Írta:
**DÓSA GYÖRGY**
**IMREH CSANÁD**

# ONLINE ALGORITMUSOK

**Egyetemi tananyag**


**2011**

LEKTORÁLTA: Dr. Iványi Antal, Eötvös Loránd Tudományegyetem Informatikai Kar Komputeralgebra Tanszék



TÁMOGATÁS:
Készült a TÁMOP-4.1.2-08/1/A-2009-0008 számú, „Tananyagfejlesztés mérnök informatikus, programtervező informatikus és gazdaságinformatikus képzésekhez” című projekt keretében.


ISBN 978 963 279 508 9

KÉSZÜLT: a Typotex Kiadó gondozásában
FELELŐS VEZETŐ: Votisky Zsuzsa
AZ ELEKTRONIKUS KIADÁST ELŐKÉSZÍTETTE: Gerner József

KULCSSZAVAK:
algoritmusok, online-problémák, legrosszabb eset korlátok, versenyképességi elemzés, optimalizálási problémák, erőforrás allokáció

ÖSSZEFOGLALÁS:

A gyakorlati problémákban gyakran fordulnak elő olyan optimalizálási feladatok, ahol a bemenetet csak részenként ismerjük meg, és a döntéseinket a már megkapott információ alapján, a további adatok ismerete nélkül kell meghoznunk. Ilyen feladatok esetén online problémáról beszélünk. Az algoritmusokat egy legrosszabb eset korlát elemzéssel szokás vizsgálni, amelyet versenyképességi elemzésnek neveznek. Az online algoritmusok elméletének igen sok alkalmazása van a számítástudomány, a közgazdaságtan és az operációkutatás különböző területein. A jegyzetnek nem a témakör részletes áttekintése a célja, hanem a területen használt alapvető algoritmustervezési és elemzési technikák bemutatása az online algoritmusok elméletének különböző részterületein (lapozás, lista karbantartás, k-szerver feladat, ütemezés, ládapakolás, számítógépes hálózatok online problémái, online tanulás) keresztül.

# Tartalomjegyzék

# Előszó

Jelen jegyzetet a Szegedi Tudományegyetem programtervező informatikus MSc képzés Online algoritmusok című törzstárgyának tematikája alapján készítettük. Ennek ellenére a jegyzet, illetve az egyes fejezetei jól használhatóak egyéb egyetemek tetszőleges algoritmusokkal foglalkozó kurzusain. A jegyzetünknek nem a témakör részletes áttekintése a célja, hanem a területen használt alapvető algoritmustervezési és elemzési technikák bemutatása az online algoritmusok elméletének különböző részterületein keresztül.

A jegyzet első fejezetében a legfontosabb fogalmakat tisztázzuk, egy bevezető egyszerű példa, a síbérlés feladatának bemutatásán keresztül. A második fejezet a lapozási probléma alapvető eredményeit mutatja be. A harmadik fejezetben a dinamikus adatszerkezetek karbantartásának területéről mutatjuk be a lista karbantartás problémáját. A negyedik fejezet a véletlenített online algoritmusokra vonatkozó általános elméleti alapokat mutatja be, majd ezek felhasználására adunk példákat az ötödik fejezetben az első három fejezetben ismertetett problémák alapján. Az hatodik fejezetben a legismertebb online feladat, a k-szerver probléma alapvető eredményeit tekintjük át. A hetedik fejezetben az online ütemezés témakörét tárgyaljuk, bemutatjuk az immár klasszikusnak számító online ütemezési modelleket, és néhány új speciálisabb területről is áttekintést adunk. A nyolcadik fejezet témája a ládapakolás problémája és a sávpakolás, ami a ládapakolás egyik többdimenziós általánosítása. A kilencedik fejezetben három, a számítógépes hálózatokhoz kapcsolódó online problémát ismertetünk. A tizedik fejezet a gépi tanulás területének az online algoritmusokhoz kapcsolódó eredményeiből ismertet néhányat. Végül az utolsó, tizenegyedik fejezetben a jegyzetben használt versenyképességi elemzés lehetséges kiterjesztéseit, módosításait mutatjuk be.

Ezúton szeretnénk köszönetet mondani Iványi Antalnak, az ELTE egyetemi tanárjának a kézirat alapos lektorálásáért és hasznos tanácsaiért.

# 1. fejezet

# Alapfogalmak, bevezető példák

## 1.1. Bevezetés

A gyakorlati problémákban gyakran fordulnak elő olyan optimalizálási feladatok, ahol a bemenetet (vagyis a feladatot definiáló számadatot) csak részenként ismerjük meg, és a döntéseinket a már megkapott információ alapján, a további adatok ismerete nélkül kell meghoznunk.

Ilyen feladatok esetén *online problémáról* beszélünk. Az online algoritmusok elméletének igen sok alkalmazása van a számítástudomány, a közgazdaságtan és az operációkutatás különböző területein.

Az online algoritmusok elméletének területéről az első eredmények az 1970-es évekből származnak, majd a 90-es évek elejétől kezdve egyre több kutató kezdett el az online algoritmusok területéhez kapcsolódó problémákkal foglalkozni. Számos részterület alakult ki és napjainkban is a legfontosabb, algoritmusokkal foglalkozó konferenciákon rendszeresen ismertetnek új eredményeket ezen témakörből. Ennek a jegyzetnek nem célja a témakör részletes áttekintése, terjedelmi okokból ez nem is lenne lehetséges ezen keretek között. További eredmények találhatóak a [10, 27, 30] művekben. Célunk néhány részterület részletesebb ismertetésén keresztül a legfontosabb algoritmustervezési technikák és bizonyítási módszerek bemutatása.

## 1.2. Fogalmak, definíciók

Mivel egy online 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 megkapható optimális megoldást szolgáltassa. Azon algoritmusokat, amelyek ismerik a teljes bemenetet, *offline algoritmusoknak* nevezzük.

Az online algoritmusok hatékonyságának vizsgálatára két alapvető módszert használnak. Az egyik lehetőség az *átlagos eset elemzése*. Ebben az esetben fel kell tételeznünk valamilyen valószínűségi eloszlást a lehetséges bemenetek terén, és a célfüggvénynek az erre az eloszlásra vonatkozó várható értékét vizsgáljuk.

Ezen megközelítés hátránya, hogy általában nincs információnk arról, hogy a lehetséges bemenetek milyen valószínűségi eloszlást követnek. E jegyzetben mi az átlagos eset elem-

zésének témakörével nem foglalkozunk, hanem az elterjedtebb versenyképességi elemzés módszerét használjuk.

A másik megközelítés egy legrosszabb-eset elemzés, amelyet *versenyképességi elemzésnek* nevezünk. Ebben az esetben az online algoritmus által kapott megoldás célfüggvényértékét hasonlítjuk össze az optimális offline célfüggvényértékkel.

Egy online minimalizálási probléma esetén egy online algoritmust *C-versenyképesnek* nevezünk, ha tetszőleges bemenetre teljesül, hogy az algoritmus által kapott megoldás költsége nem nagyobb, mint az optimális offline költség $C$-szerese. Egy *algoritmus versenyképességi hányadosa* a legkisebb olyan $C$ szám, amelyre az algoritmus $C$-versenyképes.

A továbbiakban egy tetszőleges ALG online algoritmusra az $I$ bemeneten felvett célfüggvényértéket $ALG(I)$-vel jelöljük. Az $I$ bemeneten felvett optimális offline célfüggvényértéket $OPT(I)$-vel jelöljük. Ezt a jelölésrendszert használva a versenyképességet minimalizálási problémákra a következőképpen definiálhatjuk.

Az ALG algoritmus $C$-versenyképes, ha $ALG(I) \leq C \cdot OPT(I)$ teljesül minden $I$ bemenet esetén.

Szokás használni a versenyképesség egy további változatát is. Egy minimalizálási probléma esetén az ALG algoritmus *enyhén C-versenyképes*, ha van olyan $B$ konstans, hogy $ALG(I) \leq C \cdot OPT(I) + B$ teljesül minden $I$ bemenet esetén. Egy *algoritmus enyhe versenyképességi hányadosa* a legkisebb olyan $C$ szám, amelyre az algoritmus enyhén $C$-versenyképes.

Természetesen igaz, hogy ha egy algoritmus erősen versenyképes valamely $C$ konstanssal, akkor ezzel egyidejűleg ugyanezzel a konstanssal gyengén is versenyképes.

A fentiekben a minimalizálási problémákra definiáltuk a versenyképességi analízis fogalmait. A definíciók hasonlóan értelmezhetőek maximalizálási problémák esetén is. Ekkor az ALG algoritmus $C$-versenyképes, ha $ALG(I) \geq C \cdot OPT(I)$ teljesül minden $I$ bemenet esetén, illetve enyhén $C$-versenyképes, ha valamely $B$ konstans mellett $ALG(I) \geq C \cdot OPT(I) + B$ teljesül minden $I$ bemenetre. Tehát amíg mimimalizálandó célfüggvény esetén az előbbi konstansra $C \geq 1$, addig maximalizálandó célfüggvény esetén $C \leq 1$.

## 1.3. Síbérlési feladat

A *síbérlési feladat* esetén adott egy pár síléc (röviden sí), amit vagy 1 egységért bérelhetünk vagy megvásárolhatunk $B$ egységért (ahol $B > 1$ egész szám). A feladat annak eldöntése, hogy mikor vásároljuk meg a sílécet. A probléma online, ami azt jelenti, hogy amikor egy napon el kell dönteni, béreljük vagy vásároljuk a sít, akkor nincs információnk arról, hogy a következő napokon is síelünk -e.

Vagyis elmegyünk a téli szünetben síelni, és minden nap síelünk, ha az idő megengedi, legalábbis így tervezzük. De hogy az idő alkalmas lesz-e a következő napon, illetve napokon a síelésre, azt nem tudjuk előre.

A feladatnak nagyon egyszerű a struktúrája, a bemenet egyetlen pozitív egész szám, ami megadja, hogy hány napig síelünk. Ennek megfelelően egy online algoritmus csak annyit tehet, hogy valahány napon keresztül bérli a sílécet, és ha addig nem hagytuk abba a síelést, akkor vásárol egyet. Azt az algoritmust, amely $V-1$ napig béreli a sít, aztán a $V$-edik napon megvásárolja V algoritmusnak nevezzük.

**1. tétel.** *A* V *algoritmus* $V = B$ *esetén* $(2-1/B)$*-versenyképes.*

*Bizonyítás:* A feladat $I$ bemenete azon napok száma, ahány napig síelünk. Ekkor az alábbi két esetet különböztethetjük meg $I$ értékétől függően:

- Ha $I < B$, akkor mind az online algoritmusnak, mind pedig az optimális offline algoritmusnak a költsége I, így $B(I)/OPT(I) = 1$.
- Ha $I \geq B$, akkor $OPT(I) = B$, továbbá $B(I) = B-1+B$, így $B(I)/OPT(I) = (2B-1)/B$.

Mivel mindkét esetben teljesül, hogy a vizsgált hányados legfeljebb $(2B-1)/B$, ezért az állítást igazoltuk. □

Jogosan merül fel a kérdés megadható -e jobb, kisebb versenyképességi hányadossal rendelkező algoritmus. Az alábbi tétel mutatja, hogy nincs ilyen.

**2. tétel.** *Nincs olyan online síbérlési algoritmus, amelynek kisebb a versenyképességi hányadosa, mint* $2-1/B$*.*

*Bizonyítás:* Vegyünk egy tetszőleges algoritmust. Mint a fentiekben írtuk, ez valahány napon keresztül bérli a sílécet, és ha addig nem hagytuk abba a síelést, akkor vásárol egyet. Legyen $V$ az az időpont, amikor az algoritmus megvásárolja a sít. A feladat bemenete legyen $V$ napnyi síelés. Ekkor az algoritmus költsége $V-1+B$. Különböztessük meg az alábbi két esetet $V$ értékétől függően.

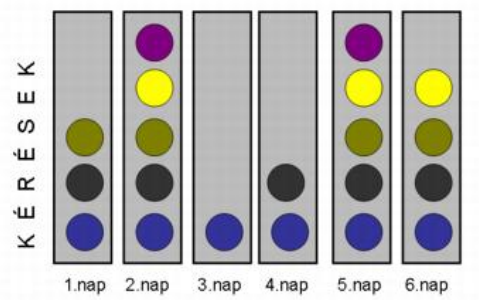
- Ha $V < B$, akkor az optimális költség $V$. Következésképpen, ekkor a vizsgált hányados $(V-1+B)/V = 1+(B-1)/V \geq (2B-1)/B$.
- Ha $V \geq B$, akkor az optimális költség $B$. Következésképpen, ekkor a vizsgált hányados $(V-1+B)/B \geq (2B-1)/B$. □

## 1.4. A síbérlési feladat általánosítása

Alább megadjuk a feladatnak egy lehetséges általánosítását, amelyre könnyen adható ugyanakkora versenyképességi hányadossal rendelkező algoritmus, mint az egyszerű esetben. Az általánosítás abban áll, hogy nem csak egy, hanem legfeljebb $K$ számú sít kölcsönözhetünk egyszerre. A feladat a következő: Tegyük fel, hogy van egy panziónk, ahol a szállóvendégek részére biztosítjuk a szükséges felszerelést. Egyszerre legfeljebb $K$ vendég szállhat meg a panzióban, tehát egyszerre legfeljebb $K$ számú síre van szükség. Tegyük fel, hogy kezdetben egyetlen síléc sincs a panzióban, hanem a panzió is kölcsönzi a síléceket, egy pár síléc kölcsönzése 1 egységbe kerül, a megvásárlása pedig $B$ pénzegységbe. A panzió személyzete naponta kölcsönzi a kívánt mennyiségű sílécet, illetve bármelyik napon vásárolhat is akármennyi sít. Nyilván nem érdemes $K$-nál több sílécet vásárolni, a kérdés abban áll, hogy mikor vásárolja meg a panzió a $K$ számú sílécet. (Ha az algoritmus soha nem vásárol meg $K$

darabot, csak kevesebbet, akkor elég hosszú idény esetén, vagyis ha a sílécek iránti igények sorozata elég hosszú, és mindig elég sok sílécet igényelnek, akkor egy ilyen algoritmus nem lehet konstans versenyképes.)

A következő algoritmus viszont $2-1/B$-versenyképes, ugyanúgy, mint $K=1$ esetén: Minden $1 \leq k \leq K$ esetén vásároljuk meg a $k$-adik sílécet azon a napon, amely napon $B$-szer érkezett már legalább $k$ darab sílécre igény. Az 1.1 ábra az algoritmus futását szemlélteti.




1.1. ábra. Az általánosított síbérlési algoritmus

Jelen esetben egy hat napos időszakot szemléltet az ábra. Minden nap igényelnek valahány sílécet. (Ha néhány napon keresztül nem érkezik igény, akkor ezeket a napokat törölhetjük a bemenetből, a versenyképességi vizsgálat szempontjából ezeknek nincs jelentősége, mert sem az optimális, sem az általunk futtatott online algoritmus költségére az ilyen napok nincsenek hatással.) Legyen példánkban $K=5$. Az első napon 3, a második napon 5, a harmadik napon egy síre érkezik igény, és így tovább. Legyen $B=3$. Ekkor az algoritmus az első sílécet a harmadik napon veszi meg, a másodikat a negyedik napon, mert a negyedik nap az a nap, amikor ebből három napon legalább kettő volt az igény. A harmadik sít az ötödik napon, mert ez az a nap, amikor már három nap volt az igény legalább három. És így tovább, a negyedik sít a hatodik napon veszi meg az algoritmus.

A versenyképesség vizsgálata a következőképpen mehet: Egyrészt, mivel ez az algoritmus általánosítása az előzőnek, az általános algoritmus versenyképességi hányadosa nem lehet jobb, mint a speciális $K=1$ eset algoritmusának a versenyképességi hányadosa, ami $2-1/B$. Másrészt, mindegyik sí megvásárlása esetén, legfeljebb $B-1$ egységnyi pénzt fizetünk ki fölöslegesen, vagyis mindegyik megvásárolt síre (soronként) ugyanazt az elemzést elvégezve, megkapjuk a kívánt legrosszabb eset hányadost.

