) összefüggéssel definiált $\bar{M}$ és a $Z + UM$ mátrix reguláris, tehát a (24.5) lineáris egyenletrendszernek létezik egyértelmű megoldása.*

A bizonyítás lineáris algebrai gondolatmenetet használ, és az Olvasóra bízzuk (lásd 24.1-12, illetve 24.1-13 gyakorlat).

A $\Delta\mathbf{u}$ és $\Delta\mathbf{z}$ elmozdulás vektorokat ***Newton-irányoknak*** nevezzük, és az alábbi formában[6] adhatók meg:

$$\begin{aligned} \Delta\mathbf{u} &= (Z + U\,M)^{-1}\,(\mathbf{w} - \mathbf{u}\,\mathbf{z}) = (U^{-1}Z + M)^{-1}(U^{-1}\mathbf{w} - \mathbf{z}) \quad \text{és} \\ \Delta\mathbf{z} &= M(U^{-1}Z + M)^{-1}(U^{-1}\mathbf{w} - \mathbf{z}) = M\Delta\mathbf{u}\,. \end{aligned} \tag{24.6}$$

A lépéshossz követelménye, hogy az új pontunkra is teljesüljön az előjel feltétel. Miután a Newton-irányban egy α lépéshosszú lépést teszünk, az új

[5] Ez a rendszer a numerikus analízisből és a többdimenziós nemlineáris egyenletrendszerek megoldásához is használt jól ismert Newton-módszer megfelelője.

[6] A Newton-irányok meghatározásánál a numerikus szempontokból hatékonyabb, ha a (24.5) lineáris egyenletrendszer egyértelmű megoldásaként határozzuk meg a $\Delta\mathbf{u}$ vektort.

$(\mathbf{u}(\alpha), \mathbf{z}(\alpha)) = (\mathbf{u}+\alpha\Delta\mathbf{u}, \mathbf{z}+\alpha\Delta\mathbf{z})$ megoldásra a következő kifejezést kapjuk:

$$\begin{aligned}\mathbf{u}(\alpha)\,\mathbf{z}(\alpha) &= (\mathbf{u}+\alpha\Delta\mathbf{u})(\mathbf{z}+\alpha\,\Delta\mathbf{z}) = \mathbf{u}\,\mathbf{z} + \alpha\,(\mathbf{u}\,\Delta\mathbf{z} + \mathbf{z}\,\Delta\mathbf{u}) + \alpha^2\,\Delta\mathbf{u}\,\Delta\mathbf{z} \\ &= \mathbf{u}\,\mathbf{z} + \alpha\,(\mathbf{w} - \mathbf{u}\,\mathbf{z}) + \alpha^2\,\Delta\mathbf{u}\,\Delta\mathbf{z}\ .\end{aligned}$$

Ez az összefüggés világossá teszi, hogy az $\mathbf{u}\,\mathbf{z}$ vektor lokális megváltozását a $\mathbf{w} - \mathbf{u}\,\mathbf{z}$ vektor határozza meg. Szerencsére ezt a vektort explicit ismerjük, amikor a Newton-lépést alkalmazzuk. Így ha α elegendően kicsi, akkor pontosan tudjuk, hogy $\mathbf{u}\,\mathbf{z}$ mely koordinátái csökkennek lokálisan (pontosan azok, amelyekre a $\mathbf{w} - \mathbf{u}\,\mathbf{z}$ vektor megfelelő koordinátái negatívak), és $\mathbf{u}\,\mathbf{z}$ mely koordinátái növekednek lokálisan (pontosan azok, amelyekre a $\mathbf{w} - \mathbf{u}\,\mathbf{z}$ vektor megfelelő koordinátái pozitívak).

A 24.7. állítást illusztrálja a 24.1-11 gyakorlat. Ebben a feladatban a Newton-irányok kiszámítására, az α lépéshossz előállítására és a megengedett pozitív $\mathbf{u}^+$, illetve $\mathbf{z}^+$ meghatározására láthatunk példát.

Ezeket az észrevételeket fejezi ki pontosabban a következő állítás is.

24.8. állítás. *Legyen az M ferdén szimmetrikus mátrix, és $\mathbf{u} \in \mathcal{F}^0$, azaz $(\mathbf{u},\, z(\mathbf{u})) > \mathbf{0}$. Legyen továbbá, $\hat{\mathbf{w}} \in \mathbb{R}^n$, $\hat{\mathbf{w}} > \mathbf{0}$ és $\mathbf{w} = \mathbf{u}\,z(\mathbf{u})$. Definiáljuk a*

$$\mathcal{T}(\hat{\mathbf{w}},\, \mathbf{w}) = \{\, \mathbf{u} \in \mathbb{R}^n \ :\ \hat{w}_i \le u_i \le w_i \ \ \textit{vagy}\ \ w_i \le u_i \le \hat{w}_i,\ \forall\, i \,\}$$

***tégla** halmazt. Ha $\operatorname{int}\mathcal{T}(\hat{\mathbf{w}},\, \mathbf{w}) \neq \emptyset$, akkor a $\hat{\mathbf{w}}$ vektorra kiszámított $(\Delta\mathbf{u},\, \Delta\mathbf{z})$ Newton-irányra létezik olyan $\alpha_* \in (0, 1)$ lépéshossz, hogy az*

$$\mathbf{u}^+ = \mathbf{u} + \alpha_*\Delta\mathbf{u}\ , \quad \mathbf{z}^+ = \mathbf{z} + \alpha_*\,\Delta\mathbf{z}\ , \quad \mathbf{w}^+ = \mathbf{u}^+\mathbf{z}^+\ ,$$

vektorokra a $\mathbf{w}^+ \in \operatorname{int}\mathcal{T}(\hat{\mathbf{w}},\, \mathbf{w})$, valamint $\mathbf{u}^+ \in \mathcal{F}^0$ teljesül.

Az előző állítás bizonyításának a legfontosabb részeit két feladatra bontjuk (lásd 24.1-14 gyakorlat), amelyek megoldását az Olvasóra bízzuk.

Az (*SP*) feladat szinthalmazai és tulajdonságaik

Először vezessük be a

$$\mathcal{L}_K = \{\mathbf{u} \in \mathcal{F}\ :\ \mathbf{u}^T z(\mathbf{u}) \le K\} = \{\mathbf{u} \in \mathcal{F}\ :\ \mathbf{q}^T\mathbf{u} \le K\}$$

szinthalmazt és bármely $\mathbf{w} \in \mathbb{R}^n_+$ vektor esetén az

$$\mathcal{L}_{\mathbf{w}} = \{(\mathbf{u},\, \mathbf{z}) \in \mathbb{R}^{2n}_{\oplus}\ :\ \mathbf{z} = M\mathbf{u} + \mathbf{q} \text{ és } \ \mathbf{u}\,\mathbf{z} \le \mathbf{w}\}$$

általánosított szinthalmazt.

24.9. lemma. *Legyen adott az (SP) feladat és tegyük fel, hogy $\mathcal{F}^0 \neq \emptyset$. Ekkor bármely $K \in \mathbb{R}$, $K > 0$ szám esetén az $\mathcal{L}_K$ szinthalmaz korlátos és zárt, azaz kompakt.*

Az előző lemma bizonyítását (lásd 24.1-16 gyakorlat) az Olvasóra bízzuk.

A 24.9. lemmához hasonló eredmények igazak az $\mathcal{L}_{\mathbf{w}}$ szinthalmazokra is. A szinthalmaz definíciójában szereplő nemlineáris kifejezés miatt kicsit bonyolultabb a zártság bizonyítása (lásd 24.1-17 gyakorlat).

24.10. lemma. *Legyen adott az (SP) feladat és tegyük fel, hogy $\mathcal{F}^0 \neq \emptyset$. Ekkor bármely $\mathbf{w} \in \mathbb{R}^n_+$, vektor esetén az $\mathcal{L}_{\mathbf{w}}$ szinthalmaz korlátos és zárt, azaz kompakt.*

A szinthalmazokkal kapcsolatos lemmákra a következő állítás bizonyításakor lesz szükségünk. Ebben a lemmában azt vizsgáljuk, hogy milyen halmazt alkotnak azok a $\mathbf{w} \in \mathbb{R}^n_\oplus$ vektorok, amelyek előállnak az (SP) feladat valamely megengedett megoldásának és a hozzátartozó eltérésváltozónak a szorzataként.

24.11. lemma. *Legyen adott az (SP) feladat és tegyük fel, hogy $\mathcal{F}^0 \neq \emptyset$. Definiáljuk a következő halmazt:*

$$\mathcal{G} = \{\mathbf{w} \in \mathbb{R}^n_\oplus \,:\, \exists \mathbf{u} \in \mathcal{F}, \quad \textit{amelyre} \quad \mathbf{u}\, z(\mathbf{u}) = \mathbf{w}\}\ .$$

Ekkor a $\mathcal{G}$ halmaz nem üres és zárt.

A lemma bizonyítását (lásd 24.1-18 gyakorlat) az Olvasóra bízzuk. Az előző eredmény segítségével belátjuk, hogy belső pont létezése mellett minden $\mathbf{w} > 0$ esetén az $\mathcal{L}_{\mathbf{w}}$ szinthalmaz nem üres.

24.12. tétel. *Legyen adott az (SP) feladat, és tegyük fel, hogy $\mathcal{F}^0 \neq \emptyset$. Ekkor bármely $\mathbf{w} \in \mathbb{R}^n_+$ vektor esetén az $\mathcal{L}_{\mathbf{w}}$ szinthalmaz nem üres.*

Bizonyítás. A 24.10. lemmában már igazoltuk, hogy az $\mathcal{L}_{\mathbf{w}}$ halmaz kompakt. Indirekt módon tegyük fel, hogy létezik egy $\hat{\mathbf{w}} \in \mathbb{R}^n_+$ vektor, amelyre $\mathcal{L}_{\hat{\mathbf{w}}} = \emptyset$.

Az $\mathcal{F}^0 \neq \emptyset$ feltevés miatt létezik egy $\bar{\mathbf{u}} \in \mathcal{F}^0$ vektor és egy $\bar{\mathbf{w}} = \bar{\mathbf{u}}\,\bar{\mathbf{z}} > \mathbf{0}$, ahol $\bar{\mathbf{z}} = M\,\bar{\mathbf{u}} + \mathbf{q} > \mathbf{0}$ vektor, amelyek esetén az $\mathcal{L}_{\bar{\mathbf{w}}}$ szinthalmaz nem üres és kompakt.

Legyen $\mathbf{w}' > \bar{\mathbf{w}}$ és $\mathbf{w}' > \hat{\mathbf{w}}$. Ekkor az $\mathcal{A} = \{\mathbf{w} \in \mathbb{R}^n_\oplus \,:\, \mathbf{e}^T\mathbf{w} \leq \mathbf{e}^T\mathbf{w}'\}$ nem üres és kompakt halmaz. Továbbá az $\mathcal{A} \cap \mathcal{G}$ halmaz sem üres és kompakt.

Definiáljuk az $f : \mathcal{A} \cap \mathcal{G} \to \mathbb{R}^n_\oplus$ függvényt a következő kifejezéssel

$$f_i(\mathbf{w}) = \begin{cases} 0, & \text{ha } \ w_i \leq \hat{w}_i\ , \\ w_i - \hat{w}_i & \text{különben}\ . \end{cases}$$

A 24.1-19 gyakorlat szerint $\|f\|_\infty$ felveszi a minimumát, azaz létezik

$$\gamma = \|f(\tilde{\mathbf{w}})\|_\infty = \min_{\mathbf{w}\in\mathcal{A}\cap\mathcal{G}} \|f(\mathbf{w})\|_\infty \le \|f(\bar{\mathbf{w}})\|_\infty \,.$$

Mivel az $\mathcal{L}_{\hat{\mathbf{w}}} = \emptyset$ (indirekt feltevés), ezért $\gamma = \|f(\tilde{\mathbf{w}})\|_\infty > 0$.

Ha $\operatorname{int}\mathcal{T}(\tilde{\mathbf{w}}, \hat{\mathbf{w}}) \neq \emptyset$, akkor létezik $\alpha \in (0,1)$, amelyre $\mathbf{u}^+ = \bar{\mathbf{u}} + \alpha\,\Delta\mathbf{u}$ és $\mathbf{z}^+ = \bar{\mathbf{z}} + \alpha\,\Delta\mathbf{z}$ olyan vektorok, amelyekre $\mathbf{w}^+ = \mathbf{u}^+\,\mathbf{z}^+ \in \operatorname{int}\mathcal{T}(\tilde{\mathbf{w}}, \hat{\mathbf{w}})$, azaz

$$\|f(\mathbf{w}^+)\|_\infty < \|f(\tilde{\mathbf{w}})\|_\infty = \gamma \,,$$

ami ellentmond Weierstrass tételének.

Ha $\operatorname{int}\mathcal{T}(\tilde{\mathbf{w}}, \hat{\mathbf{w}}) = \emptyset$, akkor legyen $\mathbf{w}^* \in \mathcal{B}_{\gamma/3}(\hat{\mathbf{w}}) \cap \operatorname{int}\mathcal{T}(\mathbf{0}, \hat{\mathbf{w}})$ vektor[7], amely esetén $\operatorname{int}\mathcal{T}(\tilde{\mathbf{w}}, \mathbf{w}^*) \neq \emptyset$ és megismételhetjük az előző gondolatmenetet. ■

24.1.2. Centrális út

Az (SP) feladat optimalitási kritériumának relaxáltja a

$$\left.\begin{aligned} -M\mathbf{u} + \mathbf{z} &= \mathbf{q} \\ \mathbf{u} \ge \mathbf{0},\ \mathbf{z} &\ge \mathbf{0} \\ \mathbf{u}\,\mathbf{z} &= \mu\mathbf{e} \end{aligned}\right\} \quad (CP)\,,$$

az úgynevezett ***centrális út feladat.*** Belátjuk, hogy adott $\mu > 0$ paraméter esetén a (CP) feladatnak egyértelmű megoldása van. Ezt a megoldást μ-***centrumnak*** nevezzük, amelyet az $(\mathbf{u}(\mu), \mathbf{z}(\mu))$ vektorral jelölünk.

24.13. definíció. *A*

$$\mathcal{C} = \{(\mathbf{u}(\mu), z(\mathbf{u}(\mu))) : \mathbf{u}(\mu) \in \mathcal{F}^0,\ \mathbf{u}(\mu)\mathbf{z}(\mu)$$

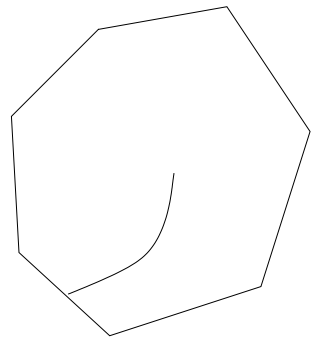
$$= \mu\mathbf{e}, \quad \textit{valamely } \mu \in \mathbb{R}_+ \textit{ paraméterre}\}$$

halmazt az (SP) feladat ***centrális útjának*** *nevezzük.*

A centrális út tehát a μ-centrumok által alkotott görbe (lásd 24.1. ábra). Bizonyítható, hogy belső pont létezése esetén a centrális út létezik és egyértelmű, sőt a centrális út egy környezete is egyértelmű. Ennél többet bizonyítunk be a következő tételben, mégpedig a fenti állítások ekvivalenciáját.

24.14. tétel. *Legyen adott az (SP) feladat és tegyük fel, hogy $\mathcal{F} \neq \emptyset$. Ekkor a következő állítások ekvivalensek:*

[7] $\mathcal{B}_\gamma(\mathbf{w}) = \{\mathbf{u} \in \mathbb{R}^n \ :\ \|\mathbf{u} - \mathbf{w}\| < \gamma\}$, azaz a $\mathbf{w}$ középpontú, γ sugarú gömböt jelöli.



24.1. ábra. Egy játékfa.

1. $\mathcal{F}^0 \neq \emptyset$;
2. $\forall \mathbf{w} \in \mathbb{R}^n_+$ *esetén* $\exists!\ (\mathbf{u},\mathbf{z}) > 0\ :\ M\mathbf{u}+\mathbf{q}=\mathbf{z}$ *és* $\mathbf{u}\,\mathbf{z}=\mathbf{w}$;
3. $\forall \mu > 0$ *esetén* $\exists!\ (\mathbf{u},\mathbf{z}) > 0\ :\ M\mathbf{u}+\mathbf{q}=\mathbf{z}$ *és* $\mathbf{u}\,\mathbf{z}=\mu\mathbf{e}$.

Bizonyítás. A 3. állítás a 2. speciális esete, továbbá a 3. állításból következik az 1., ugyanis $\mathbf{u}(\mu)\mathbf{z}(\mu)=\mu\mathbf{e}$ esetén $\mathbf{u}(\mu)\in\mathcal{F}^0$.
Az 1. $\Rightarrow$ 2. implikációt indirekt bizonyítjuk. Tegyük fel, hogy $\exists\,\hat{\mathbf{w}}\in\mathbb{R}^n_+$ úgy, hogy $\nexists\,\hat{\mathbf{u}}\in\mathcal{F}^0$ és $\hat{\mathbf{z}}=M\hat{\mathbf{u}}+\mathbf{q}$, amire teljesülne $\hat{\mathbf{w}}=\hat{\mathbf{u}}\,\hat{\mathbf{z}}$. Mivel fennáll a belső pont feltétel, a 24.12. tétel és a 24.10. lemma miatt $\mathcal{L}_{\hat{\mathbf{w}}}\neq\emptyset$ és kompakt.

Legyen $f:\mathcal{L}_{\hat{\mathbf{w}}}\to\mathbb{R}_+$ függvény, $f(\mathbf{u},\mathbf{z})=\mathbf{u}^T\mathbf{z}$. Az indirekt feltevésünk miatt $f(\mathbf{u},\mathbf{z})=\mathbf{u}^T\mathbf{z}<\mathbf{e}^T\hat{\mathbf{w}}$ minden $(\mathbf{u},\mathbf{z})\in\mathcal{L}_{\hat{\mathbf{w}}}$. Az f folytonos függvény, így Weierstrass tétele miatt felveszi a maximumát az $\mathcal{L}_{\hat{\mathbf{w}}}$ kompakt halmazon, azaz

$$\mathbf{e}^T\overline{\mathbf{w}}=\overline{\mathbf{u}}^T\overline{\mathbf{z}}=f(\overline{\mathbf{u}},\overline{\mathbf{z}})=\max_{(\mathbf{u},\mathbf{z})in\mathcal{L}_{\hat{\mathbf{w}}}} f(\mathbf{u},\mathbf{z})=\max_{(\mathbf{u},\mathbf{z})\in\mathcal{L}_{\hat{\mathbf{w}}}}\mathbf{u}^T\mathbf{z}<\mathbf{e}^T\hat{\mathbf{w}}\ , \qquad (24.7)$$

ahol $\overline{\mathbf{w}}=\overline{\mathbf{u}}\overline{\mathbf{z}}$ és $\overline{\mathbf{w}}\leq\hat{\mathbf{w}}$, de $\overline{\mathbf{w}}\neq\hat{\mathbf{w}}$ az indirekt feltevésünk miatt.
Két eset lehetséges:

1. Ha $\operatorname{int}\mathcal{T}(\overline{\mathbf{w}},\hat{\mathbf{w}})\neq\emptyset$, akkor $\exists\,\mathbf{w}^+\in\mathcal{T}(\overline{\mathbf{w}},\hat{\mathbf{w}})\ :\ \overline{\mathbf{w}}<\mathbf{w}^+<\hat{\mathbf{w}}$ és $\mathbf{w}^+=\mathbf{u}^+\mathbf{z}^+$, ahol $\mathbf{u}^+\in\mathcal{F}^0$. Így $\mathbf{e}^T\overline{\mathbf{w}}<\mathbf{e}^T\mathbf{w}^+$, és ez ellentmond a (24.7) egyenlőtlenségnek, hiszen $(\mathbf{u}^+,\mathbf{z}^+)\in\mathcal{L}_{\hat{\mathbf{w}}}$.
2. Ha $\operatorname{int}\mathcal{T}(\overline{\mathbf{w}},\hat{\mathbf{w}})=\emptyset$, akkor legyen $\delta=\mathbf{e}^T(\hat{\mathbf{w}}-\overline{\mathbf{w}})>0$, és legyen

$$\tilde{\mathbf{w}}\in\mathbb{R}^n_+\ :\ \hat{\mathbf{w}}>\tilde{\mathbf{w}}>\hat{\mathbf{w}}-\frac{\delta}{n}\mathbf{e}\ .$$

Ekkor $\tilde{\mathbf{w}}$ választási szabálya miatt $\mathcal{T}(\overline{\mathbf{w}},\tilde{\mathbf{w}})\neq\emptyset$, azaz létezik $\mathbf{w}^+\in\mathcal{T}(\overline{\mathbf{w}},\tilde{\mathbf{w}})$ úgy, hogy $\mathbf{w}^+=\mathbf{u}^+\mathbf{z}^+$, $\mathbf{u}^+=\overline{\mathbf{u}}+\alpha\Delta\mathbf{u}$, $\mathbf{z}^+=\overline{\mathbf{z}}+\alpha\Delta\mathbf{z}$, ahol $\mathbf{u}^+\in\mathcal{F}^0$ és $\alpha\in$

$(0,1)$. A továbbiakban megmutatjuk, hogy $\mathbf{e}^T\mathbf{w}^+ > \mathbf{e}^T\overline{\mathbf{w}}$, ami ellentmondás (24.7) miatt. Felhasználva a $\Delta\mathbf{u}$, $\Delta\mathbf{z}$ vektorok ortogonalitását, illetve a $\overline{\mathbf{z}}\,\Delta\mathbf{u}+\overline{\mathbf{u}}\,\Delta\mathbf{z} = \tilde{\mathbf{w}} - \overline{\mathbf{u}}\,\overline{\mathbf{z}}$ egyenlőséget, a következőt kapjuk:

$$\begin{aligned}\mathbf{e}^T(\mathbf{w}^+ - \overline{\mathbf{w}}) &= \overline{\mathbf{u}}^T\overline{\mathbf{z}} + \alpha(\overline{\mathbf{z}}^T\Delta\mathbf{u} + \overline{\mathbf{u}}^T\Delta\mathbf{z}) + \alpha^2(\Delta\mathbf{u})^T\Delta\mathbf{z} - \overline{\mathbf{u}}^T\overline{\mathbf{z}} \\ &= \alpha(\overline{\mathbf{z}}^T\Delta\mathbf{u} + \overline{\mathbf{u}}^T\Delta\mathbf{z}) = \alpha \qquad (24.8)\end{aligned}$$

$$mathbfe^T(\tilde{\mathbf{w}} - \overline{\mathbf{w}})\ . \qquad (24.9)$$

$\tilde{\mathbf{w}}$ definíciója miatt

$$\tilde{\mathbf{w}} - \overline{\mathbf{w}} > (\hat{\mathbf{w}} - \overline{\mathbf{w}}) - \frac{\delta}{n}\mathbf{e}, \quad \text{így} \quad \mathbf{e}^T(\tilde{\mathbf{w}} - \overline{\mathbf{w}}) > \mathbf{e}^T(\hat{\mathbf{w}} - \overline{\mathbf{w}}) - \frac{\delta}{n}\mathbf{e}^T\mathbf{e} = \delta - \delta = 0\ .$$

A (24.8) egyenlőséget figyelembe véve $\mathbf{e}^T\mathbf{w}^+ > \mathbf{e}^T\overline{\mathbf{w}}$, amit bizonyítani szerettünk volna, azaz ellentmondásra jutottunk.
A fentiekben beláttuk, hogy $\forall\mathbf{w} \in \mathbb{R}^n_+$ esetén $\exists\,(\mathbf{u},\mathbf{z}) > 0 :\ M\mathbf{u} + \mathbf{q} = \mathbf{z}$, $\mathbf{w} = \mathbf{u}\,\mathbf{z}$.
Az egyértelműség bizonyítását az Olvasóra bízzuk (lásd 24.1-22 gyakorlat). ■

A következőkben megmutatjuk, hogy a centrális út limesz pontja az (SP) feladat egy optimális, sőt szigorúan komplementáris megoldása.

24.15. tétel. *Legyen adott az (SP) feladat, amelyre teljesül a belső pont feltétel. Ekkor létezik* $(\mathbf{u}^*,\mathbf{z}^*)$ *az alábbi tulajdonságokkal:*

1. $(\mathbf{u}^*,\mathbf{z}^*) = \lim\limits_{\mu\to 0}(\mathbf{u}(\mu),\mathbf{z}(\mu))$;
2. $\mathbf{u}^* \in \mathcal{F}^*$;
3. $(\mathbf{u}^*,\mathbf{z}^*)$ *szigorúan komplementáris megoldás.*

Bizonyítás. Az 1. és a 2. állítások bizonyítását (lásd 24.1-23 gyakorlat) az Olvasóra bízzuk.

Az utolsó állítás bizonyításához vezessük be a következő jelölést. Legyen $\mathbf{u} \in \mathbb{R}^n_\oplus$ és jelölje $\sigma(\mathbf{u}) = \{i\ :\ u_i > 0\}$ az $\mathbf{u}$ vektor tartóját. Felhasználva az M mátrix ferdén szimmetrikusságát

$$0 = (\mathbf{u}^* - \mathbf{u}(\mu))^T\,(\mathbf{z}^* - \mathbf{z}(\mu)) = (\mathbf{u}^*)^T\mathbf{z}^* + \mathbf{u}(\mu)^T\mathbf{z}(\mu) - (\mathbf{u}^*)^T\mathbf{z}(\mu) - (\mathbf{z}^*)^T\mathbf{u}(\mu)$$

egyenlet adódik. Figyelembe véve a második állítást és az $u(\mu)_i z(\mu)_i = \mu$ összefüggést

$$\begin{aligned}\mu n &= (\mathbf{u}^*)^T\mathbf{z}(\mu) + (\mathbf{z}^*)^T\mathbf{u}(\mu) = \sum_{i\,:\,u_i^*>0} u_i^*(z(\mu))_i + \sum_{i\,:\,z_i^*>0} z_i^*(u(\mu))_i \\ n &= \sum_{i\,:\,u_i^*>0} u_i^*\frac{(z(\mu))_i}{\mu} + \sum_{i\,:\,z_i^*>0} z_i^*\frac{(u(\mu))_i}{\mu} = \sum_{i\,:\,u_i^*>0}\frac{u_i^*}{(u(\mu))_i} + \sum_{i\,:\,z_i^*>0}\frac{z_i^*}{(z(\mu))_i}\ .\end{aligned}$$

Határátmenettel, amikor $\mu \to 0$, a

$$|\sigma(\mathbf{u}^*)| + |\sigma(\mathbf{z}^*)| = n$$

egyenletet kapjuk, ami csak úgy teljesülhet, ha az $(\mathbf{u}^*, \mathbf{z}^*)$ szigorúan komplementáris megoldás. ■

Az előző tétel harmadik állítása Goldman és Tucker eredménye, ami a következő formában ismert.

24.16. következmény. (Goldman-Tucker-tétel belső pont feltevés mellett). *Legyen adott az (SP) feladat és tegyük fel, hogy $\mathcal{F}^0 \neq \emptyset$. Ekkor létezik szigorúan komplementáris megoldás.*

Optimális partíció, analitikus centrum

Az alábbiakban a centrális út limesz pontját tanulmányozzuk. Ehhez definiáljuk a vektorok indexhalmazának egy partícióját, melyről a következőkben további tulajdonságokat látunk be.

Vezessük be a következő jelöléseket:

$$\begin{aligned}\mathcal{B} &= \{i \ : \ u_i > 0, \quad \text{valamely} \quad u \in \mathcal{F}^* \quad \text{esetén}\}\ , \\ \mathcal{N} &= \{i \ : \ z(u)_i > 0, \quad \text{valamely } u \in \mathcal{F}^* \quad \text{esetén}\}\ .\end{aligned}$$

A fent szereplő $(\mathcal{B}, \mathcal{N})$ felbontás az $\mathcal{I} = \{1, 2, \ldots, n\}$ indexhalmaz egy partíciója, amit az alábbiakban bizonyítunk be.

24.17. állítás. *Legyen adott az (SP) feladat, és tegyük fel, hogy teljesül a belső pont feltétel. Ekkor $\mathcal{I}$ indexhalmaznak a $(\mathcal{B}, \mathcal{N})$ valóban partíciója.*

Bizonyítás. $\mathcal{B} \cup \mathcal{N} = \mathcal{I}$, mert a 24.16. következmény miatt létezik $(\mathbf{u}^*, \mathbf{z}^*)$ szigorúan komplementáris megoldás. A 24.5. lemma miatt pedig $\mathcal{B} \cap \mathcal{N} = \emptyset$ is teljesül. ■

Ezt követően már jogos a következő definíció.

24.18. definíció. *Az $\mathcal{I}$ indexhalmaz $(\mathcal{B}, \mathcal{N})$ felbontását **optimális partíciónak** nevezzük.*

A centrális út az optimális megoldások halmazának egy speciális pontjához tart, amikor μ tart nullához, ez az úgynevezett analitikus centrum, amit a következőképpen definiálunk

24.19. definíció. *Jelölje $\overline{\mathbf{u}} \in \mathcal{F}^*$, $\overline{\mathbf{z}} = z(\overline{\mathbf{u}})$ azon vektorpárt, amely maximalizálja a*

$$\prod_{i\in\mathcal{B}} u_i \prod_{i\in\mathcal{N}} z_i$$

*szorzatot az $\mathcal{F}^*$ optimális halmazon. Ekkor az $\overline{\mathbf{u}}$ vektort az $\mathcal{F}^*$ optimális halmaz **analitikus centrumának** nevezzük.*

A 24.1-20 gyakorlat szerint belső pont feltevés mellett az $\mathcal{F}^*$ halmaz korlátos, így ekkor létezik az optimális halmaz analitikus centruma. Továbbá létezik szigorúan komplementáris megoldás a 24.16. következmény alapján, így a definícióban szereplő szorzat maximumértéke pozitív, tehát az analitikus centrum egyben szigorúan komplementáris megoldás is.

24.20. tétel. (Sonnevend). *Legyen a centrális út határpontja $(\mathbf{u}^*, \mathbf{z}^*)$. Ekkor $\mathbf{u}^*$ az $\mathcal{F}^*$ halmaz analitikus centruma.*

Bizonyítás. Legyen $\overline{\mathbf{u}} \in \mathcal{F}^*$ tetszőleges, és $\overline{\mathbf{z}} = z(\overline{\mathbf{u}})$. A 24.15. tétel bizonyításában leírtakhoz hasonlóan az alábbi összefüggésre jutunk:

$$\sum_{i\in\mathcal{B}} \frac{\overline{u}_i}{u_i^*} + \sum_{i\in\mathcal{N}} \frac{\overline{z}_i}{z_i^*} = n\ .$$

Mivel $u_i^* > 0\ \forall i \in \mathcal{B}$ és $z_i^* > 0\ \forall i \in \mathcal{N}$, alkalmazhatjuk a számtani-mértani közép közti egyenlőtlenséget:

$$\left(\prod_{i\in\mathcal{B}} \frac{\overline{u}_i}{u_i^*} \prod_{i\in\mathcal{N}} \frac{\overline{z}_i}{z_i^*}\right)^{\frac{1}{n}} \le \frac{1}{n}\left(\sum_{i\in\mathcal{B}} \frac{\overline{u}_i}{u_i^*} + \sum_{i\in\mathcal{N}} \frac{\overline{z}_i}{z_i^*}\right) = 1\ .$$

Így megkapjuk a kívánt

$$\prod_{i\in\mathcal{B}} \overline{u}_i \prod_{i\in\mathcal{N}} \overline{z}_i \le \prod_{i\in\mathcal{B}} u_i^* \prod_{i\in\mathcal{N}} z_i^*$$

egyenlőtlenséget. ∎

24.1.3. Erős dualitás tétel

Az előzőekben az (SP) feladatra belső pont feltétel teljesülése mellett már bizonyítottuk a Goldman-Tucker-tételt (24.16. következmény). Célunk általános esetben is belátni a tételt. Vegyük észre, hogy a (24.1) Goldman-Tucker-modell nem teljesítheti a belső pont feltételt a 24.3. tétel miatt. Ezért szükségünk van a (24.1) homogén feladat beágyazására olyan ekvivalens (SP) feladatba, mely teljesíti a belső pont feltételt.

Legyen $\mathbf{u}^0 = \mathbf{z}^0 = \mathbf{e}$. Ezek a vektorok pozitívak, de általában nem elégítik ki a (24.1) feltételeket. Definiáljuk az $\mathbf{r}$ hibavektort

$$\mathbf{r} = \mathbf{e} - M\mathbf{e}\,, \quad \text{és legyen} \quad \lambda = n+1\,.$$

Ekkor

$$\begin{pmatrix} M & \mathbf{r} \\ -\mathbf{r}^T & 0 \end{pmatrix}\begin{pmatrix} \mathbf{e} \\ 1 \end{pmatrix} + \begin{pmatrix} \mathbf{0} \\ \lambda \end{pmatrix} = \begin{pmatrix} M\mathbf{e}+\mathbf{r} \\ -\mathbf{r}^T\mathbf{e}+\lambda \end{pmatrix} = \begin{pmatrix} \mathbf{e} \\ 1 \end{pmatrix}\,.$$

A fenti konstrukció alapján definiáljuk a beágyazási $(\overline{SP})$ feladatot a következőképpen:

$$\min\left\{\lambda\vartheta \;:\; \begin{pmatrix} M & \mathbf{r} \\ -\mathbf{r}^T & 0 \end{pmatrix}\begin{pmatrix} \mathbf{u} \\ \vartheta \end{pmatrix} - \begin{pmatrix} \mathbf{z} \\ \nu \end{pmatrix} = -\begin{pmatrix} \mathbf{0} \\ \lambda \end{pmatrix};\; \begin{pmatrix} \mathbf{u} \\ \vartheta \end{pmatrix}, \begin{pmatrix} \mathbf{z} \\ \nu \end{pmatrix} \geq \mathbf{0}\right\}\,.$$

Ez a feladat kielégíti a belső pont feltételt, mivel a csupa egyesből álló vektor megengedett megoldást ad. Ez a feladat (SP) alakú, ahol

$$\overline{M} = \begin{pmatrix} M & \mathbf{r} \\ -\mathbf{r}^T & 0 \end{pmatrix}\,, \qquad \overline{\mathbf{u}} = \begin{pmatrix} \mathbf{u} \\ \vartheta \end{pmatrix} \qquad \text{és} \qquad \overline{\mathbf{z}} = \begin{pmatrix} \mathbf{z} \\ \nu \end{pmatrix}\,.$$

Mivel az $(\overline{SP})$ feladat célfüggvénye a ϑ változó pozitív többszöröse, így ennek a változónak minden optimális megoldásban a 24.4. lemma miatt nulla az értéke. Tehát az $(\mathbf{u}, \vartheta, \mathbf{z}, \nu)$ akkor és csak akkor szigorúan komplementáris megoldása az $(\overline{SP})$ feladatnak, ha $(\mathbf{u}, \mathbf{z})$ szigorúan komplementáris megoldása a (24.1) Goldman-Tucker-modellnek. Figyelembe véve, hogy az $(\overline{SP})$ feladat kielégíti a belső pont feltételt, a 24.15. tétel miatt létezik szigorúan komplementáris megoldása, ezzel beláttuk a Goldman-Tucker-tételt általános esetben is.

Az eddigieket összefoglalva: Minden lineáris programozási feladat beágyazható egy, az (SP) alakban adott önduális $(\overline{SP})$ feladatba oly módon, hogy $(\overline{\mathbf{u}}, \overline{\mathbf{z}}) = (\mathbf{e}, 1, \mathbf{e}, 1)$, azaz a csupa egyesből álló vektor az $(\overline{SP})$ feladat megengedett megoldása. Továbbá a (24.1) ferdén szimmetrikus feladat bármely erősen komplementáris megoldása vagy az eredeti lineáris programozási feladat egy optimális megoldását adja, vagy bizonyítja, hogy az adott lineáris programozási feladatnak nincs optimális megoldása.

A fenti eredmények segítségével könnyen bebizonyítható az úgynevezett erős dualitás tétel.

24.21. tétel. (erős dualitás tétel). *Legyenek a (P) és a (D) feladatok adottak. Ekkor a következő két alternatíva valamelyike teljesül:*

1. *$\mathcal{P} \neq \emptyset$ és $\mathcal{D} \neq \emptyset$, valamint létezik $(\tilde{\mathbf{x}}, \tilde{\mathbf{y}})$ szigorúan komplementáris megoldás.*

2. *$\mathcal{P}^* = \emptyset$ és $\mathcal{D}^* = \emptyset$, ez pontosan akkor fordulhat elő, ha létezik $(\overline{\mathbf{x}}, \overline{\mathbf{y}})$, melyre*

$$A\overline{\mathbf{x}} \geq \mathbf{0}\,, \quad A^T\overline{\mathbf{y}} \leq \mathbf{0}\,, \quad \mathbf{c}^T\overline{\mathbf{x}} \leq \mathbf{b}^T\overline{\mathbf{y}}\,.$$

Bizonyítás. A 24.16. következmény miatt a lineáris programozási feladat Goldman-Tucker-rendszerének van egy erősen komplementáris megoldása. Egy ilyen megoldásban vagy $\xi > 0$, és ebben az esetben a 24.3. tétel 3. pontjából következik, hogy létezik optimális megoldás pár nulla dualitásréssel, vagy $\zeta > 0$ a Goldman-Tucker-rendszer erősen komplementáris megoldásában. Az utóbbi esetben a 24.3. tétel 4. pontja miatt (P) vagy (D) vagy mindkettő nemmegengedett. ■

A lineáris programozás fontos és alapvető eredménye a több mint 100 éves Farkas-lemma, melyre a Goldman-Tucker-tétel segítségével most egy újabb bizonyítást adunk.

24.22. lemma. (Farkas-lemma). *A következő két egyenlőtlenségrendszer közül pontosan az egyiknek van megoldása*

$$\left.\begin{array}{rl} A\mathbf{x} & \geq \mathbf{b}\,, \\ \mathbf{x} & \geq \mathbf{0} \end{array}\right\} \qquad \textit{és} \qquad \left.\begin{array}{rl} A^T\mathbf{y} & \leq \mathbf{0}\,, \\ \mathbf{b}^T\mathbf{y} & > 0\,, \\ \mathbf{y} & \geq \mathbf{0} \end{array}\right\}\,,$$

ahol $A \in \mathbb{R}^{m\times n}$, $\mathbf{b} \in \mathbb{R}^m$, $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{y} \in \mathbb{R}^m$.

Bizonyítás. Tekintsük a következő primál és duál lineáris programozási feladatot

$$\begin{array}{ll} (P) & \min\left\{\mathbf{0}^T\mathbf{x} \;:\; A\mathbf{x} \geq \mathbf{b},\;\; \mathbf{x} \geq \mathbf{0}\right\}\,, \\ (D) & \max\left\{\mathbf{b}^T\mathbf{y} \;:\; A^T\mathbf{y} \leq \mathbf{0},\;\; \mathbf{y} \geq \mathbf{0}\right\}\,. \end{array}$$

Elkészíthető a (P) és (D) feladathoz az (SP) ferdén szimmetrikus önduális feladat

$$\left.\begin{array}{rrrl} \min\ \mathbf{0}^T\mathbf{y} & +\mathbf{0}^T\mathbf{x} & +0\,\xi & \\ & A\mathbf{x} & -\mathbf{b}\,\xi & \geq \mathbf{0}\,, \\ -A^T\mathbf{y} & & +\mathbf{0}\,\xi & \geq \mathbf{0}\,, \\ \mathbf{b}^T\mathbf{y} & -\mathbf{0}^T\mathbf{x} & & \geq \mathbf{0}\,, \\ & & \mathbf{y},\mathbf{x},\xi & \geq \mathbf{0} \end{array}\right\} \quad (SP)\,.$$

Az előző tételek alapján ennek a feladatnak létezik szigorúan komplementáris megoldása, $(\overline{\mathbf{y}}, \overline{\mathbf{x}}, \overline{\xi})$. Vezessük be a következő jelöléseket:

$$s(\overline{\mathbf{y}}) = -A^T\overline{\mathbf{y}}\,, \qquad s(\overline{\mathbf{x}}) = A\overline{\mathbf{x}} - \mathbf{b}\overline{\xi}\,, \qquad s(\overline{\xi}) = \mathbf{b}^T\overline{\mathbf{y}}\,.$$

Ekkor két eset lehetséges: (i) $\overline{\xi} > 0$, illetve (ii) $\overline{\xi} = 0$.

Az (i) esetben $\overline{\mathbf{x}} \geq \mathbf{0}$, $s(\overline{\mathbf{x}}) \geq \mathbf{0}$, így $A\overline{\mathbf{x}} > \mathbf{b}\overline{\xi}$, tehát az $\overline{\mathbf{x}}/\overline{\xi}$ megoldása az első egyenletrendszernek. Az (ii) esetben – mivel $\overline{\xi} = 0 - s(\overline{\xi}) > 0$, tehát $\mathbf{b}^T\overline{\mathbf{y}} > 0$, $\overline{\mathbf{y}} \geq \mathbf{0}$. Valamint $s(\overline{\mathbf{y}}) \geq \mathbf{0}$ miatt $A^T\overline{\mathbf{y}} \leq 0$, azaz ebben az esetben az $\overline{\mathbf{y}}$ vektor megoldása a második rendszernek. A két rendszernek nem lehet egyszerre megoldása, ami elemien belátható. ■

Gyakorlatok

24.1-1. Legyen $\mathbf{x}$ és $\mathbf{y}$ primál, illetve duál megengedett megoldás, ekkor a következő két állítás ekvivalens:

1. $\mathbf{c}^T\mathbf{x} - \mathbf{b}^T\mathbf{y} = (\mathbf{c} - A^T\mathbf{y})^T\mathbf{x} + \mathbf{y}^T(A\,\mathbf{x} - \mathbf{b}) = 0$, illetve
2. $(\mathbf{c} - A^T\mathbf{y})\,\mathbf{x} = \mathbf{0}$ és $\mathbf{y}\,(A\,\mathbf{x} - \mathbf{b}) = \mathbf{0}$.

24.1-2. Bizonyítsuk be a *gyenge egyensúlyi tételt.*

24.1-3. Bizonyítsuk be a 24.3. tétel 1. és 3. állítását.

24.1-4. Legyen $A \in \mathbb{R}^{m\times k}, \mathbf{b} \in \mathbb{R}^m, \mathbf{c} \in \mathbb{R}^k$ adott mátrix, illetve vektorok. Tekintsük a (P) és (D) kanonikus lineáris programozási feladatokat. Tegyük fel, hogy létezik $\bar{\mathbf{x}}$ megengedett megoldása a (P) feladatnak, amelyre az $A\,\bar{\mathbf{x}} \geq \mathbf{0}$ és $\mathbf{c}^T\bar{\mathbf{x}} < 0$ összefüggések is teljesülnek. Ekkor nem létezik $\mathbf{y} \in \mathbb{R}^m$ megengedett megoldása a (D) feladatnak.

24.1-5. Legyen $A \in \mathbb{R}^{m\times k}, \mathbf{b} \in \mathbb{R}^m, \mathbf{c} \in \mathbb{R}^k$ adott mátrix, illetve vektorok. Bizonyítsuk be, hogy az

$$M = \begin{pmatrix} 0 & A & -\mathbf{b} \\ -A^T & 0 & \mathbf{c} \\ \mathbf{b}^T & -\mathbf{c}^T & 0 \end{pmatrix},$$

mátrix ferdén szimmetrikus, azaz $M = -M^T$.

24.1-6. Bizonyítsuk be, hogy az (SP) feladat önduális.

24.1-7. Bizonyítsuk be a 24.4. lemmát.

24.1-8. Tekintsük a következő lineáris programozási feladatot

$$\begin{array}{lrrrrrl} \min & -17\,x_1 & +13\,x_2 & +2\,x_3 & -8\,x_4 & & \\ & 2\,x_1 & - \;\; x_2 & & - \;\; x_4 & \leq & 4 \\ & - \;\; x_1 & & + \;\; x_3 & - \;\; x_4 & \geq & -3 \\ & x_1 & +3\,x_2 & -2\,x_3 & & \leq & 4\,, \end{array}$$

ahol $x_1, x_2, x_3 \geq 0$.

a. Írjuk fel a primál és duál lineáris programozási feladatot kanonikus alakban.

b. Adjuk meg a komplementaritási feltételeket.

c. Írjuk fel a Goldman–Tucker-feladatot. Határozzuk meg az M mátrixot.

24.1-9. Tekintsük a következő lineáris programozási feladatot

$$\begin{array}{lrrrrrrl} \min & & & & & & x_6 & \\ & & 2\,x_2 & +x_3 & +\ x_4 & -x_5 & & = \ 0 \\ & 3\,x_1 & -2\,x_2 & -x_3 & +2\,x_4 & & & \leq 15 \\ & x_1 & & +x_3 & & & & \leq \ 5 \\ & -9\,x_1 & +8\,x_2 & +x_3 & -2\,x_4 & & -x_6 & = \ 0\,, \end{array}$$

ahol $x_1, x_2, \ldots, x_5 \geq 0$ és az x_6 előjelkötetlen változó.

a. Írjuk fel a primál és duál lineáris programozási feladatot kanonikus alakban.

b. Adjuk meg a komplementaritási feltételeket.

c. Írjuk fel a Goldman-Tucker-feladatot. Határozzuk meg az M mátrixot.

24.1-10. Legyenek a következő adatok egy kanonikus primál feladat adatai

$$\mathbf{c} = \begin{pmatrix} -9 \\ 8 \\ 1 \\ -2 \end{pmatrix}, \qquad A = \begin{pmatrix} 3 & -2 & -1 & 2 \\ 1 & 0 & 1 & 0 \end{pmatrix}, \qquad \mathbf{b} = \begin{pmatrix} 15 \\ 5 \end{pmatrix}.$$

a. Írjuk fel a kanonikus primál feladatot.

b. Keressünk egy pozitív primál megoldást.

c. Írjuk fel a duál feladatot és keressünk egy (lehetőleg pozitív) duál megoldást. *Útmutatás.* Ha nem tudunk duál megengedett megoldást találni, akkor próbáljunk meg olyan primál megengedett megoldást találni, amelyikre $A\,\mathbf{x} \geq \mathbf{0}$ és $\mathbf{c}^T\mathbf{x} < 0$ teljesül. Mit jelent ez?

d. Írjuk fel a Goldman-Tucker-feladatot és keressünk egy pozitív megoldást, mely az első hat feltételt kielégíti. Mit jelent az, ha ilyen nem létezik? Vizsgáljuk meg, hogy teljesül-e a hetedik feltétel is?

24.1-11. Legyenek a következő adatok egy kanonikus primál feladat adatai

$$\mathbf{c} = \begin{pmatrix} \frac{19}{10} \\ \frac{9}{5} \\ 3 \\ \frac{2}{5} \end{pmatrix}, \qquad A = \begin{pmatrix} \frac{1}{10} & \frac{11}{10} & -\frac{3}{5} \\ 1 & -\frac{3}{10} & -\frac{4}{5} \\ -\frac{1}{5} & 0 & \frac{4}{5} \end{pmatrix}, \qquad \mathbf{b} = \begin{pmatrix} -\frac{2}{5} \\ -\frac{11}{10} \\ -\frac{2}{5} \end{pmatrix}.$$

a. Írjuk fel a kanonikus primál feladatot.

b. Keressünk egy pozitív primál megoldást.

c. Írjuk fel a duál feladatot és keressünk egy pozitív duál megoldást.

d. Mi következik abból, hogy létezik belső pont mind a primál, mind pedig a duál feladat esetén?

e. Írjuk fel a Goldman-Tucker-feladatot és keressünk egy olyan pozitív megoldást, amely kielégíti az első hat feltételt, ám a ζ értéke -6 lesz. (Ha a dualitásrést a következő alakban írjuk fel, akkor $\mathbf{x}^T\mathbf{s} + \mathbf{y}^T\mathbf{t} = 6$ kell, hogy teljesüljön a megoldására.)

24.1-12. Bizonyítsuk be a 24.7. állítást. *Útmutatás.* Alkalmazzunk indirekt bizonyítást. Tegyük fel, hogy az $\bar{M}$ mátrix szinguláris.

24.1-13. Bizonyítsuk be, hogy ha U, Z pozitív diagonális, továbbá M ferdén szimmetrikus mátrix, akkor $Z + UM$ reguláris mátrix.

24.1-14. Tekintsük a 24.8. állítást.

a. Határozzuk meg a legnagyobb α lépéshosszt, melyre az $\mathbf{x}^+ > \mathbf{0}$ és $\mathbf{s}^+ > \mathbf{0}$ teljesül.

b. A $\mathbf{w}^+ \in \operatorname{int}\mathcal{T}$ elvárásból, koordinátánként a következő két összefüggés valamelyikét nyerjük:

$$w_i < w_i^+ < \hat{w}_i \quad \text{vagy} \quad \hat{w}_i < w_i^+ < w_i \,.$$

Milyen lépéshossz korlátok számíthatók ki az első, illetve második egyenlőtlenségből?

24.1-15. Tekintsük a 24.1-11 gyakorlatban szereplő lineáris programozási feladatot.

a. Írjuk fel a centrális út feladatot arra a megoldásra, amelyikhez tartozó dualitásrés értéke 6 és határozzuk meg a μ paraméter értékét.

b. Legyenek $\mathbf{w}^T = (1,1,1,1,1,1)$ és $\hat{\mathbf{w}}^T = (1/2,1/2,1/2,2,2,2)$ adott pontok. Tegyünk egy Newton-lépést az $\mathbf{u}$ és $\mathbf{z}$ pontokból a $\mathbf{u}\,\mathbf{z} = \mathbf{w}$ pontból a $\hat{\mathbf{w}}$ pont felé, azaz számítsuk ki a $\Delta\mathbf{u}$ és $\Delta\mathbf{z}$ vektorokat. Legyen az M mátrix a Goldman-Tucker-feladatnál, az előző adatokkal meghatározott mátrixnak, a 6×6-os bal felső részmátrixa, míg a $\mathbf{q}^T = (\mathbf{b}^T\,, -\mathbf{c}^T)$. *Útmutatás.* A számításokat a MATLAB programcsomaggal végezzük, mert úgy gyorsabb.

c. Határozzuk meg az α lépéshosszt úgy, hogy az új vektorok a feladat szigorúan pozitív megoldásai legyenek.

24.1-16. Bizonyítsuk be a 24.9. lemmát. *Útmutatás.* Számoljuk ki az $(\mathbf{u} - \bar{\mathbf{u}})^T M\,(\mathbf{u} - \bar{\mathbf{u}})$ értéket, ahol $\bar{\mathbf{u}} \in \mathcal{F}^0$ és $(\mathbf{u},\mathbf{z}) \in \mathcal{L}_K$, majd pedig számítsunk ki korlátokat az u_j és z_j koordinátákra.

24.1-17. Bizonyítsuk be a 24.10. lemmát. *Útmutatás.* Hasonló az előző gyakorlathoz, de itt vegyük figyelembe az $\mathcal{L}_{\mathbf{w}}$ általánosított szinthalmaz definícióját és azt is, hogy az $f : \mathbb{R}_+^{2n} \to \mathbb{R}_+^n$, $f(\mathbf{x},\mathbf{s}) = \mathbf{x}\,\mathbf{s}$ függvény folytonos. Milyen az f függvény ősképe?

24.1-18. Bizonyítsuk be a 24.11. lemmát. *Útmutatás.* Azt kell belátni, hogy a $\mathcal{G}$ halmaz tartalmazza az összes limeszpontját.

24.1-19. Bizonyítsuk be, hogy a 24.12. tétel bizonyításában szereplő f függvény esetén $\|f\|_\infty$ felveszi a minimumát az $\mathcal{A} \cap \mathcal{G}$ halmazon. *Útmutatás.*

Bizonyítsuk, be, hogy $\|f\|_\infty$ függvény folytonos, majd alkalmazzuk rá a Weierstrass-tételt.
indexWeierstrass, Karl Theodor Wilhelm (1815–1897)
24.1-20. Legyen adott az (SP) feladat és tegyük fel, hogy $\mathcal{F}^0 \neq \emptyset$. Bizonyítsuk be, hogy az $\mathcal{F}^*$ halmaz kompakt.
24.1-21. Legyen $M \in \mathbb{R}^{n\times n}$ egy ferdén szimmetrikus mátrix. Bizonyítsuk be, hogy bármely $\mathbf{z} \in \mathbb{R}^n \setminus \{0\}$ vektor esetén létezik olyan j index, melyre $z_j \neq 0$ és $z_j(M\mathbf{z})_j \geq 0$. *Útmutatás.* Legegyszerűbb indirekt bizonyítást adni az állításra.
24.1-22. Bizonyítsuk be a 24.14. tétel 2. állításában szereplő $(\mathbf{u}, \mathbf{z})$ vektorok egyértelműségét. *Útmutatás.* Alkalmazzunk indirekt bizonyítást és használjuk fel az előző gyakorlat állítását.
24.1-23. Bizonyítsuk be a 24.15. tétel 1. és 2. állítását.

24.2. Dikin-féle affin skálázású algoritmus

Az előző részben megmutattuk, hogy ha létezik belső pont, akkor a centrális utat követve a ferdén szimmetrikus önduális feladat megoldásához tartunk. Ennek a résznek a célja annak bizonyítása, hogy ezt a gondolatmenetet követve *polinom időben* a ferdén szimmetrikus önduális feladat megoldásához tetszőlegesen közeli pontot tudunk előállítani, azaz tetszőleges $\varepsilon > 0$ számhoz olyan $(\mathbf{u}, \mathbf{z})$ pontpárt, amely a megengedett tartományban van és $\mathbf{u}^T\mathbf{z} \leq \varepsilon$. E célból egy konkrét algoritmust, egy affin skálázású primál-duál módszert ismertetünk, melynek első változatát Dikin orosz matematikus 1967-ben fogalmazta meg. Ezt követően a $(\mathcal{B}, \mathcal{N})$ partíció meghatározásáról ejtünk pár szót, majd ennek segítségével ismertetjük a kerekítési eljárást, azaz azt, hogy a Dikin-módszerrel kapott jó közelítő megoldásból milyen módon lehet egy szigorúan komplementáris megoldást előállítani (a szakirodalomban például az ellipszoid módszer esetén is találkozhatunk kerekítési eljárással). Ezzel eloszlatjuk azt a tévhitet, ami a mai napig a köztudatban él, miszerint lineáris optimalizálási feladat pontos megoldása belsőpontos módszerrel polinom időben nem állítható elő.

Csökkenési irány meghatározása

Tekintsük az (SP) feladatot, és legyen $\mathbf{u} \in \mathcal{F}^0$, $\mathbf{z} = z(\mathbf{u}) > 0$. Keressük $(\Delta\mathbf{u}, \Delta\mathbf{z}) \in \mathbb{R}^{2n}$:

$$\begin{aligned} -M\Delta\mathbf{u} + \Delta\mathbf{z} &= \mathbf{0}\,, \\ \mathbf{u} + \Delta\mathbf{u},\ \mathbf{z} + \Delta\mathbf{z} &\geq \mathbf{0}\,. \end{aligned}$$

Ennél mi többet szeretnénk elérni, mégpedig: $\mathbf{u}^+ = \mathbf{u} + \Delta\mathbf{u} > \mathbf{0}$ és $\mathbf{z}^+ =$

$\mathbf{z}+\Delta\mathbf{z}>\mathbf{0}$, valamint, hogy közben csökkenjen a dualitásrés is. Felhasználva, hogy M ferdén szimmetrikus mátrix, az új dualitásrés értéke:

$$\begin{aligned}\mathbf{q}^T\mathbf{u}^+ &= (\mathbf{u}^+)^T\mathbf{z}^+ = (\mathbf{u}+\Delta\mathbf{u})^T(\mathbf{z}+\Delta\mathbf{z}) = \mathbf{u}^T\mathbf{z}+\mathbf{z}^T\Delta\mathbf{u}+\mathbf{u}^T\Delta\mathbf{z}+(\Delta\mathbf{u})^T\Delta\mathbf{z} = \\ &= \mathbf{u}^T\mathbf{z}+\mathbf{z}^T\Delta\mathbf{u}+\mathbf{u}^T\Delta\mathbf{z} = \mathbf{q}^T\mathbf{u}+\mathbf{z}^T\Delta\mathbf{u}+\mathbf{u}^T\Delta\mathbf{z}\ .\end{aligned}$$

A dualitásrés csökkenésének a feltétele az, hogy

$$\mathbf{q}^T\mathbf{u}^+ < \mathbf{q}^T\mathbf{u}, \qquad \text{azaz} \qquad \mathbf{z}^T\Delta\mathbf{u}+\mathbf{u}^T\Delta\mathbf{z}<0\ .$$

Ebben az esetben a $(\Delta\mathbf{u},\Delta\mathbf{z})\in\mathbb{R}^{2n}$ vektort ***csökkenési iránynak*** nevezzük. Célunk a (lokálisan) legjobb csökkenési irány keresése, ami a következőképpen fogalmazható meg:

$$\left.\begin{aligned}\min\{\mathbf{z}^T\Delta\mathbf{u}+\mathbf{u}^T\Delta\mathbf{z}\}\\ -M\Delta\mathbf{u}+\Delta\mathbf{z}=\mathbf{0}\\ \mathbf{u}+\Delta\mathbf{u}\geq\mathbf{0}\ ,\ \mathbf{z}+\Delta\mathbf{z}\geq\mathbf{0}\end{aligned}\right\}\quad (IF)\ .$$

A fenti ***iránymeghatározási segédfeladatot*** szeretnénk megoldani, de ez egy – az eredetivel azonos „nehézségű” – lineáris programozási feladat. Nyilvánvaló, hogy egy olyan számítási eljárás, amelynek egy-egy iterációjában az eredeti feladattal ekvivalens segédfeladatot kellene megoldani, nem igazán kecsegtet sikerrel. Régóta köztudott viszont, hogy lineáris függvényt könnyen lehet minimalizálni (zárt) gömb felett (lásd 24.2-1 gyakorlat), hiszen legyen $\mathbf{c}\in\mathbb{R}^n$, akkor a

$$\min_{\|\mathbf{u}\|\leq 1}\ \mathbf{c}^T\mathbf{u} \tag{24.10}$$

feladat optimális megoldása az $\mathbf{u}=-\mathbf{c}/\|\mathbf{c}\|$. Másfelől egy tetszőleges adott dimenziós, zárt ellipszoid, egy ugyanolyan dimenziós zárt gömbbé transzformálható át (lásd 24.2-2 gyakorlat).

Dikin ötlete (1967): Keressünk egy könnyebben megoldható feladatot, amely dualitásrés csökkenést biztosít. Tegyük ezt úgy, hogy a pozitivitási megkötést „relaxáljuk”. A közelítést egy (konvex) zárt ellipszoiddal oldjuk meg.

Legyen $\mathbf{u}$, $\mathbf{z}>0$, és ekkor a ***Dikin-ellipszoid:***

$$\mathcal{E}_D(\mathbf{u},\mathbf{z}) = \left\{(\bar{\mathbf{u}},\bar{\mathbf{z}})\in\mathbb{R}^{2n}\ :\ \bar{\mathbf{u}}=\mathbf{u}+\Delta\mathbf{u},\ \bar{\mathbf{z}}=\mathbf{z}+\Delta\mathbf{z},\ \left\|\frac{\Delta\mathbf{u}}{\mathbf{u}}+\frac{\Delta\mathbf{z}}{\mathbf{z}}\right\|\leq 1\right\}\ .$$

Legyen

$$\mathcal{F}_z^0 = \{(\mathbf{u},\mathbf{z})\in\mathbb{R}^{2n}\ :\ \mathbf{u}\in\mathcal{F}^0\ ,\ \mathbf{z}>\mathbf{0}\}\ .$$

Ekkor $\mathcal{F}_z^0 \cap \mathcal{E}_D(\mathbf{u},\mathbf{z}) \neq \emptyset$, mert $(\mathbf{u},\mathbf{z}) \in \mathcal{F}_z^0 \cap \operatorname{int} \mathcal{E}_D(\mathbf{u},\mathbf{z})$. Belátható, hogy $\mathcal{F}_z^0 \cap \mathcal{E}_D(\mathbf{u},\mathbf{z})$ egy nem üres kompakt halmaz.

Legyen $(\mathbf{u}+\Delta\mathbf{u}, \mathbf{z}+\Delta\mathbf{z}) \in \mathcal{F}_z^0 \cap \mathcal{E}_D(\mathbf{u},\mathbf{z})$, ekkor

$$1 \geq \left\| \frac{\Delta\mathbf{u}}{\mathbf{u}} + \frac{\Delta\mathbf{z}}{\mathbf{z}} \right\| = \left\| \frac{\mathbf{z}\Delta\mathbf{u} + \mathbf{u}\Delta\mathbf{z}}{\mathbf{uz}} \right\| = \left\| \frac{\mathbf{z}\Delta\mathbf{u} + \mathbf{u}M\Delta\mathbf{u}}{\mathbf{uz}} \right\| =$$

$$= \|(XS)^{-1}(S+XM)\Delta\mathbf{u}\| = \|S^{-1}(X^{-1}S+M)\Delta\mathbf{u}\| \,.$$

A 24.2-3 és 24.2-4 gyakorlat alapján $S^{-1}(X^{-1}S+M)$ pozitív definit mátrix, ezért az

$$\|S^{-1}(X^{-1}S+M)\Delta\mathbf{u}\|^2$$

a $\Delta\mathbf{u}$ konvex kvadratikus függvénye, azaz

$$\{\Delta\mathbf{u} : \|S^{-1}(X^{-1}S+M)\Delta\mathbf{u}\| \leq 1\}$$

korlátos konvex nemüres zárt halmaz. Vagyis az (IF) feladat relaxálható a következő módon:

$$\left.\begin{array}{r} \min\{\mathbf{z}^T\Delta\mathbf{u} + \mathbf{u}^T M\Delta\mathbf{u}\} \\ \|S^{-1}(X^{-1}S+M)\Delta\mathbf{u}\| \leq 1 \end{array}\right\} \quad (\overline{IF}) \,.$$

Az $(\overline{IF})$ lineáris célfüggvény minimalizálása konvex kompakt nem üres halmaz felett, így a halmaz korlátosságát is figyelembe véve, a Weierstrass-tétel miatt létezik megoldása.

Átskálázás

Az $(\overline{IF})$ feladat helyett tekinthetjük a

$$\left.\begin{array}{r} \min\{\mathbf{z}^T\Delta\mathbf{u} + \mathbf{u}^T\Delta\mathbf{z}\} \\ -M\Delta\mathbf{u} + \Delta\mathbf{z} = \mathbf{0}, \\ \|\frac{\Delta\mathbf{u}}{\mathbf{u}} + \frac{\Delta\mathbf{z}}{\mathbf{z}}\| \leq 1 \end{array}\right\} \quad (SF)$$

segédfeladatot, amelyik esetén az első feltétel egy homogén lineáris egyenletrendszer, míg a második egy zárt elliptikus henger, amellyel a nemnegativitási feltételeket relaxáltuk.

A célunk az, hogy olyan ***skálázást*** (koordináta transzformációt) vezessünk be, amely után (SF) lineáris célfüggvényét egy gömbön kell minimalizálnunk.

Legyen

$$\mu = \frac{\mathbf{u}^T\mathbf{z}}{n} \,, \quad \mathbf{d} = \sqrt{\frac{\mathbf{u}}{\mathbf{z}}} \,, \quad \mathbf{v} = \sqrt{\frac{\mathbf{uz}}{\mu}} \,.$$

Ekkor $\sqrt{\mathbf{uz}} = \sqrt{\mu}\mathbf{v}$ és $\mathbf{d}^{-1}\mathbf{u} = \sqrt{\mu}\mathbf{v} = \mathbf{dz}$. Vezessük be a $\mathbf{p}_u$ és $\mathbf{p}_z$ vektorokat úgy, hogy

$$\mathbf{p}_u = \frac{\mathbf{d}^{-1}\Delta\mathbf{u}}{\sqrt{\mu}} \quad \text{és} \quad \mathbf{p}_z = \frac{\mathbf{d}\Delta\mathbf{z}}{\sqrt{\mu}} \ .$$

Ekkor

$$\frac{\Delta\mathbf{u}}{\mathbf{u}} = \frac{\mathbf{d}^{-1}\Delta\mathbf{u}}{\mathbf{d}^{-1}\mathbf{u}} = \frac{\sqrt{\mu}\mathbf{p}_u}{\sqrt{\mu}\mathbf{v}} = \frac{\mathbf{p}_u}{\mathbf{v}} \quad \text{és} \quad \frac{\Delta\mathbf{z}}{\mathbf{z}} = \frac{\mathbf{d}\Delta\mathbf{u}}{\mathbf{du}} = \frac{\sqrt{\mu}\mathbf{p}_z}{\sqrt{\mu}\mathbf{v}} = \frac{\mathbf{p}_z}{\mathbf{v}} \ ,$$

amiből a

$$\frac{\Delta\mathbf{u}}{\mathbf{u}} + \frac{\Delta\mathbf{z}}{\mathbf{z}} = \frac{\mathbf{p}_u + \mathbf{p}_z}{\mathbf{v}}$$

összefüggés következik. Bevezetve a $\mathbf{p} = \mathbf{p}_u + \mathbf{p}_z$ jelölést a

$$\frac{\mathbf{p}}{\mathbf{v}} = \frac{\Delta\mathbf{u}}{\mathbf{u}} + \frac{\Delta\mathbf{z}}{\mathbf{z}} \ ,$$

illetve a

$$\mathbf{z}\Delta\mathbf{u}+\mathbf{u}\Delta\mathbf{z} = (\mathbf{zd})(\mathbf{d}^{-1}\Delta\mathbf{u})+(\mathbf{ud}^{-1})(\mathbf{d}\Delta\mathbf{z}) = \sqrt{\mu}\mathbf{v}\sqrt{\mu}\mathbf{p}_u+\sqrt{\mu}\mathbf{v}\sqrt{\mu}\mathbf{p}_z = \mu\mathbf{vp}$$

egyenleteket kapjuk. Tehát az (SF) átskálázott alakja rögzített μ paraméter mellett

$$\left.\begin{array}{r} \min\ \mu\mathbf{v}^T\mathbf{p} \\ \|\frac{\mathbf{p}}{\mathbf{v}}\| \le 1 \ . \end{array}\right\}$$

Legyen $\bar{\mathbf{p}} = \mathbf{p}/\mathbf{v}$. Ekkor az (SF) a következő alakot ölti:

$$\left.\begin{array}{r} \min\ \mu(\mathbf{v}^2)^T\bar{\mathbf{p}} \\ \|\bar{\mathbf{p}}\| \le 1 \ , \end{array}\right\}$$

ami gömb felett egy lineáris függvény minimalizálása, tehát létezik egyértelmű minimuma:

$$\bar{\mathbf{p}} = -\frac{\mathbf{v}^2}{\|\mathbf{v}^2\|} \ , \quad \text{és ekkor} \quad \mathbf{p} = -\frac{\mathbf{v}^3}{\|\mathbf{v}^2\|} \ .$$

A $(\Delta\mathbf{u}, \Delta\mathbf{z})$ csökkenési irány meghatározása
Mivel $\mathbf{0} = -M\Delta\mathbf{u} + \Delta\mathbf{z} = -MD(D^{-1}\Delta\mathbf{u}) + D^{-1}(D\Delta\mathbf{z}) = -\sqrt{\mu}(MD\mathbf{p}_u - D^{-1}\mathbf{p}_z)$, ezért

$$\mathbf{p}_z = DMD\mathbf{p}_u \ , \quad \text{és így } amathbfp = \mathbf{p}_u + \mathbf{p}_z = (I + DMD)\mathbf{p}_u \ .$$

A 24.2-3 gyakorlat szerint $I + DMD$ pozitív definit mátrix, így

$$\mathbf{p}_u = (I + DMD)^{-1}\mathbf{p} \quad \text{és} \quad \mathbf{p}_z = DMD(I + DMD)^{-1}\mathbf{p} \ .$$

Figyelembe véve a $\mathbf{d}^{-1}\Delta\mathbf{u} = \sqrt{\mu}\mathbf{p}_u$ és $\mathbf{d}\Delta\mathbf{z} = \sqrt{\mu}\mathbf{p}_z$ összefüggéseket

$$\Delta\mathbf{u} = \sqrt{\mu}D(I + DMD)^{-1}\mathbf{p} \quad \text{és} \quad \Delta\mathbf{z} = \sqrt{\mu}MD(I + DMD)^{-1}\mathbf{p}$$

adódik. Az így kapott $(\Delta\mathbf{u}, \Delta\mathbf{z})$ vektort ***Dikin-iránynak*** nevezzük.
Ezt meglépve a dualitásrés csökkenése:

$$\mu(\mathbf{v}^2)^T\bar{\mathbf{p}} = \mu(\mathbf{v}^2)^T\left(-\frac{\mathbf{v}^2}{\|\mathbf{v}^2\|}\right) = -\mu\frac{\|\mathbf{v}^2\|^2}{\|\mathbf{v}^2\|} = -\mu\|\mathbf{v}^2\| = -\mu\left\|\frac{\mathbf{uz}}{\mu}\right\| = -\|\mathbf{uz}\| .$$

A $\Delta\mathbf{u}$ vektorra kapott kifejezésbe behelyettesítve $\mathbf{d}$, $\mathbf{p}$, illetve $\mathbf{p}$ képletébe $\mathbf{v}$ definícióját, könnyen adódik a következő állítás.

24.23. állítás. *A fenti jelölésekkel*

$$\Delta\mathbf{u} = \sqrt{\mu}D(I + DMD)^{-1}\mathbf{p} = -(S + XM)^{-1}\frac{\mathbf{u}^2\mathbf{z}^2}{\|\mathbf{uz}\|} \qquad \textit{teljesül.}$$

A 24.23. állítás első egyenlőségét már bizonyítottuk, a másodiknak a bizonyítását, amelyik az eredeti adatokat használja, az Olvasóra bízzuk (lásd 24.2-5 gyakorlat).

Az algoritmus során szükségünk van a pontunk pozitivitásának megtartására, melyet a következő ***centralitási mérték***

$$\partial_c(\mathbf{u}) = \frac{\max(\mathbf{u}z(\mathbf{u}))}{\min(\mathbf{u}z(\mathbf{u}))} = \frac{\max(\mathbf{v}^2)}{\min(\mathbf{v}^2)} = \delta_c(\mathbf{v}) , \qquad \mathbf{u} \in \mathcal{F}^0$$

segítségével teszünk meg, ahol $\max(\mathbf{v})$, illetve $\min(\mathbf{v})$ a $\mathbf{v}$ vektor legnagyobb, illetve legkisebb koordinátáját jelöli. Ekkor $\partial_c(\mathbf{u}) \geq 1$ teljesül bármely $\mathbf{u} \in \mathcal{F}^0$, $\mathbf{z} = z(\mathbf{u}) > \mathbf{0}$ esetén, valamint $\partial_c(\mathbf{u}) = 1$ pontosan akkor, ha $\mathbf{u} \in \mathcal{F}^0$: $\mathbf{u}\,z(\mathbf{u}) = \mu\,\mathbf{e}$, azaz a minimális centralitási mértékű megoldások a centrális úton vannak.

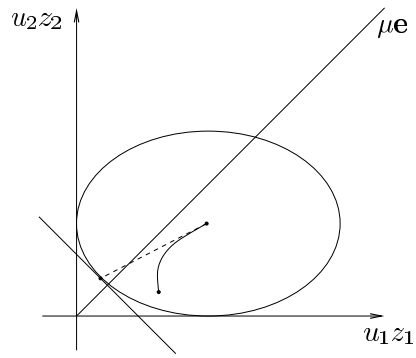
Nevezzünk egy α ***lépéshosszt megengedettnek,*** ha az új pontunkra is teljesül a pozitivitási megkötés, azaz $\mathbf{u} + \alpha\Delta\mathbf{u} > \mathbf{0}$, $\mathbf{z} + \alpha\Delta\mathbf{z} > \mathbf{0}$.

Mind az algoritmus elméleti elemzése során, mind pedig a gyakorlati számítógépes megvalósításakor két célunk van: az $\alpha > 0$ lépéshosszt úgy meghatározni, hogy (i) az $\mathbf{u} + \alpha\Delta\mathbf{u} > \mathbf{0}$ és $\mathbf{z} + \alpha\Delta\mathbf{z} > \mathbf{0}$, valamint (ii) a pontunk centralitási mértéke egy adott korlát alatt maradjon, azaz $\partial_c(\mathbf{u}) \leq \tau$ teljesüljön[8], valamely adott $\tau > 1$ szám esetén.

Legyen $\tau > 1$, $\tau \in \mathbb{R}$ és $\mathbf{u} \in \mathcal{F}$, $\partial_c(\mathbf{u}) \leq \tau$. Továbbá mivel $\mathbf{v} = \sqrt{\mathbf{u}\,\mathbf{z}/\mu}$, ekkor a

$$\partial_c(\mathbf{u}) = \frac{\max(\mathbf{v}^2)}{\min(\mathbf{v}^2)} \leq \tau \iff \max(\mathbf{v}^2) \leq \tau\min(\mathbf{v}^2)$$

[8] A $\partial_c(\mathbf{u}) \leq \tau$ egyenlőtlenség a centrális útnak egy ún. ***τ-környezetét*** jelöli ki.



24.2. ábra. A Dikin-algoritmus egy iterációja az átskálázott térben, ahol $u_1z_1 = \mu\, v_1^2$ és $u_2z_2 = \mu\, v_2^2$.

összefüggésből következik, hogy létezik $\tau_1, \tau_2 > 0$ és $\tau_2 = \tau\tau_1$, amelyekre

$$\tau_1\mathbf{e} \leq \mathbf{v}^2 \leq \tau_2\mathbf{e}\ .$$

Az eddigieket összefoglalva, az algoritmusunk a következő:
Bemenő adatként meg kell adnunk az $\varepsilon > 0$ megállási paramétert, a $\tau \geq 1$ környezeti paramétert, az $\alpha > 0$ lépés paramétert, illetve egy $\mathbf{u}^0$ belső pontot, ami a megadott τ környezeten belül helyezkedik el, azaz $\mathbf{u}^0 \in \mathcal{F}^0$, $z(\mathbf{u}^0) > \mathbf{0}$ és $\partial_c(\mathbf{u}^0) \leq \tau$.

DIKIN-FÉLE-AFFIN-SKÁLÁZÁS

$\mathbf{u} = \mathbf{u}^0$
$\mathbf{z} = z(\mathbf{u^0})$
while $\mathbf{u}^T\mathbf{z} \geq \varepsilon$
    $\Delta\mathbf{u} = -(S + XM)^{-1}\mathbf{u}^2\mathbf{z}^2 / \|\mathbf{u}\,\mathbf{z}\|$
    számítsuk ki az $\alpha > 0$ megengedett lépéshosszt
    $\mathbf{u} \leftarrow \mathbf{u} + \alpha\Delta\mathbf{u}$
    $\mathbf{z} \leftarrow z(\mathbf{u})$

Az algoritmus egy lépését a 24.2. ábra szemlélteti.

A DIKIN-FÉLE-AFFIN-SKÁLÁZÁS fenti változata kétféleképpen is elemezhető: gyakorlati, illetve elméleti szempontból. A folytonos optimalizálásban gyakran megfigyelt jelenség, hogy az iteratív algoritmusok gyakorlatban megfigyelt lépésszáma messze felülmúlja az elméletileg bizonyítható lépésszámot főleg a hosszú lépéses és nagy környezetes belsőpontos algoritmusok

esetén. Az elméleti hatékonyság esetén az ún. *legrosszabb eset elemzése* a cél. Ezzel szemben a gyakorlatban az a cél, hogy minél jobban kihasználjuk a megoldandó feladat összes tulajdonságát, még akkor is, ha az általános elméleti elemzések során nem minden apró részletet tudunk figyelembe venni.

24.2.1. Dikin-algoritmus: gyakorlati szempontok

A DIKIN-FÉLE-AFFIN-SKÁLÁZÁS számítógépes megvalósítása során két paraméternek van jelentős hatása: a környezeti paraméter, $\tau \geq 1$ megválasztásának, illetve a τ rögzítése után, a lehető legnagyobb α megengedett lépéshossz meghatározásának.

A gyakorlati alkalmazások során akár $\tau = 10\,000$ is lehet. Ekkor például $\tau_1 = 1/100$ és $\tau_2 = 100$ is lehetséges.[9]

A megengedett lépéshossz, $\alpha > 0$ megválasztásánál, mohó módon járunk el, azaz minden iterációban, miután meghatároztuk a csökkenési irányt, kiszámoljuk a maximális megengedett $\alpha > 0$ lépéshosszt.[10] Először azt szeretnénk elérni, hogy az új megoldás vektorok pozitívak legyenek, azaz $\mathbf{u} + \alpha\Delta\mathbf{u} \geq \mathbf{0}$ és $\mathbf{z} + \alpha\Delta\mathbf{z} \geq \mathbf{0}$, ami

$$0 < \alpha < \bar{\alpha} = \min\left\{ \min_{i\in\mathcal{I}}\left\{ -\frac{u_i}{\Delta u_i} : \Delta u_i < 0 \right\}, \min_{i\in\mathcal{I}}\left\{ -\frac{z_i}{\Delta z_i} : \Delta z_i < 0 \right\}, 1 \right\} \tag{24.11}$$

esetén biztosan teljesülni fog, ahol $\mathcal{I} = \{1, 2, \ldots, n\}$. (Ezen állítás helyességét a 24.2-6 gyakorlat eredménye garantálja.) Másfelől fontos, hogy a centralitási mértékünk a megadott érték alatt maradjon az új megoldások esetén is. Mivel az

$$\mathbf{u}^+ = \mathbf{u}+\alpha\Delta\mathbf{u} = \mathbf{d}(\mathbf{d}^{-1}\mathbf{u})+\alpha\mathbf{d}(\mathbf{d}^{-1}\Delta\mathbf{u}) = \mathbf{d}\sqrt{\mu}\,\mathbf{v}+\alpha\mathbf{d}\sqrt{\mu}\,\mathbf{p}_u = \sqrt{\mu}\,\mathbf{d}(\mathbf{v}+\alpha\mathbf{p}_u)\ ,$$

és

$$z(\mathbf{u}^+) = \mathbf{z}^+ = \mathbf{z} + \alpha\Delta\mathbf{z} = \sqrt{\mu}\,\mathbf{d}^{-1}(\mathbf{v} + \alpha\mathbf{p}_z),$$

ezért

$$\begin{aligned}(\mathbf{u}^+)z(\mathbf{u}^+) &= (\mathbf{u} + \alpha\Delta\mathbf{u})\,(\mathbf{z} + \alpha\Delta\mathbf{z}) = \mu\,(\mathbf{v} + \alpha\mathbf{p}_u)(\mathbf{v} + \alpha\mathbf{p}_z) \\ &= \mu\,(\mathbf{v}^2 + \alpha(\mathbf{p}_u + \mathbf{p}_z)\mathbf{v} + \alpha^2\mathbf{p}_u\mathbf{p}_z)\ .\end{aligned}$$

[9] Később az elméleti elemzésekből szembetűnő lesz, hogy elméletileg minél nagyobb a τ értéke, annál rosszabb lesz az iteráció szám. Ezt a gyakorlat nem igazolta, de annak érdekében, hogy elméletileg elemezhető legyen az algoritmus, és hogy a polinomiális lépésszámot az algoritmus egy változatára általánosan bizonyítani tudjuk, a legrosszabb eset elemzést kell használnunk. A legrosszabb esettől a gyakorlati problémák sokszor eltérnek, de nem mindig.

[10] Ez az elképzelés is jelentősen eltér az elméletileg bizonyítható helyzetektől, amikor is közvetlenül τ függvényében adjuk meg a korlátokat a megengedett lépéshosszra, nem törődve a lokálisan legjobb, meghatározható érték kiszámításával. Cserébe elméletileg is elemezhető algoritmus változatot kapunk, amelynek a lépésszáma polinomiális.

A

$$\mathbf{p} = \mathbf{p}_u + \mathbf{p}_z = -\frac{\mathbf{v}^3}{\|\mathbf{v}^2\|}$$

összefüggést figyelembe véve

$$\mathbf{u}^+ z(\mathbf{u}^+) = \mu \left(\mathbf{v}^2 - \alpha \frac{\mathbf{v}^4}{\|\mathbf{v}^2\|} + \alpha^2 \mathbf{p}_u \mathbf{p}_z \right) \tag{24.12}$$

adódik. A $\mu > 0$ és az $\bar{\alpha}$ korlát miatt az $\mathbf{u}^+ z(\mathbf{u}^+) > 0$ teljesül, de mi még azt is szeretnénk, hogy

$$\tau_1 \, \mathbf{e} \leq \mathbf{v}^2 - \alpha \frac{\mathbf{v}^4}{\|\mathbf{v}^2\|} + \alpha^2 \mathbf{p}_u \mathbf{p}_z \leq \tau_2 \, \mathbf{e}$$

is igaz legyen, azaz bármely i index esetén

$$\tau_1 \leq u_i^2 - \alpha \frac{u_i^4}{\|\mathbf{v}^2\|} + \alpha^2 (\mathbf{p}_u \mathbf{p}_z)_i \leq \tau_2 \; . \tag{24.13}$$

A bal oldali, az α ismeretlenre nézve kvadratikus, egyenlőtlenségből az alábbi korlátokat kapjuk:

$$\alpha_1 = \min_{i \in \mathcal{I}} \left\{ \frac{\frac{u_i^4}{\|\mathbf{v}^2\|} + \sqrt{\vartheta_1}}{2 \left(\mathbf{p}_u \mathbf{p}_z \right)_i} : \vartheta_1 \geq 0, \, (\mathbf{p}_u \mathbf{p}_z)_i > 0 \right\} \quad \text{és}$$

$$\alpha_2 = \min_{i \in \mathcal{I}} \left\{ \frac{\frac{u_i^4}{\|\mathbf{v}^2\|} - \sqrt{\vartheta_1}}{2 \left(\mathbf{p}_u \mathbf{p}_z \right)_i} : \vartheta_1 \geq 0, \, (\mathbf{p}_u \mathbf{p}_z)_i < 0 \right\} \; ,$$

ahol $\vartheta_1 = u_i^8 / \left\|\mathbf{v}^2\right\|^2 - 4 \left(\mathbf{p}_u \mathbf{p}_z \right)_i \left(u_i^2 - \tau_1 \right)$. Hasonlóan a jobb oldali egyenlőtlenségből a következő korlátok adódnak α értékére

$$\alpha_3 = \min_{i \in \mathcal{I}} \left\{ \frac{-\frac{u_i^4}{\|\mathbf{v}^2\|} - \sqrt{\vartheta_2}}{-2 \left(\mathbf{p}_u \mathbf{p}_z \right)_i} : \vartheta_2 \geq 0, \, (\mathbf{p}_u \mathbf{p}_z)_i > 0 \right\} \quad \text{és}$$

$$\alpha_4 = \min_{i \in \mathcal{I}} \left\{ \frac{-\frac{u_i^4}{\|\mathbf{v}^2\|} + \sqrt{\vartheta_2}}{-2 \left(\mathbf{p}_u \mathbf{p}_z \right)_i} : \vartheta_2 \geq 0, \, (\mathbf{p}_u \mathbf{p}_z)_i < 0 \right\} \; ,$$

ahol $\vartheta_2 = u_i^8 / \left\|\mathbf{v}^2\right\|^2 - 4 \left(\mathbf{p}_u \mathbf{p}_z \right)_i \left(u_i^2 - \tau_2 \right)$.

24.2.2. Dikin-algoritmus: elméleti elemzés

Az iterációk során csak egy tompított[11] Dikin-lépést teszünk meg. A lépéshossz megfelelő megválasztásával biztosítjuk az új pontunk pozitivitását,

[11] Legyen az új pontunk $(\mathbf{u} + \alpha \Delta \mathbf{u}, \mathbf{z} + \alpha \Delta \mathbf{z})$, ha $0 < \alpha < 1$, akkor α paraméterrel tompított lépésről, míg $\alpha = 1$ esetén teljes lépésről beszélünk.

így a megengedettségét, valamint a pontunkat nem engedjük ki a centrális út τ környezetéből. Ezen elvárásoknak megfelelő α l épéshosszra ad elégséges feltételt a következő lemma, amelynek bizonyítását az Olvasóra bízzuk (lásd 24.2-9 gyakorlat).

24.24. lemma. *Legyen $\tau \in \mathbb{R}$, $\tau > 1$. Tegyük fel, hogy $\mathbf{u} > \mathbf{0}$, $\mathbf{z} = z(\mathbf{u}) > \mathbf{0}$, $\mathbf{u} \in \mathcal{F}$ és $\partial_c(\mathbf{u}) \le \tau$. Ekkor bármely α lépéshossz, amely kielégíti az*

$$\alpha \le \frac{\|\mathbf{v}^2\|}{2\tau_2} \quad \text{és} \quad \alpha < \frac{4\tau_1}{\|\mathbf{v}^2\|}$$

egyenlőtlenségeket, megengedett lépéshossz, és az $\mathbf{u}^+ = \mathbf{u} + \alpha\Delta\mathbf{u}$ vektorra $\partial_c(\mathbf{u}^+) \le \tau$.

Tekintettel arra, hogy az előző lemma számunkra egy olyan lépéshosszt biztosít, amelyik esetén az új megoldásunk megengedett lesz, és rá a megfelelő centralitási elvárás is teljesül, rátérhetünk annak a vizsgálatára, hogy ez mekkora dualitásrés csökkenést fog eredményezni.

24.25. lemma. *Ha az α lépéshossz megengedett, akkor*

$$(\mathbf{u}^+)^T\mathbf{z}^+ \le \left(1 - \frac{\alpha}{\sqrt{n}}\right)\mathbf{u}^T\mathbf{z} \,.$$

Bizonyítás. Induljunk ki a (24.12) egyenlőségből. Mivel az α megengedett lépéshossz és $\mu > 0$, ezért

$$\mathbf{v}^2 - \alpha\frac{\mathbf{v}^4}{\|\mathbf{v}^2\|} + \alpha^2\mathbf{p}_u\mathbf{p}_z > \mathbf{0} \,.$$

Az M mátrix ferdén szimmetrikus, ezért $(\Delta\mathbf{u})^T\Delta\mathbf{z} = 0$, azaz

$$\mathbf{p}_u^T\mathbf{p}_z = \mathbf{e}^T(\mathbf{d}^{-1}\Delta\mathbf{u}\,\mathbf{d}\,\Delta\mathbf{z})\frac{1}{\mu} = \frac{1}{\mu}(\Delta\mathbf{u})^T\Delta\mathbf{z} = 0 \,. \tag{24.14}$$

Számítsuk ki az új dualitásrés és a μ paraméter hányadosát:

$$\frac{(\mathbf{u}^+)^T\mathbf{z}^+}{\mu} = \mathbf{e}^T\mathbf{v}^2 - \alpha\frac{\mathbf{e}^T\mathbf{v}^4}{\|\mathbf{v}^2\|} + \alpha^2\mathbf{p}_u^T\mathbf{p}_z = \|\mathbf{v}\|^2 - \alpha\|\mathbf{v}^2\| \,,$$

ahol figyelembe vettük a $\mathbf{p}_u$ és $\mathbf{p}_z$ vektorok ortogonalitását és az $\mathbf{e}^T\mathbf{v}^4 = (\mathbf{v}^2)^T\mathbf{v}^2 = \|\mathbf{v}^2\|^2$ összefüggést. A Cauchy-Schwarz egyenlőtlenség alapján

$$\|\mathbf{v}^2\| = \frac{1}{\sqrt{n}}\|\mathbf{e}\|\,\|\mathbf{v}^2\| \ge \frac{\mathbf{e}^T\mathbf{v}^2}{\sqrt{n}} = \frac{\|\mathbf{v}\|^2}{\sqrt{n}} \,,$$

behelyettesítve

$$(\mathbf{u}^+)^T\mathbf{z}^+ \leq \mu\left(1-\frac{\alpha}{\sqrt{n}}\right)\|\mathbf{v}\|^2 = \left(1-\frac{\alpha}{\sqrt{n}}\right)\mathbf{u}^T\mathbf{z}$$

adódik. ∎

A 24.24. lemma szerint az $\alpha = 1/(\tau\sqrt{n})$ választás[12] megfelelő, azaz ebben az esetben az algoritmus jól definiált. Az alábbi tételben megmutatjuk, hogy ezen érték mellett az algoritmus polinomiális.

24.26. tétel. *Legyen $\tau = \max(2, \partial_c(\mathbf{u}^0))$ és $\alpha = 1/(\tau\sqrt{n})$. Ha $n \geq 2$, akkor a Dikin-féle affin skálázású algoritmus a ferdén szimmetrikus lineáris programozási feladatot legfeljebb*

$$\left\lceil \tau n \lg \frac{\mathbf{q}^T\mathbf{u}^0}{\varepsilon} \right\rceil$$

lépésben oldja meg. A megoldás $\mathbf{u} \in \mathcal{F}$, amelyre $\partial_c(\mathbf{u}) \leq \tau$ és $\mathbf{q}^T\mathbf{u} \leq \varepsilon$.

Bizonyítás. Az $\mathbf{u}^0 \in \mathcal{F}^0$ induló megoldásra $\partial_c(\mathbf{u}^0) \leq \tau$ összefüggés teljesült. A 24.24. lemma alapján a lépéshossz megválasztható úgy, hogy minden egyes lépésben az új pont pozitív, megengedett megoldás legyen és a centralitási mértéke τ értékénél kisebb maradjon. Azt kell megmutatni, hogy a tétel feltételeiben megadott α érték kielégíti a 24.24. lemmában adott korlátokat. Mivel $n \geq 2$, ezért

$$\alpha = \frac{1}{\tau\sqrt{n}} = \frac{\tau_1}{\tau_2\sqrt{n}} \leq \frac{\tau_1\sqrt{n}}{2\tau_2} = \frac{\|\tau_1\mathbf{e}\|}{2\tau_2} \leq \frac{\|\mathbf{v}^2\|}{2\tau_2}\,,$$

ahol felhasználtuk azt is, hogy $\mathbf{0} \leq \tau_1\mathbf{e} \leq \mathbf{v}^2$. A $\|\mathbf{v}^2\| \leq \tau_2\sqrt{n}$ miatt

$$\frac{4\tau_1}{\|\mathbf{v}^2\|} \geq \frac{4\tau_1}{\tau_2\sqrt{n}} = \frac{4}{\tau\sqrt{n}} > \alpha$$

adódik, azaz a tétel feltételeiben megadott lépéshossz megengedett, mert teljesíti a 24.24. lemmában adott korlátokat.

Az induló megoldásnál a célfüggvény értéke $\mathbf{q}^T\mathbf{u}^0$ volt. (Könnyen megmutatható, hogy $\mathbf{q}^T\mathbf{u}^0 = (\mathbf{u}^0)^T z(\mathbf{u}^0)$, ami pedig a dualitásrés.) A 24.25. lemma alapján a dualitásrés minden lépésben $(1 - 1/(n\tau))$ szorzóval csökken. Így k lépés után a célfüggvény értéke ε alá csökken, ha

$$\left(1-\frac{1}{n\tau}\right)^k \mathbf{q}^T\mathbf{u}^0 \leq \varepsilon\,.$$

[12] Mivel az α paraméter függ az n számtól, azaz a feladat dimenziójától, rövid lépéses algoritmusról beszélünk.

Szeretnénk meghatározni a k értékét. Vegyük az előző kifejezés logaritmusát. Ekkor

$$k \lg \left(1 - \frac{1}{n\tau}\right) + \lg(\mathbf{q}^T \mathbf{u}^0) \leq \lg \varepsilon \, .$$

Átrendezés után

$$\lg \frac{\mathbf{q}^T \mathbf{u}^0}{\varepsilon} \leq -k \lg \left(1 - \frac{1}{n\tau}\right)$$

adódik. Mivel $-\lg(1-t) \geq t$, ezért a következő egyenlőtlenség a fentinél erősebb feltétel

$$\frac{k}{n\tau} \geq \lg \frac{\mathbf{q}^T \mathbf{u}^0}{\varepsilon} \, .$$

Ebből pedig már következik a tétel állítása. ■

24.2.3. Az optimális partíció meghatározása

Ahhoz, hogy a változók koordinátáira a centrális út mentén egy korlátot tudjunk adni, szükségünk lesz egy, az optimális megoldások halmazát jellemző mérőszámra. Az előzőek alapján az $\mathbf{u} \in \mathcal{F}$ vektorra pontosan akkor igaz, hogy $\mathbf{u} \in \mathcal{F}^*$, ha $\mathbf{u}_\mathcal{N} = \mathbf{0}$ és $\mathbf{z}_\mathcal{B} = \mathbf{0}$ teljesül[13], ahol $(\mathcal{B}, \mathcal{N})$ a feladat optimális partícióját jelöli (lásd a 24.18. definíciót). Tetszőleges $\mathbf{u} \in \mathcal{F}^*$ optimális megoldásra fennáll az

$$\mathbf{u}\, z(\mathbf{u}) = \mathbf{0} \qquad \text{és} \qquad \mathbf{u} + z(\mathbf{u}) \geq \mathbf{0}$$

összefüggés. Első lépésként az optimális megoldások nemnulla koordinátáinak nagyságára keresünk egy jellemző mennyiséget. Definiáljuk az (SP) feladat kondíciószámát a alábbi módon.

24.27. definíció. *Legyen*

$$\sigma^u_{SP} = \begin{cases} \min\limits_{i \in \mathcal{B}} \max\limits_{(\mathbf{u},\mathbf{z}) \in \mathcal{F}^*} \{u_i\}, & \text{ha } \mathcal{B} \neq \emptyset \, , \\ \infty & \text{különben} \, , \end{cases}$$

és

$$\sigma^z_{SP} = \begin{cases} \min\limits_{i \in \mathcal{N}} \max\limits_{(\mathbf{u},\mathbf{z}) \in \mathcal{F}^*} \{z(u)_i\}, & \text{ha } \mathcal{N} \neq \emptyset \, , \\ \infty & \text{különben} \, . \end{cases}$$

Az (SP) feladat ***kondíciószáma*** $\sigma_{SP} = \min\{\sigma^u_{SP}, \sigma^z_{SP}\}$.

Mivel a $\mathcal{B}$ és $\mathcal{N}$ halmazok egyszerre nem lehetnek üresek, az $\mathcal{F}^*$ halmaz pedig korlátos, σ^u_{SP} és σ^z_{SP} számok közül legalább az egyik véges. Továbbá $\sigma^u_{SP}, \sigma^z_{SP} > 0$, azaz a kondíciószám egy véges pozitív szám. Valamint mivel

[13] Jelölje $\mathbf{u}_\mathcal{N}$ az $\mathbf{u}$ vektor $\mathbf{u}_\mathcal{N}$ halmazbeli indexekből álló részvektorát. Hasonlóan definiáljuk a $\mathbf{z}_\mathcal{B}$ részvektort.

erősen komplementáris megoldás létezik és minden $(\mathbf{u},\mathbf{z}) \in \mathcal{F}^*$ komplementáris megoldás, következésképpen

$$\sigma_{SP} = \min_i \max_{(\mathbf{u},\mathbf{z})\in\mathcal{F}^*} \{u_i + z(u)_i\} \ .$$

Első célunk a σ_{SP} értékére egy alsó korlát meghatározása.

24.28. tétel. *Legyen adott az (SP) feladat, melyre teljesül a belső pont feltétel. Ha az M mátrix és a $\mathbf{q}$ vektor egészértékű, akkor az (SP) feladat kondíciószámára a következő alsó becslés teljesül:*

$$\sigma_{SP} \geq \frac{1}{\prod_{j=1}^{n} \|\mathbf{m}_j\|} \ ,$$

ahol $\mathbf{m}_j$ az M mátrix j-edik oszlopa.

Bizonyítás. Tegyük fel, hogy a $(\mathcal{B},\mathcal{N})$ partíciót ismerjük. Ekkor

$$\begin{pmatrix} \mathbf{z}_\mathcal{B} \\ \mathbf{z}_\mathcal{N} \end{pmatrix} = \begin{pmatrix} M_{\mathcal{B}\mathcal{B}} & M_{\mathcal{B}\mathcal{N}} \\ M_{\mathcal{N}\mathcal{B}} & M_{\mathcal{N}\mathcal{N}} \end{pmatrix} \begin{pmatrix} \mathbf{u}_\mathcal{B} \\ \mathbf{u}_\mathcal{N} \end{pmatrix} + \begin{pmatrix} \mathbf{q}_\mathcal{B} \\ \mathbf{q}_\mathcal{N} \end{pmatrix} \ .$$

Legyen $(\mathbf{u},\mathbf{z}) \in \mathcal{F}$ szigorúan komplementáris megoldás. A komplementaritás miatt az $(\mathbf{u},\mathbf{z})$ vektor optimális (24.2. következmény), azaz $\mathbf{q}^T\mathbf{u} = 0$. Továbbá felhasználva a $(\mathcal{B},\mathcal{N})$ partíció definícióját ($\mathbf{u}_\mathcal{N} = \mathbf{0}$) azt kapjuk, hogy $\mathbf{q}_\mathcal{B}^T\mathbf{u}_\mathcal{B} = 0$. Figyelembe véve a szigorú komplementaritást $\mathbf{u}_\mathcal{B} > \mathbf{0}$, így szükségszerűen $\mathbf{q}_\mathcal{B} = \mathbf{0}$. Ezeket felhasználva a fenti rendszerünk a következő alakú:

$$\begin{pmatrix} \mathbf{0} \\ \mathbf{z}_\mathcal{N} \end{pmatrix} = \begin{pmatrix} M_{\mathcal{B}\mathcal{B}} & M_{\mathcal{B}\mathcal{N}} \\ M_{\mathcal{N}\mathcal{B}} & M_{\mathcal{N}\mathcal{N}} \end{pmatrix} \begin{pmatrix} \mathbf{u}_\mathcal{B} \\ \mathbf{0} \end{pmatrix} + \begin{pmatrix} \mathbf{0} \\ \mathbf{q}_\mathcal{N} \end{pmatrix} \ ,$$

így

$$\begin{pmatrix} \mathbf{0} \\ \mathbf{z}_\mathcal{N} \end{pmatrix} = \begin{pmatrix} M_{\mathcal{B}\mathcal{B}}\mathbf{u}_\mathcal{B} \\ M_{\mathcal{N}\mathcal{B}}\mathbf{u}_\mathcal{B} + \mathbf{q}_\mathcal{N} \end{pmatrix} .$$

Tehát

$$\begin{pmatrix} M_{\mathcal{B}\mathcal{B}} & 0_{\mathcal{B}\mathcal{N}} \\ M_{\mathcal{N}\mathcal{B}} & -I_{\mathcal{N}\mathcal{N}} \end{pmatrix} \begin{pmatrix} \mathbf{u}_\mathcal{B} \\ \mathbf{z}_\mathcal{N} \end{pmatrix} = \begin{pmatrix} \mathbf{0}_\mathcal{B} \\ -\mathbf{q}_\mathcal{N} \end{pmatrix}, \qquad \mathbf{u}_\mathcal{B} \geq \mathbf{0},\ \mathbf{u}_\mathcal{N} = \mathbf{0},\ \mathbf{z}_\mathcal{B} = \mathbf{0},\ \mathbf{z}_\mathcal{N} \geq \mathbf{0} \ .$$

Az utóbbi rendszer éppen az $\mathcal{F}^*$ leírása, ami egy nem üres, kompakt halmaz és szigorúan komplementáris megoldást is tartalmaz, így az

$$R = \begin{pmatrix} M_{\mathcal{B}\mathcal{B}} & 0_{\mathcal{B}\mathcal{N}} \\ M_{\mathcal{N}\mathcal{B}} & -I_{\mathcal{N}\mathcal{N}} \end{pmatrix}, \quad \mathbf{y} = \begin{pmatrix} \mathbf{u}_\mathcal{B} \\ \mathbf{z}_\mathcal{N} \end{pmatrix}, \quad \mathbf{r} = \begin{pmatrix} \mathbf{0}_\mathcal{B} \\ -\mathbf{q}_\mathcal{N} \end{pmatrix}$$

választással a 24.2-11 gyakorlat eredményét alkalmazhatjuk, és így

$$\max_{\mathbf{u}} u_i \geq \frac{1}{\prod_{j\in\mathcal{B}} \|\mathbf{m}_j\|}, \quad \forall i \in \mathcal{B} \qquad \text{és} \qquad \max_{\mathbf{z}} z_i \geq \frac{1}{\prod_{j\in\mathcal{B}} \|\mathbf{m}_j\|}, \quad \forall i \in \mathcal{N}$$

adódik. Ezekből pedig a tétel állítását, a

$$\sigma_{SP} \geq \frac{1}{\prod_{j\in\mathcal{B}} \|\mathbf{m}_j\|} \geq \frac{1}{\prod_{j=1}^{n} \|\mathbf{m}_j\|}$$

egyenlőtlenséget nyerjük. ■

Természetesen az előző tétel akkor is igaz marad, ha az M mátrix és a $\mathbf{q}$ vektor adatai racionálisak. Ebben az esetben a nevezők legkisebb közös többszörösével kell megszoroznunk a mátrix és a vektor elemeit. Ezután az előző tétel alkalmazásával a kondíciószámra egy alsó korlátot tudunk kiszámolni.

A változók mérete a centrális út mentén

A $\sigma = \sigma_{SP}$ kondíciószám segítségével alsó és felső korlátokat adhatunk a változókra a centrális út mentén. Legyen $(\mathcal{B},\mathcal{N})$ az (SP) feladat optimális partíciója.

24.29. lemma. *Minden $(\mathbf{u}(\mu),\mathbf{z}(\mu)) \in \mathcal{C}$, $\mu > 0$ esetén az alábbi egyenlőtlenségek igazak:*

$$u_i(\mu) \geq \frac{\sigma}{n} \quad i \in \mathcal{B}\,, \qquad u_i(\mu) \leq \frac{n\mu}{\sigma} \quad i \in \mathcal{N}\,,$$
$$z_i(\mu) \leq \frac{n\mu}{\sigma} \quad i \in \mathcal{B}\,, \qquad z_i(\mu) \geq \frac{\sigma}{n} \quad i \in \mathcal{N}\,.$$

A lemma, amelynek a bizonyítását az Olvasóra bízzuk (lásd 24.2-12 gyakorlat), alapján természetesen adódnak a következő elnevezések:

24.30. definíció. *Legyen $(\mathbf{u}(\mu),\mathbf{z}(\mu)) \in \mathcal{C}$, $\mu > 0$. Ekkor az $u_i(\mu)$ koordinátát **nagynak** mondjuk, ha $i \in \mathcal{B}$, illetve a $z_i(\mu)$ koordinátát **nagynak** mondjuk, ha $i \in \mathcal{N}$. Ellenkező esetben a koordinátákat **kicsinek** nevezzük.*

Az elnevezés is mutatja, hogy az $\mathbf{u}$ vektor nagy koordinátáinak indexei alkotják a $\mathcal{B}$ halmazt, míg a $\mathbf{z}$ nagy koordinátáinak indexei az $\mathcal{N}$ halmazt. A lemma alapján ha a μ paraméter elég kicsi, akkor egyértelműen eldönthető, hogy az $\mathbf{u}(\mu)$ és a $\mathbf{z}(\mu)$ vektorok mely koordinátái nagyok és melyek kicsik, azaz a $(\mathcal{B},\mathcal{N})$ partíció egyértelműen meghatározható.

24.31. következmény. *Ha a $\mu < \sigma^2/n^2$ centrális út paraméterhez ismert az $(\mathbf{u}(\mu),\mathbf{z}(\mu)) \in \mathcal{C}$ megoldás, akkor meghatározhatjuk a $(\mathcal{B},\mathcal{N})$ optimális partíciót.*

Bizonyítás. A megadott μ érték mellett $n\mu/\sigma < \sigma/n$ teljesül, így az előző lemma alapján a

$$\mathcal{B} = \left\{ i \ : \ u_i(\mu) \geq \frac{\sigma}{n} \right\} , \qquad \mathcal{N} = \left\{ i \ : \ z_i(\mu) \geq \frac{\sigma}{n} \right\}$$

szétosztás egyértelmű és megfelelő. ■

Tekintettel arra, hogy a belsőpontos módszerek, az iterációk során, a legritkábban állítanak elő a centrális úton lévő pontot, ezért az előző eredmények megfelelőit a centrális út egy környezetében is bizonyítanunk kell.

A változók mérete a centrális út egy környezetében

Legyen adott $\mathbf{u}^0 \in \mathcal{F}^0$, $\mathbf{z}^0 > 0$, melyek a $\mu^0 = 1$ paraméterhez tartoznak, azaz $(\mathbf{u}^0)^T\mathbf{z}^0 = \mathbf{q}^T\mathbf{u}^0 = n\mu^0 = n$. Legyen $\tau = 2$, ekkor a 24.26. tétel alapján a Dikin-lépéses algoritmus $\lceil 2n \lg \frac{n}{\varepsilon} \rceil$ iterációban előállít egy $\mathbf{u} \in \mathcal{F}^0$ pontot, melyre $\partial_c(\mathbf{u}) \leq 2$ és $\mathbf{q}^T\mathbf{u} \leq \varepsilon$.

Fontos lenne a centrális út környezetében tudni a változók nagyságrendjét. Erre szolgál a következő lemma, amelynek bizonyítását az Olvasóra bízzuk (lásd 24.2-13 gyakorlat).

24.32. lemma. *Legyen $\mathbf{u} \in \mathcal{F}^0$, $\mathbf{z} > 0$, melyekre $\partial_c(\mathbf{u}) \leq \tau$. Ekkor*

$$u_i \geq \frac{\sigma}{\tau n} \quad i \in \mathcal{B} , \qquad u_i \leq \frac{\mathbf{u}^T\mathbf{z}}{\sigma} \quad i \in \mathcal{N} ,$$
$$z_i \leq \frac{\mathbf{u}^T\mathbf{z}}{\sigma} \quad i \in \mathcal{B} , \qquad z_i \geq \frac{\sigma}{\tau n} \quad i \in \mathcal{N} ,$$

ahol σ a feladat kondíciószáma.

Megfelelő feltétel mellett a 24.32. lemma segítségével az $\mathbf{u}$, illetve az $\mathbf{z}$ vektor koordinátáit nagyság szerint szétválogatva meghatározhatjuk a $(\mathcal{B},\mathcal{N})$ partíciót.

24.33. lemma. *Legyen $\mathbf{u} \in \mathcal{F}^0$, $\mathbf{z} > 0$, melyekre $\partial_c(\mathbf{u}) \leq \tau$. Ha $\mathbf{u}^T\mathbf{z} \leq \sigma^2/(\tau n)$, akkor az (SP) feladat $(\mathcal{B},\mathcal{N})$ partíciója a következő módon kapható meg:*

$$\mathcal{B} = \{ i \ : \ u_i > z_i \} \qquad \text{és} \qquad \mathcal{N} = \{ i \ : \ u_i < z_i \} .$$

Bizonyítás. A kicsi és nagy változókat szét tudjuk választani, ha $\mathbf{u}^T\mathbf{z}/\sigma < \sigma/(\tau n)$, azaz $\mathbf{u}^T\mathbf{z} < \sigma^2/(\tau n)$. Tehát az előző lemma szerint a fent megadott $\mathcal{B}$ és $\mathcal{N}$ halmazok jól definiáltak, és az indexhalmaz optimális partícióját adják. ■

A 24.33. lemma feltételeit kielégítő $(\mathbf{u},\mathbf{z})$ pontpár a 24.26. tétel szerint $\lceil 2n \lg \frac{2n^2}{\sigma^2} \rceil$ lépésben a Dikin-algoritmussal előállítható, ennek tükrében az alábbi következmény könnyen adódik.

24.34. következmény. *Legyen* $\mathbf{u}^0 \in \mathcal{F}^0$, $\mathbf{z}^0 > 0$, *melyekre* $\partial_c(\mathbf{u}^0) \leq \tau$ *és* $(\mathbf{u}^0)^T\mathbf{z}^0 = n$, $\tau = 2$. *Ekkor a Dikin-lépéses algoritmus* $\lceil 2n \lg \frac{2n^2}{\sigma^2} \rceil$ *lépésben egy olyan* $\mathbf{u} \in \mathcal{F}^0$ *megoldást ad, amely esetén a* $(\mathcal{B},\mathcal{N})$ *partíció meghatározható.*

Mint már az első részben említettük az $\mathbf{u} = \mathbf{0}$ mindig optimális megoldása az (SP) feladatnak. A következőkben megmutatjuk, hogy ez akkor és csak akkor az egyetlen optimális vektor, ha a $\mathcal{B}$ indexhalmaz üres. (Ezzel a speciális, triviális esettel a továbbiakban nem foglalkozunk.)

24.35. tétel. *Legyen* $\mathcal{F}^0 \neq \emptyset$. *Ekkor* $\mathbf{u} = \mathbf{0}$ *az (SP) feladat egyetlen optimális megoldása akkor és csak akkor, ha* $\mathbf{q} > \mathbf{0}$. *Ez az eset pontosan akkor fordul elő, ha* $\mathcal{B} = \emptyset$.

Bizonyítás. Ha $\mathcal{B} = \emptyset$, akkor $\mathbf{u} = \mathbf{0}$ optimális, sőt szigorúan komplementáris is $\mathcal{F}^0 \neq \emptyset$ mellett, azaz $z(\mathbf{u}) = \mathbf{q} > \mathbf{0}$ kell, hogy teljesüljön.
Másfelől $z(\mathbf{0}) = \mathbf{q}$ és $z(\mathbf{0}) > \mathbf{0}$ miatt $\mathbf{q} > \mathbf{0}$, így $\mathbf{u} = \mathbf{0}$ egyértelmű optimum. ■

Vizsgáljuk meg az $\mathbf{u} \in \mathcal{F}^0$, $z(\mathbf{u}) = \mathbf{z} > \mathbf{0}$ ε-optimális megoldást. Ha az ε kellően kicsi, akkor a $(\mathcal{B},\mathcal{N})$ partíció ismert, így a rendszerünk a következő alakú:

$$\begin{pmatrix} \mathbf{z}_\mathcal{B} \\ \mathbf{z}_\mathcal{N} \end{pmatrix} = \begin{pmatrix} M_{\mathcal{B}\mathcal{B}} & M_{\mathcal{B}\mathcal{N}} \\ M_{\mathcal{N}\mathcal{B}} & M_{\mathcal{N}\mathcal{N}} \end{pmatrix} \begin{pmatrix} \mathbf{u}_\mathcal{B} \\ \mathbf{u}_\mathcal{N} \end{pmatrix} + \begin{pmatrix} \mathbf{q}_\mathcal{B} \\ \mathbf{q}_\mathcal{N} \end{pmatrix} .$$

Az előző tétel szerint a $\mathcal{B} = \emptyset$ eset triviális, ezért tegyük fel, hogy $\mathcal{B} \neq \emptyset$, ekkor az optimalitás miatt $\mathbf{q}_\mathcal{B} = \mathbf{0}$.

$$\mathbf{z}_\mathcal{B} = M_{\mathcal{B}\mathcal{B}}\,\mathbf{u}_\mathcal{B} + M_{\mathcal{B}\mathcal{N}}\,\mathbf{u}_\mathcal{N} + \mathbf{q}_\mathcal{B} , \quad \text{tehát}$$

$$M_{\mathcal{B}\mathcal{B}}\,\mathbf{u}_\mathcal{B} = \mathbf{z}_\mathcal{B} - M_{\mathcal{B}\mathcal{N}}\,\mathbf{u}_\mathcal{N} . \tag{24.15}$$

Ha $\tilde{\mathbf{u}} \in \mathcal{F}^*$,
tilde$\mathbf{z} = z(\tilde{\mathbf{u}})$, akkor$\tilde{\mathbf{u}}_\mathcal{N} = \mathbf{0}$,$\tilde{\mathbf{z}}_\mathcal{B} = \mathbf{0}$, és a szigorú komplementaritás miatt $\tilde{\mathbf{u}}_\mathcal{B} > \mathbf{0}$, valamint a (24.15) egyenlőség miatt $M_{\mathcal{B}\mathcal{B}}\,\tilde{\mathbf{u}}_\mathcal{B} = \mathbf{0}$, azaz az $M_{\mathcal{B}\mathcal{B}}$ mátrix szinguláris. Ennek következtében az

$$M_{\mathcal{B}\mathcal{B}}\,\xi = \mathbf{z}_\mathcal{B} - M_{\mathcal{B}\mathcal{N}}\,\mathbf{u}_\mathcal{N} \tag{24.16}$$

egyenlet megoldása nem egyértelmű, egy lehetséges megoldása a $\xi = \mathbf{u}_\mathcal{B}$. Legyen ξ tetszőleges megoldása a(24.16) egyenletnek, ekkor definiálhatjuk az $\overline{\mathbf{u}}_\mathcal{B} = \mathbf{u}_\mathcal{B} - \xi$, $\overline{\mathbf{u}}_\mathcal{N} = \mathbf{0}$ megoldást és a $\overline{\mathbf{z}} = z(\overline{\mathbf{u}})$ eltérésváltozót, ahol $\overline{\mathbf{z}}_\mathcal{B} = M_{\mathcal{B}\mathcal{B}}\,\overline{\mathbf{u}}_\mathcal{B} + M_{\mathcal{B}\mathcal{N}}\,\overline{\mathbf{u}}_\mathcal{N} + \mathbf{q}_\mathcal{B} = M_{\mathcal{B}\mathcal{B}}\,(\mathbf{u}_\mathcal{B} - \xi) = \mathbf{0}$. Tehát az $(\overline{\mathbf{u}},\overline{\mathbf{z}})$ komplementáris megoldás, de $\mathbf{u}$ és $\mathbf{u}$ nem feltétlenül pozitív vektorok. Az $(\overline{\mathbf{u}},\overline{\mathbf{z}})$ pozitivitásához szükséges:

$$\overline{\mathbf{u}}_\mathcal{B} = \mathbf{u}_\mathcal{B} - \xi > \mathbf{0} , \quad \text{illetve}$$

$$\begin{aligned}\overline{\mathbf{z}}_{\mathcal{N}} &= M_{\mathcal{N}\mathcal{B}}\,\overline{\mathbf{u}}_{\mathcal{B}} + M_{\mathcal{N}\mathcal{N}}\,\overline{\mathbf{u}}_{\mathcal{N}} + \mathbf{q}_{\mathcal{N}} = M_{\mathcal{N}\mathcal{B}}\,\mathbf{u}_{\mathcal{B}} - M_{\mathcal{N}\mathcal{B}}\,\xi + \mathbf{q}_{\mathcal{N}} \\ &= \mathbf{z}_{\mathcal{N}} - M_{\mathcal{N}\mathcal{N}}\,\mathbf{u}_{\mathcal{N}} - M_{\mathcal{N}\mathcal{B}}\,\xi > \mathbf{0}\ .\end{aligned}$$

Tehát ε megfelelő értéke mellett elő tudjuk állítani az $(\overline{\mathbf{u}}, \overline{\mathbf{z}})$ vektort, ami az előző feltételek teljesülése mellett szigorúan komplementáris megoldás lesz. A következő részben ezen megoldás előállításával foglalkozunk.

Szigorúan komplementáris megoldás előállítása

Vezessük be a következő jelöléseket:

$$w = \|M\|_\infty\ , \qquad \mathcal{B}^* = \{i \in \mathcal{B}\ :\ M_{\mathcal{B}i} \text{ oszlopvektor nem azonosannulla}\}.$$

Definiáljuk a $\pi_{\mathcal{B}} \in \mathbb{N}$ számot a következő módon

$$\pi_{\mathcal{B}} = \begin{cases} 1, & \text{ha } \mathcal{B}^* = \emptyset\ , \\ \prod\limits_{j \in \mathcal{B}^*} \|M_{\mathcal{B}j}\| & \text{különben}\ . \end{cases}$$

Célunk a belsőpontos algoritmussal meghatározott, kellően pontos közelítő megoldásból az (SP) feladat egy szigorúan komplementáris megoldásának előállítása. A következő algoritmus bemeneti adataként adjunk meg egy $\mathbf{u} \in \mathcal{F}^0$, $\mathbf{z} > 0$ pontpárt, melyre $\partial_c(\mathbf{u}) \le \tau = 2$ és $\mathbf{u}^T\mathbf{z} \le \sigma^2 / \left(6nw^2\sqrt{|\mathcal{B}|}\pi_{\mathcal{B}}\right)$ teljesül.

SZIGORÚAN-KOMPLEMENTÁRIS-MEGOLDÁS

1 határozzuk meg a $(\mathcal{B}, \mathcal{N})$ partíciót az $(\mathbf{u}, \mathbf{z})$ pontpárból
2 oldjuk meg Gauss-eliminációval az
$\quad M_{\mathcal{B}\mathcal{B}}\,\xi = \mathbf{z}_{\mathcal{B}} - M_{\mathcal{B}\mathcal{N}}\,\mathbf{u}_{\mathcal{N}}$ egyenletrendszert // Kerekítési eljárás.
3 $\bar{\mathbf{u}}_{\mathcal{B}} = \mathbf{u}_{\mathcal{B}} - \xi$
4 $\bar{\mathbf{u}}_{\mathcal{N}} = \mathbf{0}$
5 $\bar{\mathbf{u}} = (\bar{\mathbf{u}}_{\mathcal{B}}, \bar{\mathbf{u}}_{\mathcal{N}})$
6 $\bar{\mathbf{z}} = z(\bar{\mathbf{u}})$

A fenti eljárással kapott $(\bar{\mathbf{u}}, \bar{\mathbf{z}})$ pontpár szigorúan komplementáris megoldása a feladatnak, mint ahogy ezt a következő lemmában ki is mondjuk.