# 2. fejezet

# Lapozási (memóriakezelési) probléma

A lapozási problémában a számítógépek gyorsmemóriájának a kezelését modellezik. Adott lapok egy univerzuma, és ebből származó lapok egy sorozata a bemenet. Az algoritmusnak egy $k$ lap kapacitású gyorsmemóriát kell kezelnie. Ha az aktuálisan igényelt lap nincs a memóriában, akkor a lapot az algoritmusnak be kell tennie és ehhez, amennyiben már nincs hely, valamely lapot ki kell raknia. Ezt az eseményt, amikor egy igényelt lap nincs a memóriában, hibának hívjuk, és a cél a hibák számának minimalizálása. A probléma online, ami azt jelenti, hogy az algoritmusnak a lap elhelyezéséhez szükséges döntést (valamely lap kivétele a memóriából) a további igények ismerete nélkül kell meghozni. Az online problémát a [48] cikkben definiálták. A következő állítás azt mutatja, hogy nincs kicsi versenyképességi hányadossal rendelkező online algoritmus a feladat megoldására.

**3. tétel.** [48] *Nincs olyan online algoritmus a lapozási problémára, amelynek versenyképességi hányadosa kisebb, mint k.*

*Bizonyítás:* Vegyünk $k+1$ lapot és legyen A egy tetszőleges online algoritmus. Legyen $L_n$ az az $n$ elemből álló bemenet, amelyben az új elem mindig az a lap, amely lap éppen hiányzik A memóriájából. Legyen továbbá $n$ osztható $k$-val. Ekkor A minden lapnál hibázik, így $A(L_n) = n$. Legyen LFD (longest forward distance) az az offline algoritmus, amely mindig azt a lapot rakja ki a memóriából, amelyre a következő igény a legkésőbb érkezik. Fontosnak tarjuk megjegyezni, hogy valójában LFD az optimális offline megoldást adja, de ezt nem igazoljuk, mivel a bizonyítás során az algoritmus optimalitására nincs szükség. Vegyük észre, hogy amennyiben LFD hibázik, akkor a következő $k-1$ kérés során nem hibázhat újra. Ez a tulajdonság azért teljesül, mert az algoritmus következő hibája akkor következik be, amikor azt a lapot igényeljük, amelyet kitett a memóriából. És az LFD szabály miatt ezt a kérést meg kell hogy előzze a másik $k-1$ lapra vonatkozó kérés. Tehát azt kaptuk, hogy $LFD(L_n) \leq n/k$. Másrészt $OPT(L_n) \leq LFD(L_n)$, tehát $A(L_n)/OPT(L_n) \geq n/(n/k)$, amivel a tétel állítását igazoltuk. (Egy additív kostans sem tehet lehetővé $k$-nál kisebb versenyképességi hányadost, mert tetszőlegesen hosszú sorozatot vehetünk.) □

Másrészt számos $k$-versenyképes determinisztikus algoritmus van, itt $k$-versenyképes algoritmusok egy osztályát, a bélyegző algoritmusokat ismertetjük.

**Bélyegző (továbbiakban B) algoritmus**

Az algoritmus egyes, már kért lapok megjelölésére bélyegeket használ a memóriában. Kezdetben egyetlen lap sincs megjelölve. Egy kérés érkezésekor az algoritmus a következő lépéseket hajtja végre.

1. Ha a kért lap a memóriában van, akkor amennyiben ez a lap még jelöletlen, akkor megjelöljük.

2. Ha a kért lap nincs a memóriában, és nincs már jelöletlen lap a memóriá- ban, akkor az összes jelölést töröljük.

3. Ezt követően veszünk egy jelöletlen lapot a memóriából (az előző lépés miatt van ilyen) a kért lapot ennek a lapnak a helyére berakjuk, majd megjelöljük.

Az algoritmus viselkedése nagymértékben függ attól, hogy milyen szabály alapján választjuk ki a törlendő lapot, de a következő tétel mutatja, hogy a versenyképességi hányadosa nem függ ettől.

**4. tétel.** [48] *A* B *algoritmus enyhe versenyképességi hányadosa k.*

*Bizonyítás:* A tétel bizonyításához elegendő belátni, hogy az algoritmus $k$-versenyképes, az általános alsó korlát alapján adódik, hogy nem lehet kisebb a versenyképességi hányadosa. Vegyünk egy tetszőleges bemenetet, jelölje $I$. Bontsuk fel ezt a bemenetet fázisokra a következőképpen. Az első fázis kezdődjön az első elemnél. Ezt követően minden egyes fázis a megelőző fázis utolsó eleme után jövő elemnél kezdődik, és a leghosszabb olyan sorozatát tartalmazza a kéréseknek, amelyek legfeljebb $k$ különböző lapot igényelnek. (Tehát a következő fázis akkor kezdődik, amikor a $k+1$-edik különböző lap megjelenik.) Érdemes megjegyezni, hogy a fázis a bélyegek kiosztásával is definiálható, akkor kezdődik új fázis, amikor a B algoritmus törli a bélyegeket a memóriában.

Az algoritmus definíciójából következik, hogy $B$ legfeljebb $k$ hibát vét egy fázis alatt, (ha egy lapon hibázik, azon többet már nem fog a fázisban, hiszen megjelölte). Most vizsgáljuk az optimális offline algoritmust. Egy tetszőleges fázis második kérésénél az offline memóriában benne van a fázis első eleme. Ezt követően a következő fázis első eleméig $k$ további elem jelenik meg, így az offine algoritmusnak legalább egy hibát kell vétenie, az adott fázis második elemétől, a következő fázis első eleméig tartó kéréssorozaton. Következésképpen minden fázishoz (kivéve esetleg az utolsót) hozzárendeltük az offline algoritmus egy-egy hibáját. Jelölje $r$ a fázisok számát. Ekkor $B(I) \leq rk+k-1$ és $OPT(I) \geq r$, következésképp $B(I) \leq k \cdot OPT(I)+k-1$, amivel a tétel állítását igazoltuk. □

A gyakorlatban a feladat megoldására az LRU (least recently used) algoritmust használják, amelyet akkor kapunk, ha minden esetben azt a lapot töröljük a memóriából, amelyet a legrégebben használtuk. Egyszerűen látható, hogy az algoritmus a bélyegző algoritmusok családjába tartozik, így a fenti tétel alapján adódik, hogy az algoritmus $k$-versenyképes.

A feladat megoldására egy másik logikusan adódó algoritmus a FIFO eljárás, amely mindig azt a lapot rakja ki a memóriából, amely a legrégebben került be. Ez az algoritmus már nem tartozik a bélyegző algoritmusok családjába, de a fenti bizonyításhoz hasonlóan igazolható, hogy $k$-versenyképes.

# 3. fejezet

# Lista hozzáférési probléma

A dinamikusan változó adatszerkezetek karbantartása tipikus online probléma, hiszen nem tudhatjuk, hogy a jövőben milyen műveleteket kell végrehajtanunk az adott adatszerkezeten. Számos dolgozat foglalkozik ilyen kérdésekkel, az alapvető eredményeket foglalja össze a [3] dolgozat és a [10] könyv. Mi itt csak a legegyszerűbb dinamikus adatszerkezettel, a láncolt listával foglalkozunk, és a [10] könyv alapján mutatjuk be a legalapvetőbb eredményeket.

A lista hozzáférési problémában adott egy láncolt lista, amelyen a következő műveleteket tudjuk végrehajtani:

- Keres($x$)
- Töröl($x$)
- Beszúr($x$)

Mi a továbbiakban csak a statikus modellt vizsgáljuk, ahol a lista nem változik, hanem adottak a lista elemei és csak Keres($x$) műveleteket hajtunk végre. Amennyiben a lista $i$-edik elemét keressük, akkor a keresés költsége $i$. Tehát az algoritmus bemenete elemek egy kezdeti $x_1,\ldots,x_n$ listája, és kérések egy $\sigma = \sigma_1,\ldots,\sigma_m$ sorozata, ahol $\sigma_i \in \{x_1,\ldots,x_n\}$. A $i$-edik kérés hatására a KERES($\sigma_i$) műveletet kell végrehajtani. Az algoritmus a keresések során megváltoztathatja az elemek sorrendjét és a célja a teljes költség minimalizálása.

A lista karbantartásához az alábbi karbantartási műveleteket engedjük meg:

- a Keres($x$) művelet során az $x$ elemet ingyen tetszőleges helyre előre mozgathatjuk,
- fizetett cserék: bármely két egymás melletti elemet felcserélhetünk 1 költséggel.

A fizetett cserék segíthetnek az optimális költség csökkentésében, amint ezt az alábbi példa is mutatja.

**Példa:** Legyen a lista $x_1,x_2,x_3$, a kéréssorozat pedig $x_3,x_2,x_3,x_2$. Ha egy algoritmus egyáltalán nem mozgatja a lista elemeit, akkor a költsége $3+2+3+2=10$. Esetszétválasztással igazolható, hogy ha egy algoritmus mozgatja a kért elemeket, de nem használ fizetett cseréket, akkor a költsége legalább 9. (Az első kérés költsége 3, ezt követően megvizsgálva, hogy hova rakja az algoritmus az $x_3$ elemet, igazolható az állítás.) Ezzel szemben, ha először két

fizetett cserével az utolsó helyre visszük $x_1$-et, akkor a cserék költsége 2 és utána már nem mozdítva több elemet a kérések további költsége 6, azaz az összes költség csak 8.

A probléma megoldására több online algoritmust is kidolgoztak, itt elsőként az MTF (Move to Front) algoritmust mutatjuk be.

**MTF algoritmus:** Az algoritmus a kért elemet a lista elejére mozdítja.

**5. tétel.** [48] *Az* MTF *algoritmus 2-versenyképes.*

*Bizonyítás:* Vegyünk egy tetszőleges $I$ bemenetet, továbbá egy optimális offline algoritmust, amit OPT-tal jelölünk. Az állítást a potenciálfüggvény technika segítségével igazoljuk, amelynek lényege az, hogy az egyes algoritmusok által aktuálisan fenntartott sorrendek közötti különbséghez egy potenciálértéket rendelünk, majd az online algoritmus és az optimális algoritmus lépésenkénti költségeinek összehasonlításakor a potenciál változását is figyelembe vesszük.

Legyen a $\Phi(i)$ potenciálfüggvény az inverziók száma az $i$-edik kérés után az MTF által karbantartott listában az optimálishoz képest, azaz azon $x,y$ párok száma, amelyekre $x$ megelőzi $y$-t OPT listájában de nem előzi meg MTF listájában.

Vizsgáljuk meg a $k$-adik kérés, Keres($\sigma_k$) művelete során fellépő költségeket és potenciálváltozást. Legyen $p$ azon elemek száma, amelyek mind MTF mind OPT listájában megelőzik $\sigma_k$-t, $q$ azon elemek száma, amelyek csak MTF listájában előzik meg $\sigma_k$-t. Ekkor MTF költsége $p+q+1$, az optimális költség legalább $p+1$. Azon $q$ darab elem, amely csak az MTF listájában előzte meg $\sigma_k$-t, a kérés kiszolgálása előtt inverziót alkotott. Ezen inverziók $\sigma_k$ előremozdításával megszűntek. Másrészt az előremozdítás generált $p$ új inverziót, azon elemekkel, amelyek mindkét listában $\sigma_k$ előtt voltak. Tehát a $\Phi$ függvény értéke $(-q+p)$-vel változik. Következésképpen

$$MTF(\sigma_k)+\Phi(k)-\Phi(k-1)=p+q+1-q+p=2p+1<2OPT(\sigma_k).$$

Ha OPT fizetett cserét használ, akkor a költsége 1-gyel nő, és a potenciálfüggvény értéke is legfeljebb 1-gyel növekszik. Továbbá, ha OPT előre mozgatja a kért elemet a kiszolgálás során, akkor az inverziók száma nem növekedhet, mivel MTF a kért elemet az első helyre mozdítja. Következésképpen a fenti egyenlőtlenség igaz marad OPT lépése során is.

Ha a fenti egyenlőtlenséget vesszük minden $i$-re, és az egyenlőtlenségeket összeadjuk, akkor a baloldalon a $\Phi(k)$ értékek rendre 1 és $-1$ együtthatóval is rendelkeznek, így azt kaptuk, hogy $MTF(\sigma)+\Phi(n)-\Phi(0)\leq 2OPT(\sigma)$, amivel a tétel állítását igazoltuk. □

A feladat megoldására más logikusan adódó algoritmusokat is megvizsgáltak. Ezekből az alábbiakban bemutatunk kettőt, amelyek egyike sem konstans versenyképes.

**FC (frequency count) algoritmus:** Az elemeket mindig a gyakoriságuk sorrendjében tartjuk sorban. (Ezt megtehetjük fizetett cserék nélkül, mivel mindig csak a keresett elem gyakorisága növekszik, így esetleg azt kell előre mozgatni.)

**1. lemma.** FC *nem konstans versenyképes.*

*Bizonyítás:* Legyen a kezdeti lista $x_1,\ldots,x_n$, a kéréssorozat pedig legyen $n$-szer $x_1$, $n-1$-szer $x_2$, és így tovább $n+1-i$-szer $x_i$, végül 1-szer $x_n$. Ekkor FC sosem változtat a listán és a kiszolgálás teljes költsége

$$\sum_{i=1}^{n}(n+1-i)i = (n+1)\sum_{i=1}^{n} i - \sum_{i=1}^{n} i^2 = (n+1)^2 n/2 - n(n+1)(2n+1)/6 = \Theta(n^3).$$

Ezzel szemben MTF költsége $\sum_{i=1}^{n} i + \sum_{i=1}^{n-1} i = \Theta(n^2)$. Következésképpen $FC(I)/OPT(I) \geq FC(I)/MTF(I) \to \infty$, ha $n$ tart végtelenbe. □

**TRANSPOSE algoritmus:** Ha a keresett elem nem az első a listában, akkor egy hellyel előrébb mozgatjuk.

**2. lemma.** TRANSPOSE *nem konstans versenyképes.*

*Bizonyítás:* Legyen a lista $x_1,\ldots,x_n$ és vegyük a következő kéréssorozatot: $(x_n,x_{n-1})^M$, vagyis felváltva a két utolsó elemet kérjük, mindegyiket $M$-szer. Ekkor TRANSPOSE a kért utolsó elemet mindig felcseréli az előtte álló elemmel, így a teljes költsége $2M \cdot n$. Másrészt az MTF algoritmusnak, amely mindig előre hozza az aktuális elemet, a költsége az első két kéréstől eltekintve mindig 2, így összesen $2n + 2(M-1)2$. Következésképpen $TRANSPOSE(I)/OPT(I) \geq 2M \cdot n/(2n+4(M-1)) \to n/2$ ha $M$ tart végtelenbe. □

Az alábbi állítás azt mutatja, hogy nem adható meg olyan algoritmus, amelynek a versenyképességi hányadosa kisebb, mint az MTF algoritmusé. Az ilyen online algoritmusokat amelyek versenyképességi hányadosa a lehető legjobb, optimális algoritmusoknak nevezzük.

**6. tétel.** [48] *Ha egy online algoritmus $c$-versenyképes a lista karbantartási feladatra, akkor $c \geq 2$.*

*Bizonyítás:* Vegyünk egy tetszőleges online algoritmust. Definiáljunk egy $m$ hosszú bemenetet, amely mindig az algoritmus listájának utolsó elemét kéri. Ekkor az algoritmus költsége $m \cdot n$.

Az optimális költség becsléséhez vegyünk két statikus algoritmust. STAT1 nem mozdít egyetlen elemet sem, STAT2 pedig először megfordítja az elemek sorrendjét és utána nem mozdít egyetlen elemet sem.

Ekkor bármilyen kérés érkezik, annak a teljes költsége a két algoritmusra nézve pontosan $n+1$. Továbbá STAT2 esetén az elemek kezdeti sorrendje megfordításának költsége $(n-1)n/2$. Következésképpen STAT1 és STAT2 együttes költsége a bemenetre $(n-1)n/2 + m(n+1)$, így az optimális költség legfeljebb $((n-1)n/2 + m(n+1))/2$.

Tehát azt kaptuk, hogy erre a bemenetre az $ALG(I)/OPT(I)$ hányadosra kapott alsó korlát tart a $2n/(n+1)$ értékhez, ahogy $m$ tart a végtelenbe. Másrészt ezen hányados határértéke 2, ha $n$ tart a végtelenbe, amivel a tételt igazoltuk. □

# 4. fejezet

# Véletlenített online algoritmusok

## 4.1. Alapvető definíciók

Véletlenített algoritmusról beszélünk abban az esetben, ha az algoritmus véletlen döntéseket is hoz, és azon döntések kimenetelétől függ az algoritmus futása. Másként megfogalmazva arról van szó, hogy egy véletlenített algoritmus egy véletlen eloszlást definiál a determinisztikus algoritmusok terén, hiszen a véletlen döntéseknek minden lehetséges kimenete egy deterministikus algoritmushoz vezet. Mivel az algoritmus kimenete függhet a véletlen döntésektől, ezért optimalizálási feladatoknál a véletlenített algoritmusok esetén a kapott megoldás célfüggvényértéke nem minden futás esetén ugyanaz. Ilyen esetekben ennek a célfüggvénynek a várható értékét szokás vizsgálni.