24.36. lemma. *Legyen adott egy egész együtthatós (SP) feladat. Legyen $\mathbf{u} \in \mathcal{F}^0$, $\mathbf{z} > 0$, melyekre $\partial_c(\mathbf{u}) \le \tau = 2$. Ha $\mathbf{u}^T\mathbf{z} \le \varepsilon$, ahol $\varepsilon = \sigma^2 / \left(6nw^2\sqrt{|\mathcal{B}|}\pi_{\mathcal{B}}\right)$, akkor a kerekítési eljárás $O(|\mathcal{B}^*|^3)$ aritmetikai művelettel előállítja az (SP) feladat egy szigorúan komplementáris megoldását.*

A 24.36. lemma, amelynek az bizonyítását az Olvasóra bízzuk (lásd 24.2-14

gyakorlat), állítása a korábban már említett okok miatt racionális együtthatók esetén is igaz marad. Az eddigi eredményeket a következő tételben foglaljuk össze.

24.37. tétel. *A Dikin-féle affin skálázási algoritmussal egy* $\mathbf{u}^0 \in \mathcal{F}^0$ $(\mathbf{z}^0 > \mathbf{0})$ *pontból, amelyre* $\partial_c(\mathbf{u}^0) \leq \tau = 2$ *és* $(\mathbf{u}^0)^T\mathbf{z}^0 = n$ *teljesül, előállítható egy olyan* $\mathbf{u} \in \mathcal{F}^0$ $(\mathbf{z} > \mathbf{0})$ *pont, amelyből a kerekítési eljárással szigorúan komplementáris megoldás számolható ki. Az* $(\mathbf{u}, \mathbf{z})$ *pont előállításához* $\left\lceil 2n \lg \left(6n^2 w^2 \sqrt{|\mathcal{B}|} \pi_{\mathcal{B}} / \sigma^2 \right) \right\rceil$ *iteráció szükséges.*

Gyakorlatok

24.2-1. Bizonyítsuk be, hogy a (24.10) optimalizálási feladatnak a $-\mathbf{c}/\|\mathbf{c}\|$ vektor az optimális megoldása.

24.2-2. Tekintsük az egyszerűség kedvéért a $B(\mathbf{0}, 1) = \{\mathbf{u} \in \mathbb{R}^n : \|\mathbf{u}\| \leq 1\}$ gömböt és az $E = \{\mathbf{y} \in \mathbb{R}^n : \mathbf{y}^T Q \mathbf{y} \leq 1\}$ ellipszoidot, ahol a $Q \in \mathbb{R}^{n \times n}$ szimmetrikus pozitív szemidefinit mátrix. Bizonyítsuk be, hogy létezik egy-egy értelmű megfeleltetés a $B(\mathbf{0}, 1)$ és az E halmazok között.

24.2-3. Legyen $D \in \mathbb{R}^{n \times n}$ pozitív definit, $M \in \mathbb{R}^{n \times n}$ ferdén szimmetrikus mátrix. Bizonyítsuk be, hogy a $D+M$ és az $I+DMD$ pozitív definit mátrixok.

24.2-4. Legyenek $A, B \in \mathbb{R}^{n \times n}$ pozitív definit mátrixok. Bizonyítsuk be, hogy az $A^{-1}B$ is pozitív definit mátrix.

24.2-5. Bizonyítsuk be a 24.23. állítás második egyenlőségét.

24.2-6. Ellenőrizzük, hogy az $\bar{\alpha}$ valóban megfelelő felső korlát a megengedett lépéshosszra. *Útmutatás.* Az $\bar{\alpha}$ értéket a (24.11) képlettel definiáltuk.

24.2-7. Ellenőrizzük az (24.13) egyenlőtlenségből nyerhető $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ korlátokat (lásd 510. oldal).

24.2-8. A skálázásnál bevezetett jelöléseket felhasználva bizonyítsuk be, hogy

$$\|\mathbf{p}_u \, \mathbf{p}_z\|_\infty \leq \frac{1}{4} \|\mathbf{p}_u + \mathbf{p}_z\|^2 = \frac{1}{4} \|\mathbf{p}\|^2 \, .$$

24.2-9. Bizonyítsuk be a 24.24. lemmát. *Útmutatás.* Induljunk ki a (24.12) egyenlőség jobb oldalán álló kifejezésből és vezessük be az

$$f : \quad t \mapsto t - \alpha \frac{t^2}{\|\mathbf{v}^2\|}$$

függvényt, ahol az $\mathbf{v}^2$ vektor helyett írtuk be a t változót. A függvény elemzéséből kapjuk az első korlátot. A második korlát kiszámításakor használnunk kell a centralitási mértékre való elvárásokat és a 24.2-8 gyakorlat eredményét.

24.2-10. Legyen az $R \in \mathbb{Z}^{m\times m}$ reguláris mátrix és $\mathbf{r} \in \mathbb{Z}^m$ vektor. Bizonyítsuk be, hogy az $R\,\mathbf{u} = \mathbf{r}$ egyenlet bármely megoldása esetén, ha $y_i \neq 0$, akkor

$$y_i \geq \frac{1}{\prod_{j=1}^{n} \|\mathbf{r}_j\|} ,$$

ahol $\mathbf{r}_j$ az R mátrix j. oszlopa. *Útmutatás.* Alkalmazzuk a Cramer-szabályt és a Hadamard-egyenlőtlenséget.

24.2-11. Tekintsük a következő megengedettségi feladatot:

$$\mathcal{M} = \{\mathbf{y} \in \mathbb{R}^n \,;\; R\,\mathbf{y} = \mathbf{r},\; \mathbf{y} \geq \mathbf{0}\} ,$$

ahol $R \in \mathbb{Z}^{m\times n}$ és $\mathbf{r} \in \mathbb{Z}^m$, $\mathbf{r} \neq \mathbf{0}$. Legyen az $\mathcal{M}$ korlátos halmaz és tegyük fel, hogy tartalmaz egy pozitív vektort. Bizonyítsuk be, hogy bármely $i \in \mathcal{I}$ index esetén

$$\max_{\mathbf{y}\in\mathcal{M}} y_i \geq \frac{1}{\prod_{j=1}^{n} \|\mathbf{r}_j\|} .$$

Útmutatás. Vegyük figyelembe az R mátrix rangjának az értékét – legfeljebb m – és alkalmazzuk az előző gyakorlatot.

24.2-12. Bizonyítsuk be a 24.29. lemmát. *Útmutatás.* Számoljuk ki a következő összefüggést $(\mathbf{u}(\mu) - \mathbf{u}^*)^T(\mathbf{z}(\mu) - \mathbf{z}^*)$, ahol $(\mathbf{u}^*, \mathbf{z}^*)$ optimális megoldása az (SP) feladatnak.

24.2-13. Bizonyítsuk be a 24.32. lemmát.

24.2-14. Bizonyítsuk be a 24.36. lemmát. *Útmutatás.* Bizonyítsuk be, hogy az (24.16) egyenletnek van olyan megoldása, amelyre $\mathbf{u}_\mathcal{B} - \xi > \mathbf{0}$ és $\overline{\mathbf{z}}_\mathcal{N} = \mathbf{z}_\mathcal{N} - M_{\mathcal{N}\mathcal{N}}\,\mathbf{u}_\mathcal{N} - M_{\mathcal{N}\mathcal{B}}\,\xi > \mathbf{0}$ teljesül, az ε megállási paraméterre tett feltevés mellett.

24.2-15. Részletezzük a 24.37. tétel bizonyítását. *Útmutatás.* Alkalmazzuk a 24.36. lemmát és a 24.26. tételt.

24.2-16. Készítsük el a Dikin-algoritmus gyakorlati változatának MATLAB megvalósítását. Használjuk fel a MATLAB adta numerikus lehetőségeket (Cholesky-faktorizáció, ritka mátrixok használata stb.) és körültekintően (figyeljünk a beágyazásra, az induló belsőpontos megoldás meghatározására, az algoritmus paramétereinek a kellő megválasztására, az eredmény megfelelő értelmezésére stb.) végezzük el a számítógépes megvalósítást. Ha gondosan járunk el, akkor akár néhány száz feltételes és több száz változós gyakorlati feladatot – általában – száznál nem több iterációval meg tudunk oldani.

24.3. Primál-duál belsőpontos algoritmusok

A primál-duál belsőpontos algoritmusok a gyakorlatban is hatékonyan használható módszerek. Az ilyen algoritmusoknak két sarkalatos pontja van. Az egyik a μ centralitási paraméter iterálási módja, a másik pedig, hogy milyen centralitási mértéket használunk a pontunk és a centrális út távolságának mérésére. Az első kérdés szempontjából három csoportba oszthatjuk a primál-duál belsőpontos algoritmusokat. Általánosan a centralitási paramétert a $\mu^+ = (1-\theta)\,\mu$ szabály szerint frissítjük, ahol $0 < \theta < 1$, és a θ értékének megfelelően osztályozunk.

Ha θ egy konstans szám, akkor az algoritmus hosszú lépéses, ha θ a feladat dimenziójától, azaz n értékétől függ, akkor pedig rövid lépéses módszerről beszélünk. A harmadik csoportot az adaptív algoritmusok alkotják, ahol a θ paraméter ellentétben az előző két esettel az iterációk során változik, függ az aktuális ponttól. A rövid lépéses módszerek $\mathcal{O}\left(\sqrt{n}\lg\frac{n}{\varepsilon}\right)$, míg a hosszú lépéses módszerek esetén $\mathcal{O}\left(n\lg\frac{n}{\varepsilon}\right)$ a bizonyított lépésszám felső korlát. Ez ellentmond a gyakorlati tapasztalatoknak, miszerint a hosszú lépéses algoritmusok hatékonyabbak.

A másik lényeges kérdés, a centralitási mérték megválasztása esetén is többfajta megoldással találkozunk a szakirodalomban. A centrális úton az $\mathbf{v}$ vektor minden koordinátája egyenlő, így kézenfekvő a

$$\delta_c(\mathbf{v}) = \frac{\max(\mathbf{v}^2)}{\min(\mathbf{v}^2)}$$

módon definiálni a centralitási mértéket, ezt használtuk a Dikin-féle affin skálázású algoritmus elemzése során is. Ez a fajta centralitási mérték nem mindig felel meg céljainknak, mert a vektornak csak a maximális, illetve minimális elemét veszi figyelembe. Az alábbi mérték már a vektor összes koordinátájától függ, valamint a centrális úton lévő pontok esetén nulla az értéke, ami jobban tükrözi célunkat, miszerint a centrális út és az aktuális pont távolságát szeretnénk jellemezni vele.

$$\delta_0(\mathbf{v}) = \frac{1}{2}\left\|\mathbf{e} - \mathbf{v}^2\right\| .$$

Az ebben a részben használt centralitási mérték a szakirodalomban egyik leggyakrabban alkalmazott mérték, amelyet a következő képlettel definiálunk:

$$\delta_1(\mathbf{v}) = \frac{1}{2}\left\|\mathbf{v} - \mathbf{v}^{-1}\right\| .$$

A $\delta_1(\mathbf{v})$ centralitási mérték még inkább megfelel az elvárásainknak, mint a $\delta_0(\mathbf{v})$. Nem csak hogy eltűnik a centrális úton, hanem ha egy bizonyos korlát alatt tartjuk, akkor nem engedi a pontunkat a megengedett tartomány

határhoz túl közel, ugyanis akár az $\mathbf{x}$, akár az $\mathbf{s}$ vektor valamely koordinátája tart nullához, vagy végtelenbe, akkor a pont centralitási mértéke is tart végtelenbe. (Ennek a tulajdonságnak csak a fele teljesül a δ_0 mérték esetén.) A δ_1 centralitási mérték általánosításának tekinthetők a legújabb, önreguláris függvények segítségével definiált mértékek (lásd a 24.3.5. pontot).

Fő célunk, amelyet a következő részben fejtünk ki, a gyakorlatban is hatékony belsőpontos algoritmusok bemutatása. Először a ***primál-duál Newton-lépéses belsőpontos algoritmust*** ismertetjük, mely teljes Newton-lépések megtételével konvergál a megoldáshoz. Egyszerűen bizonyítható az eljárás polinomialitása, belátható, hogy az algoritmus lépésszáma $O(\sqrt{n}\lg\frac{n}{\varepsilon})$. A primál-duál Newton-lépéses belsőpontos algoritmusok továbbfejlesztésének tekinthetjük a prediktor-korrektor algoritmusokat, melyek egyik első változatát Sonnevend, Stoer és Zhao vezette be lineáris programozási feladatokra. Sonnevendék algoritmusának a prediktor lépés után több korrektor lépésre volt szüksége ahhoz, hogy a centrális út megfelelő, előírt környezetébe visszajusson. Mizuno, Todd és Ye olyan ***prediktor-korrektor belsőpontos algoritmust*** publikált lineáris programozási feladatra, amelynél egy prediktor lépést csupán egy korrektor lépés követett, és az algoritmus lépésszámára adott felső korlát a lineáris programozás szakirodalmából ismert legjobb. Ez utóbbi algoritmus egy változatát fogjuk ismertetni. A rövid lépéses prediktor-korrektor algoritmus elemzése után egy adaptív változatot is bemutatunk, amely esetén már kvadratikus konvergencia is bizonyítható.

24.3.1. Primál-duál Newton-lépéses belsőpontos algoritmus

A lineáris programozási feladatok különböző, egymással ekvivalens formában adhatók meg. Azt, hogy éppen melyik alakját használjuk a lineáris programozási feladatnak, elsősorban az határozza meg, hogy milyen algoritmussal szeretnénk megoldani, illetve a feladat milyen megadása segíti legjobban elő elemzéseink egyszerű bemutatását. A lineáris programozás belsőpontos szakirodalmában a ***primál*** és ***duál feladatokat*** leggyakrabban a következő alakban adjuk meg:

$$\left.\begin{array}{lcl} \min & \mathbf{c}^T\mathbf{x} & \\ A\mathbf{x} & = & \mathbf{b} \\ \mathbf{x} & \geq & \mathbf{0} \end{array}\right\} \quad (P), \qquad \left.\begin{array}{lcl} \max & \mathbf{b}^T\mathbf{y} & \\ A^T\mathbf{y} & \leq & \mathbf{c} \end{array}\right\} \quad (D)\,,$$

ahol $A \in \mathbb{R}^{m\times n}$, $\mathbf{c}, \mathbf{x} \in \mathbb{R}^n$és $\mathbf{y}, \mathbf{b} \in \mathbb{R}^m$. Az általánosság korlátozása nélkül feltehető, hogy $\operatorname{rang}(A) = m$. Legyen a ***primál,*** illetve a ***duál megengedett megoldások*** halmaza rendre

$$\mathcal{P} = \{\mathbf{x} \in \mathbb{R}^n_{\oplus} \,:\, A\mathbf{x} = \mathbf{b}\} \quad \text{és} \quad \mathcal{D} = \{(\mathbf{y}, \mathbf{s}) \in \mathbb{R}^{m+n} :\, A^T\mathbf{y} + \mathbf{s} = \mathbf{c},\ \mathbf{s} \geq \mathbf{0}\}\,.$$

Jelölje

$$\mathcal{P}_+ = \{\mathbf{x} \in \mathcal{P} : \mathbf{x} > \mathbf{0}\} \qquad \text{és} \qquad \mathcal{D}_+ = \{(\mathbf{y}, \mathbf{s}) \in \mathcal{D} : \mathbf{s} > \mathbf{0}\}$$

a ***primál,*** illetve ***duál belső pontok halmazát***.
Belső pont feltétel: létezik $\bar{\mathbf{x}} \in \mathcal{P}_+$ és létezik $(\bar{\mathbf{y}}, \bar{\mathbf{s}}) \in \mathcal{D}_+$.

A belső pont feltétel nem jelent megszorítást a 24.1. alfejezetben bemutatott beágyazási feladat (497. oldal) segítségével bármely lineáris programozási feladat beágyazható egy olyan önduális feladatba, mely teljesíti a belső pont feltételt, és a beágyazási feladat megoldása az eredeti feladat egy megoldását adja. A belsőpontos feltételek teljesülése esetén, azaz ha $\mathcal{P}_+ \neq \emptyset$ és $\mathcal{D}_+ \neq \emptyset$, akkor az erős dualitás tétel alapján létezik primál (és duál) optimális megoldás.
A ***primál,*** illetve a ***duál optimális megoldások halmazát*** a következőképpen jelöljük:

$$\mathcal{P}^* = \{\mathbf{x}^* \in \mathcal{P} : \mathbf{c}^T\mathbf{x}^* \leq \mathbf{c}^T\mathbf{x}, \ \ \forall \mathbf{x} \in \mathcal{P}\}$$
$$\text{és} \quad \mathcal{D}^* = \{\mathbf{y}^* \in \mathcal{D} : \mathbf{b}^T\mathbf{y}^* \geq \mathbf{b}^T\mathbf{y}, \ \ \forall \mathbf{y} \in \mathcal{D}\} .$$

Dualitásrés: $\mathbf{c}^T\mathbf{x} - \mathbf{b}^T\mathbf{y} = (A^T\mathbf{y} + \mathbf{s})^T\mathbf{x} - \mathbf{y}^T(A\,\mathbf{x}) = \mathbf{s}^T\mathbf{x}$.

A következő állítás könnyen belátható, ezért az Olvasóra bízzuk (lásd 24.3-2 gyakorlat).

24.38. állítás. *Legyenek az $\mathbf{x}, \mathbf{s} \in \mathbb{R}^n$ olyan vektorok, amelyekre $\mathbf{x} \geq \mathbf{0}$ és $\mathbf{s} \geq \mathbf{0}$ teljesül. Az $\mathbf{x}$ és $\mathbf{s}$ vektorok pontosan akkor ortogonálisak, ha $x_i\, s_i = 0$ teljesül $i = 1, 2, \ldots, n$ esetén.*

Felhasználva az előző állítást az ***optimalitási kritériumok*** az alábbi formában írhatók

$$\begin{array}{rrcll} & A\,\mathbf{x} & = & \mathbf{b}, & \mathbf{x} \geq \mathbf{0}\,, \\ A^T\mathbf{y}+ & \mathbf{s} & = & \mathbf{c}, & \mathbf{s} \geq \mathbf{0}\,, \\ & \mathbf{x}\,\mathbf{s} & = & \mathbf{0}\,. & \end{array}$$

24.39. állítás. *Ha az $\bar{\mathbf{x}} \in \mathcal{P}_+$ és $(\bar{\mathbf{y}}, \bar{\mathbf{s}}) \in \mathcal{D}_+$, akkor $\bar{\mathbf{x}}\,\bar{\mathbf{s}} = \bar{\mathbf{w}} \in \mathbb{R}^n_+$, azaz az optimum belső pontban nem vétetik fel.*

A 24.39. állítás miatt, amelynek az bizonyítását az Olvasóra bízzuk (lásd 24.3-3 gyakorlat), az optimalitási kritériumokat relaxáljuk, olyan primál és duál belsőpontos megoldáspárt szeretnénk előállítani, amelyre

$$\begin{array}{rrcll} & A\,\mathbf{x} & = & \mathbf{b}, & \mathbf{x} > \mathbf{0}\,, \\ A^T\mathbf{y}+ & \mathbf{s} & = & \mathbf{c}, & \mathbf{s} > \mathbf{0}\,, \\ & \mathbf{x}\,\mathbf{s} & = & \mu\,\mathbf{e} & \end{array}$$

teljesül, ahol $\mu > 0$ adott. Az $(\bar{\mathbf{x}}, \bar{\mathbf{s}}) > 0$ primál-duál megoldásból szeretnénk az előző ***relaxált optimalitási kritériumok*** egy megoldását előállítani oly módon, hogy valamilyen ismeretlen $\Delta\mathbf{x}$, $\Delta\mathbf{y}$, $\Delta\mathbf{s}$ értékkel módosítjuk a megengedett megoldást:

$$\begin{array}{rrrcll} & A\,(\bar{\mathbf{x}}+\Delta\mathbf{x}) & & = & \mathbf{b}, & \bar{\mathbf{x}}+\Delta\mathbf{x}>\mathbf{0}\,, \\ A^T(\bar{\mathbf{y}}+\Delta\mathbf{y})+ & & (\bar{\mathbf{s}}+\Delta\mathbf{s}) & = & \mathbf{c}, & \bar{\mathbf{s}}+\Delta\mathbf{s}>\mathbf{0}\,, \\ & & (\bar{\mathbf{x}}+\Delta\mathbf{x})\,(\bar{\mathbf{s}}+\Delta\mathbf{s}) & = & \mu\,\mathbf{e}\,. & \end{array}$$

Egyszerű számítások után kapjuk, hogy

$$\begin{array}{rrrcll} & A\,\Delta\mathbf{x} & & = & \mathbf{0}, & \bar{\mathbf{x}}+\Delta\mathbf{x}>\mathbf{0}\,, \\ A^T\Delta\mathbf{y}+ & & \Delta\mathbf{s} & = & \mathbf{0}, & \bar{\mathbf{s}}+\Delta\mathbf{s}>\mathbf{0}\,, \\ \bar{\mathbf{s}}\,\Delta\mathbf{x}+ & \bar{\mathbf{x}}\,\Delta\mathbf{s}+ & \Delta\mathbf{x}\,\Delta\mathbf{s} & = & \mu\,\mathbf{e}-\bar{\mathbf{x}}\,\bar{\mathbf{s}}\,, & \end{array}$$

ahol az utolsó egyenlet nemlineáris (kvadratikus). Ezt a nemlineáris egyenlőtlenségrendszert egyszerűsítjük. Két dolgot hagyunk el: a pozitivitási megkötéseket és a másodrendű tagot. Ekkor a harmadik egyenlet a következő alakúvá válik.

$$\bar{\mathbf{s}}\,\Delta\mathbf{x}+\bar{\mathbf{x}}\,\Delta\mathbf{s}=\mu\,\mathbf{e}-\bar{\mathbf{x}}\,\bar{\mathbf{s}}\,.$$

Newton-rendszer: keresendő a $(\Delta\mathbf{x}, \Delta\mathbf{y}, \Delta\mathbf{s}) \in \mathbb{R}^{n+m+n}$ vektor úgy, hogy

$$\begin{array}{rrcl} & A\,\Delta\mathbf{x} & = & \mathbf{0}\,, \\ A^T\Delta\mathbf{y}+ & \Delta\mathbf{s} & = & \mathbf{0}\,, \\ \bar{\mathbf{s}}\,\Delta\mathbf{x}+ & \bar{\mathbf{x}}\,\Delta\mathbf{s} & = & \mu\,\mathbf{e}-\bar{\mathbf{x}}\,\bar{\mathbf{s}}\,. \end{array}$$

24.40. állítás. *Legyen* $\operatorname{rang}(A) = m$. *A Newton rendszer megoldása egyértelmű.*

A bizonyítást az Olvasóra bízzuk (lásd 24.3-4 gyakorlat).

Legyenek $\bar{S} = \operatorname{diag}(\bar{\mathbf{s}})$ és $\bar{X} = \operatorname{diag}(\bar{\mathbf{x}})$ pozitív mátrixok. Ekkor az utolsó egyenletet a következő formára hozzuk:

$$\bar{S}\,\Delta\mathbf{x}+\bar{X}\,\Delta\mathbf{s}=\mu\,\mathbf{e}-\bar{S}\,\bar{\mathbf{x}}\,,\quad \text{azaz}\quad \Delta\mathbf{x}+\bar{S}^{-1}\bar{X}\,\Delta\mathbf{s}=\mu\,\bar{\mathbf{s}}^{-1}-\bar{\mathbf{x}}\,.$$

Alkalmazzuk az A mátrixot a kapott n-dimenziós vektorokra és használjuk fel az $\bar{\mathbf{x}} \in \mathcal{P}$ összefüggést, illetve a Newton-rendszer első egyenletét. Ekkor

$$A\bar{X}\bar{S}^{-1}\Delta\mathbf{s}=A\,\Delta\mathbf{x}+A\bar{X}\bar{S}^{-1}\Delta\mathbf{s}=\mu\,A\,\bar{\mathbf{s}}^{-1}-A\,\bar{\mathbf{x}}=\mu\,A\,\bar{\mathbf{s}}^{-1}-\mathbf{b}\,.$$

A Newton-rendszer második egyenletére alkalmazzuk az $A\bar{X}\bar{S}^{-1}$ mátrixot, majd pedig rendezzük át az egyenletet. Ekkor

$$-A\,\bar{X}\,\bar{S}^{-1}\,A^T\Delta\mathbf{y}=A\bar{X}\bar{S}^{-1}\Delta\mathbf{s}=\mu\,A\bar{\mathbf{s}}^{-1}-\mathbf{b}\,.$$

Mivel $\operatorname{rang}(A) = m$, ezért $A\bar{X}\bar{S}^{-1}A^T \in \mathbb{R}^{m\times m}$ és nem szinguláris mátrix, hiszen az A teljes rangú, így $\Delta\mathbf{y}$ egyértelműen kiszámítható

$$\Delta\mathbf{y} = (A\bar{X}\bar{S}^{-1}A^T)^{-1}(\mathbf{b} - \mu\, A\bar{\mathbf{s}}^{-1}) \,. \tag{24.17}$$

Ezek után egyszerűen és egyértelműen kiszámíthatók a $\Delta\mathbf{s}$ és a $\Delta\mathbf{x}$ vektorok, azaz

$$\Delta\mathbf{s} = -A^T\Delta\mathbf{y} \qquad \text{és} \qquad \Delta\mathbf{x} = \mu\,\bar{\mathbf{s}}^{-1} - \bar{\mathbf{x}} - \bar{X}\bar{S}^{-1}\Delta\mathbf{s} \,. \tag{24.18}$$

Tekintettel arra, hogy kiszámoltuk az adott megoldás módosítására szolgáló vektorokat, amelyek segítségével a μ-centrumot megközelíthetjük, minden lényeges információval rendelkezünk ahhoz, hogy megfogalmazzuk az algoritmusunkat. A μ-centrumtól vett távolságot a $\partial_1(\mathbf{x},\mathbf{s},\mu) = \delta_1(\mathbf{v})$ centralitási mértékkel mérjük, ahol $\mathbf{v} = \sqrt{\mathbf{xs}/\mu}$.

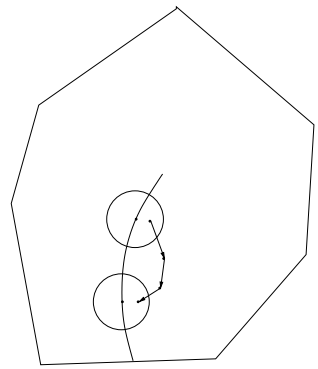
Az eljárás bemenő adataként megadjuk az $\varepsilon > 0$ megkívánt számítási pontosságot, a $\tau \geq 0$ eltérés paramétert, a θ $(0 < \theta < 1)$ előre rögzített csökkenési paramétert, valamint egy $(\mathbf{x}^0,\mathbf{y}^0,\mathbf{s}^0)$ belső pontot, amely a τ által meghatározott környezetben van, azaz $(\mathbf{x}^0,\mathbf{y}^0,\mathbf{s}^0) \in \mathcal{P}_+\times\mathcal{D}_+$, $(\mathbf{x}^0)^T\mathbf{s}^0 = \mu^0 n$ és $\partial_1(\mathbf{x}^0,\mathbf{s}^0,\mu^0) \leq \tau$.

PRIMÁL-DUÁL-NEWTON-LÉPÉSES-BELSŐPONTOS

```
1  x = x^0
2  s = s^0
3  μ = μ^0
4  while nμ > ε
5        do μ = (1 − θ)μ
6           while ∂1(x, s; μ) > τ
7                 do határozzuk meg Δx-et és Δs-t (24.18) alapján
8                    számítsuk ki az α > 0 megengedett lépéshosszt
9                    x ← x + αΔx
10                   s ← s + αΔs
```

A fenti algoritmus egy belső iterációjának lépéseit a 24.3. ábra szemlélteti.

Az algoritmus esetén a gyakorlati és elméleti változatokat az különbözteti meg, hogy a τ és a θ paraméterek, valamint az α lépéshossz megválasztása hogyan történik. Világos az algoritmus szerkezetéből, hogy a θ paraméter megválasztása befolyásolni fogja azt, hogy a belső ciklust hányszor kell majd végrehajtani. A belső ciklus elvégzésének a számát befolyásolhatjuk azzal, hogy a τ értékét a θ értékétől függően választjuk meg. A θ rögzítése természetesen kihat az α lehetséges értékére is. Az α lépéshossz megválasztásakor



24.3. ábra. Primál-duál algoritmus iterációi a primál változók terében.

az a célunk, hogy a

$$\partial_1(\mathbf{x}+\alpha\,\Delta\mathbf{x},\mathbf{s}+\alpha\,\Delta\mathbf{s};\mu)$$

centralitási mértéket minimalizáljuk az α paraméterben.

Mielőtt rátérnénk az algoritmus egy lehetséges gyakorlati, illetve elméleti változatának az elemzésére, néhány olyan állítással foglalkozunk, amelyek ismeretére mindenfeleképpen szükségünk lesz. Emellett az iterációnként előforduló feladatot, ún. ***átskálázással*** egységes formára hozzuk. Ez az egységes forma a számításainkat és az elemzéseinket is leegyszerűsíti.

24.41. lemma. *$\Delta\mathbf{x}$ és $\Delta\mathbf{s}$ ortogonális vektorok.*

A $\Delta\mathbf{x}$ és a $\Delta\mathbf{s}$ Newton-lépések merőlegességét, amelynek az bizonyítását az Olvasóra bízzuk (lásd 24.3-5 gyakorlat), felhasználva szükséges és elégséges feltételt adhatunk a teljes Newton-lépés megengedettségére. (Ennek az bizonyítása is egy újabb feladat az Olvasó számára, lásd 24.3-6 gyakorlat.)

24.42. lemma. *A (P) és (D) feladatok esetén $\bar{\mathbf{x}}+\Delta\mathbf{x}\geq\mathbf{0}$ és $\bar{\mathbf{s}}+\Delta\mathbf{s}\geq\mathbf{0}$ pontosan akkor teljesül, ha $\mu\,\mathbf{e}+\Delta\mathbf{x}\,\Delta\mathbf{s}\geq\mathbf{0}$, ahol $\bar{\mathbf{x}}\in\mathcal{P}_+$, $(\bar{\mathbf{y}},\bar{\mathbf{s}})\in\mathcal{D}_+$ és $\mu>0$ a rögzített cél centrum paramétere.*

A továbbiakban jelölje

$$\mathbf{x}^+=\bar{\mathbf{x}}+\Delta\mathbf{x}\,,\quad \mathbf{y}^+=\bar{\mathbf{y}}+\Delta\mathbf{y}\quad\text{és}\quad \mathbf{s}^+=\bar{\mathbf{s}}+\Delta\mathbf{x}$$

a Newton-lépéssel kapott új pontokat. Az új pontok által meghatározott dualitásrés kiszámítása is az Olvasó feladata lesz (lásd 24.3-8 gyakorlat), amelyet az alábbi lemmában fogalmazunk meg.

24.43. lemma. *A (P) és (D) feladatok esetén $\bar{\mathbf{x}}+\Delta\mathbf{x}\geq\mathbf{0}$ és $\bar{\mathbf{s}}+\Delta\mathbf{s}\geq\mathbf{0}$ pontosan akkor teljesül, ha $\mu\,\mathbf{e}+\Delta\mathbf{x}\,\Delta\mathbf{s}\geq\mathbf{0}$, ahol $\bar{\mathbf{x}}\in\mathcal{P}_+$, $(\bar{\mathbf{y}},\bar{\mathbf{s}})\in\mathcal{D}_+$ és $\mu>0$ a rögzített cél centrum paramétere.*

24.44. lemma. *A (P) és (D) feladatok esetén, ha a $(\Delta\mathbf{x},\,\Delta\mathbf{s})$ olyan, hogy $\mathbf{x}^+\in P_+$, $(\mathbf{y}^+,\mathbf{s}^+)\in D_+$, akkor $(\mathbf{x}^+)^T\mathbf{s}^+=\mu\,n$, ahol $\mu>0$ a rögzített cél centrum paramétere.*

Átskálázás

Célunk olyan jelölések bevezetése, melyekkel a Newton-rendszer egyszerűbb alakra hozható, ezzel kiemelve a feladat struktúráját. Az alábbi vektorok segítségével átírt Newton-rendszer harmadik feltételének jobb oldalán megjelenik a $\mathbf{v}^{-1}-\mathbf{v}$ vektor, melynek normája fogja mérni a pontunk távolságát a centrális úttól, ugyanis ha a pontunk rajta van a centrális úton, akkor ez a kifejezés nullával egyenlő. Legyen $\bar{\mathbf{x}}\in\mathcal{P}_+$ és $(\bar{\mathbf{y}},\,\bar{\mathbf{s}})\in\mathcal{D}_+$. Definiáljuk a következő vektorokat:

$$\mathbf{d}=\sqrt{\frac{\bar{\mathbf{x}}}{\bar{\mathbf{s}}}}\,,\quad \mathbf{v}=\sqrt{\frac{\bar{\mathbf{x}}\,\bar{\mathbf{s}}}{\mu}}\,,\qquad \text{ekkor}\qquad \frac{\mathbf{d}^{-1}\,\bar{\mathbf{x}}}{\sqrt{\mu}}=\frac{\mathbf{d}\,\bar{\mathbf{s}}}{\sqrt{\mu}}=\mathbf{v}\,.$$

Továbbá

$$\mathbf{d}_x=\frac{\mathbf{d}^{-1}\,\Delta\mathbf{x}}{\sqrt{\mu}}\qquad\text{és}\qquad \mathbf{d}_s=\frac{\mathbf{d}\,\Delta\mathbf{s}}{\sqrt{\mu}}\,.$$

Ekkor

$$\mathbf{x}^+=\bar{\mathbf{x}}+\Delta\mathbf{x}=\sqrt{\mu}\,\mathbf{d}\,(\mathbf{v}+\mathbf{d}_x)\,,\quad\text{és}\quad \mathbf{s}^+=\bar{\mathbf{s}}+\Delta\mathbf{s}=\sqrt{\mu}\,\mathbf{d}^{-1}\,(\mathbf{v}+\mathbf{d}_s)\,,\quad\text{valamint}$$

$$\mathbf{x}^+\,\mathbf{s}^+=\mu\,\mathbf{e}+\Delta\mathbf{x}\,\Delta\mathbf{s}=\mu\,\mathbf{e}+\sqrt{\mu}\,\mathbf{d}\,\mathbf{d}_x\,\sqrt{\mu}\,\mathbf{d}^{-1}\,\mathbf{d}_s=\mu\,(\mathbf{e}+\mathbf{d}_x\,\mathbf{d}_s)\,.$$

Az új jelölésekkel a 24.43. lemma eredménye a következő alakban írható, amelynek a bizonyítása az Olvasó feladata lesz (lásd 24.3-9 gyakorlat).

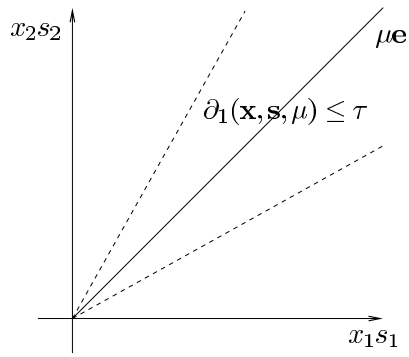
24.45. következmény. *A (P) és (D) feladatok esetén $\bar{\mathbf{x}}+\Delta\mathbf{x}\geq\mathbf{0}$ és $\bar{\mathbf{s}}+\Delta\mathbf{s}\geq\mathbf{0}$ pontosan akkor teljesül, ha $\mathbf{e}+\mathbf{d}_x\,\mathbf{d}_s\geq\mathbf{0}$, ahol $\bar{\mathbf{x}}\in\mathcal{P}_+$ és $(\bar{\mathbf{y}},\bar{\mathbf{s}})\in\mathcal{D}_+$.*

Mivel $\bar{\mathbf{s}}\,\mathbf{d}=\bar{\mathbf{x}}\,\mathbf{d}^{-1}=\sqrt{\mu}\,\mathbf{v}$, ezért

$$\bar{\mathbf{x}}\,\Delta\mathbf{s}+\bar{\mathbf{s}}\,\Delta\mathbf{x}=\bar{\mathbf{x}}\sqrt{\mu}\,\mathbf{d}^{-1}\,\mathbf{d}_s+\bar{\mathbf{s}}\sqrt{\mu}\,\mathbf{d}\,\mathbf{d}_x=\sqrt{\mu}\,(\bar{\mathbf{s}}\,\mathbf{d}\,\mathbf{d}_x+\bar{\mathbf{x}}\,\mathbf{d}^{-1}\,\mathbf{d}_s)=\mu\,\mathbf{v}\,(\mathbf{d}_x+\mathbf{d}_s)\,.$$

Másfelől $\mu\,\mathbf{e}-\bar{\mathbf{x}}\,\bar{\mathbf{s}}=\bar{\mathbf{x}}\,\Delta\mathbf{s}+\bar{\mathbf{s}}\Delta\mathbf{x}=\mu\,\mathbf{v}\,(\mathbf{d}_x+\mathbf{d}_s)$, és $\mu\,\mathbf{e}-\bar{\mathbf{x}}\,\bar{\mathbf{s}}=\mu\,\mathbf{e}-\mu\,\mathbf{v}^2=\mu\,\mathbf{v}\,(\mathbf{v}^{-1}-\mathbf{v})$, tehát $\mathbf{d}_x+\mathbf{d}_s=\mathbf{v}^{-1}-\mathbf{v}$. Bevezetve a $\mathbf{d}_y=\Delta\mathbf{y}/\sqrt{\mu}$ vektort, az átskálázott Newton-rendszer a következő lesz:

$$\begin{aligned} AD\,\mathbf{d}_x \qquad\quad &= \mathbf{0}\,,\\ (AD)^T\mathbf{d}_y+\qquad \mathbf{d}_s &= \mathbf{0}\,,\\ \mathbf{d}_x+\quad \mathbf{d}_s &= \mathbf{v}^{-1}-\mathbf{v}\,. \end{aligned}$$



24.4. ábra. Centrális út környezete az átskálázott térben, ahol $\mu = \mathbf{x}^T\mathbf{s}/n$ és $x_1s_1 = \mu\, v_1^2$, $x_2s_2 = \mu\, v_2^2$.

Tehát a $\mathbf{d}_x \in \mathrm{Null}(AD)$, és a $\mathbf{d}_s \in \mathrm{rowsp}(AD) = \mathrm{Null}(HD^{-1})$, ahol H tetszőleges olyan mátrix, amelynek nulltere megegyezik az A mátrix sorterével.[14] Ekkor

$$\mathbf{d}_x = P_{AD}(\mathbf{v}^{-1} - \mathbf{v}) \;, \quad \text{és} \quad \mathbf{d}_s = P_{HD^{-1}}(\mathbf{v}^{-1} - \mathbf{v}) \;, \tag{24.19}$$

és mivel $(\Delta\mathbf{x})^T\Delta\mathbf{s} = \mathbf{d}_x^T\mathbf{d}_s = 0$, ezért

$$\|\mathbf{d}_x\|^2 + \|\mathbf{d}_s\|^2 = \|\mathbf{v}^{-1} - \mathbf{v}\|^2 \;.$$

A $\mathbf{d}_x + \mathbf{d}_s = \mathbf{0}$ egyenlőség pontosan akkor teljesül, ha $\mathbf{v}^{-1} - \mathbf{v} = \mathbf{0}$, ami azzal ekvivalens, hogy $\mathbf{v}^{-1} = \mathbf{v}$, azaz $\mathbf{v} = \mathbf{e}$. Átskálázás után centralitási mértékünk a következő alakú lesz

$$\partial_1(\bar{\mathbf{x}}, \bar{\mathbf{s}}; \mu) = \frac{1}{2}\left\| \sqrt{\frac{\mu\,\mathbf{e}}{\bar{\mathbf{x}}\,\bar{\mathbf{s}}}} - \sqrt{\frac{\bar{\mathbf{x}}\,\bar{\mathbf{s}}}{\mu}} \right\| = \frac{1}{2}\,\|\mathbf{v}^{-1} - \mathbf{v}\| \;.$$

A 24.4. ábra a centrális út környezetét szemlélteti, azaz azon pontok halmazát az átskálázott terében, melyek centralitási mértéke egy megadott korlát alatt marad.

24.3.2. Primál-duál Newton-lépéses belsőpontos algoritmus: gyakorlati változat

Az algoritmus gyakorlati változatának az elemzésekor a következő feltételezésekkel élünk: a τ és a θ paraméterek értéke nem függ a feladat változóinak a

[14] $\mathrm{Null}(M)$ az M mátrix nullterét, míg rowsp(M) a mátrix sorterét jelöli. P_M pedig az a projekció, mely az M mátrix nullterére képez.

számától, legyen például $\tau = 100$ és $\theta = 0.9$.

Tegyük fel, hogy adottak az $\bar{\mathbf{x}} \in \mathcal{P}^+$ és az $(\bar{\mathbf{y}}, \bar{\mathbf{s}}) \in \mathcal{D}^+$ megoldás vektorok, amelyekhez $\bar{\mu} = \bar{\mathbf{x}}^T \bar{\mathbf{s}}/n$ dualitásrés tartozik, és teljesítik a $\partial_1(\bar{\mathbf{x}}, \bar{\mathbf{s}}; \bar{\mu}) \leq \tau = 100$ egyenlőtlenséget.

A Primál-duál-Newton-lépéses-belsőpontos algoritmust numerikus szempontból elemezve észrevehetjük, hogy az algoritmus leginkább munkaigényes része a $\Delta\mathbf{y}$ meghatározása, (24.17), hiszen az $R = A\,\bar{X}\,\bar{S}^{-1}\,A^T$ mátrix inverzének a kiszámítását igényeli. Ezzel szemben $\Delta\mathbf{s}$ és $\Delta\mathbf{x}$ (24.18) szerinti meghatározása már csak mátrix-vektor szorzást, illetve mátrix-vektor szorzást és vektor-vektor összeadást igényel.

Tegyük fel, hogy a MATLAB programrendszer segítségével szeretnénk a primál-duál Newton-lépéses belsőpontos algoritmus egy implementációját elkészíteni, ekkor már a $\Delta\mathbf{y}$ kiszámítására is több lehetőségünk nyílik:

1. `Δ y = inv(R)*r;`
2. `Δ y = R \ r;`

MATLAB utasítással számoljuk ki a $\Delta\mathbf{y}$ vektort, ahol $\mathbf{r} = \mathbf{b} - \mu\,A\,\bar{\mathbf{s}}^{-1}$.

Köztudott, hogy a mátrix invertálás numerikus szempontból az egyik leginkább instabil számítási eljárás, ezért más módszerhez célszerű folyamodnunk. A megoldandó feladatunk numerikus szempontból ugyanis az

$$R\ \Delta\mathbf{y} = \mathbf{r} \tag{24.20}$$

lineáris egyenletrendszer megoldása, ahol az $R \in \mathbb{R}^{m\times m}$ reguláris mátrix. Figyelembe véve, hogy az $\bar{X}, \bar{S}^{-1}$ pozitív diagonális mátrixok és $\operatorname{rang}(A) = m$, ezért az R mátrix szimmetrikus és pozitív definit lesz. A pozitív definit mátrixok Cholesky-faktorizációja numerikusan jóval stabilabb számítási eljárás, mint az invertálás, és az eredménye $R = L\,L^T$ alakú, ahol $L \in \mathbb{R}^{m\times m}$ alsó háromszög mátrix. Ekkor az egyenletünk

$$R\,\Delta\mathbf{y} = L\,(L^T\,\Delta\mathbf{y}) = \mathbf{r}$$

alakú lesz, és az

$$L\,\mathbf{z} = \mathbf{r} \quad \text{és} \quad L^T\,\Delta\mathbf{y} = \mathbf{z}$$

helyettesítésekkel a (24.20) egyenletrendszer megoldását két háromszög mátrixszal adott lineáris egyenletrendszer megoldására vezettük vissza. Tehát a harmadik lehetőségünk a Cholesky-faktorizáció

3. `L = chol(R) ;  z = L \ r;  Δ y = L' \ z;`

Ebben az esetben célszerű elkészíteni a visszahelyettesítésen alapuló lineáris egyenletrendszert megoldó eljárást. Megjegyezzük, hogy a MATLAB speciális esetben a 2. pontban tárgyalt LU felbontás esetén Cholesky-faktorizációt hajt végre.

Numerikus bonyodalmak

Tekintettel arra, hogy az $\mathbf{x}\,\mathbf{s} = \mu\,\mathbf{e}$ egyenlet megoldásait közelítjük az $\bar{\mathbf{x}}$ és $\bar{\mathbf{s}}$ vektorokkal, és a μ paraméterrel nullához tartunk (külső ciklus), nem meglepő, hogy minél kisebb lesz a μ, annál inkább rosszul kondicionált lesz a (24.20) egyenletrendszer mátrixa. Ennek a természetesen fellépő numerikus jelenségnek a kiküszöbölésére az ún. ***regularizációt***[15] ajánljuk, azaz az R mátrix helyett dolgozzunk az

$$R_\rho = \rho\, I + A\,(\rho\, I + \bar{X}\,\bar{S}^{-1})\,A^T$$

mátrixszal, ahol $\rho = 10^{-8}$. (Feltételezzük, hogy 32-bites gépen dolgozunk, dupla pontosságú lebegőpontos aritmetikával.) Az R_ρ mátrix, az R mátrixhoz hasonlóan pozitív definit lesz, de a sajátértékei soha sem válhatnak 10^{-8}-nál kisebbé.[16] A regularizáció a következő megfontolásokon alapul: mivel a numerikus gondok forrása az, hogy az $\bar{X}\,\bar{S}^{-1}$ mátrix diagonális elemei kicsikké válnak, a $\rho\, I$ mátrix hozzáadásával elértük, hogy a diagonális elemek értéke 10^{-8}-nál nem lehet kisebb.[17] Másfelől, az R mátrix rosszul kondicionáltságának az oka lehet maga az A mátrix is. Célszerű tehát az $A\,(\rho\, I + \bar{X}\,\bar{S}^{-1})\,A^T$ mátrixhoz is hozzáadni a $\rho\, I$ mátrixot. Ezt a módosítást az algoritmus legelején vezessük be, mert amíg az R sajátértékei 10^{-8}-nál nagyobbak, addig ennek úgy sincsen jelentős hatása, amikor pedig kisebbé válnak, akkor a regularizáció a számítások elvégzéséhez nélkülözhetetlen lesz.

Természetesen más módszer is ismert a szakirodalomból a mátrixok kondíciószámának a karbantartására.

Az α lépéshossz meghatározása

Miután kiszámítottuk a $\Delta\mathbf{x}$ és $\Delta\mathbf{s}$ irányokat, az $\alpha > 0$ lépéshosszt szeretnénk úgy meghatározni, hogy az megengedett legyen, azaz az

$$\bar{\mathbf{x}} + \alpha\,\Delta\mathbf{x} > \mathbf{0} \qquad \text{és} \qquad \bar{\mathbf{s}} + \alpha\,\Delta\mathbf{s} > \mathbf{0}$$

egyenlőtlenségek teljesüljenek. Ezekből az egyenlőtlenségekből kiszámítható az $\bar{\alpha} > 0$ felső korlát, (24.11). Világos, hogy létezik olyan index, amelyre vagy $\bar{x}_i + \bar{\alpha}\,(\Delta\mathbf{x})_i = 0$ vagy $\bar{s}_i + \bar{\alpha}\,(\Delta\mathbf{s})_i = 0$, teljesül. Ezért az $\bar{\alpha}$ érték esetén a ∂_1 centralitási mérték nem értelmezett. Elvileg a $\partial_1(\bar{\mathbf{x}} + \alpha\,\Delta\mathbf{x}, \bar{\mathbf{s}} + \alpha\,\Delta\mathbf{s}; \mu)$ minimumát kell meghatározni a $(0, \bar{\alpha})$ nyílt intervallumon. Figyelembe véve a ∂_1 függvény domináns büntető tulajdonságát, amikor a feltételek felé közelítünk, intuitíven, numerikus stabilitást is figyelembe véve belátható, hogy a ∂_1 függvényt minimalizáló α értéket a $[0.005, 0.995]$ intervallumban kell keresnünk.

---

[15] A regularizáció kérdésére a *Megjegyzések a fejezethez* című részben még kitérünk.

[16] Az R_ρ mátrix kondíciószáma korlátos marad.

[17] Ez azt jelenti, hogy a 10^{-8}-nál kisebb számokat 10^{-8}-nak tekintjük.

Alkalmazhatnánk iránymenti minimalizálást is, de a ∂_1 függvény tulajdonságai alapján nyilvánvaló, hogy a keresett α érték a felső korlát közelében lesz. Egy egyszerű visszaléptetéses algoritmussal kereshetjük meg közelítőleg a ∂_1 minimumát adó α értéket.

Ezekből az elemekből összeállítható MATLAB-ban olyan megvalósítás, amely esetén néhány száz feltételes, és több száz (akár ezer) változós gyakorlati feladatot is rövid idő alatt megoldhatunk, akár $\varepsilon = 10^{-8}$-os pontossággal. A tapasztalat szerint, ha a θ értéket nagyobbra választjuk, például $\theta = 0.99$, akkor a megadott méretű gyakorlati feladatoknál, ahol az A mátrix ritka, általában 30-nál nem lesz több iterációra szükségünk.

Nagyméretű lineáris programozási feladatok megoldásához jobban ki kell használnunk a gyakorlati feladatok mátrixának ritkaságát a Cholesky-faktorizációnál, ezért az ilyen feladatok esetén általában szükséges a MATLAB Cholesky-faktorizációjánál jobb faktorizációs eljárást alkalmazni.

24.3.3. Primál-duál Newton-lépéses belsőpontos algoritmus: elméleti változat

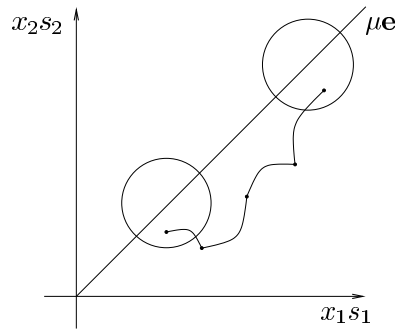
Az elméleti algoritmus változat elemzésének a szokásos menete a következő: tetszőlegesen rögzítsük le a θ és a τ paraméterek értékét, a változók számától függetlenül. Legyen a $\mu^+ = (1-\theta)\mu$ és tegyük fel, hogy az adott $\bar{\mathbf{x}} \in \mathcal{P}_+$, $(\bar{\mathbf{y}}, \bar{\mathbf{s}}) \in \mathcal{D}_+$ megoldás esetén $\partial_1(\bar{\mathbf{x}}, \bar{\mathbf{s}}; \mu^+) > \tau$ teljesül.[18] A belső ciklusban az a célunk, hogy néhány iterációval a μ^+-centrum τ-környezetébe kerüljünk (lásd a 24.5. ábrát, mely egy belső ciklus lépéseit szemlélteti). A lineáris programozás belsőpontos szakirodalmából ismert a következő meglepő eredmény: az $(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{s}}) \in \mathcal{P}_+ \times \mathcal{D}_+$ megoldásból indulva az algoritmusban az előírt utasításokat végrehajtva, legfeljebb $O(n)$ számú megoldás kiszámításával, a μ^+-centrum τ-környezetébe jutunk. Tehát a belső ciklus meghívására legfeljebb $O(n)$ alkalommal kerülhet sor, mielőtt a külső ciklusban a μ centralitási paraméter értékét csökkentenénk.

24.46. tétel. *A* Primál-duál-Newton-lépéses-belsőpontos *algoritmus esetén egy adott* $\bar{\mathbf{x}} \in \mathcal{P}_+$ *és* $(\bar{\mathbf{y}}, \bar{\mathbf{s}}) \in \mathcal{D}_+$ *megoldásból indítva az algoritmust legfeljebb*

$$\left\lceil \frac{1}{\theta} \lg \frac{\bar{\mathbf{x}}^T \bar{\mathbf{s}}}{\varepsilon} \right\rceil$$

alkalommal módosítjuk a μ *centralitási paramétert, mielőtt az algoritmus előállít egy* $\hat{\mathbf{x}} \in \mathcal{P}_+$ *és* $(\hat{\mathbf{y}}, \hat{\mathbf{s}}) \in \mathcal{D}_+$ *megoldást, amely esetén az* $\hat{\mathbf{x}}^T \hat{\mathbf{s}} \leq \varepsilon$ *teljesül.*

[18] Ha nem így van, akkor a μ paraméter értékét ismét $(1-\theta)$-szorosára csökkentjük és újra kiszámítjuk a centralitási mértékét az aktuális cél centrum paraméterével és a megoldás segítségével.



24.5. ábra. Primál-duál algoritmus iterációi az átskálázott térben, ahol $x_1 s_1 = \mu\, v_1^2$ és $x_2 s_2 = \mu\, v_2^2$.

Bizonyítás. Legyen $(\mathbf{x}^0, \mathbf{s}^0) = (\bar{\mathbf{x}}, \bar{\mathbf{s}})$. Tekintsük azt a megoldás sorozatot, amelyre a $\partial_1(\mathbf{x}^i, \mathbf{s}^i; \mu^i) \leq \tau$ teljesül, ezek azok a megoldások, amelyek előállítása után a μ centralitási paramétert csökkenti az algoritmus. Legyen ez az

$$(\mathbf{x}^0, \mathbf{s}^0),\ (\mathbf{x}^1, \mathbf{s}^1),\ \ldots,\ (\mathbf{x}^k, \mathbf{s}^k),\ \ldots$$

pontsorozat, amelynek az elemei rendre a $\mu^0, \mu^1, \ldots, \mu^k, \ldots$ paraméterű centrumokhoz tartoznak, és az azoktól mért távolságuk nem nagyobb, mint τ. Sőt a pontpárokhoz tartozó dualitásrés egyre kisebb, hiszen $(\mathbf{x}^k)^T \mathbf{s}^k = \mu^k\, n = (1-\theta)^k\, \mu^0\, n$, ahol $1-\theta < 1$, tehát ez a sorozat tart a nullához. Azaz bármely $\varepsilon > 0$ adott pontossághoz létezik k index úgy, hogy

$$(\mathbf{x}^k)^T \mathbf{s}^k = \mu^k\, n = (1-\theta)^k\, \mu^0\, n \leq \varepsilon, \qquad \text{ekkor} \qquad k\, \lg(1-\theta) + \lg(\mu_0\, n) \leq \lg \varepsilon\ ,$$

a logaritmus monoton növekedése miatt. Figyelembe véve a $-\lg(1-\theta) \geq \theta$ összefüggést, az előző egyenlőtlenségnél egy erősebbet követelünk meg, amely valamely, az előzőleg megadott k indexnél nagyobbra teljesül

$$k\, \lg(1-\theta) + \lg(\mu^0\, n) \leq -k\, \theta + \lg(\mu^0\, n) \leq \lg \varepsilon\ ,$$

tehát

$$\lg(\mu^0\, n) - \lg \varepsilon \leq k\, \theta\ ,$$

ezért

$$\frac{1}{\theta}\, \lg \frac{\mu^0\, n}{\varepsilon} \leq k\ ,$$

valamely $k \in \mathbb{N}$ esetén. Tehát az $(\hat{\mathbf{x}}, \hat{\mathbf{s}}) = (\mathbf{x}^k, \mathbf{s}^k)$ jelölést használva, teljesül $\hat{\mathbf{x}}^T \hat{\mathbf{s}} \leq \varepsilon$. ∎

Tekintettel arra, hogy a belső ciklus iteráció számának a meghatározása meghaladná a fejezet terjedelmét és a felhasznált szakmai eszköztárát, inkább

egy érdekes, speciális paraméter választás mellett bizonyítjuk be a PRIMÁL-DUÁL-NEWTON-LÉPÉSES-BELSŐPONTOS algoritmus polinomiális lépésszámának legfőbb részleteit.

Az egyszerűség kedvéért válasszuk a $\theta = 1/(2\sqrt{n})$ és $\tau = 1/\sqrt{2}$ paraméter értékeket. Ebben az esetben a külső ciklus kiértékelésére

$$2\sqrt{n}\,\lg\frac{\mu^0 n}{\varepsilon} \le k, \quad \text{azaz} \quad \left\lceil 2\sqrt{n}\,\lg\frac{\mu^0 n}{\varepsilon}\right\rceil = k$$

alkalommal kerül sor.[19] Elemzésünk fő tétele az lesz, hogy az adott paraméterek mellett a belső ciklus kiértékelésére minden alkalommal pontosan egyszer kerül sor és ekkor $\alpha = 1$, tehát teljes Newton-lépést teszünk a célul kitűzött μ-centrum irányába, és az egy lépéssel előállított megoldás a centrum τ-környezetében lesz.

Nyilvánvaló, hogy az algoritmus iterációinak számára felső becslést kapunk, ha a belső ciklus iterációinak felső korlátját – speciális esetünkben az 1-et – megszorozzuk a külső ciklus kiértékelésének a számával. Ebből egyszerűen következik, hogy a primál-duál teljes Newton-lépéses belsőpontos algoritmus $O(\sqrt{n}\,\lg\frac{n}{\varepsilon})$ iterációban előállít egy olyan $\hat{\mathbf{x}} \in \mathcal{P}_+$ és $(\hat{\mathbf{y}}, \hat{\mathbf{s}}) \in \mathcal{D}_+$ megoldást, amely esetén az $\hat{\mathbf{x}}^T\hat{\mathbf{s}} \le \varepsilon$ teljesül.

Rátérünk a primál-duál teljes Newton-lépéses belsőpontos algoritmus részletes elemzésére.

24.47. állítás. *Legyen* $\bar{\mathbf{x}} \in \mathcal{P}_+$, $(\bar{\mathbf{y}}, \bar{\mathbf{s}}) \in \mathcal{D}_+$, *és* $\mu > 0$. *Ha* $\partial = \partial_1(\bar{\mathbf{x}}, \bar{\mathbf{s}}, \mu)$, *akkor*

$$\|\mathbf{d}_x\,\mathbf{d}_s\|_\infty \le \partial^2 \quad \textit{és} \quad \|\mathbf{d}_x\,\mathbf{d}_s\| \le \partial^2\sqrt{2}\,.$$

Az előző állítás a 24.3-10 gyakorlatot felhasználva, $\mathbf{a} = \mathbf{d}_x$ és $\mathbf{b} = \mathbf{d}_s$ szereposztással, valamint figyelembe véve a $\mathbf{d}_x + \mathbf{d}_s = \mathbf{v}^{-1} - \mathbf{v}$ összefüggést, egyszerűen bizonyítható (lásd 24.3-11gyakorlat).

A következőkben megmutatjuk, hogy ha a centrális út $\partial_1(\mathbf{x}, \mathbf{s}, \mu) < 1$ környezetében vagyunk, akkor a teljes Newton-lépés megengedett, illetve az így kapott pont is ebben a környezetben lesz.

24.48. tétel. *Ha* $\partial = \partial_1(\bar{\mathbf{x}}, \bar{\mathbf{s}}; \mu) \le 1$, *akkor* $\mathbf{x}^+ \in \mathcal{P}$, $(\mathbf{y}^+, \mathbf{s}^+) \in \mathcal{D}$. *Továbbá, ha* $\partial < 1$, *akkor* $\mathbf{x}^+ > \mathbf{0}$, $\mathbf{s}^+ > \mathbf{0}$, *és*

$$\partial_1(\mathbf{x}^+, \mathbf{s}^+; \mu) \le \frac{\partial^2}{\sqrt{2\,(1-\partial^2)}}\,.$$

[19] Az általánosság korlátozása nélkül feltehetjük, hogy $\mu^0 = 1$, például a beágyazás miatt (lásd 24.1. alfejezet).

Bizonyítás. Az $\mathbf{x}^+$, és $\mathbf{s}^+$ pontosan akkor megengedett, ha $\mathbf{e}+\mathbf{d}_x\,\mathbf{d}_s \geq \mathbf{0}$, aminek elégséges feltétele $\|\mathbf{d}_x\,\mathbf{d}_s\|_\infty \leq 1$. Legyen $\mathbf{v}^+ = \sqrt{\mathbf{x}^+\,\mathbf{s}^+/\mu}$, ekkor $(\mathbf{v}^+)^2 = \mathbf{x}^+\,\mathbf{s}^+/\mu = (\mathbf{e}+\mathbf{d}_x\,\mathbf{d}_s)$. Másfelől

$$\begin{aligned} 2\,\partial^+ &= \left\|(\mathbf{v}^+)^{-1} - \mathbf{v}^+\right\| = \left\|(\mathbf{v}^+)^{-1}(\mathbf{e} - (\mathbf{v}^+)^2)\right\| = \left\|(\mathbf{v}^+)^{-1}\,(\mathbf{e} - \mathbf{e} - \mathbf{d}_x\,\mathbf{d}_s)\right\| \\ &= \left\|\frac{\mathbf{d}_x\,\mathbf{d}_s}{\sqrt{\mathbf{e}+\mathbf{d}_x\,\mathbf{d}_s}}\right\| \leq \frac{\|\mathbf{d}_x\,\mathbf{d}_s\|}{\sqrt{1-\|\mathbf{d}_x\,\mathbf{d}_s\|_\infty}} \leq \frac{\sqrt{2}\,\partial^2}{\sqrt{1-\partial^2}}\,, \end{aligned}$$

és így teljesül a

$$\partial^+ \leq \frac{\partial^2}{\sqrt{2\,(1-\partial^2)}}$$

egyenlőtlenség. ∎

Világos, hogy ha $\partial \leq 1/\sqrt{2}$, akkor a tétel szerint $\partial^+ = \partial_1(\mathbf{x}^+, \mathbf{s}^+; \mu) \leq \partial^2$, azaz ez esetben az algoritmus lokálisan kvadratikusan konvergens. Ez ad részben magyarázatot arra is, hogy miért választottuk a $\tau = 1/\sqrt{2}$ értéket a teljes Newton-lépéses algoritmus esetén.

A 24.3-14 gyakorlat eredményét felhasználva pontosabb felső korlátot tudunk adni új pontunk centralitási mértékére.

24.49. tétel. *Ha* $\partial = \partial_1(\bar{\mathbf{x}}, \bar{\mathbf{s}}, \mu) < 1$, *akkor*

$$\partial_1(\mathbf{x}^+, \mathbf{s}^+, \mu) \leq \frac{\partial^2}{\sqrt{2\,(1-\partial^4)}}\,.$$

Bizonyítás. Legyen $\partial^+ = \partial_1\,(\mathbf{x}^+, \mathbf{s}^+, \mu)$, $\mathbf{x}^+ = \bar{\mathbf{x}} + \Delta\mathbf{x}$, $\mathbf{s}^+ = \bar{\mathbf{s}} + \Delta\mathbf{s}$ és $\mathbf{v}^+ = \sqrt{\frac{\mathbf{x}^+\,\mathbf{s}^+}{\mu}} = \sqrt{\mathbf{e}+\mathbf{d}_x\,\mathbf{d}_s}$. Ekkor a $\mathbf{d}_x^T\mathbf{d}_s = 0$ összefüggést is figyelembe véve

$$\begin{aligned} \left\|\mathbf{v}^+\right\|^2 &= \mathbf{e}^T(\mathbf{v}^+\,\mathbf{v}^+) = \mathbf{e}^T(\mathbf{v}^+)^2 = \mathbf{e}^T\left(\sqrt{\mathbf{e}+\mathbf{d}_x\,\mathbf{d}_s}\right)^2 = \mathbf{e}^T(\mathbf{e}+\mathbf{d}_x\,\mathbf{d}_s) \\ &= \mathbf{e}^T\mathbf{e} + \mathbf{d}_x^T\mathbf{d}_s = n \end{aligned}$$

adódik. Itt Ling 2. lemmáját (24.3-14 gyakorlat) alkalmazva azt kapjuk, hogy

$$\begin{aligned} 4\,(\partial^+)^2 &= \left\|(\mathbf{v}^+)^{-1} - \mathbf{v}^+\right\|^2 = \left((\mathbf{v}^+)^{-1} - \mathbf{v}^+\right)^T\left((\mathbf{v}^+)^{-1} - \mathbf{v}^+\right) = \\ &= \left\|(\mathbf{v}^+)^{-1}\right\|^2 + \left\|\mathbf{v}^+\right\|^2 - 2\,n = \left\|(\mathbf{v}^+)^{-1}\right\|^2 - n = \left\|\frac{\mathbf{e}}{\sqrt{\mathbf{e}+\mathbf{d}_x\,\mathbf{d}_s}}\right\|^2 - n = \\ &= \mathbf{e}^T\left(\frac{\mathbf{e}}{\mathbf{e}+\mathbf{d}_x\,\mathbf{d}_s} - \mathbf{e}\right)\,. \end{aligned}$$

A 24.3-14 gyakorlatot a $\mathbf{u} = \mathbf{d}_x$ és $\mathbf{v} = \mathbf{d}_s$ vektorokkal alkalmazva

$$4\left(\partial^+\right)^2 \leq \frac{2\partial^4}{1-\partial^4}$$

adódik, ugyanis $\|\mathbf{d}_x + \mathbf{d}_s\| = 2\partial$ a centralitási mérték definíciója szerint, ahol $\partial < 1$. Átrendezés után pedig megkapjuk a tétel állítását. ■

A 24.48., valamint a 24.49. tételekben beláttuk, hogy a centrális út megfelelő környezetéből indulva a teljes Newton-lépés sem vezet ki a környezetből, tehát a teljes Newton-lépéses algoritmus megfelelő τ (például $\tau < 1$) esetén jól definiált lesz.

Az algoritmus lépésszámának meghatározása érdekében először vizsgáljuk meg, hogy hogyan változik a centralitási mérték a μ paraméter változtatása következtében. A következő lemma bizonyítását az Olvasóra bízzuk (lásd 24.3-16 gyakorlat).

24.50. lemma. *Legyen* $\mathbf{x} \in \mathcal{P}_+$, $(\mathbf{y}, \mathbf{s}) \in \mathcal{D}_+$ *és* $\mu > 0$, *amelyre* $\mathbf{x}^T\mathbf{s} = \mu n$. *Továbbá legyen* $\partial = \partial_1(\mathbf{x}, \mathbf{s}, \mu)$ *és* $\mu^+ = (1-\theta)\,\mu$. *Ekkor*

$$\partial_1(\mathbf{x}, \mathbf{s}, \mu^+)^2 = (1-\theta)\,\partial^2 + \frac{\theta^2 n}{4\,(1-\theta)}\,.$$

Az előző eredmény ismeretében már könnyen bizonyítható, hogy a Primál-duál-Newton-lépéses-belsőpontos algoritmus esetén a belső ciklusban pontosan egy iterációra kerül sor, ami egy teljes Newton-lépésnek felel meg.

24.51. tétel. *Legyen* $\tau = 1/\sqrt{2}$ *és* $\theta = 1/(2\sqrt{n})$, *akkor a* Primál-duál-Newton-lépéses-belsőpontos *algoritmus pontosan egy teljes Newton-lépést tesz a belső ciklusban.*

Bizonyítás. Legyen $(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{s}}) \in \mathcal{P}_+ \times \mathcal{D}_+$ olyan megoldás, amelynek a távolsága a μ-centrumától $\partial_1(\bar{\mathbf{x}}, \bar{\mathbf{s}}; \bar{\mu}) \leq \tau = 1/\sqrt{2}$. Ekkor a primál-duál Newton-lépéses belsőpontos algoritmus alapján $\mu^+ = (1-\theta)\,\bar{\mu}$ lesz az új μ-centrum paramétere (külső ciklus). Az algoritmust alkalmazva tegyünk egy teljes Newton-lépést a célul kitűzött, új centrum felé (belső ciklus). Ekkor az $(\mathbf{x}^+, \mathbf{y}^+, \mathbf{s}^+) \in \mathcal{P}_+ \times \mathcal{D}_+$ pontot kapjuk, amely megengedettségét a 24.48. tétel biztosítja. A 24.49. tétel alapján az új pont és a régi centrum távolsága a ∂_1 centralitási mérték szerint

$$\partial_1\left(\mathbf{x}^+, \mathbf{s}^+, \mu\right) \leq \frac{\partial^2}{\sqrt{2\,(1-\partial^4)}} \leq \frac{\tau^2}{\sqrt{2\,(1-\tau^4)}} = \frac{\frac{1}{2}}{\sqrt{\frac{3}{2}}} < \frac{1}{2}$$

lesz. Továbbá $(\mathbf{x}^+)^T \mathbf{s}^+ = \mu^+ n$ a 24.44. lemma alapján. A 24.50. lemmát felhasználva

$$\begin{aligned}\partial_1\left(\mathbf{x}^+,\mathbf{s}^+,\mu^+\right)^2 &= (1-\theta)\,\partial_1\left(\mathbf{x}^+,\mathbf{s}^+,\mu\right)^2 + \frac{n\,\theta^2}{4(1-\theta)} < \frac{1-\theta}{4} + \frac{n\,\theta^2}{4(1-\theta)} \\ &= \frac{1-\theta}{4} + \frac{n\,\left(\frac{1}{4n}\right)}{4\,(1-\theta)} = \frac{1-\theta}{4} + \frac{1}{16\,(1-\theta)} \\ &\le \frac{1}{4} + \frac{1}{8} = \frac{3}{8} < \frac{1}{2} = \tau^2\end{aligned}$$

adódik, figyelembe véve a $0 < \theta = 1/(2\sqrt{n}) \le 1/2$ összefüggést, amely bármely n esetén teljesül. Az iteráció után az új pont ismét a centrális út megfelelő környezetében van, tehát nincsen szükség további iterációra a belső ciklusban. ■

24.3.4. Primál-duál prediktor-korrektor belsőpontos algoritmus

Az előző részben ismertetett primál-duál algoritmust továbbfejlesztve jutunk el a ***prediktor-korrektor algoritmushoz.*** A bevezetett jelöléseket használjuk a továbbiakban is. Mint már láttuk, a $\mathbf{d}_x$ és $\mathbf{d}_s$ vektorok merőlegesek egymásra, valamint $\mathbf{d}_x$ az AD, míg $\mathbf{d}_s$ a HD^{-1} mátrixok nullterére vett vetülete a $\mathbf{v}^{-1} - \mathbf{v}$ vektornak (lásd (24.19)).

Bontsuk szét a $\mathbf{d}_x$ és $\mathbf{d}_s$ vektorokat egy ún. affin skálázási és egy centralizáló rész összegére a következőképpen:

centralizáló lépés:	$\mathbf{d}_x^c$	$= P_{AD}(\mathbf{v}^{-1})$	és	$\mathbf{d}_s^c$	$= P_{HD^{-1}}(\mathbf{v}^{-1})$,
affin skálázási lépés:	$\mathbf{d}_x^a$	$= P_{AD}(-\mathbf{v})$	és	$\mathbf{d}_s^a$	$= P_{HD^{-1}}(-\mathbf{v})$.

Azaz

$$\mathbf{d}_x = \mathbf{d}_x^c + \mathbf{d}_x^a \qquad \text{és} \qquad \mathbf{d}_s = \mathbf{d}_s^c + \mathbf{d}_s^a\ ,$$

továbbá a $\mathbf{v}^{-1}$ és $\mathbf{v}$ vektorok felbontását kapjuk a következő értelemben

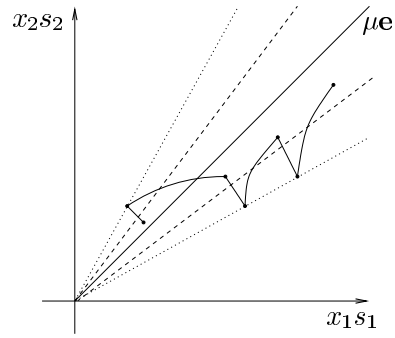
$$\mathbf{d}_x^c + \mathbf{d}_s^c = \mathbf{v}^{-1}, \tag{24.21}$$

$$\mathbf{d}_x^a + \mathbf{d}_s^a = -\mathbf{v}\ . \tag{24.22}$$

Az eredeti skálázással analóg módon definiáljuk az affin skálázási, illetve a centralizáló lépésben a Newton-irányokat:

$$\Delta^a\mathbf{x} = \sqrt{\mu}\,\mathbf{d}\,\mathbf{d}_x^a, \qquad \Delta^a\mathbf{s} = \sqrt{\mu}\,\mathbf{d}^{-1}\mathbf{d}_s^a\ ,$$

$$\Delta^c\mathbf{x} = \sqrt{\mu}\,\mathbf{d}\,\mathbf{d}_x^c, \qquad \Delta^c\mathbf{s} = \sqrt{\mu}\,\mathbf{d}^{-1}\mathbf{d}_s^c\ .$$



24.6. ábra. Prediktor-korrektor algoritmus iterációi az átskálázott térben, ahol $x_1s_1 = \mu\, v_1^2$ és $x_2s_2 = \mu\, v_2^2$.

Az affin skálázási lépésben α hosszúságú Newton-lépést teszünk meg, míg a centralizáló lépésben meglépjük a teljes Newton-lépést, így az $(\mathbf{x}, \mathbf{s})$ pontból a következő pontokba lépünk:

$$\mathbf{x}^a(\alpha) = \mathbf{x} + \alpha\Delta^a\mathbf{x}, \qquad \mathbf{s}^a(\alpha) = \mathbf{s} + \alpha\Delta^a\mathbf{s}\ ,$$

$$\mathbf{x}^c = \mathbf{x} + \Delta^c\mathbf{x}, \qquad \mathbf{s}^c = \mathbf{s} + \Delta^c\mathbf{s}\ .$$

A fenti azonosságok alapján könnyen beláthatóak az alábbi összefüggések az affin skálázási, illetve a centralizáló lépésekre:

$$\mathbf{x}\,\Delta^a\mathbf{s} + \mathbf{s}\,\Delta^a\mathbf{x} \;= -\mathbf{x}\,\mathbf{s}\ , \tag{24.23}$$

$$\mathbf{x}\,\Delta^c\mathbf{s} + \mathbf{s}\,\Delta^c\mathbf{x} \;= \mu\,\mathbf{e}\ , \tag{24.24}$$

ugyanis $\mathbf{x}\Delta^a\mathbf{s} = \sqrt{\mu}\,\mathbf{x}\,\mathbf{d}^{-1}\mathbf{d}_s^a = \mu\,\mathbf{v}\,\mathbf{d}_s^a$, valamint $\mathbf{s}\Delta^a\mathbf{x} = \sqrt{\mu}\,\mathbf{s}\,\mathbf{d}\,\mathbf{d}_x^a = \mu\,\mathbf{v}\,\mathbf{d}_x^a$. Az utóbbi két egyenlőséget összeadva, és figyelembe véve (24.22) összefüggést, a (24.23) egyenlőséget kapjuk

$$\mathbf{x}\,\Delta^a\mathbf{s} + \mathbf{s}\,\Delta^a\mathbf{x} = \mu\,\mathbf{v}\,\mathbf{d}_s^a + \mu\,\mathbf{v}\,\mathbf{d}_x^a = \mu\,\mathbf{v}(-\mathbf{v}) = -\mathbf{x}\,\mathbf{s}\ .$$

A centralizáló lépésre vonatkozó (24.24) összefüggést hasonlóan bizonyíthatjuk felhasználva a definíciókat és a (24.21) egyenlőséget (lásd 24.3-17 gyakorlat).

24.52. lemma. *Legyen $\mathbf{x}^T\mathbf{s} = n\mu$. Tegyük fel, hogy megengedett lépéseket teszünk, ekkor a α hosszúságú affin skálázási lépésben $(1-\alpha)$-szorosára csökken a dualitásrés, míg a centralizáló lépés során amennyiben megengedett, megduplázódik.*

Bizonyítás. Először vizsgáljuk meg az affin skálázási lépés esetén a dualitásrést:

$$\mathbf{x}^a(\alpha)\,\mathbf{s}^a(\alpha) = (\mathbf{x}+\alpha\Delta^a\mathbf{x})(\mathbf{s}+\alpha\Delta^a\mathbf{s}) = \mathbf{x}\,\mathbf{s}+\alpha(\mathbf{x}\,\Delta^a\mathbf{s}+\mathbf{s}\,\Delta^a\mathbf{x})+\alpha^2\Delta^a\mathbf{x}\,\Delta^a\mathbf{s}\;.$$

A (24.23) egyenlőséget felhasználva

$$\mathbf{x}^a(\alpha)\,\mathbf{s}^a(\alpha) = (1-\alpha)\,\mathbf{x}\,\mathbf{s} + \alpha^2\Delta^a\mathbf{x}\,\Delta^a\mathbf{s} \tag{24.25}$$

adódik. Mivel $\Delta^a\mathbf{x}$ és $\Delta^a\mathbf{s}$ merőlegesek egymásra (ugyanis $\mathbf{d}_x^a$ és $\mathbf{d}_s^a$ vektorok merőlegesek a 24.41. lemma bizonyításához hasonló megfontolások alapján), a következőt kapjuk:

$$\mathbf{x}^a(\alpha)^T\mathbf{s}^a(\alpha) = \mathbf{e}^T((1-\alpha)\mathbf{x}\,\mathbf{s} + \alpha^2\Delta^a\mathbf{x}\,\Delta^a\mathbf{s}) = (1-\alpha)\mathbf{x}^T\mathbf{s}\;,$$

amivel beláttuk az első állításunkat. A centralizáló lépésnél hasonlóan járunk el:

$$\mathbf{x}^c\,\mathbf{s}^c = (\mathbf{x}+\Delta^c\mathbf{x})(\mathbf{s}+\Delta^c\mathbf{s}) = \mathbf{x}\,\mathbf{s} + (\mathbf{x}\,\Delta^c\mathbf{s}+\mathbf{s}\,\Delta^c\mathbf{x}) + \Delta^c\mathbf{x}\,\Delta^c\mathbf{s}\;.$$

A (24.24) összefüggést figyelembe véve az

$$\mathbf{x}^c\,\mathbf{s}^c = \mathbf{x}\,\mathbf{s} + \mu\,\mathbf{e} + \Delta^c\mathbf{x}\,\Delta^c\mathbf{s}$$

egyenlőséget kapjuk. Az előzőekhez hasonlóan az $\Delta^c\mathbf{x}$ és $\Delta^c\mathbf{s}$ vektorok is merőlegesek egymásra, így

$$(\mathbf{x}^c)^T\mathbf{s}^c = \mathbf{e}^T(\mathbf{x}\,\mathbf{s} + \mu\,\mathbf{e} + \Delta^c\mathbf{x}\,\Delta^c\mathbf{s}) = \mathbf{x}^T\mathbf{s} + \mu\,\mathbf{e}^T\mathbf{e} = 2n\mu\;,$$

ezzel beláttuk a második állítást is. ∎

Az előző lemma világossá teszi, hogy a dualitásrés csökkenését az affin skálázási lépéssel biztosítjuk, míg a fenti módon definiált centralizáló lépés következtében a dualitásrés megduplázódna, ezzel elrontva az algoritmus konvergenciáját, ezért ehelyett egy teljes Newton-lépést teszünk meg az aktuális μ érték felé. Ennek következtében, mint már az előző részben is láttuk (24.44. lemma), a pontunk dualitásrése változatlan marad, de közelebb kerülünk a centrális úthoz. Ezen tulajdonsága miatt nevezzük ezt a lépést centralizáló lépésnek. Ennek megfelelően az affin lépést prediktor, míg a centralizáló lépést korrektor lépésnek is nevezik.

Célunk olyan megoldáspár előállítása, melynek a dualitásrése közel nulla, azaz a dualitásrést szeretnénk minél gyorsabban csökkenteni, vagyis az affin skálázási lépés hatékonysága fontos kérdése az algoritmusnak. Ezen a gondolaton alapszik a prediktor-korrektor algoritmus. Megjegyezzük, hogy ha az

affin skálázási lépés esetén a teljes Newton-lépés megengedett, akkor e lépéssel egy nulla dualitásrésű ponthoz jutunk, azaz egy optimális megoldáshoz, ám ez az esetek többségében nem következik be.

A prediktor-korrektor algoritmus esetén ahelyett, hogy az affin skálázási és a centralizáló lépésből egy Newton-lépést készítenénk, az előző primál-duál algoritmusunk Newton-lépését két részre szedjük szét. Először egy centralizáló lépést, majd egy affin skálázási lépést teszünk. Mivel a teljes affin lépés nem megengedett, egy α paraméterrel tompítjuk. Az α megfelelő megválasztásával egyrészt biztosítjuk a lépésünk megengedettségét, másrészt korlátozzuk az új pontunk túlzott eltávolodását a centrális úttól. A centralizáló lépésünk célja visszajutni a centrális út megfelelő környezetébe, ezzel biztosítva az algoritmus konvergenciáját.

Az algoritmus működését a 24.6. ábra szemlélteti, ahol a szaggatott vonal a szűkebb, míg a pontozott a tágabb környezete a centrális útnak. Mint ahogy az ábrán is látható, egy affin lépés során eltávolodunk a centrális úttól, de nem hagyjuk el a tágabb környezetet, míg a centralizáló lépés során visszajutunk a szűkebb környezetbe.

A prediktor-korrektor algoritmus bemenő adataként — hasonlóan a primál-duál Newton-lépéses belsőpontos algoritmushoz – meg kell adnunk az $\varepsilon > 0$ megkívánt számítási pontosságot, a $1 > \tau \geq 0$ eltérés paramétert, valamint egy $(\mathbf{x}^0, \mathbf{y}^0, \mathbf{s}^0)$ belső pontot, amely a τ által meghatározott környezetben helyezkedik el, azaz $(\mathbf{x}^0, \mathbf{y}^0, \mathbf{s}^0) \in \mathcal{P}_+ \times \mathcal{D}_+$, $(\mathbf{x}^0)^T\mathbf{s}^0 = \mu^0\, n$ és $\partial_1(\mathbf{x}^0, \mathbf{s}^0, \mu^0) \leq \tau$.

PREDIKTOR-KORREKTOR-BELSŐPONTOS

```
1  x = x^0
2  s = s^0
3  μ = μ^0
4  while n μ ≥ (1 − α) ε
5        μ = x^T s/n
6        számítsuk ki a Δx és Δs értékét a Newton rendszer (524. oldal
         megoldásával                          // korrektor lépés (Newton-lép
7        x = x + Δx
8        s = s + Δs
9        számítsuk ki a Δ^a x és Δ^a s értékét
                                    // prediktor lépés (affin skálázási lépé
```

medskip

10 határozzuk meg a maximális α lépéshosszt úgy, hogy
$\partial_1(\mathbf{x}(\alpha), \mathbf{s}(\alpha), \mathbf{x}(\alpha)^T\mathbf{s}(\alpha)/n) \le \tau$
$(\mathbf{x}(\alpha) = \mathbf{x} + \alpha\Delta^a\mathbf{x},\ \mathbf{s}(\alpha) = \mathbf{s} + \alpha\Delta^a\mathbf{s})$
11 $\mathbf{x} = \mathbf{x} + \alpha\Delta^a\mathbf{x}$
12 $\mathbf{s} = \mathbf{s} + \alpha\Delta^a\mathbf{s}$
13 $\mu = (1-\alpha)\,\mu$

Affin skálázási lépés

A továbbiakban megvizsgáljuk, hogy egy affin skálázási lépés során hogyan változik a centralitási mérték a lépéshossz függvényében. Legyen $(\mathbf{x}, \mathbf{s})$ pozitív megengedett primál-duál pár. Legyen továbbá $\mu > 0$, melyre $\mathbf{x}^T\mathbf{s} = n\mu$. Elemzésünk során az előző részben definiált

$$\partial = \partial(\mathbf{x}, \mathbf{s}, \mu) = \frac{1}{2}\|\mathbf{v}^{-1} - \mathbf{v}\|, \text{ ahol } \quad \mathbf{v} = \sqrt{\frac{\mathbf{x}\,\mathbf{s}}{\mu}}\,,$$

centralitási mértéket használjuk. A számolások során szükségünk lesz a $\mathbf{v}$ vektor becslésére. A következő lemmában alsó és felső korlátot adunk a koordinátákra, a bizonyítást az Olvasóra bízzuk (lásd 24.3-18 gyakorlat).

24.53. lemma. *Legyen* $\rho(\partial) = \partial + \sqrt{1+\partial^2}$. *Ekkor*

$$\frac{1}{\rho(\partial)} \le v_i \le \rho(\partial), \qquad 1 \le i \le n\,.$$

Az $\mathbf{v}$ vektorra adott korlátok ismeretében az α lépéshossz megengedettségére a következő elégséges feltételt kapjuk.

24.54. lemma. *Legyen* $(\mathbf{x}, \mathbf{s})$ *megengedett pont, melyre* $\mathbf{x}^T\mathbf{s} = n\mu$ *és* $\partial = \partial(\mathbf{x}, \mathbf{s}, \mu) < \tau$. *Jelölje továbbá* $\mathbf{x}^+ = \mathbf{x} + \alpha\Delta^a\mathbf{x}$ *és* $\mathbf{s}^+ = \mathbf{s} + \alpha\Delta^a\mathbf{s}$ *az affin lépés után kapott pontot. Ekkor* $\partial^+ = \partial(\mathbf{x}^+, \mathbf{s}^+, (1-\alpha)\mu) \le \tau$, *ha az* α *paraméterre teljesül az alábbi egyenlőtlenség*

$$\frac{\alpha^2 n}{1-\alpha} \le 2\sqrt{2}\left(\tau\sqrt{\frac{4}{\rho(\partial)^2} + 4\partial\rho(\partial)\sqrt{2} + 2\tau^2} - 2\partial\rho(\partial) - \tau^2\sqrt{2}\right)\,.$$

Továbbá rögzített τ *mellett a fenti kifejezés jobb oldala* ∂ *monoton csökkenő függvénye.*

Bizonyítás. A (24.25) egyenlőséget felhasználva a következőt kapjuk

$$\mathbf{x}^+\mathbf{s}^+ = (1-\alpha)\mathbf{x}\,\mathbf{s} + \alpha^2\Delta^a\mathbf{x}\,\Delta^a\mathbf{s} = \mu\left((1-\alpha)\mathbf{v}^2 + \alpha^2\mathbf{d}_x^a\,\mathbf{d}_s^a\right)\,.$$

Vezessük be az alábbi jelölést

$$\mathbf{v}^+ = \sqrt{\frac{\mathbf{x}^+\mathbf{s}^+}{(1-\alpha)\mu}}, \text{ ekkor } \quad (\mathbf{v}^+)^2 = \mathbf{v}^2 + \frac{\alpha^2 \mathbf{d}_x^a\, \mathbf{d}_s^a}{1-\alpha} \ .$$

Az új pontunk centralitási mértékére

$$\partial^+ = \frac{1}{2}\left\|(\mathbf{v}^+)^{-1}\left(\mathbf{e}-\mathbf{v}^+\right)\right\| \le \frac{1}{2}\left\|(\mathbf{v}^+)^{-1}\right\|_\infty \left\|\mathbf{e}-(\mathbf{v}^+)^2\right\|$$

áll fenn. A továbbiakban külön-külön felső korlátot adunk a jobb oldali szorzat két tényezőjére, először a második kifejezést vizsgáljuk.

$$\begin{aligned}\left\|\mathbf{e}-(\mathbf{v}^+)^2\right\| &= \left\|\mathbf{e}-\mathbf{v}^2-\frac{\alpha^2\mathbf{d}_x^a\,\mathbf{d}_s^a}{1-\alpha}\right\| \le \left\|\mathbf{e}-\mathbf{v}^2\right\| + \frac{\alpha^2}{1-\alpha}\left\|\mathbf{d}_x^a\,\mathbf{d}_s^a\right\| \\ &\le \left\|\mathbf{e}-\mathbf{v}^2\right\| + \frac{\alpha^2 n}{2\sqrt{2}\,(1-\alpha)} \ .\end{aligned}$$

Az utolsó egyenlőtlenség a 24.47. következmény miatt igaz, figyelembe véve az $\|\mathbf{d}_x^a + \mathbf{d}_s^a\|^2 = \|\mathbf{v}\|^2 = n$ egyenlőséget. A 24.53. lemmabeli eredmények segítségével kapjuk az alábbiakat

$$\left\|\mathbf{e}-\mathbf{v}^2\right\| = \left\|\mathbf{v}\,\frac{\mathbf{e}-\mathbf{v}^2}{\mathbf{v}}\right\| \le \|\mathbf{v}\|_\infty \left\|\frac{\mathbf{e}-\mathbf{v}^2}{\mathbf{v}}\right\| \le 2\partial\rho(\partial) \ .$$

Tehát a második tényezőre a következő egyenlőtlenség teljesül:

$$\left\|\mathbf{e}-(\mathbf{v}^+)^2\right\| \le 2\partial\rho(\partial) + \frac{\alpha^2 n}{2\sqrt{2}\,(1-\alpha)} \ .$$

Az első tényező becsléséhez ismét használjuk fel a 24.53. lemmát és a 24.47. következményt.

$$(v_i^+)^2 \ge v_i^2 - \frac{\alpha^2}{1-\alpha}\left\|\mathbf{d}_x^a\,\mathbf{d}_s^a\right\|_\infty \ge \frac{1}{\rho(\partial)^2} - \frac{\alpha^2 n}{4\,(1-\alpha)} \ ,$$

azaz

$$\left\|(\mathbf{v}^+)^{-1}\right\|_\infty \le \frac{1}{\sqrt{\frac{1}{\rho(\partial)^2} - \frac{\alpha^2 n}{4\,(1-\alpha)}}} \ .$$

A kapott felső korlátokat behelyettesítve a következőre jutunk

$$\partial^+ \le \frac{2\partial\rho(\partial) + \frac{\alpha^2 n}{2\sqrt{2}\,(1-\alpha)}}{2\sqrt{\frac{1}{\rho(\partial)^2} - \frac{\alpha^2 n}{4\,(1-\alpha)}}} \ .$$

A fentiek alapján a $\partial^+ \leq \tau$ fennállásának elégséges feltétele

$$\left(2\partial\rho(\partial) + \frac{\alpha^2 n}{2\sqrt{2}\,(1-\alpha)}\right)^2 \leq \frac{4\,\tau^2}{\rho(\partial)^2} - \frac{\alpha^2 n \tau^2}{1-\alpha}\,, \tag{24.26}$$

vigyük át a jobb oldal második tagját a bal oldalra, majd adjunk hozzá $4\sqrt{2}\tau^2\partial\rho(\partial)$-t mindkét oldalhoz. Átrendezések után

$$\left(2\partial\rho(\partial) + \frac{\alpha^2 n}{2\sqrt{2}\,(1-\alpha)}\right)^2 + 2\,\tau^2\sqrt{2}\left(2\partial\rho(\partial) + \frac{\alpha^2 n}{2\sqrt{2}\,(1-\alpha)}\right) \leq \frac{4\,\tau^2}{\rho(\partial)^2} + 4\,\tau^2\partial\rho(\partial)\sqrt{2}\,,$$

$$\left(2\partial\rho(\partial) + \frac{\alpha^2 n}{2\sqrt{2}\,(1-\alpha)} + \tau^2\sqrt{2}\right)^2 \leq \frac{4\,\tau^2}{\rho(\partial)^2} 4\,\tau^2\partial\rho(\partial)\sqrt{2} + 2\,\tau^4\,.$$

Mindkét oldalból gyököt vonva

$$2\partial\rho(\partial) + \frac{\alpha^2 n}{2\sqrt{2}(1-\alpha)} + \tau^2\sqrt{2} \leq \tau\sqrt{\frac{4}{\rho(\partial)^2} + 4\partial\rho(\partial)\sqrt{2} + 2\,\tau^2}\,,$$

és α paraméterre rendezve

$$\frac{\alpha^2 n}{2\sqrt{2}\,(1-\alpha)} \leq \tau\sqrt{\frac{4}{\rho(\partial)^2} + 4\partial\rho(\partial)\sqrt{2} + 2\,\tau^2} - 2\partial\rho(\partial) - \tau^2\sqrt{2}\,,$$

ami a lemma első állítása. A második állítás bizonyításához vegyük észre, hogy a lemmabeli egyenlőtlenség ekvivalens a (24.26) egyenlőtlenséggel, így elegendő az utóbbit vizsgálni. Könnyen belátható, hogy a (24.26) egyenlőtlenség bal oldali kifejezése mind α, mind ∂ változóban monoton nő, míg a jobb oldali kifejezés mindkettő szerint monoton fogy. Azaz ha α kielégíti az egyenlőtlenséget valamely ∂ érték mellett, akkor ugyanezen α kielégíti kisebb ∂ értékre is. Mivel az előbb említett két egyenlőtlenség ekvivalens, ugyanez igaz a lemmabeli egyenlőtlenségre is, ezzel a lemma második állítását is beláttuk. ■

Megmutatható, hogy ha $\tau = 1/2$ és $\alpha = 1/(2\sqrt{n})$, akkor minden iteráció után olyan $(\mathbf{x}, \mathbf{s})$ pontba érkezünk, melyre $\partial(\mathbf{x}, \mathbf{s}, \mu) \leq \tau$, azaz az algoritmus jól definiált (lásd 24.3-19 gyakorlat).

Ezek után megfogalmazhatjuk fő tételünket, mely a Prediktor-korrektor-belsőpontos-algoritmus lépésszámát határozza meg. A bizonyítás a Primál-duál-Newton-lépéses-belsőpontos algoritmus lépésszám bizonyításához (24.51. tétel) teljesen hasonlóan megy.

24.55. tétel. *Legyen* $\tau = 1/2$ *és* $\alpha = 1/(2\sqrt{n})$, *ekkor a* Prediktor-korrektor-belsőpontos *algoritmus legfeljebb*

$$\left\lceil 2\sqrt{n} \lg \frac{n\mu^0}{\varepsilon} \right\rceil$$

lépésben állít elő egy olyan $\mathbf{x} \in \mathcal{P}_+$, $(\mathbf{y}, \mathbf{s}) \in \mathcal{D}_+$ *primál-duál pontpárt, amelyre* $\mathbf{x}^T\mathbf{s} \leq \varepsilon$.

A Prediktor-korrektor-belsőpontos algoritmus lépésszámát tovább csökkenthetjük, ha a lépéshossz megválasztási stratégiáját megváltoztatjuk. A fenti algoritmusban az α értéke egy fix szám. Ezzel szemben, ha az algoritmus során a pontunktól függően változtatjuk a α értékét, azaz a dualitásrés csökkentése mértékét, akkor egy hatékonyabb, gyorsabb algoritmust kapunk.

Először α megengedett értékére adunk egy megfelelőbb korlátot a 24.54. lemma eredményének a javításával. A bizonyítás is az említett lemmához hasonlóan történik, így az Olvasóra bízzuk (lásd 24.3-20 gyakorlat).

24.56. lemma. *Legyen* $(\mathbf{x}, \mathbf{s})$ *megengedett pont, melyre* $\mathbf{x}^T\mathbf{s} = n\mu$ *és* $\partial = \partial(\mathbf{x}, \mathbf{s}, \mu) < \tau$. *Jelölje továbbá* $\mathbf{x}^+ = \mathbf{x} + \alpha\Delta^a\mathbf{x}$ *és* $\mathbf{s}^+ = \mathbf{s} + \alpha\Delta^a\mathbf{s}$ *az affin lépés után kapott pontot. Ekkor* $\partial^+ = \partial(\mathbf{x}^+, \mathbf{s}^+, (1-\alpha)\mu) \leq \tau$, *ha az* α *lépéshosszra teljesül az alábbi egyenlőtlenség*

$$\frac{\alpha^2}{1-\alpha} \| \mathbf{d}_x^a \mathbf{d}_s^a \| \leq 2\tau \left(\sqrt{\frac{1}{\rho(\partial)^2} + 2\partial\rho(\partial) + \tau^2} - \tau \right) - 2\partial\rho(\partial) \; .$$

Az adaptív algoritmus esetén is létezik megfelelő τ, illetve α értékek, melyekre az eljárás jól definiált lesz, mégpedig $\tau = 1/3$ és $\alpha = 2/(1 + \sqrt{1 + 13\,\|\mathbf{d}_x^a \mathbf{d}_s^a\|})$ választás eleget tesz elvárásainknak (lásd 24.3-21 gyakorlat).

Kvadratikus konvergencia

Könnyen látható, hogy a konvergencia rendje α paraméter választásától függ. A kvadratikus konvergencia feltétele

$$(1-\alpha)\mu = O(\mu^2), \qquad \text{azaz} \qquad 1 - \alpha = O(\mu) \; .$$

Megmutatjuk, hogy az előző adaptív algoritmus estén megadott α érték teljesíti ezt a feltételt, amikor a μ elegendően kicsi, azaz a fent tárgyalt adaptív prediktor-korrektor algoritmus konvergenciája kvadratikus. Ehhez először az $1-\alpha$ kifejezésre adunk felső becslést, melynek bizonyítását az Olvasóra bízzuk (lásd 24.3-22 gyakorlat).

Sorszm	Vektor	$\mathcal{B}$	$\mathcal{N}$
1	$\mathbf{x}$	$\Theta(1)$	$\Theta(\mu)$
2	$\mathbf{s}$	$\Theta(\mu)$	$\Theta(1)$
3	$\mathbf{v}$	$\Theta(1)$	$\Theta(1)$
4	$\mathbf{d}$	$\Theta(1/\sqrt{\mu})$	$\Theta(\sqrt{\mu})$
5	$\mathbf{d}_x^a$	$O(\mu)$	$O(1)$
6	$\mathbf{d}_s^a$	$O(1)$	$O(\mu)$
7	$\Delta^a\mathbf{x}$	$O(\mu)$	$O(\mu)$
8	$\Delta^a\mathbf{s}$	$O(\mu)$	$O(\mu)$

24.1. táblázat. A változók mérete, amikor μ elegendően kicsi.

24.57. lemma. *A lépéshossz* $\alpha = 2/(1+\sqrt{1+13\,\|\mathbf{d}_x^a\mathbf{d}_s^a\|})$ *választása esetén*

$$1-\alpha \leq \frac{13}{4}\,\|\mathbf{d}_x^a\,\mathbf{d}_s^a\| \ .$$

Azaz az adaptív prediktor-korrektor algoritmus konvergenciája kvadratikus, ha $\|\mathbf{d}_x^a\mathbf{d}_s^a\| = O(\mu)$ *teljesül.*

A 24.1 táblázatban, amikor μ elegendően kicsi, összefoglaljuk a bevezetett vektorok koordinátáinak nagyságrendjét aszerint, hogy az indexük a $\mathcal{B}$ vagy az $\mathcal{N}$ halmazba esik. (A bizonyítástól technikai jellege miatt eltekintünk.)

Ezek után megfogalmazhatjuk alfejezetünk fő tételét

24.58. tétel. *Az adaptív prediktor-korrektor algoritmus kvadratikusan konvergens.*

Bizonyítás. Az előzőekben belátottak alapján $\mathbf{d}_x^a\,\mathbf{d}_s^a = O(\mu)$, ez pedig a 24.57. lemma szerint pontosan azt jelenti, hogy az algoritmus kvadratikusan konvergens. ■

24.3.5. Önreguláris függvényen alapuló belső pontos algoritmus

A következőkben ismertetett elmélet kiindulópontja az előző rész bevezetőjében említett ellentmondás, miszerint a rövid lépéses algoritmusok $O(\sqrt{n}\lg\frac{n}{\varepsilon})$ bizonyított lépésszáma jobb, mint a hosszú lépéses algoritmusok esetén ismert $O(n\lg\frac{n}{\varepsilon})$, ezzel szemben a gyakorlatban a hosszú lépéses módszerek hatékonyabbnak bizonyultak. Peng, Roos és Terlaky ennek az ellentmondásnak a feloldása céljából a hosszú lépéses eljárásokat új centralitási mérték bevezetése mellett vizsgálta meg. Az új mértékeket az önreguláris függvények[20] segítségével definiálták. A megváltoztatott centralitási

[20] Az önreguláris függvény elnevezés a függvénycsalád azon tulajdonságából származik, hogy – mint látni fogjuk – az első és második deriváltak egymást korlátozzák, azaz a függvény egyféleképpen

mértéknek megfelelően a Newton-rendszert is módosították. Az így kapott új algoritmus tekinthető az előző részben bevezetett ∂_1 centralitási mérték, illetve a hozzá tartozó Newton-rendszer általánosításának. Ebben a részben az önreguláris függvények tulajdonságaira épülő belsőpontos algoritmusok egy speciális esetét ismertetjük Peng, Roos és Terlaky egyik cikke alapján, melyben a módosított belsőpontos algoritmus lépésszáma $O(\sqrt{n}\lg n \lg(\frac{n}{\varepsilon}))$, amely a rövid lépéses algoritmusok lépésszámát igen jól megközelíti.

24.59. definíció. *Egy $\psi : (0,\infty) \to \mathbb{R}$ kétszer folytonosan deriválható függvényt* ***önreguláris függvénynek*** *nevezünk, ha teljesíti a következő feltételeket:*

1. *$\psi(t)$ szigorúan konvex függvény és a globális minimumpontjában, $t = 1$ pontban eltűnik, azaz $\psi(1) = \psi'(1) = 0$. Továbbá léteznek $\nu_2 \geq \nu_1 > 0$ és $p \geq 1$, $q \geq 1$ konstansok úgy, hogy*

$$\nu_1(t^{p-1} + t^{-1-q}) \leq \psi''(t) \leq \nu_2(t^{p-1} + t^{-1-q}) \ , \quad \forall t \in (0,\infty) \ .$$

2. *Bármely $t_1, t_2 > 0$ esetén*

$$\psi(t_1^r t_2^{1-r}) \leq r\psi(t_1) + (1-r)\psi(t_2) \ , \quad \forall r \in [0,1] \ .$$

Ha a ψ önreguláris függvény, akkor a q paramétert a ***büntetés mértékének*** (a szám nullához tartását büntetjük), míg a p paramétert a ***növekedés mértékének*** nevezzük.

Az önreguláris függvények két speciális családját emelnénk ki. Az első családot a

$$\Upsilon_{p,q}(t) = \frac{t^{p+1}-1}{p(p+1)} + \frac{t^{1-q}-1}{q(q-1)} + \frac{p-q}{pq}(t-1), \quad \text{ahol } p \geq 1,\ q > 1 \ ,$$

függvény definiálja. Ekkor a definícióban szereplő konstansok $\nu_1 = \nu_2 = 1$. A második család a

$$\Gamma_{p,q}(t) = \frac{t^{p+1}-1}{p+1} + \frac{t^{1-q}-1}{q-1} \ , \quad \text{ahol } p \geq 1\, q > 1 \ ,$$

függvény által definiált, és a konstansok értéke $\nu_1 = 1$, illetve $\nu_2 = q$.
Ezen függvények segítségével definiáljuk az eljárás során használt távolságfüggvényt

$$\Psi(\mathbf{v}) = \sum_{i=1}^{n} \psi(v_i)$$

---

regularizálja önmagát.

egyenlőséggel.

Ebben a részben a második családba tartozó

$$\psi(t) = \Gamma_{1,q}(t) = \frac{t^2-1}{2} + \frac{t^{1-q}-1}{q-1}\,, \quad q > 1$$

önreguláris függvénycsaládot használjuk. Ekkor a távolságfüggvény

$$\Psi_q(\mathbf{x},\mathbf{s},\mu) = \Psi_q(\mathbf{v}) = \frac{\mathbf{e}^T\mathbf{v}^2 - n}{2} + \frac{\mathbf{e}^T\mathbf{v}^{1-q} - n}{q-1}\,.$$

A Ψ függvény első tagja a $\mathbf{v}$ vektor koordinátáinak növekedését, míg a második tag a nullához tartásukat bünteti. Jelölje a második tagot

$$\Psi_0(\mathbf{v}) = \sum_{i=1}^{n} \psi_0(v_i), \text{ ahol} \qquad \psi_0(t) = \frac{t^{1-q}-1}{q-1}\,.$$

Ezekkel a jelölésekkel a távolságfüggvényünk ***magfüggvénye*** a következő alakban írható

$$\psi(t) = \frac{t^2-1}{2} + \psi_0(t)\,. \tag{24.27}$$

Az előző részhez hasonlóan definiáljuk a Newton-rendszert, és annak átskálázását. A Newton-irányt megváltoztatjuk az önreguláris távolságfüggvény segítségével a következőképpen:

$$\begin{array}{rcrcl} & AD\,\mathbf{d}_x & & = & \mathbf{0}\,, \\ (AD)^T\mathbf{d}_y \;+ & & \mathbf{d}_s & = & \mathbf{0}\,, \\ & \mathbf{d}_x \;+ & \mathbf{d}_s & = & -\nabla\Psi(\mathbf{v}) = \mathbf{v}^{-q} - \mathbf{v}\,. \end{array}$$

Az előző belsőpontos algoritmusokhoz hasonlóan az Önreguláris-primál-duál-belsőpontos algoritmus bemeneti adataként meg kell adnunk az $\varepsilon > 0$ számítási pontosságot, a τ eltérési paramétert $(0 \le \tau < 1)$, a θ csökkentési paramétert $(0 < \theta < 1)$, valamint egy $(\mathbf{x}^0, \mathbf{y}^0, \mathbf{s}^0) \in \mathcal{P}_+ \times \mathcal{D}_+$ kezdeti megoldást, amelyre $(\mathbf{x}^0)^T\mathbf{s}^0 = \mu^0\, n$.

ÖNREGULÁRIS-PRIMÁL-DUÁL-BELSŐPONTOS

```
1  x = x^0
2  s = s^0
3  μ = μ^0
4  while n μ ≥ ε
5        μ ← (1 − θ)μ
6        while Ψ(v) ≤ (τ − 1)n/2
7              számítsuk ki a Δx és Δs értékét
8              α minimalizálja Ψ-t
9              x = x + αΔx
10             s ← s + αΔs
```

Az új távolságfüggvényünknek és az új Newton-rendszernek megfelelően definiálja az elemzésnél használt centralitási mértéket

$$\sigma(\mathbf{v}) = \|\nabla\Psi(\mathbf{v})\| = \left\|\mathbf{v}^{-q} - \mathbf{v}\right\| = \|\mathbf{d}_x + \mathbf{d}_s\| = \|(\mathbf{d}_x, \mathbf{d}_s)\| \tag{24.28}$$

kifejezéssel, ahol az utolsó egyenlőség $\mathbf{d}_x$ és $\mathbf{d}_s$ vektorok merőlegessége miatt igaz (ezt az előző részben már beláttuk).

Korlátok a v vektorra és a lépéshosszra

Első lépésként alsó és felső korlátot adunk a $\mathbf{v}$ vektor koordinátáira, melynek bizonyítását az Olvasóra bízzuk (lásd **??** gyakorlat).

24.60. lemma. *Legyen $\sigma = \sigma(\mathbf{v})$. Ekkor*

$$v_{\min} \geq (1+\sigma)^{-\frac{1}{q}}, \qquad v_{\max} \leq 1 + \sigma\ .$$

Vezessük be a következő jelöléseket:

$$\Delta_x = \frac{\Delta\mathbf{x}}{\mathbf{x}} = \frac{\mathbf{d}_x}{\mathbf{v}}, \qquad \Delta_s = \frac{\Delta\mathbf{s}}{\mathbf{s}} = \frac{\mathbf{d}_s}{\mathbf{v}}\ . \tag{24.29}$$

Továbbá jelölje az új pontunkat az α lépéshosszal tompított Newton-lépés után $(\mathbf{x}^+, \mathbf{s}^+)$, azaz $\mathbf{x}^+ = \mathbf{x} + \alpha\,\Delta\mathbf{x}$ és $\mathbf{s}^+ = \mathbf{s} + \alpha\,\Delta\mathbf{s}$. Ekkor a következőt írhatjuk:

$$\mathbf{x}^+ = \mathbf{x}\,(\mathbf{e} + \alpha\,\Delta_x), \qquad \mathbf{s}^+ = \mathbf{s}\,(\mathbf{e} + \alpha\,\Delta_s)\ .$$

Könnyen látható, hogy α pontosan akkor megengedett lépéshossz, ha $\mathbf{e} + \alpha\,\Delta_x \geq 0$ és $\mathbf{e} + \alpha\,\Delta_s \geq 0$ teljesül, így a legnagyobb megengedett lépéshosszra fennáll a következő:

$$1 \leq \alpha_{\max}\,\|(\Delta_x, \Delta_s)\|_\infty \leq \alpha_{\max}\,\|(\Delta_x, \Delta_s)\|\ . \tag{24.30}$$

Ennek segítségével alsó becslést adunk a legnagyobb megengedett lépéshosszra.

24.61. lemma. *Legyen (Δ_x, Δ_s) a (24.29) formulák szerint definiált vektor, ekkor*

$$\|(\Delta_x, \Delta_s)\| \leq \sigma(1+\sigma)^{\frac{1}{q}}$$

teljesül. Valamint az $\alpha_{\max}$ legnagyobb megengedett lépéshosszra fennáll az

$$\alpha_{\max} \geq \frac{1}{\sigma(1+\sigma)^{\frac{1}{q}}}$$

egyenlőtlenség.

Bizonyítás. A 24.60. lemma és a σ mérték (24.28) definíciója alapján

$$\|(\Delta_x, \Delta_s)\| = \left\| \left(\frac{\mathbf{d}_x}{\mathbf{v}}, \frac{\mathbf{d}_s}{\mathbf{v}} \right) \right\| \leq \frac{\|(\mathbf{d}_x, \mathbf{d}_s)\|}{v_{\min}} = \frac{\sigma}{v_{\min}} \leq \sigma(1+\sigma)^{\frac{1}{q}} .$$

A (24.30) egyenlőtlenséget átrendezve

$$\alpha_{\max} \geq \frac{1}{\|(\Delta_x, \Delta_s)\|}$$

alsó becslést kapjuk, amiből az előző egyenlőtlenség figyelembevételével következik a lemma második állítása. ∎

Vizsgáljuk meg a távolságfüggvény csökkenési mértékét az α lépéshossz függvényében. Az elemzés során megadunk egy α^* megengedett lépéshosszt, majd alsó korlátot adunk a dualitásrés új értékére α^* függvényében. Egy Newton-lépés megtétele után jelölje $\mathbf{v}$ vektor új értékét $\mathbf{v}^+ = \sqrt{\mathbf{x}^+\mathbf{s}^+/\mu}$. Ekkor

$$\begin{aligned}(\mathbf{v}^+)^2 &= \frac{(\mathbf{x}+\alpha\Delta\mathbf{x})(\mathbf{s}+\alpha\Delta\mathbf{s})}{\mu} = \frac{1}{\mu}\frac{\mathbf{x}}{\mathbf{v}}\left(\mathbf{v}+\alpha\frac{\mathbf{v}\,\Delta\mathbf{x}}{\mathbf{x}}\right)\frac{\mathbf{s}}{\mathbf{v}}\left(\mathbf{v}+\alpha\frac{\mathbf{v}\,\Delta\mathbf{s}}{\mathbf{s}}\right) \\ &= (\mathbf{v}+\alpha\,\mathbf{d}_x)(\mathbf{v}+\alpha\,\mathbf{d}_s) .\end{aligned}$$

Figyelembe véve a $\mathbf{d}_x$ és $\mathbf{d}_s$ vektorok ortogonalitását

$$\mathbf{e}^T(\mathbf{v}^+)^2 = \mathbf{e}^T\mathbf{v}^2 + \alpha\,\mathbf{v}^T(\mathbf{d}_x+\mathbf{d}_s) = \mathbf{e}^T\mathbf{v}^2 + \alpha\,\mathbf{v}^T(\mathbf{v}^{-q}-\mathbf{v}) .$$

A távolságfüggvény értéke az új pontban

$$\begin{aligned}\Psi(\mathbf{x}^+,\mathbf{s}^+,\mu) &= \Psi(\mathbf{v}^+) = \frac{\mathbf{e}^T(\mathbf{v}^+)^2 - n}{2} + \Psi_0(\mathbf{v}^+) \\ &= \frac{\mathbf{e}^T\mathbf{v}^2 + \alpha\,\mathbf{v}^T(\mathbf{v}^{-q}-\mathbf{v}) - n}{2} + \Psi_0\left(\sqrt{(\mathbf{v}+\alpha\,\mathbf{d}_x)(\mathbf{v}+\alpha\,\mathbf{d}_s)}\right)\end{aligned} \quad (24.31)$$

A következő lemma a ψ_0 függvény egy fontos tulajdonságára mutat rá.

24.62. lemma. *Legyen $t_1 > 0$ és $t_2 > 0$. Ekkor*

$$\psi_0\left(\sqrt{t_1 t_2}\right) \leq \frac{1}{2}\left(\psi(t_1) + \psi_0(t_2)\right) \ .$$

Bizonyítás. Könnyen belátható, hogy az állítás pontosan akkor igaz, ha a $\psi_0(e^z)$ mint z függvénye konvex, ami akkor és csak akkor áll fenn, ha $\psi_0'(t) + t\,\psi_0''(t) \geq 0$ teljesül minden $t \geq 0$ esetén. Mivel

$$\psi_0'(t) = -t^{-q} \ , \qquad \psi_0''(t) = q\,t^{-1-q} \ ,$$

így $\psi_0'(t) + t\,\psi_0''(t) = (q-1)t^{-q} > 0$, ezzel a lemmát beláttuk. ∎

Folytassuk a távolságfüggvény új pontbeli értékének vizsgálatát a lemma eredményét felhasználva

$$\Psi_0(\mathbf{v}^+) = \Psi_0\left(\sqrt{(\mathbf{v}+\alpha\,\mathbf{d}_x)(\mathbf{v}+\alpha\,\mathbf{d}_s)}\right) \leq \frac{1}{2}\sum_{i=1}^{n}\left(\psi_0(v_i+\alpha\,d_{xi})+\psi_0(v_i+\alpha\,d_{si})\right) .$$

A fenti becslést a (24.31) egyenlőségbe helyettesítve

$$\Psi(\mathbf{x}^+,\mathbf{s}^+,\mu) \leq \frac{\mathbf{e}^T\mathbf{v}^2 + \alpha\,\mathbf{v}^T(\mathbf{v}^{-q}-\mathbf{v}) - n}{2} + \frac{1}{2}\sum_{i=1}^{n}\left(\psi_0(v_i+\alpha\,d_{xi})+\psi_0(v_i+\alpha\,d_{si})\right)$$

adódik. Legyen $f(\alpha)$ a Ψ függvény megváltozása egy lépés során a lépéshossz függvényében, azaz

$$f(\alpha) = \Psi(\mathbf{x}^+,\mathbf{s}^+,\mu) - \Psi(\mathbf{x},\mathbf{s},\mu) \leq f_1(\alpha) \ ,$$

ahol

$$\begin{aligned} f_1(\alpha) = \ & -\Psi(\mathbf{x},\mathbf{s},\mu) + \tfrac{\mathbf{e}^T\mathbf{v}^2+\alpha\,\mathbf{v}^T(\mathbf{v}^{-q}-\mathbf{v})-n}{2} + \\ & + \tfrac{1}{2}\textstyle\sum_{i=1}^{n}\left(\psi_0(v_i+\alpha d_{xi}) + \psi_0(v_i+\alpha d_{si})\right) . \end{aligned}$$

Vegyük észre, hogy $f(0) = f_1(0) = 0$. Számoljuk ki az f_1 függvény α lépéshossz szerinti első, illetve második deriváltját.

$$f_1'(\alpha) = \frac{1}{2}\mathbf{v}^T(\mathbf{v}^{-q}-\mathbf{v}) + \frac{1}{2}\sum_{i=1}^{n}\left(\psi_0'(v_i+\alpha d_{xi})d_{xi} + \psi_0'(v_i+\alpha d_{si})d_{si}\right) .$$

A Newton-rendszer harmadik egyenletét, valamint σ (24.28) definícióját használva

$$\begin{aligned} f_1'(0) \ &= \ \frac{1}{2}\mathbf{v}^T(\mathbf{v}^{-q}-\mathbf{v}) - \frac{1}{2}\sum_{i=1}^{n} v_i^{-q}(d_{xi}+d_{si}) = \frac{1}{2}(\mathbf{d}_x+\mathbf{d}_s)^T(\mathbf{v}-\mathbf{v}^{-q}) \\ &= \ -\frac{1}{2}\sigma^2 \end{aligned} \qquad (24.32)$$

adódik. Továbbá tekintetbe véve, hogy a $\psi''(t) = qt^{-1-q}$ függvény monoton csökken az

$$\begin{aligned} f_1''(\alpha) &= \frac{1}{2}\sum_{i=1}^{n}\left(\psi_0''(v_i+\alpha d_{xi})d_{xi}^2 + \psi_0''(v_i+\alpha d_{si})d_{si}^2\right) \\ &= \frac{1}{2}\sum_{i=1}^{n}\left(\left(q(v_i+\alpha d_{xi})^{-q-1}\right)d_{xi}^2 + \left(q(v_i+\alpha d_{si})^{-1-q}\right)d_{si}^2\right) \\ &\leq \frac{q}{2}\sum_{i=1}^{n}(v_{\min}-\alpha\sigma)^{-q-1}(d_{xi}^2+d_{si}^2) \end{aligned}$$

egyenlőtlenség adódik, ahol az utolsó egyenlőtlenségnél a következőket használtuk fel

$$v_i + \alpha d_{xi} \geq v_{\min} - \alpha\,\|\mathbf{d}_x\| \geq v_{\min} - \alpha\sigma,$$
$$v_i + \alpha d_{si} \geq v_{\min} - \alpha\,\|\mathbf{d}_s\| \geq v_{\min} - \alpha\sigma\ .$$

A (24.28) azonosságot figyelembe véve

$$f_1''(\alpha) \leq h(\alpha) = \frac{1}{2}q\sigma^2(v_{\min}-\alpha\sigma)^{-1-q} \tag{24.33}$$

egyenlőtlenség adódik. Minden olyan α lépéshosszra, melyre $v_{\min} - \alpha\sigma \geq 0$ teljesül, integrálással az

$$f_1'(\alpha) = f_1'(0) + \int_0^\alpha f_1''(\xi)d\xi \leq f_1'(0) + \int_0^\alpha h(\xi)d\xi$$

egyenlőtlenség adódik. Még egyszer integrálva az egyenlőtlenséget, valamint felhasználva az $f(\alpha) \leq f_1(\alpha)$ egyenlőtlenséget is, és azt, hogy $f_1(0) = 0$, az

$$f(\alpha) \leq f_1(\alpha) = \int_0^\alpha f_1'(\zeta)d\zeta \leq f_2(\alpha) = f_1'(0)\alpha + \int_0^\alpha\int_0^\zeta h(\xi)d\xi d\zeta$$

egyenlőtlenséget kapjuk. Nyilván $f_2''(\alpha) = h(\alpha) > 0$, így f_2 konvex függvény. Mivel $f_2(0) = 0$, $f_2'(0) = f_1'(0) < 0$ és $f_2''(\alpha) = h(\alpha)$ végtelenhez tart, ha α a $v_{\min}/\sigma$ értékhez tart, ezért az f_2 függvény a minimumát azon $\tilde{\alpha} > 0$ helyen veszi fel, mely az

$$f_2'(\alpha) = f_1'(0) + \int_0^\alpha h(\xi)d\xi = 0$$

egyenlet megoldása. A (24.33) összefüggés alapján az előző egyenlet a következővel ekvivalens:

$$f_1'(0) + \frac{1}{2}q\sigma^2\int_0^\alpha (v_{\min}-\xi\sigma)^{-1-q}d\xi = 0\ .$$

Ezen egyenletet α-ra megoldva az

$$\tilde{\alpha} = \frac{v_{\min}}{\sigma}\left(1 - (1 + \sigma v_{\min}^{q})^{\frac{1}{-q}}\right) \tag{24.34}$$

kifejezésre jutunk. A 24.60. lemma alapján

$$v_{\min}^{q} \geq \frac{1}{\sigma + 1}\,,$$

amit a (24.34) összefüggésbe helyettesítve, az f_2 függvény minimum helyére a

$$\tilde{\alpha} \geq \frac{v_{\min}}{\sigma}\left(1 - \left(1 + \frac{\sigma}{\sigma+1}\right)^{\frac{1}{-q}}\right)\,.$$

becslést kapjuk. A 24.3-24 gyakorlat eredményét alkalmazva

$$\left(1 + \frac{\sigma}{\sigma+1}\right)^{-\frac{1}{q}} = \left(1 - \frac{\sigma}{2\sigma+1}\right)^{\frac{1}{q}} \leq 1 - \frac{1}{\sigma}\,q(2\sigma+1)$$

adódik. Az előző egyenlőtlenség és a 24.60. lemma segítségével az $\tilde{\alpha}$ lépéshosszra a

$$\tilde{\alpha} \geq \frac{v_{\min}}{\sigma}\,\frac{\sigma}{q(2\sigma+1)} = \frac{v_{\min}}{q(2\sigma+1)} \geq \frac{1}{q(2\sigma+1)(\sigma+1)^{\frac{1}{\sigma}}}\,.$$

alsó korlátot nyerjük. A továbbiakban feltesszük, hogy $\sigma \geq 1$ és az

$$\alpha^{*} = \frac{1}{3q\sigma(\sigma+1)^{\frac{1}{q}}} \tag{24.35}$$

lépéshosszal számolunk. Vegyük észre, hogy $\alpha^{*} \leq \tilde{\alpha}$.

Alkalmazzuk a 24.3-25 gyakorlatot az f_2 függvényre. Először ellenőrizzük, hogy az előző lemma feltételei teljesülnek-e: $f_2(0) = 0$ és $f_2'(0) < 0$. Továbbá $f_2''(\alpha) = h(\alpha) > 0$, és

$$f_2'''(\alpha) = h'(\alpha) = \frac{q(q+1)\sigma^3}{2}(v_{\min} - \alpha\sigma)^{-2-q} > 0\,.$$

Tehát alkalmazhatjuk a lemmát, valamint a (24.32) azonosságot figyelembe véve az $f(\alpha^{*})$ értékére a következő felső becslést kapjuk:

$$f(\alpha^{*}) \leq f_2(\alpha^{*}) \leq \frac{\alpha^{*} f_2'(0)}{2} = \frac{\alpha^{*} f_1'(0)}{2} = -\frac{\alpha^{*}\sigma^2}{4}\,.$$

Végül behelyettesítve α^{*} (24.35) definícióját és tekintetbe véve $\sigma \geq 1$ feltevésünket

$$f(\alpha^{*}) \leq \frac{-\sigma}{12q(\sigma+1)^{\frac{1}{q}}} \leq \frac{-\sigma}{12q(2\sigma)^{\frac{1}{q}}} \leq \frac{-\sigma^{\frac{q-1}{q}}}{24q}\,. \tag{24.36}$$

A μ paraméter iterálásának hatása

A következőkben néhány technikai eredményt ismertetünk, melyekre a konkrét elemzések során lesz szükségünk. Mivel

$$\psi'(t) = t - t^{-q}\,, \qquad \psi''(t) = 1 + qt^{-q-1}\,,$$

ezért $\psi''(t) \geq 1$ minden $t > 0$ esetén. Továbbá ψ függvényt második deriváltja és integrálás segítségével felírva alsó, illetve felső becslést adhatunk meg (a bizonyításhoz lásd 24.3-26 gyakorlatot).

24.63. lemma. *A ψ magfüggvényre a következő egyenlőtlenség teljesül:*

$$\frac{1}{2}(t-1)^2 \leq \psi(t) \leq \frac{1}{2}\psi'(t)^2\,, \qquad t > 0\,.$$

A fenti lemmában a távolságfüggvény magfüggvényére kapott alsó becslés segítségével a $\mathbf{v}$ vektor normájára felső becslést adhatunk.

24.64. következmény. *A következő egyenlőtlenség teljesül*

$$\|\mathbf{v}\| \leq \sqrt{n} + \sqrt{2\Psi(\mathbf{v})}\,.$$

Bizonyítás. Felhasználva a 24.63. lemmát és a Cauchy-Schwarz-egyenlőtlenséget a

$$\begin{aligned} 2\Psi(\mathbf{v}) = \;& 2\textstyle\sum_{i=1}^{n}(v_i-1)^2 = \|\mathbf{v}\|^2 - 2\mathbf{e}^T\mathbf{v} + n \\ \geq \;& \|\mathbf{v}\|^2 - 2\,\|\mathbf{e}\|\,\|\mathbf{v}\| + \|\mathbf{e}\|^2 = (\|\mathbf{v}\| - \|\mathbf{e}\|)^2 \end{aligned}$$

egyenlőtlenséget kapjuk. Ezt átrendezve

$$\|\mathbf{v}\| \leq \|\mathbf{e}\| + \sqrt{2\Psi(\mathbf{v})} = \sqrt{n} + \sqrt{2\Psi(\mathbf{v})}$$

adódik, amiből már következik az állítás. ∎

Az előző lemmában a ψ függvényre adott felső korlátot összevetve a σ definíciójával a távolságfüggvényre felső becslést eredményez.

24.65. következmény. $\Psi(\mathbf{v}) \leq \frac{1}{2}\sigma(\mathbf{v})^2$.

Bizonyítás. Ismét a 24.63. lemmát használva, illetve a (24.28) alapján

$$\Psi(\mathbf{v}) = \sum_{i=1}^{n}\psi(v_i) \leq \frac{1}{2}\sum_{i=1}^{n}\psi'(v_i)^2 = \frac{1}{2}\,\|\nabla\Psi(\mathbf{v})\|^2 = \frac{1}{2}\,\sigma(\mathbf{v})^2\,.$$

Ezzel a bizonyítást befejeztük. ∎

A következőkben megvizsgáljuk, hogyan változik a magfüggvény értéke, ha a függvény argumentumát egy $\beta \geq 1$ számmal megszorozzuk.

24.66. lemma. *Legyen $\beta \geq 1$. Ekkor*

$$\psi(\beta t) \leq \psi(t) + \frac{1}{2}(\beta - 1)t^2 \ .$$

Bizonyítás. A (24.27)-ban a bevezetett jelölést használva

$$\psi(\beta t) = \frac{\beta^2 t^2 - 1}{2} + \psi_0(\beta t) = \psi(t) + \frac{1}{2}(\beta^2 t^2 - t^2) + \psi_0(\beta t) - \psi_0(t) \ .$$

Mivel a ψ_0 függvény monoton fogyó, $\psi_0(\beta t) - \psi_0(t) \leq 0$. Ebből pedig már következik a bizonyítandó egyenlőtlenség. ■

A μ paraméter megváltozásának a távolságfüggvényre gyakorolt hatásának vizsgálatakor használjuk fel a fenti lemmában kapott egyenlőtlenséget. A 24.67. lemmában megfogalmazott egyenlőtlenség bizonyítását az Olvasóra bízzuk (lásd 24.3-27 gyakorlat).

24.67. lemma. *Legyen $\theta \in (0,1)$ és $\mu^+ = (1-\theta)\mu$. Ekkor*

$$\Psi(\mathbf{x}, \mathbf{s}, \mu^+) \leq \Psi(\mathbf{x}, \mathbf{s}, \mu) + \frac{\theta}{2(1-\theta)} \left(2\Psi(\mathbf{v}) + \sqrt{2n\Psi(\mathbf{v})} + n \right) \ .$$

A szükséges becslések előállítása után most rátérünk az algoritmus lépésszámának meghatározására. Az eljárás külső ciklusa elején (a μ paraméter iterálása előtt) $\Phi(\mathbf{x}, \mathbf{s}, \mu) = \Psi(\mathbf{v}) \leq \tau$ teljesül. A 24.67. lemma alapján a μ frissítése után

$$\Phi(\mathbf{x}, \mathbf{s}, \mu^+) \leq \tau + \frac{\theta}{2(1-\theta)} \left(2\tau + \sqrt{2n\tau} + n \right) \tag{24.37}$$

áll fenn. A (24.35) alatt meghatározott α^* értéket használva (24.36) alapján a távolságfüggvény értékének csökkenése legalább

$$\frac{\sigma^{\frac{q-1}{q}}}{24q} \ . \tag{24.38}$$

Megjegyeznénk, hogy ez az eredmény csak a $\sigma \geq 1$ feltétel teljesülése mellett igaz, ám a 24.65. következménybeli egyenlőtlenség alapján ennek elégséges feltétele a $\Phi(\mathbf{x}, \mathbf{s}, \mu) \geq 1$ egyenlőtlenség. Ami pedig $\tau \geq 1$ mellett biztosan fennáll minden belső ciklus megkezdésekor. A 24.65. következmény és (24.38) alapján minden belső iteráció során a távolságfüggvény csökkenése legalább

$$\frac{\Psi^{\frac{q-1}{2q}}}{24q} \ , \tag{24.39}$$

ahol $\Psi = \Psi(\mathbf{x}, \mathbf{s}, \mu)$ a távolságfüggvény értéke a belső iteráció megkezdése előtt.

24.68. lemma. *Minden belső ciklusban legfeljebb*

$$\left\lceil \frac{48q^2(\tau + \frac{\theta}{2(1-\theta)}(2\tau + \sqrt{2n\tau} + n))^{\frac{q+1}{2q}}}{q+1} \right\rceil$$

iteráció történik.

Bizonyítás. Jelölje Ψ a távolságfüggvény értékét a belső ciklus elején. Egy iteráció után az új értéket pedig jelölje Ψ^+. Ekkor (24.39) alapján

$$\Psi^+ \leq \Psi - \frac{\Psi^{\frac{q-1}{2q}}}{24q} \, .$$

Emeljük az egyenlőtlenséget $(q+1)/(2q)$ hatványra, majd alkalmazzuk az $(1-t)^\alpha \leq 1 - \alpha t$, $(\alpha,\, t \in [0,1])$ összefüggést

$$\begin{aligned}(\Psi^+)^{\frac{q+1}{2q}} &\leq \left(\Psi - \frac{\Psi^{\frac{q-1}{2q}}}{24q}\right)^{\frac{q+1}{2q}} = \Psi^{\frac{q+1}{2q}} \left(1 - \frac{\Psi^{-\frac{q+1}{2q}}}{24q}\right)^{\frac{q+1}{2q}} \\ &\leq \Psi^{\frac{q+1}{2q}} \left(1 - \frac{q+1}{2q} \, \frac{\Psi^{-\frac{q+1}{2q}}}{24q}\right) = \Psi^{\frac{q+1}{2q}} - \frac{q+1}{48q^2} \, .\end{aligned}$$

Ha $\Psi^{(k)}$ jelöli a k-adik belső iteráció után a távolságfüggvény értékét, akkor a következő becslés áll fenn

$$\Psi^{(k)} \leq \Psi^{\frac{q+1}{2q}} - k\,\frac{q+1}{48q^2} \, .$$

Ha a $\Psi^{(k)}$ értékre adott felső korlát kisebb, mint τ, akkor a belső iterációk száma legfeljebb k, tehát

$$\Psi^{\frac{q+1}{2q}} - k\,\frac{q+1}{48q^2} \leq \tau \, ,$$

átrendezés után

$$\frac{48q^2}{q+1}\left(\Psi^{\frac{q+1}{2q}} - \tau\right) \leq k$$

adódik. Az alsó korlátot felülről becsülve $\tau > 0$, illetve (24.37) figyelembevételével a lemma állítását kapjuk. ∎

Az algoritmusunk hosszú lépéses módszer, mivel a θ paraméter nem függ a feladat dimenziójától. Ezt figyelembe véve a külső iterációk számát a következő lemma adja meg, melynek bizonyítását az Olvasóra bízzuk (lásd 24.3-28 gyakorlat).

24.69. lemma. *A külső iterációk száma legfeljebb* $\left\lceil \frac{1}{\theta} \lg \frac{n}{\varepsilon} \right\rceil$.

A belső iterációk maximális számának és a külső iterációk maximális számának szorzata felső becslést ad az eljárás lépésszámára.

24.70. tétel. *Az algoritmus legfeljebb*

$$\left\lceil \frac{48q^2(\tau + \frac{\theta}{2(1-\theta)}(2\tau + \sqrt{2n\tau} + n))^{\frac{q+1}{2q}}}{q+1} \right\rceil \left\lceil \frac{1}{\theta} \lg \frac{n}{\varepsilon} \right\rceil$$

lépésben egy olyan $(\mathbf{x}, \mathbf{s})$ *megengedett pontpárt ad, melyre* $\mathbf{x}^T\mathbf{s} < \varepsilon$.

Ezek alapján az algoritmus lépésszáma $0 < \theta < 1$ konstans, $\tau = n$ választás esetén

$$O\left(q\, n^{\frac{q+1}{2q}} \lg \frac{n}{\varepsilon}\right) .$$

Rögzített $q > 1$ esetén ez a hosszú lépéses algoritmusokra ismert $O(n \lg \frac{n}{\varepsilon})$ lépésszám javítását jelenti. A $q = \frac{1}{2} \lg n$ választás minimalizálja az előző felső korlátot. Ekkor az iterációszám

$$O\left(\sqrt{n} \lg n \lg \frac{n}{\varepsilon}\right),$$

ami már igen közel van a belsőpontos algoritmusok elméletében ismert legjobb $O(\sqrt{n} \lg \frac{n}{\varepsilon})$ lépésszámhoz.

Gyakorlatok

24.3-1. Tekintsük a következő adatokkal adott lineáris programozási feladatot

$$\begin{array}{lrrrl} \min & 5\,x_1 & -5\,x_2 & +7\,x_3 & \\ & 3\,x_1 & -3\,x_2 & -2\,x_3 & = -3 \\ & x_1 & -1\,x_2 & -5\,x_3 & = \quad 1\,, \end{array}$$

ahol x_1, $x_2, x_3 \geq 0$. Vizsgáljuk meg, teljesül-e a feladatra a belső pont feltétel.

24.3-2. Bizonyítsuk be a 24.38. állítást.

24.3-3. Bizonyítsuk be a 24.39. állítást.

24.3-4. Bizonyítsuk be a 24.40. állítást. *Útmutatás.* Bizonyítsuk be, hogy a

$$\begin{pmatrix} 0 & A & 0 \\ A^T & 0 & I \\ 0 & \bar{S} & \bar{X} \end{pmatrix}$$

együtthatómátrix reguláris, ahol $\bar{X}$, és $\bar{S}$ az $\bar{\mathbf{x}}$, illetve az $\bar{\mathbf{s}}$ vektorokból képzett diagonális mátrixok.

24.3-5. Bizonyítsuk be a 24.41. lemmát.

24.3-6. Bizonyítsuk be a 24.43. lemmát.

24.3-7. Legyen adott a (P) és a (D) feladat, valamint az $\alpha > 0$ szám. Tegyük fel, hogy $\bar{\mathbf{x}} \in \mathcal{P}_+$ és $(\bar{\mathbf{y}}, \bar{\mathbf{s}}) \in \mathcal{D}_+$. Bizonyítsuk be, hogy $\bar{\mathbf{x}} + \alpha\,\Delta\mathbf{x} \geq \mathbf{0}$ és $\bar{\mathbf{s}} + \alpha\,\Delta\mathbf{s} \geq \mathbf{0}$ pontosan akkor teljesül, ha

$$\bar{\mathbf{x}}\,\bar{\mathbf{s}} + \alpha\,(\mu\,\mathbf{e} - \bar{\mathbf{x}}\,\bar{\mathbf{s}}) + \alpha^2 \Delta\mathbf{x}\,\Delta\mathbf{s} \geq \mathbf{0}\ . \tag{24.40}$$

Adott $\Delta\mathbf{x}$ és $\Delta\mathbf{s}$ esetén határozzuk meg a maximális α értéket, amelyre az (24.40) egyenlőtlenség teljesül.

24.3-8. Bizonyítsuk be a 24.44. lemmát. Bizonyítsuk be azt is, hogy az adott μ-centrum dualitásrése is $n\,\mu$. *Útmutatás.* Egy $\mathbf{x} \in \mathcal{P}$ és $(\mathbf{y}, \mathbf{s}) \in \mathcal{D}$ megoldáspár μ-centrum, ha $\mathbf{x}\,\mathbf{s} = \mu\,\mathbf{e}$ teljesül.

24.3-9. Bizonyítsuk be a 24.45. következményt.

24.3-10. Legyenek $\mathbf{a}$ és $\mathbf{b}$ tetszőleges egymásra merőleges vektorok. Bizonyítsuk be, hogy

$$\|\mathbf{a}\,\mathbf{b}\|_\infty \leq \frac{1}{4}\,\|\mathbf{a}+\mathbf{b}\|^2 \quad \text{és} \quad \|\mathbf{a}\,\mathbf{b}\| \leq \frac{\sqrt{2}}{4}\,\|\mathbf{a}+\mathbf{b}\|^2\ .$$

24.3-11. Bizonyítsuk be a 24.47. állítást.

24.3-12. Legyen $\partial = \partial_1(\bar{\mathbf{x}}, \bar{\mathbf{s}}; \mu)$ és $\partial^+ = \partial_1(\mathbf{x}^+, \mathbf{s}^+; \mu)$. Határozzuk meg azt a legkisebb ∂ értéket, amely garantálja, hogy $\partial^+ \leq \partial$ teljesüljön. *Útmutatás.* Használjuk fel a 24.48. tételt.

24.3-13. (*Ling 1. lemmája* (1993)) Legyen $\mathbf{u} \in \mathbb{R}^p$ olyan vektor, amelyre $\mathbf{u} > -\mathbf{e}$ és legyen $\sigma = \mathbf{e}^T\mathbf{u}$. Bizonyítsuk be, hogy ha $\mathbf{u} \geq \mathbf{0}$ vagy $\mathbf{u} \leq \mathbf{0}$, akkor

$$\sum_{i=1}^{p} \frac{-v_i}{1+v_i} \leq \frac{-\sigma}{1+\sigma}\ ,$$

és egyenlőség pontosan akkor teljesül, ha legfeljebb egy nem nulla koordinátája van az $\mathbf{u}$ vektornak. *Útmutatás.* Használjuk fel az

$$f : (-1,\infty)^p \to \mathbb{R} \quad \text{és} \quad f(\mathbf{u}) = \sum_{i=1}^{p} \frac{-v_i}{1+v_i}$$

függvény konvexitását.

24.3-14. (*Ling 2. lemmája* (1993)) Legyenek az $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$ ortogonális vektorok, és tegyük fel, hogy $\|\mathbf{u}+\mathbf{v}\| = 2\,\rho$, ahol $\rho < 1$. Bizonyítsuk be, hogy

$$\mathbf{e}^T\left(\frac{\mathbf{e}}{\mathbf{e}+\mathbf{u}\,\mathbf{v}} - \mathbf{e}\right) \leq \frac{2\,\rho^4}{1-\rho^4}\ .$$

Útmutatás. Alkalmazzuk az első Ling-lemmát két egymásra merőleges vektor koordinátánkénti szorzatára.

24.3-15. Ismételjük meg a 24.3-12 gyakorlatot, de most használjuk fel a 24.49. tételt. Egyenlő-e a két szám? Mi ennek a jelentése?

24.3-16. Bizonyítsuk be a 24.50. lemmát.

24.3-17. Bizonyítsuk be a (24.24) összefüggést.

24.3-18. Bizonyítsuk be a 24.53. lemmát. *Útmutatás.* Használjuk fel a centralitási mérték definícióját, és hogy $\mathbf{v} > \mathbf{0}$, $-2\partial v_i \leq 1 - v_i^2 \leq 2\partial v_i$ teljesül minden indexre.

24.3-19. Bizonyítsuk be, hogy a prediktor-korrektor algoritmus $\tau = 1/2$ és $\alpha = 1/(2\sqrt{n})$ értékek esetén jól definiált, azaz minden iteráció után a kapott $(\mathbf{x}, \mathbf{s})$ pontra teljesül $\partial(\mathbf{x}, \mathbf{s}, \mu) \leq \tau$.

24.3-20. Bizonyítsuk be a 24.56. lemmát. *Útmutatás.* Kövessük a 24.54. lemma bizonyítását, azzal a különbséggel, hogy a $\|\mathbf{d}_x^a\, \mathbf{d}_s^a\|_\infty \leq n/4$ becslés helyett a $\|\mathbf{d}_x^a\, \mathbf{d}_s^a\|_\infty \leq \|\mathbf{d}_x^a\, \mathbf{d}_s^a\|$ felső korlátot alkalmazzuk.

24.3-21. Bizonyítsuk be, hogy a prediktor-korrektor algoritmus adaptív változata $\tau = 1/3$ és $\alpha = 2/(1 + \sqrt{1 + 13\,\|\mathbf{d}_x^a\, \mathbf{d}_s^a\|})$ értékek esetén jól definiált.

24.3-22. Bizonyítsuk be a 24.57. lemmát. *Útmutatás.* Vizsgáljuk meg az

$$f(x) = 1 - \frac{2}{1 + \sqrt{1 + 13x}} \qquad \text{függvényt .}$$

24.3-23. Bizonyítsuk be a 24.60. lemmát. *Útmutatás.* Induljunk ki a $\sigma = \|\mathbf{v}^{-q} - \mathbf{v}\| \geq \|v_i^{-q} - v_i\|$ becslésből és végezzünk esetszétválasztást az 1 és $v_{\min}$, illetve $v_{\max}$ értékek viszonyára tekintettel.

24.3-24. Bizonyítsuk be, hogy ha $p \in [0, 1]$, akkor $(1 - t)^p \leq 1 - pt$ teljesül minden $t \in [0, 1]$ esetén.

24.3-25. Legyen $h(t)$ kétszer differenciálható konvex függvény, továbbá $h(0) = 0$, $h'(0) < 0$ és a h függvény a $t^* > 0$ pontban veszi fel a (globális) minimumát. Bizonyítsuk be, hogy ha h'' függvény monoton nő a $[0, t^*]$ intervallumon, akkor

$$h(t) \leq \frac{t h'(0)}{2} \qquad \text{minden } 0 \leq t \leq t^* \text{ esetén .}$$

24.3-26. Bizonyítsuk be a 24.63. lemmát.

24.3-27. Bizonyítsuk be a 24.67. lemmát. *Útmutatás.* Alkalmazzuk a 24.66. lemmát a $\beta = 1/\sqrt{1 - \theta}$ értékre, majd használjuk fel a 24.64. következmény eredményét a további becslésekhez.

24.3-28. Bizonyítsuk be a 24.69. lemmát. *Útmutatás.* A 24.46. tétel bizonyításához hasonlóan igazolható.

Megjegyzések a fejezethez

A szakirodalomból ismert egyik első modell, amelyet lineáris egyenlőtlenség rendszerrel fogalmaztak meg, a Fourier-féle mechanikai elv volt, amely Farkas Gyula vizsgálódásainak is a kiindulópontját adta. A lineáris egyenlőtlenség rendszerek megoldhatóságának a kérdését Farkas Gyula [73, 75, 74] már a XIX. században tanulmányozta. Az első ismert számítási eljárást, speciális lineáris egyenlőtlenség rendszerek megoldására Fourier fogalmazta meg. Módszerét tetszőleges lineáris egyenlőtlenség rendszer megoldására Motzkin általánosította és ennek a következtében, ma is Fourier-Motzkin eliminációs módszerként [243] ismerjük. Farkas Gyulának a lineáris egyenlőtlenség rendszerek megoldhatóságáról szóló eredménye, amelyet ma Farkas-lemmaként ismer az egész világ, az optimumszámítás egyik legtöbbet idézett eredménye. Farkas Gyula életéről, optimalizáláselméleti munkásságáról és annak hatásairól Prékopa András írt egy méltatást [244].

Lineáris programozási feladat megoldásának a módszerei és a számítógépek szoros kapcsolatban fejlődtek az elmúlt évtizedeken keresztül. Az 1950-es és 1960-as évek elején, a gyakorlati lineáris programozási feladatok megoldása során, memória és merev lemez kapacitási korlátok határozták meg a lineáris programozási algoritmusok fejlesztésének irányát és a gyakorlatban megoldható feladatok méretét. Mivel a lineáris programozási modellek mérete, a gyakorlati alkalmazások követelmények megfelelően folyamatosan növekedett, egy ideig ez kihatott a számítógépek fejlesztésére is [60]. Ma nem a lineáris programozási feladatok méretének a növekedése jelenti a számítógépek fejlesztésének a motorját, habár a nagyméretű lineáris és a lineáris egészértékű feladatok megoldása továbbra is komoly számítási kapacitások meglétét igényeli.

Hacsián [152] megmutatta, hogy a Judin és Nemirovski [142] által konvex optimalizálási feladatok megoldására konstruált *ellipszoid módszert* lineáris programozási feladatra adaptálva polinomiális algoritmust eredményez. Hacsián bebizonyította, hogy az ellipszoid módszer legfeljebb $O(n^2\,L)$ iterációban, és $O(n^4\,L)$ aritmetikai művelettel bármely lineáris programozási feladatot megold. Gács és Lovász [97] dolgozatukban az ellipszoid módszernek egy egyszerű bizonyítását ismertették. Évekig tartó erőfeszítések eredményeként az ellipszoid módszer különböző számítógépes megvalósításait fejlesztették ki, de ezek egy-egy nagyon speciális kombinatorikus optimalizálási feladattól eltekintve, gyakorlati feladatok megoldása során meg sem közelítették a szimplex alapú számítógépes programcsomagok 1980-as évekbeli hatékonyságát. Már lankadni kezdett a téma iránti érdeklődés, amikor Karmarkar [149] felfedezte *projektív skálázású primál belsőpontos algoritmusát* lineáris programozási feladatok megoldására. Bebizonyította, hogy algorit-

musa nem több, mint $O(n\,L)$ iterációban oldja meg a lineáris programozási feladatokat, ahol egy iteráció számítási igénye legfeljebb $O(n^{2.5})$ aritmetikai művelet. Ezzel Karmarkar nem csak Hacsián $O(n^4\,L)$ legrosszabb esetben adott aritmetikai művelet korlátját javította $O(n^{3.5}\,L)$-re, hanem azt is állította, hogy algoritmusa a szimplex módszer implementációnál is sokkal gyorsabban old meg nagy méretű lineáris programozási feladatokat. Azóta számos kutató dolgozott azon, hogy olyan belsőpontos algoritmusokat fogalmazzanak meg, amelyek nem csak polinomiálisak, hanem numerikus szempontból is hatékonyak. Kezdetben Karmarkar algoritmusának egyszerűsített variánsait dolgozták ki. Megmutatták, hogy az ún. *projektív transzformáció* nem nélkülözhetetlen eleme a lineáris programozási feladat belsőpontos módszerrel történő megoldásánál. Több kutató észrevette, hogy a Karmarkar-algoritmus egyik variánsa egy régi nemlineáris optimalizálási feladatok megoldására kifejlesztett algoritmus, az ún. *logaritmikus büntetőfüggvényes módszer,* lineáris programozási megfelelője. Ez indította el néhány régi nemlineáris programozási algoritmus újra felfedezését, lineáris programozási feladatokra való alkalmazását. Meglepő észrevételt tettek a kutatók: a logaritmikus barrier módszer lineáris programozási megfelelőjét Frisch [93] dolgozta ki, a *középpontok módszere* Huard [122] nevéhez köthető és az általunk is tárgyalt *affin skálázási algoritmus* eredeti tisztán primál, illetve duál változatát Dikin fogalmazta meg.Természetesen az ötvenes és hatvanas években ezeknek az algoritmusoknak sem a gyakorlatban hatékony számítógépes implementálására, sem pedig a lépésszám vizsgálatára nem került sor. Mindhárom szerző messze megelőzte a korát és eredményeiket évtizedekre elfelejtette a tudományos közvélemény.

A modern belsőpontos módszerek numerikus és számítógépes variánsainak a kidolgozásában kulcsszerepet játszottak Sonnevend [274] és Megiddo [201] egymástól független eredményei, az ún. *centrális úttal* és *analitikus centrummal* kapcsolatos vizsgálataik. Sonnevend fogalmazta meg az első centrális utat követő algoritmust. Ezt követően Renegar [248] és tőle függetlenül Roos és Vial adott lineáris programozási feladatokat legfeljebb $O(\sqrt{n}L)$ iterációban és $O(n^3L)$ aritmetikai művelettel megoldó belsőpontos algoritmust. Ezek az eredmények jelentősen hozzájárultak Kojima, Mizuno és Yoshise [158] *primál-duál Newton-lépéses logaritmikus barrier algoritmusának* a megfogalmazásához, és az algoritmusra legrosszabb esetben jellemző lépésszám felső korlátjának a bizonyításához. Ma is ez az algoritmus a legtöbb számítógépes implementáció alapja. Mind elméleti, mind pedig gyakorlati szempontból igen érdekes változata a belsőpontos algoritmusoknak az ún. *prediktor-korrektor belsőpontos módszer.* Ebből az egyik legérdekesebb variánst, a Mizuno, Todd és Ye [205] prediktor-korrektor algoritmust tárgyaltuk ebben a fejezetben. A belsőpontos módszerek egyik sokat kritizált tulajdonsága, hogy az algo-

ritmusoknak ahhoz, hogy elkezdhessék a feladat megoldását, egy belsőpont megoldás szükséges. Ezt a kérdést oldották meg, a fejezeten általunk is bemutatott módon, beágyazással, Ye, Todd és Mizuno [318]. A beágyazásról, belsőpontos algoritmusokról és azok bonyolultságáról, pontos megoldás előállításáról és a témakörhöz kapcsolódó numerikus és számítógépes kérdésekről az érdeklődő olvasó további információkat talál Ye [317] és Roos, Terlaky és Vial könyvében [251].

New references are [260, 302].

25. Játékelmélet

Jelölje N a döntéshozók (a továbbiakban ***játékosok***) számát, és minden $k = 1, 2, \ldots, N$ számra legyen $\mathcal{P}_k$ a k-adik játékos és S_k a $\mathcal{P}_k$ játékos ***megengedett akcióinak halmaza.*** Az $s_k \in S_k$ elemeket $\mathcal{P}_k$ ***stratégiáinak*** nevezzük. S_k a $\mathcal{P}_k$ játékos ***stratégiahalmaza.*** A játék tetszőleges lejátszásában minden játékos választ egy stratégiát, ekkor az $\mathbf{s} = (s_1, s_2, \ldots, s_N)$ vektort a játékosok ***szimultán stratégiavektorának*** hívjuk, ahol $s_k \in S_k$, $k = 1, 2, \ldots, N$. Minden játékos megfeleltet minden $\mathbf{s} \in S = S_1 \times S_2 \times \cdots \times S_N$ szimultán stratégiavektornak egy valós számot. Ez a valós szám minden játékos esetében tekinthető úgy, mint egy hasznossági függvény értéke, amely értéket a játékos a játék adott kimeneteléhez hozzárendeli. Ez a hasznossági függvény tükrözi az adott játékos értékelését a kimenetelekről. Legyen $\mathcal{P}_k$ tetszőleges játékos, ekkor ha $f_k(\mathbf{s})$ jelöli ezt az értéket, akkor az $f_k : S \to \mathbb{R}$ függvényt a $\mathcal{P}_k$ játékos ***kifizetőfüggvényének,*** az $f_k(\mathbf{s})$ értéket $\mathcal{P}_k$ ***kifizetésének,*** az $(f_1(\mathbf{s}), f_2(\mathbf{s}), \ldots, f_N(\mathbf{s}))$ vektort pedig ***kifizetővektornak*** nevezzük. Az N szám, az S_k halmazok, az f_k kifizetőfüggvények $(k = 1, 2, \ldots, N)$ összessége teljes körűen meghatároz egy N személyes játékot. A továbbiakban az N személyes játék jelölésére a $G = \{N; S_1, S_2, \ldots, S_N; f_1, f_2, \ldots, f_N\}$ jelölést fogjuk használni.

A G játék megoldása a ***Nash-egyensúly*** (a továbbiakban röviden egyensúly), amely egy olyan $\mathbf{s}^\star = (s_1^\star, \ldots, s_N^\star)$ stratégiavektor, hogy minden k-ra

1. $s_k^\star \in S_k$;

2. Minden $s_k \in S_k$-ra

$$f_k(s_1^\star, s_2^\star, \ldots, s_{k-1}^\star, s_k, s_{k+1}^\star, \ldots, s_N^\star)$$
$$\leq f_k(s_1^\star, s_2^\star, \ldots, s_{k-1}^\star, s_k^\star, s_{k+1}^\star, \ldots, s_N^\star)\,. \tag{25.1}$$

Az 1. feltétel szerint az egyensúly k-adik komponense megvalósítható stratégia $\mathcal{P}_k$ számára, míg a 2. feltétel szerint egyik játékos sem tudja növelni a kifizetését az által, hogy egyoldalúan eltér az egyensúlyi stratégiától. Más szavakkal, minden játékosnak az az érdeke, hogy tartsa az egyensúlyt, hiszen ha bármely játékos eltér (egyoldalúan) az egyensúlytól, akkor annak a kifizetése nem nő.

		$\mathcal{P}_2$	
		N	C
$\mathcal{P}_1$	N	$(-2,-2)$	$(-10,-1)$
	C	$(-1,-10)$	$(-5,-5)$

25.1. ábra. Fogoly dilemma.

25.1. Véges játékok

Egy G játékot ***végesnek*** nevezünk, ha minden S_k stratégiahalmaz véges sok stratégiát tartalmaz[1]. A legismertebb kétszemélyes játék a ***fogoly dilemma,*** mely a következő példa tárgya.

25.1. példa. *Fogoly dilemma.* A két játékos két fogoly, akiket egy súlyos bűntett elkövetésének gyanújával vett őrizetbe a rendőrség, de az ügyészségnek még nincs elég bizonyítéka a vádemeléshez. A két fogoly két különböző cellában van fogva tartva, így nem tudnak egymással kommunikálni. Az ügyészség célja, hogy rávegye a foglyokat a hatóságokkal való együttműködésre, abból a célból, hogy a szükséges bizonyítékok meglegyenek a vádemeléshez. Tehát $N = 2$, a stratégiahalmazok mindkét játékos esetén kételeműek: együttműködni (C), vagy nem együttműködni (N). Mindkét játékossal külön-külön közölték, hogy ha csak az egyikük tesz vallomást, akkor a vallomást tevő csak 1 éves, míg a másik 10 éves börtönbüntetést kap. Ha mind a ketten vallomást tesznek, akkor mindketten 5 éves börtönbüntetést kapnak, míg ha mindketten megtagadják a vallomást, akkor csak egy kevésbé súlyos bűncselekmény miatt ítélik el őket, és mindketten 2 éves börtönbüntetést kapnak. Mindkét játékosnak az a célja, hogy minimalizálja a börtönben töltött időt, vagy ami ezzel ekvivalens, maximalizálja a börtönben töltött idő ellentettjét. A kifizetésvektorokat a 25.1. ábra tartalmazza, ahol $\mathcal{P}_1$ stratégiáit a sorok, míg $\mathcal{P}_2$ stratégiáit az oszlopok tartalmazzák, és minden kifizetésvektorban az első érték $\mathcal{P}_1$ kifizetése, míg a második szám $\mathcal{P}_2$ kifizetése.

A kifizetéseket összehasonlítva világos, hogy csak a (C, C) stratégiapáros lehet egyensúly, mivel

$$\begin{aligned} f_2(\mathrm{N,N}) = -2 &< -1 = f_2(\mathrm{N,C})\,, \\ f_1(\mathrm{N,C}) = -10 &< -5 = f_1(\mathrm{C,C})\,, \\ f_2(\mathrm{C,N}) = -10 &< -5 = f_2(C,C)\,. \end{aligned}$$

A (C, C) stratégiapáros tényleg egyensúly, mivel

$$\begin{aligned} f_1(\mathrm{C,C}) = -5 > -10 = f_1(\mathrm{N,C})\,, \\ f_2(\mathrm{C,C}) = -5 > -10 = f_2(\mathrm{C,N})\,. \end{aligned}$$

Ebben a játékban egyetlen egyensúly van.

[1] A játék definíciója szerint a játékosok száma is véges. *A fordító.*

		$\mathcal{P}_2$	
		N	C
$\mathcal{P}_1$	N	$(1,2)$	$(2,1)$
	C	$(2,4)$	$(0,5)$

25.2. ábra. Játék, melyben nincs egyensúly.

Az egyensúly létezése általában nem garantált, és ha létezik egyensúly, akkor sem feltétlenül egyetlen.

25.2. példa. *Játék, melyben nincs egyensúly.* Módosítsuk a 25.1. ábra kifizetéseit úgy, ahogy az a 25.2. ábrán látható. Könnyen látható, hogy a módosított játékban nincs egyensúly:

$$\begin{aligned} f_1(\mathrm{N,N}) = 1 &< 2 = f_1(\mathrm{C,N})\,, \\ f_2(\mathrm{C,N}) = 4 &< 5 = f_2(\mathrm{C,C})\,, \\ f_1(\mathrm{C,C}) = 0 &< 2 = f_1(\mathrm{N,C})\,, \\ f_2(\mathrm{N,C}) = 1 &< 2 = f_2(\mathrm{N,N})\,. \end{aligned}$$

Ha a kifizetések minden kimenetelre megegyeznek, akkor több egyensúly is van a játékban: minden stratégiapár egyensúly.

25.1.1. Leszámlálás

Jelölje továbbra is N a játékosok számát, és – a kényelmes jelölés kedvéért – jelölje $s_k^{(1)}, \ldots, s_k^{(n_k)}$ a $\mathcal{P}_k$ játékos megengedett stratégiáit. Tehát $S_k = \{s_k^{(1)}, \ldots, s_k^{(n_k)}\}$. Egy $\mathbf{s} = (s_1^{(i_1)}, \ldots, s_N^{(i_N)})$ stratégiavektor pontosan akkor egyensúly, ha minden $k = 1, 2, \ldots, N$ és minden $j \in \{1, 2, \ldots, n_k\} \setminus i_k$ esetén

$$\begin{aligned} &f_k(s_1^{(i_1)}, \ldots, s_{k-1}^{(i_{k-1})}, s_k^{(j)}, s_{k+1}^{(i_{k+1})}, \ldots, s_N^{(i_N)}) \\ &\leq f_k(s_1^{(i_1)}, \ldots, s_{k-1}^{(i_{k-1})}, s_k^{(i_k)}, s_{k+1}^{(i_{k+1})}, \ldots, s_N^{(i_N)})\,. \end{aligned} \tag{25.2}$$

Vegyük észre, hogy véges játékok esetén (26.1) leegyszerűsödik (26.2)-re.

A leszámlálás alkalmazásakor ellenőrizzük a (26.2) egyenlőtlenséget az összes lehetséges $\mathbf{s}^\star = (s_1^{(i_1)}, \ldots, s_N^{(i_N)})$ N-esre úgy, hogy megvizsgáljuk a (26.2) egyenlőtlenség érvényességét minden k-ra, minden j-re. Ha az ellenőrzés sikeres, akkor $\mathbf{s}^\star$ egyensúly, ellenkező esetben $\mathbf{s}^\star$ nem egyensúly. Ha az ellenőrzés alatt egy rögzített $\mathbf{s}^\star$-ra találunk olyan k-t és j-t, hogy (26.2) nem teljesül, akkor $\mathbf{s}^\star$ nem egyensúly, így elhagyhatjuk az ellenőrzést minden további k-ra és j-re. Ez egy nagyon egyszerű algoritmus, mely $N+2$, egymásba ágyazott, az $i_1, i_2, \ldots, i_N, k$ és j változókat használó ciklusból áll.

A szükséges összehasonlítások száma legfeljebb

$$\left(\prod_{k=1}^{N} n_k\right)\left(\sum_{k=1}^{N}(n_k-1)\right) .$$

A gyakorlatban azonban az összehasonlítások száma ennél sokkal kisebb lehet, hiszen ha (26.2) nem teljesül valamilyen j-re, akkor az adott stratégiavektor esetén nem kell további összehasonlításokat tenni.

Az algoritmus pszeudokódja a következő:

LESZÁMLÁL$(s_1^{(i_1)},\dots,s_N^{(i_N)})$

```
1  for i_1 = 1 to n_1
2      * for i_2 = 1 to n_2
       ...
3      * for i_N = 1 to n_N
4                kulcs = 0
5                for k = 1 to N
6                    for j = 1 to n_k
7                        if (26.2) nem teljesül *
8                            kulcs = 1 *
9                                folytassuk a 8-adik utasítással
10               if kulcs = 0 *
11                   * (s_1^(i_1), ..., s_N^(i_N)) egyensúly
12                   print "A bemenet nem egyensúly"
```

A következőkben a kétszemélyes játékokat (N=2) vizsgáljuk és bevezetjük az $(n_1 \times n_2)$-es $\mathbf{A}^{(1)}$ és $\mathbf{A}^{(2)}$ mátrixokat, melyek elemei $\mathbf{A}^{(1)}(i,j) = f_1(i,j)$, illetve $\mathbf{A}^{(2)}(i,j) = f_2(i,j)$. Az $\mathbf{A}^{(1)}$, $\mathbf{A}^{(2)}$ mátrixokat ***kifizetőmátrixoknak*** nevezzük. Az $\left(s_1^{(i_1)}, s_2^{(i_2)}\right)$ stratégiavektor pontosan akkor egyensúly, ha az (i_1,i_2) elem az $\mathbf{A}^{(1)}$ mátrixban a saját oszlopában, és az $\mathbf{A}^{(2)}$ mátrixban a saját sorában a legnagyobb. Ha $f_1 = -f_2$, akkor a játékot ***zérusösszegű*** játéknak nevezzük, és $\mathbf{A}^{(1)} = -\mathbf{A}^{(2)}$, tehát a játékot teljes körűen leírja $\mathbf{A}^{(1)}$, a $\mathcal{P}_1$ játékos kifizetőmátrixa. Ebben a speciális esetben az $(s_1^{(i_1)}, s_2^{(i_2)})$ stratégiavektor pontosan akkor egyensúly, ha az (i_1,i_2) elem az $\mathbf{A}^{(1)}$ mátrixban a saját oszlopában a legnagyobb, és a saját sorában a legkisebb. A zérusösszegű játékok egyensúlyára a ***nyeregpont*** elnevezés is használatos. Világos, hogy ebben az esetben az egyensúly megtalálása a leszámlálás módszerével leegyszerűsödik, hiszen csak egy mátrixszal kell foglalkozni.

Az egyszerűsített algoritmus a következő.

NYEREGPONT

```
1  for i_1 = 1 to n_1
2      for i_2 = 1 to n_2
3          kulcs = 0
4          for j = 1 to n_1
5              if a^(1)_{j i_2} > a^(1)_{i_1 i_2}
6                  kulcs = 1
7                  folytassuk a 12-edik sorban
8          for j = 1 to n_2
9              if a^(2)_{i_1 j} > a^(2)_{i_1 i_2}
10                 kulcs = 1
11                     folytassuk a 12-edik sorban
12         if kulcs = 0
13             "(s^(1)_{i_1}, s^(2)_{i_2}) egyensúly"
```

25.1.2. Véges fákkal ábrázolt játékok

Számos véges játék rendelkezik azzal a tulajdonsággal, hogy ábrázolható olyan irányított véges fával, melynek a következő tulajdonságai vannak:

1. a fa gyökeres, és a játék ennél a csúcsnál kezdődik;

2. a fa minden csúcsához tartozik egy játékos, és ha a játék elér egy csúcsot, akkor a csúcshoz tartozó játékos kiválaszt egy élt, mely az adott csúcsból indul ki, így dönt arról, hogy miként folytatódik a játék. Ekkor a játék a kiválasztott él végpontjánál folytatódik;

3. minden levélhez tartozik egy valós, N komponensű vektor, mely vektor tartalmazza az egyes játékosok kifizetéseit, ha a játék ebben a levélben ér véget;

4. minden játékos ismeri a fát, tudja, hogy mely csúcsokhoz van rendelve, és tudja, hogy milyen kifizetések tartoznak az egyes levelekhez.

Például a sakk játék rendelkezik a fenti tulajdonságokkal. A sakkot két játékos játssza ($N = 2$), a csúcsok a lehetséges állások a sakktáblán, egyszer a világossal játszó, egyszer a sötéttel játszó játékos szempontjából. Adott csúcsból kiinduló élek jelentik azokat a lépéseket, melyeket a csúcshoz rendelt játékos (aki lép) megtehet. A levél olyan állás a sakktáblán, mellyel a játék véget ér. A kifizetések az $\{1, 0, -1\}$ halmazból valók, ahol 1 azt jelenti, hogy

világos győzelmével, 0 azt jelenti, hogy döntetlennel, -1 azt jelenti, hogy sötét győzelmével ért véget a játék.

25.1. tétel. *Minden, véges fával ábrázolható játéknak van legalább egy egyensúlya.*

Bizonyítás. Abból a célból bizonyítjuk itt be ezt a tételt, hogy egy praktikus algoritmust mutassunk az egyensúly megtalálására. A bizonyítás indukción alapul, melyet annyiszor ismétlünk, amennyi a játék csúcsainak száma. Ha a játéknak csak egyetlen csúcsa van, akkor értelemszerűen ez az egyetlen csúcs egyensúly.

Tegyük fel, hogy a tétel igaz minden olyan játékra, ahol a csúcsok száma kisebb, mint n ($n \geq 2$), és nézzük azt a T_0 játékot, melynek n csúcsa van. Legyen r_0 a T_0 játék gyökere, és legyenek $r_1, r_2, \ldots, r_m$ ($m < n$) azok a csúcsok, melyeket él köt össze r_0-lal (r_0 gyerekei). Jelölje $T_1, T_2, \ldots, T_m$ a T_0 olyan diszjunkt részfáit, melyek gyökerei $r_1, r_2, \ldots, r_m$ a sorrendnek megfelelően (tehát r_2 T_2 gyökere). Ekkor minden részfának kevesebb, mint n csúcsa van, így mindegyiknek van egyensúlya (indukciós feltevés). Tegyük fel, hogy $\mathcal{P}_k$ tartozik az r_0 csúcshoz, legyenek $e_1, e_2, \ldots, e_m$ az egyes részfákhoz tartozó egyensúlyoknak a $\mathcal{P}_k$ játékoshoz tartozó kifizetései (tehát a T_m részfa egy egyensúlyában $\mathcal{P}_k$-nak e_m a kifizetése), és legyen $e_j = \max\{e_1, e_2, \ldots, e_m\}$. Ekkor a $\mathcal{P}_k$ játékos az r_j csúcsba lép a gyökérből, és azután folytatódik a játék a T_j részfában létező egyensúllyal[2]. ∎

Az előző tétel bizonyítása egy dinamikus programozás típusú algoritmust sugall, mely algoritmust ***visszafelé indukciónak*** nevezünk. Az algoritmus kiterjeszthető általánosabb esetre is, mely esetben a fának véletlen csúcsai vannak, melyekből a játék egy rögzített, diszkrét eloszlásnak megfelelően véletlenszerűen folytatódik.

A fenti algoritmus a következőképpen mutatható be formálisan. Tegyük fel, hogy a csúcsok úgy vannak megszámozva (természetes számokkal), hogy ha a j az i csúcs rákövetkezője, akkor $i < j$. A gyökérnek a legkisebb, az 1-es számot kell kapnia, a legnagyobb szám (n) az egyik levélhez tartozik. Jelölje $J(i)$ azon j csúcsok halmazát, melyekre van olyan él, mely i-ből j-be megy (i gyerekeinek halmaza). Minden i levél esetén $J(i)$ üres halmaz. Jelölje továbbá $\mathbf{p}^{(i)} = (p_1^{(i)}, \ldots, p_N^{(i)})$ a kifizetővektort, mely az i levélhez tartozik. Végül, k_i-vel jelöljük azt a játékost, aki az i csúcshoz tartozik. Az algoritmus az utolsó csúcsnál (n-nél) kezdődik, majd visszafelé lépeget $n, n-1, n-2, \ldots, 2$ és 1 sorrendben. Vegyük észre, hogy n egy levél, és rendeljük hozzá a $\mathbf{p}^{(n)}$ vektort. Ha az algoritmusban a következő csúcs (i) is levél, akkor rendeljük hozzá

[2] Nem minden egyensúly kapható meg ezzel a módszerrel, de az ezzel a módszerrel kapott egyensúlyok kifizetővektorai megegyeznek egymással. *A fordító.*

a $\mathbf{p}^{(i)}$ vektort, ha i nem levél, akkor keressük meg a legnagyobb értékeket a $\mathbf{p}_{k_i}^{(j)}$, $j \in J(i)$ számok közül. Tegyük fel, hogy a legnagyobb érték a j_i csúcsban van, ekkor hozzárendeljük a $\mathbf{p}^{(i)} = \mathbf{p}^{(j_i)}$ vektort az i csúcshoz, és továbblépünk az $i-1$ csúcsba. Miután minden $\mathbf{p}^{(n)}$, $\mathbf{p}^{(n-1)}$, $\dots, \mathbf{p}^{(2)}$ és $\mathbf{p}^{(1)}$ vektort meghatároztunk, a $\mathbf{p}^{(1)}$ vektor tartalmazza a kifizetéseket az egyensúlyban, és az egyensúlyi út a következő csúcsok mentén halad:

$$1 \to i_1 = j_1 \to i_2 = j_{i_1} \to i_3 = j_{i_2} \to \dots \ ,$$

amíg egy levélbe el nem értünk. Így megkaptuk az egyensúlyi utat.

Minden csúcsnál az összehasonlítások száma a csúcsból kiinduló élek száma mínusz 1. Tehát az algoritmusban az összehasonlítások száma az élek száma mínusz n.

Az algoritmus pszeudokódja a következő.

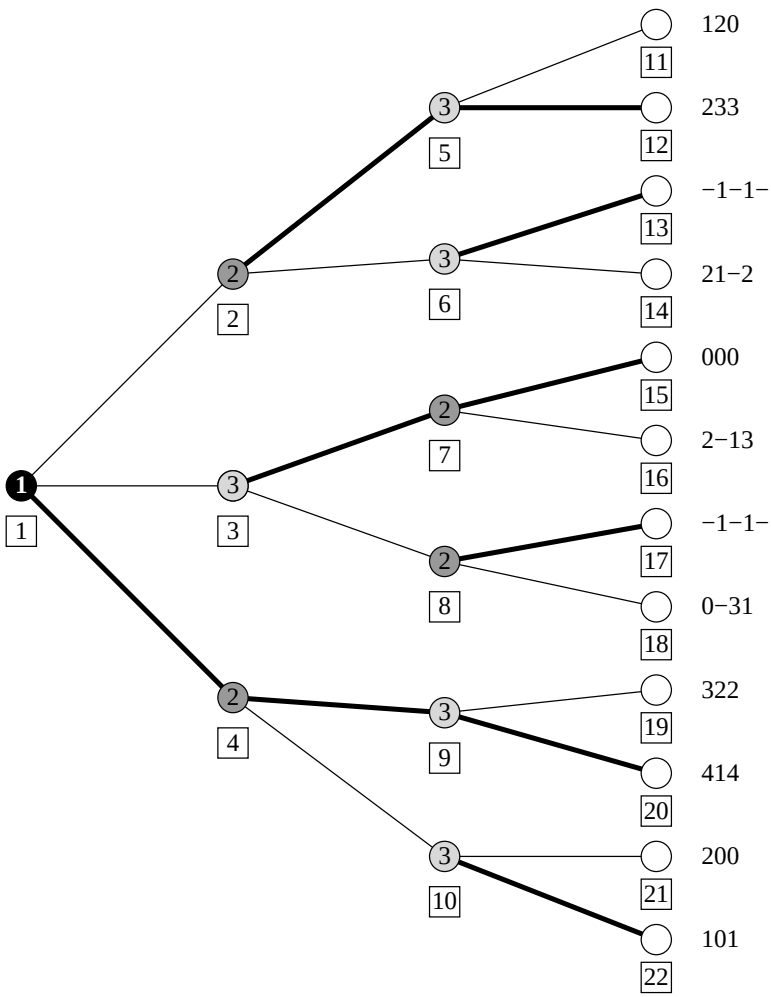
FA-EGYENSÚLYA

```
1  for i = n to 1
2      p_{K_i}^{(j_i)} = max{p_{K_i}^{(l)}, l ∈ J(i)}
3      p^{(i)} = p^{(j_i)}
4  addig nyomtassuk az 1, i_1(= j_1), i_2(= j_{i_1}), i_3(= j_{i_2}), ... sorozatot,
       amíg a végpontot el nem érjük
```

25.3. példa. *Véges fa.* A 25.3. ábrán egy véges fa látható. Minden belső csúcsnál egy kis kör látható, mely tartalmazza annak a játékosnak a jelét, aki az adott csúcshoz tartozik. A leveleknél láthatók a kifizetővektorok. Ebben a játékban három játékos van, tehát a kifizetővektoroknak három komponense van. Először megszámozzuk a csúcsokat úgy, hogy minden él kiindulópontjának kisebb legyen a sorszáma, mint a végpontjának. Ezeket a számokat tartalmazzák a csúcsok alatt látható négyzetek. Minden i csúcs esetén igaz, hogy ha $i \geq 11$, akkor i levél, tehát a visszafelé indukciót a 10-es számú csúcsnál kezdjük. Mivel a 10-es csúcshoz $\mathcal{P}_3$ tartozik, így a $(2,0,0)$ és az $(1,0,1)$ kifizetővektorok harmadik komponenseit kell összehasonlítanunk, mivel ezen két kifizetővektor tartozik azokhoz a levelekhez, melyekbe megy él a 10-es csúcsból. Mivel $1 > 0$, ezért $\mathcal{P}_3$ legjobb választása a 22-es csúcs. Tehát $j_{10} = 22$, és $\mathbf{p}^{(10)} = \mathbf{p}^{(22)} = (1,0,1)$. Ezután a 9-es csúcsot vizsgáljuk meg. A $\mathbf{p}^{(19)}$ és a $\mathbf{p}^{(20)}$ vektorok harmadik komponensét összehasonlítva világos, hogy $\mathcal{P}_3$ a 20-as csúcsot választja, így $j_9 = 20$, és $\mathbf{p}^{(9)} = \mathbf{p}^{(20)} = (4,1,4)$. Az ábrán a játékosok választásait a vastagított élek jelzik. Az eljárást a fenti logika szerint folytatva a $8, 7, \dots, 1$ csúcsokra, végül megkapjuk az 1-es csúcshoz tartozó $\mathbf{p}^{(1)} = (4,1,4)$ kifizetővektort, és az

$$1 \to 4 \to 9 \to 20$$

egyensúlyi utat.



25.3. ábra. Egy játékfa.

Gyakorlatok

25.1-1. Egy vállalkozó (E) belép a piacra, amelyet egy áruházlánc (C) tart ellenőrzése alatt. A két szereplő vetélkedése egy kétszemélyes játék. Az áruházlánc stratégiái a megengedés (S), amikor az áruházlánc megengedi, hogy a vállalkozó működjön a piacon, és az elutasítás (T), amikor igyekszik kiszorítani a vállalkozót a piacról. A vállalkozó stratégiái a maradás (I), amikor a vállalkozó a piacon marad, és a kilépés (L), amikor a vállalkozó elhagyja a piacot. A kifizetések a 25.4. ábrán láthatóak. Keressük meg az egyensúlyt.

	I	L
S	2	5
T	0	5

A C játékos kifizetései

	I	L
S	2	1
T	0	1

Az E játékos kifizetései

25.4. ábra. A 26.1-1 gyakorlat adatai.

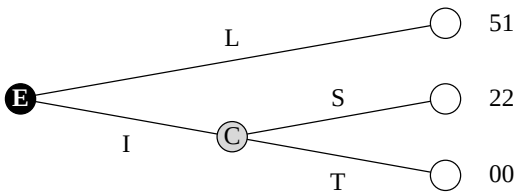
25.1-2. Egy vásárló egy három darabból álló készüléket vásárol a következő feltételekkel: ha minden darab jó, akkor a vevő fizet az eladónak α forintot, egyébként az eladó fizet a vevőnek β forintot. Mielőtt az eladó eladná az árut, ellenőrizheti bármely darabot, de az ellenőrzés költsége darabonként γ forint. Tekintsünk egy kétszemélyes játékot, ahol az eladó a $\mathcal{P}_1$ játékos, stratégiái $0, 1, 2, 3$ (az ellenőrzött darabok száma), míg az áru a $\mathcal{P}_2$ játékos, stratégiái $0, 1, 2, 3$ (hány darab hibás). Mutassuk meg, hogy ha feltesszük, hogy minden darab azonos valószínűséggel hibás, akkor a 25.5. ábrán $\mathcal{P}_1$ kifizetőmátrixa látható.

		2-es játékos			
		0	1	2	3
1-es játékos	0	α	$-\beta$	$-\beta$	$-\beta$
	1	$\alpha-\gamma$	$-\frac{2}{3}\beta-\gamma$	$-\frac{1}{3}\beta-\gamma$	$-\gamma$
	2	$\alpha-2\gamma$	$-\frac{1}{3}\beta-\frac{5}{3}\gamma$	$-\frac{4}{3}\gamma$	$-\gamma$
	3	$\alpha-3\gamma$	-2γ	$-\frac{4}{3}\gamma$	$-\gamma$

25.5. ábra. A **??** gyakorlat adatai.

25.1-3. Tegyük fel, hogy a 26.1-2. gyakorlatban bevezetett játékot úgy módosítjuk, hogy $\mathcal{P}_2$ kifizetései $\mathcal{P}_1$ kifizetéseinek az ellentettjei. Adjuk meg az α, β, γ paraméterek függvényében az egyensúlyok számát. Határozzuk meg az egyensúlyt minden esetre.

25.1-4. Tegyük fel, hogy a 26.1-2. gyakorlatban bevezetett játékban $\mathcal{P}_2$ kifizetőfüggvénye az áru értéke (V, ha minden darab jó, egyébként 0). Van-e



25.6. ábra. A 26.1-5. gyakorlat játéka.

egyensúlya ennek a játéknak?

25.1-5. Nézzük a 25.6. ábrán látható fát, mely a 26.1-1. gyakorlatban bevezetett játék fája. Keressük meg a fenti játék egyensúlyát visszafelé indukcióval.

25.1-6. Mutassuk meg, hogy egy játékos esetében a visszafelé indukció a klasszikus dinamikus programozási módszerre egyszerűsödik.

25.1-7. Tegyük fel, hogy egy véges fával ábrázolt játékban néhány csúcs úgynevezett véletlen csúcs, ami azt jelenti, hogy a játék egy ilyen csúcsból egy következő csúcsba valamilyen rögzített valószínűséggel folytatódik. Mutassuk meg, hogy ebben az általánosabb esetben is létezik egyensúly.

25.1-8. Nézzük a 25.3. ábrán adott játékot. Kétszerezzük meg $\mathcal{P}_1$ kifizetéseit, változtassuk ellentettjeire $\mathcal{P}_2$ kifizetéseit, és ne változtassunk $\mathcal{P}_3$ kifizetésein. Keressük meg ennek a módosított játéknak az egyensúlyát.

25.2. Folytonos játékok

Azokat a játékokat, ahol az S_k stratégiahalmazok egy $\mathbb{R}^{n_k}$ euklideszi tér összefüggő részhalmazai, és a kifizetőfüggvények folytonosak, ***folytonos játékoknak*** nevezzük.

25.2.1. A legjobb válaszon alapuló fixpont módszerek

Algoritmikus szemszögből nagyon intuitív és nagyon hasznos a következőkben újrafogalmazni az egyensúly fogalmát. Minden $\mathcal{P}_k$ játékosra és minden $\mathbf{s} = (s_1, s_2, \ldots, s_N) \in S = S_1 \times S_2 \times \cdots \times S_N$ stratégiavektorra definiáljuk a következő leképezést:

$$\begin{aligned} B_k(\mathbf{s}) &= \{s_k \in S_k | f_k(s_1, s_2, \ldots, s_{k-1}, s_k, s_{k+1}, \ldots, s_N) \\ &= \max_{t_k \in S_k} f_k(s_1, s_2, \ldots, s_{k-1}, t_k, s_{k+1}, \ldots, s_N)\} \,, \end{aligned} \tag{25.3}$$

amely $\mathcal{P}_k$ legjobb választásainak halmaza, a többi játékos rögzített $(s_1, s_2, \ldots, s_{k-1},\ s_{k+1}, \ldots, s_N)$ stratégiái mellett. Vegyük észre, hogy $B_k(\mathbf{s})$ nem függ s_k-tól, $B_k(\mathbf{s})$ csak a többi játékos stratégiáitól, s_l-től $(k \neq l)$

függ. Vegyük észre továbbá, hogy nincs garancia arra, hogy minden $\mathbf{s} \in S_1 \times S_2 \times \cdots \times S_N$ esetén létezik a maximum (26.3)-ban. Legyen $\sum \subseteq S$ olyan részhalmaza S-nek, hogy $B_k(\mathbf{s})$ nemüres halmaz minden k-ra, minden $\mathbf{s} \in \sum$-ra. Az $\mathbf{s}^\star = (s_1^\star, s_2^\star, \ldots, s_N^\star)$ szimultán stratégiavektor pontosan akkor egyensúly, ha $\mathbf{s}^\star \in \sum$, és $s_k^\star \in B_k(\mathbf{s}^\star)$ minden k-ra. Bevezetve a $\mathbf{B}_k(\mathbf{s}) = (B_1(\mathbf{s}), \ldots, B_N(\mathbf{s}))$ ***legjobbválasz-leképezést,*** tovább egyszerűsíthető az egyensúly fogalmának formalizmusa.

25.2. tétel. *Egy $\mathbf{s}^\star$ stratégiavektor pontosan akkor egyensúly, ha $\mathbf{s}^\star \in \sum$ és $\mathbf{s}^\star \in \mathbf{B}(\mathbf{s}^\star)$* .

Tehát az N személyes játékok egyensúlyi problémája ekvivalens azzal a problémával, hogy megtaláljuk egy halmazértékű leképezés fixpontjait.

A fixpont feladat számítási költsége függ a fixpont feladat típusától, méretétől és a választott számítási módszertől.

Az egyensúlyra vonatkozó – leggyakrabban használt – egzisztencia tételek olyan fixponttételekre támaszkodnak, mint a Brouwer-, a Kakutani-, a Banach-, a Tarski-féle fixponttétel. Bármely fixpontkereső algoritmus sikeresen alkalmazható egyensúlyok meghatározására.

A legnépszerűbb egzisztencia tétel a Kakutani-féle fixponttétel egy nyilvánvaló alkalmazása.

25.3. tétel. *Ha egy N személyes játékra minden k-ra teljesül, hogy*

1. *az S_k stratégiahalmazok egy véges dimenziós euklideszi tér nemüres, zárt, korlátos, konvex részhalmazai;*
2. *az f_k kifizetőfüggvények folytonosak S-en;*
3. *az f_k függvény konkáv az s_k változó szerint, tehát rögzített $(s_1, \ldots, s_{k-1}, s_{k+1}, \ldots, s_N)$ mellett f_k konkáv függvény,*

akkor a játéknak van legalább egy egyensúlya.

25.4. példa. *Első kétszemélyes játék.* Tekintsünk egy kétszemélyes játékot ($N = 2$), ahol a stratégiahalmazok $S_1 = S_2 = [0, 1]$, a kifizetőfüggvények $f_1(s_1, s_2) = s_1 s_2 - 2s_1^2 + 5$, és $f_2(s_1, s_2) = s_1 s_2 - 2s_2^2 + s_2 + 3$. Először mindkét játékos legjobbválasz-leképezéseit határozzuk meg. Mindkét kifizetőfüggvény lefelé nyitott parabola, melyek csúcspontjai:

$$s_1 = \frac{s_2}{4} \quad \text{és} \quad s_2 = \frac{s_1 + 1}{4} \, .$$

Minden $s_1, s_2 \in [0, 1]$ esetén ezek az értékek megvalósítható stratégiák, tehát

$$B_1(\mathbf{s}) = \frac{s_2}{4} \quad \text{és} \quad B_2(\mathbf{s}) = \frac{s_1 + 1}{4} \, .$$

Tehát az $(s_1^\star, s_2^\star)$ vektor pontosan akkor egyensúly, ha komponensei kielégítik a következő egyenlőségeket:

$$s_1^\star = \frac{s_2^\star}{4} \quad \text{és} \quad s_2^\star = \frac{s_1^\star + 1}{4} \ .$$

Könnyen látható, hogy az egyenlőségek egyetlen megoldása:

$$s_1^\star = \frac{1}{15} \quad \text{és} \quad s_2^\star = \frac{4}{15} \ ,$$

tehát $(s_1^\star, s_2^\star)$ a játék egyetlen egyensúlya.

25.5. példa. *Tengeri csatorna.* Tekintsük egy tengeri csatorna egy bizonyos részét a $[0,1]$ intervallumnak. $\mathcal{P}_2$ egy tengeralattjáró, mely az $s_2 \in [0,1]$ helyen rejtőzik. $\mathcal{P}_1$ egy repülőgép, mely bombázhat bármely $s_1 \in [0,1]$ helyet. A bombázó a tengeralattjárónak $\alpha e^{-\beta(s_1-s_2)^2}$ kárt okoz. Így egy speciális kétszemélyes játékot definiáltunk, ahol $S_1 = S_2 = [0,1]$, $f_1(s_1,s_2) = \alpha e^{-\beta(s_1-s_2)^2}$ és $f_2(s_1,s_2) = -f_1(s_1,s_2)$. Ha rögzítjük s_2-t, akkor $f_1(s_1,s_2)$ felveszi maximumát az $s_1 = s_2$ helyen, tehát $\mathcal{P}_1$ legjobbválasz-leképezése: $B_1(\mathbf{s}) = s_2$. $\mathcal{P}_2$ minimalizálni akarja $f_1(s_1,s_2)$-t, mely akkor következik be, ha $|s_1 - s_2|$ a lehető legnagyobb. Ebből következik, hogy

$$B_2(\mathbf{s}) = \begin{cases} 1, & \text{ha } s_1 < 1/2 \ , \\ 0, & \text{ha } s_1 > 1/2 \ , \\ \{0,1\}, & \text{ha } s_1 = 1/2 \ . \end{cases}$$

Világos, hogy nincs olyan $\mathbf{s} = (s_1, s_2) \in [0,1] \times [0,1]$ vektor, hogy $s_1 = B_1(\mathbf{s})$ és $s_2 \in B_2(\mathbf{s})$, tehát nincs egyensúly.

25.2.2. A Fan-egyenlőtlenség alkalmazása

Definiáljuk a $H : S \times S \to \mathbb{R}$ ***összegzőfüggvényt*** a következőképpen:

$$H_{\mathbf{r}}(\mathbf{s}, \mathbf{z}) = \sum_{k=1}^{N} r_k f_k(s_1, \ldots, s_{k-1}, z_k, s_{k+1}, \ldots, s_N) \tag{25.4}$$

minden $\mathbf{s} = (s_1, \ldots, s_N), \mathbf{z}(z_1, \ldots, z_N) \in S$-re, ahol $\mathbf{r} = (r_1, r_2, \ldots, r_N) > \mathbf{0}$ tetszőleges, rögzített.

25.4. tétel. *Az $\mathbf{s}^\star \in S$ vektor pontosan akkor egyensúly, ha*

$$H_{\mathbf{r}}(\mathbf{s}^\star, \mathbf{z}) \le H_{\mathbf{r}}(\mathbf{s}^\star, \mathbf{s}^\star) \tag{25.5}$$

minden $\mathbf{z} \in S$-re.

Bizonyítás. Először tegyük fel, hogy $\mathbf{s}^\star$ egyensúly. Ekkor a (26.1) egyenlőtlenség teljesül minden k-ra és minden $s_k \in S_k$ stratégiára. A (26.1) egyenlőtlenségeinek mindkét oldalát megszorozva az r_k együtthatókkal és összeadva őket a $k = 1, 2, \ldots, N$ értékekre, megkapjuk (26.5)-öt.

Most tegyük fel, hogy a (26.5) egyenlőtlenség teljesül minden $\mathbf{z} \in S$-re. Tetszőleges k-ra és tetszőleges $s_k \in S_k$-ra legyen $\mathbf{z} = (s_1^\star, \ldots, s_{k-1}^\star, s_k, s_{k+1}^\star, \ldots, s_N^\star)$, és alkalmazzuk (26.5)-öt. A k-adik tag kivételével minden tag egyenlő a két oldalon, így törölhetők, míg a megmaradó k-adik tag azt mutatja, hogy a (26.1) egyenlőtlenség teljesül. Tehát $\mathbf{s}^\star$ egyensúly. ■

Vezessük be a következő függvényt: $\phi(\mathbf{s}, \mathbf{z}) = H_{\mathbf{r}}(\mathbf{s}, \mathbf{z}) - H_{\mathbf{r}}(\mathbf{s}, \mathbf{s})$. Világos, hogy $\mathbf{s}^\star$ pontosan akkor egyensúly, ha

$$\phi(\mathbf{s}^\star, \mathbf{z}) \leq 0 \tag{25.6}$$

minden $\mathbf{z} \in S$-re. A (26.6) egyenlőtlenséget ***Fan-egyenlőtlenségnek*** nevezzük. A (26.6) egyenlőtlenség átírható variációs egyenlőtlenséggé (lásd később a 25.2.9. pontban) vagy fixpont feladattá. A második átírási lehetőséget mutatjuk be itt. Minden $\mathbf{s} \in S$-re legyen

$$\Phi(\mathbf{s}) = \{\mathbf{z} | \mathbf{z} \in S,\ \phi(\mathbf{s}, \mathbf{z}) = \max_{\mathbf{t} \in S} \phi(\mathbf{s}, \mathbf{t})\} \ . \tag{25.7}$$

Mivel $\phi(\mathbf{s}, \mathbf{s}) = 0$ minden $\mathbf{s} \in S$-re, így (26.6) egyenlőtlenség pontosan akkor teljesül, ha $\mathbf{s}^\star \in \Phi(\mathbf{s}^\star)$, így $\mathbf{s}^\star$ fixpontja a $\Phi : S \to 2^S$ halmazértékű leképezésnek. Tehát minden fixpontkereső módszer alkalmazható egyensúly számításra.

A fixpont probléma számítási költsége függ a fixpont probléma típusától, méretétől és a választott számítási módszertől.

25.6. példa. *Második kétszemélyes játék.* Tekintsük a 25.4 példát. A mostani esetben:

$$f_1(z_1, s_2) = z_1 s_2 - 2z_1^2 + 5 \ ,$$
$$f_2(s_1, z_2) = s_1 z_2 - 2z_2^2 + z_2 + 3 \ ,$$

így az összegzőfüggvény formája $r_1 = r_2 = 1$ esetén:

$$H_{\mathbf{r}}(\mathbf{s}, \mathbf{z}) = z_1 s_2 - 2z_1^2 + s_1 z_2 - 2z_2^2 + z_2 + 8 \ .$$

Tehát

$$H_{\mathbf{r}}(\mathbf{s}, \mathbf{s}) = 2s_1 s_2 - 2s_1^2 - 2s_2^2 + s_2 + 8 \ ,$$

és

$$\phi(\mathbf{s}, \mathbf{z}) = z_1 s_2 - 2z_1^2 + s_1 z_2 - 2z_2^2 + z_2 - 2s_1 s_2 + 2s_1^2 + 2s_2^2 - s_2 \ .$$

Vegyük észre, hogy a ϕ függvény szigorúan konkáv mind z_1 szerint, mind z_2 szerint, és ϕ szétválasztható változójú függvény. ϕ stacionárius pontja:

$$\frac{\partial \phi}{\partial z_1} = s_2 - 4z_1 = 0$$

$$\frac{\partial \phi}{\partial z_2} = s_1 - 4z_2 + 1 = 0 \, .$$

Mivel mindkét jobb oldal megvalósítható, így az optimum

$$z_1 = \frac{s_2}{4} \quad \text{és} \quad z_2 = \frac{s_1 + 1}{4} \, .$$

A fixpontban:

$$s_1 = \frac{s_2}{4} \quad \text{és} \quad s_2 = \frac{s_1 + 1}{4} \, ,$$

melyből az egyetlen megoldás:

$$s_1 = \frac{1}{15} \quad \text{és} \quad s_2 = \frac{4}{15} \, .$$

25.2.3. A Kuhn-Tucker-feltételek megoldása

Tegyük fel, hogy minden k-ra

$$S_k = \{s_k | \mathbf{g}_k(s_k) \geq \mathbf{0}\} \, ,$$

ahol $\mathbf{g}_k : \mathbb{R}^{n_k} \to \mathbb{R}^{m_k}$ az $O_k \supseteq S_k$ nyílt halmazon folytonosan differenciálható, vektor változójú és vektor értékű függvény. Tegyük fel továbbá, hogy az f_k függvény s_k szerint folytonosan parciálisan deriválható O_k-n minden k-ra, tetszőleges rögzített $s_1, \ldots, s_{k-1}, s_{k+1}, \ldots, s_N$ esetén.

Ha $\mathbf{s}^\star = (s_1^\star, \ldots, s_N^\star)$ egyensúly, akkor minden k-ra $s_k^\star$ optimális megoldása a következő feladatnak:

$$\begin{array}{rcl} f_k(s_1^\star, \ldots, s_{k-1}^\star, s_k, s_{k+1}^\star, \ldots, s_N^\star) & \to & \max \\ \mathbf{g}_k(s_k) & \geq & \mathbf{0} \, . \end{array} \tag{25.8}$$

Feltéve, hogy a Kuhn-Tucker regularitási feltételek s_k esetén teljesülnek, a megoldásnak teljesítenie kell a Kuhn-Tucker-féle szükséges feltételeket ($k = 1, 2, \ldots, N$):

$$\begin{array}{rcl} \mathbf{u}_k & \geq & \mathbf{0} \\ \mathbf{g}_k(s_k) & \geq & \mathbf{0} \\ \nabla_k f_k(\mathbf{s}) + \mathbf{u}_k^T \nabla_k \mathbf{g}_k(s_k) & = & \mathbf{0}^T \\ \mathbf{u}_k^T \mathbf{g}_k(s_k) & = & 0 \, , \end{array} \tag{25.9}$$

ahol $\mathbf{u}_k$ egy m_k komponensű oszlopvektor, $\mathbf{u}_k^T$ jelöli $\mathbf{u}_k$ transzponáltját, $\nabla_k f_k$ az f_k s_k szerinti gradiens függvénye (mint sorvektor), és $\nabla_k \mathbf{g}_k$ a $\mathbf{g}_k$ függvény Jacobi-függvénye.

25.5. tétel. *Ha $\mathbf{s}^\star$ egyensúly, akkor léteznek olyan $\mathbf{u}_k^\star$ vektorok, hogy (26.9) teljesül.*

A (26.9) relációi minden $k = 1, 2, \ldots, N$-re feltételek (általában nagy) rendszerét adja az ismeretlen s_k-ra és $\mathbf{u}_k$-ra. Ha létezik egyensúly, akkor az egyensúlynak teljesítenie kell (26.9)-et. Ha ráadásul minden k-ra $\mathbf{g}_k$ minden komponense szerint konkáv és f_k konkáv s_k szerint, akkor a Kuhn–Tucker-feltételek elégségesek is, tehát (26.9) minden megoldása egyensúly.

A (26.9) megoldásának számítási költsége (26.9) típusától, és a választott módszertől függ. Ha például (26.9) lineáris programozási feladat, melyet szimplex módszerrel oldunk meg, akkor a műveletek száma legrosszabb esetben exponenciális. Egyedi esetekben azonban a megoldás sokkal kevesebb művelettel is meghatározható.

25.7. példa. *Harmadik kétszemélyes játék.* Tekintsük ismét a 25.4 példa kétszemélyes játékát. Világos, hogy

$$S_1 = \{s_1 | s_1 \geq 0,\ 1 - s_1 \geq 0\}\ ,$$

$$S_2 = \{s_2 | s_2 \geq 0,\ 1 - s_2 \geq 0\}\ ,$$

amiből kapjuk, hogy

$$\mathbf{g}_1(s_1) = \begin{pmatrix} s_1 \\ 1 - s_1 \end{pmatrix} \text{ és } \mathbf{g}_2(s_2) = \begin{pmatrix} s_2 \\ 1 - s_2 \end{pmatrix}\ .$$

Deriválás után

$$\nabla_1 \mathbf{g}_1(s_1) = \begin{pmatrix} 1 \\ -1 \end{pmatrix},\ \ \nabla_2 \mathbf{g}_2(s_2) = \begin{pmatrix} 1 \\ -1 \end{pmatrix}\ ,$$

$$\nabla_1 f_1(s_1, s_2) = s_2 - 4s_1,\ \ \nabla_2 f_2(s_1, s_2) = s_1 - 4s_2 + 1\ ,$$

tehát a Kuhn-Tucker-feltételek a következő formában írhatóak fel:

$$\begin{aligned}
u_1^{(1)}, u_2^{(1)} &\geq 0 \\
s_1 &\geq 0 \\
s_1 &\leq 1 \\
s_2 - 4s_1 + u_1^{(1)} - u_2^{(1)} &= 0 \\
u_1^{(1)} s_1 + u_2^{(1)}(1 - s_1) &= 0 \\
u_1^{(2)}, u_2^{(2)} &\geq 0 \\
s_2 &\geq 0 \\
s_2 &\leq 1 \\
s_1 - 4s_2 + 1 + u_1^{(2)} - u_2^{(2)} &= 0 \\
u_1^{(2)} s_2 + u_2^{(2)}(1 - s_2) &= 0\ .
\end{aligned}$$

Vegyük észre, hogy f_1 konkáv s_1 szerint, f_2 konkáv s_2 szerint, és minden feltétel lineáris, tehát ennek a feltételrendszernek minden megoldása egyensúly. Módszeresen vizsgálva az egyes lehetőségek kombinációit, azt kapjuk, hogy

$$s_1 = 0,\ \ 0 < s_1 < 1,\ \ s_1 = 1\ ,$$

és

$$s_2 = 0, \quad 0 < s_2 < 1, \quad s_2 = 1 \, .$$

Könnyen látható, hogy egyetlen megoldás van:

$$u_1^{(1)} = u_1^{(2)} = u_2^{(1)} = u_2^{(2)} = 0 \, , \quad s_1 = \frac{1}{15}, \quad s_2 = \frac{4}{15} \, .$$

Túlcsordulás és többlet változókat bevezetve a Kuhn-Tucker-feltételek átírhatóak, mint egy nemnegatív rendszer. A nemnegativitási feltételek elhagyhatók, ha a változókat úgy tekintjük, mint valamely új változók négyzeteit, így a végeredmény egy plusz feltételek nélküli, (általában) nemlineáris egyenletrendszer. Számos numerikus módszer áll rendelkezésre az ilyen egyenletrendszerek megoldására.

25.2.4. Visszavezetés optimumszámítási feladatra

Tegyük fel, hogy az előző alfejezet (26.9) feltételei teljesülnek. Tekintsük a következő optimumszámítási feladatot, ahol $k = 1, 2, \ldots, N$:

$$\begin{array}{rcl} \sum_{k=1}^{N} \mathbf{u}_k^T \mathbf{g}_k(s_k) & \rightarrow & \min \\ \mathbf{u}_k & \geq & \mathbf{0} \\ \mathbf{g}_k(s_k) & \geq & \mathbf{0} \\ \nabla_k f_k(\mathbf{s}) + \mathbf{u}_k^T \nabla_k \mathbf{g}_k(s_k) & = & 0 \, . \end{array} \tag{25.10}$$

A két első feltétel miatt a célfüggvény nem negatív, így az optimális érték sem negatív. Ebből következik, hogy (26.9)-nek pontosan akkor van megengedett megoldása, ha (26.10)-ben a célfüggvény zéró. Ebben az esetben bármely optimális megoldás teljesíti (26.9)-t.

25.6. tétel. *Egy N személyes játéknak csak akkor van egyensúlya, ha* (26.10)*-ben a célfüggvény optimális értéke nulla. Ha ráadásul $\mathbf{g}_k$ minden komponense szerint konkáv, és f_k s_k szerint konkáv minden k-ra, akkor* (26.10) *minden optimális megoldása egyensúly.*

Tehát egy N személyes játék egyensúlyának meghatározása visszavezethető a (26.10) (általában) nemlineáris optimumszámítási feladat megoldására. Bármely nemlineáris programozási módszer használható ennek a problémának a megoldására.

(26.10) megoldásának számítási költsége (26.10) típusától, és a választott módszertől függ. Például, ha (26.10) egy lineáris programozási feladat, melyet a szimplex módszerrel oldunk meg, akkor a műveletek maximális száma exponenciális. Egyedi esetekben azonban a megoldás sokkal kevesebb művelettel

is meghatározható.

25.8. példa. *Negyedik kétszemélyes játék.* A 26.7 példa esetén az optimumszámítási feladat a következő formában írható fel:

$$\begin{aligned} u_1^{(1)}s_1 + u_2^{(1)}(1-s_1) + u_1^{(2)}s_2 + u_2^{(2)}(1-s_2) &\to \min \\ u_1^{(1)}, u_1^{(2)}, u_2^{(1)}, u_2^{(2)} &\geq 0 \\ s_1 &\geq 0 \\ s_1 &\leq 1 \\ s_2 &\geq 0 \\ s_2 &\leq 1 \\ s_2 - 4s_1 + u_1^{(1)} - u_2^{(1)} &= 0 \\ s_1 - 4s_2 + 1 + u_1^{(2)} - u_2^{(2)} &= 0. \end{aligned}$$

Vegyük észre, hogy az $u_1^{(1)} = u_1^{(2)} = u_2^{(1)} = u_2^{(2)} = 0$, $s_1 = 1/15$ és $s_2 = 4/15$ megoldás megengedett, a célfüggvény értéke nulla, így egyben optimális megoldás is. Ebből következik, hogy megoldása (26.9)-nek, így a 25.6. tétel miatt egyensúly.

Véges játékok kevert bővítése

Korábban láttuk, hogy egy véges játéknak nem feltétlenül van egyensúlya. Még ha egy véges játéknak van is egyensúlya, és sokszor játsszuk le az adott játékot, akkor is a játékosok szeretnek bevezetni némi véletlenszerűséget az akcióikba, abból a célból, hogy a többi játékost összezavarják, illetve azért, hogy keressenek egy sztochasztikus értelemben vett egyensúlyt. Ez a gondolat úgy modellezhető, hogy a játékosok stratégiáit valószínűség eloszlásokként vezetjük be, és a várható kifizetések lesznek a kifizetőfüggvények.

A 25.1. alfejezet jelöléseit megtartjuk: N játékosunk van, az $S_k = \{s_k^{(1)}, \ldots, s_k^{(n_k)}\}$ halmaz a $\mathcal{P}_k$ játékos véges stratégiahalmaza. Ennek a véges játéknak a ***kevert bővítésében*** minden játékos egy – a saját stratégiahalmazán értelmezett – diszkrét valószínűségeloszlást vesz, továbbá S_k elemeit a játék minden lejátszásában az adott diszkrét eloszlás szerint választja. Tehát $\mathcal{P}_k$ új stratégiahalmaza:

$$\overline{S}_k = \{\mathbf{x}_k | \mathbf{x}_k = (x_k^{(1)}, \ldots, x_k^{(n_k)}),\ \sum_{i=1}^{n_k} x_k^{(i)} = 1,\ x_k^{(i)} \geq 0 \text{ minden } i\text{-re}\}\ , \tag{25.11}$$

mely halmaz elemei n_k komponensű valószínűségi vektorok. $\mathcal{P}_k$ új kifizető függvénye várható érték függvény:

$$\overline{f}_k(\mathbf{x}_1, \ldots, \mathbf{x}_N) = \sum_{i_1=1}^{n_1} \sum_{i_2=1}^{n_2} \ldots \sum_{i_N=1}^{n_N} f_k(s_1^{(i_1)}, s_2^{(i_2)}, \ldots, s_N^{(i_N)}) x_1^{(i_1)} x_2^{(i_2)} \ldots x_N^{(i_N)}\ . \tag{25.12}$$

Vegyük észre, hogy az $\mathbf{x}_k = \mathbf{e}_k$ természetes bázisvektor választással az eredeti „tiszta" stratégiához $(s_k^{(i)})$ tartozó kifizetés kapható meg. A kevert bővítéssel kapott játék folytonos játék, és a 25.2. tétel szerint van legalább egy egyensúlya. Tehát ha adott egy véges játék, melynek nincs egyensúlya, akkor a kevert bővítésének mindig van legalább egy egyensúlya, mely egyensúly az előző alfejezetben ismertetett módszerekkel megkapható.

25.9. példa. *Ötödik kétszemélyes játék.* Tekintsünk egy kétszemélyes játékot ($N = 2$), ahol a 25.1. alfejezetben bevezetett $\mathbf{A}^{(1)}$ és $\mathbf{A}^{(2)}$ mátrixok (i,j) elemei $f_1(s_1^{(i)}, s_2^{(j)})$ és $f_2(s_1^{(i)}, s_2^{(j)})$. Ebben a speciális esetben

$$\overline{f}_k(\mathbf{x}_1, \mathbf{x}_2) = \sum_{i=1}^{n_1}\sum_{j=1}^{n_2} a_{ij}^{(k)} x_i^{(1)} x_j^{(2)} = \mathbf{x}_1^T \mathbf{A}^{(k)} \mathbf{x}_2 \quad (k = 1,2)\ . \tag{25.13}$$

Az $\overline{S}_k$ feltételei a következő formába írhatók át:

$$\begin{aligned} x_k^{(i)} &\geq 0 \qquad (i = 1,2,\ldots,n_k)\ , \\ -1 + \sum_{i=1}^{n_k} x_k^{(i)} &\geq 0\ , \\ 1 - \sum_{i=1}^{n_k} x_k^{(i)} &\geq 0\ . \end{aligned}$$

Tehát választhatjuk $\mathbf{g}_k$-t a következőképpen:

$$\mathbf{g}_k(\mathbf{x}_k) = \begin{pmatrix} x_k^{(1)} \\ \vdots \\ x_k^{(n_k)} \\ \sum_{i=1}^{n_k} x_k^{(i)} - 1 \\ -\sum_{i=1}^{n_k} x_k^{(i)} + 1 \end{pmatrix}\ . \tag{25.14}$$

A (26.10)-ben adott optimumszámítási feladat a következő feladatra egyszerűsödik:

$$\begin{aligned} \textstyle\sum_{k=1}^{2}[\sum_{i=1}^{n_k} u_k^{(i)} x_k^{(i)} + u_k^{(n_k+1)}(\sum_{j=1}^{n_k} x_k^{(j)} - 1) + u_k^{(n_k+2)}(-\sum_{j=1}^{n_k} x_k^{(i)} + 1)] \quad &\to \quad \min \\ u_k^{(i)} \quad &\geq \quad 0 \quad (1 \leq i \leq n_k + 2) \\ x_k^{(i)} \quad &\geq \quad 0 \quad (1 \leq i \leq n_k) \\ \mathbf{1}^T \mathbf{x}_k \quad &= \quad 1 \\ \mathbf{x}_2^T (\mathbf{A}^{(1)})^T + \mathbf{v}_1^T + (u_1^{(n_1+1)} - u_1^{(n_1+2)}) \mathbf{1}_1^T \quad &= \quad \mathbf{0}_1^T \\ \mathbf{x}_1^T (\mathbf{A}^{(2)}) + \mathbf{v}_2^T + (u_2^{(n_2+1)} - u_2^{(n_2+2)}) \mathbf{1}_2^T \quad &= \quad \mathbf{0}_2^T\ , \end{aligned} \tag{25.15}$$

ahol $\mathbf{v}_k^T = (u_k^{(1)}, \ldots . u_k^{(n_k)})$, $\mathbf{1}_k^T = (1^{(1)}, \ldots, 1^{(n_k)})$ és $\mathbf{0}_k^T = (0^{(1)}, \ldots, 0^{(n_k)})$, $k = 1,2$.

Vegyük észre, hogy a fenti feladat egy kvadratikus programozási feladat.

A számítási költség a választott módszertől függ. Azt is vegyük észre, hogy a fenti probléma általában nem konvex, így lehetséges, hogy az optimum keresése során beragadunk egy lokális optimumba.

Bimátrix-játékok

A kétszemélyes véges játékok kevert kiterjesztéseit ***bimátrix-játékoknak*** nevezzük. A 25.9 példában már vizsgáltunk ilyen játékot. A jelölés egységesítése érdekében a következő egyszerűsítő jelöléseket vezetjük be:

$$\mathbf{A} = \mathbf{A}^{(1)}, \mathbf{B} = \mathbf{A}^{(2)}, \mathbf{x} = \mathbf{x}_1, \mathbf{y} = \mathbf{x}_2, m = n_1 \ \text{ és } \ n = n_2 \ .$$

A következőkben azt mutatjuk meg, hogy a (26.15) feladat átírható olyan kvadratikus programozási feladattá, melyben csak lineáris feltételek vannak.

Tekintsük először a célfüggvényt. Legyenek

$$\alpha = u_1^{(m+2)} - u_1^{(m+1)}, \ \text{ és } \ \beta = u_2^{(n+2)} - u_2^{(n+1)} \ ,$$

ekkor a célfüggvény a következő formába írható át:

$$\mathbf{v}_1^T\mathbf{x} + \mathbf{v}_2^T\mathbf{y} - \alpha(\mathbf{1}_m^T\mathbf{x} - 1) - \beta(\mathbf{1}_n^T\mathbf{y} - 1) \ . \tag{25.16}$$

(26.15)-ben az utolsó két feltétel szintén egyszerűsödik:

$$\begin{aligned} \mathbf{y}^T\mathbf{A}^T + \mathbf{v}_1^T - \alpha\mathbf{1}_m^T &= \mathbf{0}_m^T \ , \\ \mathbf{x}^T\mathbf{B} + \mathbf{v}_2^T - \beta\mathbf{1}_n^T &= \mathbf{0}_n^T \ , \end{aligned}$$

amiből következik:

$$\mathbf{v}_1^T = \alpha\mathbf{1}_m^T - \mathbf{y}^T\mathbf{A}^T \ \text{ és } \ \mathbf{v}_2^T = \beta\mathbf{1}_n^T - \mathbf{x}^T\mathbf{B} \ . \tag{25.17}$$

Mivel

$$\mathbf{1}_m^T\mathbf{x} = \mathbf{1}_n^T\mathbf{y} = 1 \ ,$$

felírhatjuk a célfüggvényt egy újabb formában:

$$\begin{aligned} &(\alpha\mathbf{1}_m^T - \mathbf{y}^T\mathbf{A}^T)\mathbf{x} + (\beta\mathbf{1}_n^T - \mathbf{x}^T\mathbf{B})\mathbf{y} - \alpha(\mathbf{1}_m^T\mathbf{x} - 1) - \beta(\mathbf{1}_n^T\mathbf{y} - 1) \\ &\qquad = \alpha + \beta - \mathbf{x}^T(\mathbf{A} + \mathbf{B})\mathbf{y} \ . \end{aligned}$$

Tehát a következő kvadratikus programozási feladatot kapjuk:

$$\begin{aligned} \mathbf{x}^T(\mathbf{A} + \mathbf{B})\mathbf{y} - \alpha - \beta &\ \to \ \max \\ \mathbf{x} &\ \geq \ \mathbf{0} \\ \mathbf{y} &\ \geq \ \mathbf{0} \\ \mathbf{1}_m^T\mathbf{x} &\ = \ 1 \\ \mathbf{1}_n^T\mathbf{y} &\ = \ 1 \\ \mathbf{A}\mathbf{y} &\ \leq \ \alpha\mathbf{1}_m \\ \mathbf{B}^T\mathbf{x} &\ \leq \ \beta\mathbf{1}_n \ , \end{aligned} \tag{25.18}$$

ahol a két utolsó feltétel a $\mathbf{v}_1$, $\mathbf{v}_2$ vektorok nemnegativitásából és (26.17)-ből következik.

25.7. tétel. *Az $\mathbf{x}^\star$, $\mathbf{y}^\star$ vektorpár pontosan akkor egyensúlya az $(\mathbf{A}, \mathbf{B})$ bimátrix-játéknak, ha valamilyen $\alpha^\star$-ra és $\beta^\star$-ra $(\mathbf{x}^\star, \mathbf{y}^\star, \alpha^\star, \beta^\star)$ optimális megoldása a (26.18) feladatnak. Ekkor az optimumban a célfüggvény értéke nulla.*

Ez kvadratikus programozási feladat, ahol a számítási költség a választott módszertől függ. Általában nem konvex a feladat, így benne ragadhatunk egy lokális optimumban. Mivel tudjuk, hogy a globális optimumban a célfüggvény értéke nulla, így ellenőrizni tudjuk az optimalitást. Ha $\mathbf{A} + \mathbf{B}$ negatív szemidefinit, akkor a feladat konvex, tehát minden lokális optimum globális is.

25.10. példa. *Első bimátrix-játék.* Válasszuk $\mathbf{A}$-t és $\mathbf{B}$-t a következőképpen:

$$\mathbf{A} = \begin{pmatrix} 2 & -1 \\ -1 & 1 \end{pmatrix}$$

és

$$\mathbf{B} = \begin{pmatrix} 1 & -1 \\ -1 & 2 \end{pmatrix}.$$

Ekkor

$$\mathbf{A} + \mathbf{B} = \begin{pmatrix} 3 & -2 \\ -2 & 3 \end{pmatrix},$$

tehát (26.18) a következő formát ölti:

$$\begin{aligned} 3x_1y_1 - 2x_1y_2 - 2x_2y_1 + 3x_2y_2 - \alpha - \beta &\to \max \\ x_1, x_2, y_1, y_2 &\geq 0 \\ x_1 + x_2 &= 1 \\ y_1 + y_2 &= 1 \\ 2y_1 - y_2 &\leq \alpha \\ -y_1 + y_2 &\leq \alpha \\ x_1 - x_2 &\leq \beta \\ -x_1 + 2x_2 &\leq \beta\,, \end{aligned}$$

ahol $\mathbf{x} = (x_1, x_2)^T$ és $\mathbf{y} = (y_1, y_2)^T$.

A 25.7. tételből tudjuk, hogy az optimális célfüggvényérték nulla, így minden megengedett megoldás, melyre a célfüggvény értéke nulla, szükségszerűen optimális.

Könnyen látható, hogy

$$\mathbf{x} = \begin{pmatrix}1\\0\end{pmatrix},\ \mathbf{y} = \begin{pmatrix}0\\1\end{pmatrix},\ \alpha = 2, \beta = 1\ ,$$
$$\mathbf{x} = \begin{pmatrix}0\\1\end{pmatrix},\ \mathbf{y} = \begin{pmatrix}1\\0\end{pmatrix},\ \alpha = 1,\ \beta = 2\ ,$$
$$\mathbf{x} = \begin{pmatrix}0.6\\0.4\end{pmatrix},\ \mathbf{y} = \begin{pmatrix}0.4\\0.6\end{pmatrix},\ \alpha = 0.2,\ \beta = 0.2$$

mind optimumok, tehát mindegyik meghatároz egy egyensúlyt.

Alkalmazhatjuk (26.9)-et egyensúly keresésre. Ahelyett, hogy megoldjuk a (26.18) optimumszámítási feladatot, megoldjuk az (26.9) feltételrendszert. A bimátrix-játékok esetén a (26.9) feladat a következő formára egyszerűsödik:

$$\begin{aligned}
\mathbf{x}^T\mathbf{A}\mathbf{y} &= \alpha \\
\mathbf{x}^T\mathbf{B}\mathbf{y} &= \beta \\
\mathbf{A}\mathbf{y} &\leq \alpha\mathbf{1}_m \\
\mathbf{B}^T\mathbf{x} &\leq \beta\mathbf{1}_n \\
\mathbf{x} &\geq \mathbf{0}_m \\
\mathbf{y} &\geq \mathbf{0}_n \\
\mathbf{1}_m^T\mathbf{x} = \mathbf{1}_n^T\mathbf{y} &= 1\ ,
\end{aligned} \tag{25.19}$$

mely feltételrendszer a kvadratikus programozási feladatnál látottakkal analóg módon vezethető le.

A (26.19) feladat számítási költsége a választott módszertől függ.

25.11. példa. *Második bimátrix-játék.* Tekintsük ismét a 25.10 példát. Helyettesítsük a (25.19) feltételrendszer első és a második feltétele alapján α-t és β-t a harmadik és a negyedik feltételbe, ekkor

$$\begin{aligned}
2y_1 - y_2 &\leq 2x_1y_1 - x_1y_2 - x_2y_1 + x_2y_2 \\
-y_1 + y_2 &\leq 2x_1y_1 - x_1y_2 - x_2y_1 + x_2y_2 \\
x_1 - x_2 &\leq x_1y_1 - x_1y_2 - x_2y_1 + 2x_2y_2 \\
-x_1 + 2x_2 &\leq x_1y_1 - x_1y_2 - x_2y_1 + 2x_2y_2 \\
x_1, x_2, y_1, y_2 &\geq 0 \\
x_1 + x_2 = y_1 + y_2 &= 1\ .
\end{aligned}$$

Könnyen látható, hogy a 25.10. példában kapott megoldások kielégítik a fenti feltételrendszert, tehát egyensúlyok.

A bimátrix-játék egyensúlyi feladatát átírhatjuk kevert változós feltételrendszerbe is. Tegyük fel, hogy $\mathbf{A}$ és $\mathbf{B}$ minden eleme 0 és 1 között van. Ez a feltevés nem túl erős, hiszen lineáris transzformációk használatával

$$\overline{\mathbf{A}} = a_1\mathbf{A} + b_1\mathbf{1} \ \text{ és } \ \overline{\mathbf{B}} = a_2\mathbf{B} + b_2\mathbf{1}\ ,$$

ahol $a_1, a_2 > 0$, $\mathbf{1}$ a csupa egyesekből álló $m \times n$-es mátrix. Ekkor az egyensúly nem változik, és – megfelelő a_1, b_1, a_2 és b_2 értékek választásával – az $\overline{\mathbf{A}}$ és $\overline{\mathbf{B}}$ mátrixok minden eleme a $[0,1]$ intervallumba esik.

25.8. tétel. *Az* $\mathbf{x}$, $\mathbf{y}$ *vektorpár pontosan akkor egyensúly, ha valamilyen* α, β *számokra és* $\mathbf{u}, \mathbf{v}$ *nulla-egy vektorokra teljesül, hogy*

$$\begin{array}{rcccccl} 0 & \le & \alpha\mathbf{1}_m - \mathbf{A}\mathbf{y} & \le & \mathbf{1}_m - \mathbf{u} & \le & \mathbf{1}_m - \mathbf{x} \\ 0 & \le & \beta\mathbf{1}_n - \mathbf{B}^T\mathbf{x} & \le & \mathbf{1}_n - \mathbf{v} & \le & \mathbf{1}_n - \mathbf{y} \\ & & & & \mathbf{x} & \ge & \mathbf{0}_m \\ & & & & \mathbf{y} & \ge & \mathbf{0}_n \\ & & \mathbf{1}_m^T\mathbf{x} & = & \mathbf{1}_n^T\mathbf{y} & = & 1\,. \end{array} \tag{25.20}$$

Bizonyítás. Először tegyük fel, hogy $\mathbf{x}$, $\mathbf{y}$ vektorpár egyensúly, ekkor valamilyen α-ra, β-ra (26.19) teljesül. Legyen

$$u_i = \begin{cases} 1, & \text{ha } x_i > 0\,, \\ 0, & \text{ha } x_i = 0\,, \end{cases} \quad \text{és} \quad v_j = \begin{cases} 1, & \text{ha } y_j > 0\,, \\ 0, & \text{ha } y_j = 0\,. \end{cases}$$

Mivel az $\mathbf{A}$ és $\mathbf{B}$ mátrixok elemei $[0,1]$-ből valók, így $\alpha = \mathbf{x}^T\mathbf{A}\mathbf{y}$ és $\beta = \mathbf{x}^T\mathbf{B}\mathbf{y}$ szintén nulla és egy közöttiek. Vegyük észre, hogy

$$0 = \mathbf{x}^T(\alpha\mathbf{1}_m - \mathbf{A}\mathbf{y}) = \mathbf{y}^T(\beta\mathbf{1}_n - \mathbf{B}^T\mathbf{x})\,,$$

melyből következik, hogy (26.20) teljesül.

Most tegyük fel, hogy (26.20) teljesül. Ekkor

$$\mathbf{0} \le \mathbf{x} \le \mathbf{u} \le \mathbf{1}_m \quad \text{és} \quad \mathbf{0} \le \mathbf{y} \le \mathbf{v} \le \mathbf{1}_n\,.$$

Ha $u_i = 1$, akkor $\alpha - \mathbf{e}_i^T\mathbf{A}\mathbf{y} = 0$, és ha $u_i = 0$, akkor $x_i = 0$. Tehát

$$\mathbf{x}^T(\alpha\mathbf{1}_m - \mathbf{A}\mathbf{y}) = 0\,,$$

melyből következik, hogy $\alpha = \mathbf{x}^T\mathbf{A}\mathbf{y}$. A $\beta = \mathbf{x}^T\mathbf{B}\mathbf{y}$ egyenlőség érvényessége hasonlóan mutatható meg, így (26.19) teljesül, tehát az $\mathbf{x}, \mathbf{y}$ vektorpár egyensúly. ∎

A (26.20) feladat számítási költsége a választott módszertől függ.

25.12. példa. *Harmadik bimátrix-játék.* A 25.10 példában bevezetett bimátrix-játék esetén (26.20) a következő formát ölti:

$$\begin{array}{rcccccl} 0 & \le & \alpha - 2y_1 + y_2 & \le & 1 - u_1 & \le & 1 - x_1 \\ 0 & \le & \alpha + y_1 - y_2 & \le & 1 - u_2 & \le & 1 - x_2 \\ 0 & \le & \beta - x_1 + x_2 & \le & 1 - v_1 & \le & 1 - y_1 \\ 0 & \le & \beta + x_1 - 2x_2 & \le & 1 - v_2 & \le & 1 - y_2 \\ & & x_1 + x_2 & = & y_1 + y_2 & = & 1 \\ & & & & x_1, x_2, y_1, y_2 & \ge & 0 \\ & & & & u_1, u_2, v_1, v_2 & \in & \{0,1\}\,. \end{array}$$

Vegyük észre, hogy a 25.10 példában adott három megoldás teljesíti a fenti feltételrendszert az $\mathbf{u} = (1,0)$, $\mathbf{v} = (0,1)$, $\mathbf{u} = (0,1)$, $\mathbf{v} = (1,0)$, és $\mathbf{u} = (1,1)$, $\mathbf{v} = (1,1)$ vektorpárokkal.

Mátrixjátékok

Azokat a bimátrix-játékokat, ahol $\mathbf{B} = -\mathbf{A}$, ***mátrixjátékoknak*** nevezzük, és egy $\mathbf{A}$ mátrixszal jelöljük. Az ilyen játékokra néha $\mathbf{A}$ *mátrixjátékként* fogunk hivatkozni. Mivel $\mathbf{A}+\mathbf{B} = \mathbf{0}$, a (26.18)-bani kvadratikus programozási feladat lineáris:

$$\begin{array}{rcl} \alpha+\beta & \to & \min \\ \mathbf{x} & \geq & \mathbf{0} \\ \mathbf{y} & \geq & \mathbf{0} \\ \mathbf{1}_m\mathbf{x} & = & 1 \\ \mathbf{1}_n\mathbf{y} & = & 1 \\ \mathbf{A}\mathbf{y} & \leq & \alpha\mathbf{1}_m \\ \mathbf{A}^T\mathbf{x} & \geq & -\beta\mathbf{1}_n\,. \end{array} \tag{25.21}$$

A fenti feladatból látható, hogy az egyensúlyok halmaza konvex poliéder. Vegyük észre, hogy $(\mathbf{x},\beta)$ és $(\mathbf{y},\alpha)$ szétválasztható, amiből a következő eredmény vezethető le.

25.9. tétel. *Az $\mathbf{x}^\star$, $\mathbf{y}^\star$ vektorpár pontosan akkor egyensúlya az $\mathbf{A}$ mátrixjátéknak, ha valamilyen $\alpha^\star$-ra, $\beta^\star$-ra $(\mathbf{x}^\star,\beta^\star)$ és $(\mathbf{y}^\star,\alpha^\star)$ optimális megoldásai a következő lineáris programozási feladatpárnak:*

$$\begin{array}{rcl} \alpha & \to & \min \\ \mathbf{y} & \geq & \mathbf{0}_n \\ \mathbf{1}_n^T\mathbf{y} & = & 1 \\ \mathbf{A}\mathbf{y} & \leq & \alpha\mathbf{1}_m \end{array} \qquad \begin{array}{rcl} \beta & \to & \min \\ \mathbf{x} & \geq & \mathbf{0}_m \\ \mathbf{1}_m^T\mathbf{x} & = & 1 \\ \mathbf{A}^T\mathbf{x} & \geq & -\beta\mathbf{1}_n\,. \end{array} \tag{25.22}$$

Vegyük észre, hogy az optimumban $\alpha+\beta = 0$. Az α optimális értékét a mátrixjáték ***értékének*** nevezzük.

Ha a szimplex módszert alkalmazzuk (26.22) megoldására, akkor a műveletek száma exponenciális. Polinomiális algoritmussal (mint amilyen a belső pont módszer) a műveletek száma csak polinomiális.

25.13. példa. *Első mátrixjáték.* Tekintsük a következő mátrixjátékot:

$$\mathbf{A} = \begin{pmatrix} 2 & 1 & 0 \\ 2 & 0 & 3 \\ -1 & 3 & 3 \end{pmatrix}\,.$$

A (26.22)-t erre a feladatra felírva:

$$\begin{array}{rcl} \alpha & \to & \min \\ y_1, y_2, y_3 & \geq & 0 \\ y_1 + y_2 + y_3 & = & 1 \\ 2y_1 + y_2 - \alpha & \leq & 0 \\ 2y_1 + 3y_3 - \alpha & \leq & 0 \\ -y_1 + 3y_2 + 3y_3 - \alpha & \leq & 0 \end{array} \qquad \begin{array}{rcl} \beta & \to & \min \\ x_1, x_2, x_3 & \geq & 0 \\ x_1 + x_2 + x_3 & = & 1 \\ 2x_1 + 2x_2 - x_3 + \beta & \geq & 0 \\ x_1 + 3x_3 + \beta & \geq & 0 \\ 3x_2 + 3x_3 + \beta & \geq & 0\,. \end{array}$$

A szimplex módszerrel megkaphatjuk a fenti feladatpár megoldását: $\alpha = 9/7$, $\mathbf{y} = (3/7, 3/7, 1/7)$, $\beta = -9/7$, és $\mathbf{x} = (4/7, 4/21, 5/21)$.

A (26.21) megoldását úgy is megkaphatjuk, hogy lineáris feltételek bizonyos halmazának megengedett megoldását keressük meg. Mivel (26.21)-ben az optimumban $\alpha + \beta = 0$, $\mathbf{x}$, $\mathbf{y}$ vektorok és α, β skalárok pontosan akkor alkotnak optimális megoldást, ha

$$\begin{array}{rcl} \mathbf{x}, \mathbf{y} & \geq & \mathbf{0} \\ \mathbf{1}_m^T \mathbf{x} & = & 1 \\ \mathbf{1}_n^T \mathbf{y} & = & 1 \\ \mathbf{A}\mathbf{y} & \leq & \alpha \mathbf{1}_m \\ \mathbf{A}^T \mathbf{x} & \geq & \alpha \mathbf{1}_n\,. \end{array} \tag{25.23}$$

A (26.23) megoldásához a szimplex módszer első fázisa szükséges, mely a legkedvezőtlenebb esetben exponenciális számú műveletet igényel. A gyakorlatban általában sokkal kevesebb művelet szükséges.

25.14. példa. *Második mátrixjáték.* Nézzük megint a 25.13 példában bevezetett mátrixjátékot. Ha erre a játékra (26.23)-at felírjuk, akkor a következőt kapjuk:

$$\begin{array}{rcl} x_1, x_2, x_3, y_1, y_2, y_3 & \geq & 0 \\ x_1 + x_2 + x_3 = y_1 + y_2 + y_3 & = & 1 \\ 2y_1 + y_2 & \leq & \alpha \\ 2y_1 + 3y_3 & \leq & \alpha \\ -y_1 + 3y_2 + 3y_3 & \leq & \alpha \\ 2x_1 + 2x_2 - x_3 & \geq & \alpha \\ x_1 + 3x_3 & \geq & \alpha \\ 3x_2 + 3x_3 & \geq & \alpha\,. \end{array}$$

Könnyen látható, hogy $\alpha = 9/7$, $\mathbf{x} = (4/7, 4/21, 5/21)^T$, $\mathbf{y} = (3/7, 3/7, 1/7)^T$ kielégíti a (26.9) feltételrendszert, tehát az $\mathbf{x}$, $\mathbf{y}$ vektorpár egyensúly.

25.2.5. A fiktív lejátszás módszere

Tekintsünk most egy $\mathbf{A}$ mátrixjátékot. Az ezen alfejezetben tárgyalt módszer fő gondolata az, hogy lépésenként minden játékos meghatározza a saját legjobb válasz tiszta stratégiáját a másik játékos – az előzőekben választott – stratégiáinak átlaga mellett. Formálisan a módszer a következőképpen írható fel.

Legyen $\mathbf{x}_1$ a $\mathcal{P}_1$ játékos kezdeti (kevert) stratégiája. Válasszuk úgy $\mathbf{y}_1 = \mathbf{e}_{j_1}$-t, hogy teljesüljön

$$\mathbf{x}_1^T \mathbf{A} \mathbf{e}_{j_1} = \min_j \{\mathbf{x}_1^T \mathbf{A} \mathbf{e}_j\} \; . \tag{25.24}$$

Bármely további $k \geq 2$ lépéskor legyen

$$\overline{\mathbf{y}}_{k-1} = \frac{1}{k-1}((k-2)\overline{\mathbf{y}}_{k-2} + \mathbf{y}_{k-1}) \; , \tag{25.25}$$

és válasszuk $\mathbf{x}_k = \mathbf{e}_{i_k}$-t úgy, hogy teljesüljön

$$\mathbf{e}_{i_k}^T \mathbf{A} \overline{\mathbf{y}}_{k-1} = \max_i \{\mathbf{e}_i^T \mathbf{A} \overline{\mathbf{y}}_{k-1}\} \; . \tag{25.26}$$

Ekkor legyen

$$\overline{\mathbf{x}}_k = \frac{1}{k}((k-1)\overline{\mathbf{x}}_{k-1} + \mathbf{x}_k) \; , \tag{25.27}$$

és válasszuk $\mathbf{y}_k = \mathbf{e}_{j_k}$-t úgy, hogy

$$\overline{\mathbf{x}}_k^T \mathbf{A} \mathbf{e}_{j_k} = \min_j \{\overline{\mathbf{x}}_k^T \mathbf{A} \mathbf{e}_j\} \; . \tag{25.28}$$

A fenti általános lépés megismétlése ($k = 2, 3, \ldots$)-ra két sorozatot eredményez: $\{\overline{\mathbf{x}}_k\}$-t, és $\{\overline{\mathbf{y}}_k\}$-t. Ekkor a következő eredményt kapjuk:

25.10. tétel. *Az* $(\{\overline{\mathbf{x}}_k\}, \{\overline{\mathbf{y}}_k\})$ *sorozatpár tetszőleges torlódási pontja az* $\mathbf{A}$ *mátrixjáték egy egyensúlya.*

Bizonyítás. Mivel $\{\overline{\mathbf{x}}_k\}$ és $\{\overline{\mathbf{y}}_k\}$ valószínűségi vektorok, így korlátos, valós sorozatok, tehát van legalább egy torlódási pontjuk[3]. ■

Tegyük fel, hogy az $\mathbf{A}$ mátrix $m \times n$-es. (26.24)-ben mn szorzásra van szükségünk. (26.25)-ben és (26.27)-ben $m + n$ szorzás és osztás van. (26.26)-ban és (26.28)-ban mn szorzás van. Ha L iterációs lépést teszünk, akkor az osztások és szorzások száma:

$$mn + L[2(m+n) + 2mn] = \Theta(Lmn) \; .$$

[3] Nem minden egyensúly kapható meg ezzel a módszerrel, illetve van olyan egyensúlyi pont, amit csak akkor talál meg ez a módszer, ha az egyensúlyból indul. *A fordító.*

Az algoritmus pszeudokódja a következő (az algoritmusban $\varepsilon > 0$ a felhasználó által megválasztott hibatűrés).

FIKTÍV-LEJÁTSZÁS

1 $k = 1$
2 legyen j_1 olyan, hogy teljesüljön $\mathbf{x}_1^T \mathbf{A} \mathbf{e}_{j_1} = \min_j \{\mathbf{x}_1^T \mathbf{A} \mathbf{e}_j\}$
3 $\mathbf{y}_1 = \mathbf{e}_{j_1}$
4 $k = k + 1$
5 $\overline{\mathbf{y}}_{k-1} = \frac{1}{k-1}((k-2)\overline{\mathbf{y}}_{k-2} + \mathbf{y}_{k-1})$
6 legyen i_k olyan, hogy teljesüljön $\mathbf{e}_{i_k}^T \mathbf{A} \overline{\mathbf{y}}_{k-1} = \max_i \{\mathbf{e}_i^T \mathbf{A} \overline{\mathbf{y}}_{k-1}\}$
7 $\mathbf{x}_k = \mathbf{e}_{i_k}$
8 $\overline{\mathbf{x}}_k = \frac{1}{k}((k-1)\overline{\mathbf{x}}_{k-1} + \mathbf{x}_k)$
9 legyen j_k olyan, hogy teljesüljön $\overline{\mathbf{x}}_k^T \mathbf{A} \mathbf{e}_{j_k} = \min_j \{\overline{\mathbf{x}}_k^T \mathbf{A} \mathbf{e}_j\}$
10 $\mathbf{y}_k = \mathbf{e}_{j_k}$
11 **if** $||\overline{\mathbf{x}}_k - \overline{\mathbf{x}}_{k-1}|| < \varepsilon$ and $||\overline{\mathbf{y}}_{k-1} - \overline{\mathbf{x}}_{k-2}|| < \varepsilon$
12 $(\overline{\mathbf{x}}_k, \overline{\mathbf{y}}_{k-1})$ egyensúly
13 **else** folytassuk a 4-edik sorban

25.15. példa. *Harmadik mátrixjáték.* A fent tárgyalt módszert alkalmazzuk a 25.14 példában tárgyalt mátrixjátékra. A módszer kezdő állapota: $\mathbf{x}_1 = (1, 0, 0)^T$. 100 lépés után: $\overline{\mathbf{x}}_{101} = (0.446, 0.287, 0.267)^T$ és $\overline{\mathbf{y}}_{101} = (0.386, 0.436, 0.178)^T$. Ha ezeket az értékeket az egyensúlyhoz hasonlítjuk, akkor azt tapasztaljuk, hogy az eltérés kisebb, mint 0.126, tehát a módszer lassan konvergál.

25.2.6. Szimmetrikus mátrixjátékok

Az $\mathbf{A}$ mátrixjátékot, ahol $\mathbf{A}$ ferdén-szimmetrikus, ***szimmetrikus mátrixjátéknak*** nevezzük. Ebben az esetben $\mathbf{A}^T = -\mathbf{A}$ és a két lineáris programozási feladat (26.22)-ben megegyezik. Ebből következik, hogy $\alpha = \beta = 0$ (a játék értéke 0), és tetszőleges egyensúlyban a két játékos stratégiái megegyeznek. Tehát a következő eredmény adódik.

25.11. tétel. *Az $\mathbf{x}^\star$ vektor pontosan akkor egyensúlya az $\mathbf{A}$ szimmetrikus mátrixjátéknak, ha*

$$\begin{aligned} \mathbf{x} &\geq \mathbf{0} \\ \mathbf{1}^T \mathbf{x} &= 1 \\ \mathbf{A}\mathbf{x} &\leq \mathbf{0}\,. \end{aligned} \tag{25.29}$$

(26.29) megoldásához a szimplex módszer első fázisa szükséges, ahol a legkedvezőtlenebb esetben a műveletek száma exponenciális. A gyakorlatban azonban általában sokkal kisebb számú műveletre van szükség (26.29)

megoldásához.