**Példa:** Tegyük fel, hogy van két doboz A és B, amelyek egyike 1000 Ft-ot tartalmaz, a másik üres. 500 Ft-ért választhatunk egy dobozt, amelynek a tartalmát megkapjuk. Ekkor a feladat megoldására két determinisztikus algoritmus használható: vagy A-t választjuk, vagy B-t. Mindkét algoritmus költsége a legrosszabb esetben 500 (azon bemenet esetén, amelyben nem találtuk meg a pénzt). Másrészt, ha egy olyan véletlenített algoritmust használunk, amely $1/2$ valószínűséggel választja az egyik, illetve másik ládát, akkor mindkét bemenetre az algoritmus költségének várható értéke $1/2 \cdot 500 + 1/2 \cdot (-500) = 0$.

Véletlenített online algoritmusok esetén a versenyképesség definíciójában az $A(I)$ valószínűségi változó várható értékét használjuk, és ezt hasonlítjuk össze az optimális megoldás $OPT(I)$ értékével. A bemenetet generáló ellenféltől függően 3 modellt definiáltak:

- Hanyag ellenfél: a bemenetet az algoritmus véletlen döntéseinek eloszlása ismeretében, de a döntések kimenetének ismerete nélkül kell megadja.
- Adaptív online ellenfél: a bemenet generálása során mindig megkapja az online algoritmus döntéseinek kimenetét is, de a bemenetet az ellenfél is online oldja meg.
- Adaptív offline ellenfél: a bemenet generálása során mindig megkapja az online algoritmus döntéseinek kimenetét is, de a bemenetet az ellenfél offline, a bemeneti sorozat végén oldja meg.

A definíciók alapján jól látható, hogy az egyes fogalmak egyre erősebb ellenfelet definiálnak, így ha egy algoritmus C-versenyképes egy adaptív offline ellenfél ellen, akkor az C-versenyképes az adaptív online és a hanyag ellenfél ellen is. A továbbiakban ebben a jegyzetben csak a hanyag ellenfél esetét tárgyaljuk.

## 4.2. Játékelméleti alapfogalmak

A véletlenített algoritmusok elemzésénél jól használhatóak egyes, a mátrixjátékok vizsgálata során elért eredmények. Az alábbiakban tömören összefoglaljuk azokat a definíciókat és eredményeket, amelyeket a véletlenített algoritmusok elemzése során használni fogunk. Az érdeklődő olvasó egy részletesebb bevezetést találhat a játékelméletről a [47] tankönyvben.

Csak a játékoknak egy speciális osztályával, a véges, kétszemélyes zérusösszegű játékokkal fogunk foglalkozni. Ezeket a játékokat *mátrixjátékoknak* hívják. A játékban két játékos van, $A$ és $B$, mindkét játékosnak van véges sok stratégiája, amelyekből választhatnak. A játék zérusösszegű, ami azt jelenti, hogy amennyit az $A$ játékos nyer a $B$ játékos elveszíti, vagyis egymástól nyernek. Ekkor a játékot megadja a $C$ nyereségmátrix, a sorok $A$ stratégiáinak, az oszlopok $B$ stratégiáinak felelnek meg és a $C_{ij}$ mező $A$ nyeresége, ha a játékosok az $i$ és $j$ stratégiákat választják. Például, a

$$C = \begin{pmatrix} 500 & -500 \\ -500 & 500 \end{pmatrix}$$

játékban mindkét játékosnak két stratégiája van, és ha mindketten az első vagy mindketten a második stratégiát választják, akkor $A$ nyer 500-at, egyébként $B$ nyer 500-at, (vagyis $A$ veszít 500-at).

Ekkor az $A$ játékos bármely stratégiája esetén legrosszabb esetben a stratégiához tartozó sor minimumát nyeri, így garantálhatja, hogy nyeresége legalább a $\max_{i=1,\dots,m} \min_{j=1,\dots,n} C_{ij}$ értéket eléri. Másrészt a $B$ játékos bármely stratégiája esetén legrosszabb esetben a stratégiához tartozó oszlop maximumát veszíti el. Így garantálni tudja, hogy nem veszít többet a $\min_{j=1,\dots,n} \max_{i=1,\dots,m} C_{ij}$ értéknél. A fentiekből adódik, hogy

$$m = \max_{i=1,\dots,m} \min_{j=1,\dots,n} C_{ij} \leq \min_{i=1,\dots,m} \max_{j=1,\dots,n} C_{ij} = M.$$

Amennyiben a fenti egyenlőtlenségben egyenlőség áll fenn, akkor mindkét játékosnak azt a stratégiát érdemes játszania, amellyel a garantált maximális nyereséget illetve minimális veszteséget érik el, és ezt a közös értéket nevezik a játék értékének. Általában nem áll fenn egyenlőség a két érték között, a fentiekben használt példánkban $m = -500$ és $M = 500$. Ilyen esetekben szokás a kevert stratégiák vizsgálata.

**Kevert stratégiák:** Kevert stratégiák esetén az $A$ játékos nem egy stratégiát választ, hanem egy $p = (p_1, \dots, p_m)$ valószínűségi eloszlást definiál a stratégiái halmazán. Hasonlóan $B$ is egy $q = (q_1, \dots, q_n)$ eloszlást választ. Ekkor $p_i \cdot q_j$ annak a valószínűsége, hogy a játékosok az $i$-edik és $j$-edik stratégiát választják. Ezért a nyereség várható értéke $p^T C q = \sum_{i=1}^{m} \sum_{j=1}^{n} p_i C_{ij} q_j$.

A tiszta stratégiákhoz hasonlóan ekkor az $A$ játékos garantálni tudja a $\max_p \min_q p^T Cq$ nyereséget, a $B$ játékos pedig garantálni tudja, hogy nem veszít többet a $\min_q \max_p p^T Cq$ értéknél. A tiszta stratégiákkal ellentétben kevert stratégiák esetén ez a két érték biztosan megegyezik, amint azt az alábbi tétel mutatja.

**7. tétel.** *(**Neumann-féle Minimax Tétel**) Bármely $C$ mátrix által megadott 2 személyes zérusösszegű játékra*

$$\max_p \min_q p^T Cq = \min_q \max_p p^T Cq.$$

Ha $p$ rögzített érték, akkor a $p^T Cq$ függvény $q$-nak egy lineáris függvénye, és minimalizált az által, hogy az a $q_j$ érték 1-re van állítva, amihez a legkisebb együttható tartozik ebben a lineáris függvényben. Tehát ha a $B$ ismeri az $A$ játékos $p$ eloszlását, akkor az ő optimális stratégiája tiszta lehet. Ez fordítva is igaz, ha az $A$ ismeri az $B$ játékos $q$ eloszlását, akkor az ő optimális stratégiája tiszta lehet. Ez a minimax tétel egyszerűsített változatához vezet. Legyen $e_k$ egy egységvektor, 1-essel a $k$-adik pozícióban és 0-val a többi pozícióban. Ekkor a következő tételt kapjuk:

**8. tétel.** *(**Loomis Tétel**) Bármely 2 személyes, zérusösszegű, $C$ mátrix által adott játékra:*

$$\max_p \min_j p^T Ce_j = \min_q \max_i e_i{}^T Cq.$$

## 4.3. A játékelméleti reprezentáció

Amennyiben egy feladatnak adott méret mellett véges sok lehetséges bemenete van és véges sok lehetséges algoritmus adhat rá megoldást, akkor a probléma leírható játékelméleti eszközökkel. Az $A$ játékos generálja az $I$ bemenetet, a $B$ játékos pedig kiválasztja a ALG online algoritmust. A fejezet elején bemutatott példához a fenti mátrixjáték tartozik.

Ha egy online probléma versenyképességét vizsgáljuk, akkor a nyereségmátrix az $ALG(I)/OPT(I)$ értékeket tartalmazza, amit $A$ maximalizálni, $B$ pedig minimalizálni akar. A $B$ játékosnál a tiszta stratégia a determinisztikus algoritmusokat adja meg, a kevert a véletlenített algoritmusokat. Az $A$ játékos kevert stratégiái pedig a véletlenül generált bemeneteknek felelnek meg.

Tehát azt kaptuk, hogy ha a P online probléma olyan, hogy adott méretre véges számú bemenettel, és véges számú determinisztikus algoritmussal rendelkezik, akkor a fentieknek megfelelően a versenyképessége egy mátrixjátékkal írható le.

Loomis tétele alapján azt kapjuk, hogy a legrosszabb bemenet eloszlás esetére véve a lehető legjobb determinisztikus online algoritmust ugyanazt a versenyképességet tudjuk elérni, mint a legjobb lehetséges véletlenített algoritmussal a legrosszabb determinisztikus bemeneten. Ennek következménye a Yao elv, amit gyakran használnak alsó korlátok igazolására.

**Yao elv:** [52] Tetszőlegesen választott bemeneti eloszlásra nézve az optimális determinisztikus online algoritmus várható versenyképessége alsó korlátot ad a véletlenített online algoritmusok versenyképességére.

# 5. fejezet

# Példák véletlenített online algoritmusokra

## 5.1. Síbérlési feladat

Vegyük a következő véletlenített algoritmust a síbérlési feladatra (a sí vásárlási ára $B$ egység). Az R algoritmus $1/2$ valószínűséggel a $3/4B$ időpontig vár és utána vásárol, $1/2$ valószínűséggel pedig a $B$ időpontig vár és utána vásárol.

**9. tétel.** *Az* R *algoritmus 15/8-versenyképes.*

*Bizonyítás:* Vegyünk egy tetszőleges bemenetet, jelölje $I$. Azaz $I$ napig síelünk. Különböztessük meg a következő eseteket.

- Ha $I < 3/4B$, akkor az optimális költség $I$, továbbá $R$ költsége is $I$ mindkét döntés esetén, így az $E(R(I))/OPT(I)$ hányados 1.
- Ha $3/4B \leq I < B$, akkor az optimális költség I, továbbá R költsége vagy $B + 3/4B - 1$ vagy $I$. Tehát $E(R(I)) \leq 1/2 \cdot 7/4 \cdot B + 1/2 \cdot I$. Felhasználva, hogy $I \geq 3/4B$ kapjuk, hogy $E(R(I))/OPT(I) \leq (1/2 \cdot 7/4 \cdot B + 1/2 \cdot I)/I \leq 1/2 \cdot 7/4 \cdot 4/3 + 1/2 = 10/6$.
- Ha $B \leq I$, akkor $OPT(I) = B$. Az R algoritmus költsége pedig vagy $B + 3/4B - 1$ vagy $2B - 1$, így $E(R(I)) \leq 1/2 \cdot 7/4 \cdot B + 1/2 \cdot 2 \cdot B = 15/8B$. Következésképpen $E(R(I))/OPT(I) \leq 15/8$. □

Az első fejezetben beláttuk, hogy nem létezik $2 - 1/B$-nél kisebb versenyképességi hányadossal rendelkező determinisztikus algoritmus, így a fentiekben vizsgált véletlenített algoritmusnak kisebb a versenyképességi hányadosa, mint bármelyik determinisztikus algoritmusnak (feltéve hogy $B$ elég nagy, vagyis $B > 8$).

Az alábbiakban megmutatjuk, hogy miként használható az előző fejezetben bemutatott Yao elv a síbérlési feladat esetén.

**10. tétel.** *Nincs olyan véletlenített online algoritmus hanyag ellenfél ellen, amelynek a versenyképességi hányadosa kisebb lenne, mint* $5/4$.

*Bizonyítás:* Adjunk a Yao elv alapján egy alsó korlátot a síbérlési problémát megoldó véletlenített algoritmusok versenyképességi hányadosára. Használjuk a következő valószínűségű bemeneti eloszlást. A bemenet legyen $1/2$ valószínűséggel $B/2$ és $1/2$ valószínűséggel $3/2 \cdot B$.

Az első bemenet esetén az optimális költség $B/2$ a második esetében $B$. Ahhoz hogy használhassuk a Yao elvet, meg kell határoznunk az optimális determinisztikus algoritmusra az $A(I)/OPT(I)$ hányados várható értékét. Vegyünk egy tetszőleges determinisztikus online algoritmust. Ezt egyértelműen meghatározza egyetlen érték, ami azt adja meg, hogy hányadik napot követően vásárol sílécet, ha addig nem fejeződik be a síelés. Jelölje az algoritmust meghatározó értéket $x$.

Ha $x \leq B/2$, akkor az algoritmus költsége mindkét lehetséges bemeneten $x+B$. Tehát $E(A(I)/OPT(I)) = 1/2(x+B)/(B/2) + 1/2(x+B)/B \geq 3/2$.

Ha $B/2 < x \leq 3/2B$, akkor az algoritmus költsége az első lehetséges bemenetre $B/2$ a másodikra $x+B$, így $E(A(I)/OPT(I)) = 1/2(B/2)/(B/2) + 1/2(x+B)/B \geq 1/2 + 1/2 \cdot 3/2 = 5/4$.

Ha $3/2B \leq x$, akkor az algoritmus költsége az első lehetséges bemenetre $B/2$ a másodikra $3/2B$, így $E(A(I)/OPT(I)) = 1/2(B/2)/(B/2) + 1/2(3/2 \cdot B)/B = 5/4$.

Következésképpen minden determinisztikus online algoritmusra teljesül, hogy az adott bemeneti eloszlás mellett $E(A(I)/OPT(I)) \geq 5/4$, így a Yao elv alapján adódik, hogy nincs véletlenített algoritmus, amelynek a versenyképességi hányadosa kisebb lenne, mint $5/4$. □

## 5.2. Lapozás

Az alábbiakban a bélyegző algoritmus véletlenített változatát mutatjuk be, amelyben nem determinisztikus döntés alapján választjuk azt a lapot, amely kikerül a memóriából. A véletlenített bélyegző (RB) algoritmust a következőképpen definiálhatjuk.

**A RB algoritmus**

1. Ha a kért lap a memóriában van, akkor amennyiben még jelöletlen, megjelöljük.
2. Ha a kért lap nincs a memóriában, és nincs már jelöletlen lap a memóriában, akkor az összes jelölést töröljük.
3. Veszünk egy jelöletlen lapot egyenletes eloszlás alapján a memóriából (az előző lépés miatt van ilyen), a kért lapot ennek a lapnak a helyére berakjuk, majd megjelöljük.

**11. tétel.** [23] *A* RB *algoritmus* $2H_k$*-versenyképes hanyag ellenfél ellen, ahol* $H_k = \sum_{i=1}^{k} 1/i$.

*Bizonyítás:* Vegyünk egy tetszőleges $I$ bemenetet, és rögzítsünk egy optimális offline algoritmus, amit jelöljön OFF. A bemenetet bontsuk fázisokra ugyanúgy, mint a determinisztikus esetben. Ekkor ismét teljesül, hogy akkor kezdődik új fázis, amikor az RB algoritmus törli a bélyegeket a memóriában. Vegyük észre, hogy a fázisok nem függnek az algoritmus véletlen döntéseitől (a fázison belüli költség igen). Jelölje a fázisok számát $n$. Most

vizsgáljuk meg, hogy egy adott $i$-re mennyi RB várható költsége az $i$-edik fázisban. Ennek meghatározásához a fázis során kért lapokat két halmazba osztjuk. Réginek nevezzük azokat a lapokat, amelyek a fázis megkezdésekor RB memóriájában voltak, ezek számát jelölje $r_i$. A többi lapot (akik nem szerepeltek RB memóriájában) új lapnak nevezzük, ezek száma legyen $u_i$. Minden új lap hibát okoz, és egyetlen lap sem okoz egynél több hibát egy fázison belül, így azt kapjuk, hogy az új lapok által kapott hibák száma $u_i$.