25.16. példa. *Szimmetrikus mátrixjáték.* Tekintsük az $\mathbf{A} = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$ szimmetrikus mátrixjátékot. Ekkor (26.29) a következő formát ölti:

$$\begin{aligned} x_1, x_2 &\geq 0 \\ x_1 + x_2 &= 1 \\ x_2 &\leq 0 \\ -x_1 &\leq 0\,. \end{aligned}$$

Könnyen látható, hogy az egyetlen megoldás: $x_1 = 1$ és $x_2 = 0$, ami az első tiszta stratégia.

A következő fejezetben látni fogjuk, hogy egy lineáris programozási feladat ekvivalens egy szimmetrikus mátrixjáték egyensúlyi problémájával, tehát tetszőleges módszer, amely egy szimmetrikus mátrixjáték egyensúlyi problémájának megoldására alkalmas, alkalmas lineáris programozási feladat megoldására is, ezért az ilyen módszerek a szimplex módszer alternatíváiként szolgálnak. A következőkben azt mutatjuk meg, hogy a szimmetria nem túlságosan erős feltétel, mert tetszőleges mátrixjáték megfeleltethető egy vele ekvivalens szimmetrikus mátrixjátéknak.

Tekintsük az $\mathbf{A}$ mátrixjátékot, és szerkesszük meg a következő ferdénszimmetrikus $\mathbf{P}$ mátrixot:

$$\mathbf{P} = \begin{pmatrix} \mathbf{0}_{m\times m} & \mathbf{A} & -\mathbf{1}_m \\ -\mathbf{A}^T & \mathbf{0}_{n\times n} & \mathbf{1}_n \\ \mathbf{1}_m^T & -\mathbf{1}_n^T & 0 \end{pmatrix}\,.$$

Az $\mathbf{A}$ és $\mathbf{P}$ mátrixjátékok a következő értelemben ekvivalensek. Tegyük fel, hogy $\mathbf{A} > \mathbf{0}$, ami nem túl erős feltétel, hiszen $\mathbf{A}$ elemeihez egy megfelelő konstanst hozzáadva $\mathbf{A} > \mathbf{0}$ elérhető, és az egyensúlyok nem változnak.

25.12. tétel. *Legyen $\mathbf{P}$ szimmetrikus mátrixjáték, melyet $\mathbf{A}$ mátrixjátékból kaptunk, ekkor igaz a következő két állítás:*

1. *Ha $\mathbf{z} = \begin{pmatrix} \mathbf{u} \\ \mathbf{v} \\ \lambda \end{pmatrix}$ egyensúlyi stratégiája a $\mathbf{P}$ szimmetrikus mátrixjátéknak, akkor az $\mathbf{x} = (1/a)\mathbf{u}$, $\mathbf{y} = (1/a)\mathbf{v}$ vektorpár egyensúlya az $\mathbf{A}$ mátrixjátéknak, és az $\mathbf{A}$ mátrixjáték értéke: $v = \lambda/a$, ahol $a = (1-\lambda)/2$.*

2. *Ha az $\mathbf{x}, \mathbf{y}$ vektorpár egyensúlya az $\mathbf{A}$ mátrixjátéknak, és v az $\mathbf{A}$*

mátrixjáték értéke, akkor a

$$\mathbf{z} = \frac{1}{2+v}\begin{pmatrix}\mathbf{x}\\ \mathbf{y}\\ v\end{pmatrix}$$

vektor egyensúlya a $\mathbf{P}$ *szimmetrikus mátrixjátéknak.*

Bizonyítás. Először tegyük fel, hogy $\mathbf{z}$ egyensúly a $\mathbf{P}$ szimmetrikus mátrixjátékban (1. pont). Ekkor $\mathbf{u} \geq \mathbf{0}$, $\mathbf{v} \geq \mathbf{0}$, és $\mathbf{Pz} \leq \mathbf{0}$, tehát

$$\begin{array}{rcl} \mathbf{A}\mathbf{v} - \lambda \mathbf{1}_m & \leq & \mathbf{0} \\ -\mathbf{A}^T\mathbf{u} + \lambda \mathbf{1}_n & \leq & \mathbf{0} \\ \mathbf{1}_m^T\mathbf{u} - \mathbf{1}_n^T\mathbf{v} & \leq & 0\,. \end{array} \tag{25.30}$$

Először megmutatjuk, hogy $\lambda \in (0,1)$, tehát $a \neq 0$. Tegyük fel, hogy $\lambda = 1$, ekkor (mivel $\mathbf{z}$ valószínűségi vektor) $\mathbf{u} = \mathbf{0}_m$ és $\mathbf{v} = \mathbf{0}_n$, ami ellentmond (26.30) második egyenlőtlenségének. Ha $\lambda = 0$, akkor $\mathbf{1}_m^T\mathbf{u} + \mathbf{1}_n^T\mathbf{v} = 1$, és (26.30) harmadik egyenlőtlensége miatt $\mathbf{v}$-nek van legalább egy pozitív komponense, mely ellentmond (26.30) első egyenlőtlenségének.

Most megmutatjuk, hogy $\mathbf{1}_m^T\mathbf{u} = \mathbf{1}_n^T\mathbf{v}$. (26.30)-ból azt kapjuk, hogy

$$\begin{array}{rcl} \mathbf{u}^T\mathbf{A}\mathbf{v} - \lambda \mathbf{u}^T\mathbf{1}_m & \leq & 0\,, \\ -\mathbf{v}^T\mathbf{A}^T\mathbf{u} + \lambda \mathbf{v}^T\mathbf{1}_n & \leq & 0\,. \end{array}$$

A két egyenlőtlenséget összeadva kapjuk, hogy

$$\mathbf{v}^T\mathbf{1}_n - \mathbf{u}^T\mathbf{1}_m \leq 0\,.$$

Ezt (26.30) harmadik egyenlőtlenségével kombinálva kapjuk, hogy $\mathbf{1}_m^T\mathbf{u} - \mathbf{1}_n^T\mathbf{v} = 0$.

Legyen $a = (1-\lambda)/2 \neq 0$, ekkor $\mathbf{1}_m^T\mathbf{u} = \mathbf{1}_n^T\mathbf{v} = a$, így mind $\mathbf{x} = \mathbf{u}/a$, mind $\mathbf{y} = \mathbf{v}/a$ valószínűségi vektor, és (26.30)-ból következik, hogy:

$$\begin{array}{rcccl} \mathbf{A}^T\mathbf{x} & = & \frac{1}{a}\mathbf{A}^T\mathbf{u} & \geq & \frac{\lambda}{a}\mathbf{1}_n\,, \\ \mathbf{A}\mathbf{y} & = & \frac{1}{a}\mathbf{A}\mathbf{v} & \leq & \frac{\lambda}{a}\mathbf{1}_m\,. \end{array}$$

Legyenek $\alpha = \lambda/a$ és $\beta = -\lambda/a$, ekkor $\mathbf{x}, \mathbf{y}$ vektorpár megoldása (26.22)-nek és $\alpha + \beta = 0$, tehát $\mathbf{x}, \mathbf{y}$ vektorpár egyensúlya az $\mathbf{A}$ mátrixjátéknak.

A 2. pont hasonlóan látható be, itt nem részletezzük. ∎

25.2.7. Lineáris programozás és mátrixjátékok

Ebben az alfejezetben megmutatjuk, hogy egy lineáris programozási feladatot meg lehet oldani úgy, hogy egy szimmetrikus mátrixjáték egyensúlyi stratégiáit keressük meg. Tehát tetszőleges olyan módszer, mely alkalmas egy szimmetrikus mátrixjáték egyensúlyainak meghatározására, alkalmas a szimplex módszer kiváltására.

Tekintsük a következő primál-duál lineáris programozási feladatpárt:

$$\begin{array}{rcl} \mathbf{c}^T\mathbf{x} & \to & \max \\ \mathbf{x} & \geq & \mathbf{0} \\ \mathbf{A}\mathbf{x} & \leq & \mathbf{b} \end{array} \qquad\qquad \begin{array}{rcl} \mathbf{b}^T\mathbf{y} & \to & \min \\ \mathbf{y} & \geq & \mathbf{0} \\ \mathbf{A}^T\mathbf{y} & \geq & \mathbf{c}\,. \end{array} \tag{25.31}$$

Szerkesszük meg a következő ferdén-szimmetrikus mátrixot:

$$\mathbf{P} = \begin{pmatrix} \mathbf{0} & \mathbf{A} & -\mathbf{b} \\ -\mathbf{A}^T & \mathbf{0} & \mathbf{c} \\ \mathbf{b}^T & -\mathbf{c}^T & 0 \end{pmatrix}.$$

25.13. tétel. *Tegyük fel, hogy* $\mathbf{z} = \begin{pmatrix} \mathbf{u} \\ \mathbf{v} \\ \lambda \end{pmatrix}$ *egyensúlya a* $\mathbf{P}$ *szimmetrikus mátrixjátéknak, és* $\lambda > 0$*. Ekkor*

$$\mathbf{x} = \frac{1}{\lambda}\mathbf{v} \quad \text{és} \quad \mathbf{y} = \frac{1}{\lambda}\mathbf{u}$$

optimális megoldásai a (26.31) *primál-duál feladatpárnak (*$\mathbf{x}$ *a primálnak,* $\mathbf{y}$ *a duálnak).*

Bizonyítás. Ha $\mathbf{z}$ egyensúly, akkor $\mathbf{P}\mathbf{z} \leq \mathbf{0}$, azaz

$$\begin{array}{rcl} \mathbf{A}\mathbf{v} - \lambda\mathbf{b} & \leq & \mathbf{0} \\ -\mathbf{A}^T\mathbf{u} + \lambda\mathbf{c} & \leq & \mathbf{0} \\ \mathbf{b}^T\mathbf{u} - \mathbf{c}^T\mathbf{v} & \leq & \mathbf{0}\,. \end{array} \tag{25.32}$$

Mivel $\mathbf{z} \geq \mathbf{0}$ és $\lambda > 0$, így mind az $\mathbf{x} = (1/\lambda)\mathbf{v}$, mind az $\mathbf{y} = (1/\lambda)\mathbf{u}$ vektor nemnegatív. Osszuk el (26.32) első két egyenlőtlenségét λ-val, ekkor

$$\mathbf{A}\mathbf{x} \leq \mathbf{b} \quad \text{és} \quad \mathbf{A}^T\mathbf{y} \geq \mathbf{c}\,,$$

ahonnan következik, hogy $\mathbf{x}$ megengedett megoldása a primál feladatnak, és $\mathbf{y}$ megengedett megoldása a duál feladatnak. (26.32) harmadik egyenlőtlenségéből azt kapjuk, hogy

$$\mathbf{b}^T\mathbf{y} \leq \mathbf{c}^T\mathbf{x}\,.$$

Tudjuk azonban

$$\mathbf{b}^T\mathbf{y} \geq (\mathbf{x}^T\mathbf{A}^T)\mathbf{y} = \mathbf{x}^T(\mathbf{A}^T\mathbf{y}) \geq \mathbf{x}^T\mathbf{c} = \mathbf{c}^T\mathbf{x}\ ,$$

tehát $\mathbf{b}^T\mathbf{y} = \mathbf{c}^T\mathbf{x}$, melyből az következik, hogy a primál feladat célfüggvénye $\mathbf{x}$-ben és a duál feladat célfüggvénye $\mathbf{y}$-ban egyenlő. Ekkor az erős dualitási tétel miatt $\mathbf{x}$ optimális megoldása a primál feladatnak és $\mathbf{y}$ optimális megoldása a duál feladatnak. ■

25.17. példa. *Lineáris programozás.* Tekintsük a következő lineáris programozási feladatot:

$$\begin{aligned} x_1 + 2x_2 &\rightarrow \max \\ x_1 &\geq 0 \\ -x_1 + x_2 &\geq 1 \\ 5x_1 + 7x_2 &\leq 25\ . \end{aligned}$$

Először fel kell írnunk a feladatot mint primál feladatot. Vezessünk be két új változót:

$$x_2^+ = \begin{cases} x_2, & \text{ha } x_2 \geq 0\ , \\ 0 & \text{különben}\ , \end{cases}$$

$$x_2^- = \begin{cases} -x_2, & \text{ha } x_2 < 0\ , \\ 0 & \text{különben}\ . \end{cases}$$

és szorozzuk meg a második egyenlőtlenséget -1-gyel. Ekkor a következő feladatot kapjuk:

$$\begin{aligned} x_1 + 2x_2^+ - 2x_2^- &\rightarrow \max \\ x_1, x_2^+, x_2^- &\geq 0 \\ x_1 - x_2^+ + x_2^- &\leq -1 \\ 5x_1 + 7x_2^+ - 7x_2^- &\leq 25\ . \end{aligned}$$

Így

$$\mathbf{A} = \begin{pmatrix} 1 & -1 & 1 \\ 5 & 7 & -7 \end{pmatrix},\ \mathbf{b} = \begin{pmatrix} -1 \\ 25 \end{pmatrix},\ \mathbf{c}^T = (1, 2, -2)\ .$$

A $\mathbf{P}$ mátrix pedig a következő lesz:

$$\mathbf{P} = \left(\begin{array}{cc:ccc:c} 0 & 0 & 1 & -1 & 1 & 1 \\ 0 & 0 & 5 & 7 & -7 & -25 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ -1 & -5 & 0 & 0 & 0 & 1 \\ 1 & -7 & 0 & 0 & 0 & 2 \\ -1 & 7 & 0 & 0 & 0 & -2 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ -1 & 25 & -1 & -2 & 2 & 0 \end{array}\right).$$

25.2.8. A Neumann-módszer

A fiktív lejátszás módszere egy iterációs algoritmus, ahol a játékosok minden lépésben hozzáigazítják stratégiáikat a többi játékos stratégiáihoz. Ez a módszer tehát úgy tekinthető, mint egy diszkrét rendszer megvalósulása, ahol a játékosok stratégiaválasztásai az állapotváltozók. Neumann János a szimmetrikus játékok esetére bevezetett egy folytonos megközelítést, ahol a játékosok folyamatosan módosítják a stratégiáikat. Ez a módszer alkalmazható tetszőleges mátrixjátékra, hiszen – amint korábban láttuk – bármely mátrixjáték ekvivalens egy szimmetrikus mátrixjátékkal. Ez a módszer szintén használható lineáris programozási feladatok megoldására, hiszen korábban láttuk, hogy minden primál-duál feladatpár visszavezethető egy szimmetrikus mátrixjáték egyensúlyi problémájára.

Legyen a továbbiakban $\mathbf{P}$ n-edrendű ferdén-szimmetrikus mátrix. A $\mathcal{P}_2$ játékos $\mathbf{y}(t)$ stratégiája egy függvény, mely a $t \geq 0$ idő változótól függ. Mielőtt a rendszer dinamikáját felírnánk, bevezetjük a következő jelöléseket:

$$\begin{array}{lcccll} u_i: & \mathbb{R}^n & \to & \mathbb{R}, & u_i(\mathbf{y}(t)) & = \mathbf{e}_i^T \mathbf{P}\mathbf{y}(t) \quad (i=1,2,\ldots,n)\,, \\ \phi: & \mathbb{R} & \to & \mathbb{R}, & \phi(u_i) & = \max\{0, u_i\}\,, \\ \Phi: & \mathbb{R}^n & \to & \mathbb{R}, & \Phi(\mathbf{y}(t)) & = \sum_{i=1}^n \phi(u_i(\mathbf{y}(t)))\,. \end{array} \tag{25.33}$$

Oldjuk meg a következő nemlineáris kezdetiérték-feladatot tetszőleges $\mathbf{y}_0$-ra:

$$y_j'(t) = \phi(u_j(\mathbf{y}(t))) - \Phi(\mathbf{y}(t)) y_j(t), \quad y_j(0) = y_{j0} \quad (1 \leq j \leq n)\,. \tag{25.34}$$

Mivel a jobb oldalon lévő kifejezés folytonos, így (26.34)-nek van legalább egy megoldása. A jobb oldalon lévő kifejezés a következőképpen értelmezhető. Tegyük fel, hogy $\phi(u_j(\mathbf{y}(t))) > 0$. Ekkor ha $\mathcal{P}_2$ az $\mathbf{y}(t)$ stratégiát választja, akkor $\mathcal{P}_1$ pozitív kifizetést tud elérni az $\mathbf{e}_j$ stratégia választásával, mely választás negatív kifizetést eredményez a $\mathcal{P}_2$ játékosnak. Ha azonban $\mathcal{P}_2$ egyre úgy növeli $y_j(t)$-t, hogy $\mathbf{e}_j$ stratégiát választ ő is, akkor az $\mathbf{e}_j^T \mathbf{P}\mathbf{e}_j$ kifizetése nullává válik, tehát megnő. Ebből következik, hogy $\mathcal{P}_2$ érdeke $y_j(t)$ növelése. Pontosan ezt fejezi ki a jobb oldali kifejezés első tagja. A második tag azt biztosítja, hogy $\mathbf{y}(t)$ valószínűségi vektor maradjon minden $t \geq 0$-ra.

(26.34) jobb oldalának kiszámításához minden t-re $N^2 + N$ szorzásra van szükség. A teljes számítási költség függ a megoldás intervallumának a hosszától, a választott lépésnagyságtól, és a differenciálegyenletet megoldó módszer megválasztásától.

25.14. tétel. *Tegyük fel, hogy $t_1, t_2, \ldots$ egy szigorúan növekedő nemkorlátos sorozat. Ekkor az $\mathbf{y}(t_n)$ sorozat minden torlódási pontja egyensúlyi stratégia, és létezik egy olyan c konstans, hogy*

$$\mathbf{e}_i^T \mathbf{P}\mathbf{y}(t_k) \leq \frac{\sqrt{n}}{c + t_k} \quad (i = 1, 2, \ldots, n)\,. \tag{25.35}$$

Bizonyítás. Először azt kell megmutatnunk, hogy $\mathbf{y}(t)$ valószínűségi vektor minden $t \geq 0$-ra. Tegyük fel, hogy valamilyen j-re és $t_1 > 0$-ra $y_j(t_1) < 0$. Legyen

$$t_0 = \sup\{t | 0 < t < t_1, y_j(t) \geq 0\} .$$

Ekkor $y_j(t)$ folytonossága és $y_j(0) \geq 0$ miatt $y_j(t_0) = 0$, és minden $\tau \in (t_0, t_1)$-re, $y_j(\tau) < 0$. Az előzőekből következik, hogy minden $\tau \in (t_0, t_1]$-re

$$y_j'(\tau) = \phi(u_j(\mathbf{y}(\tau))) - \Phi(\mathbf{y}(\tau))y_j(\tau) \geq 0 .$$

A Lagrange-középérték tétel miatt létezik $\tau \in (t_0, t_1)$, hogy

$$y_j(t_1) = y_j(t_0) + y_j'(\tau)(t_1 - t_0) \geq 0 ,$$

ami ellentmondás. Tehát $y_j(t)$ nemnegatív minden $t \geq 0$-ra. A következőkben megmutatjuk, hogy $\sum_{j=1}^n y_j(t) = 1$ minden $t \geq 0$-ra. Legyen $f(t) = 1 - \sum_{j=1}^n y_j(t)$, ekkor

$$f'(t) = -\sum_{j=1}^n y_j'(t) = -\sum_{j=1}^n \phi(u_j(\mathbf{y}(t))) + \Phi(\mathbf{y}(t))(\sum_{j=1}^n y_j(t))$$

$$= \Phi(\mathbf{y}(t))(1 - \sum_{j=1}^n y_j(t)) ,$$

tehát $f(t)$ megoldása a következő homogén rendszernek:

$$f'(t) = -\Phi(\mathbf{y}(t))f(t)$$

az $f(0) = 1 - \sum_{j=1}^n y_{j0} = 0$ kezdetiérték mellett. Tehát, minden $t \geq 0$-ra $f(t) = 0$, ami azt jelenti, hogy $\mathbf{y}(t)$ valószínűségi vektor minden $t \geq 0$-ra.

Tegyük fel, hogy valamilyen $t \geq 0$ mellett $y_i(u_i(\mathbf{y}(t))) > 0$. Ekkor

$$\begin{aligned} \frac{d}{dt}\phi(u_i(\mathbf{y}(t))) &= \sum_{j=1}^n p_{ij} y_j'(t) = \sum_{j=1}^n p_{ij}[\phi(u_j(\mathbf{y}(t))) - \Phi(\mathbf{y}(t))y_j(t)] \\ &= \sum_{j=1}^n p_{ij}\phi(u_j(\mathbf{y}(t))) - \Phi(\mathbf{y}(t))\phi(u_i(\mathbf{y}(t))) . \end{aligned} \tag{25.36}$$

Szorozzuk meg mindkét oldalt $\phi(u_i(\mathbf{y}(t)))$-vel, és adjuk össze az így kapott egyenlőségeket $i = 1, 2, \ldots, n$-re:

$$\sum_{i=1}^n \phi(u_i(\mathbf{y}(t)))\frac{d}{dt}\phi(u_i(\mathbf{y}(t)))$$

$$= \sum_{i=1}^{n}\sum_{j=1}^{n} p_{ij}\phi(u_i(\mathbf{y}(t)))\phi(u_j(\mathbf{y}(t))) - \Phi(\mathbf{y}(t))(\sum_{i=1}^{n}\phi^2(u_i(\mathbf{y}(t)))) \ . \quad (25.37)$$

Mivel $\mathbf{P}$ ferdén-szimmetrikus, így az első tag nulla. Vegyük észre, hogy a fenti egyenlőség a töréspont (ahol $\phi(u_i(\mathbf{y}(t)))$ deriváltja nem létezik) kivételével akkor is érvényes marad, ha $\phi(u_i(\mathbf{y}(t))) = 0$, így (26.36) igaz marad.

Most tegyük fel, hogy valamilyen pozitív t-re $\Phi(\mathbf{y}(t)) = 0$. Ekkor minden i-re $\phi(u_i(\mathbf{y}(t))) = 0$. Mivel (26.37) átírható

$$\frac{1}{2}\frac{d}{dt}\Psi(\mathbf{y}(t)) = -\Phi(\mathbf{y}(t))\Psi(\mathbf{y}(t)) \quad (25.38)$$

formába, ahol

$$\Psi : \mathbb{R}^n \to \mathbb{R} \quad \text{és} \quad \Psi(\mathbf{y}(t)) = \sum_{i=1}^{n}\phi^2(u_i(\mathbf{y}(t))) \ ,$$

így látható, hogy $\Psi(\mathbf{y}(t))$ kielégíti a homogén egyenletet nulla kezdetiérték mellett, tehát a megoldás nulla marad minden $\tau \geq t$-re. Ebből következik, hogy $\phi(u_i(\mathbf{y}(\tau))) = 0$ megoldásra $\mathbf{P}\mathbf{y}(\tau) \leq \mathbf{0}$, azaz $\mathbf{y}(\tau)$ egyensúly.

Ha $\Phi(\mathbf{y}(\mathbf{t})) > 0$ minden $t \geq 0$-ra, akkor $\Psi(\mathbf{y}(t)) > 0$, és könnyen látható, hogy

$$\frac{1}{2}\frac{d}{dt}\Psi(\mathbf{y}(t)) \leq -\sqrt{\Psi(\mathbf{y}(t))}\Psi(\mathbf{y}(t)) \ ,$$

azaz

$$\frac{1}{2}\frac{d}{dt}\Psi(\mathbf{y}(t))(\Psi(\mathbf{y}(t)))^{-\frac{3}{2}} \leq -1 \ .$$

Mindkét oldalt a $[0,t]$ intervallumon integrálva azt kapjuk, hogy

$$-\Psi(\mathbf{y}(t))^{-(1/2)} + c \leq -t \ ,$$

ahol $c = (\Psi(\mathbf{y}(0)))^{-(1/2)}$, melyből következik

$$(\Psi(\mathbf{y}(t)))^{1/2} \leq \frac{1}{c+t} \ . \quad (25.39)$$

A Cauchy-Schwartz-egyenlőtlenség alkalmazásával kapjuk, hogy

$$\mathbf{e}_i^T\mathbf{P}\mathbf{y}(t) = u_i(\mathbf{y}(t)) \leq \phi(u_i(\mathbf{y}(t))) \leq \Phi(\mathbf{y}(t)) \leq \sqrt{n\Psi(\mathbf{y}(t))} \leq \frac{\sqrt{n}}{c+t} \ , \quad (25.40)$$

mely egyenlőtlenség az u_i folytonossága miatt még a töréspontokban is igaz. Végül vegyük a következő sorozatot: $\{\mathbf{y}(t_k)\}$, ahol t_k monoton növekedő nem korlátos sorozat. Mivel $\mathbf{y}(t_k)$-k valószínűségi vektorok, így korlátosak, tehát van legalább egy $\mathbf{y}^\star$ torlódási pontjuk. (26.40)-ből t_k-val a végtelenbe tartva azt kapjuk, hogy $\mathbf{P}\mathbf{y}^\star \leq \mathbf{0}$, tehát $\mathbf{y}^\star$ egyensúly. ■

25.18. példa. *Negyedik mátrixjáték.* Tekintsük a 25.13 példában bevezetett mátrixjátékot. A Neumann-módszer alkalmazásához először fel kell írni az ekvivalens szimmetrikus mátrixjátékot. Az átírás módszere, mely a 25.12. tételben található, megköveteli, hogy a mátrix elemei pozitívak legyenek. Anélkül, hogy az egyensúlyok megváltoznának, az $\mathbf{A}$ mátrix minden eleméhez hozzáadhatunk 2-t, ekkor a következő mátrixot kapjuk:

$$\mathbf{A} = \begin{pmatrix} 4 & 3 & 2 \\ 4 & 2 & 5 \\ 1 & 5 & 5 \end{pmatrix} ,$$

és a fent említett módszer segítségével

$$\mathbf{P} = \left(\begin{array}{ccc:ccc:c} 0 & 0 & 0 & 4 & 3 & 2 & -1 \\ 0 & 0 & 0 & 4 & 2 & 5 & -1 \\ 0 & 0 & 0 & 1 & 5 & 5 & -1 \\ \hdashline -4 & -4 & -1 & 0 & 0 & 0 & 1 \\ -3 & -2 & -5 & 0 & 0 & 0 & 1 \\ -2 & -5 & -5 & 0 & 0 & 0 & 1 \\ \hdashline 1 & 1 & 1 & -1 & -1 & -1 & 0 \end{array}\right) .$$

A (26.34) differenciálegyenletet a negyedrendű Runge–Kutta-módszerrel oldottuk meg a $[0, 100]$ intervallumon, $h = 0.01$ lépésnagysággal az $\mathbf{y}(0) = (1, 0, \ldots, 0)^T$ kezdetiérték mellett. $\mathbf{y}(100)$-ra kaptuk a következő közelítő értékeket:

$$\mathbf{x} \approx (0.563619, 0.232359, 0.241988) ,$$

$$\mathbf{y} \approx (0.485258, 0.361633, 0.115144) .$$

Ezen közelítés az eredeti játék egy egyensúlyának becslése. Hasonlítsuk össze ezeket az értékeket az egyensúlyi vektorpárral:

$$\mathbf{x} = \left(\frac{4}{7}, \frac{4}{21}, \frac{5}{21}\right) \quad \text{és} \quad \mathbf{y} = \left(\frac{3}{7}, \frac{3}{7}, \frac{1}{7}\right) .$$

Látható, hogy a maximális hiba 0.067.

25.2.9. Átlósan szigorúan konkáv játékok

Tekintsünk egy N személyes folytonos játékot, és tegyük fel, hogy a 25.2.3. pont feltevései teljesülnek. Tegyük fel továbbá, hogy minden S_k korlátos minden k-ra, $\mathbf{g}_k$ minden komponense szerint konkáv, és f_k konkáv s_k-ban tetszőlegesen rögzített $s_1, \ldots, s_{k-1}, s_{k+1}, \ldots, N$ mellett. A 25.3. tétel miatt a

fenti feltételek mellett a játéknak van legalább egy egyensúlya. Általában az egyensúly unicitása még akkor sem biztosított, ha minden f_k szigorúan kvázikonkáv s_k szerint. A következő példa ezt a tényt mutatja be.

25.19. példa. *Ellenpélda.* Tekintsük a következő kétszemélyes játékot: $S_1 = S_2 = [0,1]$ és $f_1(s_1,s_2) = f_2(s_1,s_2) = 1-(s_1-s_2)^2$. Világos, hogy mindkét kifizetőfüggvény szigorúan konkáv, és mégis végtelen sok egyensúly van: $s_1^\star = s_2^\star \in [0,1]$.

Válasszunk egy nemnegatív $\mathbf{r} \in \mathbb{R}^N$ vektort, és definiáljuk a következő függvényt:

$$\mathbf{h} : \mathbb{R}^M \to \mathbb{R}^M, \quad \mathbf{h}(\mathbf{s},\mathbf{r}) = \begin{pmatrix} r_1 \nabla_1 f_1(\mathbf{s})^T \\ r_2 \nabla_2 f_2(\mathbf{s})^T \\ \vdots \\ r_N \nabla_N f_N(\mathbf{s})^T \end{pmatrix}, \tag{25.41}$$

ahol $M = \sum_{k=1}^N n_k$, és $\nabla_k f_k$ az f_k függvény s_k szerinti gradiens(sor)vektora. Egy játékot ***átlósan szigorúan konkávnak*** hívunk, ha minden $\mathbf{s}^{(1)}, \mathbf{s}^{(2)} \in S$, $\mathbf{s}^{(1)} \neq \mathbf{s}^{(2)}$-re, és valamilyen $\mathbf{r} \geq \mathbf{0}$-ra

$$(\mathbf{s}^{(1)} - \mathbf{s}^{(2)})^T (\mathbf{h}(\mathbf{s}^{(1)},\mathbf{r}) - \mathbf{h}(\mathbf{s}^{(2)},\mathbf{r})) < 0\,. \tag{25.42}$$

25.15. tétel. *Egy átlósan szigorúan konkáv játéknak pontosan egy egyensúlya van.*

Bizonyítás. A 25.3. tétel miatt létezik egyensúly. Az egyértelműség bizonyítása céljából tegyük fel, hogy $\mathbf{s}^{(1)}$ és $\mathbf{s}^{(2)}$ két egyensúly, melyek kielégítik (26.9) feltételeket és $\mathbf{s}^{(1)} \neq \mathbf{s}^{(2)}$. Ekkor $l = 1,2$-re

$$\begin{aligned} {\mathbf{u}_k^{(l)}}^T \mathbf{g}_k(s_k^{(l)}) &= 0 \\ \nabla_k f_k(\mathbf{s}^{(l)}) + {\mathbf{u}_k^{(l)}}^T \nabla_k \mathbf{g}_k(s_k^{(l)}) &= \mathbf{0}^T\,. \end{aligned}$$

A második egyenlőséget a következő formába lehet átírni:

$$\nabla_k f_k(\mathbf{s}^{(l)}) + \sum_{j=1}^{m_k} u_{kj}^{(l)} \nabla_k g_{kj}(s_k^{(l)}) = 0\,, \tag{25.43}$$

ahol $u_{kj}^{(l)}$ a $\mathbf{u}_k^{(l)}$-nak, és g_{kj} a $\mathbf{g}_k$ j-edik komponense. Szorozzuk meg (26.43)-at $(r_k(s_k^{(2)} - s_k^{(1)})^T)$-tal $l = 1$ esetén, és $r_k(s_k^{(1)} - s_k^{(2)})^T$-tal $l = 2$ esetén. Adjuk össze az így kapott egyenlőségeket $k = 1,2,\ldots,N$-re. Ekkor a következőt

kapjuk:

$$0 = \{(\mathbf{s}^{(2)} - \mathbf{s}^{(1)})\mathbf{h}(\mathbf{s}^{(1)}, \mathbf{r}) + (\mathbf{s}^{(1)} - \mathbf{s}^{(2)})\mathbf{h}(\mathbf{s}^{(2)}, \mathbf{r})\}$$
$$+ \sum_{k=1}^{N} \sum_{j=1}^{m_k} r_k [u_{kj}^{(1)} (s_k^{(2)} - s_k^{(1)})^T \nabla_k g_{kj}(s_k^{(1)}) + u_{kj}^{(2)} (s_k^{(1)} - s_k^{(2)})^T \nabla_k g_{kj}(s_k^{(2)})] \; . \tag{25.44}$$

Vegyük észre, hogy az átlósan szigorú konkavitás miatt az első két tag összege pozitív, $\mathbf{g}_k$ komponenseinek a konkavitása miatt pedig

$$(s_k^{(2)} - s_k^{(1)})^T \nabla_k g_{kj}(s_k^{(1)}) \geq g_{kj}(s_k^{(2)}) - g_{kj}(s_k^{(1)})$$

és

$$(s_k^{(1)} - s_k^{(2)})^T \nabla_k g_{kj}(s_k^{(2)}) \geq g_{kj}(s_k^{(1)}) - g_{kj}(s_k^{(2)}) \; .$$

Tudjuk, hogy minden k-ra, l-re

$$0 = {\mathbf{u}_k^{(l)}}^T \mathbf{g}_k(s_k^{(l)}) = \sum_{j=1}^{m_k} u_{kj}^{(l)} g_{kj}(s_k^{(l)}) \; ,$$

ekkor (26.44)-ből kapjuk, hogy

$$0 > \sum_{k=1}^{N} \sum_{j=1}^{m_k} r_k [u_{kj}^{(1)} (g_{kj}(s_k^{(2)}) - g_{kj}(s_k^{(1)})) + u_{kj}^{(2)} (g_{kj}(s_k^{(1)}) - g_{kj}(s_k^{(2)}))]$$
$$= \sum_{k=1}^{N} \sum_{j=1}^{m_k} r_k [u_{kj}^{(1)} g_{kj}(s_k^{(2)}) + u_{kj}^{(2)} g_{kj}(s_k^{(1)})] \geq 0 \; ,$$

ami nyilvánvaló ellentmondás, tehát $\mathbf{s}^{(1)} = \mathbf{s}^{(2)}$. ∎

Az egyensúly egyértelműségének ellenőrzése

A következő tételben egy olyan, a gyakorlati esetekben nagyon hasznos módszert mutatunk be, melynek segítségével ellenőrizhetjük egy N személyes játék átlósan szigorúan konkavitását.

25.16. tétel. *Tegyük fel, hogy S konvex, f_k kétszer folytonosan differenciálható minden k-ra, és létezik $\mathbf{r} \geq \mathbf{0}$, hogy $\mathbf{J}(\mathbf{s}, \mathbf{r}) + \mathbf{J}(\mathbf{s}, \mathbf{r})^T$ negatív definit, ahol $\mathbf{J}(\mathbf{s}, \mathbf{r})$ a $\mathbf{h}(\mathbf{s}, \mathbf{r})$ Jakobi-mátrixa. Ekkor a játék átlósan szigorúan konkáv.*

Bizonyítás. Legyen $\mathbf{s}^{(1)}, \mathbf{s}^{(2)} \in S$ és $\mathbf{s}^{(1)} \neq \mathbf{s}^{(2)}$. Ekkor minden $\alpha \in [0, 1]$-re $\mathbf{s}(\alpha) = \alpha \mathbf{s}^{(1)} + (1 - \alpha)\mathbf{s}^{(2)} \in S$ és

$$\frac{d}{d\alpha} \mathbf{h}(\mathbf{s}(\alpha), \mathbf{r}) = \mathbf{J}(\mathbf{s}(\alpha), \mathbf{r})(\mathbf{s}^{(1)} - \mathbf{s}^{(2)}) \; .$$

Mindkét oldalt $[0,1]$-en integrálva

$$\mathbf{h}(\mathbf{s}^{(1)},\mathbf{r})-\mathbf{h}(\mathbf{s}^{(2)},\mathbf{r})=\int_0^1 \mathbf{J}(\mathbf{s}(\alpha),\mathbf{r})(\mathbf{s}^{(1)}-\mathbf{s}^{(2)})d\alpha\ ,$$

és mindkét oldalt újra megszorozva $(\mathbf{s}^{(1)}-\mathbf{s}^{(2)})^T$-tal

$$\begin{aligned}(\mathbf{s}^{(1)}-\mathbf{s}^{(2)})^T(\mathbf{h}(\mathbf{s}^{(1)},\mathbf{r})-\mathbf{h}(\mathbf{s}^{(2)},\mathbf{r}))&=\int_0^1(\mathbf{s}^{(1)}-\mathbf{s}^{(2)})^T\mathbf{J}(\mathbf{s}(\alpha),\mathbf{r})(\mathbf{s}^{(1)}-\mathbf{s}^{(2)})d\alpha\\&=\frac{1}{2}\int_0^1(\mathbf{s}^{(1)}-\mathbf{s}^{(2)})^T(\mathbf{J}(\mathbf{s}(\alpha),\mathbf{r})+\mathbf{J}(\mathbf{s}(\alpha),\mathbf{r})^T)(\mathbf{s}^{(1)}-\mathbf{s}^{(2)})d\alpha<0\ ,\end{aligned}$$

tehát a bizonyítást befejeztük. ■

25.20. példa. *Egyszerű kétszemélyes játék.* Tekintsük a következő egyszerű kétszemélyes játékot, ahol $S_1=S_2=[0,1]$, és a kifizetőfüggvények:

$$f_1(s_1,s_2)=-s_1^2+s_1-s_1s_2$$

és

$$f_2(s_1,s_2)=-s_2^2+s_2-s_1s_2\ .$$

Könnyen látható, hogy – az átlósan szigorú konkavitáson kívül – minden tulajdonság, amit ebben az alfejezetben feltettünk, teljesül. A 25.16. tételt használjuk a hiányzó tulajdonság megmutatására. Ebben az esetben

$$\nabla_1 f_1(s_1,s_2)=-2s_1+1-s_2,\quad \nabla_2 f_2(s_1,s_2)=-2s_2+1-s_1\ ,$$

így

$$\mathbf{h}(\mathbf{s},\mathbf{r})=\begin{pmatrix}r_1(-2s_1+1-s_2)\\ r_2(-2s_2+1-s_1)\end{pmatrix}.$$

A Jakobi-mátrix:

$$\mathbf{J}(\mathbf{s},\mathbf{r})=\begin{pmatrix}-2r_1 & -r_1\\ -r_2 & -2r_2\end{pmatrix}.$$

Megmutatjuk, hogy valamilyen $\mathbf{r}\geq\mathbf{0}$-ra a

$$\mathbf{J}(\mathbf{s},\mathbf{r})+\mathbf{J}(\mathbf{s},\mathbf{r})^T=\begin{pmatrix}-4r_1 & -r_1-r_2\\ -r_1-r_2 & -4r_2\end{pmatrix}$$

mátrix negatív definit. Legyen például $r_1=r_2=1$, ekkor a fenti mátrix

$$\begin{pmatrix}-4 & -2\\ -2 & -4\end{pmatrix},$$

a karakterisztikus polinom:

$$\phi(\lambda)=\det\begin{pmatrix}-4-\lambda & -2\\ -2 & -4-\lambda\end{pmatrix}=\lambda^2+8\lambda+12\ ,$$

mely polinomnak a két gyöke $\lambda_1=-2$, $\lambda_2=-6$.

Az egyensúly iteratív kiszámítása

A 25.4. tételben láttuk, hogy $\mathbf{s}^\star \in S$ pontosan akkor egyensúly, ha

$$\mathbf{H_r}(\mathbf{s}^\star, \mathbf{s}^\star) \geq \mathbf{H_r}(\mathbf{s}^\star, \mathbf{s}) \tag{25.45}$$

minden $\mathbf{s} \in S$-re, ahol $\mathbf{H_r}$ a (26.4)-ben definiált összegzőfüggvény. A következőkben feltesszük, hogy az alfejezet elején feltett tulajdonságok érvényesek, és létezik olyan $\mathbf{r} \geq \mathbf{0}$, hogy (26.42) érvényes.

Először a variációs egyenlőtlenség és (26.45) ekvivalenciáját mutatjuk meg.

25.17. tétel. *Egy $\mathbf{s}^\star \in S$ vektor pontosan akkor elégíti ki (26.45)-öt, ha*

$$\mathbf{h}(\mathbf{s}^\star, \mathbf{r})^T(\mathbf{s} - \mathbf{s}^\star) \leq 0 \tag{25.46}$$

minden $\mathbf{s} \in S$ stratégiára, ahol $\mathbf{h}(\mathbf{s}, \mathbf{r})$-et (26.41)-ben definiáltuk.

Bizonyítás. Tegyük fel, hogy $\mathbf{s}^\star$ kielégíti (26.45)-öt. Ekkor $\mathbf{H_r}(\mathbf{s}^\star, \mathbf{s})$ – mint $\mathbf{s}$ argumentumú függvény – felveszi a maximumát $\mathbf{s} = \mathbf{s}^\star$-ban, tehát

$$\nabla_{\mathbf{s}} \mathbf{H_r}(\mathbf{s}^\star, \mathbf{s}^\star)(\mathbf{s} - \mathbf{s}^\star) \leq 0$$

minden $\mathbf{s} \in S$ stratégiára. Mivel $\nabla_{\mathbf{s}} \mathbf{H_r}(\mathbf{s}^\star, \mathbf{s}^\star) = \mathbf{h}(\mathbf{s}^\star, \mathbf{r})$, így $\mathbf{s}^\star$ kielégíti (26.46)-t.

Most tegyük fel, hogy $\mathbf{s}^\star$ kielégíti (26.46)-ot. $\mathbf{H_r}(\mathbf{s}^\star, \mathbf{s}^\star)$ $\mathbf{s}$ szerinti konkavitása, és a játék átlósan szigorú konkavitása miatt

$$\mathbf{H_r}(\mathbf{s}^\star, \mathbf{s}^\star) - \mathbf{H_r}(\mathbf{s}^\star, \mathbf{s}) \geq \mathbf{h}(\mathbf{s}, \mathbf{r})^T(\mathbf{s}^\star - \mathbf{s}) \geq \mathbf{h}(\mathbf{s}, \mathbf{r})^T(\mathbf{s}^\star - \mathbf{s}) + \mathbf{h}(\mathbf{s}^\star, \mathbf{r})^T(\mathbf{s} - \mathbf{s}^\star) > 0 \,,$$

tehát $\mathbf{s}^\star$ kielégíti (26.45)-öt. ∎

Látható, hogy bármely – a variációs egyenlőtlenség megoldására alkalmas – módszer használható a játék egyensúlyi problémájának megoldására.

A következőkben definiálunk egy olyan speciális kétszemélyes, zérusösszegű-játékot, melynek egyensúlyi problémája ekvivalens az eredeti N személyes játék egyensúlyi problémájával.

25.18. tétel. *Az $\mathbf{s}^\star \in S$ vektor pontosan akkor elégíti ki (26.46)-ot, ha $(\mathbf{s}^\star, \mathbf{s}^\star)$ egyensúlya egy olyan kétszemélyes játéknak, ahol mindkét stratégiahalmaz S, a kifizetőfüggvények pedig $f_1(\mathbf{s}, \mathbf{z}) = f(\mathbf{s}, \mathbf{z})$ és $f_2(\mathbf{s}, \mathbf{z}) = -f(\mathbf{s}, \mathbf{z})$, ahol $f(\mathbf{s}, \mathbf{z}) = \mathbf{h}(\mathbf{z}, \mathbf{r})^T(\mathbf{s} - \mathbf{z})$.*

Bizonyítás. Először tegyük fel, hogy $\mathbf{s}^\star \in S$ kielégíti (26.45)-t. Ekkor kielégíti (26.46)-t is, így

$$f(\mathbf{s}, \mathbf{s}^\star) \leq 0 = f(\mathbf{s}^\star, \mathbf{s}^\star) \,.$$

Még azt kell megmutatnunk, hogy

$$-f(\mathbf{s}^\star,\mathbf{s}) \le 0 = -f(\mathbf{s}^\star,\mathbf{s}^\star)\ .$$

Indirekt módon tegyük el, hogy valamilyen $\mathbf{s}$-re $f(\mathbf{s}^\star,\mathbf{s}) < 0$. Ekkor (26.42) és (26.46) miatt

$$\begin{aligned} 0 > f(\mathbf{s}^\star,\mathbf{s}) &= \mathbf{h}(\mathbf{s},\mathbf{r})^T(\mathbf{s}^\star-\mathbf{s}) > \mathbf{h}(\mathbf{s},\mathbf{r})^T(\mathbf{s}^\star-\mathbf{s}) + (\mathbf{s}-\mathbf{s}^\star)^T(\mathbf{h}(\mathbf{s},\mathbf{r})-\mathbf{h}(\mathbf{s}^\star,\mathbf{r})) \\ &= \mathbf{h}(\mathbf{s}^\star,\mathbf{r})^T(\mathbf{s}^\star-\mathbf{s}) \ge 0\ , \end{aligned}$$

ami ellentmondás. Most tegyük fel, hogy $(\mathbf{s}^\star,\mathbf{s}^\star)$ egyensúlya a tételben bevezetett játéknak. Ekkor tetszőleges $\mathbf{s},\mathbf{z}\in S$-re

$$f(\mathbf{s},\mathbf{s}^\star) \le f(\mathbf{s}^\star,\mathbf{s}^\star) = 0 \le f(\mathbf{s},\mathbf{z})\ .$$

Az első egyenlőtlenség átírható a következő formába:

$$\mathbf{h}(\mathbf{s}^\star,\mathbf{r})^T(\mathbf{s}-\mathbf{s}^\star) \le 0\ ,$$

tehát (26.46) teljesül, így teljesül (26.45) is. ■

Tekintsük a következő iterációs eljárást.

Legyen $\mathbf{s}^{(1)}\in S$ tetszőleges, és oldjuk meg a következő feladatot:

$$\begin{array}{rcl} f(\mathbf{s},\mathbf{s}^{(1)}) & \to & \max \\ \mathbf{s} & \in & S\ . \end{array} \tag{25.47}$$

Jelöljük $\mathbf{s}^{(2)}$-vel (25.47) feladat megoldását, és legyen $\mu_1 = f(\mathbf{s}^{(2)},\mathbf{s}^{(1)})$. Ha $\mu_1 = 0$, akkor minden $\mathbf{s}\in S$-re

$$f(\mathbf{s},\mathbf{s}^{(1)}) = \mathbf{h}(\mathbf{s}^{(1)},\mathbf{r})^T(\mathbf{s}-\mathbf{s}^{(1)}) \le 0\ ,$$

így 25.17. tétel miatt $\mathbf{s}^{(1)}$ egyensúly. Mivel $f(\mathbf{s}^{(1)},\mathbf{s}^{(1)}) = 0$, így feltesszük, hogy $\mu_1 > 0$. $k\ge 2$ általános lépésben már van k vektorunk $\mathbf{s}^{(1)},\mathbf{s}^{(2)},\dots,\mathbf{s}^{(k)}$, és $k-1$ skalárunk $\mu_1,\mu_2,\dots,\mu_{k-1} > 0$. Ekkor a következő $\mathbf{s}^{(k+1)}$ vektor és μ_k skalár megoldása a következő feladatnak:

$$\begin{array}{rcll} \mu & \to & \max & \\ f(\mathbf{s},\mathbf{s}^{(i)}) & \ge & \mu & (i=1,2,\dots,k) \\ \mathbf{s} & \in & S\ . & \end{array} \tag{25.48}$$

Vegyük észre, hogy

$$f(\mathbf{s}^{(k)},\mathbf{s}^{(i)}) \ge \mu_{k-1} \ge 0 \qquad (i=1,2,\dots,k-1)$$

és

$$f(\mathbf{s}^{(k)},\mathbf{s}^{(k)}) = 0.$$

Ekkor tudjuk, hogy $\mu_k \geq 0$.

Az algoritmus pszeudokódja a következő.

ITERÁL

1 $k = 1$
2 oldjuk meg a (25.47) feladatot és legyen $\mathbf{s}^{(2)}$ egy optimális megoldás
3 **if** $f(\mathbf{s}^{(2)}, \mathbf{s}^{(1)}) = 0$
4 **return** "$\mathbf{s}^{(1)}$ egyensúly"
4 $k = k + 1$
5 oldjuk meg a (25.48) feladatot, legyen $\mathbf{s}^{(k+1)}$ egy optimális megoldás
6 **if** $||\mathbf{s}^{(k+1)} - \mathbf{s}^{(k)}|| < \varepsilon$
7 $\mathbf{s}^{(k+1)}$ egyensúly
8 **else** folytassuk a 4-edik sorban

A fenti algoritmus konvergencia tételének tárgyalása előtt megjegyezzük, hogy abban a speciális esetben, amikor a stratégiahalmazok lineáris egyenlőtlenségekkel vannak megadva (tehát a $\mathbf{g}_k$ függvények lineárisak), a (25.48) feladat minden feladata lineáris, tehát minden iterációs lépésben egy lineáris programozási feladatot kell megoldanunk.

Ha lineáris esetben a szimplex módszert alkalmazzuk minden iterációs lépésben, akkor a számítási költség exponenciális, tehát az egész eljárás számítási költsége exponenciális (rögzített lépésszám mellett).

25.19. tétel. *Az $\{\mathbf{s}^{(k)}\}$ fenti módszer által generált sorozatnak van olyan $\{\mathbf{s}^{(k_i)}\}$ részsorozata, amely az N személyes játék egyetlen egyensúlyához konvergál.*

Bizonyítás. A bizonyítás több lépésből áll.

Először azt mutatjuk meg, hogy $\lim_{k\to\infty} \mu_k = 0$. Mivel (25.48) minden iterációban egy új feltétellel bővül, tehát $\{\mu_k\}$ nem lehet növekvő. Tudjuk azt is, hogy $\{\mu_k\}$ nemnegatív, tehát konvergens. Az $\{\mathbf{s}^{(k)}\}$ sorozat korlátos, hiszen minden tagja a korlátos S halmazból való, így van egy $\{\mathbf{s}^{(k_i)}\}$ konvergens részsorozata. Vegyük észre, hogy

$$0 \leq \mu_{k_i-1} = \min_{1\leq k\leq k_i-1} \mathbf{h}(\mathbf{s}^{(k)}, \mathbf{r})^T(\mathbf{s}^{(k_i)} - \mathbf{s}^{(k)}) \leq \mathbf{h}(\mathbf{s}^{(k_i-1)}, \mathbf{r})^T(\mathbf{s}^{(k_i)} - \mathbf{s}^{(k_i-1)}) \ ,$$

ahol az egyenlőtlenség jobb oldala nullához tart. Tehát $\mu_{k_i-1} \to 0$. Mivel $\{\mu_k\}$ monoton, így $\{\mu_k\} \to 0$.

Most legyen $\mathbf{s}^\star$ az N-személyes játék egyensúlya, és legyen

$$\delta(t) = \min\{(\mathbf{h}(\mathbf{s}, \mathbf{r}) - \mathbf{h}(\mathbf{z}, \mathbf{r}))^T(\mathbf{z} - \mathbf{s}) | \|\mathbf{s} - \mathbf{z}\| \geq t, \ \mathbf{z}, \mathbf{s} \in S\} \ . \qquad (25.49)$$

(26.42) miatt $\delta(t) > 0$ minden $(t > 0)$-ra. Definiáljuk a k_i indexeket a következőképpen:

$$\delta(\|\mathbf{s}^{(k_i)} - \mathbf{s}^\star\|) = \min_{1\le k\le i} \delta(\|\mathbf{s}^{(k)} - \mathbf{s}^\star\|) \quad (i = 1, 2, \ldots) .$$

Ekkor minden $k = 1, 2, \ldots, i$-re

$$\begin{aligned} \delta(\|\mathbf{s}^{(k_i)} - \mathbf{s}^\star\|) &\le (\mathbf{h}(\mathbf{s}^{(k)}, \mathbf{r}) - \mathbf{h}(\mathbf{s}^\star, \mathbf{r}))^T(\mathbf{s}^\star - \mathbf{s}^{(k)}) \\ &= \mathbf{h}(\mathbf{s}^{(k)}, \mathbf{r})^T(\mathbf{s}^\star - \mathbf{s}^{(k)}) - \mathbf{h}(\mathbf{s}^\star, \mathbf{r})^T(\mathbf{s}^\star - \mathbf{s}^{(k)}) \\ &\le \mathbf{h}(\mathbf{s}^{(k)}, \mathbf{r})^T(\mathbf{s}^\star - \mathbf{s}^{(k)}) , \end{aligned}$$

melyből (25.48) miatt következik:

$$\begin{aligned} \delta(\|\mathbf{s}^{(k_i)} - \mathbf{s}^\star\|) &\le \min_{1\le k\le i} \mathbf{h}(\mathbf{s}^{(k)}, \mathbf{r})^T(\mathbf{s}^\star - \mathbf{s}^{(k)}) \\ &\le \max_{\mathbf{s}\in S} \min_{1\le k\le i} \mathbf{h}(\mathbf{s}^{(k)}, \mathbf{r})^T(\mathbf{s} - \mathbf{s}^{(k)}) \\ &= \min_{1\le k\le i} \mathbf{h}(\mathbf{s}^{(k)}, \mathbf{r})^T(\mathbf{s}^{(i+1)} - \mathbf{s}^{(k)}) \\ &= \mu_i . \end{aligned}$$

Ebből következik, hogy $\lim_{i\to\infty} \delta(\|\mathbf{s}^{(k_i)} - \mathbf{s}^\star\|) \to 0$. Végül vegyük észre azt, hogy $\delta(t)$ rendelkezik a következő tulajdonságokkal:

1. $\delta(t)$ folytonos;
2. ha $t > 0$, akkor $\delta(t) > 0$ (ezt mutatja (25.49));
3. ha egy $\{t^{(k)}\}$ konvergens sorozatra $\delta(t^{(k)}) \to 0$, akkor szükségszerűen $t^{(k)} \to 0$.

A 3. tulajdonság miatt $\|\mathbf{s}^{(k_i)} - \mathbf{s}^\star\| \to 0$, így $\mathbf{s}^{(k_i)} \to \mathbf{s}^\star$. ■

Gyakorlatok

25.2-1. Tekintsünk egy kétszemélyes játékot, ahol a stratégiahalmazok $S_1 = S_2 = [0, 1]$, a kifizetőfüggvények: $f_1(x_1, x_2) = x_1^2 + x_1x_2 + 2$ és $f_2(x_1, x_2) = x_1 + x_2$. Mutassuk meg, hogy a játéknak egyetlen egyensúlya van, és számítsuk ki ezt az egyensúlyt. Mutassuk meg, hogy ebben a játékban a 25.3. tétel nem alkalmazható az egyensúly létezésének bizonyítására.

25.2-2. Tekintsük az „árháború” játékot, melyben két vállalat ármeghatározó. Tegyük fel, hogy p_1 a $\mathcal{P}_1$, p_2 pedig a $\mathcal{P}_2$ játékos egy stratégiája, ahol $p_1, p_2 \in [0, p_{max}]$ (p_{max} egy rögzített pozitív valós szám), és a kifizetőfüggvények:

$$f_1(p_1, p_2) = \begin{cases} p_1, & \text{ha } p_1 \le p_2 , \\ p_1 - c, & \text{ha } p_1 > p_2 , \end{cases}$$

$$f_2(p_1,p_2)=\begin{cases} p_2, & \text{ha } p_2\le p_1\ ,\\ p_2-c, & \text{ha } p_2>p_1\ , \end{cases}$$

ahol $c<p_{max}$ rögzített. Van-e ennek a játéknak egyensúlya? Ha van egyensúlya, akkor hány van?

25.2-3. Egy tengeralattjáró rejtőzik a tenger egy részében. A tenger ezen része az egységnégyzettel modellezhető. A tengeralattjáró stratégiája az $\mathbf{x}\in[0,1]\times[0,1]$ rejtőzködési helye. Egy repülőgép bombázza az $\mathbf{y}=[0,1]\times[0,1]$-t, a tenger egy bizonyos helyét. A bombázott hely megválasztása ezen játékos stratégiája. A repülőgép kifizetése a tengeralattjárónak okozott kár nagysága: $f_2(\mathbf{x},\mathbf{y})=\alpha e^{-\beta\|\mathbf{x}-\mathbf{y}\|}$, míg a tengeralattjáró kifizetése a neki okozott kár ellentettje: $f_1(\mathbf{x},\mathbf{y})=-f_2(\mathbf{x},\mathbf{y})$. Van-e ennek a kétszemélyes játéknak egyensúlya?

25.2-4. A második-legjobb ár aukcióban egy áru kerül eladásra az N licitálók valamelyikének. A licitálók különbözőképpen értékelik a árut: $v_1<v_2<\cdots<v_N$. A licitálók egyidejűleg ajánlatot tesznek a árura úgy, hogy közben nem ismerik a többiek ajánlatát. A legmagasabb ajánlatot tevő kapja meg a árut, de a áruért csak a második legmagasabb ajánlatot kell fizetnie. Tehát a $\mathcal{P}_k$ játékos stratégiahalmaza $[0,\infty]$, stratégiája $x_k\in[0,\infty]$, és a kifizetőfüggvénye:

$$f_k(x_1,x_2,\ldots,x_N)=\begin{cases} v_k-\max_{j\ne k}x_j, & \text{ha } x_k=\max_j x_j\ ,\\ 0 & \text{különben}\ . \end{cases}$$

Határozzuk meg a $\mathcal{P}_k$ játékos legjobbválasz-leképezését. Van-e a játéknak egyensúlya?

25.2-5. Írjuk fel a Fan-egyenlőtlenséget a 25.2-1 gyakorlatra.

25.2-6. Írjuk fel és oldjuk meg a Fan-egyenlőtlenséget a 25.2-2 gyakorlatra.

25.2-7. Írjuk fel és oldjuk meg a Fan-egyenlőtlenséget a 25.2-4 gyakorlatra.

25.2-8. Tekintsük azt a kétszemélyes játékot, ahol a stratégiahalmazok $S_1=S_2=[0,1]$, a kifizetőfüggvények pedig

$$f_1(x_1,x_2)=-(x_1-x_2)^2+2x_1-x_2+1$$

$$f_2(x_1,x_2)=-(x_1-2x_2)^2-2x_1+x_2-1\ .$$

Írjuk fel a Fan-egyenlőtlenséget.

25.2-9. Legyenek $N=2$, $S_1=S_2=[0,10]$, $f_1(x_1,x_2)=f_2(x_1,x_2)=2x_1+2x_2-(x_1+x_2)^2$. Írjuk fel a Kuhn-Tucker-feltételeket, és keressük meg az egyensúlyokat. Oldjuk meg az így kapott feltételrendszert.

25.2-10. Tekintsünk egy háromszemélyes játékot, ahol $S_1=S_2=S_3=[0,1]$, $f_1(x_1,x_2,x_3)=(x_1-x_2)^2+x_3$, $f_2(x_1,x_2,x_3)=(x_2-x_3)^2+x_1$ és $f_3(x_1,x_2,x_3)=(x_3-x_1)^2+x_2$. Írjuk fel a Kuhn-Tucker-feltételeket.

25.2-11. Írjuk fel és oldjuk meg (26.9)-et a 25.2-1 és a 25.2-8 gyakorlatokban bevezetett játékokra.

25.2-12. Írjuk át 25.2-8 gyakorlat Kuhn–Tucker-feltételeit (26.10) formába, és oldjuk meg őket.

25.2-13. Írjuk fel a 26.1-1 gyakorlatban bevezetett véges játék kevert bővítését.

25.2-14. Írjuk fel és oldjuk meg (26.10)-et a 25.2-13 gyakorlatban bevezetett játékra.

25.2-15. Írjuk fel a 26.1-3 gyakorlatban bevezetett játék kevert bővítését. Írjuk fel és oldjuk meg az erre a játékra vonatkozó (26.22)-őt, ha $\alpha = 5$ és $\beta = 3$.

25.2-16. Oldjuk meg a 25.2-15 gyakorlatban bevezetett mátrixjáték egyensúlyi problémáját a fiktív lejátszás módszerével.

25.2-17. Vegyük az $\mathbf{A} = \begin{pmatrix} 2 & -1 \\ -1 & 1 \end{pmatrix}$ és $\mathbf{B} = \begin{pmatrix} 1 & -1 \\ -1 & 2 \end{pmatrix}$ mátrixokkal adott bimátrix-játékot. Oldjuk meg ezt a bimátrix-játékot a *25-1* gyakorlatban meghatározott módszerrel.

25.2-18. Oldjuk meg az $\mathbf{A} = \begin{pmatrix} 0 & 1 & 5 \\ -1 & 0 & -3 \\ -5 & 3 & 0 \end{pmatrix}$ szimmetrikus mátrixjáték egyensúlyi problémáját lineáris programozással.

25.2-19. Oldjuk meg 25.2-18 gyakorlatot a fiktív lejátszás módszerével.

25.2-20. Írjuk fel a (26.9) Kuhn–Tucker-feltételeket a 25.2-18 gyakorlatra.

25.2-21.$\star$ Tekintsük az $\mathbf{A} = \begin{pmatrix} 1 & 2 & 3 \\ -1 & 0 & 1 \end{pmatrix}$ mátrixjátékot. Oldjuk meg az $\mathbf{A}$ mátrixjáték egyensúlyi problémáját lineáris programozással, a fiktív lejátszás módszerével, és írjuk fel a Kuhn–Tucker-feltételeket. *Útmutatás.* Először határozzuk meg az ekvivalens szimmetrikus mátrixjátékot.

25.2-22. Írjuk fel lineáris programozási feladatként az $\mathbf{A} = \begin{pmatrix} 1 & 2 \\ 3 & 1 \end{pmatrix}$ mátrixjáték egyensúlyi problémáját.

25.2-23. Írjuk fel egy lineáris programozási feladat megoldását a fiktív lejátszás módszerével, és oldjuk meg az így kapott módszerrel a következő lineáris programozási feladatot:

$$\begin{aligned} x_1 + x_2 &\rightarrow \max \\ x_1, x_2 &\geq 0 \\ 3x_1 + x_2 &\leq 4 \\ x_1 + 3x_2 &\leq 4\,. \end{aligned}$$

25.2-24. Oldjuk meg a 25.2-21. gyakorlat lineáris programozási feladatát a

fiktív lejátszás módszerével.

25.2-25. Oldjuk meg a 25.2-18 gyakorlatot a Neumann-módszerrel.

25.2-26. Oldjuk meg a 25.2-21. gyakorlatot a Neumann-módszerrel.

25.2-27. Oldjuk meg a 25.2-16 gyakorlatot a Neumann-módszerrel.

25.2-28.* Ellenőrizzük a 25.2-25, 25.2-26 és 25.2-27 gyakorlatok eredményeit úgy, hogy a (26.21) lineáris programozási feladat feltételrendszerének érvényességét megvizsgáljuk nulla célfüggvényérték mellett. *Útmutatás.* Minek válasszuk α-t és β-t?

25.2-29. Ismételjük meg a 25.2-23 gyakorlatot a Neumann-módszerrel.

25.2-30. Legyenek $N = 2$, $S_1 = S_2 = [0, 10]$, $f_1(x_1, x_2) = f_2(x_1, x_2) = 2x_1 + 2x_2 - (x_1 + x_2)^2$. Mutassuk meg, hogy mindkét kifizetőfüggvény szigorú konkáv (x_1 szerint f_1, és x_2 szerint f_2). Bizonyítsuk be, hogy ennek a játéknak végtelen sok egyensúlya van, tehát a kifizetőfüggvények szigorú konkavitásából nem következik az egyensúly egyértelműsége.

25.2-31. Lehet-e egy mátrixjáték átlósan szigorúan konkáv?

25.2-32. Tekintsük azt a kétszemélyes játékot, ahol a stratégiahalmazok $S_1 = S_2 = [0, 1]$, a kifizetőfüggvények $f_1(x_1, x_2) = -2x_1^2 + x_1(1 - x_2)$, $f_2(x_1, x_2) = -3x_2^2 + x_2(1 - x_1)$. Mutassuk meg, hogy ez a játék kielégíti a 25.16. tétel feltételeit.

25.2-33. Oldjuk meg a 25.2-32 gyakorlatot a (25.47)–(25.48)-ban adott algoritmussal.

25.3. Az oligopol feladat

Az eddigiekben általános módszereket mutattunk be az egyensúlyi probléma megoldására. Majdnem minden speciális játékosztályra vannak azonban olyan speciális módszerek, melyek az adott játékosztály tagjaira alkalmazhatók. Ezen fejezet további részében egy speciális játékot, az ***oligopol játékot*** vesszük górcső alá. Az oligopol játék egy olyan valós helyzetet ír le, amikor N vállalat azonos terméket gyárt vagy azonos szolgáltatást kínál. Ez a modell a ***klasszikus Cournot-modellként*** ismert. A játékosok stratégiái az x_k gyártási mennyiségek, melyek az $S_k = [0, L_k]$ stratégiahalmazokból valók, ahol L_k az adott játékos (vállalat) gyártási kapacitásának felső határa. Feltesszük, hogy a $p(s)$ piaci ár az összesen gyártott $s = x_1 + x_2 + \cdots + x_N$ mennyiségtől függ, és minden játékos $c_k(x_k)$ gyártási költsége csak a saját gyártási szintjétől függ. A vállalatok profitfüggvényei:

$$f_k(x_1, \ldots, x_N) = x_k p\left(\sum_{l=1}^{N} x_l\right) - c_k(x_k) \; . \tag{25.50}$$

Tehát definiáltuk a $G = \{N; S_1, \ldots, S_N; f_1, \ldots, f_N\}$ játékot.

Fel szokás tenni, hogy a p és c_k $(k = 1, 2, \ldots, N)$ függvények kétszer folytonosan differenciálhatók, továbbá

1. $p'(s) < 0$;
2. $p'(s) + x_k p''(s) \leq 0$;
3. $p'(s) - c_k''(x_k) < 0$;

minden k-ra, $x_k \in [0, L_k]$-ra, és $s \in [0, \sum_{l=1}^{N} L_l]$-re.

Az 1–3. feltételek mellett 25.3. tétel feltételei teljesülnek, tehát G-nek van legalább egy egyensúlya.

Legjobbválasz-leképezések

Vegyük észre, hogy az $s_k = \sum_{l \neq k} x_l$ jelölés mellett a k játékos kifizetőfüggvénye átírható a következő formába:

$$x_k p(x_k + s_k) - c_k(x_k) \ . \tag{25.51}$$

Mivel S_k kompakt halmaz, és a kifizetőfüggvény szigorúan konkáv x_k szerint, így rögzített s_k mellett a $\mathcal{P}_k$ játékos profitmaximalizáló gyártási szintje egyértelmű, mely a k játékos legjobb válasza – jelöljük ezt $B_k(s_k)$-val.

Könnyen látható, hogy három eset lehetséges: $B_k(s_k) = 0$, ha $p(s_k) - c_k'(0) \leq 0$, $B_k(s_k) = L_k$, ha $p(s_k + L_k) + L_k p'(s_k + L_k) - c_k'(L_k) \geq 0$, egyébként $B_k(s_k)$ az egyetlen megoldása a következő monoton egyenletnek:

$$p(s_k + x_k) + x_k p'(s_k + x_k) - c_k'(x_k) = 0 \ .$$

Tegyük fel, hogy $x_k \in (0, L_k)$. Ekkor s_k szerinti implicit deriválással

$$p'(1 + B_k') + B_k' p' + x_k p''(1 + B_k') - c_k'' B_k' = 0 \ ,$$

ahonnan

$$B_k'(s_k) = -\frac{p' + x_k p''}{2p' + x_k p'' - c_k''} \ .$$

Vegyük észre, hogy a 2–3. feltevések miatt

$$-1 < B_k'(s_k) \leq 0 \ , \tag{25.52}$$

mely egyenlőtlenség a töréspontok kivételével a másik két esetben is igaz.

A 25.2.1. pontnak megfelelően vezessük be a legjobbválasz-leképezést:

$$\mathbf{B}(x_1, \ldots, x_N) = \left(B_1\left(\sum_{l \neq 1} x_l\right), \ldots, B_N\left(\sum_{l \neq N} x_l\right)\right) \ . \tag{25.53}$$

A feladat a fenti leképezés fixpontjának megkeresése. Másik lehetőség, hogy bevezetünk egy dinamikus folyamatot, amely konvergál az egyensúlyhoz.

A fiktív lejátszás diszkrét módszeréhez hasonló módszert dolgozunk ki, melyben minden vállalat kiválasztja a legjobb válaszát versenytársai előző periódusban tett lépéseire:

$$x_k(t+1) = B_k(\sum_{l\neq k} x_l(t)) \qquad (k = 1, 2, \ldots, N)\ . \tag{25.54}$$

(25.52) miatt látható, hogy ($N = 2$)-re a jobb oldalon lévő leképezés $\mathbb{R}^2 \to \mathbb{R}^2$ kontrakció, tehát konvergens. Azonban, ha $N > 2$, akkor a konvergenciát nem lehet garantálni. Tekintsük most a fenti rendszer nyilvánvaló módosítását valamilyen $K_k > 0$-val:

$$x_k(t+1) = x_k(t) + K_k(B_k(\sum_{l\neq k} x_l(t)) - x_k(t)) \tag{25.55}$$

minden $k = 1, 2, \ldots, N$-re. Világos, hogy a fenti rendszer minden stabil állapota egyensúly, és bizonyítható, hogy ha K_k megfelelően kicsi, akkor az $x_k(0), x_k(1), x_k(2), \ldots$ sorozatok konvergensek minden $k = 1, 2, \ldots, N$-ra, és az egyensúlyhoz konvergálnak.

Tekintsük most a (25.55) modell folytonos alkalmazását, ahol (a Neumann-módszerhez hasonlóan) folytonos időskálát tételezünk fel:

$$\dot{x}_k(t) = K_k(B_k(\sum_{l\neq k} x_l(t)) - x_k(t)) \qquad (k = 1, 2, \ldots, N)\ . \tag{25.56}$$

A következő eredmény a fenti rendszer konvergenciájáról szól.

25.20. tétel. *Az 1–3. feltevések mellett a (25.56) rendszer aszimptotikusan stabil, azaz ha az $x_k(0)$ kezdetiértéket az egyensúlyhoz elég közelre választjuk, ekkor $x_k(t)$ tart az egyensúlyhoz minden k-ra.*

Bizonyítás. Elég azt megmutatni, hogy a (25.56) rendszer Jakobi-mátrixának sajátértékei negatív valós számok. Látható, hogy a Jakobi-mátrix a következő:

$$\mathbf{J} = \begin{pmatrix} -K_1 & K_1 b_1 & \cdots & K_1 b_1 \\ K_2 b_2 & -K_2 & \cdots & K_2 b_2 \\ \vdots & \vdots & & \vdots \\ K_N b_N & K_N b_N & \cdots & -K_N \end{pmatrix}\ , \tag{25.57}$$

ahol $b_k = B_k'(\sum_{l\neq k} x_l)$ az egyensúlyban. (25.52)-ből tudjuk, hogy $-1 < b_k \leq 0$ minden k-ra. A $\mathbf{J}$ sajátértékeinek kiszámítása céljából szükségünk van egy nagyon egyszerű, ám annál hasznosabb tényre. Tegyük fel, hogy az $\mathbf{a}$ és a $\mathbf{b}$

N valós komponensű vektorok. Ekkor

$$\det(\mathbf{I}+\mathbf{a}\mathbf{b}^T)=1+\mathbf{b}^T\mathbf{a}\,, \tag{25.58}$$

ahol $\mathbf{I}$ az N-edrendű egységmátrix. A (25.58) egyenlőtlenség teljes indukcióval könnyen bizonyítható. (25.58)-at felhasználva, a $\mathbf{J}$ mátrix karakterisztikus polinomja:

$$\begin{aligned}\phi(\lambda) &= \det(\mathbf{J}-\lambda\mathbf{I})=\det(\mathbf{D}+\mathbf{a}\mathbf{b}^T-\lambda\mathbf{I})\\ &= \det(\mathbf{D}-\lambda\mathbf{I})\det(\mathbf{I}+(\mathbf{D}-\lambda\mathbf{I})^{-1}\mathbf{a}\mathbf{b}^T)\\ &= \det(\mathbf{D}-\lambda\mathbf{I})[1+\mathbf{b}^T(\mathbf{D}-\lambda\mathbf{I})^{-1}\mathbf{a}]\\ &= \Pi_{k=1}^N(-K_k(1+b_k)-\lambda)[1+\sum_{k=1}^N\frac{K_kb_k}{-K_k(1+b_k)-\lambda}]\,,\end{aligned}$$

ahol a következő jelöléseket használtuk:

$$\mathbf{a}=\begin{pmatrix}K_1b_1\\K_2b_2\\\vdots\\K_Nb_N\end{pmatrix},\ \mathbf{b}^T=(1,1,\ldots,1),\ \mathbf{D}=\begin{pmatrix}-K_1(1+b_1)&&\\&\ddots&\\&&K_N(1+b_N)\end{pmatrix}.$$

Az első tényező gyökei negatívak: $\lambda=-K_k(1+b_k)$, a következő egyenlet gyökei adják a többi sajátértéket:

$$1+\sum_{k=1}^N\frac{K_kb_k}{-K_k(1+b_k)-\lambda}=0\,.$$

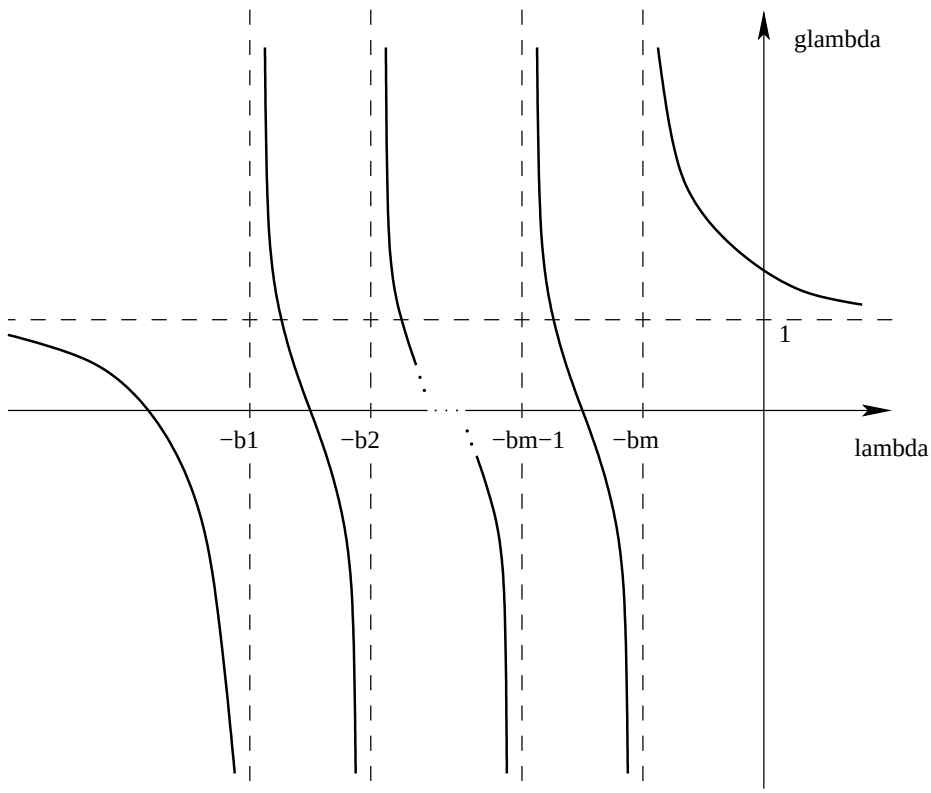
Vegyük észre, hogy közös nevezőre hozva és a tagokat összeadva az

$$1+\sum_{l=1}^m\frac{\alpha_k}{\beta_k+\lambda}=0 \tag{25.59}$$

egyenlőség adódik, ahol $\alpha_k,\beta_k>0$, és $\beta_1<\beta_2<\cdots<\beta_m$. Ha $g(\lambda)$ jelöli a bal oldalt, akkor a $\lambda=-\beta_k$-k a pólushelyek és

$$\lim_{\lambda\to\pm\infty}g(\lambda)=1,\quad \lim_{\lambda\to-\beta_k\pm0}g(\lambda)=\pm\infty\,,$$
$$g'(\lambda)=\sum_{l=1}^m\frac{-\alpha_l}{(\beta_l+\lambda)^2}<0\,,$$

így $g(\lambda)$ az értelmezési tartományának tetszőleges intervallumán szigorúan monoton fogyó. A függvény grafikonja a 25.7. ábrán látható. Vegyük észre,



25.7. ábra. A $g(\lambda)$ függvény görbéje.

hogy (25.59) ekvivalens egy m-edfokú polinom egyenletével, tehát m komplex (esetleg) valós gyöke van. A $g(\lambda)$ függvény tulajdonságai miatt egy gyök kisebb, mint $-\beta_1$, és egy-egy gyök van $-\beta_k$ és $-\beta_{k+1}$ között minden $k = 1, 2, \ldots, m-1$ esetén. Tehát minden gyök negatív valós szám. ■

(25.55) – az általános diszkrét modell – azonos módon vizsgálható. Ha $K_k = 1$ minden k-ra, akkor (25.55) visszavezethető a (25.54) egyszerű dinamikus rendszerre.

25.21. példa. *Első oligopol játék.* Tekintsünk egy háromszemélyes oligopol játékot, ahol az árfüggvény

$$p(s) = \begin{cases} 2 - 2s - s^2, & \text{ha} \quad 0 \le s \le \sqrt{3} - 1 \,, \\ 0 & \text{egyébként} \,, \end{cases}$$

a stratégiahalmazok $S_1 = S_2 = S_3 = [0, 1]$, a költségfüggvények

$$c_k(x_k) = kx_k^3 + x_k \qquad (k = 1, 2, 3) \,.$$

A $\mathcal{P}_k$ vállalat profitja a következő:

$$x_k(2 - 2s - s^2) - (kx_k^3 + x_k) = x_k(2 - 2x_k - 2s_k - x_k^2 - 2x_k s_k - s_k^2) - kx_k^3 - x_k \,.$$

A $\mathcal{P}_k$ vállalat legjobbválasz-függvényét a következőképpen kapjuk. A 25.3. alfejezet elején vázolt módszert követve, a következő három esetet különböztetjük meg. Ha $1 - 2s_k - s_k^2 \le 0$, akkor $x_k = 0$ a legjobbválasz. Ha $(-6 - 3k) - 6s_k - s_k^2 \ge 0$, akkor $x_k = 1$ a legjobbválasz. Egyébként a legjobb választ a következő egyenlet adja meg:

$$\begin{aligned} &\frac{\partial}{\partial x_k}[x_k(2 - 2x_k - 2s_k - s_k^2 - 2s_k x_k - x_k^2) - kx_k^3 - x_k] \\ &= 2 - 4x_k - 2s_k - s_k^2 - 4s_k x_k - 3x_k^2 - 3kx_k^2 - 1 = 0 \,, \end{aligned}$$

ahol az egyetlen pozitív megoldás

$$x_k = \frac{-(4 + 4s_k) + \sqrt{(4 + 4s_k)^2 - 12(1 + k)(s_k^2 + 2s_k - 1)}}{6(1 + k)} \,.$$

iután a legjobbválaszokat megtaláltuk, könnyedén megkonstruálhatjuk bármelyik korábban bemutatott módszert.

Visszavezetés egydimenziós fixpont feladatra

Tekintsünk egy N vállalatos (N személyes) oligopol játékot, ahol p az árfüggvény, és a c_k függvények a költségfüggvények $k = 1, 2, \ldots, N$ esetén. Vezessük be a

$$\Psi_k(s, x_k, t_k) = t_k p(s - x_k + t_k) - c_k(t_k) \,, \tag{25.60}$$

függvényt és definiáljuk az

$$X_k(s) = \{x_k | x_k \in S_k, \quad \Psi_k(s, x_k, x_k) = \max_{t_k \in S_k} \Psi_k(s, x_k, t_k)\} \tag{25.61}$$

leképezést minden $k = 1, 2, \ldots, N$-re. Legyen továbbá

$$X(s) = \{u | u = \sum_{k=1}^{N} x_k, \quad x_k \in X_k(s), \quad k = 1, 2, \ldots, N\} \; . \tag{25.62}$$

Vegyük észre, hogy ha $s \in [0, \sum_{k=1}^{N} L_k]$, akkor $X(s)$ minden komponense ebbe az intervallumba esik, tehát X egy egydimenziós pont-halmaz leképezés. Világos, hogy $(x_1^\star, \ldots, x_N^\star)$ pontosan akkor egyensúlya az N vállalatos oligopol játéknak, ha $s^\star = \sum_{k=1}^{N} x_k^\star$ fixpontja X-nek, és minden k-ra $x_k^\star \in X_k(s^\star)$. Tehát az egyensúlyi problémát leegyszerűsítettük egy egydimenziós pont-halmaz leképezés fixpont problémájára. Ez jelentős egyszerűsítés, hiszen a legjobbválaszok N dimenziós leképezések.

Ha az 1–3. feltételek teljesülnek, akkor $X_k(s)$ értéke egy egyelemű halmaz minden s-re, k-ra:

$$X_k(s) = \begin{cases} 0, & \text{ha } p(s) - c_k'(0) \le 0 \; , \\ L_k, & \text{ha } p(s) + L_k p_k'(s) - c_k'(L_k) \ge 0 \; , \\ z^\star & \text{egyébként} \; , \end{cases} \tag{25.63}$$

ahol $z^\star$ a következő monoton egyenlet egyetlen megoldása a $[0, L_k]$ intervallumon:

$$p(s) + z p'(s) - c_k'(z) = 0 \; . \tag{25.64}$$

A harmadik esetben a bal oldali kifejezés pozitív $z = 0$-ban, és negatív $z = L_k$-ban, a 2–3. feltételek miatt szigorúan fogyó, tehát egyetlen megoldás van.

Az egész $[0, \sum_{k=1}^{N} L_k]$ intervallumon $X_k(s)$ nemnövekvő konstans az első két esetben, és szigorúan fogyó a harmadik esetben. Tekintsük végül az egydimenziós egyenletet:

$$\sum_{k=1}^{N} X_k(s) - s = 0 \; . \tag{25.65}$$

$s = 0$-ban a bal oldal nemnegatív, $s = \sum_{k=1}^{N} L_k$-ban nempozitív, szigorúan fogyó. Tehát egyetlen megoldás van (az X fixpontja), mely megoldás bármely egydimenziós egyenletmegoldó módszerrel megkapható.

Legyen $[0, S_{max}]$ a (25.65) megoldásának értelmezési tartománya. K felező lépés után a pontosság $s_{max}/2^K$, mely kisebb, mint az $\epsilon > 0$ hibahatár, ha $K > \log_2(S_{max}/\epsilon)$.

25.22. példa. *Második oligopol játék.* Tekintsük megint a 25.21 példában látott háromszemélyes oligopol játékot. (25.63)-ból

$$X(s) = \begin{cases} 0, & \text{ha} \quad 1 - 2s - s^2 \le 0 \; , \\ 1, & \text{ha} \quad -(1 + 3k) - 4s - s^2 \ge 0 \; , \\ z^\star & \text{egyébként} \; , \end{cases}$$

ahol $z^\star$ a

$$3kz^2 + z(2s+2) + (-1+2s+s^2) = 0$$

egyenlet egyetlen megoldása. Az első eset akkor következik be, ha $s \geq \sqrt{2}-1$, a második eset soha nem következik be, míg a harmadik eset az egyetlen pozitív megoldás:

$$z^\star = \frac{-(2s+2) + \sqrt{(2s+2)^2 - 12k(-1+2s+s^2)}}{6k} \,. \tag{25.66}$$

Végül (25.65) speciális formája

$$\sum_{k=1}^{3} \frac{-(s+1) + \sqrt{(s+1)^2 - 3k(-1+2s+s^2)}}{3k} - s = 0 \,.$$

Az intervallumfelezéses módszeren alapuló program a következő megoldást adja: $s^\star \approx 0.2982$. (25.66)-ból az egyensúlyi stratégiák: $x_1^\star \approx 0.1077$, $x_2^\star \approx 0.0986$, $x_3^\star \approx 0.0919$.

A Kuhn-Tucker-feltételeken alapuló módszerek

Vegyük észre, hogy az N személyes oligopol játékok esetében $S_k = \{x_k | x_k \geq 0,\ L_k - x_k \geq 0\}$, tehát választhatjuk a

$$\mathbf{g}_k(x_k) = \begin{pmatrix} x_k \\ L_k - x_k \end{pmatrix} \tag{25.67}$$

függvényeket. Mivel a kifizetőfüggvények

$$f_k(x_1, \ldots, x_N) = x_k p(x_k + s_k) - c_k(x_k) \,, \tag{25.68}$$

a (26.9) Kuhn-Tucker-feltételek a következő formát öltik, ahol a két komponensű vektor $\mathbf{u}_k$ komponensei $u_k^{(1)}$ és $u_k^{(2)}$, és minden $k = 1, 2, \ldots, N$ indexre

$$\begin{aligned} u_k^{(1)}, u_k^{(2)} &\geq 0 \\ x_k &\geq 0 \\ L_k - x_k &\geq 0 \\ p(\textstyle\sum_{l=1}^N x_l) + x_k p'(\sum_{l=1}^N x_l) - c_k'(x_k) + (u_k^{(1)}, u_k^{(2)})\binom{1}{-1} &= 0 \\ u_k^{(1)} x_k + u_k^{(2)}(L_k - x_k) &= 0 \,. \end{aligned} \tag{25.69}$$

Két lehetőségünk van: vagy megkeressük (25.69) egy megengedett megoldását, vagy átírjuk (25.69)-et (26.10) alakú optimumszámítási feladattá, mely ebben a speciális esetben

$$\begin{aligned} \textstyle\sum_{k=1}^N (u_k^{(1)} x_k + u_k^{(2)}(L_k - x_k)) &\to \min \\ u_k^{(1)}, u_k^{(2)} &\geq 0 \ (k = 1, 2, \ldots, N) \\ x_k &\geq 0 \ (k = 1, 2, \ldots, N) \\ L_k - x_k &\geq 0 \ (k = 1, 2, \ldots, N) \\ p(\textstyle\sum_{l=1}^N x_l) + x_k p'(\sum_{l=1}^N x_l) - c_k'(x_k) + u_k^{(1)} - u_k^{(2)} &= 0 \ (k = 1, 2, \ldots, N) \,. \end{aligned} \tag{25.70}$$

A (25.69) vagy (25.70) számítási költsége a p és a c_k függvények típusától függ. Nem adható általános jellemzés.

25.23. példa. *Harmadik oligopol játék.* A 25.21 példában bevezetett játék esetén (25.70) alakja

$$\begin{aligned} \sum_{k=1}^{3}(u_k^{(1)}x_k + u_k^{(2)}(1-x_k)) &\to \min \\ u_k^{(1)}, u_k^{(2)} &\geq 0 \\ x_k &\geq 0 \\ 1 - x_k &\geq 0 \\ 1 - 2s - s^3 - 2x_k - 2x_k s - 3kx_k^2 + u_k^{(1)} - u_k^{(2)} &= 0 \\ x_1 + x_2 + x_3 &= s\ . \end{aligned}$$

Egy professzionális optimalizációs program használatával a következő megoldást kaptuk:

$$x_1^\star \approx 0.1077, \quad x_2^\star \approx 0.0986, \quad x_3^\star \approx 0.0919\ ,$$

és $u_k^{(1)} = u_k^{(2)} = 0$.

Visszavezetés komplementer feladatokra

Ha $(x_1^\star, \ldots, x_N^\star)$ egyensúly egy N személyes oligopol játékban, akkor rögzített $x_1^\star, \ldots, x_{k-1}^\star, x_{k+1}^\star, \ldots, x_N^\star$ esetén $x_k = x_k^\star$ maximumhelye a $\mathcal{P}_k$ játékos f_k kifizetőfüggvényének. Tegyük fel, hogy az 1–3. feltételek teljesülnek, f_k konkáv x_k-ban, így $x_k^\star$ pontosan akkor maximumhelye f_k-nak, ha az egyensúlyban

$$\frac{\partial f_k}{\partial x_k}(\mathbf{x}^\star) = \begin{cases} \leq 0, & \text{ha} \quad x_k^\star = 0\ , \\ = 0, & \text{ha} \quad 0 < x_k^\star < L_k\ , \\ \geq 0, & \text{ha} \quad x_k^\star = L_k\ . \end{cases}$$

Vezessük be a

$$z_k = \begin{cases} = 0, & \text{ha} \quad x_k > 0\ , \\ \geq 0, & \text{ha} \quad x_k = 0 \end{cases}$$

és a

$$v_k = \begin{cases} = 0, & \text{ha} \quad x_k < L_k\ , \\ \geq 0, & \text{ha} \quad x_k = L_k \end{cases}$$

túlcsordulás változókat és legyen

$$w_k = L_k - x_k\ . \tag{25.71}$$

(25.71) az egyensúlyban átírható, mint

$$\frac{\partial f_k}{\partial x_k}(\mathbf{x}) - v_k + z_k = 0\ . \tag{25.72}$$

A túlcsordulás változók definíciója miatt

$$z_k x_k = 0 \,, \tag{25.73}$$

$$v_k w_k = 0 \,. \tag{25.74}$$

A nemnegativitási feltételt hozzávéve

$$x_k, z_k, v_k, w_k \geq 0 \,, \tag{25.75}$$

mely esetben a (25.71)–(25.75) nemlineáris egyenlőtlenségrendszert kapjuk, mely ekvivalens az egyensúlyi feladattal.

A következőkben megmutatjuk, hogy (25.71)–(25.75) átírható nemlineáris komplementer feladattá. Az ilyen feladatok megoldására vannak közismert módszerek. Vezessük be a következő jelöléseket:

$$\mathbf{v} = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_N \end{pmatrix}, \quad \mathbf{L} = \begin{pmatrix} L_1 \\ L_2 \\ \vdots \\ L_N \end{pmatrix}, \quad \mathbf{h}(\mathbf{x}) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1}(\mathbf{x}) \\ \frac{\partial f_2}{\partial x_2}(\mathbf{x}) \\ \vdots \\ \frac{\partial f_N}{\partial x_N}(\mathbf{x}) \end{pmatrix},$$

$$\mathbf{t} = \begin{pmatrix} \mathbf{x} \\ \mathbf{v} \end{pmatrix}, \quad \text{és} \quad \mathbf{g}(\mathbf{t}) = \begin{pmatrix} -\mathbf{h}(\mathbf{x}) + \mathbf{v} \\ \mathbf{L} - \mathbf{x} \end{pmatrix}.$$

Ekkor a (25.72)–(25.75) rendszer átírható, mint

$$\begin{aligned} \mathbf{t} &\geq \mathbf{0} \\ \mathbf{g}(\mathbf{t}) &\geq \mathbf{0} \\ \mathbf{t}^T \mathbf{g}(\mathbf{t}) &= 0 \,. \end{aligned} \tag{25.76}$$

A fenti feladat a ***nemlineáris komplementer feladatok*** szokásos formája. Vegyük észre, hogy az utolsó feltétel azt követeli meg, hogy komponensenként vagy $\mathbf{t}$, vagy $\mathbf{g}(\mathbf{t})$, vagy mindkettő nulla legyen.

(25.76) számítási költsége a benne lévő függvények típusától, és a választott módszertől függ.

25.24. példa. *Negyedik oligopol játék.* A 25.21 példában bevezetett háromszemélyes oligopol játék elemzéséből kapjuk:

$$\mathbf{t} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ v_1 \\ v_2 \\ v_3 \end{pmatrix}$$

és

$$\mathbf{g}(\mathbf{t}) = \begin{pmatrix} -1 + 2\sum_{l=1}^{3} x_l + (\sum_{l=1}^{3} x_l)^2 + 2x_1 + 2x_1 \sum_{l=1}^{3} x_l + 3x_1^2 + v_1 \\ -1 + 2\sum_{l=1}^{3} x_l + (\sum_{l=1}^{3} x_l)^2 + 2x_2 + 2x_2 \sum_{l=1}^{3} x_l + 6x_2^2 + v_2 \\ -1 + 2\sum_{l=1}^{3} x_l + (\sum_{l=1}^{3} x_l)^2 + 2x_3 + 2x_3 \sum_{l=1}^{3} x_3 + 9x_3^2 + v_3 \\ 1 - x_1 \\ 1 - x_2 \\ 1 - x_3 \end{pmatrix} .$$

Lineáris oligopol játékok és kvadratikus programozás

Ebben a részben olyan N személyes oligopol játékokat fogunk vizsgálni, ahol mind az árfüggvény, mind a költségfüggvények lineárisak:

$$p(s) = As + B, \quad c_k(x_k) = b_k x_k + c_k \quad (k = 1, 2, \ldots, N) ,$$

ahol $B, b_k, c_k > 0$, de $A < 0$. Tegyük fel megint, hogy a stratégiahalmazok intervallumok: $[0, L_k]$. Ebben az esetben

$$f_k(x_1, \ldots, x_N) = x_k(Ax_1 + \cdots + Ax_N + B) - (b_k x_k + c_k) \tag{25.77}$$

minden k-ra, így

$$\frac{\partial f_k}{\partial x_k}(\mathbf{x}) = 2Ax_k + A\sum_{l \neq k} x_l + B - b_k . \tag{25.78}$$

A (25.71)–(25.75) feltételek ebben az esetben (nem az eredeti sorrendben):

$$\begin{aligned} 2Ax_k + A\sum_{l \neq k} x_l + B - b_k - v_k + z_k &= 0 \\ z_k x_k = v_k w_k &= 0 \\ x_k + w_k &= L_k \\ x_k, v_k, z_k, w_k &\geq 0 . \end{aligned}$$

Vezessük be a következő mátrixot és vektorokat:

$$\mathbf{Q} = \begin{pmatrix} 2A & A & \cdots & A \\ A & 2A & \cdots & A \\ \vdots & \vdots & & \vdots \\ A & A & \cdots & 2A \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} B \\ B \\ \vdots \\ B \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{pmatrix} ,$$

$$\mathbf{v} = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_N \end{pmatrix}, \quad \mathbf{w} = \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{pmatrix}, \quad \mathbf{z} = \begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_N \end{pmatrix}, \quad \text{és} \quad \mathbf{L} = \begin{pmatrix} L_1 \\ L_2 \\ \vdots \\ L_N \end{pmatrix} .$$

Foglaljuk össze a definiált egyenlőtlenségeket:

$$\begin{aligned} \mathbf{Qx}+\mathbf{B}-\mathbf{b}-\mathbf{v}+\mathbf{z} &= \mathbf{0} \\ \mathbf{x}+\mathbf{w} &= \mathbf{L} \\ \mathbf{x}^T\mathbf{z}=\mathbf{v}^T\mathbf{w} &= 0 \\ \mathbf{x},\mathbf{v},\mathbf{z},\mathbf{w} &\geq \mathbf{0}\,. \end{aligned} \tag{25.79}$$

A következőkben azt látjuk be, hogy $\mathbf{Q}$ negatív definit. Tetszőleges $\mathbf{a}=(a_i)$ nemnulla vektorra

$$\mathbf{a}^T\mathbf{Q}\mathbf{a} = 2A\sum_i a_i^2 + A\sum_i\sum_{j\neq i} a_i a_j = A(\sum_i a_i^2 + (\sum_i a_i)^2) < 0\,,$$

ahonnan következik a feltevésünk.

Vegyük észre, hogy (25.79)-ben a következő, szigorúan konkáv kvadratikus feladat Kuhn-Tucker-feltételei vannak:

$$\begin{aligned} \tfrac{1}{2}\mathbf{x}^T\mathbf{Q}\mathbf{x}+(\mathbf{B}-\mathbf{b})\mathbf{x} &\to \max \\ \mathbf{0}\leq\mathbf{x} &\leq \mathbf{L}\,. \end{aligned} \tag{25.80}$$

Mivel a megengedett megoldások halmaza korlátos poliéder, és a célfüggvény szigorúan konkáv, így a Kuhn-Tucker-feltételek szükségesek és elégségesek is egyben. Következésképpen, az $\mathbf{x}^\star$ vektor pontosan akkor egyensúly, ha (25.80) egyetlen optimális megoldása. (25.80) megoldására közismert módszerek találhatók az irodalomban.

Mivel (25.79) egy konvex kvadratikus feladat, így annak megoldására ismeretesek módszerek. A módszerek költsége különböző, tehát (25.79) számítási költsége a választott módszertől függ.

25.25. példa. *Kétszemélyes oligopol játék.* Tekintsünk egy duopol játékot (kétszemélyes oligopol játékot), ahol az árfüggvény $p(s)=10-s$, a költségfüggvények $c_1(x_1)=4x_1+1$ és $c_2(x_2)=x_2+1$, a kapacitáskorlátok $L_1=L_2=5$. Azaz

$$B=10,\quad A=-1,\quad b_1=4,\quad b_2=1,\quad c_1=c_2=1\,.$$

Tehát,

$$\mathbf{Q}=\begin{pmatrix}-2 & -1\\ -1 & -2\end{pmatrix},\quad \mathbf{B}=\begin{pmatrix}10\\10\end{pmatrix},\quad \mathbf{b}=\begin{pmatrix}4\\1\end{pmatrix},\quad \mathbf{L}=\begin{pmatrix}5\\5\end{pmatrix}.$$

A kvadratikus programozási feladat:

$$\begin{aligned} \frac{1}{2}(-2x_1^2-2x_1x_2-2x_2^2)+6x_1+9x_2 &\to \max \\ 0\leq x_1 &\leq 5 \\ 0\leq x_2 &\leq 5\,. \end{aligned}$$

Egyszerű deriválással látható, hogy a célfüggvény feltételek nélküli globális maximumát az $(x_1^\star,x_2^\star)^T=(1,4)^T$ pontban veszi fel. Mivel ez a pont benne van a

megengedett megoldások halmazában, így optimális megoldása a feladatnak, tehát a duopol játék egyetlen egyensúlya.

Gyakorlatok

25.3-1. Tekintsünk egy duopol játékot, ahol $S_1 = S_2 = [0,1]$, $p(s) = 2 - s$ és $c_1(x) = c_2(x) = x^2 + 1$. Vizsgáljuk meg az (25.55)-ben megadott iterációs eljárás konvergenciáját.

25.3-2. Legyen $N = 2$, $S_1 = S_2 = [0, 1.5]$, $c_k(x_k) = 1.5x_k$ $(k = 1, 2)$ és

$$p(s) = \begin{cases} 1.75 - 0.5s, & \text{ha} \quad 0 \le s \le 1.5\ , \\ 2.5 - s, & \text{ha} \quad 1.5 \le s \le 2.5\ , \\ 0, & \text{ha} \quad 2.5 < s\ . \end{cases}$$

Mutassuk meg, hogy végtelen sok egyensúly van:

$$\{(x_1^\star, x_2^\star) | 0.5 \le x_1 \le 1, \quad 0.5 \le x_2 \le 1, \quad x_1 + x_2 = 1.5\}\ .$$

25.3-3. Tekintsük a 25.3-1 gyakorlatban bevezetett duopol játékot.

a. Írjuk fel a legjobbválasz-függvényeket, és határozzuk meg az egyensúlyt.

b. Tekintsük a (25.62)-ben bevezetett egydimenziós fixpont feladatot, és határozzuk meg segítségével az egyensúlyt.

c. Írjuk fel a (25.69) Kuhn-Tucker-feltételeket.

d. Írjuk fel a (25.76) komplementer feladatot.

Feladatok

25-1 Fiktív lejátszás bimátrix-játékokra

Általánosítsuk a fiktív lejátszás módszerét bimátrix-játékokra.

25-2 Fiktív lejátszás véges játékokra

Általánosítsuk a fiktív lejátszás módszerét véges játékokra.

Megjegyzések a fejezethez

A játékelmélet témakörében eddig csak 1994-ben osztottak (közgazdasági) Nobel-díjat. A díjazottak között volt John Nash, aki a róla elnevezett Nash-egyensúly fogalmáért kapta a díjat. Ezt a fogalmat Nash 1951-ben vezette be [210].

Egy másik, szűkebb egyensúlyfogalmat alkalmaz a visszafelé indukció algoritmusa. Ezen algoritmus Kuhn nevéhez köthető és megtalálható a Kuhn

és Tucker által szerkesztett kötetben [163]. Mivel ezen algoritmus a Nash-egyensúlynál szigorúbb feltételű egyensúlyt határoz meg, az ezen módszerrel kapott egyensúly egyben Nash-egyensúly is.

Az egyensúly megtalálásának problémája, illetve magának az egyensúly létezésének problémája matematikailag a fixpontproblémának feleltethető meg. A különböző fixponttételek – mint a Brouwer-féle [37], a Kakutani-féle [143], a Tarski-féle [297] – segítségével bizonyítható az egyensúly létezése bizonyos játékosztályokban. Az egyensúly létezésének bizonyítására Kakutani-féle fixponttétellel lásd a [217] cikket.

Magának a fixpontnak (vagy fixpontoknak) a megkeresésére is ismertek módszerek, lásd például a [283] és [88] könyveket. A legnépszerűbb létezési eredmény Nikaido és Isoda nevéhez fűződik [217].

A Fan-egyenlőtlenséget, mely szerepet játszik a folytonos játékok egyensúlyának jellemzésében, részletesen tárgyalja könyvében Aubin [10]. A Kuhn-Tucker-feltételek leírása megtalálható Martos Béla könyvében [196]. A Kuhn-Tucker-feltételek eltérésváltozókkal átírhatók nemnegatív egyenletrendszerré, mely rendszerek megoldására [283] és [196] tartalmaznak numerikus módszereket.

A bimátrix-játékok egyensúlyi problémájának átírása kevert változós feladattá megtalálható Mills [204] és Shapiro [265] cikkében. A bimátrix-játékok egyensúlyi problémája felírható kvadratikus programozási feladatként is (lásd Mangasarian cikkét [194]).

A fiktív lejátszás módszere megtalálható részletesen Robinson cikkében [250]. A Neumann-módszer alkalmazásakor differenciálegyenletet kell megoldanunk – ehhez a Runge-Kutta-módszert használtuk. Ennek a módszernek a leírása megtalálható Szidarovszky könyvében [283].

Az átlósan szigorú konkáv játékok leírása Rosen cikkében [253] található meg. Az N személyes játékok egyensúlyának numerikus meghatározására Zuhovitzky, Polyak és Primak [323] javasoltak numerikus módszert.

A klasszikus Cournot-modell általánosítására nézve lásd Okuguchi és Szidarovszky könyveit [221, 222]. A 25.20. tétel bizonyítása a [279] cikkben található meg. A (25.58) lemma bizonyítására lásd a [222] monográfiát. Az intervallumfelezéses módszer leírását [283] tartalmazza. [146] olyan módszereket ír le, melyek alkalmasak nemlineáris komplementaritási feladatok megoldására. A (25.80) feladat megoldása megtalálható Hadley monográfiájában [114]. A nemlineáris programozással foglalkozik magyar nyelven Kovács Margit könyve [160].

A játékelmélet klasszikus tankönyve Neumann János és Oscar Morgenstern műve [212]. Magyar nyelven 1986-ban Szidarovszky Ferenc és Molnár Sándor [282], 1999-ben Kiss Béla és Krebsz Anna [154], 2004-ben pedig

Mészáros József [208] adtak közre tankönyvet.

New results are in [30, 186, 197, 261, 321].

26. Konfliktushelyzetek kezelése

A gyakorlati élet minden területén találkozunk olyan helyzetekkel, amikor egymásnak ellentmondó szempontokat kell egyidejűleg figyelembe vennünk. A probléma sokkal komolyabbá válik akkor, amikor több döntéshozó, érdekcsoport közös megegyezése kell ahhoz, hogy a megoldás kialakuljon.

A konfliktus helyzetek matematikai szempontból három csoportba oszthatók:

1. Egyetlen döntéshozónak kell döntenie több, ellentmondó szempont együttes figyelembevételével
2. Több döntéshozónak kell egy közös megoldást találnia, amikor mindegyik döntéshozó csak egy kritériumot vesz figyelembe
3. Több döntéshozó keres közös megoldást, de mindegyik döntéshozó egyszerre több kritériumot vesz figyelembe

Az első esetben a probléma többcélú optimalizációs probléma, mikor a különféle szempontok jelentik a célfüggvényeket. A második eset tipikus játékelméleti probléma, amikor a döntéshozók a játékosok, és a kritériumok jelentik a kifizető-függvényeket. A harmadik eset, mint Pareto-játékok szerepel az irodalomban, amikor az egyes játékosok optimum helyett csak Pareto-optimális megoldások megtalálására törekednek.

Ebben a fejezetben ennek a nagyon fontos, összetett témakörnek az alapjait fogjuk tárgyalni.

26.1. Többcélú programozás alapjai

Tegyük fel, hogy egyetlen döntéshozó kívánja a legjobb döntési alternatívát megtalálni több, általában ellentmondó kritérium alapján. A kritériumok általában döntési célokat reprezentálnak. Ezeket általában verbálisan fogalmazzák meg először, mint például tiszta levegő, olcsó üzemeltetés stb. A matematikai modell megfogalmazása előtt ezeket a célokat először kvantifikálható mutatókkal kell leírni. Gyakran előfordul, hogy egy-egy kritériumot egyszerre több mutató jellemez, ilyen például a levegő minősége, hiszen sokféle szeny-

nyezés egyidejű jelenléte befolyásolja annak minőségét. Matematikailag általában feltesszük, hogy az egyes mutatók (ezeket ***célfüggvényeknek*** fogjuk hívni a továbbiakban) nagyobb értéke kedvezőbb értéket jelent, így az összes célfüggvényt egyidejűleg kívánjuk maximalizálni. Ha valamelyik célfüggvényt eredetileg minimalizálni akarjuk, akkor nyugodtan beszorozhatjuk annak értékét (-1)-gyel, és az így nyert új célfüggvény már maximum jellegűvé válik. Ha valamelyik célfüggvény esetében valamilyen optimális érték elérése a cél, akkor az attól való eltérés (-1)-szeresét maximalizálhatjuk.

Ha X jelöli a lehetséges döntési alternatívák halmazát, és $f_i : X \to \mathbb{R}$ jelöli az i-edik célfüggvényt $(i = 1, 2, \ldots, I)$, akkor a feladat matematikailag a következőképpen fogalmazható meg:

$$f_i(x) \to max \qquad (i = 1, 2, \ldots, I) \; , \tag{26.1}$$

feltéve, hogy $x \in X$.

Egyetlen célfüggvény optimalizálása esetén ***optimális megoldást*** keresünk. Optimális megoldások eleget tesznek a következő feltételeknek:

(i) Optimális megoldás mindig jobb, mint bármely nem optimális megoldás.

(ii) Nincs olyan lehetséges megoldás, amely kedvezőbb célfüggvényeket biztosítana, mint egy optimális megoldás.

(iii) Ha egyszerre több optimális megoldás létezik, akkor azok ekvivalensek olyan szempontból, hogy azonos célfüggvényekkel rendelkeznek.

Ezek a tulajdonságok azonnal adódnak abból az egyszerű tényből, hogy a ***következménytér,***

$$H = \{u | u = f(x) \;\text{ adott }\; x \in X \text{ mellett}\} \tag{26.2}$$

a valós számegyenes részhalmaza, amely teljesen rendezett. Több célfüggvény esetén a

$$H = \{\mathbf{u} = (u_1, \ldots, u_I) | u_i = f_i(x), i = 1, 2, \ldots, I \;\text{ adott } x \in X \text{ mellett}\} \tag{26.3}$$

következménytér az I-dimenziós euklideszi tér részhalmaza, amely csak félig rendezett halmaz. Egy másik bonyodalom származik abból is, hogy általában nem létezik olyan döntési alternatíva, amely az összes célfüggvényt egyszerre maximalizálja. Jelölje

$$f_i^\star = \max\{f_i(x) | x \in X\} \tag{26.4}$$

az i-edik célfüggvény egyedi maximumát, akkor az

$$\mathbf{f}^\star = (f_1^\star, \ldots, f_I^\star)$$

pontot ***ideális pontnak*** nevezzük. Ha $\mathbf{f}^\star \in H$, akkor létezik olyan $x^\star$ döntés, amelyre $f_i(x^\star) = f_i^\star, i = 1, 2, \ldots, I$ esetén. Ilyen speciális esetekben $x^\star$ kielégíti az előzőekben tárgyalt (i)-(iii) feltételeket. Ha azonban $\mathbf{f}^\star \notin H$, akkor a helyzet ennél sokkal bonyolultabb. Optimális megoldások helyett ilyenkor un. Pareto-optimális megoldásokat keresünk.

26.1. definíció. *Egy $x \in X$ alternatívát **Pareto-optimálisnak** nevezünk, ha nincs olyan $\overline{x} \in X$, amelyre $f_i(\overline{x}) \geq f_i(x)$ $i = 1, 2, \ldots, I$ esetén és legalább egy i esetén szigorú egyenlőtlenség áll fenn.*

Egy többcélú optimalizációs probléma nem feltétlenül rendelkezik Pareto-optimális megoldással, mint azt a

$$H = \{(f_1, f_2) | f_1 + f_2 < 1\}$$

halmaz esete mutatja. Minthogy H nyílt halmaz, tetszőleges $(f_1, f_2) \in H$ és elég kis pozitív ϵ_1 és ϵ_2 mellett $(f_1 + \epsilon_1, f_2 + \epsilon_2) \in H$.

26.2. tétel. *Ha X korlátos, zárt egy véges dimenziós euklideszi térben és az összes célfüggvény folytonos, akkor létezik Pareto-optimális megoldás.*

A következő két példa egy diszkrét és egy folytonos problémát mutat be.

26.1. példa. Tegyük fel, hogy egy szennyvíztisztító állomás tervezésénél két alternatíva közül kell választani. Az első alternatíva költsége két milliárd Ft, és napi kapacitása 1500 m^3. A másik alternatíva drágább, 3 milliárd Ft napi 2000 m^3 kapacitással. Ebben az esetben $X = \{1, 2\}$, $f_1 = -$költség, $f_2 =$ kapacitás. Az alábbi táblázat foglalja össze az adatokat:

Alternatíva	f_1	f_2
1	-2	1500
2	-3	2000

26.1. ábra. Szennyvíztisztító állomás tervezése.

Mindkét alternatíva Pareto-optimális, hiszen $-2 > -3$ és $2000 > 1500$. A H következménytér két pontból áll: $(-2, 1500)$ és $(-3, 2000)$.

26.2. példa. Három technológiai variáns optimális kombinációját használják egy szennyvíztisztító állomáson. Az első variáns 3,2,1 mg/m^3 szennyezést távolít el az egyik fajta szennyezésből, és 1,3,2 mg/m^3 mennyiséget a másik fajta szennyezésből. Jelölje x_1, x_2 és $1 - x_1 - x_2$ a három technológia változat százalékos összetételét. A

korlátozó feltételek:

$$\begin{aligned} x_1, x_2 &\geq 0 \\ x_1 + x_2 &\leq 1 \,, \end{aligned}$$

az eltávolított szennyezés mennyisége:

$$\begin{aligned} 3x_1 + 2x_2 + (1 - x_1 - x_2) &= 2x_1 + x_2 + 1 \\ x_1 + 3x_2 + 2(1 - x_1 - x_2) &= -x_1 + x_2 + 2 \,. \end{aligned}$$

Minthogy a harmadik tag konstans, a következő két célfüggvényes optimum-feladatot kapjuk:

$$2x_1 + x_2, -x_1 + x_2 \longrightarrow max$$

feltéve, hogy

$$\begin{aligned} x_1, x_2 &\geq 0 \\ x_1 + x_2 &\leq 1 \,. \end{aligned}$$

A H következménytér a következőképpen határozható meg. Az

$$\begin{aligned} f_1 &= 2x_1 + x_2 \\ f_2 &= -x_1 + x_2 \end{aligned}$$

egyenletekből

$$x_1 = \frac{f_1 - f_2}{3} \quad \text{és} \quad x_2 = \frac{f_1 - 2f_2}{3} \,,$$

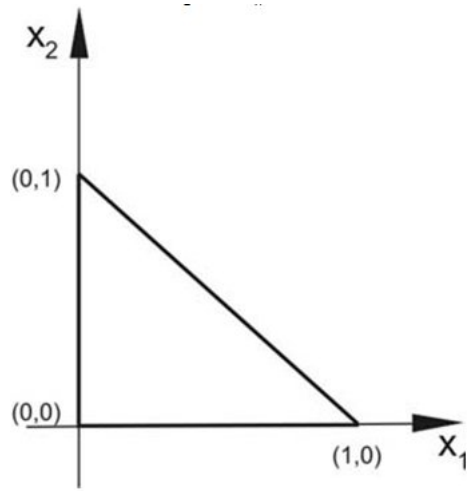
és a korlátozó feltételekből a következő feltételek adódnak az f_1 és f_2 célfüggvényekre:

$$\begin{aligned} x_1 \geq 0 &\iff f_1 - f_2 \geq 0 \\ x_2 \geq 0 &\iff f_1 + 2f_2 \geq 0 \\ x_1 + x_2 \leq 1 &\iff 2f_1 + f_2 \leq 3 \,. \end{aligned}$$

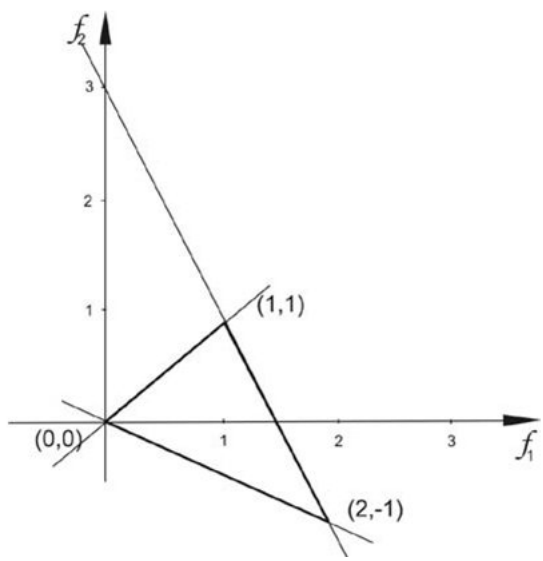
A 26.2. és **??**. ábrán az X és H halmazt ábrázoljuk.

A H halmaz képe alapján világos, hogy az $(1,1)$ és $(2,-1)$ pontokat összekötő egyenes szakasz pontjai a Pareto-optimális pontok. A $(2,-1)$ pont egyik lehetséges H-beli pontnál sem jobb, mert az első célfüggvényben a lehető legrosszabb síkokat szolgáltatja. A szakasz pontjai sem ekvivalensek egymással, az $(1,1)$ pontból lefelé haladva a $(2,1)$ pont irányába az első célfüggvény növekszik, a második pedig állandóan csökken. Így az optimális megoldásoknál látott (ii) és (iii) tulajdonság nem marad érvényben a többcélú esetben.

Az előző példában láttuk, hogy különböző Pareto-optimális megoldások különböző célfüggvény értékeket adnak, így alapvető fontosságú annak eldöntése, hogy egy-egy konkrét esetben melyiküket válasszuk. Erre a kérdésre ad választ a többcélú programozás módszertana. A legtöbb módszer alapja az, hogy valamilyen valós értékű „érték-függvénnyel" helyettesíti a célfüggvényeket, azaz a célfüggvények által generált preferenciát egyetlen valós értékű függvénnyel helyettesíti. Ebben az alfejezetben a leggyakrabban alkalmazott többcélú programozási módszerekkel foglalkozunk.



26.2. ábra. Az X halmaz képe.



26.3. ábra. A H halmaz képe.

26.1.1. Hasznossági függvények alkalmazása

Természetes módszert jelent a következő. Minden egyes célfüggvényhez hozzárendelünk egy-egy hasznossági függvényt. Legyen $u_i(f_i(x))$ az i-edik célfüggvény hasznossági függvénye. Az u_i függvény konstrukciója a

hasznossági függvények elméletében megszokott módon történhet, például a döntéshozó szubjektíven definiálhatja az u_i értékeket adott f_i értékek mellett, majd az így adódó pontokra folytonos függvényt illeszthetünk. Additíven független hasznossági függvény esetén additív, hasznosságban független hasznossági függvény esetében pedig vagy additív vagy multiplikatív összevont hasznossági függvényalakot kaphatunk. Azaz, az összevont hasznossági függvény alakja vagy

$$u(\mathbf{f}) = \sum_{i=1}^{I} k_i u_i(f_i) \tag{26.5}$$

vagy

$$ku(\mathbf{f}) + 1 = \prod_{i=1}^{I} k k_i u_i(f_i) + 1 \ . \tag{26.6}$$

Ilyen esetekben a többcélú optimalizációs feladat egyetlen célfüggvényűvé írható át:

$$u(\mathbf{f}) \longrightarrow max \tag{26.7}$$

feltéve, hogy $x \in X$, és ily módon $u(\mathbf{f})$ jelenti az "értékfüggvényt".

26.3. példa. Tekintsük ismét az előző példában szereplő döntési problémát. Az első célfüggvény értékkészlete $[0, 2]$, a második célfüggvényé pedig $[-1, 1]$. Lineáris hasznossági függvényeket feltételezve

$$u_1(f_1) = \frac{1}{2}(f_1) \ \text{ és } \ u_2(f_2) = \frac{1}{2}(f_2) + 1 \ .$$

Tegyük fel továbbá, hogy a döntéshozó megadta az

$$u(0, -1) = 0, u(2, 1) = 1, \ \text{ és az } \ u(0, 1) = \frac{1}{4}$$

értékeket. Lineáris összevont hasznossági függvényt feltételezve

$$u(f_1, f_2) = k_1 u_1(f_1) + k_2 u_2(f_2) \ ,$$

és az adott értékek alapján

$$\begin{aligned} 0 &= k_1 0 + k_2 0 \\ 1 &= k_1 1 + k_2 1 \\ \frac{1}{4} &= k_1 0 + k_2 1 \ . \end{aligned}$$

A harmadik egyenletből $k_2 = \frac{1}{4}$, a másodikból pedig $k_1 = \frac{3}{4}$, úgyhogy

$$u(f_1, f_2) = \frac{3}{4}u_1(f_1) + \frac{1}{4}u_2(f_2) = \frac{3}{4}\frac{1}{2}(2x_1 + x_2) + \frac{1}{4}\frac{1}{2}(-x_1 + x_2 + 1) = \frac{5}{8}x_1 + \frac{4}{8}x_2 + \frac{1}{8} \ .$$

Tehát a következő egyetlen célfüggvénnyel rendelkező feladatot oldjuk meg:

$$\frac{5}{8}x_1 + \frac{4}{8}x_2 \longrightarrow max$$

feltéve, hogy

$$\begin{aligned} x_1, x_2 &\geq 0 \\ x_1 + x_2 &\leq 1\,. \end{aligned}$$

Könnyen látható, hogy az optimális megoldás: $x_1 = 1$, $x_2 = 0$, vagyis az első technológiát kell csak alkalmazni.

Tegyük fel, hogy n a célfüggvények száma és a döntéshozó megad N vektort: $(f_1^{(l)}, \ldots, f_n^{(l)})$ és a hozzátartozó $u^{(l)}$ összevont hasznossági függvényértékeket. Ekkor a $k_1, \ldots, k_n$ együtthatókat a

$$k_1 u_1(f_1^{(l)}) + \cdots + k_n u_n(f_n^{(l)}) = u^{(l)} \qquad (l = 1, 2, \ldots, N)$$

egyenletrendszer megoldásával kaphatjuk. Mindig feltesszük, hogy $N \geq n$, így legalább annyi egyenletünk van mint az ismeretlenek száma. Ha az egyenletrendszer ellentmondást tartalmaz, akkor a legkisebb négyzetek módszerével a legjobban illeszkedő megoldást határozzuk meg. Tegyük fel, hogy

$$\mathbf{U} = \begin{pmatrix} u_{11} & \cdots & u_{1n} \\ u_{21} & \cdots & u_{2n} \\ \vdots & & \vdots \\ u_{N1} & \cdots & u_{Nn} \end{pmatrix} \quad \text{és} \quad \mathbf{u} = \begin{pmatrix} u^{(1)} \\ u^{(2)} \\ \vdots \\ u^{(N)} \end{pmatrix}\,.$$

A formális algoritmus ekkor a következő:

HASZNOSSÁGI-FÜGGVÉNY-MÓDSZER($\mathbf{u}$)

```
1  for i = 1 to N
2      for j ← 1 to n
3          u_ij = u_j(f_j^(i))
4  k = (U^T U)^-1 U^T u a megoldásvektor
5  return k
```

26.1.2. Súlyozásos módszer

Ennek a módszernek az alkalmazásakor az értékfüggvényt az eredeti célfüggvények lineáris kombinációjaként választjuk, vagyis a

$$\sum_{i=1}^{I} \alpha_i f_i(x) \longrightarrow max \qquad (x \in X) \tag{26.8}$$

feladatot oldjuk meg. Ha az egyes célfüggvényeket különböző dimenzióban mérjük, akkor az összevont célfüggvény nem értelmezhető, ugyanis különböző mértékegységű tagokat adunk össze. Ilyenkor a célfüggvényeket normalizálni szoktuk. Legyen m_i és M_i az f_i célfüggvény minimuma és maximuma az X halmazon. A normált i-edik célfüggvény ekkor az

$$\overline{f_i}(x) = \frac{f_i(x) - m_i}{M_i - m_i}$$

képlettel adódik, és a (26.8) problémában f_i helyett $\overline{f_i}$ szerepel:

$$\sum_{i=1}^{I} \alpha_i \overline{f_i}(x) \longrightarrow max \qquad (x \in X) \tag{26.9}$$

Kimutatható, hogyha az összes α_i súly pozitív, (26.9) optimális megoldásai Pareto-optimálisak az eredeti feladatot nézve.

26.4. példa. Tekintsük ismét a 26.2. példa esetét. A 26.3. ábrából látjuk, hogy $m_1 = 0$, $M_1 = 2$, $m_2 = -1$, és $M_2 = 1$. Tehát a normált célfüggvények:

$$\overline{f_1}(x_1, x_2) = \frac{2x_1 + x_2 - 0}{2 - 0} = x_1 + \frac{1}{2}x_2$$

és

$$\overline{f_2}(x_1, x_2) = \frac{-x_1 + x_2 + 1}{1 + 1} = -\frac{1}{2}x_1 + \frac{1}{2}x_2 + \frac{1}{2} \ .$$

Tegyük fel hogy a célfüggvények egyformán fontosak, így azonos súlyokat választunk: $\alpha_1 = \alpha_2 = \frac{1}{2}$, így az összevont célfüggvény:

$$\frac{1}{2}(x_1 + \frac{1}{2}x_2) + \frac{1}{2}(-\frac{1}{2}x_1 + \frac{1}{2}x_2 + \frac{1}{2}) = \frac{1}{4}x_1 + \frac{1}{2}x_2 + \frac{1}{4} \ .$$

Könnyen látható, hogy az X halmazon az optimális megoldás:

$$x_1 = 0, x_2 = 1 \ ,$$

azaz csak a második technológia variánst kell alkalmazni.

Tegyük fel, hogy $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_I)$. A formális algoritmus ekkor a következő.

Súlyozásos-módszer(α)

```
1  for i = 1 to I
2      m_i = (f_i(x) ⟶ min)
3      M_i = (f_i(x) ⟶ max)
4  k = (∑_{i=1}^{I} α_i f̄_i ⟶ max)
5  return k
```

26.1.3. Távolságfüggő módszerek

Amennyiben normáljuk a célfüggvényeket, az egyedi normált célfüggvények legkedvezőbb értéke 1 és legkedvezőtlenebb értéke 0. Az $\mathbf{1} = (1, 1, \ldots, 1)$ pont tehát az ideális pont és a $\mathbf{0} = (0, 0, \ldots, 0)$ pont a legrosszabb hozamvektor.

Távolságfüggő módszerek esetén vagy az $\mathbf{1}$ vektorhoz kívánunk legközelebb jutni, vagy a $\mathbf{0}$ ponttól legtávolabb, így vagy a

$$\varrho(\mathbf{f}(x), \mathbf{1}) \longrightarrow min \qquad (x \in X) \tag{26.10}$$

vagy a

$$\varrho(\mathbf{f}(x), \mathbf{0}) \longrightarrow max \qquad (x \in X) \tag{26.11}$$

feladatot oldjuk meg, ahol ϱ valamilyen $\mathbb{R}^I$-beli távolságfüggvényt jelent. A gyakorlati alkalmazásokban a következő távolságfüggvények szerepelnek leggyakrabban:

$$\varrho_1(\mathbf{a}, \mathbf{b}) = \sum_{i=1}^{I} \alpha_i |a_i - b_i| \tag{26.12}$$

$$\varrho_2(\mathbf{a}, \mathbf{b}) = \left(\sum_{i=1}^{I} \alpha_i |a_i - b_i|^2 \right)^{\frac{1}{2}} \tag{26.13}$$

$$\varrho_\infty(\mathbf{a}, \mathbf{b}) = \max_i \{\alpha_i |a_i - b_i|\} \tag{26.14}$$

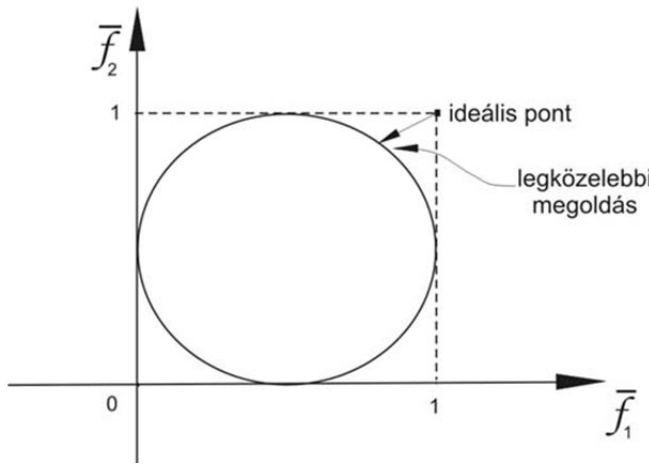
$$\varrho_g(\mathbf{a}, \mathbf{b}) = \prod_{i=1}^{I} |a_i - b_i|^{\alpha_i} \ . \tag{26.15}$$

A $\varrho_1, \varrho_1, \varrho_\infty$ távolságok a közismert Minkowski-féle metrikák $p = 1, 2, \infty$ esetben. A ϱ_g geometriai távolság nem elégíti ki a távolságfüggvények szokásos feltételeit, azonban mégis gyakran alkalmazzuk a gyakorlatban. Mint azt mi is látni fogjuk a későbbiekben, a klasszikus Nash-féle konfliktus-feloldási algoritmus is a geometriai távolságot használja. Könnyen kimutathatjuk, hogy a ϱ_1 távolságra épülő módszerek ekvivalensek a súlyozási módszerrel. Vegyük észre először, hogy

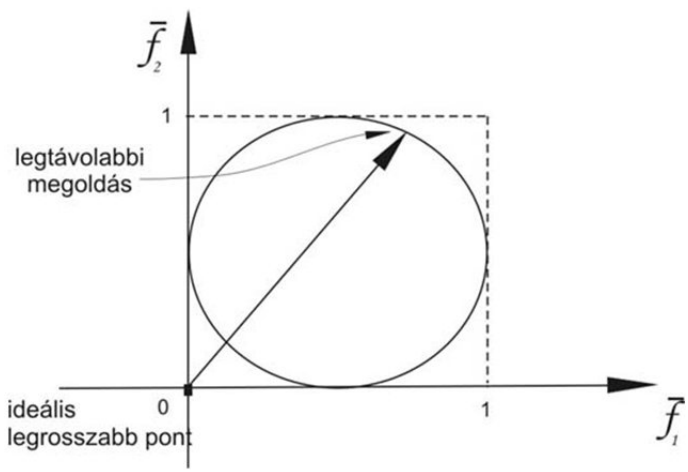
$$\varrho_1(\mathbf{f}(x), \mathbf{1}) = \sum_{i=1}^{I} \alpha_i |f_i(x) - 1| = \sum_{i=1}^{I} \alpha_i |1 - f_i(x)| = \sum_{i=1}^{I} \alpha_i - \sum_{i=1}^{I} \alpha_i f_i(x) \ , \tag{26.16}$$

ahol az első tag konstans, a második tag pedig a súlyozásos módszer célfüggvénye. Hasonlóan,

$$\varrho_1(\mathbf{f}(x), \mathbf{0}) = \sum_{i=1}^{I} \alpha_i |f_i(x) - 0| = \sum_{i=1}^{I} \alpha_i (f_i(x) - 0) = \sum_{i=1}^{I} \alpha_i f_i(x) \tag{26.17}$$



26.4. ábra. Távolság minimalizálás.



26.5. ábra. Távolság maximalizálás.

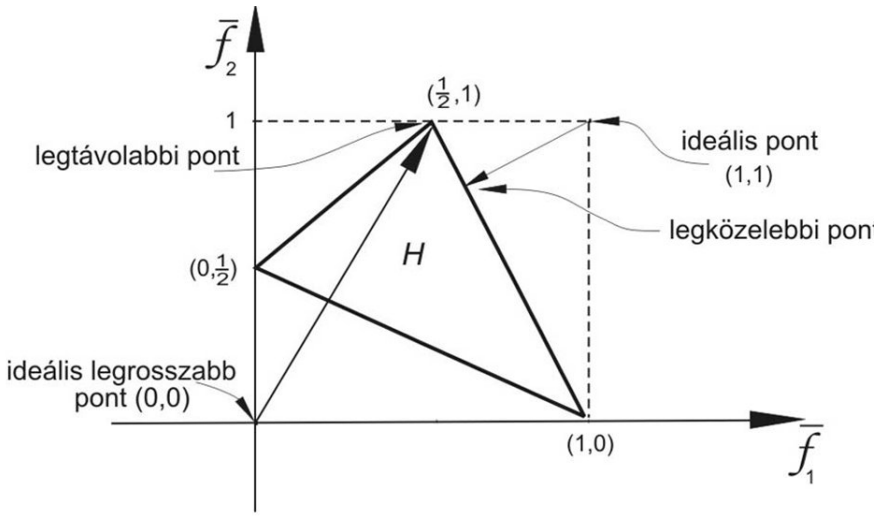
mely a súlyozásos módszer célfüggvénye.

A módszert a 26.4. és 26.5. ábra mutatja.

26.5. példa. Tekintsük ismét az előző példa problémáját. A normalizált következményeket a 26.6. ábra mutatja. A két koordináta:

$$\overline{f_1} = \frac{f_1}{2} \quad \text{és} \quad \overline{f_2} = \frac{f_2 + 1}{2} \, .$$

Az $\alpha_1 = \alpha_2 = \frac{1}{2}$ és a ϱ_2 távolság választásával az ideális ponthoz legközelebbi



26.6. ábra. A normált H halmaz képe.

$\overline{H}$-beli pont

$$\overline{f_1} = \frac{3}{5}, \overline{f_2} = \frac{4}{5} .$$

Így

$$f_1 = 2\overline{f_1} = 2x_1 + x_2 = \frac{6}{5} \quad \text{és} \quad f_2 = 2\overline{f_1} - 1 = -x_1 + x_2 = \frac{3}{5} ,$$

vagyis az optimális döntés:

$$x_1 = \frac{1}{5}, x_2 = \frac{4}{5}, 1 - x_1 - x_2 = 0 .$$

Tehát csak az első két technológiát kell alkalmazni 20% és 80% arányban.

Válasszunk újra egyforma súlyokat ($\alpha_1 = \alpha_2 = \frac{1}{2}$) és a ϱ_2 távolságot, viszont keressük meg az ideálisan legrosszabb ponttól a legtávolabbi $\overline{H}$-beli pontot. A 26.5. ábrából látjuk, hogy a megoldás

$$\overline{f_1} = \frac{f_1}{2}, \overline{f_2} = 1 ,$$

így

$$f_1 = 2\overline{f_1} = 1, f_2 = 2\overline{f_2} - 1 = 1 .$$

Tehát az optimális döntés: $x_1 = 0$ és $x_2 = 1$

A formális algoritmus a következő:

TÁVOLSÁGFÜGGŐ-MÓDSZER($\varrho, \mathbf{f}$)

```
1  for i = 1 to I
2      m_i = (f_i(x) ⟶ min)
3      M_i = (f_i(x) ⟶ max)
4      f̄_i(x) = (f_i(x) − m_i)/(M_i − m_i)
5  k = (ϱ(f̄(x), 1) ⟶ min) vagy k = (ϱ(f̄(x), 0) ⟶ max)
6  return k
```

26.1.4. Irányfüggő módszerek

Tegyük fel, hogy rendelkezésünkre áll egy olyan $\mathbf{f}_*$ pont a H halmazon, amelyen javítani szeretnénk. Az $\mathbf{f}_*$ a jelenlegi helyzetet jelenti, amelyen a döntéshozó javítani kíván, vagy tervezési szinten a lehető legrosszabb pontot választhatjuk kiinduló pontnak. Feltesszük továbbá, hogy a döntéshozó megad egy javítási irányvektort, amelyet $\mathbf{v}$ jelöl. A feladat ezután az, hogy $\mathbf{f}_*$-ből kiindulva a $\mathbf{v}$ irányvektor mentén a lehető legtávolabb jussunk el a H halmazon. Matematikailag tehát a

$$t \longrightarrow max \qquad (\mathbf{f}_* + t\mathbf{v} \in H) \tag{26.18}$$

optimumfeladatot oldjuk meg, és a hozzátartozó döntést az

$$\mathbf{f}(x) = \mathbf{f}_* + t\mathbf{v} \tag{26.19}$$

egyenlet megoldása adja az optimális t érték mellett. A módszert grafikusan az 26.7. ábra ábrázolja.

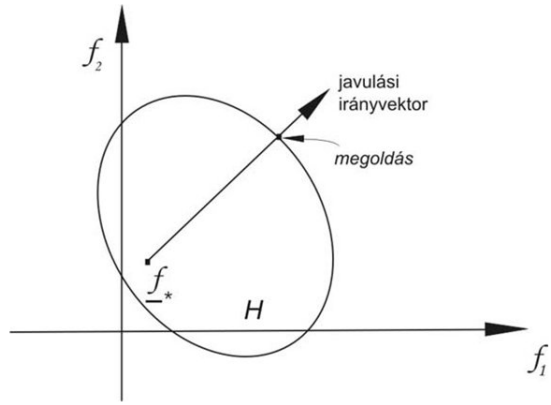
26.6. példa. Tekintsük ismét a 26.2. példa problémáját és tegyük fel, hogy $\mathbf{f}_* = (0, -1)$, ami a lehető legrosszabb célfüggvényértékeket tartalmazza komponenseiben. Ha egyenlő mértékben kívánunk a célfüggvényeken javítani, akkor a $\mathbf{v} = (1, 1)$ választással kell élnünk. A grafikus megoldást a 26.8. ábra mutatja, amely szerint

$$f_1 = \frac{4}{3} \text{ és } f_2 = \frac{1}{3} ,$$

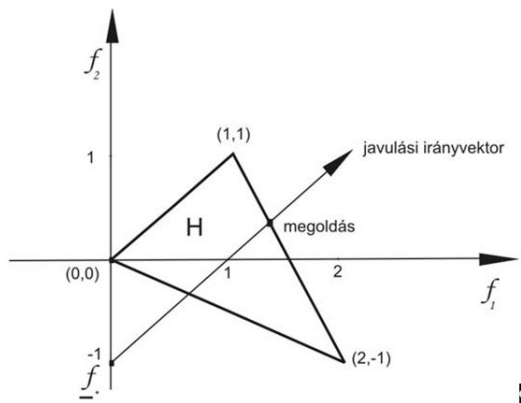
így a megfelelő döntési változó értékek a következők:

$$x_1 = \frac{1}{3} \text{ és } x_2 = \frac{2}{3} .$$

A módszernek egy nagyon ritkán alkalmazott változata az, amikor a nem lehetséges ideális pontból szisztematikusan (azaz egy adott irány mentén) csökkentjük a célfüggvényértékeket egészen addig, amíg lehetséges megoldás



26.7. ábra. Irányfüggő módszerek.



26.8. ábra. A 26.6 példa grafikus megoldása.

adódik. Ha $\mathbf{f}^*$ jelöli azt az ideális pontot, akkor a (26.18) optimumfeladat a következőképpen módosul:

$$t \longrightarrow min \qquad (\mathbf{f}^* - t\mathbf{v} \in H) \tag{26.20}$$

és a megfelelő döntést az

$$\mathbf{f} = \mathbf{f}^* - t\mathbf{v} \tag{26.21}$$

egyenlet megoldásából nyerhetjük.

26.7. példa. Visszatérve az előző példára tegyük fel, hogy $\mathbf{f}^* = (2,1)$ és $\mathbf{v} = (1,1)$, azaz azonos mértékben kívánjuk a célfüggvényeket csökkenteni. A 26.9. ábrán mutatjuk a probléma grafikus megoldását, ahol könnyen látható, hogy ugyanaz a

26.9. ábra. A 26.7 példa grafikus megoldása.

megoldás adódik, mint az előző példa esetében.

A módszer alkalmazását a (26.18) vagy a (26.20) optimumfeladat megoldása jelenti, és a megfelelő optimális döntést a (26.19) vagy a (26.21) egyenlet megoldása adja.

Gyakorlatok

26.1-1. Határozzuk meg a H következményteret a következő gyakorlathoz:

$$x_1 + x_2 \longrightarrow max \qquad x_1 - x_2 \longrightarrow max$$

feltéve, hogy

$$\begin{aligned} x_1, x_2 &\geq 0 \\ 3x_1 + x_2 &\leq 3 \\ x_1 + 3x_2 &\leq 3\,. \end{aligned}$$

26.1-2. Tegyük fel, hogy a döntéshozó hasznossági függvényei: $u_1(f_1) = f_1$ és $u_2(f_2) = \frac{1}{2}f_2$. Tegyük fel továbbá, hogy a döntéshozó megadta az $u(0,0) = 0, u(1,0) = u(0,1) = \frac{1}{2}$ értékeket is. Határozzuk meg a lineáris összevont hasznossági függvény alakját.

26.1-3. Oldjuk meg az 26.1-1. gyakorlatot a súlyozásos módszerrel a célfüggvények normalizálása nélkül. Válasszuk az $\alpha_1 = \alpha_2 = \frac{1}{2}$ súlyokat.

26.1-4. Ismételjük meg az előző gyakorlatot, de normalizáljuk a célfüggvényeket.

26.1-5. Oldjuk meg az 26.1-1. gyakorlatot normalizált célfüggvényekkel, $\alpha_1 = \alpha_2 = \frac{1}{2}$ súlyokkal és

(i) ϱ_1 távolság minimalizálásával

(ii) ϱ_2 távolság minimalizálásával

(iii) ϱ_∞ távolság minimalizálásával

26.1-6. Ismételjük meg az előző gyakorlatot, de távolság minimalizálás helyett maximalizáljuk a távolságot a $\mathbf{0}$ vektortól.

26.1-7. Oldjuk meg az 26.1-1 gyakorlatot irányfüggő módszerrel az $\mathbf{f}_* =$

$(0,-1)$ és $\mathbf{v}=(1,1)$ választással.
26.1-8. Ismételjük meg az előző gyakorlatot az $\mathbf{f}_*=(\frac{3}{2},1)$ és $\mathbf{v}=(1,1)$ választással.

26.2. Egyensúlypontok módszere

Ebben az alfejezetben feltesszük, hogy I döntéshozó érdekelt egy közös $x\in X$ döntési alternatíva kiválasztásában. Jelölje $f_i : X \mapsto \mathbb{R}$ az i-edik döntéshozó célfüggvényét, amelyet a játékelméleti irodalomban kifizetőfüggvénynek is nevezünk. A döntéshozók egymáshoz való viszonyától függően nem-kooperatív és kooperatív játékról beszélünk. Az első esetben az egyes döntéshozók csak saját hasznukkal törődnek, a másodikban pedig valamiféle megállapodásra törekednek, amikor mindegyikük jobban jár, mint a nem-kooperatív esetben. Ebben az alfejezetben a nem-kooperatív esetet tárgyaljuk, a kooperatív eset a következő alfejezet témája lesz majd.

Jelölje $i=1,2,\ldots,I$ és $x\in X$ esetén $H_i(x)$ azon döntési alternatívák halmazát, amelyekbe az i-edik döntéshozó közvetlenül áttérhet x-ből a többiek segítsége nélkül. Nyilvánvalóan $H_i(x)\subseteq X$.

26.3. definíció. *Egy $x^*\in X$ alternatíva egyensúlypontot jelent, ha tetszőleges i és $x\in H_i(x^*)$ mellett*

$$f_i(x)\le f_i(x^*)\ . \tag{26.22}$$

Ez a definíció úgy is megfogalmazható, hogy x^* stabilis abból a szempontból, hogy egyik döntéshozó sem tudja a döntési alternatívát x^*-ból egyedül megváltoztatni úgy, hogy célfüggvényérték javuljon. Nem-kooperatív játékok esetén az egyensúlypontok jelentik a játék megoldását.

Tetszőleges $x\in X$ és i döntéshozó esetén az

$$L_i(x)=\{z|z\in H_i(x)\ \text{ és minden }\ y\in H_i(x)\ \text{ esetén }\ f_i(z)\ge f_i(y)\} \tag{26.23}$$

halmazt, amelyet az i-edik döntéshozónak az x alternatívára adott legjobb válaszai halmazának is nevezzük. Világos, hogy $L_i(x)$ elemei jelentik azokat az alternatívákat amelyekre x-ből áttérhet az i-edik döntéshozó és az összes ilyen lehetőség közül a legkedvezőbb célfüggvényt biztosít számára. A (26.22) egyenlőtlenségből az is világos, hogy x^* egyensúlypont akkor és csak akkor, ha $i=1,2,\ldots,I$ esetén $x^*\in L_i(x^*)$, azaz x^* közös fixpontja az L_i ponthalmaz leképezéseknek. Az egyensúlypontok létezése tehát visszavezethető pont-halmaz leképezések közös fixpontjának létezésére, így ez a probléma a szokásos módszerekkel tárgyalható.

		$i=2$	
		1	2
$i=1$	1	$(1,2)$	$(2,1)$
	2	$(2,4)$	$(0,5)$

26.10. ábra. Játék egyensúlypont nélkül.

Igen gyakori az az eset, amikor az egyes döntéshozók egyéni döntéseiből áll össze a kollektív döntés. Jelölje X_i az i-edik döntéshozó lehetséges alternatíváinak a halmazát, $x_i \in X_i$ a konkrét alternatívákat, és legyen $f_i(x_1,\ldots,x_I)$ az i-edik döntéshozó célfüggvénye. Azaz a kollektív döntés $x=(x_1,\ldots,x_I)\in X_1\times X_2\times\cdots\times X_I = X$. Ebben az esetben

$$H_i(x_1,\ldots,x_I)=\{(x_1,\ldots,x_{i-1},z_i,x_{i+1},\ldots,x_I)|z_i\in X_i\}$$

és a (26.22) egyensúlypont definíció a következőképpen módosul:

$$f_i(x_1^*,\ldots,x_{i-1}^*,x_i,x_{i+1}^*,\ldots,x_I^*)\le f_i(x_i^*,\ldots,x_I^*)\ . \tag{26.24}$$

A játékelméleti irodalomban az egyensúlypontot ***Nash-féle egyensúlypontnak*** is nevezzük.

Egyensúlypontok létezése nem garantált általában, ennek illusztrálására tekintsük az $I=2$ esetet, amikor mindkét döntéshozó 2 alternatíva között választhat: $X_1=\{1,2\}$ és $X_2=\{1,2\}$. A célfüggvényértékeket a 26.10. ábra mutatja, ahol a zárójelbeli első szám az első, a második szám pedig a második döntéshozó célfüggvényértékét mutatja. Amennyiben létezik is egyensúlypont, annak egyértelműsége nem garantált, amit a konstans célfüggvények esete igazol, amikor minden döntési alternatíva egyensúlypontot jelent.

Amennyiben az $X_1,\ldots,X_I$ halmazok végesek, az egyensúlypont könnyen megtalálható a teljes leszámolás módszerével, amikor az összes döntési vektor $\mathbf{x}=(x_1,\ldots,x_I)$ esetén leellenőrizzük, hogy az x_i komponens kicserélhető-e úgy, hogy az f_i célfüggvény növekedjék. Ha igen, $\mathbf{x}$ nem egyensúlypont. Ha egyetlen komponens sem cserélhető ki ily módon, akkor $\mathbf{x}$ egyensúlypont. Az algoritmus formális leírására tegyük fel, hogy $X_1=\{1,2,\ldots,n_i\}$.

EGYENSÚLYPONT-KERESÉS

```
1  for i_1 = 1 to n_1
2      for i_2 = 1 to n_2
3          ...
4          for i_I = 1 to n_I
5              kulcs ← 0
6              for k = 1 to n
7                  for j = 1 to n_k
8                      if f_k(i_1, ..., i_{k-1}, j, i_{k+1}, ..., i_I) >
                          f(i_1, ..., i_I)
9                          kulcs = 1 és ugorjunk a 10. sorra
10                 if   kulcs = 0
11                     (i_1, ..., i_I) egyensúlypont
```

Egyensúlypont létezését garantálja a következő tétel.

26.4. tétel. *Tegyük fel, hogy $i = 1, 2, \ldots, I$ esetén*

(i) *X_i konvex, korlátos és zárt egy véges dimenziós euklideszi térben*

(ii) *f_i folytonos az X halmazon*

(iii) *rögzített $x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_I$ mellett f_i konkáv x_i-ben.*

Ekkor legalább egy egysúlypont létezik.

Az egyensúlypontok meghatározása általában azon az észrevételen alapul, hogy minden i esetére x_i^* az

$$f_i(x_1^*, \ldots, x_{i-1}^*, x_i, x_{i+1}^*, \ldots, x_I^*) \longrightarrow max \qquad (x_i \in X_i) \tag{26.25}$$

optimumfeladat megoldása. Az optimális megoldás szükséges feltételeit (például a Kuhn-Tucker feltételeket) felírva egy egyenlőség-egyenlőtlenségrendszert kaphatunk, amely megoldásai között szerepelnek az egyensúlypontok. Ennek a módszernek az illusztrálására tegyük fel, hogy

$$X_i = \{x_i | g_i(x_i) \geq 0\}$$

ahol x_i véges dimenziós vektor és g_i vektor értékű függvény. Ily módon (26.25) a következőképpen írható át:

$$f_i(x_1^*, \ldots, x_{i-1}^*, x_i, x_{i+1}^*, \ldots, x_I^*) \longrightarrow max \qquad (g_i(x_i) \geq 0)\ . \tag{26.26}$$

A Kuhn-Tucker szükséges feltételek ebben az esetben a következők:

$$\begin{aligned} u_i &\geq 0 \\ g_i(x_i) &\geq 0 \\ \nabla_i f_i(x) + u_i^T \nabla_i g_i(x_i) &= 0^T \\ u_i^T g_i(x_i) &= 0\ , \end{aligned} \tag{26.27}$$

ahol ∇_i az x_i szerinti gradiensképzést jelöli, és u_i olyan vektor, amely hossza megegyezik g_i hosszával. Ha a (26.27) feltételeket $i = 1, 2, \ldots, I$ esetére felírjuk, egy egyenlet-egyenlőtlenség-rendszert kapunk, amely megoldására vannak számítógépes módszerek. Könnyen látható, hogy (26.27) átírható egy nemlineáris optimalizálási feladattá is:

$$\begin{aligned} \textstyle\sum_{i=1}^{I} u_i^T g_i(x_i) &\longrightarrow min \\ u_i &\geq 0 \\ g_i(x_i) &\geq 0 \\ \nabla_i f_i(x) + u_i^T \nabla_i g_i(x_i) &= 0^T\ . \end{aligned} \tag{26.28}$$

Ha az optimális célfüggvényérték pozitív, akkor a (26.27) rendszernek nincs megoldása, és ha az optimális célfüggvényérték zérus, akkor bármely optimális megoldás egyben megoldja a (26.27) rendszert is, így az optimális megoldások között kell keresnünk az egyensúlypontokat. A Kuhn–Tucker féle feltételek elégségességéről tudjuk, hogyha f_i konkáv x_i-ben minden i esetén, akkor a Kuhn–Tucker feltételek elégségesek is, így (26.27) bármely megoldása egyensúlypontot szolgáltat.

Az algoritmus formálisan a következő:

KUHN–TUCKER-EGYENSÚLYPONT

```
1  for i = 1 to I
2      g_i = ∇_i f_i
3      J_i = ∇_i g_i(x_i)
4  (x_1, ..., x_I) = a (26.28) optimumfeladat megoldása
5  if Σ_{i=1}^{I} u_i^T g_i(x_i) > 0
6     return "nincs egyensúlypont"
7     else return (x_1, ..., x_I)
```

26.8. példa. Tegyük fel, hogy I termelőüzem gyárt valamilyen víztisztító berendezést, amelyet háztartásokba adnak el. Jelölje x_i az i-edik termelő által előállított mennyiséget, $c_i(x_i)$ a költségfüggvényét, és $p(\sum_{j=1}^{I} x_j)$ az eladási árat, amely a teljes piacra bocsátott mennyiségtől függ. Legyen továbbá L_i az i-edik termelő kapacitása. A lehetséges X_i döntési halmaz tehát a $[0, L_i]$ zárt intervallum, amely az

$$\begin{aligned} x_i &\geq 0 \\ L_i - x_i &\geq 0 \end{aligned} \tag{26.29}$$

feltételekkel írható le, így

$$g_i(x_i) = \begin{pmatrix} x_i \\ L_i - x_i \end{pmatrix} .$$

Az i-edik termelő célfüggvénye annak profitja:

$$f_i(x_1, \ldots, x_n) = x_i p(x_1 + \cdots + x_n) - c_i(x_i) \ . \tag{26.30}$$

Minthogy $g_i(x_i)$ kétdimenziós, u_i is kételemű vektor, és a (26.28) optimumfeladat a következő alakú:

$$\begin{array}{rcl} \sum_{i=1}^{I}(u_i^{(1)} x_i + u_i^{(2)}(L_i - x_i)) & \longrightarrow & min \\ u_i^{(1)}, u_i^{(2)} & \geq & 0 \\ x_i & \geq & 0 \\ L_i - x_i & \geq & 0 \\ p(\sum_{j=1}^{I} x_j) + x_i p'(\sum_{j=1}^{I} x_j) - c_i'(x_i) + (u_i^{(1)}, u_i^{(2)})\binom{1}{-1} & = & 0 \ . \end{array} \tag{26.31}$$

Vezessük be az $\alpha_i = u_i^{(1)} - u_i^{(2)}$ új változót, és a könnyebb jelölés érdekében legyen $\beta_i = u_i^{(2)}$, akkor az utolsó feltétel figyelembevételével a következő problémát nyerjük:

$$\begin{array}{rcl} \sum_{i=1}^{I}(-x_i(p(\sum_{j=1}^{I} x_j) + x_i p'(\sum_{j=1}^{I} x_j) - c_i'(x_i)) + \beta_i L_i) & \longrightarrow & min \\ \beta_i & \geq & 0 \\ x_i & \geq & 0 \\ x_i & \leq & L_i \ . \end{array} \tag{26.32}$$

Vegyük észre, hogy optimum esetén $\beta_i = 0$, így a célfüggvény utolsó tagját el is hagyhatjuk.

Tekintsük az $I = 3$, $c_i(x_i) = ix_i^3 + x_i$, $L_i = 1$, $p(s) = 2 - 2s - s^3$ speciális esetet. A (26.32) feladat most a következőképpen egyszerűsödik:

$$\begin{array}{rcl} \sum_{i=1}^{3} x_i(2 - 2s - s^2 - 2x_i - 2x_i s - 3ix_i^2 - 1) & \longrightarrow & max \\ x_i & \geq & 0 \\ x_i & \leq & 1 \\ x_1 + x_2 + x_3 & = & s \ . \end{array} \tag{26.33}$$

Egy egyszerű számítógépes program felhasználásával az optimális megoldás:

$$x_1^* = 0.1077, x_2^* = 0.0986, x_3^* = 0.0919 \ ,$$

és egyben ez az egyensúlypont is.

Gyakorlatok

26.2-1. Legyen $I = 2$, $X_1 = X_2 = [0, 1]$, $f_1(x_1, x_2) = x_1 + x_2 - x_1^2$, $f_2(x_1, x_2) = x_1 + x_2 - x_2^2$. Írjuk fel a (26.27) feltételrendszert és oldjuk is meg.

26.2-2. Az előző gyakorlathoz írjuk fel a (26.28) optimum feladatot és oldjuk is meg.

26.2-3. Legyen most is $I = 2$. $X_1 = X_2 = [-1, 1]$, $f_1(x_1, x_2) = -(x_1 + x_2)^2 + x_1 + 2x_2$, $f_2(x_1, x_2) = -(x_1 + x_2)^2 + 2x_1 + x_2$. Ismételjük meg a 26.2-1. gyakorlatot.

26.2-4. Ismételjük meg a 26.2-2. gyakorlatot az előző gyakorlatbeli problémához.

26.3. Kooperatív játékok módszerei

Az előző alfejezethez hasonlóan jelölje ismét I a döntéshozók számát, X_i az i-edik döntéshozó döntési halmazát és $x_i \in X_i$ a konkrét döntési alternatívákat. Jelölje továbbá $f_i(x_1, \ldots, x_I)$ az i-edik döntéshozó célfüggvényét. Legyen S a döntéshozók valamilyen részhalmaza, amelyet játékelméletben általában ***koalíciónak*** is szoktak nevezni. Tetszőleges $S \subseteq \{1, 2, \ldots, I\}$ mellett vezessük be a

$$v(S) = \max_{x_i \in X_i} \min_{x_j \in X_j} \sum_{k \in S} f_k(x_1, \ldots, x_I) \qquad (i \in S, j \notin S) \tag{26.34}$$

függvényt, amelyet az $1, 2, \ldots, I$ halmaz összes részhalmazán definiált ***karakterisztikus függvénynek*** is nevezünk, ha a (26.34) definíciót kiegészítjük a $v(\emptyset) = 0$ és a

$$v(\{1, 2, \ldots, I\}) = \max_{x_i \in X_i} \sum_{k=1}^{I} f_k(x_1, \ldots, x_I) \qquad (1 \le i \le I)$$

két szélsőséges esettel.

Tegyük ismét fel, hogy az X_i halmazok végesek: $X_i = \{1, 2, \ldots, n_i\}$, $i = 1, 2, \ldots, I$ esetén. Legyen S egy koalíció. A $v(S)$ érték a következő algoritmussal adódik. Itt $|S|$ jelöli S elemeinek számát és $k_1, k_2, \ldots, k_{|S|}$ az elemeket, és $l_1, l_2, \ldots, l_{I-|S|}$ az S-be nem tartozó elemeket.

KARAKTERISZTIKUS-FÜGGVÉNY(S)

```
1  v(S) = −M, ahol M egy nagyon nagy pozitív szám
2  for i_1 = 1 to n_{k_1}
3      ⋱
4      for i_{|S|} = 1 to n_{k_{|S|}}
5          for j_1 = 1 to n_{l_1}
6              ⋱
7              for j_{I−|S|} = 1 to n_{l_{I−|S|}}
8                  Z = M, ahol M egy nagy pozitív szám
9                  V = Σ_{t=1}^{|S|} f_{i_t}(i_1, ..., i_{|S|}, j_1, ..., j_{I−|S|})
10                 keyif V < Z
11                     Z ← V
12             if Z > v(S)
13                v(S) = Z
14 return v(S)
```

26.9. példa. Térjünk vissza az előző példában tárgyalt döntési problémához, és tegyük fel, hogy $I = 3$, $L_i = 3$, $p(\sum_{i=1}^{I} x_i) = 10 - \sum_{i=1}^{I} x_i$ és $c_i(x_i) = x_i + 1$, $i = 1, 2, 3$ esetén. Minthogy a költségfüggvények azonosak, a célfüggvények is azonosak:

$$f_i(x_1, x_2, x_3) = x_i(10 - x_1 - x_2 - x_3) - (x_i + 1) \ .$$

A következőkben meghatározzuk a karakterisztikus függvényt. Legyen először $S = \{i\}$, ekkor

$$v(S) = \max_{x_i} \min_{x_j} \{x_i(10 - x_1 - x_2 - x_3) - (x_i + 1)\} \qquad (j \neq i) \ .$$

Minthogy a függvény szigorúan csökken az $x_j (i \neq j)$ változókban, minimális értéke $x_j = 3$ esetén adódik, így

$$v(S) = \max_{i} x_i(4 - x_i) - (x_i + 1) = \max_{0 \leq x_i \leq 3} (-x_i^2 + 3x_i - 1) = \frac{5}{4} \ ,$$

ami egyszerű differenciálással könnyen látható. Hasonlóan az $S = \{i, j\}$ esetben

$$v(S) = \max_{i,j} \min_{k \neq i,j} \{(x_i + x_j)(10 - x_1 - x_2 - x_3) - (x_i + 1) - (x_j + 1)\} \ .$$

Hasonlóan az előző esethez a minimum az $x_k = 3$ esetben adódik, úgyhogy

$$v(S) = \max_{0 \leq x_i, x_j \leq 3} \{(x_i + x_j)(7 - x_i - x_j) - (x_i + x_j + 2)\} = \max_{0 \leq x \leq 6} \{x(7 - x) - (x + 2)\} =$$

$$= \max_{0 \leq x \leq 6} \{-x^2 + 6x - 2\} = 7$$

ahol bevezettük az új $x = x_i + x_j$ változót. Az $S = \{1, 2, 3\}$ esetben

$$v(S) = \max_{0 \le x_1, x_2, x_3 \le 3} \{(x_1+x_2+x_3)(10-x_1-x_2-x_3)-(x_1+1)-(x_2+1)-(x_3+1)\} =$$

$$= \max_{0 \le x \le 9} \{x(10-x)-(x+3)\} = \max_{0 \le x \le 9} \{-x^2+9x-3)\} = 17.25 \ ,$$

ahol most bevezettük az $x = x_1 + x_2 + x_3$ változót.

A (26.34) definíció úgy is értelmezhető, hogy a $v(S)$ karakterisztikus függvény-érték az S koalíció ***garantált*** együttes célfüggvényértékét adja meg függetlenül attól, hogy a többiek mit csinálnak. Kooperatív játékok elméletének és gyakorlatának az a központi kérdése, hogy az együttesen maximálisan elérhető $v(\{1, 2, \ldots, I\})$ együttes hasznon az egyes döntéshozók hogyan osztozzanak meg. Egy $(\phi_1, \phi_2, \ldots, \phi_I)$ elosztást ***imputációnak*** szoktak nevezni, ha

$$\phi_i \ge v(\{i\}) \tag{26.35}$$

$i = 1, 2, \ldots, I$ esetén és

$$\sum_{i=1}^{I} \phi_i = v(\{1, 2, \ldots, I\}) \ . \tag{26.36}$$

A (26.35)–(26.36) egyenlőtlenségrendszernek általában végtelen sok imputáció tesz eleget, így további feltételeket kell kikötnünk annak érdekében, hogy az imputáció-halmaz egy speciális elemét kiválaszthassuk. Hasonló helyzettel találkozhatunk a többcélú programozás tárgyalásánál is, amikor egy-egy speciális Pareto-optimális megoldást keresünk meg a konkrét módszerek alkalmazásával.

26.10. példa. Az előző esetben egy (ϕ_1, ϕ_2, ϕ_3) vektor akkor imputáció, ha

$$\begin{aligned} \phi_1, \phi_2, \phi_3 &\ge 1.25 \\ \phi_1+\phi_2, \phi_1+\phi_3, \phi_2+\phi_3 &\ge 7 \\ \phi_1+\phi_2+\phi_3 &= 17.2 \ . \end{aligned}$$

A legnépszerűbb megoldási koncepció a ***Shapley-érték,*** amely a következőképpen adható meg:

$$\phi_i = \sum_{S \subseteq \{1,2,\ldots,I\}} \frac{(s-1)!(I-s)!}{I!}(v(S) - v(S - \{i\})) \ , \tag{26.37}$$

ahol s jelöli az S koalíció elemeinek a számát.

Tegyük fel, hogy a döntéshozók teljesen kooperálnak, azaz az {1,2,. . . ,I}

koalíciót hozzák létre, és az egyes döntéshozók véletlen sorrendben csatlakoznak a koalícióhoz. A $v(S) - v(S - \{i\})$ különbség az i-edik döntéshozónak az S koalícióhoz való hozzájárulását mutatja, a (26.37) kifejezés pedig ugyanennek a döntéshozónak az átlagos hozzájárulását. Kimutatható, hogy $(\phi_1, \phi_2, \ldots, \phi_I)$ egy imputáció.

A Shapley-értékek számítása a következő algoritmussal történhet:

SHAPLEY-ÉRTÉK

```
1  for ∀S ⊆ {1, ..., I}
2      v(S) = KARAKTERISZTIKUS-FÜGGVÉNY(S)
3  for i = 1 to I
4      alkalmazzuk (26.37) -et φ_i kiszámítására
```

26.11. példa. Az előző példában kiszámítottuk a karakterisztikus függvény értékét. A szimmetria miatt $\phi_1 = \phi_2 = \phi_3$ kell legyen a Shapley-érték esetén. Minthogy $\phi_1+\phi_2+\phi_3 = v(\{1,2,3\}) = 17.25, \phi 1 = \phi_2 = \phi_3 = 5.75$. Ugyanezt az értéket kapjuk a (26.37) képlettel is. Tekintsük a ϕ_1 értéket. Ha $i \notin S$, akkor $v(S) = v(S - \{i\})$, így az összeg megfelelő tagjai zérus értékűek. A nemzérus tagok az $S = \{1\}, S = \{1,2\}, S = \{1,3\}$ és $S = \{1,2,3\}$ koalíciók esetében adódnak, így

$$\phi_1 = \frac{0!2!}{3!}(\frac{5}{4} - 0) + \frac{1!1!}{3!}(7 - \frac{5}{4}) + \frac{1!1!}{3!}(7 - \frac{5}{4}) + \frac{2!0!}{3!}(\frac{69}{4} - 7) =$$

$$\frac{1}{6}(\frac{10}{4} + \frac{23}{4} + \frac{23}{3} + \frac{82}{4}) = \frac{138}{24} = 5.75\ .$$

Egy alternatív megoldási koncepció a megoldás stabilitását követeli meg. Azt mondjuk, hogy egy vektor $\phi = (\phi_1, \ldots, \phi_I)$ majorálja a $\psi = (\psi_1, \ldots, \psi_I)$ vektort az S koalícióban, ha

$$\sum_{i \in S} \phi_i > \sum_{i \in S} \psi_i\ ,$$

azaz az S koalíció érdeke a ϕ kifizetési vektorról áttérni a ψ kifizetési vektorra, vagy ψ instabil az S koalícióra nézve. A ***Neumann-Morgenstern megoldás*** imputációk olyan V halmaza, amelyre

(i) Nincs olyan $\phi, \psi \in V$, hogy ϕ majorálja ψ-t valamilyen koalícióban (belső stabilitás)

(ii) Ha $\psi \notin V$, akkor van olyan $\phi \in V$, hogy ϕ majorálja ψ-t legalább egy koalícióban (külső stabilitás).

Ennek a koncepciónak az a fő nehézsége, hogy nincs általános egzisztencia tétel nem üres V halmaz létezésére, és nem ismert általános módszer a V halmaz konstrukciójára.

Gyakorlatok

26.3-1. Legyen $I = 3$, $X_1 = X_2 = X_3 = [0, 1]$, $f_i(x_1, x_2, x_3) = x_1 + x_2 + x_3 - x_i^2$ $(i = 1, 2, 3)$. Határozzuk meg a $v(S)$ karakterisztikus függvényt.

26.3-2. Az előző gyakorlatban adott játék esetére írjuk fel a (26.35), (26.36) feltételrendszert.

26.3-3. Határozzuk meg a ψ_i Shapley-értékeket a 26.3-1. gyakorlatban megadott játék esetére.

26.4. Csoportos döntéshozatal

Az első alfejezetekben feltettük, hogy a célfüggvényeket numerikus értékek írják le. Ezek a numerikus értékek preferenciákat is jelentenek, hiszen az i-edik döntéshozó az x alternatívát preferálja z-vel szemben, ha $f_i(x) > f_i(z)$. Ebben az alfejezetben olyan módszerekkel fogunk foglalkozni, amelyek nem igénylik célfüggvények ismeretét, csak az egyes döntéshozók preferenciáit.

Jelölje I ismét a döntéshozók számát, X a döntési alternatívák halmazát. Ha az i-edik döntéshozó az x alternatívát jobbnak tartja, mint az y alternatívát, akkor azt $x \succ_i y$ jelöli, ha x-et jobbnak vagy azonosan jónak tartja, mint az y alternatívát, akkor ezt $x \succeq_i y$ jelöli. Feltesszük, hogy

(i) Tetszőleges $x, y \in X$ esetén $x \succeq_i y$ vagy $y \succeq_i x$ (vagy mindkettő)

(ii) $x \succeq_i y$ és $y \succeq_i z$ esetén $x \succeq_i z$.

Az (i) feltétel megköveteli, hogy a $\succeq_i$ félig rendezés teljes legyen, a (ii) feltétel pedig azt, hogy tranzitív legyen.

26.5. definíció. *Egy csoportos döntéshozatali leképezés tetszőleges* $(\succeq_1, \succeq_2, \ldots, \succeq_I)$ *egyéni félig rendezéseket egyetlen félig rendezéssé sűrít, amelyet a csoport közös preferencia-struktúrájának is nevezünk.*

Néhány egyszerű példán illusztráljuk a csoportos döntéshozatali leképezés fogalmát.

26.12. példa. Legyen $x, y \in X$ tetszőleges, és az összes i esetén legyen

$$\alpha_i = \begin{cases} 1, & \text{ha } x \succ_i y, \\ 0, & \text{ha } x \sim_i y, \\ -1, & \text{ha } x \prec_i y. \end{cases}$$

Legyen $\beta_i, \beta_2, \ldots, \beta_I$ adott pozitív konstans, és

$$\alpha = \sum_{i=1}^{I} \beta_i \alpha_i \ .$$

A csoportos döntéshozatali leképezést a következő jelenti:

$$\begin{aligned} x \succ y &\iff \alpha > 0 \\ x \sim y &\iff \alpha = 0 \\ x \prec y &\iff \alpha < 0 \ . \end{aligned}$$

A ***többségi szavazati szabály*** ennek az a speciális esete, amikor $\beta_1 = \beta_2 = \cdots = \beta_I = 1$.

26.13. példa. Egy i_0 döntéshozót ***diktátornak*** hívunk, ha a csoportos döntéshozatalban az ő akarata érvényesül:

$$\begin{aligned} x \succ y &\iff x \succ_{i_0} y \\ x \sim y &\iff x \sim_{i_0} y \\ x \prec y &\iff x \prec_{i_0} y \ . \end{aligned}$$

Ezt a fajta csoportos preferenciát diktatúrának is hívjuk.

26.14. példa. A ***Borda-mérték*** esetén feltesszük, hogy α véges halmaz és az összes döntéshozó valamennyi $x \in X$ esetén egy $c_i(x)$ mértékkel fejezi ki preferenciáját. Például $c_i(x) = 1$, ha x a legjobb, $c_i(x) = 2$, ha x a második legkedvezőbb alternatíva az i-edik döntéshozó számára, és így tovább, $c_i(x) = I$, ha x a legrosszabbnak feltüntetett. Ekkor

$$\begin{aligned} x \succ y &\iff \sum_{i=1}^{I} c_i(x) > \sum_{i=1}^{I} c_i(y) \\ x \sim y &\iff \sum_{i=1}^{I} c_i(x) = \sum_{i=1}^{I} c_i(y) \\ x \prec y &\iff \sum_{i=1}^{I} c_i(x) < \sum_{i=1}^{I} c_i(y) \ . \end{aligned}$$

Egy csoportos döntéshozatali leképezést ***Paretonak*** vagy ***Pareto-leképezésnek*** hívunk, ha valamilyen $x, y \in X$ és $x \succ_i y$ $(i = 1, 2, \ldots, I)$

esetén szükségképpen $x \succ y$. Azaz, ha az összes döntéshozó x-et preferálja y-nal szemben, akkor ez ugyanígy kell legyen a csoport közös preferenciájában is. Egy csoportos döntéshozatali leképezésről azt mondjuk, hogy eleget tesz a ***páronkénti függetlenségi*** feltételnek, ha tetszőleges két $(\succeq_1, \ldots, \succeq_I)$ és $(\succeq'_1, \ldots, \succeq'_I)$ preferencia-struktúra kielégíti a következőket. Legyen $x, y \in X$ olyan, hogy tetszőleges i mellett $x \succeq_i y$ akkor és csak akkor teljesül, ha $x \succeq'_i y$, és $y \succeq_i x$ akkor és csak akkor teljesül, ha $y \succeq'_i x$. Ekkor a csoport közös preferenciájában is $x \succeq y$ akkor és csak akkor, ha $x \succeq' y$, és $y \succeq x$ akkor és csak akkor, ha $y \succeq' x$.

26.15. példa. Könnyen látható, hogy a Borda-mérték Pareto, azonban nem elégíti ki a páronkénti függetlenségi feltételt. Az első állítás nyilvánvaló, a másodikat pedig egy egyszerű példával illusztrálhatjuk. Legyen $I = 2$, $\alpha = \{x, y, z\}$. tegyük fel, hogy

$$x \succ_1 z \succ_1 y$$
$$y \succ_2 x \succ_2 z$$

és

$$x \succ'_1 y \succ'_1 z$$
$$y \succ'_2 z \succ'_2 x \quad .$$

Ekkor $c(x) = 1 + 2 = 3, c(y) = 3 + 1 = 4$, így $y \succ x$. Azonban $c'(x) = 1 + 3 = 4, c'(y) = 2+1 = 3$, következésképpen $x \succ y$. Mint látjuk az egyes döntéshozók x és y közötti preferencia rendezése ugyanaz a két esetben, de a csoport közös preferenciája különböző.

Jelölje $\mathbb{R}_I$ az I-elemű teljes és tranzitív $(\preceq_1, \ldots, \preceq_I)$ félig rendezések halmazát egy legalább három elemű X halmazon, és legyen $\preceq$ a csoport együttes preferenciája, amely Pareto és eleget tesz a páronkénti függetlenségi feltételnek. Ekkor $\preceq$ szükségképpen diktatórikus. Ez az Arrow-tól származó eredmény mutatja, hogy nincs olyan csoportos döntéshozatali leképezés, amely eleget tenne a fenti két természetes és alapvető követelménynek.

26.16. példa. A ***páronkénti összehasonlítás*** módszere a következő. Legyen $x, y \in X$ tetszőleges, és jelöljük $P(x, y)$-nal azon döntéshozók számát, amelyekre $x \succ_i y$. A csoport együttes preferenciája ezek után a következő:

$$\begin{aligned} x \succ y &\iff P(x,y) > P(y,x) \\ x \sim y &\iff P(x,y) = P(y,x) \\ x \prec y &\iff P(x,y) < P(y,x)\ , \end{aligned}$$

azaz $x \succ y$ akkor és csak akkor, ha több döntéshozó tartja x-et jobbnak, mint az y alternatívát a kettőjük összehasonlításában. Tegyük fel ismét, hogy X három elemből

Döntéshozók	Alternatívák				Súlyok
	1	2	...	N	
1	a_{11}	a_{12}	...	a_{1N}	α_1
2	a_{21}	a_{22}	...	a_{2N}	α_2
⋮	⋮	⋮		⋮	⋮
I	a_{I1}	a_{I2}	...	a_{IN}	α_I

26.11. ábra. Csoportos döntéshozatali táblázat.

áll, $x = \{x, y, z\}$ és az egyéni preferenciák $I = 3$ esetén

$$x \succ_1 y \succ_1 z$$
$$z \succ_2 x \succ_2 y$$
$$y \succ_3 z \succ_3 x \quad .$$

Az együttes preferenciában tehát $x \succ y$, mert $P(x, y) = 2$ és $P(y, x) = 1$. Hasonlóan $y \succ z$, mert $P(y, z) = 2$ és $P(z, y) = 1$, valamint $z \succ x$, hiszen $P(z, x) = 2$ és $P(x, z) = 1$. Tehát $x \succ y \succ z \succ x$, ami ellentmond a tranzitivitási követelménynek.

Az eddigiekben tárgyalt módszerek nem vették figyelembe azt a fontos körülményt, hogy a döntéshozók nem feltétlenül azonos pozíciókban vannak, azaz különböző fontosságúak is lehetnek. Ezt a fontosságot súlyokkal tudjuk például jellemezni. Ebben az általánosabb esetben a csoportos döntéshozatali módszereket kellőképpen módosítanunk kell. Tegyük fel, hogy X véges halmaz, jelölje N az alternatívák számát. Az összes döntéshozó preferenciáját az 1-től N-ig terjedő számokkal jelöljük, ahol 1 a legkedvezőbb, N pedig a legkedvezőtlenebb alternatívához van rendelve. Elképzelhető, hogy két alternatíva egyformán fontos, akkor törtszámot is használhatunk. Például, ha 2. és 3. fontossági sorrendjét nem tudjuk megkülönböztetni, akkor 2,5-et rendelünk mindkét alternatívához. Általában a meg nem különböztethető sorszámok átlagértékét rendeljük mindegyikükhöz. Ily módon a csoportos döntéshozatal problémája egy táblázatban adható meg, amely sorai a döntéshozóknak, oszlopai pedig a döntési alternatíváknak felelnek meg. A táblázat minden sora az $1, 2, \ldots, N$ számok valamilyen permutációja, esetleg néhány elemet átlagértékekkel helyettesítünk azonosan preferált esetekben. A 26.11. ábra mutatja az így adódó döntési táblázatot, ahol az utolsó oszlopban feltüntettük a döntéshozók súlyait is.

A ***többségi szavazati szabályt*** ebben az általánosabb esetben a következőképpen fogalmazhatjuk meg. Az összes j alternatívához határozzuk meg először azoknak a döntéshozóknak az együttes súlyát, akiknek a j-edik alternatíva a legjobb lehetőség, majd válasszuk ki azt az alternatívát a közösen elfogadott legjobbnak, amelyik esetén ez az összeg a legnagyobb. Ha célunk

az összes alternatíva rangsorolása, és nemcsak a legjobb kiválasztása, akkor ezen összeg szerinti csökkenő sorrendet kell tekintenünk az alternatívák rangsorolásához, ahol a legnagyobb összeg választja ki a legjobbat, és a legkisebb összeg adja a legkedvezőtlenebb alternatívát. Matematikailag legyen

$$f(a_{ij}) = \begin{cases} 1, & \text{ha } a_{ij} = 1, \\ 0 & \text{különben} \end{cases} \tag{26.38}$$

és

$$A_j = \sum_{i=1}^{I} f(a_{ij})\alpha_i \tag{26.39}$$

$j = 1, 2, \ldots, I$ esetén. A j_0-adik alternatíva a csoport által tekintett legjobb, ha

$$A_{j_0} = \max_j \{A_j\} \ . \tag{26.40}$$

A formális algoritmus a következő:

TÖBBSÉGI-SZAVAZATI-SZABÁLY(A)

```
1  A_1 = 0, A_2 = 0, ..., A_N = 0, max ← 0
2  for i = 1 to N
3      for j = 1 to I
4          if a_ji == 1
5              A_i ← A_i + α_j
6      if A_i > max
7          max ← A_i
8          ind ← i
9  return ind
```

A Borda-mérték alkalmazásakor legyen

$$B_j = \sum_{i=1}^{I} a_{ij}\alpha_i \ , \tag{26.41}$$

és a j_0 alternatíva a csoportos döntés eredménye, ha

$$B_{j_0} = \min_j \{B_j\} \ . \tag{26.42}$$

A Borda-mérték a következő algoritmussal írható le:

BORDA-MÉRTÉK-MÓDSZER(A, α)

```
1  B_1 = 0, B_2 = 0, ..., B_N = 0, max ← 0
2  for j = 1 to N
3       for i = 1 to I
4            B_j = B_j + a_ij α_i
5       if B_j > max
6          max ← B_j
7          ind = j
8  return ind
```

A ***páronkénti összehasonlítás*** módszerének alkalmazásakor legyen tetszőleges $j, j' \in X$ esetén

$$P(j, j') = \sum_{\{i|a_{ij} < a_{ij'}\}} \alpha_i \tag{26.43}$$

mely megadja mindazoknak a döntéshozóknak az együttes súlyát, akik a j-edik alternatívát tartják jobbnak, mint a j'-dik alternatívát. Az együttes döntésben

$$j \succ j' \Longleftrightarrow P(j, j') > P(j', j) \; .$$

Számos esetben az így nyert együttes félig rendezés nem ad egyértelműen legjobb alternatívát. Ilyen esetekben

$$S^* = \{j | j \in X \;\text{ és nincs olyan }\; j' \in X, \text{amelyre }\; j' \succ j\}$$

nem-dominált alternatíva-halmazon további analízis (például egy másik módszer alkalmazása) szükséges.

Az algoritmussal egy $\{0, 1\}$ elemű mátrixot konstruálunk, ahol $a_{jl} = 1$ akkor és csak akkor, ha a j alternatíva összeségében jobb, mint az l alternatíva. Döntetlen esetben $a_{jl} = \frac{1}{2}$ lesz.

Bizottsági Tagok	Alternatívák 1	2	3	4	Súlyok
1	1	3	2	4	0.3
2	2	1	4	3	0.2
3	1	3	2	4	0.2
4	2	3	1	4	0.1
5	3	1	4	2	0.1
6	1	4	2	3	0.1

26.12. ábra. A 26.17 példa adatbázisa.

PÁRONKÉNTI-ÖSSZEHASONLÍTÁS(A)

```
1  for j = 1 to N − 1
2      for l = j to N
3          z = 0
4          for i = 1 to I
5              if a_ij > a_il
6                  z = z + 1
7                  if z > N/2
8                  a_jl = 1
9                  if z = N/2
10                     a_jl ← 1/2
11                 if z < N/2
12                     a_jl ← 0
13                 a_lj = a_jl
14 return A
```

26.17. példa. Egy vegyileg szennyezett terület tisztítására négy javaslat érkezett a környezetvédelmi hatósághoz. Egy hat emberből álló bizottságnak kell a legjobb javaslatot kiválasztani és ezután a hatóság szerződést köthet annak konkrét megvalósítására. A 26.12. táblázat tartalmazza az egyes bizottsági tagok relatív súlyát és az egyéni preferenciákat.

A ***többségi szavazati szabály*** alkalmazásakor

$$\begin{aligned} A_1 &= 0.3 + 0.2 + 0.1 = 0.6 \\ A_2 &= 0.2 + 0.1 = 0.3 \\ A_3 &= 0.1 \\ A_4 &= 0\,, \end{aligned}$$

így az első alternatíva adódik a legjobbnak.

A ***Borda-mérték*** alkalmazásakor

$$\begin{aligned} B_1 &= 0.3+0.4+0.2+0.2+0.3+0.1 = 1.5 \\ B_2 &= 0.9+0.2+0.6+0.3+0.1+0.4 = 2.5 \\ B_3 &= 0.6+0.8+0.4+0.1+0.4+0.2 = 2.5 \\ B_4 &= 1.2+0.6+0.8+0.4+0.2+0.3 = 3.5\ . \end{aligned}$$

Az első alternatíva adódik ismét a legjobbnak, de ez a módszer egyformán jónak mutatja a második és harmadik alternatívát. Vegyük észre, hogy az előző módszer esetében a második alternatíva jobbnak adódott, mint a harmadik.

A ***páronkénti összehasonlítás*** módszere esetében

$$\begin{aligned} P(1,2) &= 0.3+0.2+0.1+0.1 = 0.7 \\ P(2,1) &= 0.2+0.1 = 0.3 \\ P(1,3) &= 0.3+0.2+0.2+0.1+0.1 = 0.9 \\ P(3,1) &= 0.1 \\ P(1,4) &= 0.3+0.2+0.2+0.1+0.1 = 0.9 \\ P(4,1) &= 0.1 \\ P(2,3) &= 0.2+0.1+0.1 = 0.4 \\ P(3,2) &= 0.3+0.2+0.1 = 0.6 \\ P(2,4) &= 0.3+0.2+0.2+0.1+0.1 = 0.9 \\ P(4,2) &= 0.1 \\ P(3,4) &= 0.3+0.2+0.1+0.1 = 0.7 \\ P(4,3) &= 0.2+0.1 = 0.3\ . \end{aligned}$$

Tehát $1 \succ 2, 1 \succ 3, 1 \succ 4, 3 \succ 2, 2 \succ 4$ és $3 \succ 4$. Ezeket a preferenciákat a 26.13. ábrán ábrázoljuk. Az első alternatívát jobbnak találtuk, mint az összes többit, ezért ez a nyilvánvaló választás.

A fenti példában mindhárom módszer ugyanazt az eredményt szolgáltatta. Számos gyakorlati esetben azonban különböző eredmény is adódhat, ilyenkor a döntéshozóknak kell kiválasztaniuk valamilyen egyéb kritériumok alapján a közösen elfogadott megoldást.

Gyakorlatok

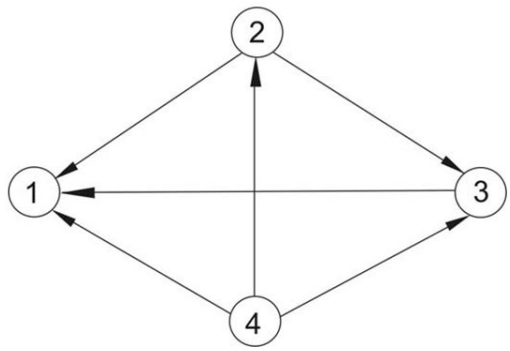
26.4-1. Tekintsük a következő csoportos döntéshozatali táblázatot:

Alkalmazzuk a többségi szavazati szabályt.

26.4-2. Alkalmazzuk a Borda-mértéket az előző gyakorlatra.

26.4-3. Alkalmazzuk a páronkénti összehasonlítás módszerét a 26.4-1. gyakorlatra.

26.4-4. Tekintsük most a következő csoportos döntéshozatali táblázatot:



26.13. ábra. A 26.17 példa referencia gráfja.

Döntéshozók	Alternatívák					Súlyok
	1	2	3	4	5	
1	1	3	2	5	4	3
2	1	4	5	2	3	2
3	5	4	1	3	2	2
4	4	3	2	1	5	1

26.14. ábra.

Döntéshozók	Alternatívák			Súlyok
	1	2	3	
1	1	2	3	1
2	3	2	1	1
3	2	1	3	1
4	1	3	2	1

26.15. ábra. Csoportos döntéshozatali táblázat

Ismételjük meg a 26.4-1. gyakorlatot erre a problémára.

26.4-5. Alkalmazzuk a Borda-mértéket az előző gyakorlatra.

26.4-6. Alkalmazzuk a páronkénti összehasonlítás módszerét a 26.4-4. gyakorlatra.

26.5. Pareto-játékok alkalmazása

Jelölje ismét I a döntéshozók számát és tegyük most fel, hogy a döntéshozók külön-külön több célfüggvénnyel rendelkeznek. Az ilyen típusú problémák kezelésére többféle lehetőség kínálkozik.

(A) Többcélú programozás alkalmazásakor jelölje α_i az i-edik döntéshozó súlyát, és legyenek $\beta_{i1}, \beta_{i2}, \ldots, \beta_{ic(i)}$ ugyanezen döntéshozó célfüggvényének a súlyai. Itt $c(i)$ jelöli az i-edik döntéshozó célfüggvényeinek a számát. Így egy $\sum_{i=1}^{I} c(i)$ célfüggvénnyel rendelkező optimalizációs problémát kaphatunk, ahol az összes döntéshozó valamennyi célfüggvénye jelenti a feladat célfüggvényeit, és az egyes célfüggvények súlyai az $\alpha_i\beta_{ij}$ sorozatok. Az így adódó probléma megoldására az 26.1. alfejezet bármely módszerét választhatjuk.

(B) Egy másik módszercsaládot nyerhetünk a következőképpen. Mindegyik döntéshozó esetére határozzunk meg egy hasznossági függvényt (a 26.1.1. szakaszban leírtak szerint), amely egyetlen függvénybe sűríti a döntéshozó preferenciáját. Ezzel a módszerrel elérjük, hogy minden döntéshozó egyetlen (új) célfüggvénnyel rendelkezik csak, így az előző alfejezetek bármelyik megoldási koncepcióját és módszerét közvetlenül alkalmazni tudjuk.

(C) Egy harmadik megoldási módszert kaphatunk oly módon, hogy az egyes döntéshozók esetén hasznossági függvények konstrukciója helyett csak a döntéshozók alternatív halmazon definiált félig rendezését határozzuk meg valamilyen módszerrel. Ezután pedig a 26.4. alfejezet bármelyik módszerét közvetlenül alkalmazni tudjuk.

26.18. példa. Módosítsuk az előző példát a következőképpen. tegyük fel ismét, hogy négy alternatívából választunk, azonban tegyük fel most, hogy háromtagú a bizottság és mindegyik bizottsági tag két célfüggvénnyel rendelkezik. Az első célfüggvényt a javasolt megoldás technikai színvonala jelenti egy szubjektív skálán. A második célfüggvény pedig a pontos megvalósítás valószínűségét jelenti, ez utóbbit szubjektív alapon külön-külön becslik meg a döntéshozók a javaslattevő cég korábbi megbízásai alapján. Az adatokat a 26.16. táblázat mutatja, ahol feltesszük, hogy az első célfüggvény egy 0-tól 100-ig terjedő szubjektív skálán becsült, így a normalizált célfüggvényértékeket 100-al való osztással kapjuk meg. A súlyozásos módszert alkalmazva külön-külön az egyes döntéshozók esetére a következő összevont célfüggvényértékek adódnak:

1. Döntéshozó

$$\begin{array}{lrcl}
\text{Első alternatíva:} & 0.9(0.5) + 0.9(0.5) & = & 0.9 \\
\text{Második alternatíva:} & 0.75(0.5) + 0.8(0.5) & = & 0.775 \\
\text{Harmadik alternatíva:} & 0.8(0.5) + 0.7(0.5) & = & 0.75 \\
\text{Negyedik alternatíva:} & 0.85(0.5) + 0.8(0.5) & = & 0.825
\end{array}$$

Döntéshozó	Célfüggvény	Alternatívák				Célfüggvény súlya	Döntéshozó súlya
		1	2	3	4		
1	1	90	75	80	85	0.5	0.4
	2	0.9	0.8	0.7	0.8	0.5	
2	1	85	80	70	90	0.6	0.3
	2	0.8	0.9	0.8	0.85	0.4	
3	1	80	90	75	70	0.7	0.3
	2	0.85	0.8	0.9	0.8	0.3	

26.16. ábra. A 26.18 példa adatbázisa.

2. Döntéshozó

$$\begin{aligned}
\text{Első alternatíva:} \quad & 0.85(0.6)+0.8(0.4) &&= 0.83\\
\text{Második alternatíva:} \quad & 0.8(0.6)+0.9(0.4) &&= 0.84\\
\text{Harmadik alternatíva:} \quad & 0.7(0.6)+0.8(0.4) &&= 0.74\\
\text{Negyedik alternatíva:} \quad & 0.9(0.6)+0.85(0.4) &&= 0.88
\end{aligned}$$

3. Döntéshozó

$$\begin{aligned}
\text{Első alternatíva:} \quad & 0.8(0.7)+0.85(0.3) &&= 0.815\\
\text{Második alternatíva:} \quad & 0.9(0.7)+0.8(0.3) &&= 0.87\\
\text{Harmadik alternatíva:} \quad & 0.75(0.7)+0.9(0.3) &&= 0.795\\
\text{Negyedik alternatíva:} \quad & 0.7(0.7)+0.8(0.3) &&= 0.73
\end{aligned}$$

Az egyedi preferenciák tehát a következők:

$$1 \succ_1 4 \succ_1 2 \succ_1 3, 4 \succ_2 2 \succ_2 1 \succ_2 3, \quad \text{és} \quad 2 \succ_3 1 \succ_3 3 \succ_3 4\,.$$

Például a Borda-mértéket alkalmazva

$$\begin{aligned}
B_1 &= 1(0.4)+3(0.3)+2(0.3) = 1.9\\
B_2 &= 3(0.4)+2(0.3)+1(0.3) = 2.1\\
B_3 &= 4(0.4)+4(0.3)+3(0.3) = 3.7\\
B_4 &= 2(0.4)+1(0.3)+4(0.3) = 2.3
\end{aligned}$$

adódik, így a négy alternatíva csoport-rendezése

$$1 \succ 2 \succ 4 \succ 3\,.$$

Gyakorlatok

26.5-1. Tekintsük a következő táblázatot:

Döntéshozó	Célfüggvény	Alternatívák			Célfüggvény súlya	Döntéshozó súlya
		1	2	3		
1	1	0.6	0.8	0.7	0.6	0.5
	2	0.9	0.7	0.6	0.4	
2	1	0.5	0.3	0.4	0.5	0.25
	2	0.6	0.8	0.7	0.5	
3	1	0.4	0.5	0.6	0.4	0.25
	2	0.7	0.6	0.6	0.4	
	3	0.5	0.8	0.6	0.2	

26.17. ábra.

Tegyük fel, hogy a célfüggvények már normalizáltak. Alkalmazzuk az (A) módszert a gyakorlat megoldására.

26.5-2. Az előző gyakorlat esetére alkalmazzuk a (B) módszert, ahol az egyes döntéshozók hasznossági függvényeit a súlyozásos módszerrel kapjuk meg, és a csoportos döntéshozatalt pedig a Borda-mérték adja meg.

26.5-3. A 26.5-2. gyakorlatban alkalmazzuk a páronkénti összehasonlítás módszerét a Borda-mérték helyett.

26.6. Axiomatikus módszerek

Az egyszerűség érdekében tegyük fel, hogy $I = 2$, azaz két döntéshozó közötti konfliktust kívánunk feloldani. Feltesszük, hogy a H következménytér konvex, korlátos, és zárt $\mathbb{R}^2$-ben, és adott egy olyan $\mathbf{f}_* = (f_{1*}, f_{2*})$ pont, amely a döntéshozók célfüggvényértékeit adja meg abban az estben, ha nem tudnak megállapodni. Feltesszük, hogy létezik olyan $\mathbf{f} \in H$, amelyre $\mathbf{f} > \mathbf{f}_*$. A $(H, \mathbf{f}_*)$ pár matematikailag jellemzi a konfliktust. A megoldás nyilvánvalóan mind H-tól és mind $\mathbf{f}_*$-tól kell függjön, így ezeknek valamilyen függvénye: $\phi(H, \mathbf{f}_*)$.

A különféle megoldási koncepciók esetén megköveteljük, hogy a megoldásfüggvény eleget tegyen bizonyos követelményeknek, amelyeket axiómaként kezelünk. Ezek az axiómák a megoldás korrektségét követelik meg, különböző axiómák más-más módon jellemzik ezt a korrektséget.

A ***klasszikus Nash-féle*** megoldás esetén a következőket tesszük fel:

(i) $\phi(H, \mathbf{f}_*) \in H$ (lehetségesség)

(ii) $\phi(H, \mathbf{f}_*) \geq \mathbf{f}_*$ (racionalitás)

(iii) $\phi(H, \mathbf{f}_*)$ Pareto-megoldás H-ban (Pareto-optimalitás)

(iv) Ha $H_1 \subseteq H$ és $\phi(H, \mathbf{f}_*) \in H_1$, akkor szükségképpen $\phi(H_1, \mathbf{f}_*) = \phi(H, \mathbf{f}_*)$ (kedvezőtlen alternatíváktól való függetlenség)

(v) Legyen $T : \mathbb{R}^2 \mapsto \mathbb{R}^2$ olyan lineáris transzformáció, amelyre $T(f_1, f_2) = (\alpha_1 f_1 + \beta_1, \alpha_2 f_2 + \beta_2)$ pozitív α_1 és α_2 esetén. Ekkor $\phi(T(H), T(\mathbf{f}_*)) = T(\phi(H, \mathbf{f}_*))$ (növekvő lineáris transzformációtól való függetlenség)

(vi) Ha H és $\mathbf{f}_*$ szimmetrikusak, azaz $f_{1*} = f_{2*}$ és $(f_1, f_2) \in H \Longleftrightarrow (f_2, f_1) \in H$, akkor $\phi(H, \mathbf{f}_*)$ komponensei is azonosak legyenek (szimmetria).

Az (i) feltétel a megoldás lehetségességét követeli meg. A (ii) feltétel azt írja elő, hogy egy racionális döntéshozó sem egyezik bele olyan megoldásba amely rosszabb, mint amit megegyezés nélkül amúgy is elérne. A (iii) feltétel alapján nincs jobb, mint a megegyezéssel elért megoldás. A (iv) követelmény szerint ha a megegyezés után bizonyos alternatívák elvesztik lehetségességüket, de a megoldás lehetséges marad, akkor a leszűkített következménytér mellett is ugyanaz a megoldás marad. Ha bármely célfüggvény dimenzióját megváltoztatjuk, a megoldás nem változhat. Ezt a követelményt fogalmazza meg (v), és az utolsó feltétel azt jelenti, hogy ha két döntéshozó teljesen azonos helyzetben van a konfliktus megfogalmazásában, akkor a megoldás esetén is azonosan kell kezeljük őket. A következő, Nash-tól származó eredmény alapvető fontosságú.

26.6. tétel. *Az (i)-(vi) feltételeket pontosan egy megoldás függvény elégíti ki, és $\phi(H, \mathbf{f}_*)$ az*

$$\begin{array}{rcll} (f_1 - f_{1*})(f_2 - f_{2*}) & \longrightarrow & max & ((f_1, f_2) \in H) \\ f_1 & \geq & f_{1*} & \\ f_2 & \geq & f_{2*} & \end{array} \tag{26.44}$$

optimumfeladat egyértelmű megoldásaként kapható meg.

26.19. példa. Tekintsük ismét a **??**. ábrán korábban bemutatott következményteret és tegyük fel, hogy $(f_{1*}, f_{2*}) = (0, -1)$, azaz a legrosszabb értékeket tartalmazza komponenseiben. Ekkor a (26.44) feladat a következő alakú:

$$\begin{array}{rcl} f_1(f_2 + 1) & \longrightarrow & max \\ f_2 & \leq & f_1 \\ f_2 & \leq & 3 - 2f_1 \\ f_2 & \geq & -\frac{1}{2} f_1 \, . \end{array}$$

Könnyen látható, hogy az optimális megoldás $f_1 = f_2 = 1$.

Vegyük észre, hogy a (26.44) probléma egy távolságfüggő módszer, amikor a geometriai távolságot maximalizáljuk az (f_{1*}, f_{2*}) ponttól. Az algoritmus a

(26.44) optimumfeladat megoldását jelenti.

A (vi) feltétel megköveteli, hogy a két döntéshozó azonosan legyen kezelve. Számos gyakorlati esetben azonban ez nem valós követelmény, ha valamelyik döntéshozó erősebb helyzetben van, mint a másik.

26.7. tétel. *Az (i)-(v) követelményeknek végtelen sok megoldás függvény tesz eleget, de valamennyi megoldás függvényben található olyan* $0 \leq \alpha \leq 1$, *hogy a megoldás az*

$$\begin{array}{rcll} (f_1 - f_{1*})^{\alpha}(f_2 - f_{2*})^{1-\alpha} & \longrightarrow & max & ((f_1, f_2) \in H) \\ f_1 & \geq & f_{1*} & \\ f_2 & \geq & f_{2*} & \end{array} \tag{26.45}$$

optimumfeladat egyértelmű megoldásaként kapható meg.

Vegyük észre, hogy az $\alpha = \frac{1}{2}$ esetben a (26.45) probléma a (26.44) feladatra redukálódik. Az algoritmus a (26.45) optimumfeladat megoldását jelenti.

Számos szerző kritizálta Nash eredeti axiómáit, és az axiómarendszer bizonyos módosításai mellett újabb és újabb megoldási koncepciókat és módszereket vezettek be. A konkrét axiómák ismertetése nélkül magukat a módszereket ismertetjük az irodalomban legfontosabbnak tartott esetekben.

A ***Kalai-Smorodinsky-megoldás*** esetén először az ideális pontot határozzuk meg, amely koordinátái:

$$f_i^* = \max\{f_i | (f_1, f_2) \in H, (f_1, f_2) \geq \mathbf{f}_*\} \ ,$$

majd az $\mathbf{f}_*$-ból az ideális pontba vezető félegyenes H-val való utolsó közös pontját fogadjuk el, mint a megoldást. A 26.18. ábra mutatja a módszert. Vegyük észre, hogy a módszer egy irányfüggő módszer, ahol a félegyenes mutatja a növekedési irányt és $\mathbf{f}_*$ a választott kiindulópont.

Az algoritmus a következő optimumfeladat megoldását igényli:

$$t \longrightarrow max$$

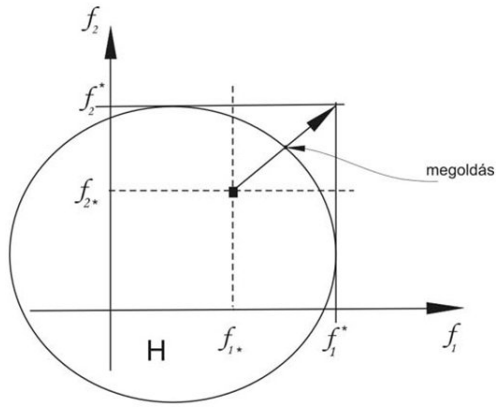
feltéve, hogy

$$\mathbf{f}_* + t(\mathbf{f}^* - \mathbf{f}_*) \in H \ .$$

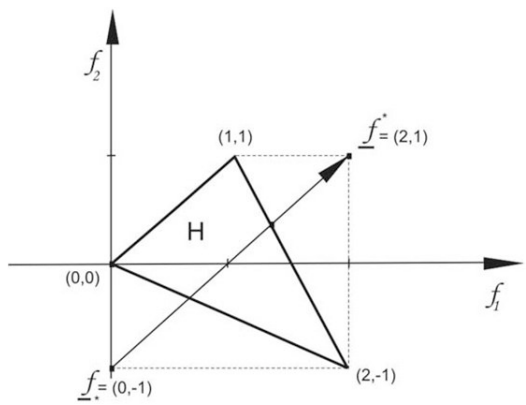
26.20. példa. Az előző példa esetén $\mathbf{f}_* = (0, -1)$ és $\mathbf{f}^* = (2, 1)$. A 26.19. ábrán láthatjuk, hogy az $\mathbf{f}_*$-ból $\mathbf{f}^*$-ba vezető félegyenes utolsó H-beli pontja a félegyenes és az $(1, 1)$ és $(2, -1)$ pontokat összekötő szakasz metszéspontja.

A félegyenes egyenlete

$$f_2 = f_1 - 1$$



26.18. ábra. Kalai-Smorodinsky-megoldás.



26.19. ábra. A 26.20 példa megoldása.

míg az összekötő szakasz egyenlete

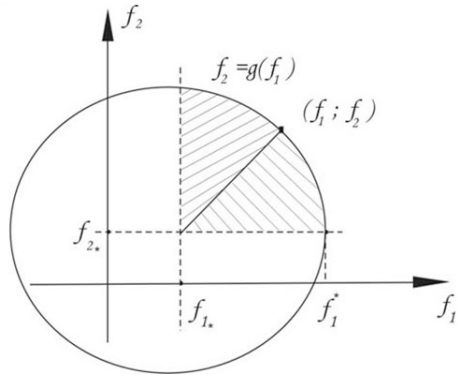
$$f_2 = -2f_1 + 3\ ,$$

így a metszéspont: $f_1 = \frac{4}{3}, f_2 = \frac{1}{3}$.

Az ***egyenlő veszteség módszerénél*** feltesszük, hogy az ideális pontból kiindulva a két döntéshozó azonos sebességgel csökkenti a célfüggvényértékeit egészen addig, míg lehetséges megoldás nem adódik. Ez a koncepció ekvivalens a

$$t \longrightarrow min \qquad ((f_1^* - t, f_2^* - t) \in H) \tag{26.46}$$

optimumfeladat megoldásával. Jelölje t^* a minimális t értéket, ekkor az $(f_1^* -$



26.20. ábra. A monoton terület módszere.

$t^*, f_2^* - t^*$) pont jelenti a konfliktus megoldását. Az algoritmust a (26.46) optimumfeladat megoldása jelenti.

26.21. példa. Az előző példa esetében $\mathbf{f}^* = (2, 1)$, így ebből a pontból kiindulva a 45°-os egyenes mentén haladva az első lehetséges megoldás ismét az $f_1 = \frac{4}{3}, f_2 = \frac{1}{3}$ pont.

A ***monoton terület*** módszere esetében az (f_1, f_2) megoldást a következőképpen kapjuk. Az (f_{1*}, f_{2*}) és (f_1, f_2) pontokat összekötő lineáris szakasz a H halmazt két részre bontja, ha (f_1, f_2) egy Pareto-optimális megoldás. Ennek a koncepciónak az alkalmazása esetén megköveteljük, hogy a két terület azonos legyen. A 26.20. ábra mutatja a koncepciót. A két terület a következőképpen adódik:

$$\int_{f_{1*}}^{f_1} (g(t) - f_{2*})dt - \frac{1}{2}(f_1 - f_{1*})(g(f_1) - f_{2*})$$

és

$$\frac{1}{2}(f_1 - f_{1*})(g(f_1) - f_{2*}) + \int_{f_1}^{f_1^*} (g(t) - f_2^*)dt$$

ahol feltesszük, hogy $f_2 = g(f_1)$ írja le a Pareto-optimális megoldások görbéjét. Ily módon egy egyszerű egyenletet kapunk az ismeretlen f_1 érték meghatározására.

Az algoritmust a következő nemlineáris, egyváltozós egyenlet megoldása jelenti:

$$\int_{f_{1*}}^{f_1} (g(t) - f_{2*})dt - \int_{f_1}^{f_{1*}} (g(t) - f_2^*)dt - (f_1 - f_{1*})(g(f_1) - f_{2*}) = 0 \, .$$

Bármely közismert módszer (intervallum-felezés, húrmódszer, szelőmódszer, Newton módszer stb.) alkalmazható a feladat megoldására.

Gyakorlatok

26.6-1. Tegyük fel, hogy $H = \{(f_1, f_2) | f_1, f_2 \geq 0, f_1 + 2f_2 \leq 4\}$. Legyen $f_{1*} = f_{2*} = 0$. Alkalmazzuk a (26.44) optimumfeladatot.

26.6-2. Az előző gyakorlatban tegyük fel, hogy a két döntéshozó nem azonosan fontos. $\alpha = \frac{1}{3}, 1 - \alpha = \frac{2}{3}$. Oldjuk meg a (26.45) optimumfeladatot.

26.6-3. Alkalmazzuk a Kalai-Smorodinsky-megoldást a 26.6-1. gyakorlatra.

26.6-4. Alkalmazzuk az egyenlő veszteség módszerét a 26.6-1. gyakorlatra.

26.6-5. Alkalmazzuk a monoton terület módszerét a 26.6-1. gyakorlatra.

Feladatok

26-1 Távolságfüggő módszerek

Bizonyítsuk be, hogy a távolságfüggő módszerek a ϱ_1 távolsággal mindig Pareto-optimális megoldást szolgáltatnak. Igaz ugyanez, ha a ϱ_∞ távolságot alkalmazzuk?

26-2 irányfüggő módszerek

Mutassuk meg egy egyszerű példával, hogy irányfüggő módszerek olyan eredményt is adhatnak, melyek nem Pareto-optimálisak.

26-3 Több egyensúly

A 26.4. tétel feltételein kívül tegyük még fel, hogy az összes f_i függvény szigorúan konkáv x_i-ben. Adjunk meg olyan példát, amelyben több egyensúlypont is létezik.

26-4 Shapley-egyensúly

Bizonyítsuk be, hogy a Shapley-értékek imputációt adnak és kielégítik a (26.35)–(26.36) feltételeket.

26-5 Csoportos döntéshozatali táblázat

Adjunk meg olyan csoportos döntéshozatali táblázatot, ahol a páronkénti összehasonlítás módszere nem elégíti ki a tranzitivitás követelményét. Azaz van olyan i, j, k alternatíva, amelyekre $i \succ j$, $j \succ k$, de $k \succ i$.

26-6 Borda-mérték alkalmazása

Konstruáljunk olyan feladatot, ahol a Borda-mérték alkalmazása azonosan jónak minősíti az összes alternatívát.

26-7 Kalai–Smorodinsky-megoldá

Mutassuk meg, hogyha a Kalai-Smorodinsky-megoldást alkalmazzuk nem konvex H esetére, akkor a megoldás nem feltétlenül Pareto-optimális.

26-8 Egyenlő veszteség módszer
Mutassuk meg, hogy nem konvex H esetén sem az egyenlő veszteség, sem a monoton terület módszere nem garantál Pareto-optimális megoldást.

26-9 Pareto-optimalitás
Bizonyítsuk be, hogy a (26.9) feladat megoldása pozitív $\alpha_1, \alpha_2, \ldots, \alpha_I$ értékek mellett mindig Pareto-optimális megoldást ad.

Megjegyzések a fejezethez

A többcélú programozás iránt érdeklődő Olvasó további részleteket és módszereket találhat a témával kapcsolatban a [281] könyvben. Az egyensúlypontok módszeréről valamint a kooperatív játékok megoldási koncepcióiról és módszereiről bővebben a [88] monográfiában olvashatnak. A [299] monográfia további módszereket és képleteket tartalmaz a csoportos döntéshozás módszertanából. A Nashtől származó 26.6. tétel részletesebben [211]-ben található meg. A tételben szereplő feltételek gyengítésével kapcsolatban [116]-ben olvashatnak. A Kalai-Smorodinsky-megoldás, az egyenlő veszteség-, valamint a monoton terület módszere részletesen rendre [144]-ben, [49]-ben és [4]-ben található meg. Megjegyezzük végül, hogy a [303] összefoglaló dolgozat tárgyalja ezeknek és újabb módszereknek az axiomatikus bevezetését és tulajdonságait.

A fejezetben szereplő eredmények részletesebben megtalálhatók Molnár Sándor és Szidarovszky Ferenc [207] jegyzetében, valamint Szidarovszky és Domoszlai angol nyelvű fejezetében [280].

27. Tömegkiszolgálás

A tömegkiszolgálás elmélete tömegesen előforduló igények és kiszolgálásuk problémájának matematikai modellezésével és megoldásával foglalkozik.

Tömegkiszolgálási rendszerek (TKR-ek) a gyakorlatban az élet szinte minden területén előfordulnak, rendkívül változatos megjelenési formájuk ellenére működésük sok esetben közös matematikai modellekkel írható le. A matematikai modellek kidolgozására és vizsgálatára már nagyon korán, a múlt század elején sor került. Az első eredmények Erlang dán matematikus nevéhez fűződnek, aki a koppenhágai telefontársaságnál dolgozott. Ezek a vizsgálatok a telefonhálózatok gyors fejlődésének következtében váltak fontossá. Innen ered az akkor kialakult és a mai napig használt terminológia is, amely sok tekintetben kötődik a telefonnal kapcsolatban megszokott fogalmakhoz (csatorna, hívás, foglaltság, hívás visszautasítása, sorhosszúság stb.).

A TKR-ekben közös az igények megjelenése, igények kiszolgálása (tágabb értelemben is), sorok képződése, várakozás (illetve elutasítás), igények távozása a rendszerből. Egy TKR-be többféle igény érkezhet, de keletkezhet új igény magában a rendszerben is. Az igények kiszolgálása a megfelelő kiszolgáló egységen történik. A kiszolgáló egység előtt lehetnek várakozási (sorbanállási) helyek, ahova a beérkezett, de a kiszolgáló egységnél még kiszolgálásra nem kerülő (vagy a kiszolgáló egységről kikerülő, s arra újból visszatérő) igények kerülnek. Ha egy TKR-ből anélkül távozhatnak igények, hogy teljes kiszolgálást nyertek volna, akkor a rendszert ***veszteségesnek*** nevezzük.

Általában elvonatkoztathatunk a vizsgált rendszerek, az igények és a kiszolgálás konkrét jellegétől. Ez annak köszönhető, hogy a vizsgálatok gyakran bizonyos időpontok halmazához (igények megjelenése a rendszerben, kiszolgálások kezdete, illetve befejezése stb.) kötődnek, ezért el lehet tekinteni az igények és kiszolgálások valós hátterétől. Az igények beérkezési időpontjai, a kiszolgálásukhoz szükséges idők stb. általában sztochasztikus jellegűek, ezért az ilyen rendszerek elemzése a valószínűségszámítás eszközeit és módszereit igényli.

Tömegkiszolgálási rendszerek matematikai leírásához a következőket kell megadni:

- A *beérkezési folyamat* leírja a rendszerbe érkező igényeket, de absztrakt igények is lehetnek, továbbá igények képződhetnek a TKR-en belül is. A rendszerbe lépő igények száma és eloszlása függhet a rendszer állapotától.

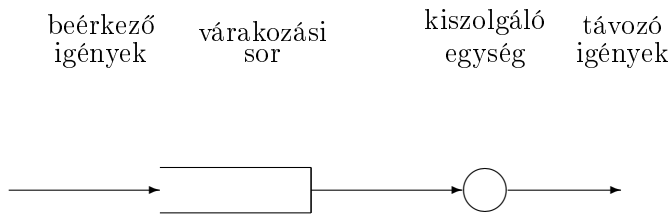
A beérkezési folyamatok általában az egymás után érkező igények közötti időintervallumokkal és az igényelt kiszolgálási időkkel írhatók le (utóbbi, például telekommunikációs rendszerek esetén, jelentheti az átviendő adatmennyiség nagyságát). A beérkezések között eltelt idők és a kiszolgálási idők nagysága általában véletlen mennyiség, de lehet determinisztikus is. Az egymás utáni igények között eltelt időktől (illetve a kiszolgálási időktől) egyfajta homogenitást – azonos eloszlást – szokás megkövetelni.

- A vizsgált rendszer *struktúrája* (*topológiája*) megadja azt, hogy az igények a TKR kiszolgálóegységei között hogyan vándorolhatnak, illetve távozhatnak a rendszerből. A vizsgált rendszer struktúrájához hozzátartozik az egyes kiszolgáló egységekhez tartozó várakozási helyek (puffer terület) nagysága is.
- *Kiszolgálási algoritmusok* (*kiszolgálási elvek*) határozzák meg az egyes igények kiszolgálási szabályait. A TKR-ek hatékonysága jelentős mértékben függ a kiszolgálási algoritmusoktól, ha a kiszolgáló eszközhöz való hozzáférés egymással konkuráló felhasználók mellett történik.
- A TKR meghatározandó *jellemzői, mutatói,* melyek összefüggnek a vizsgálat céljaival. Megemlítünk néhány jellemző példát. *Várakozási rendszereknél* az átlagos várakozási idő a kiszolgálás megkezdéséig a rendszerben eltöltött átlagos idő, átlagos sorhosszúság, a sorhosszúság eloszlása; *elutasításos rendszereknél* az elutasítás valószínűsége, az elutasítások átlagos száma; *hatékonysági mértékek*, mint például az átlagos kihasználtság, kiszolgált igények átlagos száma stb.

27.1. Tömegkiszolgálási rendszerek működésének leírása

Egy rendszer működésének, időben változó jellemzőinek leírása azt jelenti, hogy a beérkezési folyamatból és a rendszer működését meghatározó tulajdonságokból kiindulva megadjuk azt az algoritmust, amely az idő függvényében (vagy csak a megadott időpontokban) előállítja a vizsgált jellemzőket. Látni fogjuk, hogy ezeknek a – különböző jellemzők meghatározásánál különböző – számítási eljárásoknak a megadása már a legegyszerűbb rendszerek és egyszerű kiszolgálási algoritmusok esetén is bonyolult feladatot jelenthet. A beérkezési folyamat és ezzel együtt a rendszer sztochasztikus jellegének, statisztikai törvényszerűségeinek vizsgálatával itt külön nem foglalkozunk. A feladat jellegét néhány konkrét példával mutatjuk be.

A 27.1. ábra egy egyszerű egycsatornás, egy kiszolgáló egységből és korlátlan számú várakozási helyből álló kiszolgálási rendszert (a szakirodalom-



27.1. ábra. Egycsatornás kiszolgálási rendszer.

ban alkalmazott jelöléssel $G|G|1|\infty$ típusú rendszert) mutat be: egy fajta igény érkezik a rendszerbe; a kiszolgáló egység előtt (végtelen hosszúságú) sor képződhet; a kiszolgálást egy kiszolgáló egység végzi és az igény a kiszolgálás után távozik a rendszerből. (A tömegkiszolgálási rendszerek leírására általánosan használt a Kendall-jelölés által bevezetett jelölés, amely $A|B|m|n$ alakú. Ebben A a belépő folyamat jellegére utal (M, ha két belépés között eltelő idő exponenciális; G, ha általános eloszlású); B a kiszolgálási idő eloszlása (B lehetséges értékei ugyanazok lehetnek, mint A-é); m a kiszolgáló eszközök száma; n a rendszerben levő tartózkodási helyek száma (ha nem adjuk meg vagy a ∞ jelet használjuk, akkor erre nincs korlátozás).

Ha egy igény a rendszerbe érkezve üresen találja azt, akkor az igény kiszolgálása azonnal megkezdődik, különben az igény a várakozási sorba kerül. Ha egy igény kiszolgálása befejeződik a kiszolgáló egységen és a várakozási sor nem üres, akkor a sorban állók közül a korábban beérkezett igény kerül kiszolgálásra. Itt fontos megjegyezni, hogy a kiszolgálás módja és a várakozó igények közül való választás sokféle elv szerint történhet (lásd lentebb).

A vizsgálat kezdete utáni első igény a rendszerbe az X_0 időpontban érkezik be. Jelölje X_i $(i = 1, 2, \ldots)$ az egymás után a rendszerbe érkező i-edik és $(i+1)$-edik igény beérkezési időpontjai között eltelt időt, Y_i $(i = 1, 2, \ldots)$ pedig az i-edik igény kiszolgálásához szükséges időt. Ekkor, ha ismert a rendszer kiinduló állapota (például a rendszer üres az első igény beérkezéséig), akkor az X_0, (X_i, Y_i) $(i = 1, 2, \ldots)$ sorozat – a rendszer *beérkezési folyamata* – tetszőleges $t > 0$ időpontban meghatározza a rendszer állapotát és a feladat éppen az, hogy – a rendszer beérkezési folyamatából kiindulva – eljárást adjunk a rendszer különböző jellemzőinek meghatározására tetszőleges t időpontban. Az egyszerűség kedvéért a továbbiakban feltesszük, hogy X_i és Y_i pozitív mennyiségek.

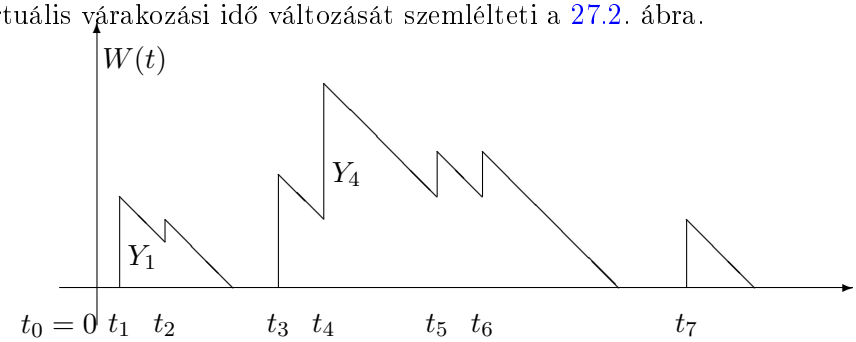
Jelölje $t_1, t_2, \ldots$ és $s_1, s_2, \ldots$ az egymás után a rendszerbe érkező igények beérkezési időpontjait, illetve azokat az időpontokat, amikor az igények kiszol-

gálása befejeződött a rendszerben (az igények távozási időpontjait). Világos, hogy $X_i = t_{i+1} - t_i$, $i = 1, 2, \ldots$, így a beérkezési időpontokra

$$t_{i+1} = X_0 + \cdots + X_i, \qquad i \geq 1 \ .$$

Megjegyezzük, hogy a távozási időpontokra, melyek a beérkezési időpontokon kívül az egyes kiszolgálási időktől és a rendszer összes tulajdonságától függhetnek, általában nem adható hasonlóan egyszerű formula. Az alábbiakban nézzük meg, hogy bizonyos, a rendszer működését időben leíró jellemzőket milyen algoritmus szerint lehet kiszámítani.

A $W(t)$ virtuális várakozási idő meghatározása. Jelölje $W(t)$ tetszőleges $0 \leq t < \infty$ időpontban a virtuális várakozási időt, vagyis azt az időt, amennyit egy igénynek a kiszolgálás megkezdéséig várakoznia kellene, ha a t időpontban érkezne a rendszerbe. Feltesszük, hogy a $t = 0$ időpontban a rendszer üresen kezdett dolgozni, vagyis $W(0) = 0$ és $W(t)$-t balról folytonosnak definiáljuk, ezáltal az ugrási pontokban egyértelműen meghatározzuk. A virtuális várakozási idő változását szemlélteti a 27.2. ábra.



27.2. ábra. Virtuális várakozási idő.

Látható, hogy a $W(t)$ virtuális várakozási időnek ugrása van a t_i időpontokban, melynek nagysága $W(t_i + 0) - W(t_i) = Y_i (i = 1, 2, \ldots)$.

$W(t)$ értéke a $t_i, i \geq 1$ ugráshelyeken a következő rekurzióval adható meg $(W_i = W(t_i), \quad W_1 = 0)$:

$$W(t_{i+1}) = [(W_i + Y_i) - X_i]^+, \quad i \geq 1 \ ,$$

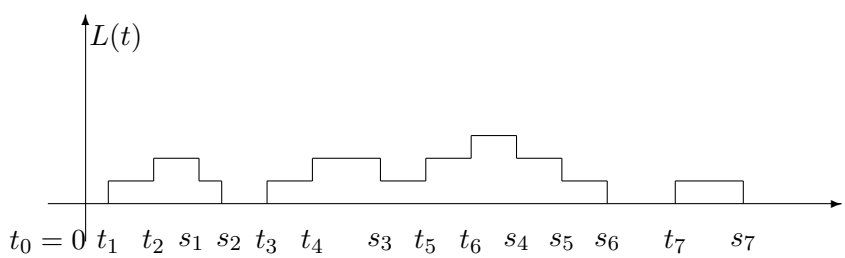
ahol $x^+ = \max\{x, 0\}, x \in \mathbb{R}$. Könnyen látható az is, hogy két ugráshely között $W(t)$ értékére érvényes a következő összefüggés:

$$\begin{aligned} &W(t) = 0, && 0 \leq t \leq t_1 \ , \\ &W(t) = [W_i + Y_i - (t - t_i)]^+, && t_i < t \leq t_{i+1}, \quad i \geq 1 \ . \end{aligned}$$

VIRTUÁLIS-VÁRAKOZÁSI-IDŐ(t)

```
1  i = 1
2  t(1) = X(0)
3  τ = X(0)
4  W(1) = 0
5  if t ≤ τ
6     return W(t) = 0
7  while t(i) < t
8           i = i + 1
9           τ = τ + X(i − 1)
10          t(i) = τ
11 n = i − 1
12 for j = 1 to n
13       W(j + 1) = max{0, W(j) + Y(j) − X(j)}
14 W(t) = max{0, W(n) + Y(n) − (t − t(n))}
15 return W(t)
```

Az $L(t)$ sorhosszúság meghatározása. Tekintsük most a t időpontban a rendszerben tartózkodó igények $L(t)$, $0 \leq t < \infty$ számát ($L(0) = 0$). Feltesszük, hogy $L(t)$ balról folytonos. $L(t)$ időbeli változását a 27.3. ábra szemlélteti.



27.3. ábra. Sorhosszúság.

Látható, hogy a rendszerben tartózkodó igények $L(t)$ száma a virtuális várakozási idő kiszámításához hasonló egyszerű algoritmussal nem határozható meg, mivel ebben az esetben $L(t)$ ugrásai nem csak a $t_1, t_2, \ldots$ beérkezési időpontokban, hanem az igények $s_1, s_2, \ldots$ távozási időpontjaiban is bekövetkeznek. Egyszerűsíti a helyzetet, hogy elegendő meghatározni $L(t)$-t a foglaltsági periódusokon, mivel rajtuk kívül értéke $L(t) = 0$.

Jelölje $N(t)$, $t > 0$ a t időpontig a rendszerbe érkező igények számát, azaz

legyen

$$N(t) = \sum_{k=1}^{\infty} I(t_k < t) \ ,$$

ahol I az esemény indikátorfüggvénye (az értéke 1, ha az esemény bekövetkezik és 0 egyébként). Legyen

$$k_1 = 1 \ ,$$
$$k_{i+1} = \min\{k: \ X_{k_i} + \cdots + X_k > Y_{k_i} + \cdots + Y_k, \quad k > k_i\}, \quad i \geq 1 \ .$$

Látható, hogy ekkor a foglaltsági periódusok a következő alakban írhatók fel:

$$\Delta_i = (t_{k_i}, t_{k_i} + Y_{k_i} + \cdots + Y_{k_{i+1}-1}], \quad i \geq 1 \ .$$

A Δ_i foglaltsági periódus kezdetén mindig fennáll

$$L(t_{k_i}) = 0, \qquad L(t_{k_i} + 0) = 1 \ .$$

Minthogy a Δ_i foglaltsági periódusban pontosan $k_{i+1} - k_i$ igény kerül kiszolgálásra és a beérkezési és a távozási időpontok ismertek, ezért tetszőleges $i \geq 1$ mellett

$$t_k = t_{k_i} + X_{k_i} + \cdots + X_k \ ,$$
$$s_k = t_{k_i} + Y_{k_i} + \cdots + Y_k, \quad k_i \leq k \leq k_{i+1} - 1 \ ,$$

és így az $L(t), \ t \in \Delta_i$ sorhosszúság a következő módon határozható meg:

$$L(t) = \sum_{k=k_i}^{N(t)} I(t_k \leq t < s_k), \quad t \in \Delta_i \ ,$$

Megjegyzés. Ha bevezetjük a rendszerből a $t \ \ (t \geq 0)$ időpontig eltávozott igények $M(t)$ számát $(M(0) = 0)$, akkor

$$M(t) = \sum_{k=1}^{\infty} I(s_k < t) \ ,$$

és $L(t)$ felírható

$$L(t) = N(t) - M(t), \quad t \geq 0$$

alakban.

A következő algoritmus meghatározza a t időpontban a rendszerben tartózkodó igények számát. Az első rész generálja az igények belépési időpontjait, a második rész meghatározza az egyes foglaltsági periódusok befejezésének időpontját és a következő foglaltsági periódus kezdő igényét. Ha

a t időpontig elkezdődő utolsó foglaltsági periódus t-ig befejeződik, akkor a rendszer szabad, ellenkező esetben a harmadik rész kiszámítja a t-ig belépett, illetve a t-ig kiszolgált igények számát, és e két érték különbségét képezve visszaadja az aktuális sorhosszt. A rendszer működését a $[0, T]$ intervallumon vizsgáljuk.

SORHOSSZ(T, t)

```
1                                          // Belépési pontok generálása T-ig.
2  t(1) = X(0)
3  i = 1
4  if t < t(1)
5     return L(t) = 0
6  while t(i) ≤ T
7          i = i + 1
8          t(i) = t(i − 1) + X(i − 1)
9                  // A foglaltsági periódusok végpontjainak meghatározása.
10 i = 1
11 j = 1
12 k(1) = 1                          // A foglaltsági periódus kezdő igénye.
13 while t(k(i)) < t
14         m = i
15         while X(k(i)) + ··· + X(j) < Y(k(i)) + ··· + Y(j)
16             j = j + 1
17                n → j                  // A teljes foglaltsági periódus alatt
                                              kiszolgált utolsó igény száma.
18                k(i + 1) = j + 1       // A következő foglaltsági periódus
                                                  kezdő igénye.
19                s(k(i + 1) − 1) = t(k(i)) + Y(k(i)) + ··· + Y(j)
20                                                // A foglaltsági periódus vége.
21                i = i + 1
22                  // Feltételezzük, hogy a ciklusváltozó értéke megőrződik.
23 if s(k(i) − 1) ≤ t
24    return L(t) = 0
25                                  // Az utolsó foglaltsági periódus teljes volt.
26 n = k(i − 1)                 // Visszaállunk a foglaltsági periódus kezdetére.
27                          // A belépések számának meghatározása az utolsó
                                          nem teljes foglaltsági periódusban.
28 j = 0
```

```
29  while t(n + j) ≤ t
30          j = j + 1
31  J = j
32                  // A kiszolgált igények számának meghatározása az utolsó
33                                  // nem teljes foglaltsági periódusban.
34  ℓ = 0
35  while t(n) + Y(n) + ··· + Y(n + ℓ) ≤ t
36          ℓ ← ℓ + 1
37  L = ℓ
38  return L(t) = J − L
```

Bonyolultabb rendszer (például többcsatornás, többféle igényt esetleg más kiszolgálási algoritmussal kiszolgáló rendszer) esetén a rendszert csak több paraméter egyidejű meghatározásával adhatjuk meg. Ennek megfelelően a rendszer jellemzőinek leírása is lényegesen bonyolultabb eljáráshoz vezethet. Sokszor könnyebben levezethető eljáráshoz jutunk, ha a további vizsgálatok számára elegendő, speciális időpontokban tekintjük a rendszert, mint például az igények beérkezési vagy távozási időpontjaiban. Erre vonatkozóan megadunk néhány példát a fejezet végén.

A fenti eljárás bármely X_0, (X_i, Y_i), $\quad i = 1, 2, \ldots$ (akár determinisztikus) beérkezési folyamat esetén megadja a rendszer t időpontbeli $W(t)$ és $L(t)$ jellemzőinek értékét, de megfelelő módon eljárva megadható a rendszer többi jellemzője is az idő függvényében.

Sztochasztikus rendszerek esetén az X_i és Y_i mennyiségek véletlen mennyiségek és leggyakrabban homogén módon viselkednek: az $(X_i, Y_i), i = 1, 2, \ldots$ valószínűségi változók függetlenek és azonos eloszlásúak, melyek függetlenek az X_0 valószínűségi változótól. Ha X_0 és $X_i, i \geq 1$ eloszlása különbözik, késleltetett esetről beszélünk. Azt is szokás feltenni, hogy minden egyes i mellett az X_i és $Y_i, i = 1, 2, \ldots$ valószínűségi változók függetlenek. Ez a feltevés nem zárja ki a determinisztikus esetet, vagyis azt, hogy X_i és Y_i 1 valószínűséggel konstans értéket vegyenek fel.

Ilyen rendszerek vizsgálatánál a kérdés az, hogy a rendszer jellemzőinek mikor létezik és hogyan határozható meg a határeloszlása, illetve, amire csak speciális esetekben tudunk válaszolni, hogyan határozhatók meg a jellemzők időtől függő eloszlásai. A jellemzők numerikus meghatározásának általános módszereként tárgyaljuk a fejezet végén a szimulációs módszert, melynél a rendszer jellemzőinek kiszámítását szolgáló (a fenti két példában is bemutatott) algoritmusok alapvető szerepet játszanak. Az analitikus megközelítés általában csak erősen korlátozottan és akkor is csak a konkrét esetre kidolgozva használható.

Tekintsük a korábban vizsgált $G|G|1|\infty$ rendszer esetén a következő jellemzőket (ezek a jellemzők más TKR-ekre is értelmezhetők):

A rendszerben tartózkodó igények átlagos száma:

$$\overline{L}_T = \frac{1}{T}\int_0^T L(t)dt \ ;$$

Egy igényre jutó átlagos várakozási idő:

$$\overline{W}_T = \frac{1}{\overline{L}_T}\frac{1}{T}\int_0^T W(t)dt \ ;$$

A következő algoritmus a rendszerben tartózkodó igények átlagos számát határozza meg. Felhasználja a sorhosszra vonatkozó algoritmus által generált belépési időpontokat ($t(j)$), az egyes foglaltsági periódusok végpontjait ($s(k(i+1)-1)$) és a t időpontig elkezdődött foglaltsági periódusok számát (m).

Az első rész generálja az egyes igények kiszolgálása befejeződéseinek időpontjait. A második rész az egymás után következő belépési és befejeződési pontokon végigmenve elvégzi a szükséges összegzést és kezeli az utolsó (nem teljes) foglaltsági periódust.

Átlagos-sorhossz(t)

```
1    // Az egyes igények kiszolgálása befejeződési időpontjainak előállítása
                    // (a t időpontig m foglaltsági periódus kezdődik el).
2   j = 1
3   k(1) = 1
4   s(1) = t(1) + Y(1)
5   for i = 1 to m
6       while t(k(i)) + Y(k(i)) + ··· + Y(j) < t(k(i+1))
7             s(j+1) = s(j) + Y(j)
8             j = j + 1
9                                                    // Összegzés.
10  S = 0
11  i = 1                                  // A foglaltsági periódus sorszáma.
12  k(1) = 1
```

```
13  τ = t(1)                                          // Az aktuális pozíció pointere.
14  a = 1                                                  // A belépési pont indexe.
15  b = 1                                                // A befejeződési pont indexe.
16  while (t(k(i)) < t) ∧ (s(k(i + 1) − 1) < t)
17          d = 1 // Jelenlévő igények száma egy foglaltsági perióduson belül.
18          while s(b) ≤ s(k(i + 1) − 1)
19                  if t(a + 1) < s(b)
20                     S = S + d · (t(a + 1) − τ)
21                     d = d + 1
22                     a = a + 1
23                     τ = t(a)
24                     if s(b) < t(a + 1)
25                        SS + d · (s(b) − τ)
26                        d = d − 1
27                        b = b + 1
28                        τ = s(b)
29                        i = i + 1
30                                // Az utolsó foglaltsági periódus befejeződik t-ig.
31  if (s(k(m + 1) − 1) ≤ t) ∨ (t(k(m + 1)) = t)
32     return L = S/t
33                                // Az utolsó foglaltsági periódus túlnyúlik t-n.
34  τ1 = t(a)
35  τ2 = t(a)
36  d = 1
37  while τ1 ≤ t
38          if τ1 ≤ t ≤ τ2
39             S = S + d · (t − τ1)
40             return L = S/t
41             if t(a + 1) < s(b)
42                τ1 = τ2
43                τ2 = t(a + 1)
44                S = S + d · (τ2 − τ1)
45                d = d + 1
46                a = a + 1
47                else   τ1 = τ2
48                       τ2 = s(b)
49                       S = S + d ∗ (τ2 − τ1)
50                       d = d − 1
51                       b = b + 1
```

Foglaltsági periódusok eloszlása: azoknak az egymás utáni leghosszabb időszakaszoknak az eloszlása, amikor a kiszolgáló egység foglalt. (A 27.3. ábrán ezek a periódusok: (t_1, s_2), (t_3, s_6), (t_7, s_7).)

Szabad periódusok eloszlása: azoknak az egymás utáni leghosszabb időszakaszoknak az eloszlása, amikor a kiszolgáló egység üres. (Ezek a periódusok a 27.3. ábrán: (t_0, t_1), (s_2, t_3), (s_6, t_7).)

Egyes vizsgálatokban alapvető kérdés, hogy létezik-e a rendszerben tartózkodó igényeknek számának

$$\pi_k = \lim_{t\to\infty} P(L(t) = k), \quad k = 0, 1, \dots, \qquad \sum_{k=0}^{\infty} \pi_k = 1$$

állandósult (stacionárius) eloszlása és az hogyan határozható meg.

Hasonló kérdés vethető fel a rendszerben tartózkodó igények átlagos számával, illetve az egy igényre jutó átlagos várakozási idővel kapcsolatban (létezik-e $T \to \infty$ mellett az $\overline{L}_T \to \overline{L}$, illetve $\overline{W}_T \to \overline{W}$ határérték), míg a foglaltsági/szabad periódusok esetében a periódushosszak eloszlása érdekes. Ezekkel a kérdésekkel jelen fejezetben részletesen nem foglalkozunk, mindössze egy egyszerű, de a gyakorlat számára fontos TKR esetében adunk formulákat.

27.2. Klasszikus tömegkiszolgálási rendszer

Tekintsük az ún. klasszikus tömegkiszolgálási rendszert, amelyre a határértékek léteznek és az eloszlások meghatározhatók (a tömegkiszolgálás-elméletben ennek a rendszernek a szokásos jelölése $M/M/1$).

Legyen a beérkezési idők közös eloszlása $\lambda > 0$, míg a kiszolgálási időké $\mu > 0$ paraméterű exponenciális eloszlás. Megjegyezzük, hogy e feltétel szerint az időegységre jutó beérkezések átlagos száma λ, míg állandó terhelés mellett a kiszolgáló egységen kiszolgált igények egy időegységre jutó átlagos száma μ. Feltesszük, hogy $\rho = \lambda/\mu < 1$. Ebben az esetben a t $(t \geq 0)$ időpontig a rendszerbe érkező igények $N(t)$ száma egy $\lambda > 0$ intenzitású $N(t)$, $t \geq 0$ Poisson-folyamattal azonosítható. Ez utóbbi egy olyan sztochasztikus folyamatot jelent, amelyre tetszőleges $x_0 = 0 < x_1 < x_2 < \cdots$ sorozat mellett az $N(x_{j+1}) - N(x_j)$, $i = 0, 1, \dots$ növekmények független valószínűségi változók és Poisson-eloszlásúak $\lambda(x_{j+1} - x_j)$ paraméterrel, vagyis

$$P(N(x_{j+1}) - N(x_j) = k) = \frac{[\lambda(x_{j+1} - x_j)]^k}{k!} e^{-\lambda(x_{j+1}-x_j)}, \qquad k = 0, 1, \dots .$$

Erre a tömegkiszolgálási rendszerre fennállnak a következő összefüggések.

Esetünkben a rendszerben tartózkodó igények határeloszlása létezik, mely megadható a következő alakban:

$$\pi_0 = 1 - \rho, \quad \pi_k = \pi_0 \rho^k, \quad k = 0, 1, \dots \ .$$

Látható, hogy a ρ mennyiségnek igen fontos jelentése van, megadja határértékben annak a valószínűségét, hogy a rendszer foglalt.

A rendszerben tartózkodó igények átlagos számának, illetve az egy igényre jutó átlagos várakozási időnek 1 valószínűséggel létező határértékére igaz

$$\overline{L} = \frac{\rho}{1-\rho}\ , \quad \text{illetve} \quad \overline{W} = \frac{1}{1-\rho}\frac{1}{\mu} = \frac{1}{\lambda - \mu}\ .$$

A szabad periódus eloszlása triviálisan adódik, éppen a beérkezési időközök közös λ paraméterű exponenciális eloszlása. A foglaltsági periódus eloszlásának sűrűségfüggvénye ugyancsak meghatározható, bár az eredmény csak speciális matematikai függvénnyel fejezhető ki:

$$h(x) = \begin{cases} \frac{1}{x\rho^{1/2}} e^{-(\lambda+\mu)x} I_1(2x(\lambda\mu)^{1/2}), & \text{ha } x > 0\ , \\ 0, & \text{ha } x \le 0\ , \end{cases}$$

ahol az $I_1(x)$ függvény az elsőfajú Bessel-függvényt jelöli.

27.3. Kiszolgálási algoritmusok

Egy TKR vizsgálatát alapvetően az határozza meg, hogy a rendszer működését milyen szempontból elemezzük. Általában olyan mutatók vizsgálata a cél, amelyek valamilyen értelemben jellemzik a rendszer működésének hatékonyságát. Ezek a vizsgálatok nem csak egy működő rendszer elemzésére és hatékonyságának növelésére irányulhatnak, hanem már a tervezés fázisában is fontos támpontokat jelenthetnek.