Most vizsgáljuk meg, hogy mennyi a régi lapok által okozott hibák várható száma. Nyilvánvalóan egy régi lap csak az első megjelenésénél okozhat hibát, pontosan akkor, ha a lapot az algoritmus már kirakta a memóriájából. Vizsgáljuk meg ennek a valószínűségét. Tegyük fel, hogy egy régi lap első megjelenésénél, a memória tartalmaz $x$ darab új lapot és $y$ darab megjelölt régi lapot (ezek már szerepeltek a fázisban). Ekkor a $k-y$ jelöletlen régi lap közül egyenletes eloszlás alapján $x$ darab nem szerepel a memóriában, így annak a valószínűsége, hogy a kért lap nem szerepel, azaz hibát okoz $x/(k-y)$. Tehát a lap általi hibák számának várható értéke $x/(k-y) \leq u_i/(k-y)$. Következésképpen a régi lapok által okozott hibák várható száma legfeljebb $\sum_{j=0}^{r_i} u_i/(k-j)$. Így azt kapjuk, hogy az algoritmus várható költsége a fázis során legfeljebb

$$u_i + \sum_{j=0}^{r_i} u_i/(k-j) \leq u_i H_k.$$

Összefoglalva azt kaptuk, hogy $RB(I) \leq \sum_{i=1}^{n} u_i H_k$.

Most vizsgáljuk meg az optimális költséget. Ehhez legyen $d_i$ azon lapok száma, amelyek az $i$-edik fázis előtt szerepelnek az offline memóriában, de nem szerepelnek RB memóriájában. Feltesszük, hogy $d_1 = 0$, azaz ugyanazzal a memóriával kezdenek az algoritmusok. Használjuk a $d_{n+1}$ értéket is, ez az algoritmus befejezésekor fennálló érték. Legyen az offline algoritmus költsége az i-edik fázisban $OFF_i$. Mivel az i-edik fázisban pontosan azokat a lapokat kérik, amelyek szerepelnek a végén RB memóriájában, ezért minden lap, ami a fázis végén ezektől különbözik egy hibát okoz az offline algoritmus számára. Következésképpen $OFF_i \geq d_{i+1}$. Másrészt azon új lapok, amelyek nem voltak az offline algoritmus memóriájában a fázis előtt, a számára is hibát okoznak. Ezen lapok száma legalább $u_i - d_i$, így azt kapjuk, hogy $OFF_i \geq u_i - d_i$. A két korlát alapján adódik, hogy $OFF_i \geq (u_i - d_i + d_{i+1})/2$. Összegezve ezeket az értékeket adódik, hogy

$$OFF(I) \geq (\sum_{i=1}^{n} u_i - d_1 + d_{n+1})/2 \geq (\sum_{i=1}^{n} u_i)/2.$$

Következésképpen azt kapjuk, hogy $RB(I) \leq 2H_k OFF(I)$, amivel a tételt igazoltuk. □

A következő tétel azt mutatja, hogy hanyag ellenfél ellen nem adható RB-nél nagyságrendileg jobb algoritmus.

**12. tétel.** [23] *Nincs olyan véletlenített algoritmus a lapozási problémára, amelynek a versenyképességi hányadosa hanyag ellenfél ellen kisebb, mint $H_k$.*

*Bizonyítás:* A tétel bizonyításához ismét a Yao elvet használjuk fel. Tehát vehetünk egy tetszőleges valószínűségi eloszlást, és az azáltal generált bemeneten kell megvizsgálnunk a

legjobb determinisztikus online algoritmust. Általában egy adott eloszlásra a legjobb determinisztikus algoritmus megtalálása igen nehéz feladat, így az ilyen esetekben gyakran használt ötlet olyan eloszlást választani, amelyen minden algoritmus ugyanazt a várható eredményt éri el. Most is ezt tesszük. Vegyünk $k+1$ különböző lapot és legyen $I$ kérések egy olyan sorozata, amelyben $n$ darab kérés érkezik, melyek mindegyike egyenletes eloszlás alapján választ egy lapot a $k+1$ lap közül. Definiáljuk a generált sorozat fázisait, az előző bizonyításokhoz hasonlóan. Az $i$-edik fázis az LFD algoritmus $i$-edik hibájánál kezdődik és a következő hibát megelőző lapig tart. Ekkor LFD költsége minden fázisban 1. Most vizsgáljuk az adott fázis várható hosszát. Akkor következik be a következő hiba, amikor arra a lapra érkezik a kérés, amit LFD kitett a memóriából, de ezt megelőzően a többi lapra is kellett kéréseknek érkeznie. Tehát egy fázis várható hossza 1-el rövidebb a legrövidebb olyan sorozat várható hosszánál, amelyben mind a $k+1$ lapra érkezik kérés. Másrészt pontosan az ilyen sorozatok várható hosszát vizsgálja a kupon gyűjtő probléma (lásd [46]). Egy ilyen legrövidebb sorozat várható hossza $(k+1)H_{k+1}$. Azaz egy fázis várható hossza $(k+1)H_{k+1}-1=(k+1)H_k$.

Most vegyük észre, hogy tetszőleges online algoritmust véve minden lépésben a hiba valószínűsége, így várható értéke $1/(k+1)$. Tehát az online algoritmus várható költsége egy fázisban $(k+1)H_k/(k+1)=H_k$. Következésképpen minden fázisra az online algoritmus várható költsége $H_k$-szor az offline költség, amivel az állítást igazoltuk. □

## 5.3. Lista hozzáférés

A lista hozzáférési feladat esetén az MTF algoritmusnak alábbi véletlenített kiterjesztését szokás vizsgálni.

**BIT algoritmus:** Kezdetben minden elemhez hozzárendelünk egy bitet, 0-át vagy 1-et, egymástól függetlenül egyenletes eloszlás alapján. Az $i$ elemhez rendelt bit $b(i)$. Ezt követően, ha egy elemre meghívják a KERES algoritmust, akkor a hozzárendelt bitet megváltoztatjuk. Amennyiben a bit 0-ról 1-re változott, akkor az elemet a lista elejére visszük.

Az alábbi, bizonyítás nélkül közölt tétel mutatja, hogy ez az algoritmus kisebb versenyképességi hányadossal rendelkezik, mint az MTF algoritmus, sőt kisebb versenyképességi hányadossal, mint bármelyik determinisztikus algoritmus.

**13. tétel.** [35] *A* BIT *algoritmus 1.75-versenyképes.*

A BIT algoritmusnál kisebb versenyképességi hányadossal rendelkezik a COMB (kombinált) algoritmus, amely a jelenleg ismert legkisebb versenyképességi hányadossal rendelkező véletlenített online algoritmus a lista hozzáférési feladatra. Az algoritmus az alábbi 2-versenyképes determinisztikus algoritmust használja.

**TIMESTAP algoritmus:** Az $x$ elemet a KERES($x$) műveletet követően a lista legelső olyan eleme elé rakjuk a listában, amelyet legfeljebb egyszer kértek $x$ utolsó kérése óta.

Ekkor a COMB algoritmust a következőképpen definiálhatjuk.

**COMB algoritmus:** 4/5 valószínűséggel a BIT algoritmust használjuk, 1/5 valószínűséggel a TIMESTAP algoritmust.

Az algoritmus versenyképességét az alábbi tétel adja meg, amely bizonyítása túlmutat a jegyzet keretein.

**14. tétel.** [2] *A COMB algoritmus 1.6 - versenyképes.*

# 6. fejezet

# A $k$-szerver probléma

Az egyik legismertebb online modell a *k-szerver* probléma. A probléma 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, vagyis $d$ szimmetrikus,
- $d(x,y) + d(y,z) \geq d(x,z)$ minden $x,y,z \in M$ esetén, vagyis teljesül a háromszög-egyenlőtlenség,
- $d(x,y) = 0$ akkor és csak akkor teljesül, ha $x = y$.

A $k$-szerver problémában adott egy metrikus tér, és van $k$ darab szerverünk, amelyek a térben mozoghatnak. A probléma során a tér pontjaiból álló kérések egy listáját kell kiszolgálni azáltal, hogy a megfelelő kérések helyére odaküldünk egy-egy szervert.

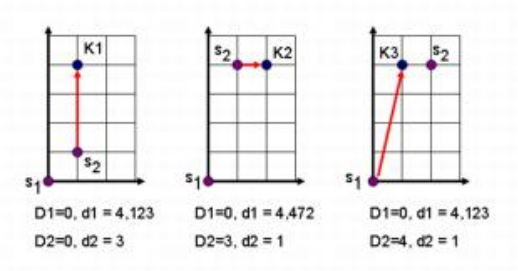
A probléma online, ami azt jelenti, hogy a kéréseket egyenként kapjuk meg, és az egyes kéréseket a további kérések ismerete nélkül azok érkezése előtt kell kiszolgálnunk. A cél a szerverek által megtett össztávolság minimalizálása. Ezen modellnek és speciális eseteinek számos alkalmazása van. A továbbiakban 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 problémára a [44] publikálták. A probléma megoldására javasolt első eljárás a következő *Egyensúly algoritmus*, amelyet a továbbiakban ES-el 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. 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 a algoritmus egy $I = P_1, \ldots, P_n$ pontsorozatra a következőképpen fut

**ES**$(I)$
for $j = 1$ **to** $n$
$i = \text{argmin}\{D_i + d(s_i, P_j)\}$
szolgáljuk ki a kérést az $i$-edik szerverrel
$D_i := D_i + d(s_i, P_j)$
$s_i := P_j$

**Példa:** Tekintsük a kétdimenziós euklideszi teret, mint metrikus teret. A pontok $(x,y)$ valós számpárokból állnak, két pontnak $(a,b)$-nek és $(c,d)$ -nek a távolsága $\sqrt{(a-c)^2+(b-d)^2}$. Legyen két szerverünk kezdetben a $(0,0)$ és $(1,1)$ pontokban. Kezdetben $D_1 = D_2 = 0$, $s_1 = (0,0)$, $s_2 = (1,1)$. 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 futását a 6.1 ábrán szemléltetjük.




6.1. ábra. Az ES algoritmus lépései

Az algoritmus hatékony speciális terek esetén, miként ezt a következő állítás mutatja. A tétel bizonyítását nem ismertetjük.

**15. tétel.** [44] *Amennyiben a metrikus tér legalább $k+1$ pontot tartalmaz, akkor az* ES *algoritmus enyhén $k$-versenyképes.*

Az alábbi állítás mutatja, hogy a $k$-szerver problémára általában nem adható meg $k$-versenyképesnél jobb algoritmus.

**16. tétel.** [44] *Nincs olyan legalább $k+1$ pontból álló metrikus tér, ahol megadható lenne olyan online algoritmus, amelynek kisebb a versenyképességi hányadosa, mint $k$ (enyhe versenyképesség esetén).*

*Bizonyítás:* Tekintsünk egy tetszőleges legalább $k+1$ pontból álló teret, és egy tetszőleges online 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 a ONL algoritmusnak nem tartózkodik szervere.

Vizsgáljuk elsőként az $ONL(I)$ 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

$$ONL(I) = \sum_{j=1}^{n} d(Q_j, Q_{j+1}),$$

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 $OPT(I)$ költséget. Az optimális offline algoritmus meghatározása helyett definiálunk $k$ darab offline 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 tehát $k$ darab offline algoritmust, jelölje őket $\mathrm{OFF}_1, \ldots, \mathrm{OFF}_k$. Tegyük fel, hogy a kiindulási állapotban az $\mathrm{OFF}_j$ 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 $\mathrm{OFF}_j$ 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 az ezen ponttól különböző $P_1, P_2, \ldots, P_{k+1}$ pontok mindegyikén, így $Q_{i-1}$-en is van szerver. Továbbá a $Q_1 = P_{k+1}$ ponton az $\mathrm{OFF}_j$ 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 ez a tulajdonság az alábbi észrevételek alapján igazolható. Amely algoritmusok nem mozdítanak szervert a kérés kiszolgálására, azoknak a szerver konfigurációja nem változik. Amely algoritmusok mozgatnak szervert, azok mind 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 az a mozgatást megelőzőleg különböző volt.

Következésképp egy $Q_i$ kérés esetén minden $\mathrm{OFF}_j$ algoritmusra más a szerver konfiguráció. Továbbá minden konfigurációnak tartalmaznia kell $Q_{i-1}$-et, tehát pontosan egy olyan $\mathrm{OFF}_j$ algoritmus van, amelynek nincsen szervere a $Q_i$ ponton. Tehát a $Q_i$ kérés kiszolgálásának a költsége az $\mathrm{OFF}_j$ algoritmusok egyikénél $d(Q_{i-1}, Q_i)$ a többi algoritmus esetén 0.

Következésképp

$$\sum_{j=1}^{k} OFF_j(I) = C + \sum_{i=2}^{n} d(Q_i, Q_{i-1}),$$

ahol $C = \sum_{j=1}^{k} C_j$ egy, a bemeneti sorozattól független konstans, amely az offline algoritmusok kezdeti konfigurációinak beállításának költsége.

Másrészt az offline optimális algoritmus költsége nem lehet nagyobb semelyik offline algoritmus költségénél sem, így $k \cdot \mathrm{OPT}(I) \leq \sum_{j=1}^{k} OFF_j(I)$. Következésképpen

$$k \cdot OPT(I) \leq C + \sum_{i=2}^{n} d(Q_i, Q_{i-1}) \leq C + ONL(I),$$

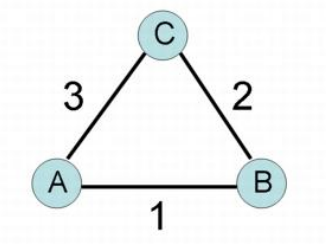
amely egyenlőtlenségből következik, hogy az ONL algoritmus versenyképességi hányadosa nem lehet kisebb, mint $k$, hiszen a bemeneti sorozat hosszúságának növelésével a $OPT(I)$ érték tetszőlegesen nagy lehet. □

A probléma felvetése nagy érdeklődést keltett. Az általános esetre konstans versenyképes algoritmust ($O(2^k)$-versenyképes algoritmust) elsőként a [25] cikkben fejlesztettek ki. Ezt követően hosszan nem sikerült lényegesen csökkenteni a felső és az alsó korlát közötti rést. Az áttörést a [41] cikkben publikált eredmény hozta meg, ahol sikerült a probléma megoldására a munkafüggvényen alapuló algoritmust elemezniük és igazolniuk, hogy az algoritmus $(2k-1)$-versenyképes. Nem sikerült meghatározniuk az algoritmus pontos versenyképességi hányadosát, bár általánosan sejtett, hogy az algoritmus valójában $k$-versenyképes. Ezen versenyképességi hányados pontos meghatározása, illetve egy $k$-versenyképes algoritmus kifejlesztése azóta is az online algoritmusok elméletének legismertebb és sokak által legfontosabbnak tartott nyílt problémája. Az alábbiakban ismertetjük a munkafüggvény algoritmust.

Legyen $A_0$ az online 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 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, amely szervernek a $P$ helyére a $w_{t-1}(A_{t-1} \setminus \{P\} \cup \{R_t\}) + d(P, R_t)$ érték a minimális.

**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 teret a 6.2 ábrán szemléltetjük. A kezdeti szerver konfiguráció legyen $A,B$, vagyis két szerverünk van, az $A$ és $B$ pontokban. 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 $w_0(\{A,B\} \setminus \{A\} \cup \{C\}) + d(A,C) = 3+3 = 5$ és $w_0(\{A,B\} \setminus \{B\} \cup \{C\}) + d(B,C) = 2+2 = 4$, így a MUNKAFÜGGVÉNY algoritmus a $B$ pontban levő szervert küldi a kérés kiszolgálására.

A munkafüggvény algoritmusra teljesül a következő állítás, amely bizonyítása túlmutat a jegyzet keretein.

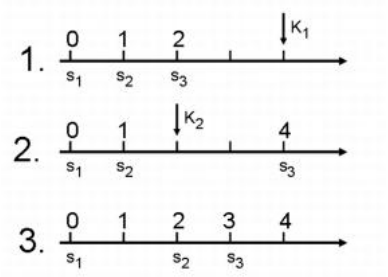



6.2. ábra. A példában szereplő metrikus tér

**17. tétel.** [41] *A* MUNKAFÜGGVÉNY *algoritmus enyhén* $(2k-1)$*-versenyképes.*

Az általános probléma vizsgálata mellett a probléma speciális eseteit számos dolgozatban vizsgálták. Amennyiben bármely két pont távolsága 1, akkor a lapozási problémát kapjuk meg. A pontok megfeleltethetőek a kért lapoknak, az egyes szerverek megfeleltethetőek a memóriában szereplő egyes helyeknek. Továbbá az, hogy egy szerver egy pontot kiszolgál azt jelenti, hogy az adott lapot berakjuk a szervernek megfelelő helyre a memóriába. 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, a [12] cikkben fejlesztettek ki 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 $d(s,P)$ egységnyivel mozdítja $P$ felé. A továbbiakban a dupla lefedő algoritmust DL-el jelöljük.