Egy bonyolultabb, több kiszolgáló egységből álló tömegkiszolgálási hálózat működésének hatékonysága nem csak attól függ, hogy az egyes elemei milyen gyorsan és jól működnek, hanem sok esetben szerepe van annak is, hogy a kiszolgálás milyen algoritmus szerint történik. ***Kiszolgáló algoritmuson*** olyan kiszolgálási eljárásokat (szabályokat) értünk, amelyek tetszőleges időpontban megmondják, hogy a rendszerben lévő igények közül melyiket kell éppen kiszolgálni, milyen sorrendben és milyen hosszú ideig tart a kiszolgálás.

A kiszolgálási algoritmusokkal szemben támasztott követelmények különbözőek lehetnek. Adatátvitel esetén lehet a kiszolgálás a késleltetéssel szemben érzékeny (például telefon, videokonferencia), de más és más lehet egy esetleges adatvesztés következménye is (például egy számítógépes program esetén

az átvitt anyag használhatatlanná válhat, ugyanakkor beszédátvitelnél esetleg csak kismértékű minőségromlással jár). Ennek megfelelően a kiszolgálási algoritmusoknak általában meghatározott kritériumoknak kell eleget tenniük. Megjegyezzük, hogy a kiszolgálási kapacitás növelése a kapacitás szempontjából sok esetben nem hatékony, viszont a megfelelő kiszolgálási algoritmus kialakításával jelentős eredményeket lehet elérni.

27.3.1. A leggyakrabban előforduló kiszolgálási algoritmusok

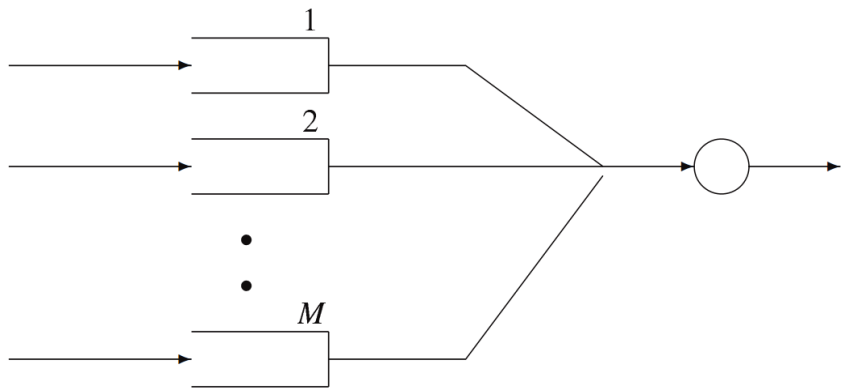
FIFO (First-in-first-out). Ha a kiszolgáló egység üres az igény beérkezésének időpontjában, akkor a kiszolgálása haladéktalanul megkezdődik és egészen addig tart, amíg a teljes kiszolgálás befejeződik. Ha a beérkező igény kiszolgáló egységet nem találja üresen, akkor várakozási sorba kerül és a továbbiakban az igények kiszolgálása a beérkezés sorrendjében történik.

LIFO (Last-in-first-out). A kiszolgáló egységnél a sorban álló igények kiszolgálása úgy történik, hogy egy igény kiszolgálása befejeződése esetén a rendszerbe utoljára belépett igény kiszolgálása kezdődik meg és tart a teljes kiszolgálásig.

Az FCFS (First-come-first-served, az először érkezett lesz először kiszolgálva) és az LCFS (Last-came-first-served, az utoljára érkezett lesz először kiszolgálva) elvek hasonlítanak a FIFO és LIFO elvekhez: a kiszolgáló egységnél a sorban álló igények közül az az igény kerül először kiszolgálásra, amelyik előbb (illetve később) érkezett. E két elvnél a rendszerből való távozás sorrendje függhet (az alábbiakban is ismertetett) más kiszolgálási elvektől is. Minthogy sok rendszerre (például egycsatornás rendszerek esetén) az eredmény ugyanaz, ezért néha a FIFO és FCFS (LIFO és LCFS) kiszolgálási eljárásokat egymás szinonimájaként használják, de jó tudni a közöttük levő különbséget.

Véletlen-sorrend (random). A sorban állók közül a választás véletlenszerűen, például egyforma valószínűséggel, azaz egyenletes eloszlás szerinti sorsolással történik és a kiszolgálás az igény teljes kiszolgálásáig tart.

Prioritásos rendszerek. Ezekben a rendszerekben az igényeket különböző véges M számú osztályba soroljuk (lásd 27.4. ábra). A rendszerbe lépő minden egyes igény egy, a típusának (osztályának) megfelelő indexet (számot), ún. prioritási indexet kap. Az igény prioritása akkor nagyobb, amikor a prioritási indexe nagyobb. Amikor a kiszolgáló egység felszabadul, akkor a sorban álló igények közül a magasabb prioritású igény kerül kiszolgálásra. Azonos típusú (prioritású) igények közül a választás a fenti elvek valamelyike szerint történhet. Az osztályba sorolás és a prioritások hozzárendelése az egyes osztályokhoz egyfajta eszközt jelent a rendszer hatékonyságának optimalizálására.

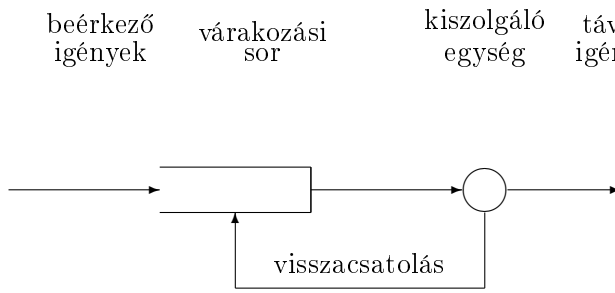


27.4. ábra. Prioritásos rendszer

Megszakításos prioritásos rendszer (abszolút prioritásos rendszer, preemptive service discipline). Ha egy igény kiszolgálási ideje alatt egy magasabb prioritású igény érkezik a rendszerbe, akkor a kiszolgálás alatt lévő alacsonyabb prioritású igény kiszolgálása azonnal befejeződik és a kiszolgáló egység a magasabb prioritású igény kiszolgálását kezdi meg. A már lezajlott kiszolgálás különböző módokon vehető figyelembe:

- Magasabb prioritású igény belépése esetén az alacsonyabb prioritású kiszolgálása megszakad és az összes magasabb prioritású igény kiszolgálása után folytatódik. Az elvégzett kiszolgálást figyelembe vesszük, a kiszolgálási idő a már elvégzett munkával csökken.
- A kiszolgálás az előbbi pontban leírt módon zajlik, de az alacsonyabb prioritású igényeknél a megszakításig elvégzett munkát nem vesszük figyelembe, a kiszolgálás elölről kezdődik.
- Magasabb prioritású igény belépése esetén az alacsonyabb prioritású kiszolgálása megszakad és az igény elvész.

Nemmegszakításos prioritásos rendszer (relatív prioritásos rendszer, nonpreemptive service discipline). A különbség az előző kiszolgálási elv és eközött abban áll, hogy ha egy igény kiszolgálási ideje alatt egy magasabb prioritású igény érkezik a rendszerbe, akkor az új igény kiszolgálása csak azután kezdődik meg, hogy az éppen kiszolgálás alatt lévő igény kiszolgálása befejeződik. A relatív prioritásos rendszer jellemző példái az LPT (Longest-processing-time, leghosszabb kiszolgálási idő) és az SPT (Shortest-processing-time, legrövidebb kiszolgálási idő) kiszolgálási diszciplínák, amelyek esetén a kiszolgálás a kiszolgálási idők csökkenő (növekvő) sorrendjében történik.



27.5. ábra. Kiszolgálás visszacsatolással.

A következő kiszolgálási algoritmus széles körben elterjedt a nagy teljesítményű és nagy forgalmú kiszolgáló egységeken történő kiszolgálás szabályozására. Igazi jelentőséget először a nagy teljesítményű multiprogramozású számítógépeknél nyert, de alapvető szerepet játszik a modern telekommunikációs rendszerekben is. Ebben az esetben egy igény egyszerre csak részkiszolgálást nyer és utána visszakerül a várakozási sorba, a kiszolgáló egység és a várakozási sor között egyfajta visszacsatolás van (lásd 27.5. ábra).

KÖRBEFORGÓ KISZOLGÁLÁS (ROUND-ROBIN). Minden beérkező igényhez hozzárendelünk egy, akár véletlen nagyságú, kiszolgálási időt (kvantumot). Ha ez alatt az igény kiszolgálása nem fejeződik be, akkor visszakerül a várakozási sorba és ott tartózkodik, amíg a jelenlévő további igények meg nem kapják a nekik előírt kiszolgálási kvantumot. Az újabb kiszolgálásra kerülésnél egy újabb kvantum kerül meghatározásra és a kiszolgálás a leírt módon folytatódik. Ha a kvantum nagysága fix és azonos valamennyi igényre, akkor szokás ezt az elvet IDŐOSZTÁSOS (TIME-SHARING) algoritmusnak nevezni.

PROCESSZOROSZTÁSOS ALGORITMUS (PROCESSOR-SHARING, PS). A processzor egyszerre dolgozik az összes igényen, k igény esetén a kapacitásának $1/k$-adával szolgálja ki az egyes igényeket. Az időosztásos algoritmus esetén meghatározott Q kvantumot egyenlő arányban szétosztjuk a jelenlévő igények között. Tekinthetjük úgy, hogy a rendszerben található igények kiszolgálása egyszerre folyik, a kvantum végén valamennyi igény hátralévő kiszolgálási ideje a Q/k értékkel csökken. Ez a kiszolgálási szabály kedvez azoknak az igényeknek, amelyek rövid kiszolgálási időt, de gyors válaszokat igényelnek.

LEGRÖVIDEBB ELTELT IDŐ ALGORITMUS (SHORTEST-

ELAPSED-TIME, SET). A különböző kiszolgálási algoritmusok gyakran párosíthatók a prioritásos kiszolgálási szabállyal. Ha a kiszolgálási időt nem ismerjük és a rövidebb kiszolgálási idejű igényeket „előnyben" akarjuk részesíteni, akkor alkalmazhatjuk a SET kiszolgálási diszciplínát a következő módon. Az aktuális igény egységnyi kiszolgálása után áttérünk az addig legkevesebb kiszolgálást kapott igényre a várakozók közül. Ha több ilyen van, akkor például a beérkezési sorrend szerint választhatunk. A rendszer prioritásosként interpretálható oly módon, hogy a k-adik prioritási szinthez rendeljük a már $k-1$ egységnyi kiszolgálást kapott igényeket. A k-adik egységnyi kiszolgálás után az igény vagy elhagyja a rendszert (ha a kiszolgálás befejeződött), vagy csatlakozik a $(k+1)$-edik sorhoz (ha további kiszolgálás szükséges).

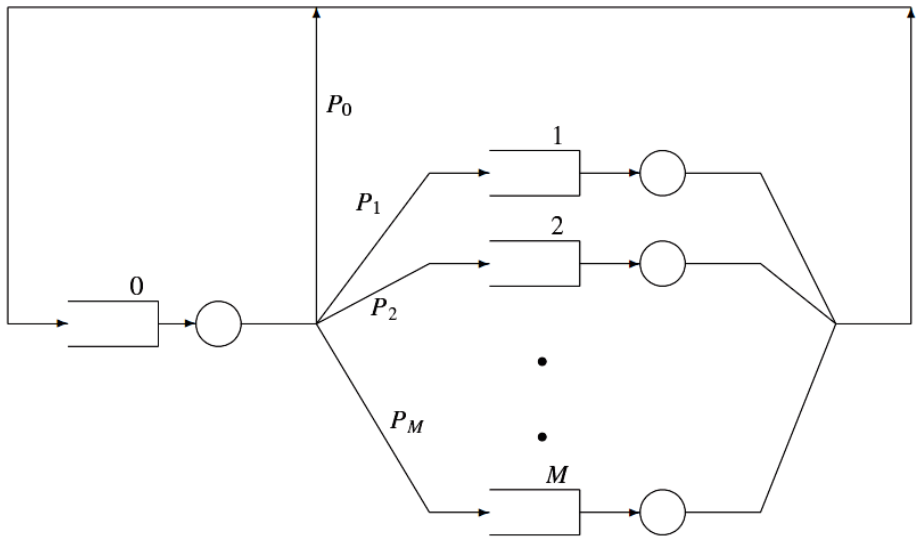
LEGRÖVIDEBB HÁTRALÉVŐ IDŐ ALGORITMUS (SHORTEST-REMAINING-PROCESSING-TIME, SRPT). Amennyiben a szükséges kiszolgálási idő nagysága ismert, akkor a kiszolgálás ütemezhető ennek alapján. Egy lehetséges megoldás a fentiekben ismertetett SPT diszciplína. Az SRPT kiszolgálási elv esetében a kiszolgáló eszközön mindig az az igény van, amelynek a kiszolgálásából a legkevesebb van hátra. Tekinthetjük úgy, hogy a rendszerbe különböző igényfolyamatok lépnek be, ezeket a szükséges kiszolgálási idő határozza meg. Miután az igény a k-adik szinten megkapja a Q nagyságú kiszolgálást, csatlakozik a $k-1$-edik szinthez. Az újonnan belépő igények a kiszolgálási idejük által meghatározott sorhoz csatlakoznak.

ISMÉTLÉSES RENDSZEREK (RETRIAL SYSTEMS). Amennyiben a rendszerbe lépő igény kiszolgálását elutasítjuk, akkor az a későbbiekben újra kezdeményezheti a kiszolgálást. Ilyen például a telefonhívások megismétlése foglaltság esetén, a repülőgépek leszállása (a várakozó repülőgép körözés esetén csak egy pontban kezdheti meg a leszállási műveletet) vagy az ütközések kezelése az ALOHA-ban. Az ismétlés történhet véletlen idő múlva, vagy valamilyen szabályozott módon.

27.4. Centrális zárt rendszerek

A számítógépek egyik egyszerűbb matematikai modelljének is tekinthető a következő, s a 27.6. ábrával szemléltetett rendszer.

A rendszerben rögzített n számú igény van állandóan (emiatt nevezzük *zártnak*) és a megadott ábra szerint vándorolhatnak az igények. Minden egyes kiszolgáló egység előtt kellő számú várakozási hely van a felmerülő (legfeljebb $n-1$) igény számára. A kiszolgáló egységeken a beérkezés sorrendjében (FIFO elv) szerint történik a kiszolgálás, a kiszolgálási idők összességükben függetlenek, s az i-edik egységen azonos $F_i(x), 0 \leq i \leq M$ eloszlásúak. Ha egy kiszolgálás befejeződik a 0-adik egységen, akkor onnan az igény $p_i, 0 \leq i \leq M$



27.6. ábra. Centrális zárt rendszer

($p_i \geq 0, p_0 + \cdots + p_M = 1$) valószínűséggel átmegy az i-edik egységre, a rendszer állapotától és a kiszolgálási időktől függetlenül. Ha pedig egy igény kiszolgálása valamely$i$-edik, $1 \leq i \leq M$, egységen ér véget, akkor onnan a 0-adik kiszolgáló egységhez kerül. A 0-adik kiszolgáló egységet ezen kitüntetett szerepe miatt *központi (centrális) kiszolgáló egységnek* nevezzük. A vizsgált rendszer ezért kapta a *centrális zárt TKR* elnevezést. Megmutatható, hogy ha az összes kiszolgálás várható értéke véges, akkor a rendszer különböző mutatóinak létezik határeloszlása (állandósult eloszlás).

Számítógépek esetén szigorúnak tűnhet az a megszorítás, hogy a rendszer zárt, vagyis állandóan n számú igény tartózkodik a rendszerben. Ha a rendszer kihasználtságát vizsgáljuk, akkor nem mindegy az, hogy milyen külső terhelés mellett tesszük, a rendszer zártsága ebben az esetben azt jelenti, hogy a rendszer külső terhelése állandó. Ekkor ha egy program futása befejeződik, akkor helyette egy másik program lép be a rendszerbe, ami a 0-adik egységről a 0-adik egységre való visszatérést jelenthet az adott esetben. Meg kell jegyezni azt is, hogy a rendszer hatékonysági mutatói (például CPU kihasználtság, időegységre jutó kiszolgált programok száma stb. jelentősen függhet a különböző kiszolgálási algoritmusoktól).

Legyen $t > 0$ egy előre rögzített időpont. Az alábbi algoritmussal meghatározható a t időpontban a rendszerben az egyes egységeknél tartózkodó igények száma és az egyes igények hátralévő kiszolgálási ideje.

Az algoritmus leírásához vezessük be a következő jelöléseket:

$L0(i)$, $Lt(i)$, $L(i)$, $0 \leq i \leq M$ – a vizsgálat kezdetén (a 0 időpontban, a t időpontban, illetve a vizsgálat aktuális időpontjaiban a rendszer egyes egységein tartózkodó igények száma;

$X0(i,j)$, $Xt(i,j)$, $1 \leq j \leq n,\ 0 \leq i \leq M$ – az előzőekhez hasonló időpontokban a rendszer egyes egységein lévő és a beérkezés sorrendjében tekintett igények hátralévő kiszolgálási idejei;

$T(k)$, $k = 1, 2, \ldots$ – az az időpont, amikor egy (vagy egyszerre több) igény kiszolgálása valahol befejeződik a rendszerben és ez sorrendben a k-adik ilyen időpont a rendszer működésének (0 időpontbeli) elkezdése óta;

generate $X^{(i)}$, illetve *generate* J – az $F_i(x)$ eloszlásfüggvény, illetve a $P(J = i) = p_i$, $1 \leq i \leq M$ eloszlás szerinti véletlen szám generálása, vagy determinisztikus esetben az előre megadott adatsorokból a soron következő érték kiválasztása.

CENTRÁLIS-ZÁRT-RENDSZER-ÁLLAPOTA(t, M)

```
1  for i = 0 to M                                  // Kezdeti értékek beállítása.
2      L(i) = L0(i)
3      for j = 1 to n
4          X(i,j) = X0(i,j)
5  K = 0
6  T(K) = 0
7  x = min(X(i,1) : X(i,1) > 0 ∧ 0 ≤ i ≤ M
8            // Időpont, amikor először fejeződött be egy igény kiszolgálása.
9  while t ≤ T(K) + x                  // A t időpontig tartó vizsgálat ciklusa.
10         K = K + 1
11         T(K) = T(K) + x
12         for i = 0 to M
13         // A kiszolgáló egység(ek) megjelölése, ahol a T(K) időpont után
                                                  // először fejeződik be kiszolgálás.
14             N(i) = 0
15             if X(i,1) == x
16               N(i) = 1
17             if X(i,1) > x        // A hátralévő kiszolgálási idő csökkentése.
18               X(i,1) = X(i,1) − x
```

```
19  // A megjelölt kiszolgáló egységekre a T(K)+x utáni állapot beállítása:
20          if N(0) == 1
21            L(0) = L(0) − 1
22            m = generate J
23            y = generate X^(m)                                  // a CPU-n.
24            L(m) = L(m) + 1
25            X(m, L(m)) = y
26            if L(0) ≥ 1
27              for j = 1 to L(0)
28                  X(0, j) = X(0, j + 1)
29            X(0, L(0) + 1) = 0
30          for i = 1 to M
31              if N(i) = 1
32                L(i) = L(i) − 1
33          z = generate X^(0) // a többi kiszolgáló egységen.
34                             L(0) = L(0) + 1
35                             X(0, L(0)) = z
36                             for j = 1 to L(i)
37                                 X(i, j) = X(i, j + 1)
38                             X(i, L(i) + 1) = 0
39          x = min(X(i, 1) : 0 ≤ i ≤ M ∧ X(i, 1) > 0)
40                        // A következő változás időpontjának meghatározása.
41  for i = 0 to M
42      Lt(i) = L(i)
43      Xt(i, 1) = max(0, X(i, 1) − (t − T(K)))
44      for j = 2 to n                          // A t időpontbeli állapot beállítása.
45          Xt(i, j) = X(i, j)
46  return Lt(i), 0 ≤ i ≤ M), (Xt(i, j), 1 ≤ j ≤ Lt(i), 0 ≤ i ≤ M
```

Megjegyzés. A 7. (és 39.) sorban levő művelet valójában egy kis önálló algoritmust takar, amely M darab nemnegatív $X(0,1),\ldots,X(m,1)$ szám közül kiválasztja a legkisebb pozitív számot. Ennek megírása igen egyszerű feladat, ezért részletezésére nem térünk ki.

A fenti algoritmust egyszerű kiegészítéssel alkalmazni lehet a rendszer $[0,t]$ időintervallumra vonatkozó átlagos hatékonysági mutatóinak kiszámítására (például az egyes egységeken az átlagos sorhosszúság, átlagosan kiszolgált igények száma, átlagos várakozási idő stb.). Ehhez mindössze annyit kell tenni, hogy a 34. sor után kumuláljuk az egyes $[T(k),T(k+1)]$ szakaszokra, illetve az utolsó, t-t meg nem haladó $T(k)$ és t közé eső időintervallumra a rend-

szerre vonatkozó integrális mennyiségeket és az algoritmus befejeződése után átlagoljuk t-vel.

27.5. A tömegkiszolgálási rendszerek vizsgálata szimulációval

A tömegkiszolgálási rendszerek jellemzőinek vizsgálatánál gyakran még egyszerűnek látszó modellek esetén sem tudunk egzakt formulákat adni. Ilyen esetekben hasznos és hatékony módszer a szimuláció. Ez a megközelítés azon alapszik, hogy a rendszer működésére nagyszámú kísérletet végzünk, mintegy a „rendszer működését szimuláljuk" és a nyert eredmények alapján vonunk le következtetéseket. A módszer alkalmazása egyébként nem szorítkozik csak a tömegkiszolgálási rendszerek vizsgálatára, sikerrel alkalmazható determinisztikus problémák megoldására is, például bonyolult integrálok kiszámítására, differenciál- és integrálegyenletek megoldására stb., ennek miként jére itt nem térünk ki.

A szimulációs módszert a már korábban vizsgált $G|G|1|\infty$ rendszeren keresztül mutatjuk be a korábbi feltételek teljesülése mellett, azonban a módszer hasonló módon alkalmazható más TKR-ek esetén is. Jelölje X_0, (X_n, Y_n), $n = 1, 2, \ldots$ a rendszer beérkezési folyamatát és vizsgáljuk a rendszer egyik jellemzőjét, mondjuk az $\overline{L}_T$ átlagos sorhosszúságot a $[0, T]$ intervallumon. A vizsgálat a következő lépésekből áll:

1. Pszeudovéletlenszám-generátor felhasználásával létrehozzuk a rendszert jellemző eloszlással a véletlen x_0, (x_n, y_n), $1 \leq n \leq n_0$ számsorozatot (véletlen beérkezési folyamatot, ahol n_0 jelenti azoknak az igényeknek a számát, melyek a $[0, T]$ intervallumon érkeznek a rendszerbe, azaz

$$n_0 = \max\{k : t_k = x_0 + \cdots + x_k < T\} .$$

2. Meghatározzuk az s_k, $1 \leq k \leq n_0$ távozási időpontokat.

3. Kiszámítjuk az

$$\overline{L}_T = \frac{1}{T} \int_0^T L(t)dt$$

átlagos sorhosszúságot. Ebben az esetben felhasználhatjuk az $L(t) = N(t) - M(t)$, $t \geq 0$ összefüggést, ahonnan

$$\overline{L}_T = \frac{1}{T} \int_0^T L(t)dt = \frac{1}{T} \int_0^T [N(t) - M(t)]dt = \frac{1}{T} \sum_{k=1}^{n_0} [\min(s_k, T) - t_k] .$$

Megjegyezzük, hogy az összegzésben a $\min(s_k, T) - t_k$ éppen azt az időtartamot jelenti a $[0, T]$ intervallumon, amennyit a k-adik igény eltölt a rendszerben.

Megjegyzés. Nagy T (nagy n_0) esetén helytakarékosabb eljárást is követhetünk: egymás utáni foglaltsági periódusokra összegezzük az integrális mennyiségeket egészen a T időpontig és a végén normálunk T-vel. Ekkor elegendő csak az új foglaltsági periódusra vonatkozó t_k és s_k időpontokat ismerni, a megelőző időpontokra a további számításoknál már nincs szükség.

Megjegyzés. Az ún. *regeneratív szimulációs módszerrel* is vizsgálhatjuk az átlagos sorhosszúságot, amely bár kicsit bonyolultabb ebben az esetben, de további következtetésekre ad módot.

Ez a megközelítés azon az észrevételen alapszik, hogy a rendszer a t_{k_i}, $i = 1, 2, \ldots$ időpontokban, az üres periódus után az éppen t_{k_i} időpontban beérkező igényt kezdi kiszolgálni és a rendszer további állapotváltozásai teljes mértékben függetlenek a $0 \leq t \leq t_{k_i}$ időintervallumon felvett állapotától, vagyis a rendszer a $t_{k_i}, i = 1, 2, \ldots$ időpontokban mintegy *regenerálódik*. Ez azt jelenti, hogy az egymást követő foglaltsági és üres periódusokból álló szakaszok függetlenek, és függetlenek lesznek azok a folyamatszakaszok is, amelyek leírják a rendszer állapotát. Innen következik, hogy a rendszernek az egyes regeneratív szakaszokon számított jellemzői független azonos eloszlású valószínűségi változókat képeznek, melyekre alkalmazható a nagy számok törvénye, a centrális határeloszlás tétel és ennek alapján például konfidencia intervallum is szerkeszthető a rendszer mutatóira. E tulajdonság felhasználásával bizonyítható például az is, hogy a rendszerben tartózkodó igények számának létezik-e határeloszlása ($T \to \infty$), vagy létezik-e (1 valószínűséggel) határértéke az $\overline{L}_T$ és $\overline{W}_T$ átlagos értékeknek. Számos TKR rendelkezik a regeneratív tulajdonsággal és hatékonyan vizsgálhatók ezzel a módszerrel. Az $M|G|1$ rendszer esetén ezen a módon az ergodikus eloszlást meghatározhatjuk a kiszolgálási idő eloszlásának ismerete nélkül, csak a közvetlenül ismert adatokra (beérkezési ráta, egy igény kiszolgálási idejének várható értéke és az egy kiszolgálás alatt belépő igények száma) támaszkodva.

Visszatérve konkrét rendszerünk vizsgálatához, meghatározzuk a korábban megadott eljárással a $[0, T]$ intervallumot lefedő minimális n_* számú Δ_j, $1 \leq j \leq n_*$ foglaltsági periódust a hozzátartozó k_j, $1 \leq j \leq n_*$ kezdő indexekkel együtt, ahol

$$n_* = \min\{k_j - 1 : \ k_j > n_0, \quad j \geq 1\} \ .$$

Legyen

$$I_j = \int\limits_{\Delta_j} L(t)dt = \sum_{k=k_j}^{k_{j+1}-1} I(s_k < T)[s_k - t_k], \quad j \geq 1$$

és

$$J_{n_*} = \int\limits_{\Delta_{k_{n*}}} L(t)dt = \sum_{k=k_{n*}}^{k_{n*+1}-1} I(s_k \geq T)[\min(s_k, T) - t_k] \ .$$

Ekkor a rendszerben tartózkodó igények átlagos száma

$$\overline{L}_T = \frac{1}{T}\int\limits_0^T L(t)dt = \frac{1}{T}\sum_{j=1}^{n_*-1} I_{k_j} + \frac{1}{T}J_{k_{n*}} = \frac{n_*}{T}\frac{1}{n_*}\sum_{j=1}^{n_*} I_{k_j} - \frac{1}{T}(I_{k_{n*}} - J_{k_{n*}}) \ .$$

Az egyenletben szereplő véletlen mennyiségekről a következőket tudjuk mondani:

- A valószínűségszámításból ismert felújításelmélet alapján a regenerációs periódusok n_* számára fennáll $T/n_* \to \eta$, $T \to \infty$ 1 valószínűséggel, ahol η jelenti egy regenerációs hossz várható értékét, amely minden regeneratív periódusra ugyanaz.
- Minthogy $I_{k_{n*}}$ eloszlása megegyezik I_{k_1} eloszlásával, ezért $J_{k_{n*}} \leq I_{k_{n*}}$ miatt nyilvánvalóan

  $$\frac{1}{T}(I_{k_{n*}} - J_{k_{n*}}) \leq \frac{1}{T}I_{k_{n*}} \to 0, \qquad T \to \infty$$

  1 valószínűséggel.
- Az I_j, $1 \leq j$ valószínűségi változók függetlenek és azonos eloszlásúak, $n_*/T \to 1/\eta$, $T \to \infty$ 1 valószínűséggel, ezért a véletlen tagszámú

  $$\frac{1}{n_*}\sum_{j=1}^{n_*} I_{k_j}$$

  összegre igaz a centrális határeloszlás tétel: legyen $\mu = E(I_{k_1})$, $\sigma = D(I_{k_j})$, akkor tetszőleges valós x szám mellett

  $$P\left(\frac{1}{\sqrt{n_*}\sigma}\sum_{j=1}^{n_*}(I_{k_j} - \mu) < x\right) \to \Phi(x), \quad T \to \infty \ ,$$

  ahol $\Phi(x)$ jelöli a standard normális eloszlásfüggvényt. Megjegyezzük, hogy mindhárom esetben külön vizsgálható a konvergencia sebessége. Ezek az összefüggések teszik lehetővé, hogy a szimulációs eredmény alapján konfidencia intervallumot is szerkesszünk az $\overline{L}$ átlagos sorhosszúságra.

27.5.1. Szimulációs eszközök

A távközlő rendszerek elemzésének későbbiekben vázolt analitikus eszközei mellett egy sokkal szélesebb körben elterjedt elemzési módszer a számítógépes szimuláció. A számítógépes szimuláció során a vizsgálandó rendszer vizsgált jellemzője szempontjából lényeges tulajdonságait kell egy szimulációs program számára leírni, és a szimulációs program ez alapján „lejátssza" a vizsgált rendszer működésének egy időszakát.

Egy távközlő rendszer viselkedését a rendszer véletlen elemeit is jellemző sztochasztikus folyamat segítségével vizsgáljuk. A szimuláció során a véletlen eseményekre, azok ismert sztochasztikus jellemzői (például független vagy együttes eloszlás) alapján sorsolunk. A sorsolást a programnyelvek túlnyomó többségében megvalósított *véletlen szám generátorral* végezzük. A tipikus véletlen szám generátorok valójában ún. pszeudovéletlen (álvéletlen) számokat adnak, azaz egy előre meghatározott véges hosszúságú sorozat elemeit adják vissza ciklikusan. Azt, hogy a program indításakor melyik elemtől kezdve kapjuk a sorozat elemeit, egy kezdeti érték (seed) határozza meg. A véletlen szám generátorok általában 0 és 1 között folytonos egyenletes eloszlású véletlen számot szolgáltatnak, amelyből egy transzformációs függvény segítségével képzünk tetszőleges eloszlású számokat. A véletlen szám generátor hossza, illetve a sorozat elemeinek tulajdonságai befolyásolhatják a szimuláció jóságát.

A számítógépes szimuláció a véletlen folyamat egy megvalósulását utánozza. A programot a véletlen szám generátor ugyanazon kezdőértékével újra indítva ugyanazt a megvalósulást fogja adni. Ez nagy segítséget nyújt a szimulációs programok hibáinak megtalálásához, mivel a mindig másfajta hibákat produkáló program kijavítása nagyon nehéz feladat. A programot a véletlen szám generátor más kezdőértékével indítva egy új megvalósulást fog létrehozni.

A vizsgált rendszernek a szimulációs programban kialakított modellje egy matematikai értelemben jól definiált sztochasztikus modell, így elvileg ennek a sztochasztikus modellnek analízis eszközökkel történő elemzésével is meghatározhatók a keresett teljesítmény jellemzők. Azonban az esetek többségében a szimulációs modellek olyan bonyolultak, hogy analitikus vizsgálatuk nem lehetséges. Javasolt azonban a szimulációs és a pontos, vagy ha nem lehetséges, akkor a közelítő analitikus vizsgálatok együttes végrehajtása, mivel bonyolult viselkedések esetén apró hibák is téves következtetésekre vezethetnek.

Szimulációs programot készíthetünk általános célú programozási nyelven (például C) vagy valamilyen szimulációs keretrendszer felhasználásával. Az első esetben a szimulációs modell mellett a szimuláció futtatását végző programrészt (szimulációs motor), valamint a vizsgált jellemzők kiszámításához

szükséges adatok gyűjtését és feldolgozását végző programrészt is el kell készíteni.

A második esetben a szimulációs keretrendszer már tartalmazza a szimulációs motort és általában támogatást nyújt a modell leírásához és a szükséges adatok megadásához. Azokat a számítógépes nyelvi vagy grafikus elemeket és a rájuk vonatkozó szabályrendszert, amelyekkel a vizsgálandó modellt leírjuk, modell leíró nyelvnek nevezzük. Alacsony szintű modell leírásnak nevezzük, ha a modell viselkedését egy általános célú programnyelv segítségével adjuk meg. Magas szintű modell leírás esetén előre definiált, esetenként grafikusan megjelenített modellrészekből építhetünk szimulációs modellt.

A magas szintű modell leírás nagyon megkönnyíti a bonyolult szimulációs modellek leírását, azonban azt a veszélyt hordozza magában, hogy a modell készítője előtt rejtve marad a modell alacsony síkjainak viselkedése, és így alapvetően egy ismeretlen rendszer elemzése történik.

A távközlő rendszerek véletlen forgalmi folyamatainak szimulációjára használható eszközök közül mutatunk be néhány példát a teljesség igénye nélkül.

Általános célú integrált szimulációs környezet

- MATLAB program SIMULINK csomagja

  (http://www.mathworks.com/products/simulink): a MATLAB az egyik legelterjedtebb programcsomag technikai számítások elvégzésére. A SIMULINK szimulációs csomag a MATLAB beépített függvényeire építve egy grafikus környezetet biztosít szimulációs modellek tervezéséhez és futtatásához.

Távközlő rendszerek szimulációja

- OMNeT++ (http://www.omnetpp.org):

  Az OMNet++ egy objektum orientált moduláris felépítésű diszkrét esemény szimulátor. Alkalmazható kommunikációs protokollok, számítógép hálózatok, forgalmi modellek, több processzoros és elosztott rendszerek szimulációjára. Támogat animációt és interaktív futtatást. Ingyenesen letölthető.

- ns2 (http://www.isi.edu/nsnam/ns/):

  Az ns2 programot kommunikációs hálózatok kutatási célú szimulációjára fejlesztették ki. Beépített modulok támogatják a TCP ptotokoll, az útvonalválasztási eljárások, a vezetékes és vezeték nélküli hálózatok vizsgálatát. Szintén ingyenesen letölthető.

- OPNET (http://www.opnet.com/):

  Az OPNET egy ipari célokra kifejlesztett szimulációs csomag, amely a

magas szintű hálózat modellezés mellett támogatja a hálózati technológiák megértését, tervezését és üzemeltetését.

- Traffic (http://www.erlang-software.com/):

  A Traffic szimulációs csomagot kifejezetten a távközlő hálózatok forgalmi viszonyainak elemzésére fejlesztették ki. Lehetőséget nyújt olyan bonyolult rendszerek elemzésére, amelyeket a hagyományos Erlang-formulák segítségével nem lehet vizsgálni.

27.6. Távközlési algoritmusok

A távközlésben tipikusan sok felhasználó, véletlenszerűen, többe-kevésbé egymástól függetlenül szeretne üzeneteket továbbítani egy közös erőforráson, a távközlő hálózaton. Ennek a feladatnak különböző további követelményeket kielégítő megoldására a távközlő algoritmusoknak egy nagyon szerteágazó széles körét dolgozták ki. Az újabb és újabb megoldások kialakításában szerepet játszottak a folyamatosan változó távközlési igények és távközlési technológiák. Az előbbit a következő alfejezet tárgyalja. A távközlési technológiák fejlődését a nagy sávszélességű digitális (például optikai) adatátvitel elterjedése, és a csomópontokban sávszélesség növekedés ellenére is növekvő komplexitású kiszolgálási funkciók jellemzik.

A fejlődés során néhány megoldási módszer elhalt (például distributed queue dual bus - DQDB), míg mások folyamatosan továbbfejlődnek (például internet protocol - IP), amit sok esetben nem a műszaki megoldás minősége, hanem a megoldás mögött felsorakozó távközlési cégek ereje befolyásolt.

Az alábbiakban egy rövid bevezető után a teljesség igénye nélkül ismertetünk néhány fontosabb kiszolgálási elvet, és azok néhány tipikus megvalósítását. A bemutatásra kerülő megoldásokat minden esetben csak a sorbanállási tulajdonságok szempontjából vizsgáljuk. A távközlési protokollok hierarchiájának további elemeit és megoldásait itt nem tárgyaljuk.

27.7. Távközlési igények változása

A kezdetben szinte kizárólag emberek közti beszédátviteli igények a vezetékes és vezeték nélküli számítógépes kommunikáció fejlődésével eltolódtak az esetenként nagy adatmennyiségek gyors átvitelét igénylő számítógépek közti adatcsere irányába, ami nagyon megváltoztatta az igények tulajdonságait és az elvárások jellegét.

A beszédátvitel főbb jellemzői a következők:

1. jól meghatározott kapcsolati időszak (beszélgetés eleje és vége között),

2. a kapcsolat idején jól meghatározott, a rendelkezésre álló összes sávszélességhez képest aránylag alacsony sávszélesség igény (beszéd ideje alatt az alkalmazott beszéd tömörítéstől függően például < 12 kbs, beszédszünet alatt 0),

3. alacsony átviteli idő elvárás (például < 100 ms),

4. néhány beszédcsomag elvesztése csak minőség romlást okoz, nem igényli az egész kommunikáció megismétlését.

Ezzel szemben a számítógépes kommunikációban

1. állandóan hálózathoz kötött számítógépek esetén nem azonosítható egyértelmű kapcsolati időszak,

2. az igényelt sávszélesség néhány csomag érdemi késleltetés korlát nélküli átvitelétől a hálózat szűk keresztmetszeteinek kapacitásával összemérhető sávszélességig terjedhet,

3. az adatátvitel lehet a késleltetés idejére érzékeny, a késleltetés ingadozásra érzékeny, illetve a késleltetésre érzéketlen,

4. megkülönböztetünk adatvesztésre érzékeny és kevésbé érzéketlen igényeket.

Az első szempont alapján a hagyományos ún. vonalkapcsolt megoldások helyett tért nyertek a csomagkapcsolt megoldások. A tisztán vonalkapcsolt esetben a távközlési erőforrások igényelt részét lefoglaljuk az adott kapcsolat idejére. Míg a tisztán csomagkapcsolt esetben az átviendő adatokat apró darabokban, ún. csomagokban továbbítjuk előzetes erőforrás foglalás nélkül.

Az igények relatív nagysága az erőforrásokhoz képest abból a szempontból meghatározó, hogy mekkora relatív ingadozású az igényfolyamat. Amennyiben a felhasználók, az erőforrásoknak csak egy kis részét igénylik, akkor sok felhasználó együttes viselkedése határozza meg az igényfolyamat relatív ingadozását, ellenben ha kevés igényforrás is túlterhelheti a rendszert, akkor az igényfolyamat relatív ingadozása lényegesen nagyobb.

A késleltetésre érzékeny igények az ún. valós idejű, vagy párbeszédes igények, mint a telefon, vagy a video konferencia. A késleltetés ingadozásra érzékeny szolgáltatás az audio vagy video folyam átvitele, mert a vevő oldali egyidejű lejátszáskor a lejátszás késletetése nem érzékelhető, viszont a csomagok egymáshoz képesti késése azt eredményezheti, hogy egy csomag még

nem érkezik meg, amikor a lejátszására kerülne a sor.

Az adatvesztést elvileg minden átvitel során kerülni kell, de amíg az audio vagy video átvitelben a csomagvesztés csak átmeneti minőségromlást okoz, addig például egy fájl átvitel során minimum az elveszett csomag sikeres átvitelig tartó ismétlését igényli.

A felsorolt jellemzőket (késleltetés, késleltetés ingadozás, csomagvesztés) gyűjtőnéven minőségi jellemzőknek nevezik (quality of service, QoS).

27.8. Igények változásának következményei

Az erőforrásokhoz mérten kis sávszélességű vonalkapcsolt igényeket kiszolgáló hálózatokban egy lehetséges kiszolgálási megoldás az erőforrások felosztása, és az aktív kapcsolatokhoz rendelése a kapcsolat idejére. Ez az eljárás a kapcsolat felépítésének pillanatában megvizsgálja, hogy rendelkezésre áll-e az igényelt erőforrás, és ha igen, akkor lefoglalja, és az adott kapcsolathoz rendeli, ha nem, akkor elutasítja az igényt.

Számítógép hálózatokban is alkalmaznak ehhez hasonló elven működő erőforrás megosztási eljárást. Abban az esetben, ha az átviendő adatok adott csomópontpárok között, hosszabb ideig, adott statisztikus jellemzőkkel rendelkeznek, akkor egy ún. hívás engedélyezési eljárás (Call admission control CAC) keretében eldöntjük, hogy az új igény kiszolgálására rendelkezésre áll-e a szükséges erőforrás, és ez alapján a hívást vagy elfogadjuk, vagy elutasítjuk.

Sok, hálózathoz kötött, időnként rövid időre aktívvá váló számítógép esetében azonban nem alkalmazható hatékonyan az erőforrás foglalás, mivel ebben az esetben a kapcsolat felépítéshez szükséges többlet tevékenység már összemérhető az alaptevékenységgel, az adatátvitellel. Ilyen forgalom források esetén kapcsolat felépítési fázis nélkül küldhetnek csomagokat a számítógépek, és a forgalom hálózatba lépési pontjában csomagfolyamot korlátozó eljárásokkal befolyásolható az erőforrások megfelelő megosztása. Ilyen forgalom korlátozó eljárások a leaky bucket és a GCRA.

A forgalmi igények különböző késleltetés és adatvesztés érzékenysége motiválta a forgalmi osztályok bevezetését, ugyanis más típusú kiszolgálás hatékony késleltetésre érzékeny és csomagvesztésre érzékeny forgalmak esetén. Ennek megfelelően adott forgalmi viszonyok esetén késleltetés érzékeny forgalom átvitelére kisebb puffer méret, míg csomagvesztés érzékeny forgalom átvitelére nagyobb pufferméret alkalmas.

Abban az esetben viszont, amikor egy közös erőforráson kell késleltetés- és csomagvesztésre érzékeny forgalmi összetevőket tartalmazó forgalmat átvinni, akkor megfelelő puffer méretezéssel nem biztosítható minden minőségi igény kielégítése. Ekkor a legegyszerűbb eljárás az átviteli kapacitás növelése addig,

amíg adott puffer méret mellett mind a késleltetés- mind a csomagvesztésre érzékeny forgalom minőségi jellemzői elérik a megkövetelt szintet.

A kiszolgálási kapacitás növelése azonban a kapacitás felhasználás szempontjából nem hatékony eljárás. Az így kialakított rendszerek aránylag egyszerűek, de alacsony hatásfokúak.

Adott minőségi paraméterek kisebb kapacitás mellett is biztosíthatók, ha a hálózat csúcspontjaiban forgalom osztály specifikus megkülönböztetést és kiszolgálást végzünk. Ez az eljárás bonyolultabb funkciók megvalósítását igényli a csomópontokban, viszont javítja a rendszer kihasználtságát.

A fent említett forgalom szabályozó és a szolgáltatás megkülönböztetést végző eljárásokkal foglalkozunk a következő fejezetben.

27.9. Forgalom szabályozó eljárások

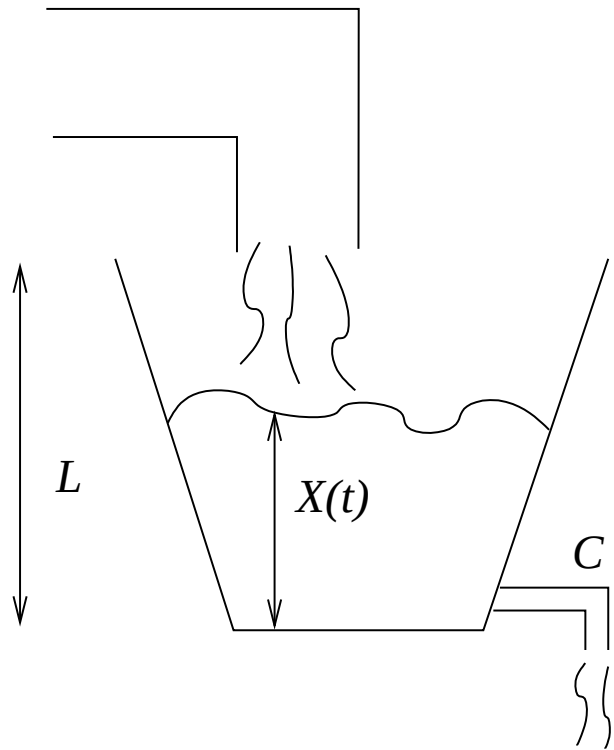
A számítógépes kommunikáció esetén a forgalom források erősen ingadozó viselkedése és nagy adatátviteli kapacitása miatt a hálózattal szemben támasztott átviteli igények erősen ingadozhatnak. Az ilyen jelentős ingadozások megnehezítik a csomagok adott minőségi elvárásoknak megfelelő átvitelét. Az ingadozások csökkentésének egyik lehetséges módja a forgalom források korlátozása, hogy csak az előre megállapított adatsebességgel küldhessenek csomagokat a hálózatba. Ezt a korlátozást valósítják meg a hálózat bemenő pontjain alkalmazott forgalom szabályozó eljárások.

Az alábbiakban az eljárások tárgyalása során két modellezési szintet különböztetünk meg. Az első az elvi modell szintje, ahol csak az átviendő adatmennyiséget tekintjük, és nem vesszük figyelembe, hogy csomagokban történik az adatok továbbítása. A második szinten az elvi modell egy valós, csomagtovábbítást végző hálózatban is alkalmazható változatát ismertetjük.

27.9.1. Lyukas vödör eljárás

Az elvi modell szintjén az átviendő adatmennyiséget folytonosnak tekintjük, fizikai analógia alapján például folyadéknak. Ebben a modellben a forgalom szabályozás feladata a hálózatba folyó folyadék ingadozását adott szabályok szerint csökkenteni (lásd 27.7. ábra).

Az eljárás lényegét jól szemlélteti az eljárás neve. A bejövő folyadékot egy L űrtartalmú vödörbe vezetjük, amelyikből maximálisan C sebességgel tud kifolyni a folyadék. Ha a folyadék beáramlási sebessége (R_{IN}) egy intervallumon meghaladja I értékét, akkor a vödörben a folyadékszint (X) emelkedik. Amikor a folyadékszint eléri L-t, akkor a többlet folyadék elvész. A folyadék kiáramlási sebessége mindig C, amíg a vödör nem üres, és amikor a vödör



27.7. ábra. Lyukas vödör eljárás.

üres, akkor a kiáramlás (R_{OUT}) és a beáramlás sebessége megegyezik.

$$\begin{aligned}
&\frac{dX(t)}{dt} = R_{IN} - C, R_{OUT} = C, && \text{ha} \quad 0 < X(t) < L \ , \\
&\frac{dX(t)}{dt} = 0, R_{OUT} = C, && \text{ha} \quad X(t) = L, R_{IN} > C \ , \\
&\frac{dX(t)}{dt} = 0, R_{OUT} = R_{IN}, && \text{ha} \quad X(t) = 0, R_{IN} < C \ .
\end{aligned} \tag{27.1}$$

27.9.2. Lyukas vödör eljárás csomagtovábbítás esetén

A lyukas vödör eljárás folytonos folyadék beáramlást feltételez. Csomagtovábbító távközlési rendszerek elemzése során azonban az adatok érkezését sok esetben pillanatszerűnek tekintik, és a csomag első vagy utolsó adategységének érkezéséhez rendelik. Ezt a modellt például az a tulajdonság indokolja, hogy a csomagok tetszőleges részének elvesztése esetén a csomag

egészét eldobják a távközlő rendszerek, így csomagokra vonatkozóan kell meghatározni a továbbítás módját. Az ismertetett lyukas vödör modell ezt a tulajdonságot nem tükrözi. A fix méretű csomagok átvitelének leírására ezért a modellnek egy módosított változatát használják.

A csomagforgalom lyukas vödör modelljében a csomagok érkezési pillanataiban (t) vizsgáljuk a vödörben a folyadék feltételezett szintjét ($X_a = X - (t - s)$) az előző sikeres csomagtovábbítás folyadékszintje (X), az azóta eltelt idő ($t - s$), és a megengedett adattovábbítási sebesség (C) alapján. Amennyiben a beérkező csomag még nem tölti tele a vödröt, akkor az a csomag sikeresen továbbítható, viszont ha az érkezés hatására kicsordulna a vödörből a folyadék, akkor eldobjuk a csomagot. Az egyszerűség kedvéért az eljárásban az adatsebesség (C) és a csomagméret (d) helyett a csomagok engedélyezett érkezési időközével (I) számolunk, ahol $I = d/C$.

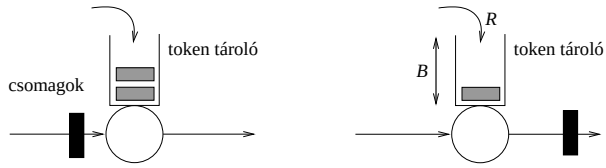
LYUKAS-VÖDÖR(X, t, s, I, L)

```
1  X_a = X − (t − s)
2  if X_a < 0
3     X = I
4     s = t
5     return IGAZ
6     else if X_a > L
7              return HAMIS
8              else X = X_a + I
9              s = t
10             return IGAZ
```

27.9.3. Tokentárolós csomagtovábbítási eljárás

Az előző eljárásokban a vödör mérete jellemezte a megengedettnél gyorsabban átvihető adatmennyiséget. Ha a vödör folyadékszintje helyett a vödörbe még betölthető folyadék ($L - X$) egész csomagméret kerekített értékével jellemezzük a rendszert, akkor egy más szemléletű modellel valósíthatunk meg a fentiekhez hasonló forgalom szabályozó eljárást.

Ebben a modellben a csomagok csak tokenekkel (vagy zsetonokkal, ami egy-egy csomag átvitelét engedélyezi) együtt továbbíthatók. A 27.8. ábrán a feketével jelzett csomag továbbításakor egy szürke token is távozik a rendszerből. A tokenek állandó sebességgel keletkeznek. I időnként keletkezik egy token, amit általában az $R = 1/I$ token keletkezési rátával jellemeznek. A tokentároló méretét a tokenek maximális számával jellemzik $B = L/d$. Ha a tokentároló megtelik, a keletkező újabb tokenek elvesznek. Ha egy csomag



27.8. ábra. Tokentárolós eljárás.

érkezésekor a tokentároló üres, akkor a csomag elvész.

27.9.4. GCRA eljárás

Az ATM hálózatok tipikus forgalom szabályozó eljárásaként elterjedt GCRA algoritmus a lyukas vödör eljárás csomagszintű érkezéseket modellező változatának egy hatékonyan implementálható módosítása, amelyben az elméleti érkezési idő (TAT - **T**heoretical **A**rrival **T**ime) helyettesíti a folyadék szintet.

GCRA(TAT, t, I, L)

```
1  if TAT < t
2     TAT = t + I
3     return IGAZ
4     else if TAT > t + L
5             return HAMIS
6             else X = X_a + I
7             return IGAZ
```

A felsorolt forgalomszabályozó eljárások mindegyikét közvetve, vagy közvetlenül két paraméter jellemzi a gyorsan továbbítható adatmennyiség (vödör méret) L és a megengedett átlagos adattovábbítási sebesség C. ATM terminológiával az mondják, hogy az ilyen forgalomszabályzón átvezetett forgalom átlagos sebessége nem nagyobb, mint C és maximális börszt mérete L.

Ezeknek a forgalomszabályozó eljárásoknak közös jellemzője, hogy késleltetést nem okoznak, viszont a csomagok egy részét eldobják. E rendszerek tipikus analízis kérdése, hogy adott tulajdonságú forgalomforrás esetén mekkora a csomag eldobási valószínűség.

27.10. Forgalom megkülönböztetést végző kiszolgálási eljárások

A távközlő hálózatok csomópontjaiban különböző felhasználóktól, különböző irányokból érkező és különböző minőségi elvárásokkal rendelkező csomagokat kell továbbítani adott kimenő irányokba. Nem szinkronizált érkezési folyamatok esetén ez a feladat még akkor is igényelhet konfliktus feloldást, ha a kimenő kapacitás meghaladja a beérkező csomagfolyamok maximális intenzitásainak összegét, ugyanis ha két csomag érkezése közt az idő kisebb, mint az előbb érkezett csomag továbbításának ideje és mindkét csomag továbbítására azonos erőforrást használunk, akkor a második csomag továbbításával várni kell az első továbbításának befejezéséig.

A konfliktusok feloldására több módszert is alkalmaznak távközlő hálózatokban:

1. erőforrás megosztás (fix vagy dinamikus),
2. közös erőforrás konfliktus feloldó algoritmussal,
3. közös erőforrás pufferrel.

Az erőforrás megosztás célja a versenyhelyzet megszüntetése. Az aktív forgalomforrások mindegyikének biztosítunk egy kizárólag számára elérhető erőforrás részt. Attól függően, hogy ezt az erőforrás felosztást milyen gyakran módosítjuk, beszélhetünk statikus vagy dinamikus erőforrás megosztásról. Dinamikus erőforrás megosztás esetén gyakori, hogy az erőforrás egy részét fent tartják új igények bejelentésére, ahol az igénybejelentő csomagok versenyeznek egymással, és egy központi vezérlő a beérkezett igények alapján végzi az erőforrás megosztást.

Közös erőforrás véletlenszerű használata esetén az igényforrások a véletlen időpillanatokban keletkező csomagjaikat a keletkezés pillanatában próbálják továbbítani (feltételezve, hogy a fennálló elvárásoknak megfelel a csomagkeletkezési folyamat). Ebben az esetben a hálózat véges kapacitásai miatt konfliktusok keletkezhetnek (például két forrás egyszerre szeretne csomagot küldeni). A konfliktusfeloldás két lehetséges módszere a konfliktusba került csomagok eldobása, vagy a hálózat csomópontjaiban lévő tárolókban várakoztatása. A csomagok eldobása esetén a forgalomforrás értesül a csomagtovábbítás sikertelenségéről, és egy egyszerű algoritmus alapján idővel megpróbálja ismételten elküldeni csomagját. Ennek az algoritmusnak biztosítania kell, hogy a konfliktusok idővel feloldódjanak, és idővel minden csomag továbbításra kerüljön.

A felsorolt eljárások közül sok esetben itt nem tárgyalt mérnöki szempontok alapján kell választani, de amennyiben forgalmi szempontok dominálnak, akkor a következő tényezőket mérlegelhetjük:

- csomagtovábbítási folyamat függése/függetlensége más források viselkedésétől,
- műszaki és algoritmikus megvalósítás bonyolultsága,
- késleltetés, és ha létezik, akkor a csomagvesztés valószínűsége.

A felsorolt tényezők szerint az konfliktusfeloldási eljárások jellemzői a következők:

1. erőforrás megosztás (fix vagy dinamikus):
   előny: nem zavarják egymás kiszolgálási jellemzőit, általában egyszerű megvalósítani (idő-, frekvencia-, kódosztás),
   hátrány: nem állandó intenzitású forgalomforrások esetén nem hatékony, lassan és nem tetszőlegesen változtatható a megosztási arány, külön protokoll szükséges a megosztás karbantartására (igény bejelentés, döntés, megosztás, igény megszűnése),
   példa: GSM telefon és bázis állomás közti kapcsolat (idő- és frekvenciaosztás);

2. közös erőforrás konfliktus feloldó algoritmussal;
   előny: igény szerinti erőforrás felhasználás, a hálózat viselkedése egyszerű (ütközés $\rightarrow$ csomag eldobás),
   hátrány: az elosztott konfliktus feloldás nem hatékony, az igényfolyamatok befolyásolják egymás kiszolgálását,
   példa: ALOHA, CSMA (Ethernet), IEEE 802.11 (wireless LAN)

3. közös erőforrás pufferrel:
   előny: igény szerinti erőforrás felhasználás, hatékony központi konfliktus feloldás
   hátrány: központi puffer és kiszolgálás vezérlést igényel, igényfolyamatok befolyásolják egymás kiszolgálását.
   példa: FIFO, Prioritás, WFQ.

Az erőforrás megosztás algoritmusai általában a kapcsolat érkezési és végződési folyamat követését végzik, ezekre nem térünk ki. A következő fejezetben tárgyaljuk a konfliktus feloldó algoritmusokat, és az azt követő fejezetet szenteljük a pufferes konfliktus feloldás vagy más néven sorbanállási és kiszolgálási rendszerek vizsgálatának.

27.11. Véletlen erőforrás hozzáférés konfliktus feloldó algoritmusai

Véletlen erőforrás hozzáférés esetén több, egymás viselkedését közvetlenül nem ismerő felhasználó próbál adatcsomagokat továbbítani egy közösen használt átviteli közegen. A felhasználók adattovábbítását olyan algoritmus szerint végzik, amelyik a többi felhasználó viselkedéséről valamilyen módon rendelkezésre álló információk alapján biztosítja, hogy

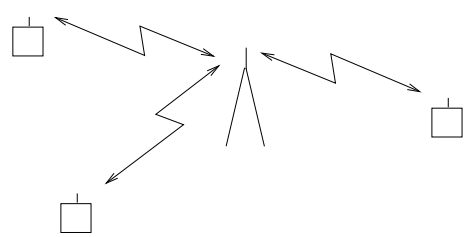
- egy rendszerfüggő terhelési szint alatt ne akadjon el az adattovábbítás,
- minden átküldésre szánt csomag idővel átjusson a rendszeren,
- amennyiben nem törekszünk a felhasználók megkülönböztetésére, akkor a rendszer igazságos minden felhasználóhoz, azaz egyforma esélyeket biztosít számukra adatcsomagjaik átvitelére.

Az alábbiakban ismertetett eljárások egy algoritmus család különböző változatai, amelyek abban különböznek egymástól, hogy

- a felhasználók milyen módon értesülnek a többi felhasználó viselkedéséről,
- milyen tervezési szempontokat érvényesítenek a forgalom ingadozása esetén.

27.11.1. ALOHA eljárás

Az algoritmus család legegyszerűbb tagját rádió terminálok és egy központ közti kommunikáció céljára fejlesztették ki (lásd a 27.9. ábrát). A rendszer két frekvenciasávot használ. Az egyiken a terminálok küldhetnek adatot a központ felé, a másikon a központ az összes terminálnak. (Az ALOHA eljárást Hawai-on fejlesztették ki, maga a név egy helyi üdvözlésből ered.)



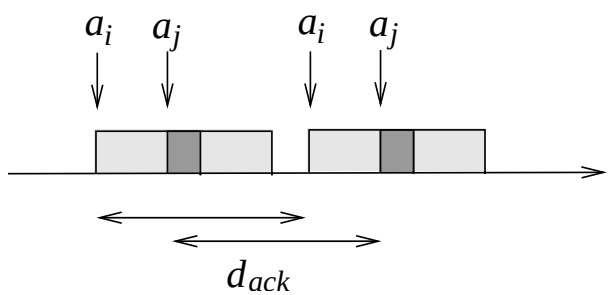
27.9. ábra. Rádió terminál rendszer.

Ha az első frekvenciasávban egyszerre több terminál szeretne adatot továbbítani, akkor az azonos frekvenciájú rádiójelek összegződése miatt a központ nem tudja az egyik terminál adatát sem fogni. Ekkor azt mondjuk, hogy a terminálok csomagjai ütköztek. Mivel a második frekvencia sávban

csak a központ sugározhat jeleket, így ott nem alakulhat ki ütközés. A sikeresen vett jeleket a központ egy következő üzenetében nyugtázza. A terminálok az ütközés bekövetkezéséről abból értesülnek, hogy az általuk elküldött csomagot a központ egy adott határidőn belül nem nyugtázza. Az ALOHA eljárás ebben a rendszerben biztosítja a felsorolt szempontoknak megfelelő adat kommunikációt.

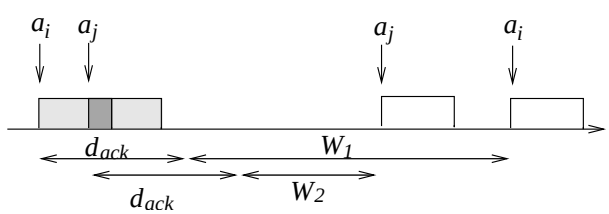
Az eljárás működése nagyon egyszerű. Amint egy terminálnak keletkezik egy átviendő adatcsomagja, azonnal elküldi. Ha a rendszerre jellemző határidőig nem érkezik nyugta a csomag sikeres átviteléről, akkor a terminál azt feltételezi, hogy a csomag ütközött. Azt mondjuk, hogy ekkor a terminál blokkolt állapotba kerül. Ebben az esetben addig ismétli a csomag továbbítását, amíg a sikeres átviteléről nyugtát nem kap (lásd a 27.10. ábrát).

A blokkolt felhasználók a csomagok ismétlését nyilván nem kezdhetik azonnal a határidő letelte után, hiszen ekkor már két ütköző felhasználó is teljesen megakaszthatná a sikeres adat továbbítást, mivel minden csomagismétlési kísérlet alkalmával újra ütköznének a csomagok.



27.10. ábra. Csomagismétlés várakozás nélkül.

Az ALOHA eljárásban ezért a felhasználók egymástól függetlenül egy véletlen késleltetési időt sorsolnak, és ennek a véletlen időnek a leteltével kísérlik meg a csomagjuk ismételt elküldését (lásd a 27.11. ábrát). Az ábrákon megkülönböztetjük a valódi ütközési időszakot (sötét szürke), és az ütközés miatt feleslegesen használt időszakot (világos szürke).



27.11. ábra. Csomagismétlés véletlen várakozással.

A rendszer minőségi viselkedése eléggé szemléletes. Amennyiben a fel-

használók „nagy” késleltetési időket sorsolnak, akkor kicsi lesz annak az esélye, hogy a kisorsolt késleltetési idők megegyeznek, és a késleltetés elteltével újra ütköznek a csomagok (akár egy, akár több csomag ütközött eredetileg). Ebben az esetben tehát „kevesebb” újraküldési kísérlet szükséges a csomagok átviteléhez, viszont az újraküldési kísérletek közti késleltetési idő „nagy”. A másik szélsőséges viselkedés, amikor a felhasználók „kis” késleltetési időket sorsolnak. Ekkor az ismétlési kísérletek közti idő alacsony, de nagyobb valószínűséggel ütköznek a csomagok, és ezért általában több ismétlési időszakra van szükség. A rendszer hatékony munkapontja e két szélsőség között keresendő.

Az ALOHA rendszerek modellezése és teljesítmény analízise az 1950-es évektől egy erősen kutatott terület, aminek mindig újabb aktualitást adott az egyre újabb ALOHA eljárás változatok bevezetése (lásd a következő alpontokat).

A kiterjedt elemzések különböző felhasználói viselkedést és modellezési megfontolásokat feltételeznek. Általában elmondhatjuk, hogy a valóságos rendszerek tényleges működésének sajátosságait figyelembe vevő analitikus leírás szinte reménytelen. A kidolgozott analitikus modellek egyszerűsítő modellezési megfontolásokat tartalmaznak, amelyek azonban sok esetben nagyon pontosan közelítik a valós rendszer működését.

Az eredeti ALOHA rendszerrel kapcsolatban néhány egyszerűen elemezhető teljesítmény jellemzőt tárgyalunk az alábbiakban.

Folytonos idejű ALOHA rendszer

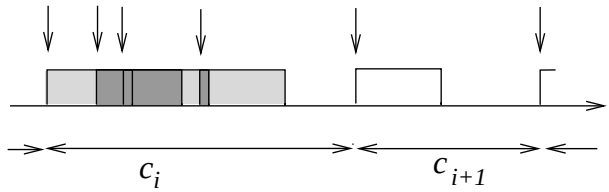
Modellezési megfontolások:

- Az új és az ismételt csomagok együttesen λ paraméterű Poisson-folyamat szerint érkeznek.

  Ez a modellezési megfontolás természetesen (mesterséges esetektől eltekintve) nem teljesül a leírt viselkedés mellett, azonban bizonyos feltételek mellett (például a blokkolt felhasználók nem generálnak lényegesen nagyobb forgalmat, mint a nem blokkoltak, a csomagtovábbítási idő lényegesen kisebb, mint az ismétlési idő) jól közelíti a rendszer valódi viselkedését. Ez a modell annyira elterjedt az ALOHA rendszerek elemzésében, hogy „0-adrendű modell”-ként említik.

- A csomagok fix méretűek, az adott átviteli sebesség mellett átviteli idejük T.

A 0-adrendű modellben azt vizsgáljuk tehát, hogy a λ paraméterű Poisson-folyamat szerint érkező csomagok közül mennyit továbbít sikeresen a rendszer, és hogy ebből mekkora csomag ismétlési valószínűség és késleltetési idő adódik.



27.12. ábra. Üres időszakokkal határolt működési ciklusok.

A csatorna működését a 27.12. ábrának megfelelően felosztjuk üres időszakok közti működési ciklusokra. Annak valószínűsége, hogy egy működési ciklusban egy csomag átvitele sikeres, megegyezik azzal, hogy a ciklus megkezdése utáni első csomagküldés T-nél később kezdődik, ami $e^{-\lambda T}$.

Annak érdekében, hogy hosszú idő átlagában meghatározzuk a rendszer sikeres, sikertelen és üres időszakainak arányát, meghatározzuk ezen időszakok várható hosszát egy működési ciklus alatt. A üres időszak hossza λ paraméterű exponenciális eloszlású, $1/\lambda$ várható értékkel. A sikeres időszak hossza pontosan T. Egy sikertelen időszak $N-1$ ($N \geq 2$) darab T-nél rövidebb érkezési időközből, és egy lezáró T hosszú időszakból áll. Az $N = 1$ a sikeres csomagtovábbítás esete. A Poisson érkezési folyamat emlékezetmentessége miatt az intervallumok hosszától függetlenül számíthatjuk a sikertelen időszakban ütköző csomagok számát,

$$P(N = n) = (1 - e^{-\lambda T})^{n-1} e^{-\lambda T} \ .$$

A T-nél rövidebb időszak (U) hosszának eloszlásfüggvénye

$$F_U(t) = P(U < t) = P(\tau < t | \tau < T) = \begin{cases} \dfrac{1 - e^{\lambda t}}{1 - e^{\lambda T}} & 0 < t < T \ , \\ 1 & T < t \ , \end{cases}$$

amiből $E(U) = \dfrac{1 - e^{-\lambda T} - \lambda T e^{-\lambda T}}{\lambda(1 - e^{-\lambda T})}$. Mindezek alapján egy működési ciklusban

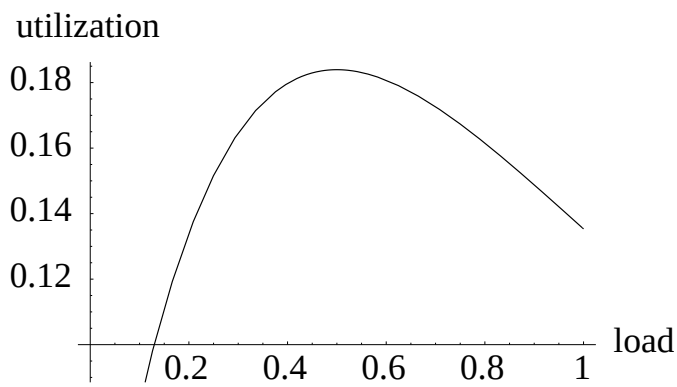
- az üres intervallum várható hossza $E(I) = 1/\lambda$,
- a sikeres csomagtovábbítás várható ideje $E(S) = e^{-\lambda T} T$,
- és a sikertelen csomagtovábbítás várható ideje

$$E(L) = \sum_{n=2}^{\infty} P(N = n) \Big((n-1)E(U) + T \Big) = \frac{1 - e^{-\lambda T} - \lambda T e^{-\lambda T}}{\lambda e^{-\lambda T}} \ .$$

A rendszer kihasználtságát a sikeres csomagtovábbítás időaránya jellemzi

$$\rho = \frac{E(S)}{E(I) + E(S) + E(L)} = \lambda T e^{-2\lambda T} \ .$$

A kihasználtság maximuma $\lambda T = 0.5$ mellett $\rho = 1/2e \sim 0,18394$. Látható, hogy a 0.5-nél nagyobb terhelés esetén a kihasználtság csökken, így ezeknél a rendszereknél ügyelni kell arra, hogy a felhasználók eredő forgalma ne emelkedjen e fölé a szint fölé.



27.13. ábra. Kihasználtság ρ a terhelés λT függvényében.

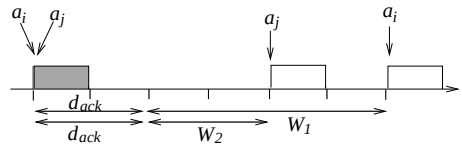
Egy Δ hosszú intervallumban az érkező csomagok várható száma $\lambda\Delta$. Ugyanezen idő alatt a sikeres csomagtovábbítás várható ideje $\rho\Delta$, amennyi idő alatt $\lfloor \rho\Delta/T \rfloor$ csomagot lehet átvinni. Ezek alapján $\Delta \to \infty$ esetén az egy sikeresen átvitt csomagra jutó csomagtovábbítási kísérletek száma, azaz a csomagküldési kísérletek várható száma $E(R) = \lambda T/\rho = e^{2\lambda T}$.

A csomagküldési kísérletek száma (R) alapján a csomagkésleltetési idő a következőképpen adódik

$$\begin{aligned} E(D) &= E\left(\sum_{r=1}^{\infty} P(R=r) \left(rT + \sum_{i=1}^{r-1} d_{ack} + W_i \right) \right) \\ &= E(R)T + (E(R)-1)(d_{ack} + E(W)) \ . \end{aligned}$$

Diszkrét idejű ALOHA rendszer

A folytonos idejű rendszer egyik legnagyobb hátránya, hogy a 27.10.–27.12. ábrákon sötét szürke színnel jelöl valódi ütközéseken túl a rendszer számára



27.14. ábra. Diszkrét idejű ALOHA rendszer.

elveszett idő az ábrákon világos szürkével jelölt időszak is, amelyek alatt a felhasználók az egyébként ütközés miatt elvesző csomagjaikkal csökkentik a további csomagok sikeres továbbításának valószínűségét.

Az ALOHA eljárás egy egyszerű módosításával megszüntethető ez a veszteség. Abban az esetben, ha minden felhasználó csak szinkronizált időrésekben küld csomagot, akkor az ütköző csomagok legfeljebb T ideig foglalják a csatornát. Ebben az esetben az ütközés esetén sorsolt véletlen késleltetés T egész számú többszöröse. Ezt a rendszert szokták réselt ALOHA rendszernek is nevezni.

A diszkrét ALOHA rendszer (lásd a 27.14. ábrát) „0-adrendű modellje" azt feltételezi, hogy a felhasználók λT paraméterű Poisson-eloszlású csomagot (új és ismételt együttesen) probálnak elküldeni minden időrésben. Ez a modell megfelel a folytonos idejű ALOHA rendszer 0-adrendű modelljének, azzal a kiegészítéssel, hogy a felhasználók egy T idejű időrésben Poisson érkezési folyamat szerint keletkező csomagjaikat a következő időrésben küldik el. E feltételezés mellett egyetlen időrés vizsgálata alapján meghatározhatók a lényeges teljesítmény jellemzők. Jelölje N az időrésben érkező csomagok számát.

$$\rho = P(\text{sikeres csomagátvitel}) = P(N = 1) = \lambda T e^{-\lambda T}$$

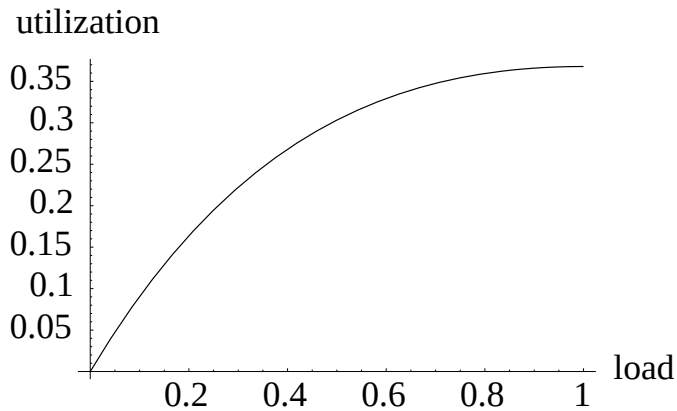
A kihasználtság maximuma $\lambda T = 1$ mellett $\rho = 1/e \sim 0,367879$. A folytonos idejű rendszerrel szemben itt a terhelés a rendszer kapacitásának szintjéig ($\lambda T = 1$) növelhető a kihasználtság növekedése mellett.

A csomag ismétlések (R) átlagos számát az időrésben sikeresen átvitt csomagok és az összes csomag aránya adja

$$E(R) = \frac{E(N)}{E(\text{sikeres csomagok})} = \frac{\lambda T}{\lambda T e^{-\lambda T}} = e^{\lambda T} \ .$$

A késleltetési idő pedig a folytonos idejű ALOHA rendszerhez hasonlóan

$$\begin{aligned} E(D) &= E\left(\sum_{r=1}^{\infty} P(R = r) \left(rT + \sum_{i=1}^{r-1} d_{ack} + W_i \right) \right) \\ &= E(R)T + (E(R) - 1)(d_{ack} + E(W)) \ . \end{aligned}$$



27.15. ábra. Diszkrét idejű ALOHA rendszer kihasználtsága.

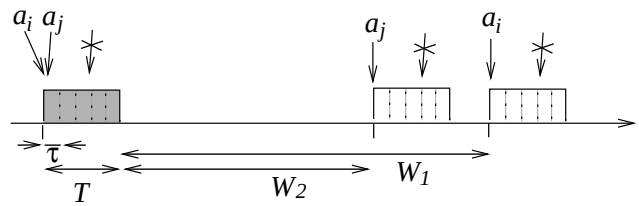
Az ALOHA rendszerek pontosabb modelljei megkülönböztetik a felhasználók bizonyos állapotait (csomag előkészítés, új csomag küldés, várakozás, késleltetés, csomag ismétlés) és ez alapján határozzák meg az érkező új és ismételt csomagok számát [59].

27.11.2. CSMA és CSMA/CD

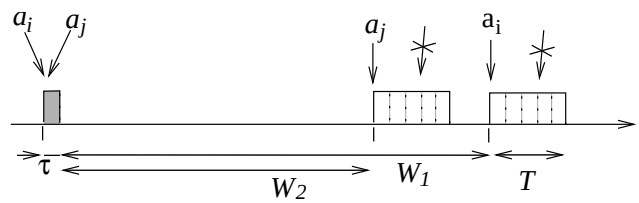
Az ALOHA protokoll család további tagjai az adott fizikai rendszerben a felhasználók számára elérhető információk segítségével javítják a csatorna kihasználtságát.

A folytonos és a diszkrét idejű ALOHA rendszer vizsgálata során már láttuk, hogy javítható a kihasználtság, ha az ütköző csomagok kevesebb ideig foglalják a csatornát. Rádiós rendszerekben nem mindig biztosított, hogy minden felhasználó minden más felhasználó adását venni tudja. Vezetékes rendszerekben viszont a terjedési idő leteltével minden felhasználó értesülhet arról, hogy valaki csomagot továbbít a csatornán. Így egyrészt elkerülhető az, hogy egy felhasználó csomagtovábbításáról értesülve egy másik felhasználó is adni kezdjen (**C**arrier **S**ense **M**ultiple **A**ccess, CSMA), másrészt, ha a terjedési idő kisebb, mint a csomagtovábbítási idő, akkor az esetleges ütközések még a csomagküldés alatt kiderülnek, ha a felhasználók adás alatt is vizsgálják a csatorna állapotát (with **C**ollision **D**etection, CD). A 27.16. és 27.17. ábrák a rendszer diszkrét idejű változatát szemléltetik, ahol a maximális jel terjedési idő τ, a csomagküldések szinkronizált τ hosszú időrések elején kezdődhetnek, és csomagütközés csak az első τ időrésben lehetséges, mert a következő időrésben már minden felhasználó tudja, hogy csomagtovábbítás zajlik a csatornán.

A felhasználók csak akkor küldhetnek csomagot, ha úgy tapasztalják, hogy a csatorna üres, így csak azonos τ időrésben kezdődő csomagok ütközhetnek. Az eljárás ütközés detekció nélküli változatában a megkezdett csomagok küldése ütközés esetén is befejeződik (27.16. ábra), míg ütközés detekció esetén az ütköző csomagok továbbítása az ütközés felismerésekor azonnal leáll (27.17. ábra).



27.16. ábra. CSMA rendszer működése.



27.17. ábra. CSMA rendszer működése.

Diszkrét idejű CSMA rendszer

A rendszer „0-ad rendű modelljének” elemzését egymást követő csomagküldési kísérletek közti intervallumokat vizsgálva végezzük. A 27.16. és 27.17. ábrákon az intervallumok kezdetét az időtengely alatti vonalkák jelzik. Előfordulhat az is, hogy az ábrától eltérően nincs üres intervallum két egymást követő csomagtovábbítási kísérlet között. A „0-adrendű modell” esetünkben azt feltételezi, hogy minden üres vagy befejező intervallum utáni τ hosszú intervallumban $\tau\lambda$ paraméterű Poisson-eloszlású csomagot küldenek a felhasználók. Tehát, az eddigi 0-adrendű modellekkel szemben a csatorna állapota befolyásolja az érkezési folyamatot.

Az átvitel sikerességét a vizsgált intervallum első τ hosszú szeletében érkező csomagok száma (N) határozza meg.

$$P(\text{sikeres csomagátvitel}) = P(N = 1|N \geq 1) = \frac{\lambda\tau e^{-\lambda\tau}}{1 - e^{-\lambda\tau}} \ .$$

A sikeres, vagy ütköző csomagok küldése után a csatorna addig üres marad, amíg újabb csomagküldés nem kezdődik. Jelölje ezt az üres időt I,

ahol I ($I \in \{0, \tau, 2\tau, \ldots\}$) geometriai eloszlású, azaz

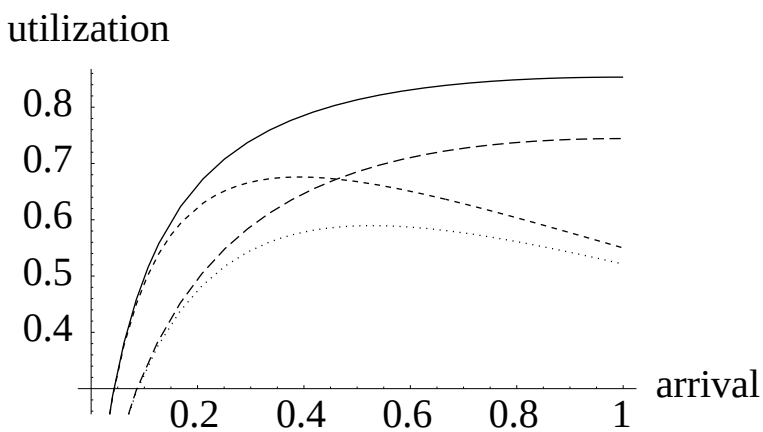
$$P(I = i\tau) = e^{-\lambda\tau i}(1 - e^{-\lambda\tau}) \,.$$

Így a vizsgált intervallumban

- az üres intervallum várható hossza $E(I) = \dfrac{\tau\ e^{-\lambda\tau}}{1 - e^{-\lambda\tau}}$,
- a sikeres csomagtovábbítás várható ideje $E(S) = \dfrac{T\ \lambda\tau e^{-\lambda\tau}}{1 - e^{-\lambda\tau}}$,
- és a sikertelen csomagtovábbítás várható ideje $E(L) = \dfrac{T\ (1 - (1 + \lambda\tau)e^{-\lambda\tau})}{1 - e^{-\lambda\tau}}$,

amiből a rendszer kihasználtsága

$$\rho = \frac{E(S)}{E(I) + E(S) + E(L)} = \frac{T\ \lambda\tau e^{-\lambda\tau}}{T(1 - e^{-\lambda\tau}) + \tau e^{-\lambda\tau}} \,.$$



27.18. ábra. Diszkrét idejű CSMA és CSMA/CDrendszer kihasználtsága.

A 27.18. ábra $\lambda\tau$ függvényében ábrázolja a kihasználtságot $\tau/T = 0.2$ (pontvonal), illetve $\tau/T = 0.1$ (rövid szaggatott vonal) esetén. Látható, hogy rövidebb terjedési idő esetén kisebb az ütközés esélye, és ezért nő a kihasználtság.

Diszkrét idejű CSMA/CD rendszer

Egy „0-ad rendű modellt” vizsgálva a diszkrét idejű CSMA/CD rendszer csak annyiban különbözik a diszkrét idejű CSMA rendszertől, hogy az ütközés érzékelése miatt a sikertelen csomagtovábbítás várható ideje lecsökken

$$E(L) = \frac{\tau\ (1 - (1 + \lambda\tau)e^{-\lambda\tau})}{1 - e^{-\lambda\tau}} \,.$$

Ennek hatására a rendszer kihasználtsága

$$\rho = \frac{E(S)}{E(I)+E(S)+E(L)} = \frac{T\ \lambda\tau e^{-\lambda\tau}}{T\lambda\tau e^{-\lambda\tau}+\tau(1-\lambda\tau e^{-\lambda\tau})}\ .$$

A 27.18. ábra a CSMA rendszer kihasználtságával együtt ábrázolja a CSMA/CD rendszer kihasználtságát $\lambda\tau$ függvényében $\tau/T = 0.2$ (hosszú szaggatott vonal), illetve $\tau/T = 0.1$ (folytonos vonal) esetén. A rövidebb terjedési idő ismét növeli a kihasználtságot, és a CSMA rendszerrel összehasonlítva azt látjuk, hogy ütközési időszak csökkentése is növeli a kihasználtságot nagyobb terhelések esetén.

Diszkrét kitartó CSMA/CD rendszer

A CSMA rendszerek eddigi vizsgálata során nem tértünk ki arra a kérdésre, hogy mit csinálnak azok a felhasználók, akiknek küldendő új vagy késleltetett csomag küldési szándéka van akkor, amikor a csatornát más felhasználó foglalja éppen. Az eddigiekben implicit módon azt feltételeztük, hogy ezek a felhasználók úgy járnak el, mintha csomagjuk ütközött volna, és egy véletlen késleltetési idő múlva próbálkoznak újra. A gyakorlatban ezt az esetet nemkitartó (non-persistent) viselkedésnek nevezik.

A felhasználók azonban a csomagméret ismeretében azt is tudják, hogy mikor fejeződik be az éppen zajló csomag továbbítása, mivel minden sikeres csomag T ideig foglalja a csatornát. Ilyenkor a véletlen késleltetési időhöz képest csökkenthető a csomagkésleltetés ideje, ha a csatorna szabaddá válása utáni időrésben próbál adni az a felhasználó, akinek a foglalt időszakban keletkezett átviendő csomagja. Ez a viselkedést nevezik kitartó viselkedésnek.

A kitartó CSMA rendszer „0-adrendű modelljében” azt feltételezzük, hogy minden τ hosszú időrésben $\tau\lambda$ paraméterű Poisson-eloszlású csomagot küldenének a felhasználók, és akiknek akkor keletkezik csomagja, amikor a csatorna foglalt, azok a csatorna szabaddá válását követő időrésben próbálják továbbítani a csomagot.

Ennek a rendszernek a vizsgálatát felbonthatjuk a sikeres (S), az ütköző (L), és az üres (I) szakaszok vizsgálatára, mivel ezeknek a szakaszoknak a kezdetén memóriamentes a rendszer. Az egyes szakaszok hossza egyszerűen számítható. A sikeres szakasz hossza $E(S) = T$, az ütköző szakasz hossza $E(L) = \tau$, és az üres szakasz várható hossza $E(I) = (1 - e^{-\lambda\tau})^{-1}$. A Π állapot átmeneti mátrix megadja, az egyes szakaszok milyen valószínűséggel

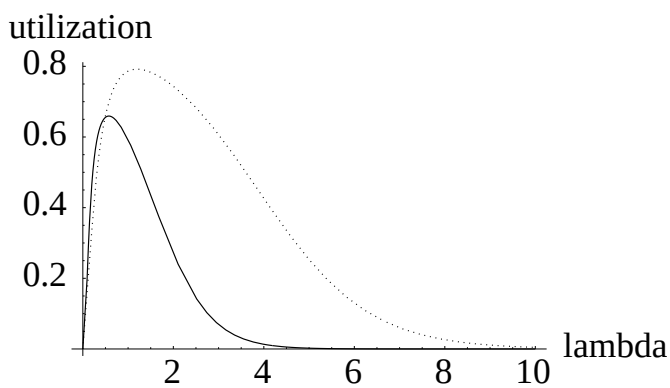
követik egymást.

$$\Pi = \begin{array}{|c|c|c|c|} \hline S & L & I & \\ \hline P(\lambda T,1) & P(\lambda T,>1) & P(\lambda T,0) & S \\ P(\lambda\tau,1) & P(\lambda\tau,>1) & P(\lambda\tau,0) & L \\ \dfrac{P(\lambda\tau,1)}{P(\lambda\tau,>0)} & \dfrac{P(\lambda\tau,>1)}{P(\lambda\tau,>0)} & 0 & I \\ \hline \end{array}$$

ahol $P(a,i) = e^{-a}a^i/i!$ és $P(a,>i) = \sum_{j=i+1}^{\infty} P(a,j)$ a Poisson-valószínűségeket jelöli. A $\pi\Pi = \pi$ egyenletet megoldva,

$$\begin{aligned} \rho &= \frac{\pi_S E(S)}{\pi_S E(S) + \pi_L E(L) + \pi_I E(I)} \\ &= \frac{e^{\lambda T}\lambda^2\tau T}{e^{\lambda T} - \lambda T + \lambda\tau\Big(1 - \lambda\tau + \lambda T - e^{\lambda\tau}\lambda T + e^{\lambda T}\left(-1 + e^{\lambda\tau} + \lambda T\right)\Big)} \\ &= \frac{\lambda\tau\ \lambda T}{1 - \lambda T e^{-\lambda T} + \lambda\tau e^{-\lambda T}\Big(1 - \lambda\tau + \lambda T(1 - e^{\lambda\tau})\Big) + \lambda\tau\Big(e^{\lambda\tau} + \lambda T - 1\Big)} \end{aligned}$$

alakban kapjuk a rendszer kihasználtságát.



27.19. ábra. Diszkrét idejű kitartó CSMA/CD rendszer kihasználtsága.

A 27.19. ábra λ függvényében ábrázolja a kihasználtságot $\tau/T = 0.1$ (pontvonal) illetve $\tau/T = 0.2$ (folytonos vonal) esetén. Az előző esetekhez hasonlóan rövidebb terjedési idő esetén kisebb az ütközés esélye, ezért nő a kihasználtság és az optimális kihasználtsághoz tartozó érkezési intenzitás is.

Összességében a kitartó viselkedés akkor előnyös, ha a csomagtovábbítási intervallumok végén keletkező ütközések miatti késleltetés kisebb, mint a nem kitartó viselkedésből adódó késleltetés. Kis forgalom esetén a kitartó viselkedés csökkenti a késleltetési időt, míg nagy forgalom esetén a nem kitartó eljárás hatékonyabb.

A kitartó és a nem kitartó eljárások között folytonos átmenetet képez a „p kitartó" eljárás, amelyben minden csomagküldési kísérlet alkalmával a felhasználó p valószínűséggel kitartó, $1-p$ valószínűséggel pedig nem kitartó eljárást követ foglalt csatorna esetén. Látható, hogy $p=1$ esetén a kitartó, $p=0$ esetén a nem kitartó eljárást kapjuk, és a köztes tartományban p segítségével beállíthatjuk a sikeres csomagátvitel utáni időrésben érkező csomag intenzitást egy optimális értékre.

27.11.3. IEEE 802.11

A napjainkban rohamosan terjedő vezetéknélküli számítógép hálózati hozzáférést biztosító eljárás (wireless fidelity, WF), amit az IEEE 802.11 azonosító számon szabványosított, szintén a diszkrét idejű (réselt) ALOHA eljárás egy módosított változata. Az eljárás tervezésékor többek között a következő szempontokat próbálták érvényesíteni:

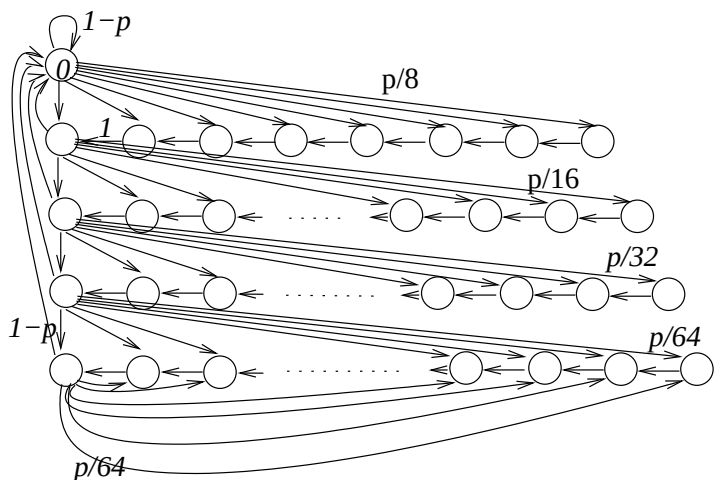
- a sorsolt késleltetési idő korlátos,
- a forgalom változására adaptív,
- prioritás a megkezdett csomagok átvitelére.

Ennek megfelelően az eredeti ALOHA eljárást többek között a következő pontok szerint módosították.

- Ütközéskor a felhasználó egy 1 és M_i közötti számot sorsol egyenletes valószínűséggel és a sorsolás szerinti üres időrés letelte után próbálkozik újra adatküldéssel.
- A sorsolás felső határa függ az adott csomag átvitelére eddig tett kísérletek számától. Az első ütközés után ez az érték $M_1=8$, és minden következő ismétlésnél duplázódik a sorsolás felső határa, azaz $M_i=8*2^{i-1}$ mindaddig, amíg el nem éri az M_{max} értékét.
- A nagyobb adatcsomagok átvitele részekre (ún. szegmensekre) bontva történik. Az ALOHA rendszerben a szegmenseket csak egyesével, egymás után lehet továbbítani úgy, hogy mindegyik szegmensnek külön kell versenyeznie a csatornáért. Ilyenkor az egész csomag átvitelének ideje elfogadhatatlanul nagyra nőhet. Az IEEE 802.11 eljárás ezért lehetőséget biztosít a felhasználóknak arra, hogy ha egyszer magukhoz ragadták a csatornát, akkor további verseny nélkül elküldhessék egy csomag összes

szegmensét. A protokoll ezt úgy valósítja meg, hogy a felhasználók akkor tekintik szabadnak a csatornát, ha azon DIFS (distributed interframe space) ideig nem volt adattovábbítás, míg azonos csomag szegmenseit egy felhasználó SIFS (short interframe space) időközökkel küldheti, és eközben a többi felhasználó végig foglaltnak látja a csatornát, mert SIFS<DIFS.

Az IEEE 802.11 szabvány az ALOHA eljárásra jellemző csatorna elemi hozzáférési módszert (basic access method) többféle erőforrás lefoglalási eljárással ötvözi, aminek az egyik példája az ismertetett szegmentált csomagtovábbítási eljárás. Ebben az anyagban az elemi hozzáférési módszer teljesítményelemzésére javasolt egyik lehetséges modellt ismertetjük. Ez a modell is egy alapvető modellezési közelítésre épül. Azt feltételezi, hogy a rendszerben olyan sok felhasználó van jelen, és ezek viselkedése annyira független egymástól, hogy egy megfigyelt felhasználó elküldött csomagja minden időrésben az előzményektől függetlenül p valószínűséggel ütközik.



27.20. ábra. 802.11 elemi hozzáférésének Markov lánc modellje.

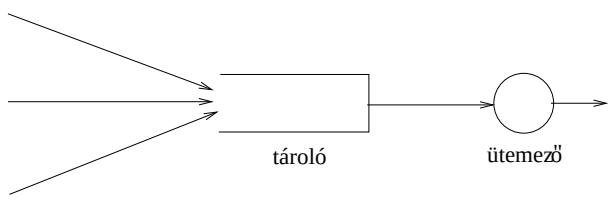
A 27.20. ábrán látható diszkrét idejű Markov-lánc azt írja le, hogy a 0 állapotból indulva $M_1 = 8$ és $M_{max} = 64$ esetén, p ütközési valószínűség mellett a rendszer hogyan továbbíthatja a vizsgált felhasználó csomagját. A Markov-lánc 0 állapotához tartozó visszatérési idő, T_0, a csomagkésleltetés ideje időrésekben van kifejezve. A 0 állapot egyensúlyi valószínűsége p_0 azt adja meg, hogy időrésenként maximálisan hány csomagot tud átküldeni a felhasználó. A $p_0 = 1/E(T_0)$ összefüggés meghatározza a késleltetési idő várható értékét, és a Markov-lánc alapján a késleltetési idő magasabb momentumai és eloszlása is számítható.

27.12. Sorban állásos csomagtovábbítási rendszerek

Ebben a fejezetben a sorbanállási elmélet egy speciális részterületét vizsgáljuk, amelyben az „igényeknek” csomagok, a „kiszolgálónak” pedig adott kapacitású átviteli rendszer felel meg, amely kapacitását teljes egészében a kiszolgálás alatt lévő csomag továbbítására használjuk. Az alapvető sorbanállási modellekhez képest azonban az a leglényegesebb eltérés, hogy megkülönböztetjük az egyes forrásokból érkező csomagokat, és vizsgálataink célja nem az egész rendszerre, hanem az egyes források csomagjaira vonatkozó teljesítményjellemzők elemzése.

A fejezet elején bevezetett kiszolgálási elveket az alábbi csoportosításnak megfelelően tárgyaljuk, ahol az egymást követő eljárások mind rugalmasabb lehetőséget biztosítanak a különböző források csomagjainak eltérő kiszolgálására.

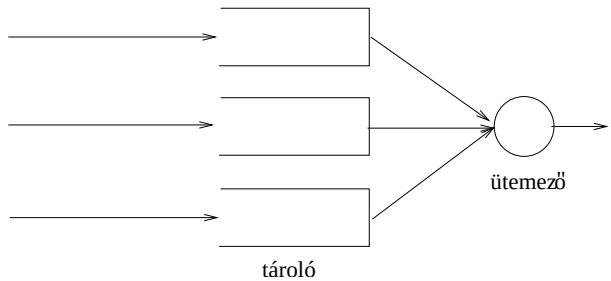
1. Sorrendi kiszolgálás
   A sorrendi kiszolgálás a csomagokat az érkezési sorrend szerint továbbítja. A gyakorlati megvalósítás részleteitől függetlenül a viselkedést tekinthetjük úgy, mintha egy közös tárolóban egymás után várakoznának a különböző forrásokból érkezett csomagok (27.21. ábra). Mivel ebben a rendszerben a különböző források csomagjainak kiszolgálását csak az érkezési pillanatuk határozza meg, ezért semmilyen forgalom megkülönböztetés nem történik, mindegyik forrás számára ugyanazok lesznek a kiszolgálás minőségi jellemzői (például várakozási idő). Ennek a kiszolgálási elvnek a vizsgálatára közvetlenül felhasználhatók az elemi sorbanállási eredmények, melyekre itt nem térünk ki.



27.21. ábra. Sorrendi kiszolgálás.

2. Prioritásos kiszolgálás A prioritásos kiszolgálás során a forrásokat forgalmi osztályokba soroljuk, és a forgalmi osztályokhoz különböző prioritási szintet rendelünk. Az azonos forgalmi osztályba tartozó csomagokat továbbra sem különböztetjük meg (egymáshoz képest továbbra is érkezesi sorrendben kerülnek kiszolgálásra), de a kiszolgáló minden esetben törekszik a nagyobb prioritású csomagokat előbb továbbítani. A rendszer működését tekinthetjük úgy, mintha a különböző os-

ztályba tartozó csomagok külön tárolókban várakoznának, és a kiszolgálónak mindig azt kell eldöntenie, hogy melyik sorban lévő csomagot továbbítsa (27.22. ábra). Ezt a feladatot, ahol a tárolókban várakozó csomagok továbbításának sorrendjét kell eldönteni, szokták ütemezésnek is nevezni. A prioritásos kiszolgálás során az ütemezést egyetlen paraméter, a prioritási szint alapján dönti el a kiszolgáló.



27.22. ábra. Kiszolgálás forgalom megkülönböztetéssel.

3. SÚLYOZOTT ERŐFORRÁS MEGOSZTÁS A prioritásos kiszolgálás rugalmasságának növelése érdekében az ütemezési döntésnél a további szempontok is figyelembe vehetők. A súlyozott erőforrás megosztás során az ütemező úgy választ a kiszolgálandó csomagok közül, hogy egy vizsgált időintervallumra vonatkozóan minden forgalmi osztályra, amelyben van továbbítandó adatcsomag, a súlyának megfelelő kiszolgálási időarány jusson.

Az egymás után felsorolt kiszolgálási módszerek mind általánosabbak, és speciális esetként tartalmazzák az őket megelőző módszereket. Hiszen, ha egy prioritásos rendszerben minden forgalmat azonos forgalomosztályba sorolunk, akkor sorrendi kiszolgálást nyerünk, illetve, ha egy két osztályos súlyozott erőforrás megosztási rendszerben az egyik osztályhoz 1, a másikhoz 0 súlyt rendelünk, akkor prioritásos kiszolgáláshoz jutunk. Többszintű prioritás többlépcsős súlyozott rendszerrel valósítható meg.

27.12.1. Prioritásos kiszolgálás

A prioritásos kiszolgálás során az ütemezőnek semmilyen bonyolult számítási feladatot nem kell elvégeznie a következő kiszolgálandó igény kiválasztásához, hiszen mindig a legnagyobb prioritású nem üres sorból kell kiszolgálni az első igényt.

A módszernek két változata lehetséges, attól függően, hogy egy megkezdett kisebb prioritású igény kiszolgálását egy nagyobb prioritású igény

érkezésekor felfüggesztjük-e vagy sem. Az első esetet megszakításos prioritásnak (preemptive priority), a másodikat megszakításmentes prioritásnak (non-preemptive priority) nevezzük.

A csomagtovábbító rendszerek többségében az egyszerű és gyors működés biztosítása érdekében megszakításmentes prioritást alkalmaznak, hiszen a csomagtovábbítás megszakítása esetén bonyolult párbeszédet (protokollt) kell kialakítani a forrás és a vevő között a csomagrészek megfelelő azonosítására.

A megszakításos rendszereket megkülönböztethetjük a megszakítás mechanizmusa alapján is. Az úgy nevezett munka megőrző (work conserving) rendszerekben a megszakításkor nem vész el a szerver által a megszakításig elvégzett munka, azaz a megszakításig átvitt adatokat nem kell megismételni a magasabb prioritású igény kiszolgálása után. A munkavesztéses (non work conserving) rendszerekben a megszakított igények átvitelét elölről kell kezdeni magasabb prioritású igény kiszolgálása után.

A munka megőrző megszakításos prioritásos rendszerek elemzése exponenciális eloszlású csomagtovábbítási idők esetén egyszerűbb, mint az megszakításmentes rendszer elemzése, mivel a rendszerben lévő igények típusa egyértelműen meghatározza, hogy melyik igényt szolgáljuk ki éppen, ami a megszakításmentes kiszolgálás esetén nem teljesül. Nem exponenciális eloszlású csomagtovábbítási idők esetén azonban a megszakításmentes eset elemzése egyszerűbb, mert a megszakításos rendszerben a hátralévő kiszolgálási idők függenek a megszakításig eltelt időtől, ami (több prioritási szint esetén) több folytonos változótól való függőséget vezet be.

A Markovi (minden prioritás osztályban Poisson érkezési folyamat és exponenciális kiszolgálási idő eloszlás) munka megőrző megszakításos prioritásos rendszerek elemzését elvégezhetjük a rendszert leíró Markov-lánc vizsgálatával, amire a 27.23. ábra mutat példát két prioritási szint és végtelen tárolók esetén. Az ábrán lévő Markov-lánc a 2-es osztály igényeit csak akkor továbbítja, amikor nincs 1-es osztályú igény.

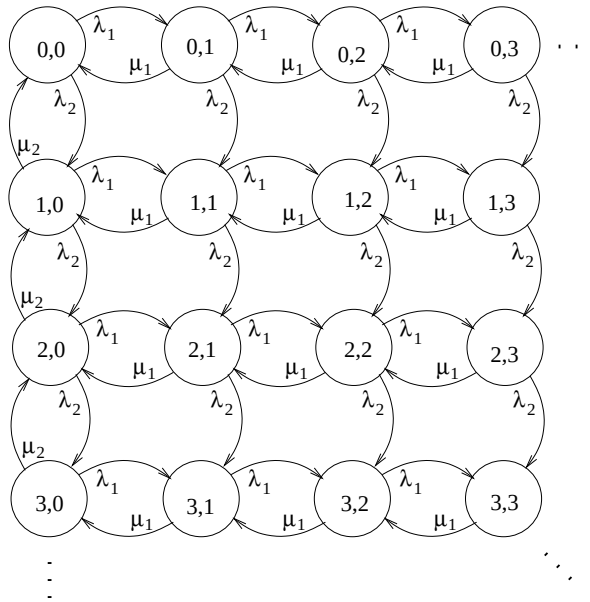
A Markovi munka megőrző prioritásos rendszerek teljesítmény jellemzői közül az egyes osztályok várható késleltetési idejét elemezzük.

Jelölje $r \in \{1, \ldots, R\}$ a prioritás szintet, úgy, hogy a legnagyobb prioritás szint 1, λ_r és μ_r az r szintű csomagok érkezési és kiszolgálási intenzitását, ebből adódóan a prioritás szint által okozott terhelés $\rho_r = \lambda_r/\mu_r$. $\overline{K}_r$ és $\overline{T}_r$ jelöli a rendszerben lévő r szintű igények várható számát, és azok rendszerben eltöltött várható idejét.

Egy megszakításos munka megőrző rendszerben egy adott szintű igény rendszerben eltöltött várható ideje ($\overline{T}_r$) a következő tényezőket tartalmazza:

- az igény kiszolgálási ideje:

$$\frac{1}{\mu_r},$$



27.23. ábra. Megszakításos prioritásos kiszolgálás Markov-lánca 2 prioritásosztály esetén.

- az igény érkezésekor a rendszerben lévő vele azonos és magasabb prioritású igények kiszolgálási ideje:

$$\sum_{s=1}^{r} \frac{\overline{K}_s}{\mu_s} ,$$

- az igény *kiszolgálási* ideje alatt érkezett magasabb prioritású igények kiszolgálási ideje:

$$\sum_{s=1}^{r-1} \frac{\overline{T}_r \lambda_s}{\mu_s} .$$

Kihasználva, hogy $\rho_r = \lambda_r/\mu_r$ a

$$\overline{T}_r = \frac{1}{\mu_r} + \sum_{s=1}^{r} \frac{\overline{K}_s}{\mu_s} + \sum_{s=1}^{r-1} \frac{\overline{T}_r \lambda_s}{\mu_s}$$

egyenlet megoldása $\overline{T}_r$-re

$$\overline{T}_r = \frac{\frac{1}{\mu_r} + \sum_{s=1}^{r} \frac{\overline{K}_s}{\mu_s}}{1 - \sum_{s=1}^{r-1} \rho_s} .$$

Egy hasonló, de megszakításmentes munka megőrző rendszerben egy adott szintű igény rendszerben eltöltött várható ideje ($\overline{T}_r$) a következő tényezőket tartalmazza:

- az igény kiszolgálási ideje: $\dfrac{1}{\mu_r}$,
- az igény érkezésekor kiszolgálás alatt lévő igény hátralévő kiszolgálási ideje:
$$\sum_{s=1}^{R} \frac{\rho_s}{\mu_s} ,$$
- az igény érkezésekor a rendszerben lévő vele azonos és magasabb prioritású igények kiszolgálási ideje:
$$\sum_{s=1}^{r} \frac{\overline{K}_s}{\mu_s} ,$$
- az igény *várakozási* ideje alatt érkezett magasabb prioritású igények kiszolgálási ideje:
$$\sum_{s=1}^{r-1} \frac{(\overline{T}_r - \frac{1}{\mu_r})\lambda_s}{\mu_s} .$$

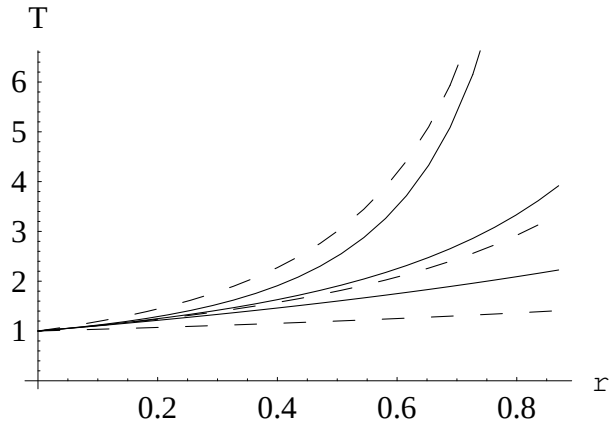
Az így nyert implicit egyenlet megoldása $\overline{T}_r$-re

$$\overline{T}_r = \frac{1}{\mu_r} + \frac{\displaystyle\sum_{s=1}^{r} \frac{\overline{K}_s}{\mu_s} + \sum_{s=r+1}^{R} \frac{\rho_s}{\mu_s}}{1 - \displaystyle\sum_{s=1}^{r-1} \rho_s} .$$

A rendszerparaméterek (λ_r, μ_r) alapján a teljesítmény jellemzők ($\overline{K}_r$, $\overline{T}_r$) meghatározása az első prioritási szintről indulva rekurzívan, a Little-szabály ($\overline{T}_r = \lambda_r \overline{K}_r$) felhasználásával lehetséges.

A 27.24. ábra szemlélteti a rendszerben töltött átlagos idő alakulását a kihasználtság függvényében 3 prioritás szintre megszakításos és megszakításmentes esetben, amikor $\lambda_1 = \lambda_2 = \lambda_3 = \rho/3$, $\mu_1 = \mu_2 = \mu_3 = 1$. Az ábrán a szaggatott vonalak jelzik a prioritás osztályok késleltetését megszakításos és a folytonos vonalak megszakításmentes prioritás mellett. Az elvárásoknak megfelelően a magasabb prioritású forgalom késleltetési ideje kisebb az alacsonyabb prioritásúakénál. Látható továbbá, hogy a nagy prioritású forgalom késleltetése csökken és a kis prioritású forgalom késleltetése nő a megszakításos rendszerben a megszakításmentes rendszerhez képest.

A prioritásos rendszerek tárgyalásának végén összegezzük e rendszerek viselkedésének lényeges tulajdonságait.



27.24. ábra. Forgalom osztályok késleltetése megszakításos és megszakítás mentes prioritás esetén a kihasználtság függvényében.

- Előnyök:
  A prioritásos kiszolgálás kis bonyolultságú forgalommegkülönböztetést végző eljárás, amelynek jellemzője, hogy jelentős különbséget tesz az egyes forrásosztályok kiszolgálása között.
- Hátrányok:
  A prioritásos kiszolgálási módszerből adódóan a kis prioritású forgalom továbbítása teljesen megáll, amíg a nagyobb prioritási szinteken van továbbítandó adat. Ha a magas prioritási szintek forgalma olyan, hogy előfordulhatnak benne hosszú aktív periódusok, akkor ez az alacsony prioritású forgalom hosszú idejű blokkolását úgy nevezett kiéheztetését eredményezi. Az esetek többségében távközlési rendszerekben törekszenek az ilyen kiéheztetés elkerülésére akkor is, ha a kis prioritású adatokat nem kell nagyon szoros határidőn belül átvinni, mert a forgalomforrásokban a kiéheztetés és más hálózati hibák hatása nem különböztethető meg.

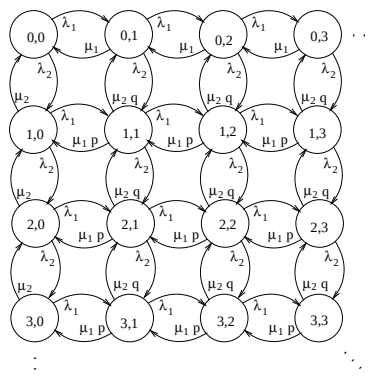
  A megszakításos prioritás esetén az alacsony prioritású forgalom semmilyen hatást nem gyakorol a magas prioritású forgalom továbbítására, azonban ez már nem igaz a megszakításmentes prioritásos rendszereknél. A legnagyobb gondot az okozza, ha a nagy prioritású (általában kis) csomagokat szoros határidővel kell továbbítani, de a nagy méretű, alacsony prioritású csomagok (a csomag méretével arányosan) hosszú ideig foglalják a kiszolgálót abban az esetben, ha a kiszolgálásuk megkezdődött már a nagy prioritású csomag érkezésekor.

Főleg az első hátrány, a kiéheztetés megakadályozására dolgozták ki a súlyozott erőforrás megosztási eljárásokat. Megszakításmentes esetben a magas

prioritású csomagok késleltetése úgy csökkenthető, hogy az alacsony prioritású csomagokat kisebb méretű adatcsomagokban továbbítjuk.

27.12.2. Súlyozott erőforrás megosztás

A súlyozott erőforrás megosztás elvileg a kiszolgáló kapacitás olyan szétosztására törekszik, amelyben a források súlyuk szerint osztoznak a kiszolgáló kapacitáson mindaddig, amíg van továbbítandó adatuk. Két forgalmi osztály, Markovi viselkedés és p, valamint $q = 1 - p$ súlyok mellett a 27.25. ábrán látható Markov-lánc írja le ezt a rendszerviselkedését.



27.25. ábra. Súlyozott erőforrásmegosztás Markov-lánca 2 forgalmi osztály esetén.

Az ismertetett elvi módszert az angol irodalomban *processor sharing* vagy *fluid fair queueing* néven tárgyalják. Az utóbbi elnevezés arra utal, hogy a kiszolgáló úgy viselkedik, mintha az átviendő adatot infinitezimális elemekre bontaná, és ezeket a súlyoknak megfelelő ütemezéssel vinné át. A gyakorlati csomagátviteli rendszerekben azonban nem ezt az elvi módszert alkalmazzák, hanem ennek teljes csomagok átvitelén alapuló közelítését. Olyan ütemező eljárásokat dolgoztak ki, amelyek hosszabb idő átlagában törekszenek a súlyoknak megfelelő erőforrás felhasználási arányok biztosítására, de minden csomagot egészben, a teljes átviteli kapacitás felhasználásával továbbítanak. Ezek közül az ütemezési eljárások közül mutatunk be néhányat a következő jelölések felhasználásával:

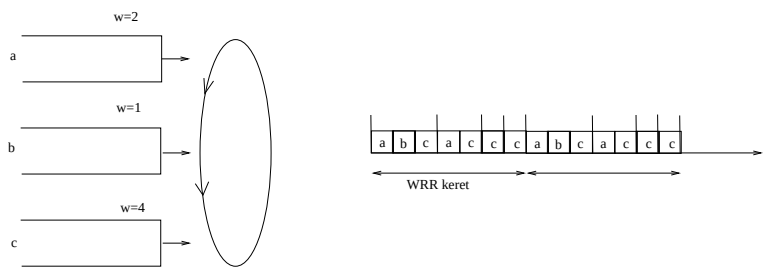
- $w_r, r \in \{1, R\}$ az r-edik forgalmi osztály súlya,
- $v_r, r \in \{1, R\}$ az r-edik forgalmi osztály normalizált súlya ($\sum_r v_r = 1$),
- C a kiszolgáló kapacitása,
- $T_{r,k}$ az r-edik forgalmi osztály k-edik csomagjának időbélyege,
- t_{act} a vizsgált csomagérkezés ideje,

- $L_{r,k}$ az r-edik forgalmi osztály k-edik csomagjának mérete.

Az ütemezési eljárások egy része fix méretű csomagokkal (például ATM cellák) dolgozik. Ezeknek az eljárásoknak nem kell könyvelniük a kiszolgált csomagok méretét, csak azok számát. A változó méretű csomagokat továbbító eljárásoknak viszont a csomagok méretét figyelembe véve kell biztosítani a súlyoknak megfelelő erőforrás megosztást. Az ütemezők egy része a beérkező csomagokra *időbélyeget* számít ki, és az időbélyeg alapján ütemezi a kiszolgálást.

Súlyozott ciklikus ütemezés (Weighted round robin, WRR)

A WRR egy fix méretű csomagok továbbításánál alkalmazott ütemezési eljárás. Az eljárás egy WRR keretet állít össze, amely keretben minden forrás a súlyának megfelelő számú csomagot továbbíthat, és a kiszolgálási folyamatban ezt a keretet ciklikusan ismétli az ütemező (27.26. ábra).

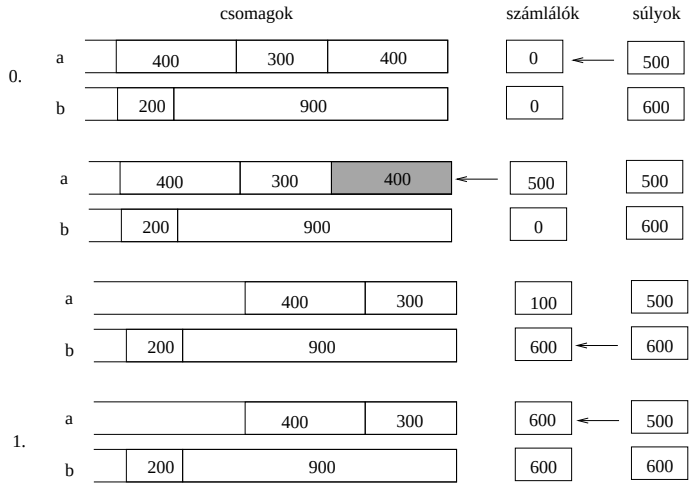


27.26. ábra. Súlyozott ciklikus ütemezés (WRR).

Ciklikus ütemezés hátralék számítással (Deficit round robin, DRR)

A DRR eljárás a WRR módszer változó méretű csomagok kezelésére alkalmas kiegészített változata. Minden adási folyam kap egy hátralékszámlálót (*deficitcounter*) és egy küszöbszámot. A küszöbszám adja meg, hogy egy folyam mennyi bitet (vagy bájtot) adhat egyszerre, amikor a sor rá kerül. Ha egy továbbításra váró csomag mérete meghaladja a küszöbszámot, akkor a küszöbszám értéke a hiányszámláló értékét növeli, így az adott folyam a következő sorra kerüléskor a megnövelt küszöbszám erejéig adhat. A szemléltetéshez tekintsük a 27.27. ábrát.

Látható, hogy ha az egyes esetekben egy adott folyam csomagja nagyobb, mint az aktuális adási méret (hiányszámláló érték), akkor az a nagy csomag előre engedi a másik folyam kisebb csomagját. A *Fair Queueing* filozófia ebben a formában érvényesül (a nagy csomag többet kénytelen várakozni a kisebb csomagnál, így arányos a várakozás időtartama a csomagmérettel). Az egyes források körben forgó sorrendben követik egymást. A módszer lehetőséget ad



27.27. ábra. Ciklikus ütemezés hátralék számítással (DRR).

súlyozásra az egyes folyamok küszöbértékének megkülönböztetésével.

Fontos szempont az ütemezési módszer kiválasztásánál az adott módszer számítási komplexitása, mivel a gyakorlatban az ütemezést nagyon gyorsan kell elvégezni. A *DRR* módszer előnye a számítás egyszerűsége, hiszen az ütemezési sorrend megállapításakor konstans számú művelet elvégzése szükséges. További előnye a módszernek, hogy változó csomagméret esetén is egyszerűen implementálható. A szakirodalomban megtalálható módszerek közül ez nem igaz mindegyikre.

Virtuális óra alapú ütemezés (Virtual clock, VC)

A VC ütemezési eljárás mindegyik forgalmi osztályhoz saját, virtuális órát rendel. A virtuális óra minden egyes érkezéskor ugrik az adott forgalmi osztályra jellemző mértékben a

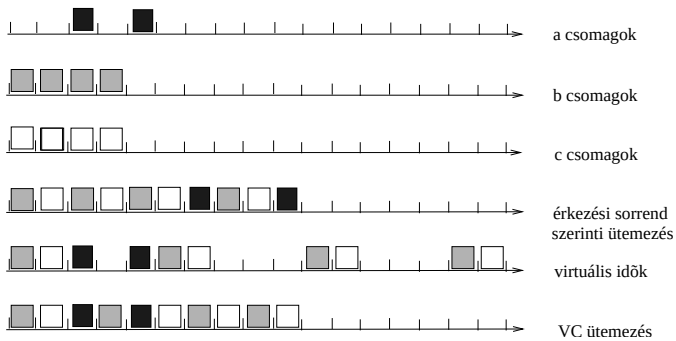
$$T_{r,k+1} = \max(t_{act}, T_{r,k}) + \frac{L_{r,k+1}}{C\ v_i}$$

összefüggés szerint. Az egyes csomagokat osztályuk virtuális órájának aktuális értékével időbélyegezzük. Az ütemező az időbélyegek növekvő sorrendjében szolgálja ki az egyes igényeket.

Az óra lépésköze forgalmi osztályonként változhat az adott osztályhoz rendelt kapacitáshányadnak megfelelő mértékben. Szemléltetésre tekintsük a következő példát.