**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 példát a 6.3 ábrán szemléltetjük.




6.3. ábra. A dupla lefedő algoritmus működése

A DL algoritmusra teljesül a következő állítás.

**18. tétel.** [12] *A* DL *algoritmus enyhén k-versenyképes ha a metrikus tér egy egyenes.*

*Bizonyítás:* Vegyünk a kéréseknek egy tetszőleges sorozatát, jelölje ezt a bemenetet $I$. Az eljárás elemzése során feltételezzük, hogy párhuzamosan fut a DL algoritmus és egy optimális offline algoritmus. Szintén feltesszük, hogy minden kérést elsőként az offline algoritmus szolgál ki, utána pedig az online algoritmus. Az online algoritmus szervereit és egyben a szerverek pozícióit (amelyek valós számok az egyenesen) $s_1,\dots,s_k$ jelöli, az optimális offline algoritmus szervereit és egyben a szerverek pozícióit $x_1,\dots,x_k$ jelöli. Mivel a szerverek rendszeres átjelölésével ez elérhető feltételezzük, hogy $s_1 \le s_2 \le \cdots \le s_k$ és $x_1 \le x_2 \le \cdots \le x_k$ mindig teljesül.

A tétel állítását a potenciálfüggvény technikájával igazoljuk. A potenciálfüggvény a szerverek aktuális pozíciójához rendel egy értéket, az online és az offline 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 az alábbi állítások.

- Amikor OPT szolgálja ki a kérést, akkor a potenciálfüggvény növekedése legfeljebb $k$-szor akkora mint az OPT szerverei által megtett távolság.
- Amikor DL szolgálja ki a kérést, akkor $\Phi$ 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 $\Phi_v - \Phi_0 \le k\cdot \mathrm{OPT}(I) - \mathrm{DL}(I)$, ahol $\Phi_v$ és $\Phi_0$ a potenciálfüggvény kezdeti és végső értékei. Mivel a potenciálfüggvény nemnegatív, ezért adódik, hogy $\mathrm{DL}(I) \le k\mathrm{OPT}(I) + \Phi_0$, 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 rész legfeljebb $kd$-vel növekszik a második rész nem változik, tehát az első tulajdonsága a potenciálfüggvénynek valóban fennáll.

Most vizsgáljuk DL szervereit. Legyen $P$ az a kérés melyet 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 $\Phi$ értéke csökken $d(s_1,P)$ egységgel.

Most tekintsük a másik esetet! Ekkor $P$-nek mindkét oldalára esik szerver, legyenek ilyen szerverek például $s_i$ 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 $\Phi$ 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. □

Az egyenes általánosításaként kaphatjuk a fa metrikus teret. Vegyünk egy tetszőleges súlyozott fát. A metrikus tér pontjai a fa csúcsai, továbbá minden élhez, minden $0<\lambda<1$ értékre definiáljuk az élet $\lambda$ és $1-\lambda$ arányban felosztó pontokat. Mivel a fa körmentes ezért bármely két pont között egyetlen út vezet. A két pont közötti távolság a közöttük vezető út hossza. Az élek belső pontjai esetén az él hosszának a felosztásnak megfelelő részét vesszük figyelembe.

Ekkor definiálhatjuk a DL algoritmus egy érdekes általánosítását. Az algoritmus definiálásához szükségünk van a következő fogalomra. Azt mondjuk, hogy egy szerver lát egy pontot, ha a közte és a pont között levő úton nincs másik szerver.

**Lefedő algoritmus:** Az algoritmus egy kérést úgy szolgál ki, hogy minden szervert, ami látja a kérés helyét egyenletes sebességgel mozgat a szerver felé. A kérést látó szerverek halmaza csökkenhet a szerverek mozgatása során, amikor az első szerver eléri a kérést a többi szerver is megáll, mert utána már nem látják.

A DL algoritmus elemzéséhez hasonlóan igazolható az alábbi állítás.

**19. tétel.** [12] *Ha a metrikus tér egy fa, akkor a lefedő algoritmus enyhén k-versenyképes.*

# 7. fejezet

# Ütemezési feladatok

## 7.1. Online ütemezési modellek

Az ütemezési feladatok elméletének igen nagy irodalma van. Az első online ütemezési eredményt tulajdonképpen a [28] cikkben publikálták, ahol a LISTA online ütemezési algoritmust elemezték. Mondhatjuk ezt annak ellenére, hogy ott még nem használták az online algoritmusok körében később elterjedt fogalomrendszert és a vizsgált algoritmust nem online algoritmusként, hanem közelítő algoritmusként tekintették. Speciálisan online ütemezési algoritmusokkal az 1990-es években kezdtek el foglalkozni és azóta számos eredmény született.

Ütemezési problémákban 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 idejeit hozzárendeljük a gépet, amelyen a munkát végrehajtjuk és a megmunkálás kezdési és befejezési időpontját, amely időpontok különbsége a megmunkálási idő kell legyen.

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 $P_j = (p_j(1), \ldots, p_j(m))$ nemnegatív vektor lehet, ahol a munka megmunkálási ideje az $i$-edik gépen $p_j(i)$, akkor független párhuzamos gépekről beszélünk.

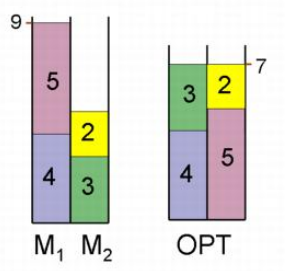
Ütemezési problémák esetén számos ütemezési célfüggvényt szokás vizsgálni, mi itt csak azt a modellt vizsgáljuk, amelyben a cél a maximális befejezési idő minimalizálása. A fejezet első részében ismertetjük a két legelterjedtebb online ütemezési modellt, és a következő két fejezetben ezen modellekkel foglalkozunk. Végül két újabb részterületet ismertetünk. Elsőként azt a modellt mutatjuk be, amelyben lehetséges a munkák visszautasítása, majd azt a modellt, ahol a gépeket is meg kell vásárolni.

### 7.1.1. 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 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 offline 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. Hasonlóan gépek és munkák egy tetszőleges halmazára azt az értéket, amit úgy kapunk, hogy a munkák megmunkálási idejeinek összegét elosztjuk a gépek számával a munkáknak az adott géphalmazra vonatkozó töltésének nevezzük.

**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 online 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. A kapott ütemezést és az optimális ütemezést mutatja a 7.1 ábra.



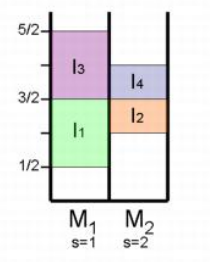

7.1. ábra. A két azonos gép példájában szereplő ütemezések

### 7.1.2. Idő modell

A második modellben, a munkáknak egy $r_j$ *érkezési ideje* is van, a munkáról semmit sem tudunk a érkezési ideje előtt (még a létezését sem), de a munka érkezése után bármikor meg-

kezdhetjük a munka végrehajtását, ha van olyan gépünk, amely nem dolgozik. Amennyiben egy munkát elkezdünk végrehajtani, akkor nem szakíthatjuk félbe.

**Példa:** Tekintsünk egy két darab hasonló párhuzamos gépből 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 a $5/2$ 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 a kisebb 2 költséggel is ütemezhette volna a munkákat. A példában szereplő ütemezést mutatja a 7.2 ábra.




7.2. ábra. Az idő modell példájában szereplő ütemezés

Fontos megjegyeznünk, hogy (a Lista modellel ellentétben) ebben a modellben bizonyos esetekben hasznos lehet várakoztatni munkákat, ezzel helyet hagyva az esetleg később érkező nagyobb megmunkálási idővel rendelkező munkák számára. Ilyen esetet látunk a következő példában.

**Példa:** Tekintsünk ismét két gépet, de ezek legyenek egyforma párhuzamos gépek, vagyis ahol mindkét gép sebessége $s = 1$. Az első két munka érkezzen a 0 időpontban, mindkettő 2 megmunkálási idővel. Tegyük fel, hogy ezeket az algoritmus rögtön elkezdi végrehajtani, egyiket az első, másikat a második gépen. Eztán az 1 időpontban érkezzen egy további munka, 3 megmunkálási idővel. Ennek végrehajtásával ekkor várni kényszerülünk, amíg a két folyamatban levő munka végrehajtása befejeződik, vagyis a 2 időpontig. A hosszabb munkát csak ekkor lehet elkezdeni. A mindkét utolsó munka az 5 időpontra készül el, de az optimális befejezés a 4 időpont lenne, amennyiben az első két munka közül az egyiket 0, a másikat a 2 időpontban kezdenénk el ugyanazon a gépen, ekkor a később érkező munkát rögtön el lehetne kezdeni az érkezésekor a másik gépen, és a teljes átfutási idő csak 4 egység lenne. Persze, nem tudhatjuk előre, hogy ilyen hosszabb munka később érkezni fog-e, ezért nem tudhatjuk előre, hogy érdemes-e még várni valamilyen további munkára, vagy sem. Csak annyit tehetünk, hogy ha várunk (további munkára), akkor nem várunk túl sokat.

## 7.2. A Lista modell

Az alábbiakban egy egyszerű online algoritmussal ismerkedünk meg. Az eljárás ismertetése előtt megismételjük, hogy az algoritmusnak csak az egyes gépekhez kell hozzárendelni az egyes munkákat, ezt követően az egyes gépeken a maximális befejezési idő minden ütemezésre, amelyben nincsenek üres időtartamok, megegyezik a géphez rendelt munkák megmunkálási idejeinek összegével.

Az első algoritmus, amelyet LISTA algoritmusnak nevezünk, a következőképpen működik.

### LISTA algoritmus

*Előkészítő rész.* A $j_1$ munkát rendeljük az $M_1$ géphez, továbbá legyen $r := 1$.

*Iterációs rész* (r-edik iteráció). Ha $r = n$, akkor vége az eljárásnak. Ellenkező esetben a $j_{r+1}$ munkát rendeljük ahhoz a géphez, amelyre a gépen levő töltés minimális. Ha több ilyen gép is van, akkor válasszuk ezek közül a legkisebb indexűt. Növeljük $r$ értékét 1-gyel, és folytassuk az eljárást a következő iterációval.

Az algoritmus az optimálishoz közeli eredményt eredményez, amint azt az alábbi tétel mutatja.

**20. tétel.** [28] *Egyforma párhuzamos gépek esetén a LISTA algoritmus versenyképességi hányadosa* $2-1/m$*, ahol* $m$ *a gépek száma.*

*Bizonyítás:* Elsőként igazoljuk, hogy LISTA $2-1/m$ versenyképes algoritmus. Legyen $\sigma = \{j_1, \ldots, j_n\}$ tetszőleges munkasorozat rendre $p_1, \ldots, p_n$ megmunkálási időkkel. Tekintsük a LISTA algoritmus által kapott ütemezést. Legyen $j_l$ az a munka, amely a legkésőbb fejeződik be. Vizsgáljuk ezen munka $S_l$ kezdési idejét. Mivel egyetlen gép sem kezdte el ezt a munkát ütemezni $S_l$ előtt, ezért minden gép szünet nélkül dolgozott az $S_l$ időpontig. Ebből azt kapjuk, hogy

$$S_l \leq \frac{1}{m}\sum_{\substack{j=1\\ j\neq l}}^{n} p_j = \frac{1}{m}\Big(\sum_{j=1}^{n} p_j - p_l\Big) = \frac{1}{m}\Big(\sum_{j=1}^{n} p_j\Big) - \frac{1}{m}p_l.$$

Következésképp

$$LISTA(\sigma) = S_l + p_l \leq \frac{1}{m}\Big(\sum_{j=1}^{n} p_j\Big) + \frac{m-1}{m}p_l.$$

Másrészt az optimális ütemezésben is végre kell hajtani az összes munkát, így $OPT(\sigma) \geq \frac{1}{m}(\sum_{j=1}^{n} p_j)$. Továbbá a $p_l$ munkát is végre kell hajtani valamely gépen, így $OPT(\sigma) \geq p_l$. Ezen becslések alapján egyből adódik, hogy

$$LISTA(\sigma) \leq \Big(1 + \frac{m-1}{m}\Big)OPT(\sigma),$$

amivel bizonyítottuk, hogy LISTA $2-1/m$ versenyképes algoritmus.

Most igazoljuk, hogy a kapott korlát éles. Vegyünk $m(m-1)$ darab munkát $1/m$ megmunkálási idővel, majd egy munkát 1 megmunkálási idővel. Ekkor a LISTA algoritmus az első $m(m-1)$ munkát egyenletesen elosztja a gépek között, majd az utolsó munkát az $M_1$ gépen ütemezi. Tehát a maximális befejezési idő $1+(m-1)/m$ lesz. Egy optimális ütemezés pedig a rövid munkákat egyenletesen osztja szét az első $m-1$ gép között, majd az utolsó munkát az $m$-edik géphez rendeli, és a maximális befejezési ideje 1 lesz. Tehát ebben az esetben az algoritmus által kapott megoldás és az optimális megoldás célfüggvényértékeinek hányadosa $2-1/m$, amivel igazoltuk az állításunkat. □

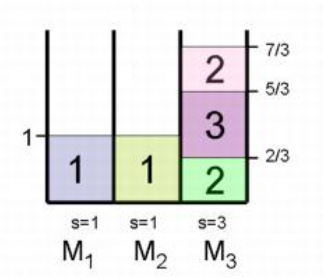
Első ránézésre nehéz elképzelni más algoritmust az online esetre, de több algoritmust fejlesztettek ki, amelyek versenyképességi hányadosa 2-nél kisebb számhoz konvergál, amennyiben a gépek száma tart a végtelenhez. Ezen algoritmusok többsége azon az ötleten alapszik, hogy a gépek többségén igyekszik 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 jelenleg legkisebb versenyképességi hányadossal $(1+\sqrt{(1+\ln 2)/2}\approx 1.9201$-hez tart, ha a gépek száma tart a végtelenhez) rendelkező algoritmust a [24] cikkben publikálták.

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.

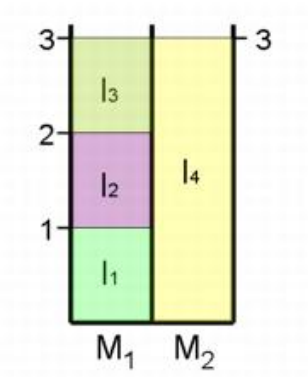
**Példa:** Tekintsük a hasonló párhuzamos gépek esetét, ahol 3 darab gép van és a sebességek $s_1=s_2=1$, $s_3=3$. Vegyük a következő $I=(2,1,1,3,2)$ munkasorozatot, ahol a munkák a megmunkálási idők által vannak megadva. Ekkor az első munka $2/3$-kor fejezhető be az $M_3$ gépen és a 2 időpontban a többi gépeken, így $M_3$ hoz rendeljük. A következő munka az 1 időpontra fejezhető be az összes gépen, így $M_1$ kapja. A következő munka a 2 időpontra fejezhető be $M_1$-en és az 1 időpontban a többi gépen, így $M_2$ kapja. A negyedik munka $M_1$-en $M_2$-n a 4 időpontban fejeződne be, az $M_3$-on $5/3$, így $M_3$-ra kerül. Végül az utolsó munka $M_1$-en $M_2$-n a 3 időpontban fejeződne be, az $M_3$-on $7/3$, így $M_3$-ra kerül. Az ütemezést a 7.3 ábra mutatja.

**Példa:** Tekintsük a független 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 az 2 időpontra fejezhető be mindkét gépen, így $M_1$ kapja. A harmadik 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 ütemezést a 7.4 ábra mutatja.

Az algoritmus versenyképességi hányadosát független gépek esetén a [4] cikkben hatá-




7.3. ábra. A mohó algoritmus a hasonló gépekre




7.4. ábra. A független párhuzamos gépek példája

rozták meg.

**21. tétel.** [4] *A* MOHÓ *algoritmus versenyképességi hányadosa m a független párhuzamos gépek esetében, ahol m a gépek száma.*

*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 $\infty$, (vagyis $p_1(1) = 1, p_1(i) = \infty, i = 2, \ldots, m-1, p_1(m) = 1+\varepsilon$), ezt követően a $j$-edik munkára a megmunkálási idő $j$ a $j$-edik gépen, $1+\varepsilon$ a $j-1$-edik gépen, és $\infty$ a többi gépen (vagyis $p_j(j-1) = 1+\varepsilon$, $p_j(j) = j$, $p_j(i) = \infty$, 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 offline 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 $\varepsilon$ 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. □

Az alábbi, a hasonló párhuzamos gépek esetére vonatkozó állítás a [4] és [11] cikkekben került publikálásra.