A kiszolgálandó igények három forgalmi osztályba tartoznak. Az egyszerűség kedvéért tegyük fel, hogy az egyes igények (csomagok) mérete állandó – így a kiszolgálás időtartama is. Az a osztályhoz rendeljük a kiszol-

gáló kapacitásának felét, a b és a c osztályhoz a kiszolgáló kapacitásának ötödét. Másként megfogalmazva: az a osztályból átlagosan két időegységenként továbbíthatunk csomagot, míg a másik két osztályból átlagosan öt időegységenként.



27.28. ábra. Virtuális óra alapú ütemezés szemléltetése (VC).

A 27.28. ábra mutatja mi történik, ha az egyes osztályok a dedikáltól eltérő intenzitással akarnak adni. Látható, hogy a dedikált kiszolgálás minőséget a VC elv jobban megvalósítja, mint például az érkezési sorrend szerinti kiszolgálás. Érkezési sorrend szerinti kiszolgálás esetén, ha egy osztály a megengedettnél több forgalmat generál, az a többi osztály kiszolgálásának a rovására megy, míg a VC ütemezésnél ez nem történik meg.

A stratégia szemléletes értelmezése lehet, hogy a szerver tulajdonképpen az egyes osztályok kiszolgált csomagjait számolja (a megfelelő súlyozás figyelembe vételével). Az elv ilyenformán az állandó csomagméretű hálózati technológiákra (például ATM) használható. Változó csomagméret esetén (például IP) annyi módosítást szükséges tenni, hogy ne a csomagok, hanem a bitek számát tartsa nyilván a kiszolgáló egység. Ebben az esetben csomagérkezéskor a virtuális óra lépésközét súlyozni kell a beérkezett csomag méretével, mint ahogy az a fenti képletben történik.

Több hasonló elven alapuló ütemezési eljárást alkalmaznak még a gyakorlatban, például starting potential fair queueing (SPFQ), worst-case fair weighted fair queueing (WF^2Q), self clock fair queueing (SCFQ), self clock fair queueing (SCFQ), amelyek ismertetésére itt nem térünk ki.

Gyakorlatok

27.12-1. Határozzuk meg a csomagtovábbítás sorrendjét három forgalmi osztály esetén DRR eljárás mellett, ha az átviendő csomagok mérete:

- 1. osztály: 254, 234, 50, 654, 51,
- 2. osztály: 42, 134, 62, 54, 612, 57,

- 3. osztály: 155, 82, 69, 63, 152,

és az osztályok küszöbszámai rendre $40, 20, 50$, illetve $20, 40, 50$.

27.12-2. Mely csomagokat dobja el az $L = 3$, $I = 2$ paraméterű GCRA algoritmus az alábbi időpontokban érkező csomagok közül: $t = 0, 1, 4, 5, 7, 8, 9, 10, 12, 13, 14, 15, 17, \ldots,$

Feladatok

27-1 Sorhossz a távozási időpontokban

Tekintsük a korábban tárgyalt $G|G|1|\infty$ rendszert. Jelölje a korábbiak szerint $s_n (n = 1, 2, \ldots)$ azt az időpontot, amikor az n-edik igény kiszolgálása befejeződik és legyen $L_n = L(s_n), n \geq 1$. Jelölje $Z_n,\ n = 1, 2, \ldots$ azon igények számát, amelyek az n-edik igény Y_n kiszolgálási ideje alatt lépnek be a rendszerbe, azaz $Z_n = N(s_n) - N(s_n - Y_n),\ n \geq 1$. Igazoljuk, hogy az $\{L_n, n \geq 1\}$ sorozatra érvényes a következő rekurrens szabály:

$$L_n = I(L_{n-1} > 0)(L_{n-1} - 1) + Z_n = (L_{n-1} - 1)^+ + Z_n \ .$$

27-2 Várakozási idő

Tekintsük a $G|G|m|\infty$ tömegkiszolgálási rendszert, amely m kiszolgáló egységből és korlátlan számú várakozó helyből áll. Vezessük be az n-edik igény *várakozási vektorát*

$$W_n = (W_{n,1}, \ldots, W_{n,m}), \qquad n = 1, 2, \ldots,$$

ahol $W_{n,j}$ jelenti azt a véletlen időmennyiséget (v.v.-t), amennyit várakoznia kellene a t_n időpillanatban beérkező n-edik igénynek ahhoz, hogy i darab kiszolgáló egység felszabaduljon az összes korábban beérkezett $(1, \ldots, n-1$ sorszámú) igénytől. Legyen $W_1 = (W_{1,1}, \ldots, W_{1,m}) = 0$, vagyis a rendszer üresen kezd dolgozni. Jelölje tetszőleges $x = (x_1, \ldots, x_m) \in \mathbb{R}^m$ mellett

$$x^+ = (x_1^+, \ldots, x_m^+), \quad \text{ahol} \quad s^+ = \max(s, 0), \quad s \in \mathbb{R}$$

és

$$R(x) = (x_{i_1}, \ldots, x_{i_n}), \quad x_{i_1} \leq x_{i_2}, \ldots \leq x_{i_n} \ ,$$

vagyis az $R(x)$ függvény az x vektorból komponenseiben nagyság szerint rendezett vektort képez. Vezessük be még az alábbi vektorokat

$$e = (\overset{(1)}{1},\ \overset{}{0}, \ldots,\ \overset{(m)}{0}), \qquad i = (\overset{(1)}{1},\ \ldots,\ \overset{(m)}{1}) \ .$$

Bizonyítandó, hogy a $\{W_n, n \geq 1\}$ sorozat a következő módon határozható meg:

$$W_{n+1} = R([(W_n + Y_n e) - X_n i]^+), \qquad n = 1, 2, \ldots .$$

27-3 Elutasításos rendszerek

$G|G|m|0$ elutasításos rendszerek. Ez a TKR m kiszolgáló egységből áll, egyáltalán nincs várakozó hely. Ebben az esetben csak virtuális értelemben beszélhetünk várakozásról, hiszen a rendszer elutasításos. Ekkor a W_n vektor $W_{n,j}$ komponense azt jelenti, hogy mennyi időt kellene várakoznia a t_n időpillanatban beérkező n-edik igénynek ahhoz, hogy i kiszolgáló egység felszabaduljon az összes korábban beérkezett $(1, \ldots, n-1$ sorszámú) igénytől (ha $W_{n,1} > 0$, akkor az n-edik nem kerül kiszolgálásra). Bizonyítandó, hogy ebben az esetben igaz a következő rekurrens összefüggés:

$$W_{n+1} = R([(W_n + Y_n I(W_{n,1} = 0)) - X_n i]^+) \; .$$

27-4 ALOHA teljesítmény elemzés

Írjunk tetszőleges szimulációs környezetben szimulációs programot folytonos idejű ALOHA rendszer teljesítmény elemzésére. N felhasználó közül azok, akik nincsenek blokkolt állapotban λ paraméterű exponenciális gondolkodási idő után generál egy csomagot, és annak sikeres átviteléig blokkolt állapotba kerül. A csomagok fix méretűek. Továbbításuk ideje T. Az ütközéseket követően az ütközött csomagok újraküldési késleltetése a $(0, b)$ intervallumon egyenletes eloszlású.

Szimulációs vizsgálat segítségével határozzuk meg b optimális értéket, amely mellett a a rendszer átvitele maximális, ha $\lambda = 1$, $N = 3$ és $T = 0.1$.

27-5 GRCA algoritmus

Szimulációs vizsgálattal határozzuk meg, hogy független p paraméterű geometriai eloszlású $(a_i = p(1-p)^{i-1})$ időnként érkező csomagok hányad részét dobja el az $L = 10$, $I = 10$ paraméterű GCRA algoritmus, ha $p = 0.1$.

Megjegyzések a fejezethez

A tömegkiszolgálási rendszerek elméletének hatalmas szakirodalma van, ennek ellenére magyar nyelven viszonylag kevés mű jelent meg, például [113, 156, 288, 304]. A klasszikus tömegkiszolgálási rendszerekkel kapcsolatos eredményeket részletesen tárgyalja Kleinrock [156] könyve, az elmélet kezdeteinek részletes bibliográfiája megtalálható Saaty [257] könyvében. A speciális területek közül megemlítjük az ismétléses (retrial) rendszereket,

amelyek esetén a visszautasított igény újabb kiszolgálást kezdeményezhet véletlen idő múlva (lásd Falin-Templeton [72]), vagy valamilyen szabályozott módon (lásd Lakatos [167]), valamint a centrális zárt rendszereket. Szeidl [277, 278], megmutatta, hogy amennyiben a centrális zárt rendszerben az összes kiszolgálás várható értéke véges, akkor a rendszer különböző mutatóinak létezik határeloszlása. A használt valószínűségszámítási eszközöket illetően a [242, 249], a Bessel-függvényekkel kapcsolatban pedig a [185] könyvet ajánljuk.

A gyakorlatban felmerülő tömegkiszolgálási problémák bonyolultsága miatt gyakran kell a szimuláció eszközéhez nyúlnunk. Ekkor pszeudóvéletlenszám-generátor felhasználásával hozzuk létre a rendszert jellemző eloszlással (lásd például Deák [62], Gentle [98]) a beérkezési folyamatot és az egyes igények kiszolgálási idejét megadó véletlen számsorozatot. Számos tömegkiszolgálási rendszer rendelkezik a regeneratív tulajdonsággal, a valószínűségszámításból ismert felújításelméletre támaszkodva (lásd Feller, Karlin-Taylor [76, 147]) hatékonyan vizsgálható ezzel a módszerrel (lásd Crane, Lemoine [54] és Shedler [266]). Példaként említhetjük a centrális zárt rendszert (Szeidl [277, 278]) és az M|G|1 rendszer egyensúlyi eloszlásának meghatározását ([168].)

A távközlési protokollok (a protokoll a távközlési kapcsolat kialakítását és működését leíró szabályrendszer) hierarchiájának elemeivel és megoldásaival foglalkozik [293], a témakörhöz kapcsolódó magyar nyelvű források közé tartozik [113, 136, 156, 288]. A sorbanállás-elmélet egy speciális részterületét alkotja a csomagok továbbítása, ebben az esetben megkülönböztetjük az egyes forrásokból érkező csomagokat és vizsgálataink célja az egyes források csomagjaira vonatkozó teljesítményjellemzők elemzése. A véletlen erőforrásokhoz való hozzáférés konfliktus feloldó ALOHA eljárásnak különböző algoritmusai ismertek [29, 59, 86]. Sorbanállásos munka megőrző prioritásos rendszerek teljesítményjellemzői közül az egyes osztályok késleltetési idejét elemezzük [33]. Több ütemezési eljárást alkalmaznak a gyakorlatban, például [276] ismerteti ezen eljárások egy teljesítményelemzéssel kiegészített részletes összefoglalását.

28. Ütemezéselmélet

Egy ütemezési feladat megoldása egy üzem, üzemrész, műhely vagy berendezés tevékenységének meghatározását jelenti rögzített időszakon belül. Ez két dolgot takar: egyrészt egy termelési feladat kiválasztását, másrészt a kiválasztott feladat megoldásának időbeli megtervezését. Ehhez számba kell venni az üzemben ható összes tényezőt, és ezeknek megfelelően kialakítani egy modellt, majd azt minden konkrét esetben megoldani. Ebben a fejezetben feltesszük, hogy az üzem működése pontosan jelezhető előre, ezért csak determinisztikus modellekkel foglalkozunk.

A termelést fel kell osztani kisebb részekre. Nem mindegy, hogy a dolog melyik oldalát ragadjuk meg. Ha a termelésnek a tevékenység jellege a fontos, akkor ***feladatokról*** vagy ***tevékenységekről*** beszélünk, ha azonban a termelés anyagi oldalát kívánjuk hangsúlyozni, akkor a ***munkadarab*** vagy a ***sorozat*** fogalmát használjuk. (Az utóbbin egyforma munkadarabokat kell érteni, amelyek együtt haladnak át a termelés minden fázisán.) A munkadarabok és a sorozatok sok tekintetben hasonlók, eltérő tulajdonságaikra külön felhívjuk a figyelmet. Ha a munkadarab vagy a sorozat fogalmával dolgozunk, akkor azt gondoljuk, hogy az adott üzem termékei fizikailag elkülöníthető darabokból állnak, a termelés pedig úgy folyik, hogy a majdani termék valamely kiinduló állapotban bekerül az üzembe, és bár változik az egyes fázisokon, lényegében mégis ugyanaz marad, azaz nem épül össze más termékekkel, illetve nem válik több részre. Ebben az esetben a munka tárgyát tekintjük alapvetőnek, de ekkor is szükség van arra, hogy a munkát, mint tevékenységet felosszuk részekre. Ebből a célból bevezetjük a ***művelet*** fogalmát. Egy művelet az az átalakítás, amit egy gép egyszerre elvégez a munkadarabon. Ha tehát egy munkadarab többször kerül ugyanarra a gépre, akkor több műveletről van szó.

A rendszer harmadik fontos elemét a technológiai körülmények alkotják, melyeken mind az általános, mind a vizsgált időszakra vonatkozó egyedi feltételeket, ezen belül a ***gépek*** kapacitását és a termelés során rendelkezésre álló és igényelt ***erőforrásokat,*** a ***raktározási*** és ***szállítási*** feltételeket értjük. Modellezési szempontból ide soroljuk a munkaerőt is. A különböző gépek általában különböző technológiai feladatokat látnak el, de előfordulhat, hogy egyes gépek helyettesíthetik egymást.

A gépekkel kapcsolatos legfontosabb kérdések a következők:

1. miben mérjük a ***kapacitásukat;***

2. mi a gépeknek a technológiai folyamatban betöltött helye;

3. mi a gépek és a megmunkálások viszonya.

Ezek a kérdések természetesen nem függetlenek egymástól.

Mivel egy vállalatnál a termelés célja nyereség elérése, ezért a modellben tekintettel kell lenni a költségekre, a termékek eladási árára és a terméken keletkező nyereségre.

Mindaz, amit eddig felsoroltunk, még csak egy elvi üzem kereteit szabja meg. Attól válik működő üzemmé, hogy egy több-kevesebb pontossággal meghatározott ***termelési feladatot*** kell elvégeznie, aminek a megszervezése a feladat, ami igen erős megszorítást jelent a lehetőségeknek a technológiai feltételek által megengedett végtelen gazdag tárházával szemben

A továbbiakban egyes esetekben nem kell megkülönböztetni a tevékenységeket, a munkadarabokat és a sorozatokat. Ilyenkor összefoglaló néven ***munkafeladatról*** fogunk beszélni.

28.1. Formális rendszer ütemezési feladatok osztályozására

Ismert egy formális rendszer, amellyel az ütemezési feladatokat nagyon tömören le lehet írni. Ez a formális rendszer nem öleli föl az összes lehetőséget, ezért a következő feltételezések állandóan érvényben vannak:

1. minden gép a teljes vizsgált időszakban rendelkezésre áll;

2. minden munkafeladat elvégzésére egyetlen technológiát határoztunk meg;

3. minden gép egyszerre csak egy munkadarabon dolgozhat;

4. minden gép, az esetlegesen szükséges átszerszámozástól eltekintve, azonnal megkezdheti a következő megmunkálást, mihelyst az előzőt befejezte;

5. a gyártásközi raktár(ak) kapacitása végtelen;

6. az előzés megengedett;

7. a soron következő munkadarab azonnal megmunkálható, mihelyst a gép szabaddá vált, feltéve, hogy a munkadarab előző megmunkálása már befejeződött.

A feladatok leírása három mezőből áll, melyeket így szokás jelölni:

$$\alpha \mid \beta \mid \gamma \ , \tag{28.1}$$

ahol α a gépekre és a legfontosabb technológiai adottságokra, β a munkafeladatokra, illetve ezeknek a gépekhez és egyéb körülményekhez való kapcsolatára vonatkozó feltételeket tartalmazza, végül γ a matematikai feladat célfüggvényét írja le. Az egyes mezők több tényezőre vonatkozó információt is tartalmazhatnak.

A j-edik munkafeladat befejezési időpontja C_j, határideje d_j. Ezen munkafeladat műveleteinek száma m_j, és az i-edik gépen való megmunkálásának ideje p_{ij}.

28.1.1. Az α mező

Itt kell megadni a gépek számát és a technológiai útvonal típusát. A technológiai útvonal azt határozza meg, hogy a munkadarab milyen sorrendben keresi fel a gépeket.

- 1,2,...: A gépek száma 1 (vagy 2 stb.), a megadott számtól függően.
- P: Azonos párhuzamos gépek vannak, azaz egyetlen homogén gépcsoportról van szó, ahol bármely művelet bármely gépen elvégezhető, és minden művelet bármelyik gépen ugyanannyi ideig tart.
- Q: Hasonló párhuzamos gépek – csak a gépek sebességében van különbség, azaz a j-edik műveletnek az i-edik gépen való elvégzése p_j/s_i ideig tart, ahol p_j csak a művelettől, s_i csak a géptől függő állandó.
- R: Általános párhuzamos gépek, ahol az egyes műveletek ideje tetszőleges lehet a különböző gépeken.
- O: A technológiai útvonal kötetlen.
- F: Egyutas ütemezési probléma, azaz minden munkadarab azonos sorrendben keresi fel a gépeket.
- J: Többutas ütemezési probléma, azaz a munkadarabok technológiai útvonala eltérő.

Például a $J3$ többutas ütemezési problémában a gépek száma 3. Minden munkadarab technológiai útvonalának leírását külön meg kell adni.

28.1.2. A β mező

Ez a mező tartalmazza az egyéb technológiai feltételeket.

- $p_{ij} = 1, p_{ij} \in \{1, 2\}$ stb.: A megmunkálási idők speciálisak, például mindegyik 1 (azonos hosszú), illetve mindegyik 1 vagy 2.
- *idle*: Olyan ütemezés is lehetséges, ahol betervezett állásidők vannak, azaz egy berendezés annak ellenére sem dolgozik, hogy van olyan munkadarab, amelyik megmunkálásra vár.
- *pmtn*: A műveletek megszakítása megengedett. Ilyenkor a megszakítás nem okoz veszteséget sem időben, sem költségben.
- *prec, tree*: A munkafeladatok között megelőzési reláció áll fenn, melyet valamely G irányított gráf ír le. Ha a mezőben *tree* szerepel, akkor G fa.
- r_j: Az egyes munkafeladatoknak különböző lehet a rendelkezésre állási ideje, illetve, ha itt még további feltétel szerepel, akkor a rendelkezésre állási idők ennek eleget tesznek.
- d_j: Az egyes munkafeladatoknak különböző lehet a határideje, illetve, ha itt még további feltétel szerepel, akkor a határidők ennek eleget tesznek. A határidőket ***puha,*** azaz megsérthető feltételeknek tekintjük. Ha a határidők ***tényleges*** feltételeket jelentenek, akkor egy szokásos jelölés $\overline{d}_j$.)
- *no-wait*: A munkadarabok nem várhatnak a gépek előtt.
- *res, res*1: Az erőforrások csak korlátozottan állnak rendelkezésre. Ha a mezőben *res*1 áll, akkor egyetlen erőforrásról van szó.
- $m_j \leq \hat{m}$: Az egyes munkadarabokhoz legfeljebb $\hat{m}$ művelet tartozik.
- *overlap*: átlapolás korlátlanul megengedett, azaz egy újabb gép már dolgozhat a munkadarabon, miközben az előző még nem fejezte be a műveletet. (Ez a sorozatok jellegzetes tulajdonsága.)

A konvenció szerint, ha egy elem nem szerepel, akkor a megfelelő megszorítás nem érvényes. Például, ha nincs a mezőben *pmtn*, akkor ez azt jelenti, hogy a megmunkálások megszakítása nem lehetséges.

28.1.3. A γ mező

A formális rendszer csak kompromisszumos, azaz a késéseket nem kizáró célfüggvényeket enged meg. Ennek oka az, hogy minden más esetben nehezen kezelhető a feladat, és csak lineáris és egészértékű programozási feladatokkal modellezhető a probléma.

Az egyes munkafeladatokhoz a következő értékeket értelmezzük:

- C_j : a befejezési időpont,

- $L_j = C_j - d_j$: a késés,
- $T_j = \max\{0, L_j\}$: a tényleges késés,
- $U_j = \operatorname{sgn}(T_j)$: egységnyi büntetés.

Mindegyikből többféleképpen lehet célfüggvényt formálni, például minimalizálhatjuk a megfelelő értékek maximumát, összegét, súlyozott összegét. Ennek megfelelően tizenkét lehetséges célfüggvény a következő

$$C_{\max},\ L_{\max},\ T_{\max},\ U_{\max}\ ,$$

$$\sum C_j,\ \sum L_j,\ \sum T_j,\ \sum U_j\ , \tag{28.2}$$

$$\sum w_j C_j,\ \sum w_j L_j,\ \sum w_j T_j, \sum w_j U_j\ ,$$

ahol a súlyok nemnegatívak.

Ha a munkafeladatok száma adott, akkor az összeg minimalizálása ekvivalens a megfelelő mennyiség *átlagának* a minimalizálásával, hiszen a két érték csak egy konstans szorzóban tér el egymástól. Azt az esetet, amikor a munkafeladatok folyamatosan érkeznek be, és a helyzet állandóan változik, *on-line* problémának nevezik. Ekkor csak a várható értéket lehet optimalizálni. Vagyis az összeg célfüggvény az on-line eset modellezését is szolgálja.

Látható, hogy a lényegesen különböző célfüggvények száma kevesebb. Ugyanis $\sum w_j C_j$ és $\sum w_j L_j$ csak a konstans $-\sum w_j d_j$ tagban különböznek egymástól. Továbbá $L_{\max}$ minimalizálása egyben minimalizálja a $T_{\max}$ és $U_{\max}$ célfüggvényeket is.

Az utóbbi kettő közül az első minimalizálja a másodikat, hiszen $U_{\max} = 0$ akkor és csak akkor, ha egyetlen munkadarab sem késik, azaz $T_{\max} = 0$. Az $U_{\max}$ célfüggvény szerepe csak annyi, hogy olyan ütemezést keresünk, amely egyetlen határidőt sem sért meg.

A (28.2)-ben megadott függvények a ***reguláris célfüggvények*** közé tartoznak, melyek értéke a befejezési időpontokban monoton növekedő. ***Irregulárisnak*** nevezünk egy célfüggvényt, ha nem rendelkezik ezzel a tulajdonsággal. Ilyen az ú.n. ***sietés*** és a ***súlyozott pontosság*** minimalizálása, ahol

- $E_j = \max\{d_j - C_j, 0\}$: a sietés,
- $\sum(v_j E_j + w_j T_j)$: a súlyozott pontosság.

Ezen célfüggvényeknek akkor van létjogosultsága, ha pontos előrejelzéssel rendelkezünk az eladások várható időpontjáról. Ekkor előre gyártani azért haszontalan, mert raktározási költséggel jár.

Némely esetben nem konkrét célfüggvényről lesz szó, hanem valamely tulajdonságokat kielégítő valamennyi célfüggvényről. Ilyen esetben a γ mezőbe

f kerül.

28.1. példa. Az $1 \mid\mid \sum C_j$ feladat az az egygépes ütemezési feladat, ahol a termékek ütemezésére a vizsgált időszakon belül nincs megszorítás, azaz nincsenek határidők, rendelkezésre állási idők, megelőzési relációk stb.

A befejezési idők összegét kívánjuk minimalizálni. Könnyen látható, hogy a feladat optimális megoldásában a munkadarabokat a megmunkálási idők növekvő sorrendjébe rakjuk. Ezt a gyakran előforduló sorrendet az angol irodalomban SPT ütemezésnek nevezik.

28.2. példa. A klasszikus egyutas ütemezési probléma háromgépes speciális esete $F3 \mid\mid C_{\max}$.

Gyakorlatok

28.1-1. Írjuk le a formális nyelven azt a feladatot, amelyben a munkák azonos sorrendben keresik fel a gépeket, és nem várhatnak a munkák a gépek előtt.

28.1-2. Van-e az $1|r_j|C_{\max}$ feladatban olyan munka, ami $t = 0$-ban nem végezhető?

munka	1. gép	2. gép	3. gép
1.	3	22	2
2.	22	20	20
3.	20	14	18

28.2. A Gantt-diagramok

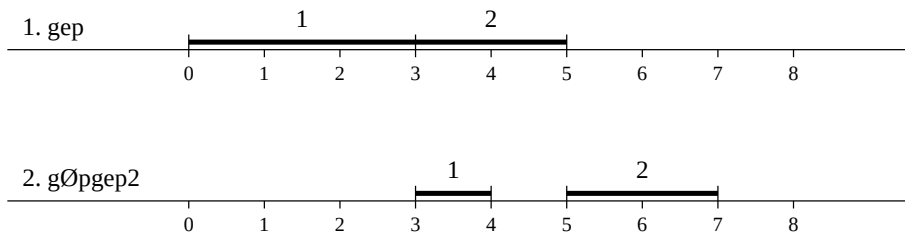
Az ütemezések szemléltetésének fontos eszköze az ú.n. Gantt-diagram. Ebben minden gépet egy-egy időtengellyel szemléltetnek. Az időtengelyeken feltüntetik, hogy az egyes munkadarabok mikor kerülnek megmunkálásra az adott gépen, illetve a gépnek mikor van állásideje.

28.3. példa. Tekintsük azt az $F2 \mid\mid C_{\max}$ feladatot, ahol $n = 2$, és a megmunkálási időket az alábbi táblázat foglalja össze.

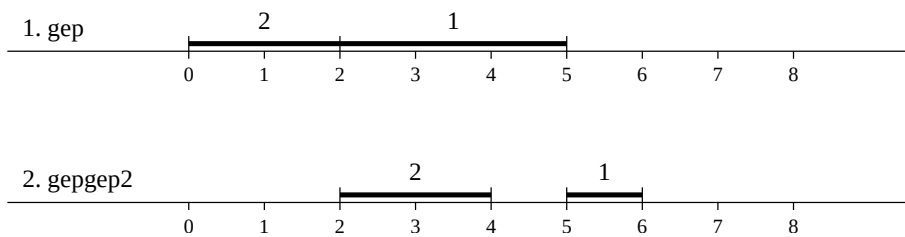
munkadarab	1. gép	2. gép
1.	3	1
2.	2	2

Ha mindkét gépen az (1,2) sorrendet alkalmazzuk, akkor a 28.1. ábrához jutunk.

Ha pedig a fordított sorrend szerinti ütemezést tekintjük, akkor a 28.2. ábrán látható Gantt-diagramot kapjuk.



28.1. ábra. A Gantt-diagram az (1,2) ütemezés esetén.

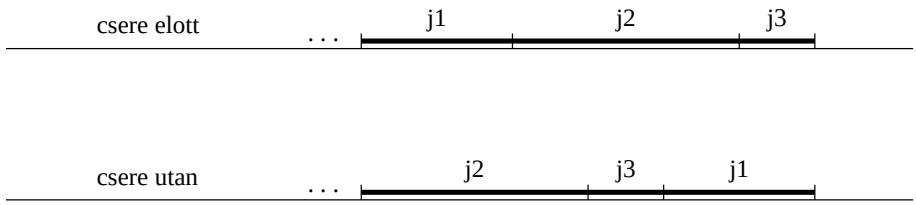


28.2. ábra. A Gantt-diagram a (2,1) ütemezés esetén.

A Gantt-diagramok fontos eszközei a matematikai bizonyítások szemléltetésének. Segítségükkel könnyen beláthatók az alábbi állítások. Itt végig feltesszük, hogy az egyes gépeken csak a megmunkálások sorrendjét kell megadni, ugyanis nincs értelme az ütemezés szerint soron következő megmunkálással várni. Ez még nem zárja ki a betervezett állásidők lehetőségét, hiszen itt még lehetséges, hogy egy gép előtt egy munkadarab bevár egy másik munkadarabot azért, hogy az megelőzze.

Tekintsünk egy tetszőleges ütemezést és egy rögzített i gépet. A gép működését a Gantt-diagram egybefüggő működő és álló szakaszokra osztja fel, hiszen mihelyst lekerül egy munkadarab a gépről, azonnal rákerül a másik, azaz úgy tekintjük, hogy a gép folyamatosan dolgozik, feltéve persze, hogy van egyáltalán az adott pillanatban végezhető munkája. Tekintsünk egy ilyen összefüggő szakaszt. Jelölje

- J az ehhez a szakaszhoz tartozó munkadarabok halmazát,
- t a szakasz kezdetének időpontját,
- r_{ij} $(j \in J)$ a j-edik munkadarab megérkezésének időpontját az i-edik gépre adott ütemezés szerint (r_{ij} tulajdonképpen a j-edik munkadarab rendelkezésre állási ideje az i-edik gépen)
- u_{ij} $(j \in J)$ a j-edik munkadarab ütemezését, vagyis a j-edik munkadarab megmunkálásának megkezdését, az i-edik gépen.



28.3. ábra. A (28.5) feltételt kielégítő cserének nincs hatása a $C_{\max}$ értékre.

Ekkor nyilvánvaló, hogy

$$\text{minden } j \in J \text{ esetén } \; t \;\leq\; r_{ij} \; . \tag{28.3}$$

Különben ugyanis közvetlenül a t időpont előtt a gép állna, miközben egy munkadarab várna a megmunkálásra. Hasonlóképpen teljesülnie kell, hogy

$$\text{minden } i \text{ és minden } j \in J \text{ esetén } \; r_{ij} \leq u_{ij} \; . \tag{28.4}$$

Az összefüggő szakaszon belül átrendezhetjük a megmunkálások sorrendjét, de természetesen az így adódó új ütemezésnek is ki kell elégítenie a (28.4) egyenlőtlenségeket. Továbbá, ha a működő szakasz összefüggő marad, akkor teljesülnie kell a

$$t \;+\; \sum_{l=1}^{k} p_{i\pi(l)} \;\geq\; r_{i\pi(k+1)}, \quad k = 0, \ldots, |\, J \,| - 1 \; , \tag{28.5}$$

egyenlőtlenségeknek, ahol π a J-beli munkadarabok új sorrendjét jelöli.

Végül tegyük fel, hogy az i-edik gép minden technológiai útvonalon az utolsó, azaz itt befejeződik a termelés. Legyen továbbá a vizsgált működő időszak az utolsó. Ekkor bármilyen átrendezés ezen a szakaszon, ami megőrzi a szakasz összefüggését, nem változtatja meg az utolsónak befejeződő munkadarab befejezési időpontját, ami

$$t \;+\; \sum_{j \in J} p_{ij} \; , \tag{28.6}$$

azaz nem változtatja meg a $C_{\max}$ értéket (lásd a 28.3. ábrát).

Gyakorlatok

28.2-1. Ábrázoljuk az alábbi egyutas ütemezési feladat (3,2,1) sorrendű ütemezését Gantt-diagramon.

munka	1. gép	2. gép	3. gép
1.	3	22	2
2.	22	20	20
3.	20	14	18

28.2-2. Hogyan változik a célfüggvény értéke az $F2||C_{\max}$ feladatban, ha úgy rendezzük át a munkákat a második gép utolsó összefüggő szakaszán, hogy az egynél több szakaszra bomlik?

28.3. Ütemezési problémák egyetlen gépen

A legegyszerűbb ütemezési feladatokban csak egyetlen gép van. A valóság általában bonyolultabb. De ez az eset részfeladatként felmerül a bonyolultabb helyzetekben. Továbbá jól alkalmazható, ha valamelyik berendezés szűk keresztmetszetet jelent.

Az alábbi három tétel azt mondja ki, hogy számos esetben nem érdemes megszakítást és betervezett állásidőt tartalmazó ütemezéssel foglalkozni.

28.1. tétel. *Ha f reguláris célfüggvény, akkor az $1 \mid d_j, prec, idle, pmtn \mid f$ és az $1 \mid d_j, prec, idle \mid f$ feladatoknak van közös optimális megoldása.*

Megjegyzés. Mivel a feladatok leírásában a rendelkezésre állási idők nem szerepelnek, ezért a konvenció szerint minden$j$ esetén $r_j = 0$.

Bizonyítás. Bármely pillanatban minden – még be nem fejeződött – munkadarab megmunkálható, feltéve természetesen, hogy előzményei elkészültek már. Ezt a feltételt azonban minden megengedett ütemezésnek teljesítenie kell. Tegyük fel, hogy valamely megengedett ütemezésben a j_1 munkadarab megmunkálását a j_2 munkadarab kedvéért megszakítottuk. Innen következik, hogy j_1 nem előzménye j_2-nek, és j_2 minden előzménye befejeződött j_1 megmunkálásának ezen, megszakított szakasza előtt. Tekintsük most már azt az ütemezést, amit úgy nyerünk, hogy j_1 megmunkálásának ezen megszakított szakaszát csatoljuk a folytatásához, közvetlenül az elé, és a j_1 ezen két megmunkálása közé eső megmunkálásokat előbbre hozzuk. Ez megtehető, hiszen j_1 ezen megmunkálások után fog befejeződni, így j_1 nem előzménye egyetlen, ebbe a szakaszban befejeződő munkadarabnak sem. Ezen átrendezés mellett egyetlen munkadarab esetén sem növeltük a C_j értéket, így a célfüggvény regularitásából adódik, hogy a célfüggvény értéke nem növekedett. ■

28.2. tétel. *Ha f reguláris célfüggvény, akkor az $1 \mid d_j, prec, idle, pmtn \mid f$ és az $1 \mid d_j, prec, pmtn \mid f$, illetve az $1 \mid d_j, prec, idle \mid f$ és az $1 \mid d_j, prec \mid f$ feladatoknak van közös optimális megoldása.*

Bizonyítás. Tegyük fel, hogy a $[t_1, t_2]$ időintervallumban betervezett állásidő van. Ekkor a megmunkálások sorrendjét nem változtatva meg, a t_2 időpontban kezdődő szakaszt előrehozhatjuk $t_2 - t_1 > 0$ idővel. Ezáltal nem zavarjuk meg a kötelező megelőzések betartását, míg a C_j értékek nem nőnek, tehát a célfüggvény regularitásából következik ismét, hogy értéke nem nőtt. ■

Ebből a két tételből adódik egy újabb állítás.

28.3. tétel. *Ha f reguláris célfüggvény, akkor az $1 \mid d_j, prec, idle, pmtn \mid f$ és az $1 \mid d_j, prec \mid f$ feladatoknak van közös optimális megoldása.*

Bizonyítás. Az $1 \mid d_j, prec, idle, pmtn \mid f$ és az $1 \mid d_j, prec, idle \mid f$ feladatnak van közös optimális megoldása a 28.1. tétel szerint, az $1 \mid d_j, prec, idle \mid f$ és az $1 \mid d_j, prec \mid f$ feladatnak pedig a 28.2. tétel szerint. Ez a közös megoldás kielégíti az $1 \mid d_j, prec, idle, pmtn \mid f$ feladat feltételeit is, mert csak annyi történt, hogy nem használtuk ki a megszakítás és az ütemezett állásidő lehetőségét. ■

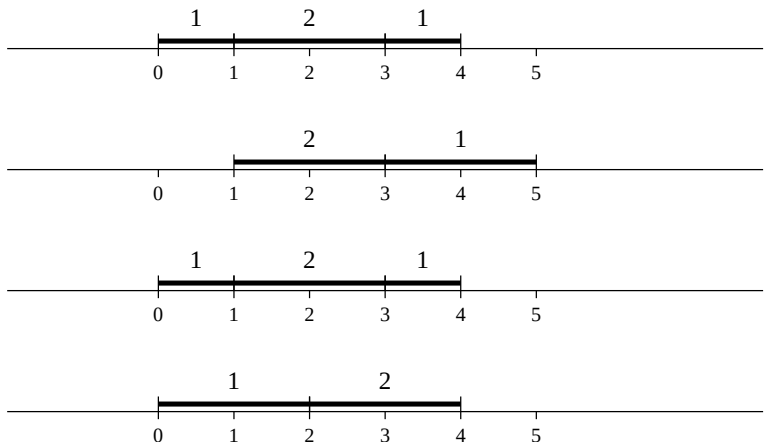
28.4. példa. Fontos rámutatni arra, hogy mennyire lényeges volt minden j esetén az $r_j = 0$ feltétel. Tekintsük azt a feladatot, amelyben csak két munkadarab van a következő adatokkal.

munkadarab	p_j	r_j	d_j
1.	2	0	4
2.	2	1	2

Észrevehető, hogy a 2. munkadarabot semmiképp sem lehet határidőre befejezni, tehát a maximális késés, azaz $T_{\max} \geq 1$. T_{max} reguláris célfüggvény. Az $1 \mid d_j, prec, idle, pmtn \mid T_{max}$, az $1 \mid d_j, prec, idle \mid T_{\max}$, az $1 \mid d_j, prec, pmtn \mid T_{\max}$ és az $1 \mid d_j, prec \mid T_{max}$ feladatok optimumát rendre a 28.4. ábrán látható ütemezések adják.

Az utolsó esetben csak egyetlen sorrend lehetséges, ami persze így optimális is. Ez az egyetlen olyan megoldás, ahol $T_{max} = 2$, minden más esetben $T_{max} = 1$. A fenti megoldások közül az első és a harmadik azonos, és a második is optimális megoldása az első feladatnak, azonban ennél mindkét munkadarab késik, míg az első megoldás esetében csak a 2. munkadarab.

Az alábbiakban néhány könnyebb feladat optimális megoldását adjuk meg. Az optimális megoldást minden esetben valamely általánosan használható, egyszerű heurisztikus szabály szolgáltatja, így a szükséges számítások mennyisége kicsi. Számos más feladat azonban ú.n. NP-teljes probléma, azaz jelenlegi tudásunk mellett csak leszámlálási módszerekkel oldható meg. Egy optimalizálási feladat megoldó eljárását akkor nevezzük leszámlálási módszernek, ha valamennyi szóba jöhető megoldást explicit módon kipróbál vagy



28.4. ábra. Eltérések az optimális megoldásban, ha a rendelkezésre állási idők különbözők.

implicit módon kizár azok közül, akik optimálisak lehetnek.

Azonban az ismertetendő heurisztikus módszerek ezekben az esetekben is képesek jó megengedett megoldások előállítására, a célfüggvény optimális értékének becslésére.

Az említett heurisztikus módszerek mindig valamilyen egyértelmű sorrendet jelentenek. Ennek a sorrendnek a jelölése: $[1], [2], \ldots, [n]$, ahol $[1]$ a sorrend első elemét, $[2]$ a sorrend második elemét jelenti stb. Ha egy bizonyításban több sorrendet vizsgálunk, akkor ezeket különböző zárójelekkel jelöljük.

28.4. tétel. *Az* $1 \mid d_j \mid L_{\max}$ *feladat egy optimális megoldását a*

$$d_{[1]} \leq d_{[2]} \leq \cdots \leq d_{[n]} \tag{28.7}$$

sorrend adja meg.

Megjegyzés. Az állításban szereplő sorrend neve a ***legkorábbi határidők sorrendje*** (EDD).

Bizonyítás. Jelöljön $< 1 >, < 2 >, \ldots, < n >$ egy tetszőleges sorrendet. Ha ez nem EDD sorrend, akkor van egy olyan l index, hogy

$$d_{<l>} > d_{<l+1>}$$

Tegyük fel, hogy $< l >$ megmunkálása a T időpontban kezdődik. A két munkadarab késése ekkor

$$L_{<l>} = C_{<l>} - d_{<l>} = T + p_{<l>} - d_{<l>} \ ,$$
$$L_{<l+1>} = C_{<l+1>} - d_{<l+1>} = T + p_{<l>} + p_{<l+1>} - d_{<l+1>} \ .$$

Tekintsük azt az $(1),(2),\ldots,(n)$ sorrendet, amely csak annyiban különbözik ettől, hogy az említett két munkadarabot felcseréljük. Így a többi munkadarab késése nem változik, míg a két felcserélt munkadarab új késése

$$\begin{aligned} L_{(l)} &= C_{(l)} - d_{(l)} = C_{(l)} - d_{<l+1>} = T + p_{<l+1>} - d_{<l+1>} , \\ L_{(l+1)} &= C_{(l+1)} - d_{(l+1)} = C_{(l+1)} - d_{<l>} = T + p_{<l>} + p_{<l+1>} - d_{<l>} . \end{aligned}$$

Könnyen látható, hogy

$$L_{<l+1>} \geq L_{(l)}, L_{(l+1)} ,$$

vagyis a célfüggvény értéke nem nőtt. Ezt a gondolatmenetet mindaddig megismételhetjük, amíg a cserékkel egy EDD sorrendet nem kapunk. ■

Az $1 \mid r_j, d_j \mid L_{max}$ feladat esetében már nem ilyen egyszerű a helyzet, mert a probléma teljes általánosságában NP-teljes. Az előbbi heurisztikus módszer következő változata speciális esetekben megoldja a feladatot.

28.5. tétel. *Ha egy, az $1 \mid r_j, d_j \mid L_{\max}$ feladatosztályhoz tartozó feladat esetében teljesül, hogy nem létezik olyan*

$$j \text{ és } l, \text{ melyekre } r_j < r_l \text{ és } d_l < d_j , \tag{28.8}$$

akkor az alábbi rekurzív módszer a feladat egy optimális megoldását adja. Tegyük fel, hogy a keresett sorrend első $j-1$ tagját már meghatároztuk. Legyen $N = \{1,\ldots,n\}\backslash\{[1],\ldots,[j-1]\}$, továbbá

$$\alpha = \max\{C_{[j-1]}, \min\{r_l : \ l \in N\}\}$$

és

$$S = \{l \in N : \ r_l \leq \alpha \} .$$

Legyen $[j] \in N$ egy olyan munkadarab, amelyre

$$d_{[j]} = \min\{ d_l : \ l \in S \} . \tag{28.9}$$

Megjegyzés. A kiválasztás logikája tehát az, hogy $[j]$ a még nem ütemezett munkadarabok közül egyike azoknak, amelyek megmunkálása a legkorábban megkezdődhet és ezek közül a legkorábbi határidővel rendelkezik.

Bizonyítás. Legyen $<1>,<2>,\ldots,<n>$ egy tetszőleges sorrend. Erre vonatkozóan legyen $N_j = \{1,\ldots,n\}\backslash\{<1>,\ldots,<j-1>\}$,

$$\alpha_j = \max\{C_{<j-1>}, \min\{r_l : \ l \in N_j\}\}$$

és

$$S_j \;=\; \{ l \in N_j : \; r_l \;\leq\; \alpha_j \} \; .$$

Nyilvánvaló, hogy az α_j értékek monoton növő sorozatot alkotnak. Ezért, ha $l \in S_j$ teljesül, akkor $l \in S_{j+1}, S_{j+2}, \ldots$ mindaddig, amíg az l munkadarab ütemezésre nem kerül. Ha az $< 1 >, < 2 >, \ldots, < n >$ sorrend nem elégíti ki a rekurzív szabályokat, akkor két eset lehetséges, nevezetesen valamely j indexre

(i) $[j]$ megmunkálása nem a legkorábban megkezdhetők közé tartozik, azaz

$$r_{[j]} \;>\; \alpha_j, \text{vagy}$$

(ii) van olyan $l \in S_j$, hogy

$$d_l \;<\; d_{[j]}, \quad r_l \;\leq\; \alpha_j \; . \tag{28.10}$$

Tekintsük először az (i) esetet. Vegyük észre, hogy α_j definíciójából következik, hogy mindig van olyan $l \in N_j$ munkadarab, amelyre $\alpha_j \geq r_l$ teljesül. Tegyük fel, hogy a vizsgált sorrendben $[k]$ az első ilyen $[j]$ után, azaz

$$r_{[j]}, r_{[j+1]}, \ldots, r_{[k-1]} \;>\; \alpha_j, \quad r_{[k]} \;\leq\; \alpha_j \; .$$

Ekkor (28.8)-ból következik, hogy

$$d_{[j]}, d_{[j+1]}, \ldots, d_{[k-1]} \;\geq\; d_{[k]} \; .$$

Tudjuk, hogy

$$L_{[k]} \;=\; C_{[k-1]} \;+\; p_{[k]} \;-\; d_{[k]} \; .$$

Innen

$$\begin{aligned} L_{[k]} \;-\; L_{[l]} \;&=\; C_{[k-1]} \;+\; p_{[k]} \;-\; d_{[k]} \;-\; C_{[l-1]} \;-\; p_{[l]} \;+\; d_{[l]} \\ &\geq\; p_{[k]} \;-\; d_{[k]} \;+\; d_{[l]} \;\geq\; p_{[k]}, \quad l = j, \ldots, k-1 \; . \end{aligned} \tag{28.11}$$

Vagyis $[k]$ késése legalább $p_{[k]}$-val nagyobb, mint a $[j], \ldots, [k-1]$ munkadarabok késése. Tekintsük most azt az $(1), (2), \ldots, (n)$ sorrendet, amelyet a

$$(l) \;=\; \begin{cases} [l], & \text{ha } \; j \;>\; l \;\; \text{vagy} \;\; j \;>\; k \; , \\ [k] \, , & \text{ha } \; j \;=\; l \; , \\ [l-1] \, , & \text{ha } \; j \;<\; l \;\leq\; k \end{cases}$$

egyenlőség definiál. Ebben a sorrendben C_l értéke $l = 1, \ldots, j-1$ esetén változatlan, $j = k+1, \ldots, n$ esetén nem nőtt, legfeljebb csökkent, mert elmarad

az $r_{[j]} - \alpha_j$ állásidő, továbbá $(l) = [l-1] \quad l = j+1, \ldots, k$ esetén pedig legfeljebb $p_{(j)} = p_{[k]}$ értékkel nőtt. Végül

$$L_{(j)} \;=\; \alpha_j \;+\; p_{(j)} \;-\; d_{(j)}\;,$$

azaz

$$L_{(j)} \;-\; L_{[k]} \;=\; \alpha_j \;-\; C_{[k-1]} \;<\; 0\;,$$

tehát ebben az esetben a késés csökkent. Innen (28.11) alapján adódik, hogy a maximális késés nem nőtt.

Tekintsük most a (ii) esetet. Ekkor válasszuk meg a k indexet úgy, hogy $[k]$ legyen a sorrendben az első, amely (28.9)-et kielégíti. Ezután az előző esetben alkalmazott gondolatmenet megismételhető. ∎

28.6. tétel. *A*

$$\frac{p_{[1]}}{w_{[1]}} \le \frac{p_{[2]}}{w_{[2]}} \le \cdots \le \frac{p_{[n]}}{w_{[n]}} \tag{28.12}$$

sorrend az $1 \,||\, \sum w_j C_j$ *feladat egy optimális megoldását adja.*

Megjegyzés. Ennek neve ***súlyozott legrövidebb megmunkálási idők sorrendje*** (SWPT).

Bizonyítás. Jelöljön $< 1 >, < 2 >, \ldots, < n >$ egy tetszőleges sorrendet. Ha ez nem SWPT sorrend, akkor van egy olyan l index, hogy

$$\frac{p_{<l>}}{w_{<l>}} > \frac{p_{<l+1>}}{w_{<l+1>}}\;. \tag{28.13}$$

Ha felcseréljük ezt a két munkadarabot, akkor a többi munkadarab befejezési ideje nem változik, vagyis a célfüggvény értékében bekövetkezett változás pusztán ettől a két munkadarabtól származik. A felcserélés előtt a

$$(C_{<l-1>} + p_{<l>})w_{<l>} \;+\; (C_{<l-1>} + p_{<l>} + p_{<l+1>})w_{<l+1>}$$

taggal járultak hozzá a célfüggvény értékéhez, míg utána a

$$(C_{<l-1>} + p_{<l+1>})w_{<l+1>} \;+\; (C_{<l-1>} + p_{<l+1>} + p_{<l>})w_{<l>}$$

mennyiséggel. A kettő különbsége

$$p_{<l>}w_{<l+1>} \;-\; p_{<l+1>}w_{<l>}\;,$$

ami pozitív, mint az azonnal látható (28.13) alapján. ∎

Ha valamennyi súly 1, vagyis pusztán az összeget minimalizáljuk, akkor

(28.12) az SPT sorrendet adja.

Amikor a tétel állításában a feladatot leírtuk, akkor a rendelkezésre állási idők azért nem szerepeltek, mert mindegyik 0 volt, amit fel is használtunk akkor, amikor egy tetszőleges sorrendben két tetszőleges, egymást követő munkadarabot felcserélhetőnek vettünk. A határidők viszont azért nem szerepeltek, mert létük nem befolyásolja a feladatot, hiszen megsérthetők, így csak a célfüggvényre lehetnek hatással.

A következő tétel elég általános módszert ad arra az esetre, amikor a sorrendet megelőzési relációk korlátozzák.

28.7. tétel. *Minden j $(j = 1, \ldots, n)$ esetén legyen f_j monoton növő függvény; legyen továbbá $f_{\max}(C_1, \ldots, C_n) = \max\{f_j(C_j) : \ j = 1, \ldots, n\}$. Jelölje N a munkadarabok halmazát, és egy tetszőleges $S \subset N$ esetén legyen $p(S) = \sum_{j \in S} p_j$ és $f^*_{\max}(S)$ azon feladatnak optimális megoldásának értékét, amelyben csak az S-beli munkadarabok szerepelnek. Végül legyen V azon munkadarabok halmaza, amelyeknek nincs rákövetkezője. Ekkor*

$$f^*_{\max}(N) \ = \ \min\{\max\{f_j(p(N)),\ f^*_{\max}(N \backslash \{j\})\} : \ \ j \in V\} \ . \qquad (28.14)$$

Megjegyzés. A tétel feltételeiből következik, hogy a célfüggvény reguláris.

Bizonyítás. Nyilvánvaló, hogy bármely megengedett sorrendben a legutolsó munkadarab csak olyan lehet, amelynek nincs rákövetkezője. Tegyük fel, hogy az optimális sorrendben a j munkadarab az utolsó. Ekkor $f^*_{\max}(N)$ értékét vagy j adja, és az ekkor $f_j(p(N))$, vagy egy másik munkadarab, és az ekkor $f^*_{\max}(N \backslash \{j\})$. ∎

A (28.14) képlet segítségével megadható egy $O(n^2)$ futási idejű algoritmus a feladat megoldására.

Mint említettük, a megszakítás lehetőségének akkor láthatjuk előnyét, amikor a rendelkezésre állási idők különbözők. Erre ad egy példát az alábbi tétel az SPT sorrend egy megfelelő általánosításával. Tekintsük az $1 \mid r_j, pmtn \mid \sum C_j$ feladatot. Legyen $T = \{r_1, \ldots, r_n\}$. Tegyük fel, hogy a t időpontig bezárólag elkészült már az ütemezés, amikor a j_t munkadarab mellett döntöttünk.

Jelölje a munkadarabok mindenkori még megmaradt megmunkálási időit $p_1', \ldots, p_n'$. A t időpont után a következő időpont, amikor dönteni kell,

$$u \ = \ \min\{t + p'_{j_t}, \min\{s \in T : \ s > t\}\} \ .$$

Azt kell tehát eldönteni, hogy az u időpontban melyik j_u munkadarab kerüljön megmunkálásra. Erre a következő kiválasztási szabályt alkalmazzuk:

$$p'_{j_u} \ = \ \min\{p'_j : \ p'_j > 0;\ r_j \leq u\} \ . \qquad (28.15)$$

A sorrend neve: ***legrövidebb megmaradt megmunkálási idők sorrendje*** (**S**hortest **R**emaining **P**rocessing **T**ime, SRPT).

28.8. tétel. *Az az SRPT ütemezés megadja az* $1 \mid r_j, pmtn \mid \sum C_j$ *feladat egy optimális megoldását.*

Bizonyítás. A bizonyítás hasonló a 28.6. tételéhez, ezért azt az Olvasóra bízzuk. ■

A feladat súlyozott változata, azaz $1 \mid r_j, pmtn \mid \sum w_j C_j$, azonban NP-teljes.

Végül egy olyan feladatot mutatunk be, ahol a célfüggvény nem reguláris: a késések és sietések összegét minimalizáljuk, feltéve, hogy a munkadarabok határideje azonos. Formális jelöléssel az $1 \mid idle, d_j = d \geq \sum_{j=1}^{n} p_j \mid \sum(E_j + T_j)$ feladatot tárgyaljuk. Itt tehát arról van szó, hogyan lehet a közös határidő körül szétosztani a munkákat úgy, hogy azok a lehető legközelebb legyenek a határidőhöz. Akkor merülhet fel ilyen vagy hasonló feladat, ha az igény erős szezonális hatást mutat, azaz a szállításoknak rövid időn belül kell megtörténniük.

Minden ütemezést egy kezdeti és egy befejező szakaszra bontunk fel. A kezdeti szakaszhoz tartoznak mindazok a munkadarabok, amelyek megmunkálása befejeződött a közös d határidőig bezárólag, a befejező szakaszhoz tartozik az összes többi munkadarab. A kezdeti szakasz, mint időintervallum, tart a szakaszhoz tartozó első munkadarab megmunkálásának kezdetétől, a szakaszhoz tartozó utolsó munkadarab megmunkálásának befejezéséig, a befejező szakasz pedig a kezdeti szakasz végétől az utolsó munkadarab megmunkálásának befejezéséig.

Most néhány lemmában leírjuk az optimális megoldások legfontosabb tulajdonságait, melyek segítségével aztán egy egyszerű algoritmus adódik azok előállítására.

28.9. lemma. *Az optimális megoldásokban az első és az utolsó megmunkálás között nincs állásidő.*

Bizonyítás. Ha van állásidő, akkor a kezdeti szakasz esetén az állásidő előtti, a befejező szakasz esetén az állásidő utáni megmunkálásokat a határidő felé tolhatjuk el úgy, hogy a sorrend változatlan marad. Ezzel a célfüggvény értéke csökken. ■

28.10. lemma. *Van olyan optimális ütemezés, amelyben egy megmunkálás pontosan a határidőkor fejeződik be.*

Bizonyítás. Tekintsünk egy olyan ütemezést, amelyben az első és az utolsó

megmunkálás között nincs állásidő, és amelyre az állítás nem igaz. Ekkor a befejező szakasz első munkadarabjának, j_b-nek, a megmunkálási időintervalluma a belsejében tartalmazza a határidőt. Toljuk el időben az egész ütemezést a következő módon:

i ha a kezdeti szakasz tartalmaz kevesebb megmunkálást, akkor j_b megmunkálásának végpontja essen egybe d-vel,

ii ha a befejező szakasz tartalmaz kevesebb megmunkálást, akkor j_b megmunkálásának kezdete essen egybe d-vel,

iii különben (i) és (ii) valamelyikét alkalmazzuk.

Az (i) és (ii) esetben csökkent a célfüggvény értéke, a (iii) esetben nem változott. ■

Az SPT sorrend fordítottjának neve LPT sorrend, azaz ***leghosszabb megmunkálási idő.***

28.11. lemma. *Bármely optimális megoldásban a megmunkálások a kezdeti szakaszban LPT, a befejező szakaszban SPT sorrendben vannak.*

Bizonyítás. Ha két szomszédos munkadarab ütemezése fordított, akkor ezeket felcserélve a célfüggvény értéke csökken. ■

28.12. lemma. *Legyen $(1), (2), \ldots, (n)$ egy olyan ütemezés, amelyben az első és az utolsó megmunkálás között nincs állásidő és amelyben egy megmunkálás pontosan a határidőkor fejeződik be. Ha a kezdeti szakaszban α, a befejező szakaszban β megmunkálás van $(\alpha + \beta = n)$, akkor a célfüggvény értéke*

$$\sum_{j=1}^{\alpha}(j-1)p_{(j)} \; + \; \sum_{j=1}^{\beta}(\beta - j + 1)p_{(\alpha+j)} \; . \tag{28.16}$$

Bizonyítás. A kezdeti szakaszban (α) sietése 0, $(\alpha-1)$ sietése $p_{(\alpha)}$, $(\alpha-2)$ sietése $p_{(\alpha-1)} + p_{(\alpha)}$ stb. Tehát $p_{(\alpha)}$ pontosan $\alpha - 1$ munkadarab sietésében szerepel, $p_{(\alpha-1)}$ $\alpha - 2$ munkadarabéban stb. Innen kapjuk az első összeget. Hasonlóan járhatunk el a befejező szakasz esetén. $(\alpha+1)$ késése $p_{(\alpha+1)}$, $(\alpha+2)$ késése $p_{(\alpha+1)} + p_{(\alpha+2)}$ stb. Tehát $p_{(\alpha+1)}$ összesen β munkadarab késésében szerepel, $p_{(\alpha+2)}$ $\beta - 1$ munkadarabéban stb. Ez adja a második összeget. ■

(28.16)-ból még kiolvasható a következő állítás.

28.13. lemma. *Az előző lemma jelöléseit használva van olyan optimális megoldás, amelyben*

$$| \alpha - \beta | \leq 1 \; .$$

Bizonyítás. Legyen $(1), (2), \ldots, (n)$ egy olyan ütemezés, amelyre a lemma állítása nem igaz. Ha $\alpha \geq \beta + 2$, akkor toljuk el az ütemezést úgy, hogy (α) legyen a befejező szakasz első munkadarabja. Ekkor (28.16) első összege $(\alpha - 1)p_{(\alpha)}$-val csökkent, a második összege $(\beta + 1)p_{(\alpha)}$-val nőtt, vagyis a teljes változás $(\beta + 2 - \alpha)p_{(\alpha)} \leq 0$. ■

Innen a következő megoldási módszer adódik. Tegyük fel, hogy az indexek szerinti sorrend egyben az LPT sorrend is. Ekkor az utolsó lemma feltételeinek eleget tevő optimális megoldás esetén (28.16)-ban p_1 szorzója 0, p_2 és p_3 szorzója 1, általában p_{2l} és p_{2l+1} szorzója l. Vagyis szabadon dönthetünk afelől, hogy a $2l$ és a $2l+1$ munkadarab közül melyik legyen elölről az $(l+1)$-edik, illetve hátulról az l-edik.

Gyakorlatok

28.3-1. Konkrét példa megadásával bizonyítsuk be, hogy a 28.4. tételben szereplő feltétel nem szükséges, azaz a határidők szerinti növekvő rendezés nem szükséges feltétele az $L_{\max}$ optimalizálásának.

28.3-2. Mutassuk meg, hogy a 28.6. tételben szereplő súlyozott legrövidebb megmunkálási idők sorrendje az adott esetben nem csak elégséges, hanem szükséges feltétel is.

28.4. Ütemezési problémák párhuzamos berendezéseken

Az ide tartozó legegyszerűbb feladatok esetében egy m gépből álló homogén gépcsoportról van szó, azaz minden gép egyforma. Így egy munkadarab megmunkálási ideje minden gépen azonos. A feladatok még ebben az esetben is – kevés kivételtől eltekintve – NP-teljesek, ezért nagy jelentősége van a heurisztikus eljárásoknak. Először a polinomiális eredményeket ismertetjük, és utána tárgyaljuk a heurisztikus módszereket.

28.14. tétel. *A következő ütemezés a $P \mid overlap \mid \sum C_j$ feladat egy optimális megoldását adja. Legyen $[1], [2], \ldots, [n]$ egy SPT sorrend. Ekkor minden gépen pontosan egyforma az ütemezés, és minden gépen a j-ediknek ütemezett megmunkálás a $[j]$ megmunkálás $(1/m)$-ed része.*

Bizonyítás. Tekintsünk egy tetszőleges ütemezést és legyen $(1), (2), \ldots, (n)$ a megmunkálások befejezési sorrendje. Legyen $1 \leq j_1 < j_2 \leq n$. Tegyük fel, hogy (j_1) egy h_1 hosszú megmunkálása az m_1 gépen később kezdődik a t_1 időpontban, mint a (j_2) egy h_2 hosszú megmunkálása az m_2 gépen a t_2 időpontban, azaz $t_1 > t_2$. Az m_1 és m_2 gép nem feltétlenül különböző. Ekkor a két megmunkálás egy $\min\{h_1, h_2\}$ szakasza felcserélhető, és így az összes

többi, valamint a (j_2) megmunkálás befejezése nem változik, (j_1) befejezése pedig vagy változatlan, vagy előbbre kerül. ■

A következő két tétel a témakör egyik legkorábbi dolgozatából származik.

28.15. tétel. *A $P \mid pmtn \mid f$ feladatban az átfutási idő legalább*

$$C^* = \max\left\{\frac{1}{m}\sum_{j=1}^{n} p_j, \max\{p_j : j = 1, \ldots, n\}\right\}. \tag{28.17}$$

Bizonyítás. Mivel az átlapolás nem megengedett, ezért egy megmunkálás akárhány részre is legyen felosztva, ezek a részek nem fedhetik át egymást, azaz minden j esetén $p_j \leq C_j$. Másfelől a teljes átfutási idő nem lehet rövidebb annál, mint amit úgy kapunk, hogy a megmunkálásokat egyenletesen osztjuk szét a gépek között. ■

28.16. tétel. *A $P \mid pmtn \mid C_{\max}$ feladat optimális célfüggvényértéke a (28.17) képletben adott mennyiség.*

Bizonyítás. Egyenként készítünk ütemezést a gépekre. Minden gépet a C^* időpontig terhelünk le munkával, hacsak el nem fogytak a megmunkálások. A megmunkálásokat tetszőleges sorrendben vesszük. Ha a soron következő megmunkálás még befér az éppen ütemezendő gép C^* időkorlátjába, akkor egyetlen egységben ezen a gépen végezzük el a megmunkálást az erre a gépre már ütemezett megmunkálások után közvetlenül. Ha a teljes megmunkálás nem fér be az időkorlátba, akkor a megmunkálás akkora részét ütemezzük erre a gépre, hogy ezzel az időkorlátot elérjük, a maradék rész pedig a következő gép első megmunkálása. Az időkorlát megválasztása miatt ez a két rész nem nyúlhat egymásba. ■

Vegyük észre, hogy az így készített ütemezésben a megszakítások száma legfeljebb $m-1$, és minden megmunkálást legfeljebb egyszer szakítunk meg. A minimálisan szükséges megszakítások számának meghatározása már NP-teljes probléma. Ha egy megmunkálást megszakítottunk, és egy része a következő gépre került, akkor a tényleges termelés során a megmunkálás ezzel az átvitt résszel kezdődik.

A problémák úgy is felfoghatók ebben a körben, mint két részfeladat együttese, először meg kell találni a megmunkálások gépek közötti optimális felosztását, majd az egyes gépeken a legjobb sorrendet. Innen az egy gép esetére mondottak alapján azonnal adódik az alábbi tétel.

28.17. tétel. *Az $R \mid\mid \sum C_j$, illetve az $R \mid\mid \sum w_j C_j$ feladat optimális megoldásában a megmunkálások minden gépen SPT, illetve SWPT sorrendben vannak.*

A következő tétel bizonyításában $O(n^3)$ művelet segítségével vezetjük vissza az ütemezési feladatot a polinomiális hozzárendelési feladatra.

28.18. tétel. *Az $R \,||\, \sum C_j$ feladat polinomiális idő alatt megoldható.*

Bizonyítás. Tekintsünk egyetlen m_l gépet. Tegyük fel, hogy ezen $n_l (n_l \leq n)$ megmunkálást akarunk végezni, az $(1), (2), \ldots, (n_l)$ sorrendben. Ekkor a célfüggvény értéke ezen a gépen

$$\sum_{j=1}^{n_l} (n_l - j + 1) p_{l(j)} \ , \tag{28.18}$$

ahogy ezt például a 28.12. lemmában is láttuk. Vezessük be a következő bináris változókat:

$$x_{(ik)j} = \begin{cases} 1, & \text{ha a } j\text{-edik megmunkálást az } i\text{-edik gépen végezzük,} \\ & \text{sorrendben hátulról a } k\text{-adiknak ,} \\ 0 & \text{különben .} \end{cases} \tag{28.19}$$

Itt k értéke 1 és n közötti egész. Két további feltételcsoport adódik még:
(i) minden megmunkálást el kell végezni,
(ii) minden gép egyszerre csak egy megmunkálással foglalkozhat.

Az elsőt az alábbi egyenletekkel, a másodikat pedig az azokat követő egyenlőtlenségekkel írhatjuk elő:

$$\sum_{i=1}^{m} \sum_{k=1}^{n} x_{(ik)j} = 1, \quad j = 1, \ldots, n \ , \tag{28.20}$$

$$\sum_{j=1}^{n} x_{(ik)j} \leq 1, \quad i = 1, \ldots, m, \quad k = 1, \ldots, n \ . \tag{28.21}$$

Ha $x_{(ik)j} = 1$, akkor a j megmunkálás költsége a célfüggvényben kp_{ij}, ezért a teljes költség

$$\sum_{i=1}^{m} \sum_{k=1}^{n} \sum_{j=1}^{n} k p_{ij} x_{(ik)j} \ , \tag{28.22}$$

amit minimalizálni kell. Az így kapott (28.19)–(28.22) feladat egy hozzárendelési probléma, ami polinomiális idő alatt megoldható. ∎

Érdemes szemügyre venni a bizonyításban szereplő transzformáció „költségeit". Az eredeti feladat mérete $m \times n$ volt. A jelenlegié pedig $(mn) \times n$

(az átfogalmazott feladatban az i, k indexpár egyetlen index szerepét játssza). A hozzárendelési feladat megoldásához szükséges műveletek száma $O(n^3)$ marad.

Reguláris célfüggvények mellett bizonyos esetekben a dinamikus programozás természetes módon adódik, mint egy pontos megoldási módszer. A 28.17. tétel azt mondja, hogy az $R \,||\, \sum C_j$ és a $R \,||\, \sum w_j C_j$ feladatok esetében, ha megtörtént a megmunkálások felosztása az egyes gépek között, akkor az egyes gépeken a sorrend kötött. Hasonló eredmény a következő.

28.19. tétel. *Az $R \,||\, \sum T_j$ feladatnak van olyan optimális megoldása, amelyben a megmunkálások minden gépen a határidők szerinti növekvő sorrendben vannak.*

Bizonyítás. Ha a feltétel nem teljesül, akkor van legalább egy gép, amelyen az ütemezés szerint egy későbbi határidejű megmunkálás közvetlenül megelőz egy korábbi határidejű megmunkálást. E két megmunkálást felcserélve a célfüggvény értéke nem növekszik. ■

Hasonló állítás mondható az $U_{\max}$ célfüggvényről, azaz a megengedett ütemezés kereséséről (lásd a 28.4. tételt). Fontos megjegyezni, hogy ebben lényeges szerepet játszik, hogy minden rendelkezésre állási idő azonos. A $C_{\max}$ célfüggvény esetén nemcsak egy, hanem bármely sorrend rendelkezik a tételbeli tulajdonsággal, mert a megmunkálásoknak gépekre való felosztása már egyértelműen meghatározza a teljes átfutási időt. Dinamikus programozással olyan feladatokat lehet viszonylag könnyen megoldani, amelyekre a következő feltétel teljesül:

A megmunkálások indexsorrendje olyan sorrend, hogy van olyan optimális megoldás, hogy minden gépen a megmunkálások ebben a sorrendben vannak. (S)

Nem jelenti az általánosság megszorítását, hogy az indexsorrendről feltételeztük, hogy a kívánt tulajdonságú.

Vizsgálni fogjuk mind az $f_{\max}$, mind a $\sum f_j$ típusú célfüggvényt. A tárgyalás nagyon hasonló, ezért az alábbi közös jelölést használjuk: $F_j(t_1, \ldots, t_m)$ az optimális célfüggvényérték, ha csak az indexsorrendben első j megmunkálást ütemezzük, és a gépek állásidő nélkül dolgozva munkájukat a $t_1, \ldots, t_m$ időpontokban fejezik be.

28.20. tétel. *Ha az (S) feltétel teljesül, akkor az $R \,||\, \sum f_j$ feladat esetén*

$$F_j(t_1, \ldots, t_m) = \min\{f_j(t_i) + F_{j-1}(t_1, \ldots, t_i - p_{ij}, \ldots, t_m) : i = 1, \ldots, m\} \quad (28.23)$$

Bizonyítás. Most utoljára a j megmunkálást ütemezzük, és ez az (S) feltétel miatt valamelyik gépen utolsónak fejeződik be az eddig ütemezettek közül. Ha ez történetesen az i gép, akkor ott a j-t közvetlenül megelőző megmunkálás a $t_i - p_{ij}$ időpontban fejeződik be. Így felmerül az előző $j-1$ megmunkáláson $F_{j-1}(t_1, \ldots, t_i - p_{ij}, \ldots, t_m)$ költség, továbbá $f_j(t_i)$ költség a j megmunkáláson. ■

28.21. tétel. *Ha az (S) feltétel teljesül, akkor az $R \,||\, f_{\max}$ feladat esetén*

$$F_j(t_1, \ldots, t_m) = \min\{\max\{f_j(t_i), F_{j-1}(t_1, \ldots, t_i - p_{ij}, \ldots, t_m)\} : \; i = 1, \ldots, m\} \tag{28.24}$$

Bizonyítás. A bizonyítás hasonló az előző tételéhez. ■

A számítási idő $\Theta(mnC^m)$, ahol C a teljes átfutási idő felső korlátja. Jobban korlátozza a módszer alkalmazását, hogy a szükséges tárolási kapacitás $\Theta(mC^m)$. Amennyiben az F függvényt tartalmazó kitöltendő táblákat explicit módon tároljuk, akkor sem a számítás mennyisége, sem a szükséges memória nagyságrendje nem csökkenthető.

A heurisztikus módszerekkel legtöbbet a $P \,||\, C_{\max}$ feladatot vizsgálták, mert ütemezési szempontból egyszerű, de NP-teljes. ***Listás ütemezésnek*** nevezik azt a heurisztikus eljárást, amely a következő két lépésből áll:

1. Meghatározzuk a megmunkálások valamely sorrendjét (lista).

2. Az adott sorrendben véve a megmunkálásokat, az éppen soron következőt az elsőnek megüresedő gépre rakjuk.

Ha L jelöli az adott listát, akkor $C(L)$ a heurisztikus megoldás célfüggvényértéke, míg az optimális megoldásé C^*, és a $C(L)/C^*$ hányadost vizsgáljuk.

28.22. tétel. *Tetszőleges L lista esetén*

$$\frac{C(L)}{C^*} \leq 2 - \frac{1}{m} \,. \tag{28.25}$$

Bizonyítás. Ha sikerül teljesen egyenletesen szétosztani a gépek között a megmunkálásokat, akkor az átfutási időre

$$\frac{1}{m} \sum_{j=1}^{n} p_j$$

adódik. Másfelől legalább egy gépen legalább annyi időt fel kell használnunk, mint a leghosszabb megmunkálási idő, hiszen az átlapolás nem megengedett,

azaz

$$\max\{p_j \ : \ j = 1, \ldots, n\}$$

egy másik alsó korlát C^*-ra. Legyen k az utolsónak befejeződő megmunkálás. Ez a $t = C(L) - p_k$ időpontban kezdődött el. Mivel az ütemezésben nincsenek felesleges állásidők, így a t időpontig valamennyi gép folyamatosan dolgozott. Ezért

$$t \ \leq \ \frac{1}{m} \sum_{j \neq k} p_j \ .$$

Végül a

$$C(L) \ = \ t \ + \ p_k \ \leq \ \frac{1}{m} \sum_{j \neq k} p_j \ + p_k = \frac{1}{m} \sum_{j=1}^{n} p_j \ + \ \frac{m-1}{m} p_k \leq \left(2 - \frac{1}{m}\right) C^* \tag{28.26}$$

egyenlőtlenséget kapjuk. ■

A korlát éles. Erre vonatkozóan lásd a *28-5* feladatot.

Számos heurisztikus eljárás pontosságára ismert a fentinél jobb felső korlát.

28.23. tétel.

$$\frac{C(LPT)}{C^*} \leq \frac{4}{3} - \frac{1}{3m} \ . \tag{28.27}$$

Bizonyítás. Indirekt módon feltesszük, hogy a tétel állítása nem igaz. Feltehető, hogy $m \geq 2$. Az első részállítás, amit bebizonyítunk, hogy feltehető az is, hogy az LPT sorrend szerinti ütemezésben utolsónak az utolsó, azaz a legrövidebb megmunkálási idejű megmunkálás fejeződik be. Tekintsünk valamely rögzített m mellett egy olyan F ütemezési feladatot, amelyre (28.27) nem igaz, és n értéke minimális. Ha az ütemezés szerint egy $p < n$ indexű megmunkálás fejeződik be utolsónak, akkor tekintsük azt az F' ütemezési feladatot, amelyet F-ből úgy kapunk, hogy az LPT sorrendből csak az első p megmunkálást tartalmazza. Itt az LPT sorrend szerinti átfutási idő nem változott, míg az optimális átfutási idő nem nőtt. Ezért a tétel állítása erre az F' feladatra sem lehet igaz, ami ellentmond n minimalitásának.

Innen (28.26) alapján kapjuk, hogy erre a feladatra

$$\frac{4}{3} - \frac{1}{3m} \ < \ \frac{C(LPT)}{C^*} \ \leq \ \frac{1}{mC^*} \sum_{j=1}^{n} p_j + \frac{(m-1)p_n}{mC^*} \leq 1 + \frac{(m-1)p_n}{mC^*} \ ,$$

ahonnan adódik, hogy

$$p_n > \frac{C^*}{3} .$$

Ez természetesen az LPT sorrend miatt valamennyi megmunkálási időre vonatkozik, azaz az optimális megoldásban legfeljebb két megmunkálás lehet egy gépen.

Tekintsünk egy tetszőleges olyan U ütemezést, ahol minden gépen legfeljebb két megmunkálás van. Megadunk néhány olyan átrendezést, amely nem növeli meg az átfutási időt. Legyen két – az i gépre ütemezett – megmunkálás ideje t_{i1} és t_{i2}, míg egy másik j gép esetén t_{j1} és t_{j2}, illetve ha ezen csak egy megmunkálás van, akkor t_j.
(i) Ha $t_{i1} > t_{j1}$ és $t_{i2} > t_{j2}$, akkor i_1 és j_1 felcserélésével a két gépen az átfutási idő csökken, hiszen $\max\{t_{i1} + t_{j2}, t_{j1} + t_{i2}\} < t_{i1} + t_{i2}$.
(ii) Ha $t_{i1} > t_j$, akkor az i_2 munkát átrakva a j gépre a két gépen szintén csökken az átfutási idő.
(iii) Az i_1 és i_2 munkák sorrendjének felcserélése nem változtatja meg az i gépen az átfutási időt.

Bármely lehetséges U ütemezésből kiindulva és elvégezve az összes lehetséges fenti átalakítást, egy olyan V ütemezést nyerünk, amelyben az átfutási idő nem hosszabb, mint U-ban, és rendelkezik a következő tulajdonságokkal:
(a) ha bármely i, j gép esetén mindkét gépen két megmunkálás van, akkor (i) alapján

$$t_{i1} > t_{j1} \quad \text{esetén} \quad t_{i2} \leq t_{j2} ,$$

(b) minden i, j gép esetén, ha az i gépen két megmunkálás van, a j gépen egy, akkor (ii) alapján

$$t_j \geq t_{i1}, t_{i2} ,$$

(c) minden i gép esetén, ha az i gépen két megmunkálás van, akkor (iii) alapján

$$t_{i1} \geq t_{i2} ,$$

(d) mivel a gépek sorrendje nem befolyásolja az átfutási időt, ezért feltehető, hogy a gépek indexsorrendje szerint csökken az első megmunkálási idő,
(e) az egy megmunkálást végző gépek indexei (b) és (d) alapján kisebbek a két megmunkálást végzőkénél,
(f) a két megmunkálást végző gépek indexsorrendje szerint nő a második – azaz a nem nagyobb – megmunkálási idő.

Tehát van olyan optimális megoldás, ami az (a)–(f) tulajdonságokkal rendelkezik, ezek a tulajdonságok azonban egyértelműen meghatározzák az ütemezést, ami nem más, mint az LPT lista szerinti ütemezés. Ez pedig ellentmond a

$$\frac{C(LPT)}{C^*} > \frac{4}{3} - \frac{1}{3m} \geq 1 \,.$$

egyenlőtlenségnek. ■

A $P \mid\mid C_{\max}$ problémához nagyon hasonló, mintegy annak duálja a ládapakolási probléma. Egyforma kapacitású ládákba kívánunk belerakni tárgyakat, ahol a tárgyak méretét egyetlen számmal lehet jellemezni. Az egy ládába rakott tárgyak összmérete a méretek összege. Meghatározandó a minimálisan szükséges ládák száma. Visszatérve a párhuzamos berendezésekhez, tegyük fel, hogy a munkákat egy meghatározott idő, például egy műszak alatt akarjuk elvégezni. Ekkor úgy merül fel a kérdés, hogy hány gépre van szükség.

FF$(n, \mathbf{p}, K)$

```
1  for i = 1 to n
2      z_i = K
3  m = 1
4  for i = 1 to n
5      j = 1
6      while z_j < p_i
7              j = j + 1
8      z_j = z_j − p_i
9      if j > m
10         m = j
11 return m
```

A ládapakolási feladat megoldására egy mohó heurisztikus eljárás a FF. Legyen K a ládák kapacitása. Feltesszük, hogy nincs olyan tárgy, aminek mérete nagyobb lenne, mint K, hiszen ekkor nincs megengedett megoldás. Ha ez a feltétel teljesül, és a tárgyak száma n, akkor legfeljebb n ládát kell használnunk. A FF algoritmus sorra veszi a tárgyakat, és mindegyiket az első (azaz a legkisebb indexű) olyan ládába rakja, ahova belefér. A következő pszeudokódban a bemenő adat a tárgyak n száma, a tárgyak méretét tartalmazó $\mathbf{p}$ vektor, valamint a ládák K kapacitása, kimenő adat a felhasznált ládák m száma. A kódban z_i az i-edik láda még fel nem használt kapacitása.

A FF és más ládapakolási algoritmusok elemzése megtalálható a harmadik kötetben, a 34.4. alfejezetben.

A FF algoritmust akkor tudjuk felhasználni, ha tudunk felső korlátot adni a maximálisan szükséges kapacitásra, azaz az átfutási időre.

28.24. tétel. *Bármely $P \mid\mid C_{\max}$ feladat esetén az optimális C^* átfutási időre teljesül, hogy*

$$C^* \leq \max\left\{\frac{2}{m}\sum_{j=1}^{n} p_j, \max\{p_j : j = 1, \ldots, n\}\right\} . \tag{28.28}$$

Bizonyítás. Legyen C a (28.28) egyenlőtlenség jobboldalán szereplő mennyiség. Indirekt módon tegyük fel, hogy a tétel állítása nem igaz. Alkalmazzuk a FF algoritmust úgy, hogy a ládák kapacitása C és a megmunkálások a megmunkálási idők szerinti, azaz a ládapakolási feladat nyelvén a tárgyak mérete szerinti csökkenő sorrendben vannak. Ha a tétel állítása hamis, akkor az algoritmus kénytelen valamely k tárgyat az $(m+1)$-edik ládába rakni. Nyilvánvalóan $k \geq m+1$. Mivel $p_k \leq p_{k-1} \leq \cdots \leq p_1$, ezért minden láda legalább p_k-ig fel van töltve. Mivel azonban p_k nem fér az első m ládába, ezért ezeket már jobban feltöltöttük, mint $C/2$. Innen adódik, hogy

$$\sum_{j=1}^{n} p_j \;>\; \frac{mC}{2} \;\geq \sum_{j=1}^{n} p_j \;,$$

ami ellentmondás. ∎

A bizonyításból érezni lehet, hogy a FF algoritmust úgy lesz célszerű alkalmazni, ha előtte a megmunkálási időket csökkenő sorrendbe rakjuk, azaz az LPT sorrendet használjuk. Tegyük fel, hogy megbecsüljük a szükséges átfutási időt, és erre alkalmazzuk a FF algoritmust. Természetesen egy jó megengedett megoldáshoz jutunk, ha az eljárás talál ilyet, azaz nem használ több ládát (gépet), mint amennyit felhasználhat. De ha nem talál megengedett megoldást, akkor lényegében nincs semmi a kezünkben, csak az a durva közelítő megoldás, ami az előző tétel bizonyításából származik. Viszont a 28.15. tételből ismerünk egy alsó korlátot az átfutási időre. Az alsó és felső korlátból kiindulva, logaritmikus kereséssel egy pontosabb korlátot lehet kapni. Ez az FF ismételt alkalmazását jelenti, ahol csökkentjük a felső korlátot, ha találtunk megengedett megoldást, és növeljük az alsó korlátot, ha nem. Ne feledjük, hogy a 28.24. tétel bizonyítása mindenképpen ad egy megengedett megoldást, a most vázolt algoritmus ezt kívánja javítani.

A fentiekben felvázolt algoritmus leírásánál az alábbi jelöléseket alkalmazzuk:

K_a	az aktuális alsó korlát,
K_f	az aktuális felső korlát,
K	a pillanatnyi (azaz a kipróbálás alatt álló) korlát,
s	a megteendő iterációs lépések száma.

Továbbá részeljárásként működik némi módosítással a MOHÓ algoritmus. Mivel a párhuzamos berendezések esetén nem az a kérdés, hogy hány berendezést kell használnunk, ha K időn belül végezni akarunk, hanem az, hogy K időn belül m géppel végezni tudunk-e, ezért a FF algoritmus módosítását megvalósító MF algoritmus mind a K kapacitáskorlátot, mind a gépek m számát paraméterként kapja meg.

Az algoritmus eredeti angol neve „Multi-Fit”, amit rövidítve megtartunk (magyar jelentése: többszörös beillesztés).

MF(m, K)

```
1  K_a = max{(1/m) Σ_{j=1}^{n} p_j, max{p_j : j = 1,...,n}}
2  K_f = max{(2/m) Σ_{j=1}^{n} p_j, max{p_j : j = 1,...,n}}
3  rendezzük a megmunkálásokat a megmunkálási idők szerinti csökkenő sorrendbe
4  for i = 1 to s
5      K = (K_a + K_f)/2
6      MOHÓ(K, m, elég)
7      if elég == IGAZ
8          K_f ← K
9          else K_a = K
10 return K_a és K_f
```

Annak eldöntésére, hogy az adott esetben m darab K kapacitású láda elegendő-e, a MOHÓ eljárást használjuk. Ennek eredménye az *elég* logikai változó, melynek értéke akkor IGAZ, ha MOHÓ talált megengedett megoldást.

MOHÓ(K, m)

```
1  elég = IGAZ
2  for i = 1 to m
4      z_i = K
5  for i = 1 to n
6      j = 1
7      while z_j < p_i
8              j = j + 1
9              if j > m
10                 elég = HAMIS
11                 return elég
12             z_j = z_j − p_i
```

n tárgy méretének összehasonlításos rendezéséhez $O(n \lg n)$ lépés elegendő. Az FF algoritmus beépített változata minden tárgyra legfeljebb m

lépést tesz meg, ez $O(nm)$ számítási igény minden korlátra. Tehát az algoritmus műveleti igénye $O(n \lg n + nms)$. A számítási igény ennyi is lesz abban az esetben, ha az első $m-1$ tárgy olyan nagy, hogy egyenként kitöltenek egyet-egyet az első $m-1$ gép közül úgy, hogy más megmunkálás már nem fér melléjük.

28.5. példa. Tekintsük azt a feladatot, amelyben $n = 5$, $m = 2$ és a megmunkálási idők rendre 7, 7, 5, 5, 5. Ekkor a listás ütemezés a két leghosszabb megmunkálást külön gépre rakja, majd mindkét gépre tesz egy-egy 5 idejűt, végül az utolsó megmunkálást kénytelen az első gépre rakni. Az átfutási idő ekkor 17. Az optimális megoldás viszont az, amikor az azonos megmunkálási idejű munkák ugyanazon a gépen vannak, mert így az átfutási idő csak 15. A 28.23. tételben megadott felső korlát most 7/6. Ezzel megszorozva az optimális megoldás értékét, vagyis 15-öt, 17.5-et kapunk, vagyis az eljárás lényegében a hibahatáron dolgozik.

Alkalmazzuk most ugyanerre a feladatra az MF algoritmust, $m = 2$ mellett.

A kezdeti alsó korlát 14.5, amit 15-re lehet felkerekíteni.

A kezdeti felső korlát 29, innen az induló korlát 22-nek adódik.

Ekkor az első gépen a megmunkálási idők rendre $7, 7, 5$, míg a második gépen $5, 5$. Mivel találtunk megoldást, ezért az alsó és a pillanatnyi felső korlát átlagát kell venni, ami kerekítéssel 18. Erre már az optimális megoldást kapjuk.

28.6. példa. Legyen most $n = 6$, $m = 2$ és a megmunkálási idők 10, 7, 7, 6, 6, 4. A listás ütemezés az első gépre a 10, 6, 4 megmunkálási idejű munkákat rakja, míg a másodikra a 7, 7, 6 idejűket. Mivel mindkét gépen 20 az átfutási idő, ezért ez az optimális megoldás. Ha az MF algoritmust futtatjuk, akkor ott a kezdeti alsó korlát éppen ez a 20 lesz. Azonban 20 vagy annál nagyobb korlát esetén az eljárás mindig összerakja a 10 megmunkálási idejű munkát legalább az egyik 7 idejűvel, így sohasem fogja megtalálni az optimális megoldást. A két kezdeti korlát 20 és 40. Ezért az eljárás a 30 korláttal indul. Ehhez van megoldás: első gép 10, 7, 7, 6, a második gép 6, 4. Így a következő korlát 25. Ekkor a 10, 7, 7, illetve 6, 6, 4 megoldást kapjuk. Innen a következő korlát 22. Az ehhez tartozó megoldás 10, 7, 4 és 7, 6, 6. Itt az átfutási idő csak 21. Ezért a megoldás nem változik a következő korlátra sem, ami éppen ennyi. Innen már a 20 korlát következik, de mint azt említettük, ehhez nincs megoldás.

Ahhoz, hogy az algoritmus pontosságát elemezni tudjuk, vissza kell térni a ládapakolási feladathoz.

Ha a ládák méretét minden határon túl növeljük, akkor egyre kevesebb ládára lesz szükség, egészen addig, amíg a ládák mérete el nem éri a tárgyak méreteinek összegét, mert ettől kezdve csak egy ládát kell használnunk. Az ütemezési probléma megoldása szempontjából bennünket rögzített számú láda felhasználása érdekel. Vegyük észre, hogy a 28.15. tételben megadott alsó korlát és a 28.24. tételbeli felső korlát között csak egy legfeljebb 2-es tényező

van. Ez az alábbi állítást sugallja: ha megnöveljük a ládák méretét a minimálisan szükségesnek legfeljebb a kétszeresére, akkor a FF algoritmus talál megengedett megoldást, feltéve, hogy a tárgyak méret szerint csökkenően rendezettek. Az alábbiakban azt mutatjuk be, hogy ez az állítás lényegében igaz. Ezen felül az is teljesül, hogy az említett 2 korlátnál, ami rosszabb volna, mint a már elért 4/3 korlát, sokkal jobb az eljárás.

Legyen $\mathbf{p}$ a tárgyak méreteit tartalmazó vektor, továbbá $C_m^*(\mathbf{p})$ a ládák azon legkisebb mérete, amely mellett még van olyan pakolás, amely legfeljebb m ládát használ fel. Ne feledjük, hogy véges sok tárgyat csak véges sok féleképpen lehet szétosztani a ládák közt, ezért $C_m^*(\mathbf{p})$ nem csak infimumként, hanem minimumként is létezik. Mivel itt heurisztikus eljárásról van szó, így egyáltalán nem biztos, hogy a FF algoritmus bármely $\mathbf{p}$ és m esetén talál m ládát használó megoldást a $C_m^*(\mathbf{p})$ korláthoz. Felmerül a kérdés, hogy hányszorosra kell megnövelni a ládák méretét $C_m^*(\mathbf{p})$-hez képest, hogy az algoritmus már találjon ilyent. Legyen $\mathrm{FF}(n, \mathbf{p}, K)$ az FF algoritmus megoldása által használt ládák száma. Jelölje r azt a tényezőt, ahányszorosára a ládák méretét növeltük. Legyen r_m az a legkisebb tényező, amely bármely feladat esetén biztosítja, hogy algoritmusunk legfeljebb m ládát használ, azaz

$$r_m = \inf\{r : \text{minden } \mathbf{p} \text{ esetén } \mathrm{FF}(n, \mathbf{p}, rC_m^*(\mathbf{p})) \le m\} .$$

28.25. tétel. *Bármely* $\mathbf{p}$ *és* $r \ge r_m$ *esetén* $\mathrm{FF}(n, \mathbf{p}, rC_m^*(\mathbf{p})) \le m$.

Bizonyítás. Legyen először $r = r_m$. Ha most a tétel állítása nem igaz, akkor van egy olyan $\mathbf{p}$ méretvektor, hogy $\mathrm{FF}(n, \mathbf{p}, r_m C_m^*(\mathbf{p})) > m$. Ha egy tárgyat nem tudunk betenni egy ládába, annak az az oka, hogy a ládába tett tárgyak összmérete ezen tárggyal együtt meghaladná az $r_m C_m^*(\mathbf{p})$ korlátot. Tekintsük ezeket, az algoritmus végrehajtása során fellépő, az adott korlátnál nagyobb mennyiségeket. Legyen ezek közül a minimális $r_m C_m^*(\mathbf{p}) + \varepsilon$. Mivel csak véges sok mennyiségről van szó, ezért $\varepsilon > 0$. Legyen C olyan korlát, hogy $r_m C_m^*(\mathbf{p}) < C < r_m C_m^*(\mathbf{p}) + \varepsilon$. Ekkor C megválasztásából következik, hogy $\mathrm{FF}(n, \mathbf{p}, C) > m$, ami $r = C/C_m^*(\mathbf{p}) > r_m$ mellett ellentmond r_m megválasztásának.

Legyen most már $r > r_m$. Tegyük fel, hogy van olyan $\mathbf{p}$ méretvektor, hogy $\mathrm{FF}(n, \mathbf{p}, rC_m^*(\mathbf{p})) > m$. Tekintsük azt a $\mathbf{q}$ méretvektort, amelyben a ládák kapacitását $(r/r_m)C_m^*(\mathbf{p})$-ra növeltük, továbbá véges sok, a $\mathbf{p}$ vektor legkisebb eleménél kisebb tárgyat adunk a $\mathbf{p}$ tárgyaihoz úgy, hogy ezekkel a kis tárgyakkal kiegészíthető legyen $\mathbf{p}$ optimális megoldása úgy, hogy a megnagyobbított ládák pontosan fel legyenek töltve. Ez azt jelenti, hogy $C_m^*(Q) = (r/r_m)C_m^*(\mathbf{p})$. Ekkor azonban $r_m C_m^*(Q) = rC_m^*(\mathbf{p})$. Másfelől viszont feltettük, hogy $\mathrm{FF}(n, \mathbf{p}, rC_m^*(\mathbf{p})) = \mathrm{FF}(n, \mathbf{p}, r_m C_m^*(Q)) > m$, ami ellentmond a bizonyítás első felében r_m-re igazoltaknak. ∎

Ebből a tételből már azonnal következik annyi, hogy a MF algoritmus legalább a logaritmikus keresés folyamán fellépő legkisebb olyan korlátra fog megengedett megoldást találni, amely korlát nem esik $r_m C_m^*(P)$ alá.

28.26. tétel. *Bármely* $\mathbf{p}$ *méretvektor és* $k \in Z_+$ *egész esetén az* MF *algoritmus által szolgáltatott megoldás* C *értékére igaz, hogy*

$$\frac{C}{C_m^*(\mathbf{p})} \;\leq\; r_m + 2^{-k} \;. \tag{28.29}$$

Bizonyítás. Legyen az algoritmus kezdeti alsó és felső, valamint a befejező alsó és felső korlátja, illetve eredménye rendre A, F, BA, BF és C. Itt C a talált megengedett megoldás átfutási ideje, amire nyilvánvaló, hogy $BF \geq C$. Ha az állítás nem igaz, akkor van olyan $\mathbf{q}$ méretvektor, ahol rosszabb, azaz nagyobb értéket kapunk a (28.29) képletben megadott korlátnál. Mivel azonban az algoritmus által adott korlát legfeljebb C, adódik, hogy

$$BF \;\geq\; C \;>\; (r_m + 2^{-k}) C_m^*(\mathbf{p}). \tag{28.30}$$

A logaritmikus keresés tulajdonságai alapján $A \leq C_m^*(\mathbf{p}) \leq F \leq 2A$, ezért

$$BF \;-\; BA \;=\; 2^{-k}(F - A) \;\leq\; 2^{-k} C_m^*(\mathbf{p}) \;.$$

Innen (28.30) alapján adódik, hogy

$$BA \;>\; r_m C_m^*(P) \;.$$

Ezért az eljárás részeként szereplő FF algoritmust alkalmazni kellett a BA korlát mellett is, és ekkor FF az előző tétel állításával szemben több, mint m gépet használt. ■

Bizonyítás nélkül említjük meg a következő tételt.

28.27. tétel. *Minden* m *esetén* $r_m \leq 1.22$.

Gyakorlatok

28.4-1. Bizonyítsuk be, hogy a 28.25–28.27. tételekben szereplő r_m mennyiség minden $m \geq 2$ esetén nagyobb, mint 1.

28.4-2. Oldjuk meg azt a $P2||C_{\max}$ feladatot, melyben a megmunkálási idők: 7, 7, 6, 6, 5, 5, 4, 4, 4.

28.5. Az egyutas ütemezési probléma

Bár az egyutas ütemezési problémát nagyon sokat vizsgálták az irodalomban, a gyakorlatban viszonylag ritkán fordul elő, ezért csak a leglényegesebb eredmények ismertetésére szorítkozunk.

Csak az $F \mid\mid C_{\max}$ feladatot tárgyaljuk. Tehát minden gépből pontosan egy darab van, a gépek nem helyettesíthetik egymást, és minden munkadarab ugyanazon sorrendben halad végig a gépeken. Az általánosság megszorítása nélkül feltesszük, hogy ez a sorrend a gépek indexsorrendje. Erőforrások és megelőzési feltételek nem korlátozzák az ütemezést. A feladatnak még így is két esete van, az előzéses és az előzés nélküli. Az utóbbi azt jelenti, hogy a munkadarabok minden gépre ugyanolyan sorrendben kerülnek fel, míg az előző esetben a munkadarabok sorrendje a gépek között változhat. Az előzés nélküli esetet fogjuk részletesen tárgyalni, de első célunk az, hogy megmutassuk, hogy 2 és 3 gép esetén a két feladat azonos.

Minden további tárgyaláshoz szükségünk lesz a következő jelölésre. Legyen a egy tetszőleges munkadarab és k $(1 \leq k \leq m)$ egy tetszőleges gép. Ekkor valamely rögzített ütemezés mellett $F(a,k)$ az a munkadarab befejezési ideje a k gépen.

28.28. tétel. *Ha $m > 1$, akkor az előzéses $F \mid\mid C_{\max}$ problémának van olyan optimális megoldása, ahol az utolsó két gépen a megmunkálások sorrendje azonos.*

Bizonyítás. Tekintsünk egy tetszőleges optimális ütemezést, amelyben az utolsó előtti gépen $(1),(2),\ldots,(n)$, az utolsó gépen $[1],[2],\ldots,[n]$ a munkadarabok sorrendje. Először azt mutatjuk meg, hogy az átfutási idő megnövelése nélkül megváltoztatható úgy a sorrend az utolsó gépen, hogy a két gépen az utolsó munkadarab azonos legyen. Legyen L azon munkadarabok halmaza, melyek az utolsó gép utolsó összefüggő termelési szakaszában vannak, vagyis az utolsó állásidő után. Ebben a szakaszban minden kijelölt megmunkálás azonnal megkezdődik, mihelyst az előtte lévő a gépen befejeződött. $[n]$ mindenképpen eleme L-nek. Ha L nem tartalmaz mást, akkor ez azt jelenti, hogy minden más megmunkálás az m gépen már azelőtt befejeződött, hogy az $m-1$ gép $[n]$ megmunkálásával végzett volna, ezért szükségképpen $(n) = [n]$. Vizsgáljuk azt az esetet, amikor L több elemet is tartalmaz. Legyen $[n] = (k)$. Ha $k = n$, akkor készen vagyunk. Különben rendezzük át az utolsó gépen a sorrendet úgy, hogy az $[n] = (k)$ munkadarab megmunkálását minden (l) munkadarab megmunkálása elé visszük, ahol $k < l \leq n$. Mivel (l) csak az $F((l), m-1) > F([n], m-1)$ időpontban vagy az után kerülhet az m gépre, ezért az $[n]$ munkadarab legfeljebb az $F([n], m-1)$ időpontig kerül előbbre.

Így azonban nem keletkezhet újabb állásidő.

Ha ezen átrendezés után a két gépen nem lenne azonos az utolsó munkadarab, akkor ismételten alkalmazható egy hasonló átrendezés, mindaddig, amíg a kívánt állapotot el nem értük. Ekkor a gondolatmenet megismételhető az utolsó előtti munkadarabokra, és így tovább. ■

Az alábbiakban a következő jelöléseket fogjuk használni, tekintettel arra, hogy először az előzéses feladatot vizsgáljuk. A j gépen a munkadarabok sorrendje $[1]_j, [2]_j, \ldots, [n]_j$. A szokásos feltételezések miatt az ütemezést egyértelműen meghatározza, ha a megmunkálások sorrendje minden gépen adott. Ezért az ütemezés azonosítható a fenti m sorrend együttesével. Továbbá ha a egy tetszőleges munkadarab, akkor $j(a)$ az a munkadarab pozíciója a j gépen, azaz $[j(a)]_j = a$.

28.29. tétel. *Minden* i, j $(1 \le i \le n,\ \ 1 \le j \le m)$ *esetén*

$$F([i]_j, j) \ = \ \max\{F([i]_j, j-1),\ F([i-1]_j, j)\} \ + \ p_{[i]_j j}\ . \tag{28.31}$$

Bizonyítás. Az $[i]_j$ munkadarab megmunkálása a j gépen nem kezdődhet előbb, mint ahogy az $[i]_j$ munkadarab megmunkálása a $j-1$ gépen befejeződik, és mint ahogy az előző megmunkálás, azaz $[i-1]_j$-é, a j gépen befejeződik. Ekkor azonban azonnal meg is kezdődik és $p_{[i]_j j}$ ideig tart. ■

28.30. tétel. $F([n]_m, m) =$

$$= \max\left\{ \sum_{j=1}^{m} \sum_{\alpha=j([i_{j-1}]_{j-1})}^{i_j} p_{[\alpha]_j j} : \ \textit{minden } j \textit{ esetén } j([i_{j-1}]_{j-1}) \le i_j;\ 1 = 1([i_0]);\ i_m = n \right\}. \tag{28.32}$$

Bizonyítás. A 28.29. tételből következik, hogy az $F([n]_m, m)$ teljes átfutási idő vagy $F([n]_m, m-1) + p_{[n]_m m}$ vagy $F([n-1]_m, m) + p_{[n]_m m}$. Tehát az első esetben egy géppel, a második esetben egy munkadarabbal lépünk hátulról visszafelé. Akármelyik kifejezés legyen is a teljes átfutási idő, arra mindaddig alkalmazható a (28.31) rekurzív képlet, míg egy olyan összeget nem kapunk, ami a (28.32) képletben is szerepel.

Tekintsünk most egy tetszőleges

$$\sum_{j=1}^{m} \sum_{\alpha=j([i_{j-1}]_{j-1})}^{i_j} p_{[\alpha]_j j} \tag{28.33}$$

alakú összeget, ahol az indexhatárokra a megfelelő feltételek teljesülnek. Vegyük észre, hogy ebben egyetlen megmunkálás sem kezdődhet előbb, mint az összegzési sorrendben közvetlen előtte lévő. Ugyanis ha $j([i_{j-1}]) < \alpha \le i_j$, akkor az $[\alpha]_j$ munkadarabnak a j gépen való megmunkálásáról van szó, amit

$[\alpha - 1]_j$ ugyanezen a gépen való megmunkálásának meg kell előznie. Ha $\alpha = j([i_{j-1}])$, akkor $[\alpha]_j = [i_{j-1}]_{j-1}$, és $[\alpha]_j$-nek a j-edik gépen való megmunkálását meg kell előznie a $(j-1)$-edik gépen való megmunkálásnak. Tehát a (28.33) alakú összegek alsó korlátok a teljes átfutási időre. Mivel egy közülük éppen egyenlő vele, ezért a tétel állítása igaz. ■

28.31. definíció. *Egy feladat duáljának nevezzük azt a feladatot, ahol az i munkadarab p_{ij}^d megmunkálási idejét a j gépen a*

$$p_{ij}^d = p_{i,m+1-j}$$

képlet adja meg.

28.32. tétel. *Egy $F \mid\mid C_{\max}$ feladatnak és duáljának optimális célfüggvényértéke megegyezik.*

Bizonyítás. Tekintsük a feladat egy ütemezését. Vizsgáljuk a duál feladat azon ütemezését, ahol az $(m-j)$-edik gépen a megmunkálások sorrendjét a primál feladat adott ütemezéséből úgy kapjuk, hogy a j-edik gépen való sorrendet megfordítjuk, azaz a duál feladatban a sorrend az $m-j$ gépen $[n]_j, [n-1]_j, \ldots, [1]_j$. Ekkor azonban mind a primál, mind a duál feladat esetén ugyanazok az összegek szerepelnek a (28.32) képletben, csak az összeadandók sorrendje fordított. Így a két átfutási idő azonos, következésképp az optimális átfutási idők is azonosak. ■

28.33. tétel. *Az $F \mid\mid C_{\max}$ előzéses feladatnál $m \leq 3$ esetén van olyan optimális megoldás, hogy valamennyi gépen ugyanaz a megmunkálások sorrendje.*

Bizonyítás. $m = 1$ esetén nincs előzés. Az állítás $m = 2$ esetén a 28.28. tételből adódik. Ezen utóbbi tételből és az előzőből pedig következik, hogy a feladatnak van olyan optimális megoldása, ahol az első két gépen azonos a sorrend. Viszont $m = 3$ esetén mind az első két, mind az utolsó két gép egyike a második gép, így van olyan optimális megoldás, ahol mindhárom gépen azonos a sorrend. ■

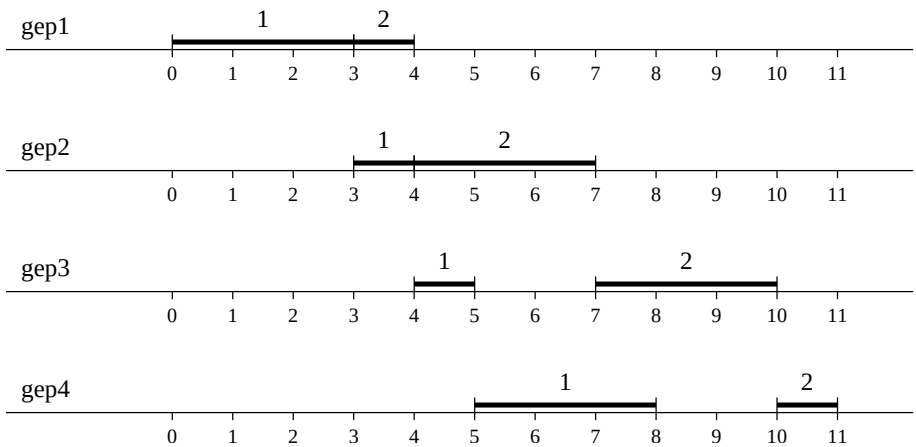
28.7. példa. Az előzés gyengíti a feltételeket, azaz több megoldást enged meg. Ezért az előzéses feladat optimum értéke nem nagyobb, mint az előzés nélkülié. Az olyan feladatnak, ahol az előzéses megoldás határozottan jobb, legalább négy gépet és két munkadarabot kell tartalmaznia. Ilyen minimális méretű ellenpélda létezik, és a megmunkálási időket az alábbi táblázat tartalmazza.

gép	1. munkadarab	2. munkadarab
1.	3	1
2.	1	3
3.	1	3
4.	3	1

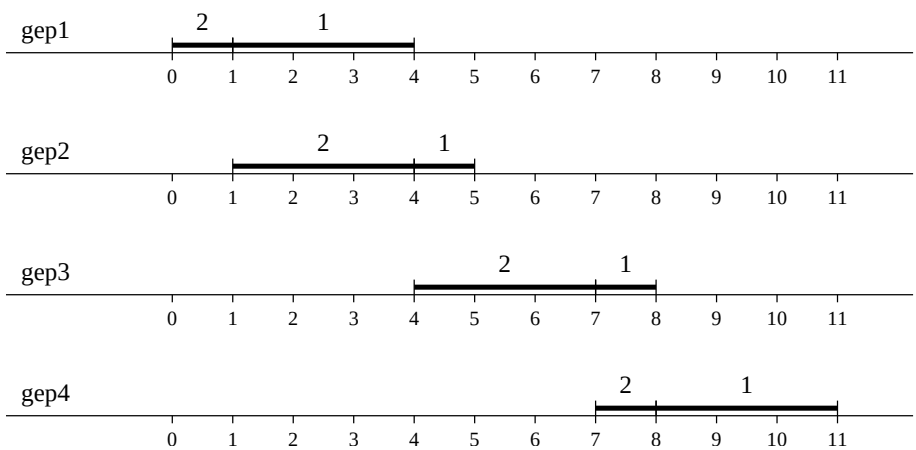
Az előzés nélküli esetben az $(1,2)$ sorrend esetén 11 az átfutási idő (lásd a 28.5. ábrát).

A $(2,1)$ sorrend esetén is 11 az átfutási idő (lásd 28.6. ábra).

Ha viszont az első két gépen $(2,1)$, a második két gépen $(1,2)$ a sorrend, akkor az átfutási idő csak 10 (lásd 28.7. ábra).



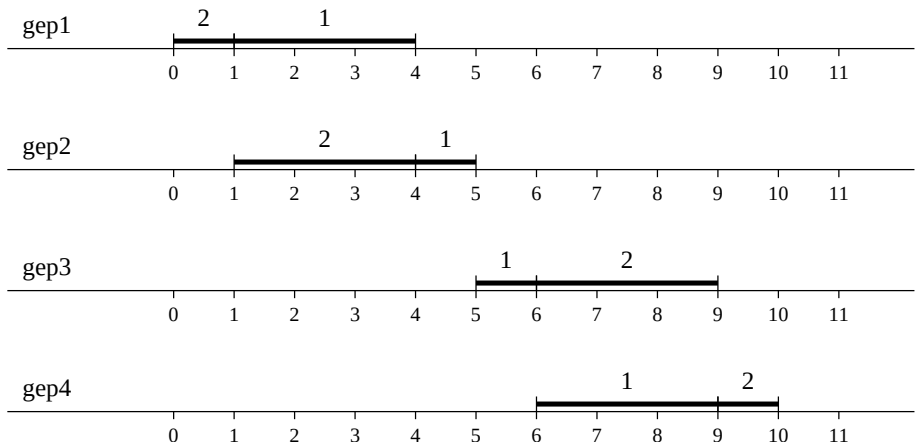
28.5. ábra. Az átfutási idő az (1,2) ütemezés esetén .



28.6. ábra. Az átfutási idő a (2,1) ütemezés esetén..

$m \geq 3$ esetén még az előzés nélküli eset is NP-teljes. Két gépre viszont Johnson már 1954-ben megoldotta a problémát.

28.34. tétel. *Az $F2 \mid\mid C_{\max}$ feladat optimális megoldását a következő algo-*



28.7. ábra. Az átfutási idő előzéses ütemezés esetén..

ritmus adja:

1. *Rendezzük a munkadarabokat a rövidebb megmunkálási idejük szerinti növekvő sorrendbe.*

2. *Ezután ebben a sorrendben döntünk a munkák sorrendjéről. Tegyük fel, hogy már van egy K kezdeti és egy B befejező sorozat. Ekkor a soron következő megmunkálás aszerint kerül K végére, illetve B elejére, hogy a rövidebb megmunkálási ideje az első, illetve a második gépen volt-e.*

Bizonyítás. Tekintsünk két munkát, indexük legyen j és k, amelyek ebben a sorrendben közvetlenül egymás után következnek. Legyen T_1, illetve T_2 az az időpont, amikor j előtt minden megmunkálás befejeződik az első, illetve a második gépen. Ez azt jelenti, hogy a j munkadarab megmunkálása az első gépen a T_1 időpontban kezdődik meg, míg a második gépen a $\max\{T_1 + p_{1j}, T_2\}$ időpontban, azaz itt a $\max\{T_1 + p_{1j} + p_{2j}, T_2 + p_{2j}\}$ időpontban fejeződik be. Ezért k megmunkálása a második gépen nem kezdődhet meg ennél, valamint $(T_1 + p_{1j} + p_{1k})$-nál korábban. Összefoglalva azt kapjuk, hogy k befejezésének időpontja a második gépen

$$\max\{T_1 + p_{1j} + p_{2j} + p_{2k}, T_2 + p_{2j} + p_{2k}, T_1 + p_{1j} + p_{1k} + p_{2k}\} \,. \quad (28.34)$$

Ha felcseréljük a két munkadarab sorrendjét, akkor a j befejezésének időpontja a második gépen

$$\max\{T_1 + p_{1k} + p_{2k} + p_{2j}, T_2 + p_{2j} + p_{2k}, T_1 + p_{1k} + p_{1j} + p_{2j}\} \,. \quad (28.35)$$

Vegyük észre, hogy az utolsó két képlet középső tagjai azonosak. Ezért, ha meg tudunk határozni egy olyan sorrendet, amelyben a két szélső tag maximuma mindig (28.34)-ban lesz kisebb, akkor ez lesz az optimális sorrend. Vegyük észre azt is, hogy a két szélső tagban a T_1 additív konstans szerepel, ami nem befolyásolja a maximum helyét, ezért ezt további vizsgálatainkból kihagyjuk. Legyen

$$A = p_{1j} + p_{2j} + p_{2k}, \quad B = p_{1j} + p_{1k} + p_{2k} \tag{28.36}$$

$$C = p_{1k} + p_{2k} + p_{2j}, \quad D = p_{1k} + p_{1j} + p_{2j} \,. \tag{28.37}$$

Arra keresünk feltételt, hogy mikor lesz $\max\{A, B\} \leq \max\{C, D\}$. Látható, hogy

$$\begin{aligned} A \leq C &\text{ akkor és csak akkor, ha } p_{1j} \leq p_{1k} \\ A \leq D &\text{ akkor és csak akkor, ha } p_{2k} \leq p_{1k} \\ B \leq C &\text{ akkor és csak akkor, ha } p_{1j} \leq p_{2j} \\ B \leq D &\text{ akkor és csak akkor, ha } p_{2k} \leq p_{2j} \,. \end{aligned}$$

A megmunkálási idők hossza szerint négy esetet különböztetünk meg.

(i) Ha $p_{1j} \leq p_{2j}$ és $p_{1k} \geq p_{2k}$, akkor a fentiekből azonnal adódik a kívánt reláció. Ez az az eset, amikor a rövidebb megmunkálási idő az elől álló munka esetében az első, a mögötte lévőnél pedig a második gépen van.

(ii) Ha $p_{1j} \leq p_{2j}$ és $p_{1k} < p_{2k}$, akkor $B \leq C$ és $A > D$. Ezért még az $A \leq C$ egyenlőtlenségre is szükség van, amiből $p_{1j} \leq p_{1k} < p_{2k}$, tehát mindkét munkának az első gépen rövidebb a megmunkálási ideje, és ezek közül is az elől álló munkáé a rövidebb.

(iii) Ha $p_{1j} > p_{2j}$ és $p_{1k} \geq p_{2k}$, akkor $B > C$ és $A \leq D$, ezért az egyenlőtlenség, amire szükségünk van, $B \leq D$. Ez akkor igaz, ha $p_{2k} \leq p_{2j} < p_{1j}$, vagyis mindkét munkának a második gépen rövidebb a megmunkálási ideje, és ezek közül is a hátul álló munkáé a rövidebb.

(iv) Ha $p_{1j} > p_{2j}$ és $p_{1k} < p_{2k}$, akkor $B > C$ és $A > D$, tehát a kívánt reláció nem teljesülhet, vagyis nem lehetséges, hogy az elől álló munkának a második, a hátul állónak pedig az első gépen legyen határozottan rövidebb a megmunkálási ideje. ■

A jelen szakasz további részében az előzés nélküli feladattal foglalkozunk.

Az egyutas ütemezési probléma pontos megoldási módszerei többnyire vagy korlátozás és szétválasztás, vagy implicit leszámlálás típusú módszerek. A korlátozás és szétválasztás lényege, hogy a megengedett megoldások halmazát egyre kisebb részhalmazokra bontja fel, miközben (a) egyre pontosabban becsüli a részhalmazokba eső legjobb megengedett megoldások célfüggvényértékét, és (b) a becslő eljárás során időnként megengedett megoldásokat

állít elő. Ily módon feleslegessé válnak azok a részhalmazok, ahol a becslés rosszabb, mint az addig talált legjobb megoldás értéke. Az implicit leszámlálásnál a feltételek kielégíthetőségéből tudunk levonni következtetéseket. Az ütemezési feladatok esetében ez azt jelenti, hogy bizonyos típusú sorrendek esetén a célfüggvény értéke meghaladna egy meghatározott értéket.

Tanulságosak azok a teljes átfutási időre vonatkozó korlátok, amelyeket a korlátozás és szétválasztás típusú módszereknél kell használni.

Az alsó korlátok mind a 28.30. tételt használják. Két fajtájuk van, a gépre, illetve a munkára alapozott alsó korlát. Mindkettő abból indul ki, hogy az adott objektummal kapcsolatos összes megmunkálást (vagyis az adott gépen vagy az adott munkadarabon végrehajtandó összes megmunkálást) el kell végezni és ez sikerül is állásidő nélkül, és ez előtt és után is a lehető legkevesebb időt használjuk fel. Mindezt az alábbi két tételben fogalmaztuk meg.

28.35. tétel. *Az $F \,||\, C_{\max}$ feladat optimális célfüggvényértéke legalább*

$$\max\{\sum_{j\neq l} \min\{p_{1j}, p_{mj}\} \;+\; \sum_{i=1}^{m} p_{il} : \;\; l = 1, \ldots, n\} \,. \tag{28.38}$$

Bizonyítás. Tekintsünk egy tetszőleges l munkát. Ennek végig kell mennie valamennyi gépen, vagyis a teljes átfutási időnek tartalmaznia kell ezen munka megmunkálási időinek összegét (ez a második összeg a fenti képletben). Ezen felül minden más j munka vagy előtte van a sorrendben, és akkor ezen j munkának az első gépen való megmunkálási ideje hátrébb tolja l megmunkálásának kezdetét, vagy mögötte van, ekkor j megmunkálása az utolsó gépen csak l befejezése után történik, vagyis a teljes átfutási idő l befejezéséhez képest ennyivel kitolódik. Akkor járunk a legjobban, ha pontosan azok a munkák vannak l előtt, amelyeknek a megmunkálása az első gépen rövidebb, mint az utolsón. Vagyis – a 28.30. tételhez hasonlóan – a két összeg olyan megmunkálásokat tartalmaz, amelyeknek időben egymás után kell következniük. ■

28.36. tétel. *Az $F \,||\, C_{\max}$ feladat optimális célfüggvényértéke legalább*

$$\max\{\min_{j} \sum_{l=1}^{i-1} p_{lj} \;+\; \sum_{j=1}^{n} p_{ij} \;+\; \min_{j} \sum_{l=i+1}^{m} p_{lj} : \;\; i = 1, \ldots, m\} \,. \tag{28.39}$$

Bizonyítás. A hármas összeg tagjainak jelentése a következő. Az első tag azt a legrövidebb időt adja meg, amennyinek mindenképpen el kell telnie ahhoz, hogy az első megmunkálás megkezdődhessék az i-edik gépen, és ez nem más,

mint az a legrövidebb idő, ami alatt egy munkadarab az i-edik gépet eléri. A második összeg a megmunkálások ideje az i-edik gépen. Ezek természetesen csak azután kezdődhetnek meg, hogy az első munkadarab elérte a gépet. Végül a harmadik összeg az a legrövidebb idő, ami alatt egy munkadarab elkészítését be lehet fejezni azután, hogy lekerült az i-edik gépről. Vagyis hasonlóan a 28.30. tételhez, a három tag olyan megmunkálásokra tartalmaz alsó becslést, amelyeknek időben egymás után kell következniük. ■

Általános szokás, hogyha egy nehéz feladat egy speciális esetét meg lehet oldani valamely egyszerű eljárással, akkor úgy készítenek heurisztikus eljárást az általános esetre, hogy „visszavezetik" azt a megoldott speciálisra. Mivel az általános egyutas feladat is nehéz, de ugyanakkor Johnson algoritmusa az $m = 2$ esetet megoldja, tehát pontosan a fenti helyzettel állunk szemben, pusztán az a kérdés, hogy hogyan lehet értelmezni a két „gépet". Az idézőjel itt arra vall, hogy ez a két gép csak absztrakt gép lesz, nem az eredeti feladat valamely gépe.

Mielőtt a fenti kérdésre válaszolnánk, érdemes szemügyre venni a Johnson algoritmusa mögött meghúzódó gondolatot. Emlékezzünk rá, hogy azok a munkák kerültek előre, amelyeknek a rövidebb megmunkálási ideje az első gépen, a hosszabb pedig a másodikon volt. A sorrend végére pedig azok kerültek, amelyeknél a helyzet éppen fordított volt. Az ilyen sorrend azt célozza, hogy minél előbb legyen munkája a második gépnek, vagyis ott kevés legyen az állásidő, majd miután az első gép befejezte a munkáját (természetesen állásidő nélkül), akkor minél rövidebb ideig kelljen a második gépnek dolgoznia. Kiterjesztve ezt több gép esetére, olyan sorrendet szeretnénk, ahol az első gépeken kevés időt használó munkák kerülnek előre, a hátsó gépeken keveset használók pedig a sorrend végére.