**22. tétel.** [4, 11] *A* MOHÓ *algoritmus versenyképességi hányadosa* $\Theta(\log m)$ *hasonló párhuzamos gépek esetén.*

Elsőként megmutatjuk, hogy az algoritmus versenyképességi hányadosa nem lehet kisebb, mint $\Omega(\log m)$. Tekintsük a gépeknek a következő halmazát. 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 $G_i$ 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 $m = 1 + \frac{2}{3}(4^k - 1)$.

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 offline algoritmus ütemezheti az $i$-edik fázis munkáit a $G_{k+1-i}$ géphalmaz gépein, ekkor a maximális befejezési idő 1, így az optimális offline 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, és így tovább 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 $k = \Omega(\log m)$, ezért az állítást igazoltuk.

Most megmutatjuk, hogy az algoritmus $O(\log m)$-versenyképes. Ehhez vegyünk egy tetszőleges bemenetet. Legyen $L$ az algoritmus által kapott maximális befejezési idő, $L^*$ pedig

az optimális offline algoritmus által kapott maximális befejezési idő. A bizonyítás alapötlete az alábbi lemmákon alapul, amelyek az egyes gépeken levő töltés mennyiségére adnak alsó korlátot.

**3. lemma.** *A töltés a leggyorsabb gépen legalább $L - L^*$*

Vegyük azt a munkát, amelynek a befejezési ideje megegyezik a maximális befejezési idővel. Ha a munka a leggyorsabb gépen 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 töltés a leggyorsabb gépen legalább $L - L^*$ kellett legyen, hiszen különben a munkát ott ütemeztük volna.

**4. lemma.** *Amennyiben a töltés minden legalább $v$ sebességű gépen legalább $l$, akkor a töltés legalább $l - 4L^*$ minden legalább $v/2$ sebességű gépen.*

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 offline algoritmus lassabb gépen ütemez (hiszen ellenkező esetben nem lehetne az offline maximális befejezési idő $L^*$).

Legyen egy ilyen munka $j$. Mivel $v$-nél lassabb gépen ütemezi az offline algoritmus, ezért a megmunkálási súlya legfeljebb $vL^*$. Következésképpen ezen munka megmunkálási ideje a legalább $v/2$ sebességű gépeken legfeljebb $2L^*$. Mivel ezen munka befejezési ideje 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. □

Most rátérünk a tétel bizonyítására. Legyen $v_{\max}$ a leggyorsabb gép sebessége, ekkor ezen a gépen a töltés legalább $L - L^*$. Ezt követően többször alkalmazva a fenti 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 $L - (1 + 4\lceil \log m \rceil)L^*$. Jelölje $I$ a legfeljebb $v_{\max}/m$ sebességű gépek halmazát!

Vizsgáljuk most meg a munkák megmunkálási súlyainak $W$ összegét! Mivel az offline algoritmus úgy osztja el a munkákat, hogy 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 online 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

$$L-(1+4\lceil\log m\rceil)L^* \leq 2L^*,$$

amiből adódik, hogy $L \leq 3+4\lceil\log m\rceil)L^*$, azaz igazoltuk, hogy az algoritmus $O(\log m)$-versenyképes. □

### 7.2.1. Az Idő modell

Ebben a modellben egyetlen algoritmust elemzünk, amelynek alapötlete, hogy a munkákat az érkezési idők alapján szétosztja és az egyes munkahalmazokat a munkák ismeretében optimálisan offline módon ütemezi. Ezt a algoritmust *intervallumonkénti ütemező algoritmusnak* nevezzük és INTV algoritmusként jelöljük. Az algoritmust a [49] cikkben publikálták. Legyen $t$ az első munka érkezési ideje, ettől az időponttól kezdve az algoritmus a következőképpen viselkedik.

**INTV Algoritmus**

*amíg a munkasorozat nem ér véget*

1. legyen $H$ a $t$ időpontig megérkezett és ütemezetlen munkák halmaza
2. legyen $OFF$ ezen munkákra egy optimális offline ütemezés
3. rendeljük hozzá a munkákat a gépekhez a kapott ütemezésnek megfelelően
4. legyen $q$ a kapott maximális befejezési idő
5. ha érkezett a $(t,q]$ intervallumban új munka vagy a munkasorozatnak vége van, legyen $t := q$
6. egyébként legyen $t$ a következő munka érkezési ideje

A INTV algoritmus versenyképességére vonatkozik a következő állítás.

**23. tétel.** [49] *Az Idő modellben az* INTV *algoritmus 2-versenyképes.*

*Bizonyítás:* Vegyük a munkáknak az algoritmus által kapott ütemezését. Az algoritmus tulajdonképpen fázisokban dolgozik, minden offline ütemezési megoldás végrehajtása egy fázisnak felel meg. Jelölje $i$ ezen fázisok számát, vagyis azt ahányszor az optimális ütemezést megkerestük a már megérkezett de még nem ütemezett munkákra. Továbbá jelölje $t_j$ a $j$-edik fázis kezdetét minden $j$-re és $T$ az ütemezés befejezési idejét. Legyen $T_3 = T - t_i$, $T_2 = t_i - t_{i-1}$, $T_1 = t_{i-1}$ és $T_{OPT}$ az optimális offline költség. Ekkor $T_2 \leq T_{OPT}$. Ez az észrevétel nyilvánvaló abban az esetben ha az $(i-1)$-edik fázis befejezésekor nincs ütemezendő munka. Ha az utolsó fázis rögtön az $(i-1)$-edik fázis befejezésekor kezdődik, akkor az egyenlőtlenség azért teljesül, mert az $i$-edik lépésben 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 lépésben 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. Az [50] dolgozatban igazolták, hogy az ONLINE 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 az [50] dolgozatban igazolták az alábbi állítást.

**24. tétel.** [50] *Nincs olyan online algoritmus az online ütemezés Idő modelljében a maximális befejezési idő minimalizálására, amelynek a versenyképességi hányadosa kisebb, mint 4/3.*

*Bizonyítás:* A bizonyításban egy némileg erősebb állítást igazolunk. Legyen $\alpha \approx 0,3473$ az $\alpha^3 - 3\alpha + 1 = 0$ egyenlet $[1/3, 1/2]$ intervallumba eső megoldása. Igazoljuk, hogy egyetlen online algoritmusnak se lehet kisebb a versenyképességi hányadosa, mint $1+\alpha$. Vegyünk egy tetszőleges online 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 $ALG(I)/OPT(I) > 1+\alpha$, 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 pedig $1+\alpha/(1-\alpha) - S_2$. Ekkor egy optimális offline 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 online algoritmus esetében az utolsó $m+1$ munkából legalább egyet csak az első vagy a második munka befejezése után kezdhetünk el, így erre a bemenetre $ALG(I) \geq 1 + 2\alpha/(1-\alpha)$, 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 $S_1 + 1 - \alpha/(1-\alpha)$ az utolsó $m$ munka egyikét sem kezdte el az online algoritmus, ezért van olyan gép, amelyen ezután az időpont uán 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 $OPT(I)/ALG(I)$ hányados az $S_1 = \alpha$ érték mellett minimális, és ebben az esetben a hányados $1+\alpha$, amivel az állítást igazoltuk. □

## 7.3. Visszautasításos modellek

A visszautasításos modellt a [7] cikkben definiálták. Ebben a modellben is $m$ darab gép van, minden munkának van egy $p_j$ megmunkálási ideje de a munkáknak van egy $b_j$ büntetés érté-

ke, amely a visszautasítás költségét adja meg. A munkákat vissza lehet utasítani, az elfogadott munkákat ütemezni kell, a minimalizálandó teljes költség a kapott ütemezés maximális befejezési idejének és a visszautasított munkák összbüntetésének az összege. Itt a probléma online változatát vizsgáljuk, amelyben a munkák egyenként érkeznek és minden munka esetén a munka érkezésekor el kell döntenünk, hogy a munkát visszautasítjuk-e. Amennyiben a munkát elfogadjuk akkor azon nyomban ütemeznünk kell. Az algoritmusok elemzése során használni fogjuk a következő jelöléseket: tetszőleges $H$ halmazra $B_H = \sum_{i\in H} b_i$, $P_H = \sum_{i\in H} p_i$, $M_H = \max_{i\in H} p_i$, továbbá legyen $\varphi = \frac{1+\sqrt{5}}{2} \approx 1.62$. Az algoritmusok elemzése során használjuk a $P_0$ halmazt, amely azokat a munkákat tartalmazza, amelyekre $b_j \leq p_j/m$, továbbá egy rögzített optimális megoldás esetén $OP$ és $OS$ jelöli az elutasított és az ütemezett munkák halmazát.

Az első vizsgált algoritmus csak a büntetést és a megmunkálási időt veszi figyelembe a visszautasításnál.

**$\mathrm{RP}_\alpha$ Algoritmus:**

- Ha a munkára $b_j \leq \alpha p_j$ teljesül, akkor utasítsuk vissza a munkát, egyébként ütemezzük a LISTA algoritmus szerint. Az algoritmusban $\alpha$ egy pozitív paraméter.

Az algoritmus versenyképességi hányadosát két gépre az alábbi tétel adja meg, amit bizonyítás nélkül közlünk.

**25. tétel.** [7] *Két gép esetén az $RP_{\varphi-1}$ algoritmus $\varphi$ versenyképes.*

Megjegyezzük, hogy az algoritmus nem konstans versenyképes ha a gépek száma tart a végtelenbe. A második algoritmus az általános esetre is jó megoldást szolgáltat.

**$\mathrm{RTP}_\alpha$ Algoritmus**

- Amennyiben a munka a $P_0$ halmaz eleme, akkor utasítsuk el.
- Egyébként legyen $B$ a $P_0$ halmazon kívüli, visszautasított munkák összbüntetése. Ha $B + b_j \leq \alpha p_j$, akkor utasítsuk vissza a munkát, ellenkező esetben ütemezzük a LISTA algoritmus szerint.

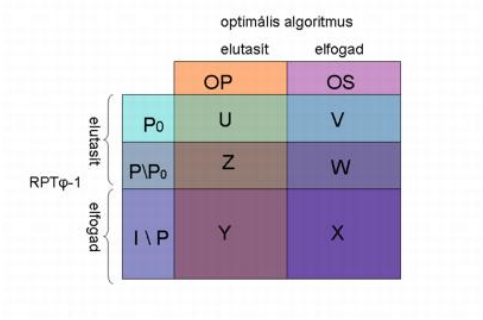
**26. tétel.** [7] *Az $\mathrm{RTP}_{\varphi-1}$ algoritmus $(1+\varphi)$-versenyképes.*

*Bizonyítás:* Vegyünk egy tetszőleges $I$ bemenetet és rögzítsünk egy optimális megoldást. Legyen $\alpha = \varphi - 1$. Legyen $ROPT(I)$ az optimális teljes büntetés és $SOPT(I)$ az optimális ütemezési költség. Legyen $P$ az $\mathrm{RTP}_{\varphi-1}$ algoritmus által elutasított munkák halmaza. Ekkor az algoritmus definíciója alapján kapjuk, hogy $P_0 \subseteq P$. Definiáljuk a következő halmazokat:

$$X = OS \setminus P, \quad Y = OP \setminus P,$$

$$Z = OP \cap (P \setminus P_0), \quad U = OP \cap P_0,$$

$$V = OS \cap P_0, \quad W = OS \cap (P \setminus P_0).$$




7.5. ábra. A felhasznált halmazok

A halmazokat a 7.5 ábra szemlélteti. Állítjuk, hogy ezekre a halmazokra a következő egyenlőtlenségek teljesülnek:

$$(1)\quad B_W \leq \alpha \cdot M_W,$$

$$(2)\quad \alpha \cdot M_Y \leq B_Z + B_W + B_Y.$$

$$(3)\quad B_H \leq \frac{P_H}{m} \text{ ha } B \subseteq P_0$$

$$(4)\quad B_H \geq \frac{P_H}{m} \text{ ha } B \cap P_0 = \emptyset$$

Elsőként igazoljuk (1)-et. Legyen $j$ az utolsó munka a $W$ halmazból. Amikor a munkát visszautasítottuk, akkor $B + b_j \leq \alpha \cdot p_j$ teljesült. Másrészt $j$ az utolsó munka $W$-ből, így a használt $B + b_j$ érték pontosan $B_W$, továbbá $p_j \leq M_W$ nyilvánvalóan teljesül.

Most igazoljuk (2)-t. Legyen $j \in Y$ a maximális méretű munka: $p_j = M_Y$. Amikor a munkát elfogadtuk ütemezésre akkor teljesült $B + b_j > \alpha p_j$. Másrészt a $P_0$-n kívüli visszautasított munkákat a $Z \cup W$ halmaz tartalmazza. Így az algoritmusban vizsgált $B + b_j$ érték legfeljebb $B_Z + B_W + B_Y$, amivel (2)-t igazoltuk.

A (3) és (4) egyenlőtlenségek egyből adódnak a $P_0$ halmaz definíciója alapján.

Mivel a LISTA algoritmust használjuk az ütemezéshez, ezért azt kapjuk, hogy

$$RTP(I) \leq \frac{P_X + P_Y}{m} + M_{X \cup Y} + B_Y + B_U + B_V + B_W.$$

Elsőként tegyük fel, hogy $M_{X \cup Y} = M_X$. Ekkor használva az (1)–(4) egyenlőtlenségeket azt kapjuk, hogy

$$RTP(I) \leq \frac{P_X}{m} + B_Y + M_X + B_Y + B_U + \frac{P_V}{m} + \alpha \cdot M_W \leq$$

$$(1+\alpha)SOPT(I) + 2 \cdot ROPT(I) \leq (1+\varphi)OPT(I).$$

Most tegyük fel, hogy $M_{X \cup Y} = M_Y$. Ekkor használva az (1)–(4) egyenlőtlenségeket azt kapjuk, hogy

$$RTP(I) \leq \frac{P_X}{m} + B_Y + \frac{B_Z + B_Y}{\alpha} + M_W + B_Y + B_U + \frac{P_V}{m} + \alpha \cdot M_W \leq$$

$$(2+\alpha)SOPT(I) + (1+1/\alpha)ROPT(I) \leq (1+\varphi)OPT(I),$$

mert

$$2+\alpha = 1+\varphi \text{ valamint } 1+1/\alpha = 1+\frac{1}{\varphi-1} = 1+\varphi$$

Mivel az összes esetet megvizsgáltuk, ezért az állítást igazoltuk. □

A lehetséges versenyképességi hányadosokra teljesül az alábbi alsó korlát.

**27. tétel.** [7] *Nincs olyan online algoritmus amely jobb, mint c-versenyképes az m-gépes visszautasításos ütemezési feladatra, ahol c az $x^{m-1}+x^{m-2}+\cdots+1=x$. egyenlet megoldása.*

*Bizonyítás:* Tegyük fel, hogy van olyan algoritmus, amelynek kisebb a versenyképességi hányadosa, mint $c$. Vegyük a következő munkasorozatot. Legyen az $i$-edik munka $(1, 1/c^i)$ $i = 1, \ldots, m-1$, (ahol tehát $p_i = 1$ és $b_i = 1/c^i$), és az $m$-edik munka $(1,1)$. Ha az algoritmus az első $m-1$ munka közül valamelyiket elfogadja, akkor a sorozat véget ér. Tegyük fel, hogy a $j$-edik munkát fogadta elsőként. Ekkor az algoritmus költsége $1+\sum_{i=1}^{j-1} 1/c^i$ az optimális költség $\sum_{i=1}^{j} 1/c^i$, így a hányados $c$, ami ellentmondás. Tehát az algoritmus el kell fogadja az első $m-1$ munkát, így függetlenül attól, hogy az utolsó munkát elfogadja -e a költsége $1+\sum_{i=1}^{m-1} 1/c^i$ lesz, míg az optimális költség 1. Tehát a hányados ismét $c$, amivel ismét ellentmondáshoz jutottunk. □

Visszautasításos modelleket vizsgálnak még például a következő dolgozatok is: [19, 20].

## 7.4. A gépköltséges ütemezési feladat

A gépköltséges ütemezési feladatot a [34] cikkben definiálták. Ellentétben a hagyományos ütemezési feladatokkal, amikor a gépek $m$ száma előre adott, és ezekkel a gépekkel kell valahány munkát elvégezni. A gépköltséges ütemezési feladat esetén kezdetben egyetlen gépünk sincs, a gépeket is meg kell vásárolni. A legtöbb vizsgált modellben minden gép vásárlási költsége 1. A feladat online, vagyis egyenként érkeznek az elvégzendő munkák. A legelső munka érkezésekor megvesszük az első gépet.