Úgy lehet két gépet „csinálni" a sokból, hogy bizonyos gépeket „összevonunk", azaz úgy tekintjük, mintha az ezeken végzendő megmunkálások egyetlen nagy megmunkálást adnának ki, ahol a megmunkálási idő az eredeti megmunkálási idők összege. A fentiek szellemében ezt két különböző módon is meg lehet csinálni. Az egyik esetben elölről is és hátulról is ugyanannyi gépet tekintünk, míg a gépek középső részét figyelmen kívül hagyjuk. Ha r a tekintetbe veendő gépek száma mindkét oldalról, akkor a két mesterséges gépen a megmunkálási idők

$$p^r_{1j} = \sum_{i=1}^{r} p_{ij}, \quad p^r_{2j} = \sum_{i=m-r+1}^{m} p_{ij}, \quad j = 1, \ldots, n \tag{28.40}$$

lesznek. Itt $r = 1, \ldots, \lceil m/2 \rceil$. Az így definiált kétgépes feladatokat minden r esetén külön-külön megoldjuk Johnson módszerével, majd az így kapott sor-

rendre kiszámítjuk az eredeti feladat átfutási idejét. Végül az így kapott sorrendek közül a legjobbat választjuk. Még ha m páratlan is, r felső határának ilyetén megválasztása biztosítja, hogy minden, az eredeti feladatban szereplő gépet legalább egyszer figyelembe veszünk.

A másik lehetőség, hogy a gépeket két részre osztjuk, és eszerint határozzuk meg a mesterséges gépeken a megmunkálási időket. Ha az első csoportban r gép van, akkor a megmunkálási időket

$$p_{1j}^r = \sum_{i=1}^{r} p_{ij},\ p_{2j}^r = \sum_{i=r+1}^{m} p_{ij},\quad j = 1, \ldots, n \tag{28.41}$$

adja meg. Itt $r = 1, \ldots, m-1$. Minden másban úgy járunk el, mint fent.

Befejezésül azt mutatjuk meg, hogy az $F \mid \textit{no-wait} \mid C_{\max}$ probléma egy utazóügynök feladatra vezethető vissza. Ebben a feladatban tehát a munkák nem várhatnak a gépek előtt, hanem éppen fordítva, a gépeknek kell várniuk a munkákra.

Ha a j-edik munkának sehol nem kell várnia a gépek előtt, és gyártása a 0 időpontban kezdődik, akkor a megmunkálása az r gépen a

$$\sum_{i=1}^{r} p_{ij}$$

időpontban fejeződik be. Tegyük fel, hogy közvetlenül utána a k munka következik, melynek gyártása a $t \geq 0$ időpontban kezdődik. Tudjuk, hogy k csak akkor kerülhet az r gépre, ha a megmunkálása az $r-1$ gépen befejeződött, ami a

$$t + \sum_{i=1}^{r-1} p_{ik}$$

időpontban következik be. Ekkor azonban az r gépnek szabadnak kell lennie, azaz teljesülnie kell a

$$t + \sum_{i=1}^{r-1} p_{ik} \geq \sum_{i=1}^{r} p_{ij}$$

egyenlőtlenségnek. Vagyis t-nek el kell érnie legalább a

$$\sum_{i=1}^{r} p_{ij} \ - \ \sum_{i=1}^{r-1} p_{ik}$$

értéket. Mivel ennek minden gépre teljesülnie kell, ezért a k munka a j munka

után

$$\max\left\{\sum_{i=1}^{r} p_{ij} - \sum_{i=1}^{r-1} p_{ik} \quad r = 1, \ldots, m\right\} \tag{28.42}$$

idővel indítható. (Itt az üres összeg értéke szokás szerint 0, továbbá a megmunkálási időkről feltesszük, hogy nemnegatívak, így a fenti érték is nemnegatív.) Mivel a feltételezés szerint a megmunkálási időket nem tudjuk változtatni, csak úgy tudjuk a teljes átfutási időt a lehető legrövidebbé tenni, hogy minimalizáljuk a munkák indításai között eltelő várakozási idők összegét. Ez azonban nem más, mint az az utazóügynök feladat, ahol a városok a munkák, a közöttük lévő távolságot pedig a (28.42) képlet adja meg.

28.6. A többutas ütemezési probléma

A többutas problémák családja tartalmazza a legbonyolultabb ütemezési feladatokat. Itt semmiféle megkötés nincs a technológiai útvonalakra, még az is megengedett, hogy egy munka ne kerüljön minden gépre, és hogy egyes gépeket többször is felkeressen. Egy megszorítás van, amit általánosan fel szoktak tenni, nevezetesen, hogy nincsenek párhuzamos berendezések, azaz minden gépből csak egy van. Ez annyit jelent, hogy minden megmunkálást egy meghatározott gépen kell elvégezni.

Ahhoz, hogy egy feladatot pontosan meg lehessen fogalmazni és oldani, valamilyen matematikai modell kell. A két leggyakrabban használt modell az ú.n. diszjunktív gráf modell és a diszkrét programozási modell. Először ezeket tárgyaljuk.

Az alábbiakban az angol nyelvű irodalomhoz hasonlóan a megmunkálás szó helyett a művelet kifejezést alkalmazzuk. Ennek oka az, hogy megmunkáláson inkább értjük az egy munkadarabon végzett átalakítást, míg művelet bármi lehet, még két különálló darab összehegesztése vagy egyetlen elem feldarabolása is.

Különösen áll ez az elsőnek tárgyalandó diszjunktív gráf modellre. Ennek mélyebb megértéséhez hasznos a kritikus út módszerének ismerete.

28.6.1. A kritikus út módszere

Ez a módszer (CPM) nem képezi a jelen fejezet tárgyát, ezért itt csak a legfontosabb tudnivalókat foglaljuk össze. Tegyük fel, hogy valamilyen tevékenységek sorozatát – esetünkben valamely termékek gyártásához tartozó műveleteket – akarjuk elvégezni. Ezen tevékenységek között lehetnek bizonyos megelőzési kapcsolatok, amelyek azt fejezik ki, hogy valamely

tevékenységnek be kell fejeződnie ahhoz, hogy egy másik elkezdődhessék. A feladat több különböző módon is leírható egy G irányított gráffal. Nekünk arra a változatra lesz szükségünk, ahol a gráf csúcsai a tevékenységek. Ekkor egy α csúcsból (tevékenységből) irányított él vezet minden olyan β csúcsba (tevékenységbe), amely őt közvetlenül követheti. Az él hossza az α tevékenység elvégzéséhez szükséges idő, azaz az α csúcsból kiinduló valamennyi él hossza azonos. Feltételezzük továbbá egy 0 hosszúságú $\circ$ kezdeti és $*$ befejező tevékenység létezését, amelyek a következő tulajdonságokkal rendelkeznek.
(i) A $\circ$ csúcsba nem vezet él.
(ii) A $*$ csúcsból nem indul ki él.
(iii) Legyen α tetszőleges, $\circ$-től és $*$-tól különböző tevékenység. Ekkor $\circ$-ből vezet irányított út α-ba, és α-ból vezet irányított út $*$-ba.

A G gráf egy fontos további tulajdonsága még, hogy
(iv) A G gráf irányított kört nem tartalmaz.
Ha ugyanis irányított kört tartalmazna, akkor a körbe tartozó valamennyi tevékenység megkezdésének az volna a feltétele, hogy a kör többi tevékenysége befejeződjön, azaz egyetlen, a körhöz tartozó tevékenységet sem lehetne elkezdeni.

Feltételezve, hogy az egész tevékenységsorozat a 0 időpontban kezdődik, igaz a következő

28.37. tétel. *(a) Az egész tevékenységsorozat lehetséges legrövidebb átfutási ideje egyenlő a $\circ$-ből $*$-ba vezető leghosszabb út hosszával.*
(b) Semmilyen α tevékenység sem kezdődhet előbb, mint a $\circ$-ből az α-ba vezető leghosszabb út hossza.
(c) Ha a tevékenységsorozat lehetséges legrövidebb átfutási ideje T, akkor egyetlen α tevékenység sem kezdődhet később, mint

$$T - (\alpha\text{-ból } *\text{-ba vezető leghosszabb út hossza})$$

anélkül, hogy a teljes átfutási időt ne tolná ki T-n túlra.

A tétel (a) részében említett leghosszabb úthoz tartozó tevékenységek csak egyetlen jól meghatározott időpontban kezdődhetnek a teljes átfutási idő meghosszabbítása nélkül, ezért ezeket „kritikusnak” hívják, innen ered az egész módszer elnevezése.

Jelölje $p_\alpha, q_\alpha, r_\alpha$ rendre az α tevékenység elvégzéséhez szükséges időt, α lehetséges legkésőbbi befejezési idejét a teljes átfutási idő meghosszabbítása nélkül és végül α lehetséges legkorábbi kezdési idejét. Az utóbbi két mennyiség egy dinamikus programozási algoritmusból is meghatározható. (Ebben fontos szerepet játszik, hogy a gráf nem tartalmaz irányított kört.)

A megfelelő Bellman-egyenleteket az alábbi tétel adja meg.

28.38. tétel. *Jelölje N a G gráf csúcsainak, A pedig éleinek halmazát. Ekkor*

$$r_\circ = 0\ , \tag{28.43}$$

$$\textit{minden } \alpha \in N \setminus \{\circ\} \textit{ esetén } \quad r_\alpha \ = \ \max\{r_\beta + p_\beta : (\beta,\alpha) \in A\}\ , \tag{28.44}$$

$$q_* \ = \ T\ , \tag{28.45}$$

$$\textit{minden } \alpha \in N \setminus \{*\} \textit{ esetén } \ q_\alpha \ = \ \min\{q_\beta - p_\beta : (\alpha,\beta) \in A\}\ . \tag{28.46}$$

28.6.2. A diszjunktív gráf modell

Tekintsünk két tetszőleges α, β műveletet. Ezek az alábbi három viszony valamelyikében állnak egymással:
(i) az egyiknek meg kell előznie a másikat (pl. azért, mert ugyanahhoz a munkadarabhoz tartozó műveletekről van szó, és a technológia előírja a sorrendjüket),
(ii) egyszerre nem végezhetők (pl. azért, mert mindkettő ugyanazt a gépet igényli),
(iii) nincsenek közvetlen hatással egymásra (pl. különböző munkadarabokon különböző gépekkel végzendő műveletek).

A fenti helyzetet a $G = (N, C \cup D)$ ú.n. diszjunktív gráf írja le, ahol N a műveletek halmaza, C az (i) kategóriába eső relációkat leíró, konjunktívnak nevezett éleket tartalmazza, D pedig a (ii) kategóriába tartozó relációkat reprezentáló, diszjunktív élpárokat. Ez azt jelenti, hogyha az α és β művelet nem végezhető egyszerre, akkor mind az (α,β), mind a (β,α) él eleme a D halmaznak. Továbbá minden $\alpha, \beta \in N$ esetén feltesszük, ha $(\alpha,\beta) \in C \cup D$, akkor az (α,β) él hossza p_α, azaz az α művelet elvégzéséhez szükséges idő, ami független az él végpontjától, azaz β-tól. Az élek hosszát, azaz a megmunkálási időket, mindvégig nemnegatívnak tételezzük fel.

28.39. definíció. *Az $A \subset D$ halmazt **programnak** nevezzük, ha $(\alpha,\beta) \in A$ akkor és csak akkor, ha $(\beta,\alpha) \notin A$.*

Ha tehát A egy program, akkor a $G' = (N, C \cup A)$ egy szokásos irányított gráf. Ennek élei határozzák meg, hogy a műveletek milyen sorrendben végezhetők. Ha minden műveletet a lehetséges legkorábbi időpontra ütemezünk, azaz nem iktatunk be állásidőt olyan esetben, amikor mind a gép

rendelkezésre áll, mind a sorrend szerint következő művelet elvégezhető volna, akkor G' egyértelműen meghatároz egy termelési programot.

A diszjunktív gráf modell nyelvén a $J||C_{\max}$ feladat úgy fogalmazható meg, hogy meghatározandó a G gráfhoz egy A program úgy, hogy az így keletkező G' gráfban a kritikus út hossza minimális legyen.

Kritikus útról természetesen csak akkor lehet beszélni, ha G' nem tartalmaz irányított kört. Ezért az alábbiakban mindvégig feltesszük, hogy *csak olyan A programokat vizsgálunk, hogy G' nem tartalmaz irányított kört.*

A diszjunktív gráf modell megoldására szolgáló módszerek kevés kivételtől eltekintve mind a korlátozás és szétválasztás elvén alapulnak. Ehhez természetesen a kritikus út hosszára vonatkozó alsó korlátra van szükség. Az alábbiakban tárgyaljuk a leggyakoribbakat.

Amikor az optimális A programot keressük, az élpárok egy-egy tagját külön-külön választjuk ki. Így az algoritmusok során mindig olyan helyzet áll fenn, hogy a C halmazhoz még hozzáválasztunk néhány további élt. Ezt a halmazt fogjuk B-vel jelölni. Tehát valamely alkalmas A program mellett mindig igaz a $C \subset B \subset C \cup A$ reláció. Az (N, B) gráfot H jelöli. Hangsúlyozzuk, hogy H tartalmazza az összes konjunktív élt, bizonyos diszjunktív élpárokból egyet, míg a többi diszjunktív élpárból egyik élt sem.

28.40. definíció. *Legyen M a műveletek egy részhalmaza. Azt mondjuk, hogy M* ***diszjunktív,*** *ha bármely $\alpha, \beta \in M$ $(\alpha \neq \beta)$ esetén α-t és β-t a H gráfban vagy egy konjunktív élekből álló irányított út köti össze, vagy egy diszjunktív élpár.*

Szükségünk lesz még az alábbi két mesterséges műveletre, melyek bevezetése megkönnyíti a számításokat: $\circ$ egy 0 hosszúságú kezdő művelet, $*$ pedig 0 hosszúságú befejező művelet.

Ezen két műveletről bármely (N, B) gráfra vonatkozóan, tehát még a (N, C) esetén is a következőket tesszük fel:
(1) A $\circ$ csúcsba nem vezet él.
(2) A $*$ csúcsból nem indul ki él.
(3) Legyen α tetszőleges, $\circ$-től és $*$-tól különböző művelet. Ekkor $\circ$-ból vezet irányított út α-ba, és α-ból vezet irányított út $*$-ba.

Továbbá az alábbi jelöléseket alkalmazzuk, amelyek mind a $H = (N, B)$ gráfra vonatkoznak:

- r_α az α művelet megkezdésének lehetséges legkorábbi időpontja,

- q_α az az idő, aminek minimálisan el kell telnie az α művelet befejezése

után minden művelet befejezéséig.

Könnyen látható, hogy r_α nem más, mint a $\circ$-ből α-ba vezető leghosszabb út hossza a H gráfban, míg q_α ugyanitt az α csúcsból a $*$ csúcsba vezető leghosszabb út hossza p_α nélkül. Ezért az r_α, q_α mennyiségekre a (28.43)–(28.46) képletekhez hasonlóan az alábbi rekurzív képletek adhatók meg:

$$r_\circ = 0\ , \tag{28.47}$$

$$\text{minden } \alpha \in N \setminus \{\circ\} \text{ esetén } \quad r_\alpha = \max\{r_\beta + p_\beta : (\beta,\alpha) \in B\}\ , \tag{28.48}$$

$$q_* \ = \ 0\ , \tag{28.49}$$

$$\text{minden } \alpha \in N \setminus \{*\} \text{ esetén } \quad q_\alpha = \max\{q_\beta - p_\beta : (\alpha,\beta) \in B\}\ . \tag{28.50}$$

Ha M a műveletek egy diszjunktív részhalmaza, és ebben az α és β művelet olyan, hogy konjunktív élekből álló út vezet α-ból β-ba, akkor az

$$r_\alpha \ \le \ r_\beta, \ \ q_\alpha \ \ge \ q_\beta$$

egyenlőtlenségek automatikusan teljesülnek. Ha a B halmazhoz hozzáveszünk egy addig diszjunktív (α,β) élt, akkor az r és q értékeket azonnal úgy kell módosítani, hogy a Bellman-egyenletek továbbra is igazak maradjanak.

28.41. tétel. *Bármely A program esetén, amelyre $B \subset C \cup A$ teljesül, a teljes átfutási idő legalább*

$$\max\{r_\alpha + p_\alpha + q_\alpha : \quad \alpha \in N\}\ . \tag{28.51}$$

Bizonyítás. Az $r_\alpha + p_\alpha + q_\alpha$ összeg a $\circ$-ből α-n keresztül $*$-ba vezető leghosszabb út hossza. Tehát a képlet a kritikus út hosszát adja meg a H gráfban. Ez a hossz nem csökkenhet, ha további nemnegatív hosszú éleket vezetünk be a gráfba. ■

A 28.36. tételnek az volt a logikája, hogy megvizsgáltuk, hogy milyen gyorsan tud egyáltalán eljutni valami egy i gépre, ott optimális esetben állásidő nélkül el lehet végezni az összes megmunkálást, majd az utolsó munkadarabnak még be is kell fejeződnie a további gépeken, ezt is a lehető legrövidebbnek feltételezve. Ehhez hasonló módon számos alsó korlát nyerhető.

28.42. tétel. *A többutas ütemezési feladat átfutási ideje a műveletek bármely diszjunktív M halmaza esetén legalább*

$$\min_{\alpha\in M} r_\alpha \ + \ \sum_{\alpha\in M} p_\alpha \ + \ \min_{\alpha\in M} q_\alpha\ . \tag{28.52}$$

Bizonyítás. Gyengítsük a feladat feltételeit úgy, hogy az M halmaz kivételével az összes többi diszjunktív élpártól eltekintünk. Ekkor is minimálisan el kell annyi időnek telnie, amíg az M-beli műveletek közül az első megkezdődhet, az összes M-beli művelet befejeződik, végül az M-beli műveleteket követő megmunkálások is véget érnek. Ezen három tényezőre ad rendre alsó korlátot a (28.52) kifejezés három tagja. ■

28.43. tétel. *Legyen a műveletek bármely diszjunktív M halmaza esetén az M-beli műveleteknek egy, a q_j értékek szerinti monoton csökkenő sorrendje* $[1], [2], \ldots, [\mid M \mid]$. *A többutas ütemezési feladat átfutási ideje legalább*

$$\min_{\alpha \in M} r_\alpha \; + \; \max\{\sum_{i=1}^{j} p_{[i]} + q_{[j]} : \;\; j = 1, \ldots, \mid M \mid\} \; . \tag{28.53}$$

Bizonyítás. Gyengítsük ugyanúgy a feltételeket, mint az előző tételnél. Továbbá tegyük fel, hogy minden művelet rendelkezésre áll a $\min_{\alpha \in M} r_\alpha$ időpontra. Legyen a teljes tevékenység végső határideje T. Innen a gyengített feltételek alapján adódik, hogy az M-beli α művelet határideje $T - q_\alpha$. Az eredeti feladat teljes átfutási idejére T olyan értéke ad alsó becslést, amely mellett a gyengített feladatban a maximális késés nemnegatív. Nyilvánvalóan ezek közül az a legjobb T érték, amely mellett a maximális késés éppen 0. A 28.4. tételből tudjuk, hogy a monoton növő határidők szerinti ütemezés minimalizálja a maximális késést, és esetünkben a q_j értékek szerinti monoton csökkenő sorrend ilyen. Az adott sorrend szerinti $[j]$ művelet befejezési időpontja

$$\min_{\alpha \in M} r_\alpha \; + \; \sum_{i=1}^{j} p_{[i]} \; .$$

Mivel a maximális késés 0, ezért innen a

$$\min_{\alpha \in M} r_\alpha \; + \; \max\{\sum_{i=1}^{j} p_{[i]} \; - \; (T \; - \; q_{[j]}) : \;\; j = 1, \ldots, \mid M \mid\} \; \leq \; 0$$

egyenlőtlenség adódik. Ebből T maximális értékére a (28.53)-beli kifejezés kapható. ■

28.44. tétel. *Legyen a műveletek egy diszjunktív M halmaza esetén* $[1], [2], \ldots, [\mid M \mid]$ *az M-beli műveleteknek egy, a rendelkezésre állási idők szerinti monoton csökkenő sorrendje. A többutas ütemezési feladat átfutási ideje legalább*

$$\min_{\alpha \in M} q_\alpha \; + \; \max\{\sum_{i=1}^{j} p_{[i]} + r_{[j]} : \;\; j = 1, \ldots, \mid M \mid\} \; . \tag{28.54}$$

Bizonyítás. Alkalmazzuk az előző tételt arra az ütemezési feladatra, amelyben a megmunkálási idők azonosak a vizsgált problémában szereplőkkel, míg a rendelkezésre állási időket a q_j értékek adják meg és r_j időnek kell eltelnie a j művelet befejezése után a műveletek teljes befejezéséig. ■

A korlátozás és szétválasztási módszerek gyakran keverednek az implicit leszámlálással, s nem csupán az ütemezési problémák megoldásánál. Az utóbbinak az a lényege, hogy a szóba kerülő megoldások egy részét ki akarjuk zárni, anélkül, hogy ténylegesen megvizsgálnánk őket. Itt egy ilyen lehetőséget mutatunk be a 28.42. tételre támaszkodva.

28.45. tétel. *Legyen M műveletek egy halmaza és α egy további úgy, hogy az $M \cup \{\alpha\}$ halmaz diszjunktív. Legyen továbbá $C > 0$ egy tetszőleges szám. Ha*

$$r_\alpha \; + \sum_{\beta \in M} p_\beta \; + \; p_\alpha \; + \; \min_{\beta \in M} q_\beta \; \geq \; C \tag{28.55}$$

és

$$\min_{\beta \in M} r_\beta \; + \sum_{\beta \in M} p_\beta \; + \; p_\alpha \; + \; \min_{\beta \in M} q_\beta \; \geq \; C \; , \tag{28.56}$$

akkor minden olyan ütemezésben, amelynek az átfutási ideje rövidebb, mint C, α az összes M-beli művelet mögött áll.

Bizonyítás. A (28.55) egyenlőtlenség baloldala alsó korlát a teljes átfutási időre abban az esetben, ha α megelőzi az összes M-beli műveletet, azaz ebben az esetben nem lehet az átfutási idő C-nél rövidebb. Hasonlóképpen a (28.56) egyenlőtlenség baloldala alsó korlát a teljes átfutási időre abban az esetben, ha α az M-beliek között van, azaz sem nem első, sem nem utolsó. ■

A tételben $\mid M \mid +1$ műveletről esik szó, ezek összesen $(\mid M \mid +1)!$ sorrendet alkothatnak. Ha az állítás feltételei teljesülnek, akkor már csak $\mid M \mid !$ sorrend kerülhet szóba, azaz a megvizsgálandó esetek számát $1/(|M|+1)$-ed részére redukáltuk. Ez az oka annak, hogy az ilyen teszteket akkor is érdemes igen gyakran ellenőrizni, ha számos esetben nem adnak pozitív eredményt.

A korlátozás és szétválasztás eljárásában a részhalmazok felbontásának számos módszere ismert. Ezek közül a legegyszerűbb az, amikor egy halmazt egy még nem vizsgált diszjunktív élpárnak megfelelően bontanak két részhalmazra. Egy finomabb eljárás az ú.n. *kritikus blokkon* alapul. Tegyük fel, hogy már ismert egy ütemezés C átfutási idővel. Tekintsünk a pillanatnyi, esetleg még teljes ütemezést nem adó megoldásban egy olyan α műveletet, ha ilyen

egyáltalán van, amelyre

$$r_\alpha + p_\alpha + q_\alpha \geq C \tag{28.57}$$

teljesül. Ha van ilyen művelet, akkor egyetlen olyan teljes ütemezés sem adhat jobbat a már ismert megoldásnál, mely teljes ütemezés a jelenlegi megoldásból keletkezik kiegészítéssel. Ez a helyzet például éppen akkor, amikor egy minden korábbinál jobb ütemezést találtunk, hiszen ettől a pillanattól kezdve ennél jobbat keresünk, és a (28.57) feltétel legalább egy műveletre teljesül. Legyen α egy olyan művelet, amire (28.57) igaz, de egyetlen $\circ$-ből α-ba vezető útba eső műveletre sem áll fenn a fenti egyenlőtlenség. A $\circ$-ből α-ba vezető útba eső műveletek alkotják az ú.n. kritikus blokkot. Nyilvánvalóan csak akkor tudunk a már ismertnél jobb ütemezést találni, ha sikerül α-t a kritikus blokkon belül előbbre ütemeznünk. Tehát minden, a későbbiekben felbontandó halmaznak elég csak azokat a részhalmazait tekinteni, amelyekbe eső elemek ezt a követelményt teljesítik. Itt persze adott esetben számos lehetőség merülhet fel, melyek kezelésének részleteibe nem megyünk bele. Ha a (28.57) feltétel semmilyen α mellett sem teljesül, akkor a felbontásra a fent említett egyszerű szabály alkalmazható.

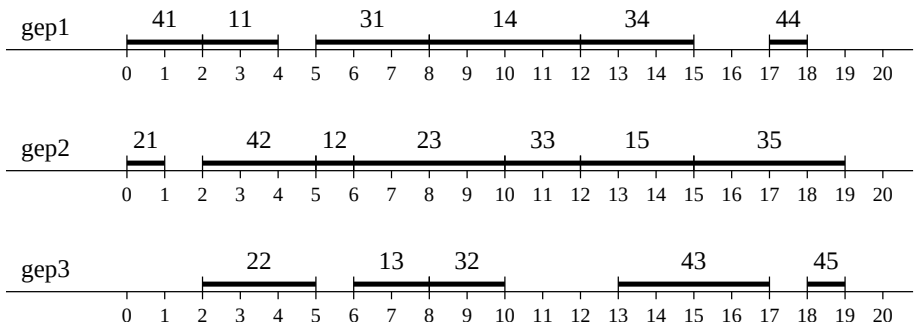
28.8. példa. Tekintsük azt a – négy munkát és három gépet tartalmazó – feladatot, melyet az alábbi adatok határoznak meg. A táblázat belsejében az első szám a gép indexe, a második a művelet elvégzéséhez szükséges idő.

munka	1. művelet	2. művelet	3. művelet	4. művelet	5. művelet
1.	1/2	2/1	3/2	1/4	2/3
2.	2/1	3/3	2/4		
3.	1/3	3/2	2/2	1/3	2/4
4.	1/2	2/3	3/4	1/1	3/1

A műveleteket kétjegyű indexekkel azonosítjuk, az első jegy a munka indexe, a második jegy pedig azt mutatja meg, hogy adott munkához tartozó hányadik műveletről van szó. Ha még egyetlen műveletet sem ütemeztünk, vagyis az A halmaz üres, akkor a megfelelő r és q értékek a következők:

$$\begin{array}{lllll}
r_{11}=0 & r_{12}=2 & r_{13}=3 & r_{14}=5 & r_{15}=9 \\
r_{21}=0 & r_{22}=1 & r_{23}=4 & & \\
r_{31}=0 & r_{32}=3 & r_{33}=5 & r_{34}=7 & r_{35}=10 \\
r_{41}=0 & r_{42}=2 & r_{43}=5 & r_{44}=9 & r_{45}=10 \\
q_{11}=10 & q_{12}=9 & q_{13}=7 & q_{14}=3 & q_{15}=0 \\
q_{21}=7 & q_{22}=4 & q_{23}=0 & & \\
q_{31}=11 & q_{32}=9 & q_{33}=7 & q_{34}=4 & q_{35}=0 \\
q_{41}=9 & q_{42}=6 & q_{43}=2 & q_{44}=1 & q_{45}=0
\end{array}$$

Tekintsük a második gépet. Az ezen végzendő műveletek 12, 15, 21, 23, 33, 35,



28.8. ábra. Egy optimális ütemezés.

42. Így a fenti tételek alapján a következő becsléseket kapjuk. Mivel most minden művelet esetén az $r_\alpha + p_\alpha + q_\alpha$ érték nem más, mint az adott munkához tartozó megmunkálási idők összege, ezért a 28.41. tétel alapján mind a 33, mind a 35 műveletre a 14 értéket kapjuk. Lévén $r_{21} = q_{35} = 0$, ezért a 28.42. tétel alapján a második gépen végzendő műveletek összidejét kapjuk, ami 18. Ugyanezt az értéket adja a 28.43. tétel is. A rendelkezésre állási idők szerinti csökkenő sorrend nem egyértelmű, a két lehetséges sorrend 35, 15, 33, 23, 12, 42, 21 és 35, 15, 33, 23, 42, 12, 21. Innen a 28.44. tétel alapján a 19 alsó korlát adódik, nem tekintve ugyanis a 21 műveletet, kapjuk, hogy

$$q_{35} + p_{35} + p_{15} + p_{33} + p_{23} + p_{12} + p_{42} + r_{42} = 19 \ .$$

Ez azt sugallja, hogy a második gépen a műveleteket 21, 42, 12, 23, 33, 15, 35 sorrendben kell végezni, és valóban, ez ugyanis az alábbi ütemezéshez vezet, ahol t_j a művelet kezdési, c_j pedig a befejezési időpontja.

	11	12	13	14	15	21	22	23	31	32	33	34	35	41	42	43	44	45
t_j	2	5	6	8	12	0	2	6	5	8	10	12	15	0	2	13	17	18
c_j	4	6	8	12	15	1	5	10	8	10	12	15	19	2	5	17	18	19

Az ütemezés Gantt-diagramja a 28.8. ábrán látható.

28.6.3. Egészértékű programozási modellek

Az első elterjedt modell a $J \,||\, f$ feladatra vonatkozik azon egyszerűsítő feltételezés mellett, hogy minden munka legfeljebb egyszer kerül egy gépre. A képletekben az alábbi indexeket és állandókat használjuk:

- i, j: a munkák indexei,
- n: munkák száma,
- m_i: az i munkán végzendő műveletek száma,

- k, l: a műveletek indexei,
- g: a gép indexe,
- d_i: az i munka határideje,
- p_{ig}: az i munka g gépen végzendő műveletének megmunkálási ideje,
- g_{ik}: az a gép, amelyen az i munka k műveletét végezni kell,
- M: egy nagy szám, például $\sum_{i=1}^{n} \sum_{k=1}^{m_i} p_{ig_{ik}}$.

A feladat változói pedig

- $x_{ijg} = \begin{cases} 1, & \text{ha az } i \text{ munka megelőzi a } j \text{ munkát a } g \text{ gépen,} \\ 0 & \text{különben;} \end{cases}$
- t_{ig} – az i-edik munka g gépen végzendő műveletének kezdeti időpontja.

Két feltételcsoportot kell felírnunk. Az első azt biztosítja, hogy ugyanannak a munkának a műveletei a meghatározott sorrendben kövessék egymást, és időben ne legyen átfedés közöttük, míg a második csoport különböző munkák ugyanazon a gépen végzendő műveletei közti átfedést akadályozza meg.

Az első esetben egyszerűen csak azt kell biztosítani, hogy az i munka k és $k+1$ műveletének megkezdése között legalább $p_{ig_{ik}}$ idő elteljen, azaz a

$$t_{ig_{ik}} + p_{ig_{ik}} \leq t_{ig_{i,k+1}}, \quad i = 1, \ldots, n, \;\; k = 1, \ldots, m_i - 1 \tag{28.58}$$

egyenlőtlenségek teljesülését kell megkövetelni. Tegyük fel, hogy $g = g_{ik} = g_{jl}$, azaz az i munka k és a j munka l műveletét azonos gépen kell végezni. Ekkor a

$$t_{jg} - t_{ig} \geq p_{ig}, \quad t_{ig} - t_{jg} \geq p_{jg} \tag{28.59}$$

egyenlőtlenségek közül egynek teljesülnie kell. Az első azt fejezi ki, hogy a g gépen i megelőzi a j munkát, azaz $x_{ijg} = 1$. Ennek felhasználásával a

$$M x_{ijg} + t_{ig} - t_{jg} \geq p_{jg} \;, \tag{28.60}$$

$$M(1 - x_{ijg}) + t_{jg} - t_{ig} \geq p_{ig} \tag{28.61}$$

egyenlőtlenségek adódnak. Ha $x_{ijg} = 1$, akkor az első automatikusan teljesül, míg a második kikényszeríti, hogy a j munka megfelelő idővel később kerüljön a g gépre, mint i, és ha $x_{ijg} = 0$, akkor éppen fordítva. A változókra természetes módon adódnak a

$$\text{minden } i, j, g \text{ esetén } \;\; 0 \leq t_{ig} \leq M, \quad x_{ijg} \in \{0,\; 1\} \tag{28.62}$$

megszorítások. További feltételekre lehet szükség a célfüggvény kezelésére, illetve akkor, ha a munkákra határidők is vannak. Az utóbbi esetben azt kell megkövetelni, hogy az egyes munkák utolsó művelete a határidőnél ne később fejeződjék be, azaz

$$t_{ig_{im_i}} + p_{ig_{im_i}} \leq d_i, \quad i = 1, \ldots, n. \tag{28.63}$$

A célfüggvények közül $\sum w_j C_j$ és így $\sum w_j L_j$ írható fel a legkönnyebben, hiszen az előbbi

$$\sum_{i=1}^{n} w_i (t_{ig_{im_i}} + p_{ig_{im_i}}) = \sum_{i=1}^{n} w_i t_{ig_{im_i}} + \sum_{i=1}^{n} w_i p_{ig_{im_i}} \ .$$

Az utóbbi tag állandó, így el is hagyható. Egy új változóra, legyen ez C, és n további feltételre van szükség $C_{\max}$ kezeléséhez. Az új feltételek

$$C \geq t_{ig_{im_i}} + p_{ig_{im_i}} \quad i = 1, \ldots, n \ .$$

A célfüggvény megfelelő alakja pedig

$$\min C \ .$$

Hasonlóképpen intézhető el $L_{\max}$ is, csak ott C_i értéke helyett L_i értékére kell az egyenlőtlenségeket megkövetelni.

Vegyük számba a modell változóit és feltételeit. Mint láttuk, x_{ijg} és x_{jig} közül elég csak az egyiket bevezetni, így ezek maximális száma $n(n-1)m/2$. Mivel minden munka csak egyszer kerülhet egy gépre, ezért a t_{ig} változók száma legfeljebb nm, összesen pontosan $\sum_{i=1}^{n} m_i$. A (28.58) feltételek száma munkánként $m_i - 1$, azaz összesen $\sum_{i=1}^{n}(m_i - 1) \leq nm - n$. A (28.60)–(28.61) feltételeké pedig legfeljebb $n(n-1)m$.

A következő másik modell általánosabb, és tekintettel tud lenni különböző erőforrásokra is. Ez a modell egy ú.n. *oszlopgeneráló eljárás* alapja.

Matematikai programozásban akkor neveznek így egy módszert, ha az egy olyan modellen (feladaton) alapszik, aminek explicit felírása nem lehetséges, mert az legalább egyenértékű volna a feladat megoldásával, de a modell, illetve annak matematikai tulajdonságai lehetővé teszik, hogy egyre újabb oszlopokat, azaz változókat (a hozzájuk tartozó együtthatókkal együtt), vezessünk be a lineáris feltételeket tartalmazó modell megoldása során, és teszteljük a mindenkori részmegoldás optimalitását. Így végső soron a feladatot annak explicit felírása nélkül tudjuk megoldani.

A modell a $J \mid res \mid \sum f_j$ feladatra vonatkozik, azzal a további könnyítéssel, hogy nem követeli meg, hogy az egyes munkákhoz tartozó műveletekre egyértelműen előírt sorrend legyen, hanem megengedi, hogy a szükséges

megelőzési feltételeket munkánként külön-külön egy háló írja le.

Viszont feltesszük, hogy minden j esetén az f_j célfüggvény reguláris, továbbá, hogy a megmunkálási idők egészek.

Legyen a j munka műveleteinek halmaza O_j. A j munka ütemezésének nevezzük a t_α számokat, ha minden $\alpha, \beta \in O_j$ esetén $t_\alpha + p_\alpha \leq t_\beta$ teljesül, feltéve, hogy az α műveletnek meg kell előznie a β műveletet. Itt t_α egyben azt is jelenti, hogy minden α művelet esetén a műveletet a $[t_\alpha, t_\alpha + p_\alpha]$ időintervallumban végzik el. Egy további megszorítás, hogy feltesszük, hogy a feladat véges T időkorláton belül megoldható. Az adatok egész voltából következik, hogy feltehető a t_α számok egész volta is. Végül azt a következtetést kapjuk, hogy csak véges sok ütemezés van, ami közül választanunk kell. Ezt a számot a j munka esetén w_j jelöli.

Az erőforrásokra vonatkozó feltételezések a következők. A figyelembe veendő erőforrások száma s. Minden α művelet az elvégzése során állandó intenzitással használja valamennyi erőforrást, ide értve azt az esetet is, amikor a művelet elvégzéséhez nem kell valamely erőforrás. Ezt a mennyiséget, vagyis azt, hogy az α művelet mennyit igényel a k erőforrásból, $b_{\alpha k}$ jelöli. Ilyen módon bármely j munka és annak h ütemezése esetén megmondható, hogy a k erőforrásból a t időpontban az adott munkának mekkora mennyiségre van szüksége, amit a_{jhkt} jelöl. A k erőforrásból a t időpontban elérhető mennyiség e_{kt}.

Hasonlóképpen bármely j munka és h ütemezés esetén egyértelműen meghatározott a j munka során felmerülő költség, ami f_{jh}.

Az alapfeladatban a költségeket akarjuk minimalizálni úgy, hogy semmikor sem lépjük túl az erőforrások szabta korlátokat és minden munkára pontosan egy ütemezést választunk ki. Így a fenti jelölésekkel az alábbi feladathoz jutunk:

$$\min \sum_{j=1}^{n} \sum_{h=1}^{w_j} f_{jh} x_{jh} \tag{28.64}$$

$$\sum_{j=1}^{n} \sum_{h=1}^{w_j} a_{jhkt} x_{jh} \leq e_{kt}, \quad k = 1, \ldots, s, \ t = 0, \ldots, T \ , \tag{28.65}$$

$$\sum_{h=1}^{w_j} x_{jh} = 1 \quad j = 1, \ldots, n \ , \tag{28.66}$$

$$x_{jh} \in \{0, 1\}, \quad j = 1, \ldots, n, \ h = 1, \ldots, w_j \ . \tag{28.67}$$

Tehát x_{jh} az a döntési változó, ami akkor és csak akkor 1, ha a j munka h ütemezését használjuk, különben 0. Itt a (28.65) egyenlőtlenségek jelentik az ú.n. összekötő feltételeket. Ha ezek nem volnának, akkor a feladat igen egyszerű lenne, ugyanis minden munkára a legkisebb költségű ütemezést kellene választani.

Legyenek a λ_{kt} számok tetszőleges rögzített, nemnegatív mennyiségek ($k = 1, \ldots, s;\ t = 0, \ldots, T$). Könnyen látható, hogy ekkor a (28.64)–(28.67) feladat célfüggvényének értékét minden megengedett megoldás esetén alulról becsüli az alábbi, ú.n. Lagrange-feladat célfüggvényének értéke a változók ugyanazon értéke mellett

$$\min \sum_{j=1}^{n} \sum_{h=1}^{w_j} f_{jh} x_{jh} + \sum_{k=1}^{s} \sum_{t=0}^{T} \sum_{j=1}^{n} \sum_{h=1}^{w_j} a_{jhkt} x_{jh} \lambda_{kt} - \sum_{k=1}^{s} \sum_{t=0}^{T} e_{kt} \lambda_{kt} \ , \quad (28.68)$$

$$\sum_{h=1}^{w_j} x_{jh} = 1, \quad j = 1, \ldots, n \ ; \quad (28.69)$$

$$x_{jh} \in \{0,\ 1\} \quad j = 1, \ldots, n \ h = 1, \ldots, w_j \ . \quad (28.70)$$

A középső tagban az összegzés sorrendjét megváltoztatva a célfüggvény az alábbi alakra hozható

$$\min \sum_{j=1}^{n} \sum_{h=1}^{w_j} \left(f_{jh} \ + \ \sum_{k=1}^{s} \sum_{t=0}^{T} a_{jhkt} \lambda_{kt} \right) x_{jh} \ - \ \sum_{k=1}^{s} \sum_{t=0}^{T} e_{kt} \lambda_{kt} \ . \quad (28.71)$$

A fentiek csak a modell felső szintjét jelentik, amelyben csak közvetve, az a_{jhkt} számokon keresztül, jelennek meg a műveletek ütemezései és erőforrásfelhasználásai. Az eddig mondottakat ütemezések megkonstruálására akarjuk felhasználni. Ezt a célt szolgálja a célfüggvény (28.71) formája is. Itt ugyanis jól látható egy munka ütemezésének hatása a Lagrange-feladat célfüggvényére. A továbbiakban ezt a tagot akarjuk becsülni. Az α tevékenységhez szükséges erőforrások felhasználása a fenti jelölések mellett

$$u_\alpha(t_\alpha) = \sum_{k=1}^{s} b_{\alpha k} \sum_{t=t_\alpha}^{t_\alpha + p_\alpha - 1} \lambda_{kt}$$

értékkel járul hozzá az adott ütemezésnek a célfüggvénybeli együtthatójához, vagyis a

$$\sum_{k=1}^{s} \sum_{t=0}^{T} a_{jhkt} \lambda_{kt}$$

összeghez. Tegyük fel, hogy a j munkából már ütemeztük az $\hat{O}_j \subset O_j$ tevékenységeket. Ekkor az előző modell jelöléseit használva az éppen konstruálandó h ütemezés együtthatója legalább

$$\max\{f_j(t_\alpha + p_\alpha + q_\alpha) : \alpha \in \hat{O}_j\} + \sum_{\alpha \in \hat{O}_j} u_\alpha(t_\alpha)$$

lesz.

A további részleteket elhagyjuk. Az eljárás lényege, amit a fentiekben igyekeztünk demonstrálni, az, hogy ilyen módon az ütemezések konstrukciója során mindvégig becsülni tudjuk a célfüggvény értékét alulról, és így ki lehet szűrni a nem elegendően jó ütemezéseket.

A felsorolt adatokból is látható, hogy bár a kezelhető feladatok mérete állandóan növekszik, azonban számos gyakorlati probléma mérete jóval nagyobb, így érthető, hogy a heurisztikus módszereknek nagy jelentőségük van.

Mielőtt azonban ezekre rátérnénk, a pontos eredmények közül utolsónak Jackson észrevételét adjuk közre, amelyben kiterjesztette Johnson módszerét a kétgépes, többutas probléma megoldására.

28.46. tétel. *Legyen a $J2 \mid m_j \leq 2 \mid C_{\max}$ feladat esetén*

- *J_1 azon munkák halmaza, amelyeket csak az első gépen kell megmunkálni,*
- *J_2 azon munkák halmaza, amelyeket csak a második gépen kell megmunkálni,*
- *J_{12} azon munkák halmaza, amelyeket először az első gépen, utána a második gépen kell megmunkálni,*
- *J_{21} azon munkák halmaza, amelyeket először a második gépen, utána az első gépen kell megmunkálni.*

Ekkor a feladat egy optimális ütemezése a következő:

- *az első gépen először a J_{12}-beli munkák itteni műveleteit végezzük el a Johnson-algoritmus szerinti sorrendben, utána a J_1-beli műveleteket tetszőleges sorrendben, végül a J_{21}-beli munkák itteni műveleteit ugyancsak a Johnson-algoritmus szerinti sorrendben.*
- *a második gépen először a J_{21}-beli munkák itteni műveleteit végezzük el a Johnson algoritmus szerinti sorrendben, utána a J_2-beli műveleteket tetszőleges sorrendben, végül a J_{12}-beli munkák itteni műveleteit ugyancsak a Johnson-algoritmus szerinti sorrendben.*

Megjegyzés. Johnson algoritmusát külön-külön alkalmazzuk a J_{12}- és J_{21}-beli munkákra. Azt szokás mondani, hogy a J_1- és J_2-beli munkák esetén csak egy művelet van. Nem jelent megszorítást azonban, ha akárhány műveletet is

megengedünk, mint azt látni fogjuk. Annyi azonnal látható, hogy akárhány műveletet is kelljen elvégezni egy munkán, ezek megtehetők egyetlen menetben, hiszen mihelyst véget ért egy művelet, a munkadarab azonnal feltehető ugyanarra a gépre.

Bizonyítás. Tegyük fel, hogy az első gépen a műveletek sorrendje nem azonos az állításbelivel. Ha két szomszédos művelet közül

- az első J_1-beli, a második J_{12}-beli, akkor ezek felcserélése nem változtatja meg az állásidőt az első gépen és nem növeli a második gépen,
- az első J_{21}-beli, a második J_{12}-beli, akkor ezek felcserélése nem növeli az állásidőt egyik gépen sem,
- mindkettő J_{12}-beli, de nem a Johnson algoritmus szerinti sorrendben vannak, akkor ezek felcserélése nem változtatja meg az állásidőt az első gépen és nem növeli a második gépen.

Tehát az állításban szereplő sorrend az első gépen semmilyen más sorrendnél sem rosszabb. Hasonló átrendezések végezhetők szükség esetén a második gépen is. ■

28.6.4. Heurisztikus módszerek

A továbbiakban rátérünk a heurisztikus módszerek tárgyalására.

Mindvégig feltesszük, hogy a célfüggvény reguláris. Az alább bevezetendő fogalmak ilyenekhez előállítandó közelítő megoldások leírására szolgálnak.

28.47. definíció. *Egy ütemezést **aktívnak** nevezünk, ha egyetlen műveletet sem lehet időben előbbre tolni anélkül, hogy valamely más művelet ütemezését hátrébb kellene csúsztatni.*

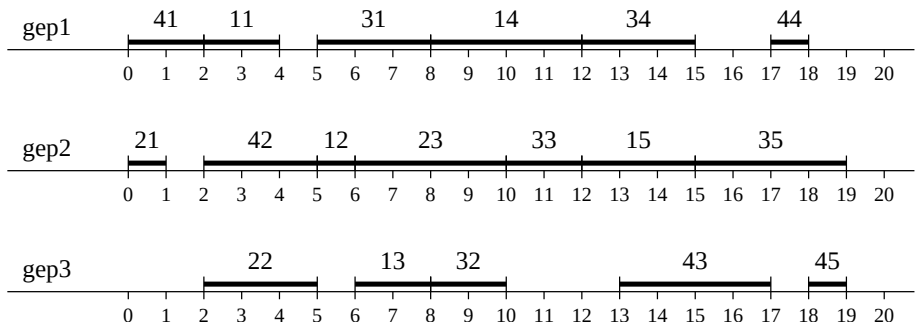
28.9. példa. ábrán látható ütemezéshez jutunk.

Reguláris célfüggvény esetén mindig van aktív optimális megoldás, mert egy kezdési időpont előbbre hozása egy másik kezdési időpont növelése nélkül nem növelheti a célfüggvény értékét. Az aktív ütemezések úgy is felfoghatók, hogy először meghatározzuk a műveletek sorrendjét a gépeken, aztán minden művelet kezdési időpontját a lehető legkorábbinak választjuk úgy, hogy a gépeken előírt sorrend ne sérüljön meg.

Az aktív ütemezések egy fontos alosztálya a következő:

28.48. definíció. *Egy ütemezés **állásidő nélküli,** ha minden gép dolgozik, ha egyáltalán van elvégezhető művelet.*

28.10. példa. 3.15. példa: Az előző példában szereplő módosított optimális



28.9. ábra. Egy optimális megoldás, mely aktív ütemezés.

megoldás nem állásidő nélküli, mert a 43 művelet már $t = 5$-ben rendelkezésre áll. Nem előnyös azonban ekkor megkezdeni, mert akkor mind a 13, mind a 32 műveletet kitolnánk, és így végső soron megnövelnénk a teljes átfutási időt.

A példa mutatja, hogy az állásidő nélküli ütemezés nem ad mindig optimális megoldást. Az irodalom mégis nagy figyelmet szentel neki, mint természetes heurisztikának, amely különösen nagyméretű feladatoknál jó közelítő megoldásokat ad.

Az állásidő nélküli ütemezések előállítása általában egy ú.n. prioritási szabályon alapszik. Ilyeneket láttunk egyetlen gép és párhuzamos berendezések esetében, az utóbbinál lista szerinti ütemezés volt a neve. A leggyakrabban alkalmazott szabályok

- az SPT sorrend,
- az LPT sorrend,
- a legkevesebb megmaradt megmunkálás (LWR), ahol munkánként megvizsgálják a még hátralévő megmunkálások összidejét, és annak a munkának a soron következő megmunkálását választják, ahol az összidő a legrövidebb,
- a leghosszabb megmaradt megmunkálás (rövidítve MWR) az előzőnek a fordítottja,
- a hátralévő műveletek maximális száma (MOPRNR),
- a FIFO sorrend,
- az EDD sorrend.

A második kettő az első kettő általánosításának tekinthető. Általában az a helyzet, hogy ha van egy szabály, akkor annak az ellenkezőjét is használják bizonyos esetekben. Például az említett FIFO sorrend mellett előfordul a LIFO sorrend alkalmazása is.

Egy prioritási szabályon alapuló mohó elindítási módszer az alábbi két követelményt teljesíti:

1. Ha egy munkadarab megérkezik egy éppen nem dolgozó gép elé, akkor a gép azonnal megkezdi a megmunkálását.

2. Ha egy megmunkálás véget ér, akkor a munkadarab azonnal megérkezik a technológiai útvonalban soron következő gép elé. A most felszabaduló gép pedig az előtte várakozó munkadarabok közül a prioritási szabály alapján kiválasztott megfelelő műveletét kezdi meg azonnal.

28.11. példa. Nézzük meg, hogy mit ad a mohó módszer az MWR prioritási szabály alapján a 28.8 példa feladatára. A prioritási szabályhoz felhasználandó értékek műveletenként a következők:

$$\begin{array}{lllll} pr_{11}=12 & pr_{12}=10 & pr_{13}=9 & pr_{14}=7 & pr_{15}=3 \\ pr_{21}=8 & pr_{22}=7 & pr_{23}=4 & & \\ pr_{31}=14 & pr_{32}=11 & pr_{33}=9 & pr_{34}=7 & pr_{35}=4 \\ pr_{41}=11 & pr_{42}=9 & pr_{43}=6 & pr_{44}=2 & pr_{45}=1 \end{array}$$

Az ütemezés elkészítését az alábbi táblázat foglalja össze. Ha ebben a munkák alatt egyetlen szám található, akkor a munka éppen azon a gépen van. Ha pedig például $1v12$ áll egy munka oszlopában, akkor ez azt jelenti, hogy a munka az első gép előtt vár 12 prioritási értékkel.

idő	1. munka	2. munka	3. munka	4. munka	1. gép	2. gép	3. gép
1.	$1v12$	2	1	$1v11$	31	21	–
2.	$1v12$	3	1	$1v11$	31	–	22
3.	1	3	$3v11$	$1v11$	11	–	22
4.	1	2	3	$1v11$	11	23	32
5.	$2v10$	2	3	1	41	23	32
6.	$2v10$	2	$2v9$	1	41	23	–
7.	$2v10$	2	$2v9$	$2v9$	–	23	–
8.	2	–	$2v9$	$2v9$	–	33	13
9.	3	–	2	$2v9$	–	33	13
10.	3	–	2	$2v9$	–	33	13
11.	1	–	$1v7$	2	14	42	–
12.	1	–	$1v7$	2	14	42	–
13.	1	–	$1v7$	2	14	42	–
14.	1	–	$1v7$	3	14	–	43
15.	2	–	1	3	34	15	43
16.	2	–	1	3	34	15	43
17.	2	–	1	3	34	15	43
18.	–	–	2	1	44	35	–
19.	–	–	2	3	–	35	45
20.	–	–	2	–	–	35	–
21.	–	–	2	–	–	35	–

Ha a cél a teljes átfutási idő minimalizálása, akkor ez az eredmény nem olyan jó, mint az optimális megoldásé, de értéke közel van hozzá.

Érdemes összehasonlítani a pontos módszereket a fenti heurisztikus módszerekkel. A pontos módszerek az egész termelő rendszerre, ide értve most a termékeket is, vonatkozó teljes információt követelnek meg, hiszen csak így lehet a megfelelő modellt felállítani. Ezzel szemben a mohó módszer valós idejű eljárásnak tekinthető, hiszen mindig csak akkor dönt, azaz választ ki egy műveletet, ha erre szükség van. Ehhez nem használ mást, mint a rendszer pillanatnyi állapotának ismeretét.

Tehát teljesen figyelmen kívül hagyja azokat a műveleteket, amelyek az adott pillanatban nem végezhetők el, függetlenül attól, hogy ennek mi az oka, azaz, hogy a megelőző műveletek még nem fejeződtek be, vagy a munkadarab még meg sem érkezett a rendszerbe, és függetlenül attól is, hogy ezek későbbi megjelenése milyen hatást okoz. Ennek megvan az az előnye, hogy azonnal reagálni képesek váratlan eseményekre. Például egy munkadarab meghibásodása esetén az ezen végzendő további műveletek egyszerűen nem jelennek meg, és így nincs szükség az előre elkészített sorrendek átdolgozására. Ez a módszer tulajdonképpen nem más, mint a rendszer működésének szimulációja, ahol az egyes gépek irányítását a műveletek szintjén egy nagyon egyszerű szabály határozza meg.

A mohó módszer hátrányaként említhető, hogy egy prioritási szabály bizonyos célokat jól szolgál, ugyanakkor nagyon rossz eredményeket adhat más vonatkozásban. Egy tipikus példa erre, hogy az SPT szabály kiváló az átlagos átfutási idő minél alacsonyabb értéken való tartására, ugyanakkor számos esetben a határidők súlyos megsértéséhez vezet.

A többutas probléma kezelésének fenti két megközelítésében külön-külön rejlő nehézségek leküzdésének egy természetes módja a *kétszintes modell,* ahol a felső szinten a többutas probléma egy diszkrét programozási modellje található. Elképzeléseik szerint ezt a problémát csak ritkán, például naponta vagy hetente oldják meg. Az alatta lévő szinten döntik el gépenként, hogy a gép előtt várakozó műveletek sorából éppen melyiket végezzék el. A felső szint eredményei alapján számolnak költségeket minden művelet minden (diszkrét) időpontban való megkezdésére. Ezek a költségek fejezik ki az alsó szinten az egyes műveletek fontosságát, azaz súlyát. Az éppen elvégzendő műveletnek valamely prioritási szabály szerint első elemét választják. A költségeket pedig ezen feladat optimális Lagrange-szorzói alapján számítják.

Gyakorlatok

28.6-1. Mi a 28.45. tétel párja, ami arra ad feltételt, hogy az α műveletnek

az M-beli műveletek előtt kell állnia?

28.6-2. Írjuk fel a $Pm||C_{\max}$ feladat diszjunktív gráf modelljét.

Feladatok

28-1 SPT optimalitása
Bizonyítsuk be, hogy az SPT sorrend optimális megoldása az $1 \,||\, \sum L_j$ feladatnak.

28-2 Késésmentes optimális megoldás
Bizonyítsuk be, hogy ha az $1 \mid d_j \mid \sum C_j$ feladatnak van olyan optimális megoldása, amiben nincs késés, akkor ezen megoldásban az α megmunkálás akkor és csak akkor állhat az utolsó helyen, ha $d_\alpha \geq \sum_1^n p_j$ és minden i : $d_i \geq \sum_1^n p_j$ esetén $p_\alpha \geq p_i$.

28-3 Minimális késés maximalizálása
Legyen egy $1 \mid d_j \mid f$ feladat esetén $[1], [2], \ldots, [n]$ olyan sorrend, amelyre teljesül, hogy

$$d_{[1]} - p_{[1]} \leq d_{[2]} - p_{[2]} \leq \cdots \leq d_{[n]} - p_{[n]} \,.$$

Bizonyítsuk be, hogy ez a sorrend maximalizálja a minimális késést. (*Útmutatás.* Vegyük észre, hogy betervezett állásidő használata nem megengedett. Alkalmazzuk a szokásos „felcseréléses" módszert.)

28-4 Legjobb és legrosszabb lista
Tegyük fel, hogy m azonos gépünk van, amelyeken $2m-1$ megmunkálást kell elvégezni. A megmunkálási idők közül egy értéke m, további $m-1$ értéke $m-1$ és még további $m-1$ értéke 1. Adjuk meg a legjobb és a legrosszabb értéket adó listát.

28-5 LPT-korlát élessége
Bizonyítsuk be, hogy ha m gép van és a megmunkálási idők $2m-1, 2m-1, 2m-2, 2m-2, \ldots, m+1, m+1, m, m, m$, akkor tetszőleges m esetén az LPT ütemezés pontossága a 28.23. tételben megadott felső korláttal egyezik meg.

28-6 Azonos párhuzamos gépek
Általánosítsuk a 9.4-2. gyakorlat eredményét két gép és tetszőleges m mellett a $2m-1$, $2m-1$, $2m-2$, $2m-2$, $\ldots$, $m+1$, $m+1$, m, m, m megmunkálási idők esetére.

28-7 Egyutas ütemezési feladat
Oldjuk meg az alábbi egyutas ütemezési feladatot. Ábrázoljuk az optimális megoldást Gantt-diagrammal.

munka	1. gép	2. gép	3. gép
1.	3	22	2
2.	22	20	20
3.	20	14	18

28-8 Kétgépes feladat

Bizonyítsuk be, hogy ha az $F3 \mid\mid C_{\max}$ probléma esetén a

$$\min_j t_{1j} \; > \; \max_j t_{2j}$$

és a

$$\min_j t_{3j} \; > \; \max_j t_{2j}$$

feltételek legalább egyike teljesül, akkor a feladat optimális megoldása ugyanaz, mint azé a kétgépes feladaté, ahol a megmunkálási idők munkánként

$$t_{1j} \; + \; t_{2j}, \;\; t_{1j} \; + \; t_{2j}.$$

Útmutatás. Alkalmazzuk a 28.34. tétel bizonyításának módszerét.

28-9 Többutas ütemezés

Tekintsük a többutas ütemezési feladatot. Bizonyítsuk be, hogy ha M páronként diszjunktív műveletek egy tetszőleges halmaza és α egy további művelet, mely diszjunktív valamennyi M-belivel, végül $C > 0$ egy tetszőleges szám, akkor

$$\min_{\beta \in M} r_\beta \; + \; \sum_{\beta \in M} p_\beta \; + \; p_\alpha \; + \; q_\alpha \; \geq \; C$$

esetén α egyetlen olyan ütemezésben sem követheti valamennyi M-beli műveletet, amelyben a teljes átfutási idő kevesebb, mint C.

28-10 Diszjunkt gráf modell

Tegyük fel, hogy egy, a diszjunktív gráf modellen alapuló eljárás során a konjunktív élek pillanatnyi halmaza B. Tekintsünk egy diszjunktív $\{(\alpha, \beta), (\beta, \alpha)\}$ élpárt, amelyiknek egyik tagja sincs B-ben. Adjunk egyszerű elegendő feltételt arra, hogy az élpár egyik tagjának hozzá vétele se hozzon létre kört a gráfban.

28-11 Többutas probléma célfüggvényei

A többutas probléma (28.58)–(28.62) modellje esetén írjuk fel a szükséges feltételek és változók bevezetésével a $\sum w_j T_j$ és a $\sum w_j U_j$ célfüggvényeket.

Megjegyzések a fejezethez

Az ütemezési algoritmusok tulajdonságainak jó összefoglalója Bruckner [38] 2007-ben megjelent monográfiája.

Az ütemezési feladatok tömör leírásának rendszere Grahamtől [104] és Lawlertől [174] származik.

A Gantt-diagramok használatát Henry Laurence Gantt javasolta 1903-ban.

Az $1 \mid r_j, d_j \mid L_{\max}$ feladat esetében már nem ilyen egyszerű a helyzet, mert a probléma teljes általánosságában NP-teljes.A 9.5. tétel forrása [123, 262].

Az SPT sorrend fogalmát Schmidt [272] vezette be.

A 28.7. tétel alapjául szolgáló algoritmus megtalálható Lawler cikkében [173].

Az 28.15. és 28.16. tételek a témakör legkorábbi eredményeihez tartoznak és McNaughton cikkében [200] találhatók meg.

Az 28.18. tétel bizonyítása Horntól [119] származik.

A 28.24. tétel, a 28.25. tételt megelőző elemzés, valamint a 28.27. tétel forrása Coffman, Garey és Johnson cikke [51]. A 28.27. tételben szereplő korlátot Friesen [91] 1.20-ra javította. A módszert magát Friesen és Langston [92] finomították. Ennek a változatnak az ismert hibakorlátja már $72/61+2^{-k}$. Bár a szükséges műveletek számának nagyságrendje nem változott, az a konstans, amit a nagy ordó tartalmaz, igen nagy. A további munkák közül megemlítjük még Sahni cikkét [258], ahol a korlát $1 + 1/k$, de a műveletek száma $O(n(n^2k)^{m-1})$. Igaz továbbá, hogy minden m esetén $r_m > 1$ (lásd a *28-7.* feladatot).

A $P \mid pmtn, d_j \mid U_{\max}$ feladatra, vagyis ahol csak a határidőket betartó, megengedett ütemezés megtalálása a feladat, ha ilyen létezik, illetve annak kimutatása, ha nem létezik, Sahni [259] javasolt $O(n \lg nm)$ futási idejű algoritmust, amely olyan megoldást állít elő (ha van megoldás), amelyben legfeljebb $n - 2$ megmunkálást kell megszakítani.

Az $U_{\max}$ célfüggvényre vonatkozó, a 28.19. tételhez hasonló állítás Hujter Mihálytól [123] származik.

Az első, dinamikus programozással elért ütemezési eredmény Rothkopf [254] nevéhez fűződik. Lawler és társai [174] később egyszerűsítették a Rothkopf által használt képleteket.

A $C(L)/C^*$ hányadosra vonatkozó első eredményt – a 28.22. tételt – Graham bizonyította 1966-ban [102]. Ugyancsak Grahamtől származik a 28.23. tétel.

A ládapakolási algoritmusokra vonatkozó eredmények összefoglalása megtalálható Coffman, Csirik János és Woeginger [52] cikkében.

A korlátozás és szétválasztás részletes leírása megtalálható Vizvári Béla jegyzetében [310]. Az egyutas ütemezési problémával részletesen foglalkoznak a [109, 166, 199, 291, 289, 290] dolgozatok.

Az egyutas problémának a (28.42) képlethez kapcsolódó változatát Piehler [238] tanulmányozta.

A diszjunktív gráf modell először a [256] dolgozatban jelent meg.

A (28.47)–(28.50) képletek közvetlenül adódnak [310] 5. fejezete alapján.

Az irodalomban ismert alsó korlátok egy nagyon jó összefoglalását adja [165]. A (28.52) korlátot gyengébb formában, a harmadik tag nélkül közli [44] és [262]. Maga a korlát élesíthető a 28.4. tétel alapján. A (28.53) korlát először a [132] cikkben fordult elő, míg (28.54) megfordítása [36]-ben.

A 28.45. tétel alapja [42].

A kritikus blokk fogalmát Grabowski [100] vezette be. A kritikus blokkal kapcsolatos további részletek találhatók a [21] dolgozatban.

Az egészértékű programozás első modellje [195] alapján terjedt el a szakirodalomban. A másik tárgyalt modell több vonatkozásban általánosabb, és tekintettel tud lenni különböző erőforrásokra is [83, 82]. Ebben a fejezetben csak a modellt tárgyaljuk – a Fisher által javasolt módszer részletesen megtalálható a [310] könyvben.

A (28.64)–(28.67) feladat célfüggvényének elemzése [310] 7. fejezetéből származik.

Az általános egészértékű programozási feladatra vonatkozóan eddig említett módszerek csak szerény méretű feladatokat képesek pontosan megoldani. [18] még nem számol be számítástechnikai eredményekről. [83] esetében a legnagyobb vizsgáltproblémában a munkák száma 5, az erőforrásoké 4. [165] mindössze három feladatot oldott meg, melyek közül a legnagyobb esetén a műveletek száma 36 és mind a munkák, mind a gépek száma 6. [21] már tovább növelte a méretet, a pontosan megoldott legnagyobb feladatokban 8 gép és 64 művelet volt. [42] az, amelyben először oldottak meg egy, az irodalomból jól ismert, és állandó kihívást jelentő 10 munkát és 10 gépet tartalmazó feladatot, továbbá ugyanez a dolgozat beszámol számos, a korábbiaknál nagyobb probléma, például20 munka, 5 gép és számos művelet megoldásáról is.

Johnson módszerének a kétgépes, többutas problémára való kiterjesztése Jackson [133] érdeme.

A többutas probléma kezelésére Roundy és Maxwell [255] javasolták a kétszintes modellt.

Az on-line algoritmusokra vonatkozó eredmények összefoglalása megtalálható ennek a könyvnek a második kötetében (**??**. fejezet).

Az ütemezéselmélet egyik fontos alkalmazási területe a számítógépes feladatmegoldás. Ezzel kapcsolatban ajánljuk Blazewicz és társai [32], Coffman

[50] és Pinedo [239, 240] monográfiáit. Pinedo két ajánlott könyve a determinisztikus ütemezés mellett a sztochasztikus ütemezés módszereit és eredményeit is tárgyalja.

A *28-2*. feladat forrása [272], a *28-4*. feladat [96]-ból, a *28-5*. feladat [103]-ből, a *28-8*. feladat pedig Johnson cikkéből [137]-ból származik.

Irodalomjegyzék

[1] R. G. Addie, M. Zukerman, T. Neame. Broadband traffic modeling: Simple solutions to hard problems. *IEEE Communications Magazine*, 36(8):88–95, 1998. 133, 134

[2] S. Albers, H. Bals. Dynamic TCP acknowledgement, penalizing long delays. In *Proceedings of the 25th ACM-SIAM Symposium on Discrete Algorithms*, 47–55. o., 2003. 480

[3] G. M. Amdahl. Validity of the single-processor approach to achieving large-scale computer capabilities. In *AFIPS Conference Proceedings*, vol. 30, 483–485. o., 1967. 300

[4] N. Anbarci. Noncooperative foundations of the area monotonic solution. *The Quaterly Journal of Economics*, 108:245–258, 1993. 659

[5] D. Anick, D. Mitra, M. Sondhi. Stochastic theory of a data handling system with multiple sources. *The Bell System Technical Journal*, 61:1871–1894, 1982. 133

[6] arxiv.org. Quantum physics on arxiv site, 2013. quant-ph. 176

[7] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, O. Waarts. On-line load balancing with applications to machine scheduling and virtual circuit routing. *Journal of the ACM*, 44:486–504, 1997. 480

[8] H. Attiya, C. Dwork, N. A. Lynch, L. Stockmeyer. Bounds on the time to reach agreement in the presence of timing uncertainty. *Journal of the ACM*, 41:122–142, 1994. 241

[9] H. Attiya, J. Welch. *Distributed Computing, Fundamentals, Simulations and Advanced Topics.* McGraw-Hill, 1998. 241, 242

[10] J-P. Aubin. *Mathematical Methods of Game and Economic Theory.* North-Holland, 1979. 617

[11] B. Awerbuch. Complexity of network synchronization. *Journal of the ACM*, 32(4):804–823, 1985. 241

[12] B. Awerbuch, Y. Azar S. Plotkin. Throughput-competitive online routing. In *Proceedings of the 34th Annual Symposium on Foundations of Computer Science*, 32–40. o., 1993. 480

[13] Y. Azar. On-line load balancing. In *Lecture Notes in Computer Science, 1442. kötet*, 178–195. o. Springer-Verlag, 1998. 480

[14] L. Bacsárdi. *Efficient Quantum Based Space Communication.* Lambert Academic Publishing, 2013. 135, 176

[15] A. Bagyinszkiné Orosz. Petri-hálók I. ELTE, Numerikus és Gépi Matematika Tanszék, 1977. 385, 386

[16] A. Bagyinszkiné Orosz. *Petri-hálók és formális nyelvek.* PhD thesis, Kandidátusi értekezés, Magyar Tudományos Akadémia, 1979. 386

[17] B. S. Baker, J. S. Schwartz. Shelf algorithms for two dimensional packing problems. *SIAM Journal on Computing*, 12:508–525, 1983. 480

[18] E. Balas. Sequencing via disjunctive graphs: an implicit enumeration algorithm. *Operation Research*, 17:941–953, 1969. 779

[19] J. Balogh, J. Békési, G. Galambos. New lower bounds for certain classes of bin packing algorithms. *Theoretical Computer Science*, 440–441:1–13, 2012. 480

[20] J. Banks, J. Carson, B. Nelson. *Discrete-Event Simulation.* Prentice Hall, 1996. 133

[21] J. R. Barker, G. B. McMahon. Scheduling the general job-shop. *Management Science*, 31:594–598, 1985. 779

[22] N. Bell, M. Garland. Implementing sparse matrix-vector multiplication on throughput-oriented processors. In *SC'09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, 1–11. o. ACM, 2009. 49

[23] J. Beran, R. Sherman, M. Taqqu, W. Willinger. Long-range dependence in variable-bit-rate video traffic. *IEEE Transactions on Communications*, 43:1566–1579, 1995. 133

[24] P. Berman, J. Garay. Cloture votes: $n/4$-resilient distributed consensus in $t+1$ rounds. *Mathematical Systems Theory*, 26(1):3–19, 1993. 242

[25] K. A. Berman, J. L. Paul. *Sequential and Parallel Algorithms*. PWS Publishing Company, 1996. 299

[26] E. Bernstein, U. Vazirani. Quantum complexity theory. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing (STOC '93)*, 11–20. o. ACM, 1993. 176

[27] E. Best, R. Devillers, M. Koutny. *Petri Net Algebra*. Springer-Verlag, 2001. 386

[28] E. Best, R. Hopkins, J. G. Hall. B(PN)2 – a basic Petri net programming notation, in: A. B. M. Reeve and G. Wolf (szerk.). In *Parallel Architectures and Languages Europe (PARLE'93)*, Lecture Notes in Computer Science 684. kötete. Springer-Verlag, 1973, 179–190. 386

[29] G. Bianchi. Performance analysis of the IEEE 802.11 distributed coordination function. *IEEE Journal on Selected Areas of Communications*, 18(3):535–547, 2000. 718

[30] G.-I. Bisci, L. Sbragia, F. Szidarovszky. Learning the demand function in a repeated cournot oligopoly game. *International Journal of Systems Science*, 39(4):403–419, 2008. 618

[31] E. J. Bittner, Cs. Imreh, , J. Nagy-György. Online k-server problem with rejection. *Discrete Optimization*, 13:1–15, 2014. 479

[32] J. Blazewicz, K. Ecker, E. Pesch, G. Schmidt, J. Weglarz. *Scheduling Computer and Manufacturing Processes*. Springer-Verlag, 2001 (második kiadás). 779

[33] G. Bolch, S. Greiner, H. Meer, de, K. Trivedi. *Queueing Networks and Markov Chains, Modeling and Performance Evaluation with Computer Science Applications*. John Wiley & Sons, 1998. 718

[34] A. Borodin R. El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 1998. 479

[35] R. P. Brent. The parallel evaluation of general arithmetic expressions. *Journal of the ACM*, 21:201–206, 1974. 300

[36] G. Brooks, C. R. White. An algorithm for finding optimal or near optimal solutions to the production scheduling problem. *Journal ofIndustrial Engineering*, 16:34–40, 1965. 779

[37] L. E. J. Brouwer. Uber Abbildung von Manningfaltigkeiten. *Mathematische Annalen*, 97–115. o., 1912. 617

[38] P. Brucker. *Scheduling Algorithms*. Springer-Verlag, 2007 (ötödik kiadás). 778

[39] J. E. Burns. A formal model for message passing systems. Technical Report91, Indiana University, 1980. 242

[40] J. E. Burns, N. A. Lynch. Bounds on shared memory for mutual exclusion. *Information and Computation*, 107(2):171–184, 1993. 242

[41] CACI. *COMNET III*. CACI Products Co., 1997. 133

[42] J. Carlier, E. Pinson. Algorithm for solving the job-shop problem. *Management Science*, 35:164–176, 1989. 779

[43] R. Chandra, L. Dagum, D. Kohr, D. Maydan, J. McDonald, R. Menon. *Parallel Programming in OpenMP*. Morgan Kaufmann Publishers, 2000. 300

[44] J. Charlton, C. C. Death. A method of solution for general machine scheduling problems. *Operations Research*, 18:689–707, 1970. 779

[45] Y. Cho, S. Sahni. Bounds for list schedules on uniform processors. *SIAM Journal on Computing*, 9(1):91–103, 1980. 480

[46] M. Chrobak, L. Larmore. An optimal algorithm for k-servers on trees. *SIAM Journal on Computing*, 20:144–148, 1991. 479

[47] M. Chrobak, L. Larmore. The server problem and on-line games. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 7. kötet*. American Mathematical Society, 1992, 11–64. 479

[48] M. Chrobak, H. J. Karloff, T. Payne, S. Vishwanathan. New results on the server problem. *SIAM Journal on Discrete Mathematics*, 4:172–181, 1991. 479

[49] Y. M. Chun. The equal loss principle for bargaining problems. *Economics Letters*, 26:103–106, 1988. 659

[50] E. Coffman. *Computer and Job Shop Scheduling*. John Wiley & Sons, 1976. 780

[51] E. Coffman, M. Garey, D. Johnson. An application of bin-packing to multiprocessor scheduling. *SIAM Journal on Computing*, 7:1–17, 1978. 778

[52] E. G. Coffman, J. Csirik, G. J. Woeginger. Approximate solutions to bin packing problems. In P. M. Pardalos, M. G. C. Resende (szerk.), *Handbook of Applied Optimization*, 607–615. o. Oxford University Press, 2002. 778

[53] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein. *Introduction to Algorithms* 3rd edition. The MIT Press/McGraw-Hill, 2009. 299, 300

[54] M. Crane, A. Lemoine. *An Introduction to the Regenerative Method for Simulation Analysis.* Springer-Verlag, 1977. 718

[55] M. Crovella, A. Bestavros. Self-similarity in world wide web traffic: Evidence and possible causes. *IEEE/ACM Transactions on Networking*, 5(6):835–846, 1997. 133, 134

[56] J. Csirik, G. Woeginger. On-line packing and covering problems. In *Lecture Notes in Computer Science, 1442. kötet.* Springer-Verlag, 1998, 147–177. 480

[57] J. Csirik, G. J. Woeginger. Shelf algorithms for on-line strip packing. *Information Processing Letters*, 63:171–175, 1997. 480

[58] D. E. Culler, R. M. Karp, D. Patterson, A. Sahay, E. E. Santos, K. E. Schauser, R., T. von Eicken. Logp: A practical model of parallel computation. *Communication of the ACM*, 39(11):78–85, 1996. 300

[59] Gy. Dallos. *Hírközlési csatornák véletlen hozzáférési módszerei.* Akadémiai Kiadó, 1984. 699, 718

[60] G. Dantzig. Impact of linear programming on computer development. *OR/MS Today*, 18(8):12–17, 1988. 558

[61] A. Darte, Y. Robert, F. Vivien. *Scheduling and Automatic Parallelization.* Birkhäuser Boston, 2000. 438

[62] I. Deák. *Random Number Generators and Simulation.* Akadémiai Kiadó, 1990. 718

[63] D. Deutsch, R. Jozsa. Rapid solution of problems by quantum computation. *Proceedings of the Royal Society of London; Series A, Mathematical and Physical Sciences*, 439:553–558, 1992. 176

[64] D. Dolev, R. Strong. Authenticated algorithms for Byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983. 242

[65] D. R. Dooly, S. A. Goldman, S. D. Scott. On-line analysis of the TCP acknowledgement delay problem. *Journal of the ACM*, 48:243–273, 2001. 480, 481

[66] Gy. Dósa, Y. He. Better online algorithms for scheduling with machine cost. *SIAM Journal on Computing*, 33(5):1035–1051, 2004. 481

[67] N. Duffield, N. Oconnell. Large deviation and overflow probabilities for the general single-server queue, with applications. *Mathematical Proceedings of the Cambridge Philosophical Society*, 118:363–374, 1995. 133

[68] D. Duffy, A. McIntosh, M. Rosenstein, W. Willinger. Statistical analysis of ccsn/ss7 traffic data from working ccs subnetworks. *IEEE Journal on Selected Areas Communications*, 12:544–551, 1994. 133

[69] D. Eberly. *Game Physics.* Morgan Kaufmann Publishers, 2004. 60

[70] W. Engel (szerk.). *GPU Pro.* A K Peters, 2010. 63

[71] A. Erramilli, O. Narayan, W. Willinger. Experimental queueing analysis with long-range dependent packet-traffic. *IEEE/ACM Transactions on Networking*, 4(2):209–223, 1996. 133

[72] G. Falin, J. Templeton. *Retrial Queues.* Chapman, 1997. 718

[73] Gy. Farkas. A Fourier-féle mechanikai elv alkalmazásai. *Mathematikai és Természettudományi Értesítő*, 12:457–472, 1894. 558

[74] Gy. Farkas. Theorie der einfachen ungleichungen. *Journal für die Reine und Angewandte Mathematik*, 124:1–27, 1898. 558

[75] Gy. Farkas. Paraméteres módszer fourier mechanikai elvéhez. *Mathematikai és Physikai Lapok*, 7:63–71, 1898. 558

[76] W. Feller. *An Introduction to Probability Theory and Its Applications.* John Wiley & Sons *3. kiadás)*, 1968, (Magyarul: *Bevezetés a valószínűségszámításba*, Műszaki Könyvkiadó, 1978). 718

[77] F. Fernando (szerk.). *GPU Gems.* Addison-Wesley, 2004. 63

[78] A. Fiat, Y. Rabani, Y. Ravid. Competitive k-server algorithms. *Journal of Computer and System Sciences*, 48:410–428, 1994. 479

[79] A. Fiat, G. Woeginger (szerk.). *Online Algorithms. The State of Art.* Springer-Verlag, 1998. 479

[80] M. J. Fischer, N. A. Lynch. A lower bound for the time to assure interactive consistency. *Information Processing Letters*, 14(4):183–186, 1982. 242

[81] M. J. Fischer, N. A. Lynch, M. S. Paterson. Impossibility of distributed consensus with one faulty proces. *Journal of the ACM*, 32(2):374–382, 1985. 242

[82] M. L. Fisher. Optimal solution of scheduling problems using lagrange multipliers: Part II,. In S. E. Elmaghraby (szerk.), *Symposium on the Theory of Scheduling and Its Applications.* Springer-Verlag, 1973. 779

[83] M. L. Fisher. Optimal solution of scheduling problems using lagrange multipliers: Part I. *Operations Research*, 21:1114–1127, 1973. 779

[84] R. Fleischer, M. Wahl. On-line scheduling revisited. *Journal of Scheduling*, 3(6):343–353, 2000. 480

[85] M. J. Flynn. Very high-speed computer systems. *Proceedings of the IEEE*, 5(6):1901–1909, 1966. 299

[86] C.-H. Foh, M. Zukerman. Performance analysis of the IEEE 802.11 MAC protocol. In *Proceedings of European Wireless*, 2002. 718

[87] W. Fokkink. *Distributed Algorithms: An Intuitive Approach.* The MIT Press, 2013. 242

[88] F. Forgó, J. Szép, F. Szidarovszky. *Introduction to the Theory of Games: Concepts, Methods and Applications.* Kluwer Academic Publishers, 1999. 617, 659

[89] S. Fortune, J. Wyllie. Parallelism in random access machines. In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, 114–118. o., 1978. 300

[90] I. Foster, C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure* második kiadás. Morgan Kaufman Publisher, 2004. 300

[91] D. K. Friesen. Tighter bounds for the multifit processor scheduling algorithm. *Journal of Algorithms*, 7:35–59, 1984. 778

[92] D. K. Friesen, M. A. Langston. Evaluation of a multifit-based scheduling algorithm. *Journal of Algorithms*, 7:35–59, 1986. 778

[93] R. Frisch. The logaritmic potential method for solving linear programming problems. Memorandum, University Institute of Economics, Oslo, 1955. 559

[94] N. Fujimoto. Faster matrix-vector multiplication on geforce 8800gtx. In *Parallel and Distributed Processing,* IPDPS 2008, 1–8. o. IEEE, 2008. 49

[95] H. Garcia-Molina, J. Seiferas. Elections in a distributed computing systems. *IEEE Transactions on Computers*, C-31(1):47–59, 1982. 241

[96] M. R. Garey, R. L. Graham. Performance guarantees for scheduling algorithm. *Operations Research*, 26:3–21, 1978. 780

[97] P. Gács, L. Lovász. Khachiyan's algorithm for linear programming. *Mathematical Programming Study*, 14:61–68, 1981. 558

[98] J. E. Gentle. *Random Number Generation and Monte Carlo Methods.* Springer-Verlag, 1998. 718

[99] P. B. Gibbons, Y. Matias, V. Ramachandran. Can a shared-memory model serve as a bridging model for parallel computation. *Theory of Computing Systems*, 32(3):327–359, 1999. 300

[100] J. Grabowski. On two-machine scheduling with release and due-dates to minimize maximum lateness. *Opsearch*, 17:133–154, 1980. 779

[101] R. L. Graham. Bounds for certain multiprocessor anomalies. *The Bell System Technical Journal*, 45:1563–1581, 1966. 480

[102] R. L. Graham. Bounds for certain multiprocessor anomalies. *The Bell Systems Technical Journal*, 45:1563–1581, 1966. 778

[103] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17:416–429, 1969. 780

[104] R. L. Graham, E. L. Lawler, J. K. Lenstra, A. H. G. Rinnoy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326, 1979. 778

[105] A. Grama, A. Gupta, G. Karypis, V. Kumar. *Introduction to Parallel Computing* (2. kiadás). Addison-Wesley, 2003. 299

[106] J. N. Gray. Notes on database operating system. In R. Bayer, R. M. Graham, G. Seegmuller (szerk.), *Operating Systems: An Advanced Course*, Lecture Notes in Computer Science, volume 60, 393–481. o. Springer-Verlag, 1978. 242

[107] W. Gropp, M. Snir, B. Nitzberg, E. Lusk. *MPI: The Complete Reference*. Scientific and Engineering Computation Series. The MIT Press, 1998. 300

[108] L. K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing (STOC'96)*, 212–219. o. ACM, New York, NY, 1996. 176

[109] J. N. D. Gupta, S. S. Reddi. Improved dominance conditions for the three-machine flowshop scheduling problem. *Operations Research*, 26:200–203, 1978. 779

[110] R. Gusella. A measurement study of diskless workstation traffic on an Ethernet. *IEEE Transactions on Communications*, 38:1557–1568, 1990. 133

[111] J. Gustafson. Reevaluating Amdahl's law. *Communications of ACM*, 28(1):532–535, 1988. 300

[112] T. Gyires. Simulation of the harmful consequences of self-similar network traffic. *The Journal of Computer Information Systems*, 42(4):94–111, 2002. 133

[113] L. Győrfi, I. Páli. *Tömegkiszolgálás informatikai rendszerekben*. Műegyetemi Kiadó, 1996. 717, 718

[114] G. Hadley. *Nonlinear and Dynamic Programming*. Addison-Wesley, 1964. 617

[115] M. J. Harris, W. V. Baxter, T. Scheuerman, A. Lastra. Simulation of cloud dynamics on graphics hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, HWWS'03, 92–101. o. Eurographics Association, 2003. 54

[116] J. F. Harsanyi, R. Selten. A generalized Nash solution for two-person bargaining with incomplete information. *Management Science*, 12(2):80–106, 1972. 659

[117] H. Hefles, D. Lucantoni. A markov modulated characterization of packetized voice and data traffic and related statistical multiplexer performance. *IEEE Journal on Selected Areas in Communication*, 4:856–868, 1986. 133

[118] F. Heylighen. Principia Cybernetica Project. http://pespmc1.vub.ac.be/HEYL.html, 2004. 134

[119] W. A. Horn. Minimizing average flow time with parallel machines. *Operations Research*, 21:845–847, 1973. 778

[120] E. Horowitz, S. Sahni. *Fundamentals of Computer Algorithms*. Computer Science Press, 1978. 299

[121] P. Horváth, D. Illés. Sph-based fluid simulation in distributed environment. In *MIPRO 2009: 32nd International Convention on Information and Communication Technology, Electronics and Microelectronics*, 249–257. o. Croatian Society for Information and Communication Technology, 2009. 62

[122] P. Huard. Resolution of mathematical programming with nonlinear constraints by the method of centers. In J. Abadie (szerk.), *In Nonlinear Programming*, 207–219. o. North Holland Publ. House, 1967. 559

[123] M. Hujter. Egy ütemezéselméleti probléma vizsgálata és alkalmazásai (Investigation of a scheduling problem and its applications), 1987. Doktori értekezés, ELTE TTK. 778

[124] K. Hwang, Z. Xu. *Scalable Parallel Computing*. McGraw-Hill, 1998. 300

[125] Cs. Imreh. Online strip packing with modifiable boxes. *Operations Research Letters*, 66:79–86, 2001. 481

[126] Cs. Imreh, J. Noga. Scheduling with machine cost. In *Proceedings of APPROX'99, Lecture Notes in Computer Science, 1540. kötet*, 168–176. o., 1999. 481

[127] S. Imre, F. Balázs. *Quantum Computing and Communications: An Engineering Approach*. Wiley, 2005. 135, 176

[128] S. Imre, F. Gyöngyösi. *Quantum Computing and Communications: An Engineering Approach*. Wiley, 2012. 176

[129] Interoute. http://wwww.interoute.com/glossary.html, 2004. 133

[130] A. Iványi. A munkahatékonyság korlátai prefixszámításnál és összefésülésnél. *Alkalmazott Matematikai Lapok*, 2014 (benyújtva). 300

[131] A. Iványi. *Párhuzamos algoritmusok*. ELTE Eötvös Kiadó, 2003. Elektronikusan: *Párhuzamos algoritmusok*, ELTE IK, 2011. 299

[132] J. R. Jackson. Research Report 43, Management Science Research Project, University of California, 1955. 779

[133] J. R. Jackson. An extension of Johnson's results on job shop scheduling. *Naval Research Logistics Quaterly*, 3:201–224, 1956. 779

[134] R. Jain, S. Routhier. Packet trains: Measurements and a new model for computer network traffic. *IEEE Journal on Selected Areas in Communication*, 4:986–995, 1986. 133

[135] J. Jaja. *An Introduction to Parallel Algorithms*. Addison-Wesley, 1992. 299

[136] L. Jereb, M. Telek (szerk.). *Sorbanállásos rendszerek*. BME Hiradástechnikai Tanszék jegyzete, 1999. 718

[137] S. M. Johnson. Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quaterly*, 1:61–68, 1954. 780

[138] D. S. Johnson. Near-optimal Bin Packing Algorithms. PhD értekezés, MIT Department of Mathematics, 1973. 480

[139] D. S. Johnson. Fast algorithms for bin packing. *Journal of Computer and System Sciences*, 8:272–314, 1974. 480

[140] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, R. L. Graham. Worst-case performance-bounds for simple one-dimensional bin packing algorithms. *SIAM Journal on Computing*, 3:299–325, 1974. 480

[141] G. A. Jones, J. Jones. *Information and Coding Theory*. Springer-Verlag, 2000. 133

[142] D. B. Judin, A. S. Nemirovski. Informational complexity and effective methods for the solution of convex extremal problems (in Russian). *Ekonomika i Matematcheskie Metody*, 12:357–369, 1976. 558

[143] S. Kakutani. A generalization of Brouwer's fixed point theorem. *Duke Mathematical Journal*, 8:457–459, 1941. 617

[144] E. Kalai, M. Smorodinsky. Other solution to Nash' bargaining problem. *Econometrica*, 43:513–518, 1975. 659

[145] M. Kandemir, J. Ramanujam, A. Choudhary. Compiler algorithms for optimizing locality and parallelism on shared and distributed-memory machines. *Journal of Parallel and Distributed Computing*, 60:924–965, 2000. 300

[146] S. Karamardian. The nonlinear complementarity problems with applications. I, II. *Journal of Optimization Theory and Applications*, 4:87–98 and 167–181, 1969. 617

[147] S. Karlin, M. T. Taylor. *A First Course in Stochastic Processes*. Academic Press, 1975 (magyarul: *Sztochasztikus folyamatok*, Gondolat Kiadó, 1985). 718

[148] A. R. Karlin, C. Kenyon, D. Randall. Dynamic TCP acknowledgement and other stories about $e/(e-1)$. In *Proceedings stoc31*, 502–509. o., 2001. 480

[149] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984. 558

[150] R. M. Karp, R. E. Miller, S. Winograd. The organization of computations for uniform recurrence equations. *Journal of the ACM*, 14:563–590, 1967. 438

[151] K. Kennedy, R. Allen. *Optimizing Compilers for Modern Architectures*. Morgan Kaufman Publishers, 2001. 300

[152] L. Khacijan. A polynomial time algorithm in linear programming. *Soviet Mathematics Doklady*, 20:191–194, 1979. 558

[153] Khronos. In *OpenCL Overview*, 2010. http://www.khronos.org/opencl/. 20

[154] B. Kiss, A. Krebsz. *Játékelmélet*. Széchenyi István Főiskola, 1999. 617

[155] S. Kleiman, D. Shah, B. Smaalders. *Programming with Threads*. Prentice Hall, 1996. 300

[156] L. Kleinrock. *Queueing Systems*. John Wiley & Sons, 1975 (Magyarul: *Sorbanállás – kiszolgálás: Bevezetés a tömegkiszolgálási rendszerek elméletébe*, Műszaki Könyvkiadó, 1979). 133, 717, 718

[157] C. H. Koelbel, D. B. Loveman, R. S. Schreiber, G. Steele Jr., M. E. Zosel. *The High Performance Fortran Handbook.* The MIT Press, 1994. 300

[158] M. Kojima, S. Mizuno, A. Yoshise. A primal-dual interior point algorithm for linear programming. In N. Megiddo (szerk.), *Progress in Mathematical Programming: Interior Point and Related Methods*, 29–47. o. Springer-Verlag, 1989. 559

[159] E. Koutsoupias, C. Papadimitriou. On the k-server conjecture. *Journal of the ACM*, 42:971–983, 1995. 479

[160] M. Kovács. *Nemlineáris programozás.* Typotex, 1997. 617

[161] L. Kozma, L. Varga. *A szoftvertechnológia elméleti kérdései (Theoretical questions of software technology.* ELTE Eötvös Kiadó, 2003. 242

[162] I. Kátai. *Szimuláció (Simulation).* Tankönyvkiadó, 1981. 133

[163] H. W. Kuhn, A. Tucker. *Contributions to the Theory of Games. II.* Princeton University Press, 1953. 617

[164] H. T. Kung, C. E. Leiserson. Systolic arrays (for VLSI). In I. S. Duff, G. W. (szerk.), *Sparse Matrix Proceedings*, 256–282. o. SIAM, 1978. 438

[165] B. Lageweg, J. K. Lenstra, A. Rinnoy Kan. Job-shop scheduling by implicit enumeration. *Management Science*, 24:441–450, 1977. 779

[166] B. Lageweg, J. K. Lenstra, A. Rinnoy Kan. A general bounding scheme for the permutation flow-shop problem. *Operations Research*, 26:53–67, 1978. 779

[167] L. Lakatos. On a simple continuous cyclic-waiting system. *Annales Universitatis Scientiarum Budapestinensis de Rolando Eötvös Nominatae. Sectio Computatorica*, 14:105–113, 1994. 718

[168] L. Lakatos. Equilibrium distributions for the $M|G|1$ and related systems. *Publicationes Mathematicae Debrecen*, 55:123–140, 1999. 718

[169] L. Lamport. A new solution of dijkstra's concurrent programming problem. *Communications of the ACM*, 18(8):453–455, 1974. 242

[170] L. Lamport. A fast mutual exclusion algorithm. *ACM Transactions on Computers*, 5(1):1–11, 1987. 242

[171] L. Lamport, R. Shostak M. Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982. 242

[172] A. Law, W. Kelton. *Simulation Modeling and Analysis (3. kiadás).* McGraw-Hill Higher Education, 1999. 133

[173] E. L. Lawler. Optimal sequencing of a single machine subject to precedence constraints. *Management Science*, 19:544–546, 1973. 778

[174] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnoy Kan, D. B. Shmoys. Sequencing and scheduling: Algorithms and complexity. In S. C. Graves, A. Kan, P. Zipkin, P. H.szerk.), *Handbooks in Operations Research and Management Science, Volume 4: Logistics of Production and Inventory*, 445–522. o. Elsevier, 1993. 778

[175] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays-Trees-Hypercubes.* Morgan Kaufman Publishers, 1992. 299

[176] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Algorithms and VSLI.* Morgan Kaufman Publishers, 2001. 299

[177] W. Leland, M. Taqqu, W. Willinger, D. Wilson. Statistical analysis and stochastic modeling of self-similar data-traffic. In J. Labetoulle, J. W. Roberts (szerk.), *The Fundamental Role of Teletraffic in the Evolution of Telecommunications Networks.* Elsevier Science B. V., 1994. 133

[178] W. Leland, M. Taqqu, W. Willinger, D. Wilson. On the self-similar nature of ethernet traffic (extended version). *IEEE/ACM Transactions on Networking*, 2(1), 1994. 133

[179] W. Leland, M. Taqqu, D. Wilson. On the self-similar nature of ethernet traffic. *Computer Communication Reviews*, 23, 1993. 133

[180] S. Leonardi. On-line network routing. In *Lecture Notes in Computer Science, 1442. kötet.* Springer-Verlag, 1998, 242–267. 480

[181] C. Leopold. *Parallel and Distributed Computing.* John Wiley & Sons, 2001. 299, 300

[182] B. Lewis, D. J. Berg. *Multithreaded Programming with Phtreads.* Prentice Hall, 1998. 300

[183] M. Listanti, V. Eramo, R. Sabella. Architectural and technological issues for future optical internet networks. *IEEE Communications Magazine*, 38(9):82–92, 2000. 133

[184] L. Lovász, P. Gács. *Algoritmusok*. Műszaki Könyvkiadó és Tankönyvkiadó, 1978. 14

[185] Y. Luke. *Mathematical Functions and their Approximations*. Academic Press, 1975. 718

[186] Y. Luo, F. Szidarovszky, Y. Al-Nashif, S. Hariri. Game theory based wnetwork security. *Journal of Information Security*, 1(1):41–44, 2010. 618

[187] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufman Publisher, 2001 (ötödik utánnyomás. Magyarul: *Osztott algoritmusok*. Kiskapu Kiadó, 2002). 439

[188] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufman Publisher, 2010 (nyolcadik kiadás. Magyarul: *Osztott algoritmusok*, szerk. Iványi A., Kiskapu Kiadó, 2002). 242

[189] N. A. Lynch, M. J. Fischer. On describing the behavior and implementation of distributed systems. *Theoretical Computer Science*, 13(1):17–43, 1981. 241

[190] M. Magdics, G. Klár. Rule-based geometry synthesis in real-time. In W. Engel (szerk.), *GPU Pro: Advanced Rendering Techniques*, 41–66. o. A K Peters, 2010. 26

[191] M. Manasse, L. McGeoch, D. Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, 11:208–230, 1990. 479

[192] B. Mandelbrot. *The Fractal Geometry of Nature*. W. H. Freeman, 1982. 133

[193] B. Mandelbrot, J. W. Van Ness. Fractional brownian motions, fractional noises and applications. *SIAM Review*, 10:422–437, 1968. 133

[194] O. Mangasarian, H. Stone. Two-person zero-sum games and quadratic programming. *Journal of Mathematical Analysis and its Applications*, 9:348–355, 1964. 617

[195] A. S. Manne. On the job-shop scheduling problem. *Operations Research*, 9:219–223, 1960. 779

[196] B. Martos. *Nonlinear Programming Theory and Methods*. Akadémiai Kiadó, 1975. 617

[197] A. Matsumoto, F. Szidarovszky. Stability, bifurcation, and chaos in *N*-firm nonlinear Cournot games. *Discrete Dynamics in Nature and Society*, 2011:Article ID 380530, 22 pages, 2011. 618

[198] McAfee. Sniffer technologies. http://www.nai.com/us/index.asp, 2004. 134

[199] G. B. McMahon. Optimal production schedules for flow shops. *Canadian Operation Research Society Journal*, 7:141–151, 1969. 779

[200] R. McNaughton. Scheduling with deadlines loss functions. *Management Science*, 6:1–12, 1959. 778

[201] N. Megiddo. Pathways to the optimal set in linear programming. In N. Megiddo (szerk.), *Progress in Mathematical Programming: Interior Point and Related Methods*, 131–158. o. Springer-Verlag, 1989. 559

[202] B. Melamed. An overview of tes processes and modeling methodology. In L. Donatiello, A. R. Nelson (szerk.), *Models and Techniques for Performance Evaluation of Computer and Communications Systems*, Lecture Notes in Computer Science, 359–393. o. Springer-Verlag, 1993. 133

[203] Microsoft. In *HLSL*, 2010. http://msdn.microsoft.com/en-us/library/bb509561(v=VS.85).aspx. 20

[204] H. Mills. Equilibrium points of finite games. *SIAM Journal of Applied Mathematics*, 8:397–402, 1976. 617

[205] S. Mizuno, Shinji, M. Todd, Y. Ye. On adaptive-step primal-dual interior-point algorithms for linear programming. *Mathematics of Operations Research*, 18(4):964–981, 1993. 559

[206] S. Molnár, A. Vidács, A. Nilsson. Bottlenecks on the way towards characterization of network traffic: Estimation and interpretation of the Hurst parameter. http://hsnlab.ttt.bme.hu/ molnar (konferencia előadások), 1997. 134

[207] S. Molnár, F. Szidarovszky. *Konfliktushelyzetek megoldási módszerei*. SZIE, 2010. 659

[208] J. Mészáros. *Játékelmélet*. Gondolat Kiadó, 2004. 618

[209] T. Murata. Petri nets, properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989. 385

[210] J. Nash. Noncooperative games. *Annals of Mathematics*, 54:286–295, 1951. 616

[211] J. Nash. The bargaining problem. *Econometrica*, 18:155–162, 1950. 659

[212] J. Neumann, O. Morgenstern. *Theory of Games and Economical Behaviour* (2. kiadás). Princeton University Press, 1947. 617

[213] M. F. Neuts. A versatile markovian point process. *Journal of Applied Probability*, 18:764–779, 1979. 133, 134

[214] M. F. Neuts. *Structured Stochastic Matrices of M/G/1 Type and Their Applications*. Marcel Dekker, 1989. 133

[215] H. Nguyen (szerk.). *GPU Gems*. Addison-Wesley, 2008. 63

[216] M. A. Nielsen, L. Chuang (szerk.). *Quantum Computation and Quantum Information*. Cambridge University Press, 2000. 176

[217] H. Nikaido, K. Isoda. Note on noncooperative games. *Pacific Journal of Mathematics*, 5:807–815, 1955. 617

[218] NVIDIA. In *Cg homepage*, 2010. http://developer.nvidia.com/page/cg_main.html. 20, 63

[219] NVIDIA. In *CUDA Zone*, 2010. http://www.nvidia.com/object/cuda_home_new.html. 20, 63

[220] S. Oaks, H. Wong. *Java Threads*. O'Reilly, 1999. 300

[221] K. Okuguchi. *Expectation and Stability of Oligopoly Models*. Springer, 1976. 617

[222] K. Okuguchi, F. Szidarovszky. *The Theory of Oligopoly with Multi-Product Firms* (2. kiadás). Springer, 1999. 617

[223] OpenMP Website. http://www.openmp.org, 2004. 300

[224] OPNET. Opnet Modeler Documentation. www.opnet.com, 2004. 133

[225] J. D. Owens, D. Luebke, N. Govindaraju, M. J. Harris, J. Krüger, A. Lefohn, T. Purcell. A survey of general-purpose computation on graphics hardware. *Computer Graphics Forum*, 26(1):80–113, 2007. 63

[226] S. Owicki, D. Gries. An axiomatic proof technique for parallel programs I. *Acta Informatica*, 6(4):319–340, 1976. 241

[227] S. Owicki, L. Lamport. An axiomatic proof technique for parallel programs i. *ACM Transactions on Programming Languages and Systems*, 4(3):455–495, 1982. 241

[228] C. Partridge. The end of simple traffic models. *IEEE Network*, 7(5):3–3, 1993. Editor's Note. 133

[229] A. Pataricza. Petri-hálók. in: A. Pataricza (szerk). In *Formális módszerek a számítástudományban*. Typotex, 2004, 31–111. 385, 386

[230] A. K. Pati, S. L. Braunstein (szerk.). *Deutsch-Jozsa algorithm for continuous variables*. Kluwer Academic Publishers, 2003. 176

[231] V. Paxson, S. Floyd. Wide-area traffic: The failure of poisson modeling. *IEEE/ACM Transactions on Networking*, 3:226–244, 1995. 133

[232] M. Pease, R. Shostak L. Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, 1980. 242

[233] Peptool. B(PN)^2 nyelv honlapja. http://theoretica.informatik.uni-oldenburg.de/ pep/, 2005. 386

[234] G. Peterson, J. Fischer. Economical solutions for the critical section problem in distributed systems. In *Proceedings of the 9th ACM Symposium on Theory of Computing*, 91–97. o. IEEE Computer Society Press, 1977. 241, 242

[235] C. Petri. *Kommunikation mit Automaten*. PhD thesis, Technical University Darmstadt, 1962. 385

[236] G. F. Pfister. *In Search of Clusters* második kiadás. Prentice Hall, 1998. 300

[237] M. Pharr (szerk.). *GPU Gems 2*. Addison-Wesley, 2005. 63

[238] J. Piehler. Ein Beitrag zum Reihenfolgeproblem, Unternehmensforschung. *Unternehmerforschung*, 4:138–142, 1960. 779

[239] M. Pinedo. *Scheduling: Theory, Algorithms and Systems*. Pearson Education, 2001 (2. kiadás). 780

[240] M. Pinedo. *Planning and Scheduling in Manufacturing and Services.* Springer-Verlag, 2009 (második kiadás). 780

[241] Párhuzamos folyamatok. B(PN)^2 ábrák. http://plc.inf.elte.hu/parh_foly, 2005. 386

[242] A. Prékopa. *Valószinűségelmélet műszaki alkalmazásokkal.* Műszaki Könyvkiadó, 1962. 133, 718

[243] A. Prékopa. Az optimalizáláselmélet kialakulásának a történetéről. *Alkalmazott Matematikai Lapok*, 4:165–191, 1978. 558

[244] A. Prékopa. Farkas Gyula élete és munkásságának jelensége az optimalizálás elméletében. In S. Komlósi, T. Szántai (szerk.), *Új utak a magyar operációkutatásban: In memoriam Farkas Gyula.* Dialóg Campus Kiadó, 1999. 558

[245] P. Quinton. Automatic synthesis of systolic arrays from uniform recurrent equations. In *Proceedings of the 11th Annual International Symposium on Computer Architecture*, 208–214. o., 1984. 438

[246] S. K. Rao. Regular iterative algorithms and their implementations on processor arrays. Doktori értekezés, Stanford University, 1985. 438

[247] W. Reisig. *Elements of Distributed Algorithms, Modeling and Analysis with Petri Nets.* Springer-Verlag, 1998. 386

[248] J. Renegar. A polynomial-time algorithm, based on Newton's method, for linear programming. *Mathematical Programming*, 40:59–93, 1988. 559

[249] A. Rényi. *Probability Theory.* Akadémiai Kiadó/North Holland Publ. House, 1970 (Magyarul: *Valószínűségelmélet,* Tankönyvkiadó, Budapest, 1973). 133, 718

[250] J. Robinson. An iterative method of solving a game. *Annals of Mathematics*, 154:296–301, 1951. 617

[251] C. Roos, T. Terlaky, J.-P. Vial. *Theory and Algorithms for Linear Optimization: An Interior Point Approach.* John Wiley & Sons, 1997. 560

[252] S. H. Roosta. *Parallel Processing and Parallel Algorithms.* Springer-Verlag, 1999. 299

[253] J. Rosen. Existence and uniqueness of equilibrium points for concave n-person games. *Econometrica*, 33:520–534, 1965. 617

[254] M. H. Rothkopf. Scheduling independent tasks on parallel machines. *Management Science*, 12:437–447, 1966. 778

[255] R. O. Roundy, W. L. Maxwell, Y. T. Herer, S. R. Tayur, A. W. Getzler. A price-directed approach of production operations. *IIE Transactions*, 23:149–160, 1991. 779

[256] B. Roy, B. Sussmann. Les Problémes d'Ordonnuncement avec Contraintes Disjonctives. Note DS no. 9 bis, SEMA, 1964. 779

[257] T. L. Saaty. *Elements of Queueing Theory with Applications.* McGraw-Hill Book Co., 1961. 717

[258] S. Sahni. Algorithms for scheduling independent tasks. *Journal of the ACM*, 23:116–127, 1976. 778

[259] S. Sahni. Preemptive scheduling with due dates. *Operations Research*, 27:929–934, 1979. 778

[260] M. Salahi, T. Terlaky. Self-regular interior-point methods for semidefinite optimization. In *Handbook on Semidefinite, Conic and Polynomial Optimization*, 437–454. o. Springer-Verlag, 2012. 560

[261] R. Salazar, F. Szidarovszky, E. Coppola, A. Rojano. Application of game theory for a groundwater conflict in mexico. *Journal of Environmental Management*, 84(4):560–571, 2007. 618

[262] L. Schrage. Solving resource-constrained network problems by implicit enumeration. *Operations Research*, 18:263–278, 1970. 778, 779

[263] A. Segall. Distributed network protocols. *IEEE Transactions on Information Theory*, IT-29(1):23–35, 1983. 241

[264] J. Sgall. On-line scheduling. In *Lecture Notes in Computer Science, 1442. kötet*, 1998, 196–231. o. Springer-Verlag, 1998. 480

[265] H. N. Shapiro. Note on a computation method in the theory of games. *Communications on Pure and Applied Mathematics*, 11:587–593, 1958. 617

[266] G. Shedler. *Regeneration and Networks of Queues.* Springer-Verlag, 1987. 718

[267] D. Shmoys, J. Wein, D. P. Williamson. Scheduling parallel machines online. *SIAM Journal on Computing*, 24:1313–1331, 1995. 480

[268] P. W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Proceedings of 35th Annual Symposium on Foundations of Computer Science*, 124–134. o., 1994. 176

[269] D. Sima, T. Fountain, P. Kacsuk. *Advanced Computer Architectures: a Design Space Approach* 1998 (2. kiadás. Magyarul: *Korszerű számítógéparchitektúrák tervezésitér-megközelítésben*. Szak Kiadó, 1998). Addison-Wesley Publishing Company, 1998. 133, 299, 439

[270] D. R. Simon. On the power of quantum computation. In *Proceedings of the 35th Annual IEEE Symposium on the Foundations of Computer Science*, 116–123. o., 1994. It appeared also in *SIAM Journal on Computing*, number 5, volume **26**, pages 1474–1483. 176

[271] D. Sleator R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28:202–208, 1985. 481

[272] W. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quaterly*, 3:59–66, 1956. 778, 780

[273] M. Snir, S. W. Otto, S. D. W. Huss-Lederman, D. W. Walker, J. Dongarra. *MPI: The Complete Reference*. The MIT Press, 1996. 300

[274] Gy. Sonnevend. Az analytic center for polyhedrons and new classes of global algorithms for linear (smooth, convex) programming. In A. Prékopa, J. Szelezsán, B. Straziczky (szerk.), *System Modelling and Optimization: Proceedings of the 12th IFIP-Conference* Lecture Notes in Control and Information Sciences, ***84***, 866–876. o. Springer-Verlag, 1985. 559

[275] J. Stam. Stable fluids. In *Proceedings of SIGGRAPH 99*, Computer Graphics Proceedings, Annual Conference Series Santa Fe,November 22–24, 1994, 121–128. o., 1999. 52

[276] D. Stiliadis. *Traffic scheduling in packet switched networks: Analysis, Design, and Implementation*. PhD thesis, University of California Santa Cruz, 1996. 718

[277] L. Szeidl. On the estimation of moment of regenerative cycles in a general closed central-server queueing network. In *Stability problems for stochastic models*, Lecture Notes in Mathematics, **1233**, 182–189. o., 1987. 718

[278] L. Szeidl. Estimation of the moment of the regeneration period in a closed central-server queueing network. *Theory of Probability and Applications*, 33(2):309–313, 1988. 718

[279] F. Szidarovszky, C. Chiarella. Dynamic oligopolies, stability and bifurcation. *Cubo Mathemática Educational*, 3(2):267–284, 2001. 617

[280] F. Szidarovszky, L. Domoszlai. Conflict situations. In A. Iványi (szerk.), *Algorithms of Informatics, Volume 3*, 1351–1387. o. AnTonCom, 2011. 659

[281] F. Szidarovszky, M. E. Gershon. *Techniques for Multiobjective Decision Making in Systems Management*. Elsevier Press, 1986. 659

[282] F. Szidarovszky, S. Molnár. *Játékelmélet műszaki alkalmazásokkal: Többcélú programozás, klasszikus és differenciáljátékok*. Műszaki Könyvkiadó, 1986. 617

[283] F. Szidarovszky, S. Yakowitz. *Principles and Procedures of Numerical Analysis*. Plenum Press, 1998. 617

[284] L. Szirmay-Kalos. *Számítógépes grafika*. Addison-Wesley, 2005. 62

[285] L. Szirmay-Kalos, Gy. Antal, F. Csonka. *Háromdimenziós grafika, animáció és játékfejlesztés*. Computer Books, 2003. 22

[286] L. Szirmay-Kalos, L. Szécsi, M., Sbert. *GPU-Based Techniques for Global Illumination Effects*. Morgan and Claypool Publishers, 2008. 63

[287] L. Szirmay-Kalos, B. Tóth, M. Magdics, D. Légrády, A. Penzov. Gamma Photon Transport on the GPU for PET. *Lecture Notes on Computer Science*, 5910:433–440, 2010. 41

[288] J. Sztrik. Bevezetés a sorbanállási elméletbe és alkalmazásaiba (introduction to queueing theory and its applications). http://irh.inf.unideb.hu/user/jsztrik/, 2004. 133, 717, 718

[289] W. Szwarc. Optimal elimination methods in the $m \times n$ sequencing problem. *Operations Research*, 21:1250–1259, 1973. 779

[290] W. Szwarc. Dominance conditions for the three-machine flow-shop problem. *Operations Research*, 26:203–206, 1978. 779

[291] W. Szwarc. Elimination methods in the $m \times n$ sequencing problem. *CWI Quarterly*, 18:295–305, 1971. 779

[292] A. S. Tanenbaum. *Modern Operating Systems.* Prentice Hall, 2001. 300

[293] A. S. Tanenbaum. *Computer Networks* (4. kiadás). Prentice Hall, 2004 (magyarul: *Számítógép-hálózatok.* Panem, 2004). 133, 718

[294] A. S. Tanenbaum, M. van Steen. *Distributed Systems. Principles and Paradigms* (magyarul: *Elosztott rendszerek.* Panem, 2003). Prentice Hall, 2002. 300

[295] A. S. Tanenbaum, M. van Steen. *Distributed Systems. Principles and Paradigms (második kiadás).* Prentice Hall, 2006 (Magyarul: *Elosztott rendszerek.* Panem, 2003). 242

[296] M. Taqqu, W. Teverovsky, W. Willinger. Estimators for long-range dependence: an empirical study. *Fractals*, 3(4):785–788, 1995. 133

[297] A. Tarski. A lattice-theoretical fixpoint theorem and its application. *Pacific Journal of Mathematics*, 5:285–308, 1955. 617

[298] N. Tatarchuk, P. Sander, J. L. Mitchell. Early-z culling for efficient GPU-based fluid simulation. In W. Engel (szerk.), *ShaderX 5: Advanced Rendering Techniques*, 553–564. o. Charles River Media, 2006. 57

[299] A. D. Taylor, A. Pacelli. *Mathematics and Politics.* Springer-Verlag, 2008. 659

[300] J. Teich, L. Thiele. Control generation in the design of processor arrays. *International Journal of VLSI and Signal Processing*, 3(2):77–92, 1991. 438

[301] G. Tel. *Introduction to Distributed Algorithms.* Cambridge University Press, 2000 (második kiadás). 242

[302] T. Terlaky (ed.). *Interior Point Methods of Mathematical Programming (Applied Optimization).* Springer-Verlag, Berlin, 2011. First edition: 1996. 560

[303] W. Thomson. Cooperative models of bargaining. in r. j. aumann and s. hart (szerk.). In *Handbook of Game Theory.* Elsevier, 1994. 659

[304] J. Tomkó. *A Markov folyamatok elmélete és néhány operációkutatási vonatkozása.* Bolyai János Matematikai Társulat, 1968. 717

[305] Top 500 Supercomputer Sites. http://www.top500.org, 2004. 299

[306] Trusoft International Inc.. *Benoit 1.1.* Trusoft International Inc., 2004. 134

[307] L. G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, 1990. 300

[308] A. van Vliet. Lower and upper bounds for on-line bin packing and scheduling heuristics. PhD értekezés, Erasmus University, Rotterdam, 1995. 480

[309] A. van Vliet. An improved lower bound for on-line bin packing algorithms. *Information Processing Letters*, 43:277–284, 1992. 480

[310] B. Vizvári. *Egészértékű programozás.* ELTE TTK, 1990. 779

[311] Web Dictionary of Cybernetics, Systems. http://pcp.wub.ac.be/ASC.html, 2004. 133

[312] W. Willinger, V. Paxson. Discussion of „heavy tail modeling and teletraffic data" by S. R. Resnick. *The Annals of Statistics*, 25(5):1805–1869, 1997. 134

[313] W. Willinger, M. Taqqu, R. Sherman, D. Wilson. Self-similarity through high-variability: statistical analysis of Ethernet LAN traffic at the source level. *IEEE/ACM Transactions on Networking*, 5:71–86, 1997. 134

[314] W. Willinger, D. Wilson, W. Leland, M. Taqqu. On traffic measurements that defy traffic models (and vice versa): self-similar traffic modeling for high-speed networks. *Connections*, 8(11):14–24, 1994. 133

[315] M. J. Wolfe. *High Performance Compilers for Parallel Computing.* Addison Wesley Longman, 1996. 300

[316] A. C. C. Yao. New algorithms for bin packing. *Journal of the ACM*, 27:207–227, 1980. 481

[317] Y. Ye. *Interior Point Algorithms: Theory and Analysis.* John Wiley & Sons, 1997. 560

[318] Y. Ye, M. J. Todd, S. Mizuno. An $O(\sqrt{n}L)$-iteration homogeneous and self-dual linear programming algorithm. *Mathematics of Operations Research*, 19:53–67, 1994. 560

[319] N. Young. On-line file caching. *Algorithmica*, 33:371–383, 2002. 480

[320] E. Zehendner. *Entwurf systolischer Systeme: Abbildung regulärer Algorithmen auf synchrone Prozessorarrays*. B. G. Teubner Verlagsgesellschaft, 1996. 438

[321] J. Zhao, M. N. Szilagyi, F. Szidarovszky. An n-person battle of sexes game. *Physica A: Statistical Mechanics and its Applications*, 387(14):3669–3677, 2008. 618

[322] F. Zhe, F. Qiu, A. Kaufman, S. Yoakum-Stover. GPU cluster for high performance computing. In *Proceedings of the 2004 ACM/IEEE Conference on Supercomputing*, SC'04, 47–59. o. IEEE Computer Society, 2004. 63

[323] S. I. Zuhovitsky, R. A. Polyak, M. E. Primak. Concave n-person games (numerical methods). *Ékonomika i Matematicheskie Methody*, 7:888–900, 1971 (oroszul). 617

[324] A. Vestjens. On-line machine scheduling, 1997. PhD értekezés, Eindhoven University of Technology. 480, 481

Ezt az irodalomjegyzéket HBibTEXsegítségével állítottuk össze. A dokumentumokat elsősorban a szerzők neve (első szerző, második szerző stb.), másodsorban a megjelenés éve, harmadsorban pedig a dokumentumok címe alapján rendeztük.

Az aláhúzás azt jelzi, hogy az aláhúzott szövegrészre kattintva a megfelelő honlapra juthatunk. A hivatkozás végén lévő kék számok pedig azt mutatják, mely oldalakon van hivatkozás az adott dokumentumra.

Tárgymutató

Ez a tárgymutató a következő szempontok szerint készült.

Először a matematikai jelöléseket, azután a tárgyszavakat soroljuk fel (a és á, e és é, i és í, o, ó, ö és ő, u, ú, ü és ű között nem teszünk különbséget).

A számokat és görög betűket tartalmazó tárgyszavakat kiejtésük szerint rendeztük: például a „2-3-4 fá"-t „kettőháromnégy fa"-ként, a „d-kupac"-ot „dkupac"-ként, az „ϵ-sűrű gráf"-ot „epszilonsűrű gráf"-ként.

Az algoritmusok (pszeudokódok) nevét a betűmérettel is megkülönböztettük a többi névtől (függvények, modellek, problémák, nyelvek stb.): például Binomiális-kupacban-keres, illetve VAGY-függvény, ÚT, SAT. A kezdőbetűkből álló szavak betűi azonban mindenütt azonos méretűek: például Alap-FFT, LKR-Nyomtat, illetve CRCW, FIFO, MFF, MFH.

Ha egy tárgyszó nem a fő szövegre utal, akkor az oldalszámot kiegészítés követi: *gy* gyakorlatot, *fe* feladatot, *áb* ábrát, *lá* lábjegyzetet jelent.

A különböző típusú objektumokat lehetőség szerint tipográfiailag is megkülönböztettük. A matematikai jelöléseket és a programokban használt változók nevét dőlt betűk emelik ki, mint például $\Omega(n \lg n)$ vagy $rang[kulcs]$. Az algoritmusok neveit kiskapitális betűkkel írtuk, mint például Gyorsrendez. Az algoritmusok kódjában a programozási alapszavakat félkövéren szedtük, mint például **if, then, for**.

Az algoritmusok nevében kiskötőjelet használtunk, mint például Utazó-ügynök. Az egyes fogalmak meghatározásának és az algoritmusok pszeudokódjának helyére a tárgymutató dőlt oldalszámmal utal.

Elsősorban az algoritmusokat tárgyaló tankönyvek matematikai jelöléseit alkalmaztuk. Az oldalszámok felsorolásánál nem törekedtünk teljességre.