Ha már úgy látjuk jónak, hogy a következő munkát ne a meglevő egyetlen gépünkre ütemezzük, akkor vehetünk egy második gépet, ismét 1 egységnyi áron, és arra is ütemezhetjük az éppen megérkezett munkát, és így tovább.

A kérdés az, hogy mikor vásároljunk új gépet, másrészt pedig a meglévő gépeken a munkákat hogyan ütemezzük.

Tehát egy-egy új munka érkezésekor döntünk arról, hogy veszünk-e új gépet, és arról is hogy az aktuális munkát melyik gépre ütemezzük. Eddig leginkább azt a modellt vizsgálták, amikor a cél a gépekre kifizetett pénz (vagyis a vásárolt gépek száma), és a teljes átfutási idő összegének minimalizálása.

Az első algoritmus, amit a [34] cikkben mutattak be a feladatra, az a következő. Folyamatosan számoljuk a már megérkezett munkák $S$ összhosszát. Amint ez az $S$ érték eléri a következő négyzetszám értékét, vagyis $S \geq k^2$, ahol $k$ egész szám, akkor megvesszük a $k$-adik gépet, (egyébként nem veszünk új gépet), a munkák ütemezését pedig mindig a LISTA algoritmus szerint végezzük.

Az algoritmus versenyképességi hányadosa az aranymetszés aránya, ami $\varphi = \frac{\sqrt{5}+1}{2} \approx 1.62$. A cikkben továbbá bebizonyították hogy a feladatnak $4/3 \approx 1.333$ alsó korlátja.

Az előbbi algoritmus alapgondolata a következő: Ha sok kicsi méretű munka érkezik, akkor ezeknek egy-egy téglalapot megfeleltetve amelyeknek a szélessége 1, míg a magassága pedig a munka mérete, ezekre akkor kapunk optimális megoldást, ha a gépek száma megközelítőleg egyenlő a teljes átfutási idővel (mert ekkor egy nagy négyzetbe pakolhatók be a kicsi téglalapok). Ugyanis a célfüggvény a gépek száma plusz a teljes átfutási idő, ez a befoglaló téglalap félkerülete (szélessége plusz magassága), adott terület esetén pedig a kerület akkor minimális, ha a téglalap négyzet. Emiatt tehát akkor kell új gépet vásárolni, amikor a kis téglalapok összterülete már nagyobb mint az aktuális gépszám négyzete. Ez az ötlet jól működik akkor, ha kicsi a téglalapok mérete. (Ilyenkor valójában a $4/3$ versenyképességi hányados is elérhető, ami a feladat alsó korlátjával egyenlő, tehát ilyenkor optimális az előbbi algoritmus.)

Ha azonban valamikor egy jókora méretű munka érkezik, akkor az előbbi technika már nem vezet optimális algoritmushoz. A [21] cikk egy olyan algoritmust javasol, amelynek a versenyképességi hányadosa 1.6-nál már némileg kisebb, pontosabban $\left(2\sqrt{6}+3\right)/5 \approx 1.5798$.

Ez az algoritmus a következőképpen működik: Ütemezzük a munkát a jelenlegi gépek egyikére a LISTA algoritmus szerint, ha ezáltal a teljes átfutási idő nem növekszik $2k$ fölé, ahol $k$ a gépek aktuális száma. Egyébként vegyünk új gépet, és arra ütemezzük a következő munkát.

Ez az algoritmus némiképp képes tehát kivédeni azt az esetet, ha utoljára (vagy valamikor) egy hosszabb méretű munka érkezik.

A jelenleg legjobb versenyképességi aránnyal rendelkező algoritmust a [18] cikk közli, ennek versenyképességi hányadosa $\left(2+\sqrt{7}\right)/3 \approx 1.5486$, ami tehát közelítőleg három századdal jobb, de még mindig 1.5 fölött marad. A versenyképesség bizonyítása itt már több mint 6 oldalt igényel. Ez az algoritmus a következő munka ütemezésénél már figyelembe veszi a teljes átfutási idő esetleges növekedését, a munkák összhosszát, valamint ezen kívül még a legnagyobb munka hosszát is.

A versenyképességi hányados felsőkorlátjának bizonyítása sok feladat esetében úgy történik, hogy az algoritmus által kapott célfüggvény-értéket nem közvetlenül az optimális megoldás értékével hasonlítjuk össze, hanem egy, a bemenetre vonatkozó alsó korláttal, $LB(I)$-vel. Ennek oka az, hogy ez utóbbi sok esetben könnyen kiszámítható, míg $OPT(I)$ értékét sok esetben nem ismerjük pontosan. A gépköltséges feladat esetén a két szokásos alsó korlát a következő: $2\sqrt{P}$, ahol $P$ a munkák összmérete, valamint $L+\frac{P}{L}$, ha $L > \sqrt{P}$, ahol $L$ a legnagyobb munkaméret. A gépköltséges feladat esetén tehát $LB(I) = 2\sqrt{P}$, ha $\sqrt{P} \geq L$, és $LB(I) = L+\frac{P}{L}$, ha $L > \sqrt{P}$. A [18] cikkben szerepel annak a bizonyítása is, hogy nincs olyan algoritmus, amelyre ennek az alsó korlátnak a segítségével $ALG(I)/LB(I) < 1.5$ bizonyítha-

tó lenne. Ez nem azt jelenti hogy ennél hatékonyabb algoritmus nem konstruálható (ennek eldöntése még nyitott kérdés), csak azt, hogy ha található olyan algoritmus amelynek a versenyképességi hányadosa 1.5-nél kisebb, akkor ennek bizonyításához további, újfajta alsó korlátok felhasználására lesz szükség.

A gépköltséges feladat alsó és felső korlátja közötti rés a másik oldalról is szűkült, mert a cikk a feladat alsó korlátját $4/3$-ról $\sqrt{2}$-re javítja. Ez a jelenleg ismert legjobb alsó korlát. Vagyis egy optimális algoritmus versenyképességi hányadosa 1.414 és 1.548 között van.

A feladat általánosabb változatával foglalkozik [33], ahol a gépek vásárlásának költsége egy általános költségfüggvénnyel van megadva.

# 8. fejezet

# Ládapakolás és általánosításai

## 8.1. Ládapakolási modellek

A ládapakolási problémában bemenetként tárgyak egy $L$ 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, és a csoportok száma legyen minimális.

A ládapakolási problémák elemzésére a versenyképességi hányados helyett az aszimptotikus versenyképességi hányadost vizsgálják. (A versenyképességi hányadost a legtöbb ládapakolási modellben nem vizsgálják, ha igen akkor abszolút versenyképességi hányadosnak hívják.) Egy A algoritmus aszimptotikus versenyképességi hányadosa ($R_A^\infty$) a következő formulákkal definiálható:

$$R_A^n = \max\{A(L)/OPT(L) \mid OPT(L) = n\}$$

$$R_A^\infty = \limsup_{n\to\infty} R_A^n.$$

Az aszimptotikus hányados fő tulajdonsága az, hogy azt vizsgálja, miként viselkedik az algoritmus akkor, ha a bemenet mérete nő, pontosabban ha az optimális költség végtelenhez tart. Ez azt jelenti, hogy az algoritmus szabadon helyezheti el a kezdeti elemeket a listáról.

A fentiekben definiált ládapakolási problémának számos változata, általánosítása van. Az alábbiakban megemlítünk néhányat. A ládapakolási probléma három különböző módon általánosítható több dimenzióra. Az első általánosítás a vektorpakolás, amelyben a tárgyak $d$-dimenziós vektorok és úgy kell elpakolni őket, hogy minden ládában minden koordinátára az ott szereplő értékek összege legfeljebb 1 legyen. A második általánosítás a dobozpakolás, ahol többdimenziós dobozokat kell pakolni többdimenziós egységládákba. Végül a harmadik általánosítás a sávpakolás, ahol egy egységnyi széles sávba pakolunk tárgyakat és célunk a szükséges magasság minimalizálása. Fontos még megemlítenünk a ládafedési problémát, amelyben célunk, hogy a tárgyakkal a lehető legtöbb ládát töltsük legalább egységnyi méretűre. Ezeket a modelleket itt nem tárgyaljuk, részletek találhatók a [15] áttekintő dolgozatban.

## 8.2. Az NF algoritmus, helykorlátos algoritmusok

Érdemes még megemlítenünk azt a modellt, amelyben az egyszerre 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 az egyetlen algoritmus, amely használható, a következő a [37, 39] dolgozatokban bemutatott és elemzett NF algoritmus.

**NF algoritmus:** Amennyiben a tárgy elfér a nyitott ládában tegyük oda! Ellenkező esetben zárjuk be a nyitott ládát, nyissunk egy új ládát és tegyük abba a tárgyat!

**28. tétel.** [37, 39] *Az* NF *algoritmus aszimptotikus versenyképességi hányadosa 2.*

*Bizonyítás:* Vegyünk egy tetszőleges $\sigma$ tárgysorozatot. Jelölje $n$ az NF algoritmus által használt ládák számát, továbbá legyen $S_i$, $i = 1,\dots,n$ az $i$-edik ládában levő tárgyak méreteinek összege. Ekkor $S_i + S_{i+1} \geq 1$, 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 legalább $\lfloor n/2 \rfloor$. Másrészt az optimális offline algoritmus sem rakhat 1-nél több összméretű tárgyakat egy ládába, így azt kapjuk, hogy $OPT(\sigma) \geq \lfloor n/2 \rfloor$. Ez azt jelenti, hogy

$$\frac{NF(\sigma)}{OPT(\sigma)} \leq \frac{n}{\lfloor n/2 \rfloor} \leq 2 + 1/\lfloor n/2 \rfloor.$$

Másrészt ha $n$ tart végtelenbe, akkor $1/\lfloor n/2 \rfloor$ tart a 0-hoz, amivel igazoltuk, hogy az algoritmus aszimptotikusan 2-versenyképes.

Most megmutatjuk, hogy az algoritmus aszimptotikus versenyképességi hányadosa legalább 2. Ehhez tekintsük minden $n$-re a következő $\sigma_n$ sorozatot. A sorozat $4n$ tárgyból áll, a $(2i-1)$-edik tárgy mérete $1/2$, a $2i$-edik tárgy mérete $1/2n$, ahol $i = 1,\dots,2n$. Ekkor az NF algoritmus az $i$-edik ládába a $(2i-1)$-edik és a $(2i)$-edik tárgyat teszi, és $NF(\sigma_n) = 2n$. Az optimális algoritmus az $1/2$ méretű tárgyakat párosítja, és a kis tárgyakat egy további ládába teszi, így $OPT(\sigma_n) = n+1$. Mivel $NF(\sigma_n)/OPT(\sigma_n) = 2 - 2/(n+1)$ a 2 értékhez konvergál, ha $n$ tart végtelenbe, ezért igazoltuk, hogy az algoritmus aszimptotikus versenyképességi hányadosa legalább 2. Fontosnak tartjuk megjegyezni, hogy valójában azt is igazoltuk, hogy az algoritmus abszolút versenyképessége is 2. □

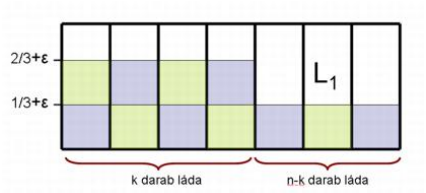
Itt érdemes megemlítenünk, hogy amennyiben egynél több (de korlátozott számú) láda lehet nyitva, az NF algoritmusnál jobb algoritmusok is ismertek. A jelenlegi legjobb algoritmusok a harmonikus algoritmusok családjába tartoznak, ahol az alapötlet az, hogy a $(0,1]$ intervallumot részintervallumokra osztjuk, 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.

## 8.3. Alsó korlátok online algoritmusokra

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.

**29. tétel.** *Nincs olyan online algoritmus a ládapakolási problémára, amelynek az aszimptotikus versenyképességi hányadosa kisebb, mint* $4/3$.

*Bizonyítás:* Legyen A egy tetszőleges online algoritmus. Tekintsük tárgyaknak a 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. Az $L_1$ lista pakolását a 8.1 ábrán szemléltetjük.




8.1. ábra. Az bizonyításban használt pakolás

Ekkor az algoritmus költsége $A(L_1) = k + (n-2k) = n-k$. Másrészt az optimális offline 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 online 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, a többihez mindhez új ládát kell nyitnia. Tehát $A(L_1L_2) \geq n-k+(n-(n-2k)) = n+k$. Mászrészt az optimális offline 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 online 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 pakolási minták módszere

A fenti bizonyítás alapötlete, hogy egy hosszabb tárgysorozatot (a fentiekben $L_1L_2$) veszünk és az algoritmus viselkedésétől függően választjuk ki a sorozatnak azt a kezdőszeletét,

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 $L = L_1 L_2 \dots L_k$, ahol $L_i$ $n_i = \alpha_i n$ egyforma méretű tárgyat tartalmaz, amelyek mérete $a_i$. Amennyiben egy $A$ algoritmus $C$-versenyképességű, akkor minden $j$-re teljesülnie kell a

$$C \geq \limsup_{n\to\infty} \frac{A(L_1 \dots L_j)}{OPT(L_1 \dots L_j)}$$

feltételnek. A fentiekben tekinthetjük azt az algoritmust, amelyre az általunk adható alsó korlát minimális, így célunk az

$$R = \min_A \max_{j=1,\dots,k} \limsup_{n\to\infty} \frac{A(L_1 \dots L_j)}{OPT(L_1 \dots L_j)}$$

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 részlistákból hány elemet tartalmaz a láda. A *pakolási minta* egy $k$-dimenziós vektor $(p_1, \dots, p_k)$, 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 $A$ értéket a következő vegyes egészértékű programozási feladat megoldásával számíthatjuk ki.

**Min** $R$

$$\sum_{p\in T} p_j n(p) = n_j, \qquad 1 \le j \le k$$
$$\sum_{i=1}^{j}\sum_{p\in T_i} n_p \le R\cdot OPT(L_1\dots L_j), \qquad 1 \le j \le k$$
$$n(p) \in \{0,1,\dots\}, \qquad p \in T$$

Az első $k$ feltétel azt írja le, hogy az összes tárgyat el kell helyeznünk a ládákban. A második $k$ feltétel az írja le, hogy az $R$ érték valóban nem kisebb, mint az algoritmus költségének és az optimális költségnek a hányadosa a vizsgált részlistákra. Az $L_1L_2\dots L_k$ lista alapján a pakolási minták $T$ halmaza és az optimális $OPT(L_1\dots L_j)$ értékek meghatározhatók.

A problémában a változók igen nagy értékeket vehetnek fel és a változók száma is nagy lehet, ezért a probléma helyett a lineáris programozási relaxációt szokás tekinteni. Továbbá a megoldást azon feltétel mellett kell meghatároznunk, hogy $n$ tart a végtelenbe, és belátható, hogy ezen feltétel mellett az egészértékű feladat és a relaxáció ugyanazokat a korlátokat adják.

Az eljárást megfelelően választott listákra alkalmazva kapták meg azt az alsó korlátot, amely azt mondja ki, hogy nincs olyan online algoritmus, amelynek kisebb a versenyképességi hányadosa, mint 1.5401. Ezen tétel részletes bizonyítása, a módszer részletesebb tárgyalása egyéb ládapakolási alkalmazásokkal együtt megtalálható a [51] doktori disszertációban. A módszer egy továbbfejlesztését mutatja be a [9] dolgozat.

## 8.4. Az FF algoritmus, és a súlyfüggvény technika

Ebben a részben egy módszert mutatunk be, amelyet gyakran használnak a ládapakolási algoritmusok elemzése során. A módszert az FF (First Fit) algoritmuson ismertetjük. Az FF algoritmus az NF algoritmus továbbfejlesztett változata, arra az esetre, amelyben nincs korlátozva a nyitott ládák száma.

**FF algoritmus:** A tárgyat a legkorábban kinyitott ládába tesszük ahol elfér. Ha nem fér el egyik ládában sem, kinyitunk egy új ládát és abba rakjuk.

Itt definiáljuk a hasonló elven működő BF (Best Fit) algoritmust.

**BF algoritmus:** A tárgyat a legnagyobb össztöltéssel rendelkező ládába tesszük ahol elfér. Ha nem fér el egyik ládában sem, kinyitunk egy új ládát és abba rakjuk.

A FF algoritmusra teljesül a következő állítás.

**30. tétel.** [40] *Az* FF *algoritmus aszimptotikus versenyképességi hányadosa 1.7.*

*Bizonyítás:* Mivel ezen rész fő célja a súlyfüggvény technika bemutatása, ezért pusztán azzal a résszel foglalkozunk, amely azt igazolja, hogy az algoritmus aszimptotikusan 1.7-versenyképes. A korlát élességének bizonyítása megtalálható a [40] dolgozatban. 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 mind az offline és az online célfüggvények értékét. Definiáljuk a következő súlyfüggvényt:

$$w(x) = \begin{cases} 6x/5 & 0 \le x \le 1/6 \\ 9x/5 - 1/10 & 1/6 \le x \le 1/3 \\ 6x/5 + 1/10 & 1/3 \le x \le 1/2 \\ 6x/5 + 2/5 & 1/2 < x. \end{cases}$$

A tárgyak egy tetszőleges $H$ halmazára legyen $w(H) = \sum_{i \in H} w(a_i)$. Ekkor a súlyfüggvényre teljesülnek az alábbi állítások.

**5. lemma.** *Amennyiben tárgyak egy $H$ halmazára teljesül, hogy $\sum_{i \in H} a_i \le 1$, akkor ezen tárgyakra $w(H) \le 17/10$.*

**6. lemma.** *Tárgyak tetszőleges $L$ listájára $w(L) > FF(L) - 2$.*

*Bizonyítás:* Mindkét lemma bizonyítása azon alapul, hogy eseteket különböztetünk meg a tárgyak méretétől függően. A bizonyítások hosszúak és sok technikai részletet tartalmaznak, ezért a jelen jegyzetben eltekintünk a bemutatásuktól. Az érdeklődő olvasó megtalálhatja a részleteket a [40] cikkben.

A lemmák alapján könnyen igazolható, hogy az algoritmus aszimptotikusan 1.7 versenyképes. Tekintsük tárgyaknak egy tetszőleges $L$ listáját. Mivel az optimális offline algoritmus el tudja pakolni a lista elemeit $OPT(L)$ ládába úgy, hogy minden ládába a tárgyak méreteinek összege legfeljebb 1, ezért az első lemma alapján $w(L) \le \frac{17}{10} OPT(L)$. Másrészt a második lemma alapján $FF(L) - 2 \le w(L)$, így azt kapjuk, hogy $FF(L) \le \frac{17}{10} OPT(L) + 2$, amiből következik, hogy az algoritmus 1.7-versenyképes. Valójában az $FF(L)/OPT(L)$ hányadosra pontosabb becslések is ismertek. Ha $OPT(L) = n$, akkor a hányados a $(\lfloor 1.7n \rfloor)/n$ és $(\lceil 1.7n \rceil)/n$ értékek közé esik, ezen eredmények részletei megtalálhatóak a [38] dolgozatban. □

Érdemes megjegyeznünk, hogy számos az $FF$ algoritmusnál jobb algoritmus került kifejlesztésre. A jelenleg ismert legjobb algoritmus aszimptotikus versenyképességi hányadosa 1.5888.

## 8.5. Többdimenziós változatok

### 8.5.1. Online 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 online 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 online 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 választani 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 elhelyezzük rajta.

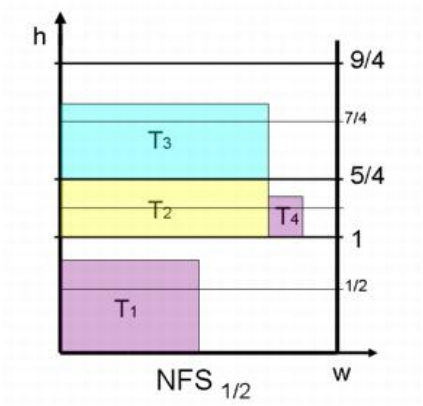
Csak egy eljárást vizsgálunk részletesen a fenti feladat megoldására. Ezt az algoritmust, amit $\mathrm{NFS}_r$ algoritmusnak neveznek, a [6] cikkben mutatták be. Az algoritmus egy $r < 1$ paramétertől függ. Az algoritmus minden $j$-re legfeljebb egy $r^j$ magasságú aktív polcot tart fent és egy tárgy 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 rajta. 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).

**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. A kapott megoldást szemlélteti a 8.2 ábra.

A $\mathrm{NFS}_r$ algoritmus versenyképességére igazak az alábbi állítások.

**31. tétel.** [6] *Az* $\mathrm{NFS}_r$ *algoritmus* $\left(\frac{2}{r} + \frac{1}{r(1-r)}\right)$*-versenyképes. Az* $\mathrm{NFS}_r$ *algoritmus aszimptotikusan* $(2/r)$*-versenyképes.*




8.2. ábra. Az NFS algoritmus által kapott megoldás

*Bizonyítás:* Tekintsük téglalapok tetszőleges $L$ listáját, és jelölje $H$ a legmagasabb téglalap magasságát. Mivel ekkor $OPT(L) \geq H$ teljesül, ezért a tétel állításának igazolásához elegendő belátnunk, hogy erre a sorozatra

$$NFS_r(L) \leq \frac{2}{r} OPT(L) + \frac{H}{r(1-r)}.$$

Jelölje $H_A$ az $L$ lista végén az aktív polcok összmagasságát, $H_Z$ pedig a többi, lezárt polcok 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 téglalapnak az összterülete legalább $\sum_{i=0}^{\infty} wn_ihr^{i+1}/2$, így $OPT(L) \geq \sum_{i=0}^{\infty} n_ihr^{i+1}/2$. Másrészt a lezárt polcok összmagassága $H_Z = \sum_{i=0}^{\infty} n_ihr^i$, így azt kaptuk, hogy $H_Z \leq 2OPT(L)/r$. Mivel $NFS_r(L) = H_A + H_Z$, 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 a [14] cikkben publikálták, amely algoritmus a harmonikus ládapakolási algoritmust használja az adott polctípusokhoz rendelt tárgyak elhelyezésére. Ezen algoritmusnak az aszimptotikus versenyképességi hányadosa tetszőlegesen megközelíti a 1.69103 értéket, és azt is igazolták, hogy ennél kisebb versenyképességű polc algoritmus nem létezik.

### 8.5.2. Online sávpakolás nyújtható tárgyakkal

Amennyiben a sávpakolási feladattal azt az erőforrás allokációs problémát modellezzük, amelyben a tárgyak szélessége az erőforrásigény a tárgyak magassága pedig az idő, akkor használhatjuk azt a változatot, amelyben a tárgyak mérete nem rögzített, hanem az egyes tárgyakat megnyújthatjuk a területüket változatlanul hagyva (ld. [29]). Egész pontosan egy-egy téglalapot függőleges irányban meg szabad nyújtani a területét változatlanul hagyva, de szélesíteni nem szabad.

Ebben a részben az egyszerűbb tárgyalás érdekében feltesszük, hogy a sáv szélessége $w = 1$. Egyszerűen látható, hogy ebben az esetben az offline probléma könnyen megoldható. Az offline optimum értéke $OPT(L) = \max\{S, H\}$, ahol $S = \sum_{i \in L} w_i \cdot h_i$ a teljes terület, és and $H = \max_{i \in L} h_i$ a maximális magasság. Az aszimptotikus versenyképességi hányadost használva tetszőlegesen közel juthatunk az 1 értékhez ha nagyon nagy additív konstanst engedünk meg az algoritmus elemzésében, következésképpen ebben az esetben az abszolút versenyképességi hányadost érdemes használni. A feladat megoldására egyből adódik az $\mathrm{NFS}_r$ algoritmus következő módosítása, amely algoritmust $\mathrm{MNFS}_r$-nek nevezzünk.

A $p_i = (w_i, h_i)$ téglalap érkezése után válasszunk egy olyan $k$ értéket, amelyre teljesül, hogy $r^{k+1} < h_i \leq r^k$. Nyújtsuk meg a téglalapot $r^k$ magasságúra. Amennyiben van $r^k$ magasságú aktív polc, és az új 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).

Az így kapott algoritmusra igaz a következő állítás.

**32. tétel.** [29] *Az* $\mathrm{MNFS}_r$ *algoritmus* $\left(2 + \frac{1}{r(1-r)}\right)$*-versenyképes a nyújtható téglalapok modelljében.*

*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

$$\mathrm{NFS}_r(L) \leq 2\mathrm{OPT}(L) + H/r(1-r).$$

Jelölje $H_A$ az $L$ lista végén az aktív polcok összmagasságát, $H_Z$ pedig a többi lezárt polcok 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 1. Továbbá a téglalapokat megnyújtjuk mielőtt az aktuális polcra raknánk, így minden téglalap magassága pontosan $hr^i$. Tehát párosítva a lezárt polcokat és használva az aktív $hr^i$ magasságú polcot is, ha a lezárt polcok száma páratlan (ha a lezárt polcok száma páros nincs az aktív polcra szükség), azt kapjuk, hogy az ilyen polcokon elhelyezett tárgyak összterülete legalább $n_i hr^i/2$.

Következésképpen az összes téglalapnak az összterülete legalább $\sum_{i=0}^{\infty} n_i hr^i/2$, így $\mathrm{OPT}(L) \geq \sum_{i=0}^{\infty} n_i hr^i/2$. Másrészt a lezárt polcok összmagassága $H_Z = \sum_{i=0}^{\infty} n_i hr^i$, így azt kaptuk, hogy $H_Z \leq 2\mathrm{OPT}(L)$. Mivel $\mathrm{NFS}_r(L) = H_A + H_Z$, ezért a fentiek alapján adódik a kívánt egyenlőtlenség. □

Jegyezzük meg, hogy $r(1-r) \leq 1/4$ miatt az előbbi algoritmus versenyképességi hányadosa legalább 6. Ennél jobb versenyképességi hányadossal rendelkezik a következő DS algoritmus.

**DS ALGORITMUS**

Az első téglalap érkezése után vegyünk egy $h_1$ magas polcot a sáv legalján és legyen ez az aktív polc. A további elemeket pakoljuk az alábbi szabály szerint:

*1. lépés* Ha lehetséges a téglalapot berakni az aktív polcra, azt követően, hogy megnyújtjuk az aktív polc magasságára, akkor nyújtsuk meg és rakjuk az aktív polcra annyira balra, amennyire lehetséges. Egyébként menjünk a 2. lépésre.

*2. lépés* Vegyünk egy új aktív polcot, amely kétszer olyan magas, mint az előző, és tegyük az eddigi polcok tetejére. Lépjünk az 1. lépésre.

Az algoritmusra teljesül a következő állítás.

**33. tétel.** *A* DS *algoritmus versenyképességi hányadosa* 4.

*Bizonyítás:* Elsőként igazoljuk, hogy az algoritmus 4 versenyképes. Vegyük a téglalapoknak egy tetszőleges $L$ listáját. Legyen $H$ az utolsó aktív polc magassága. Amikor az algoritmus ezt a polcot megkonstruálta, akkor az aktuális téglalap nem fért el az előző $H/2$ magas aktív polcon, következésképpen $H \leq 2 \cdot OPT(L)$. Másrészt a használt polcmagasságok $H, H/2, H/4, \ldots, H/2^i$ valamely $i$-re, így a teljes felhasznált sávmagasság legfeljebb $2H \leq 4 \cdot OPT(L)$.

A korlát élességének igazolásához vegyük a következő $L_i$ listát. Az első $i$ darab téglalap legyen $(1/4, 2^j)$, $j = 1, \ldots, i$, az utolsó pedig legyen $(1/4, 2^i + 1)$. Végrehajtva az algoritmust, és megkonstruálva az optimális offline megoldást azt kapjuk, hogy $DS(L_i)/OPT(L_i) = 2^{i+2}/(2^i+1)$, ami tart 4-hez, amint $i$ tart a végtelenbe. □

# 9. fejezet

# Problémák a számítógépes hálózatokban

## 9.1. Nyugtázás

A számítógépes hálózatok kommunikációja során a küldő a címzettnek adatcsomagokat küld. Amennyiben a kommunikációs csatorna nem teljesen megbízható (pl. vezeték nélküli), akkor lényeges a csomagok megérkezéséről a címzettnek nyugtát küldeni, így lehetővé tehető az elveszett adatcsomagok újraküldése. A nyugtázási probléma 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 a [16] dolgozatban fejlesztették ki 2001-ben. Az alábbiakban ismertetjük a kifejlesztett modell lényegét és az alapvető eredményeket.

A nyugtázási probléma matematikai modelljében a bemenetet a csomagok $a_1, \ldots, a_n$ érkezési idejei adják. Az algoritmusnak meg kell határoznia, hogy mikor küld nyugtákat, ezeket az időpontokat $t_1, \ldots, t_k$ jelöli. A nyugtázási probléma költségfüggvénye:

$$k + \sum_{j=1}^{k} \nu_j,$$

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 probléma online, azaz egy adott $t$ időpontban csak a $t$-ig megérkezett csomagok érkezési idejeit ismerjük és nincs semmi információnk a további csomagokról.

A probléma 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 az $a_{j+1}$ érkezésekor 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:

$$1 = |\sigma_j| e_j + \sum_{a_i \in \sigma_j} (a_j - a_i).$$

ahol $\sigma_j$ az $a_j$ csomag érkezése után meglevő még nyugtázatlan csomagok halmaza.

Az algoritmus versenyképességére teljesül a következő állítás.

**34. tétel.** [16] *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 offline algoritmus által küldött nyugták száma. Ha $k^* \geq k$, akkor $\mathrm{OPT}(I) \geq k$ és adódik, hogy az algoritmus valóban 2-versenyképes. Amennyiben $k^* < k$, akkor az ÉBRESZT algoritmus nyugtái által meghatározott $k$ közül legalább $k - k^*$ intervallumban OPT-nak nincs nyugtája, ez legalább $k - k^*$ késedelemből származó költséget jelent OPT számára, így ismét $OPT(I) \geq k$ 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.

**35. tétel.** [16] *Nem létezik olyan online algoritmus az online nyugtázási problémára, amelynek kisebb a versenyképességi hányadosa, mint 2.*

*Bizonyítás:* Vegyünk egy tetszőleges online 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, de addig nem jött új csomag! A számítások egyszerűsítéséhez feltesszük, hogy az új csomag érkezéséig eltelő nagyon rövid idő 0, ennek ellenére az új csomagot a nyugta nem nyugtázza. (Ez a feltétel elkerülhető lenne kicsi $\varepsilon > 0$ értékek használatával.) Ekkor egy $2n$ csomagból álló sorozat esetén az online algoritmus költsége $ONL(I_{2n}) = 2n + t_{2n}$, 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

$$EV(I_{2n}) = n + \sum_{i=0}^{n-1} (t_{2i+1} - t_{2i}) + 1,$$

és

$$ODD(I_{2n}) = n + \sum_{i=1}^{n} (t_{2i} - t_{2i-1}).$$

Innen következik, hogy $EV(I_{2n}) + ODD(I_{2n}) = ONL(I_{2n}) + 1$. Másrészt az optimális offline algoritmusra $OPT(I_{2n}) \leq \min\{EV(I_{2n}), ODD(I_{2n})\}$, így azt kapjuk, hogy $ONL(I_{2n})/OPT(I_{2n}) \geq 2 - 1/OPT(I_{2n})$. Ezen egyenlőtlenség alapján adódik, hogy ONL nem lehet jobb, mint 2-versenyképes, hiszen a csomagok egy elegendően hosszú sorozatát véve az $OPT(I_{2n})$ érték tetszőlegesen nagy lehet. □

A fenti állítások mutatják, hogy az ÉBRESZT algoritmus a lehető legkisebb versenyképességi hányadossal rendelkezik. Ezt az okozza, hogy az ébresztőt arra az értékre állítjuk be, amely minimalizálja a versenyképességi hányadost. Felmerül a kérdés, hogy más értékek nem adnának- e jobb eredményt az átlagos esetben, valós adatokon. A következő, a [32] dolgozatban bemutatott algoritmus megpróbálja megtanulni az ébresztő beállításának az optimális értékét.

Az algoritmus fázisokban dolgozik, minden fázis k nyugtáig tart. Minden fázisban egy ébresztő típusú algoritmust használunk. Az ébresztőt úgy állítjuk be, hogy az ébresztő időpontjában a teljes késedelem $p$ legyen, jelölje ezt az algoritmus $\text{ÉBRESZT}_p$. Az első fázisban $p = 1$, azaz az ÉBRESZT algoritmust használjuk, utána $p$-t mindig arra az értékre állítjuk be, amelyik a legjobb megoldást adja az utolsó $10k$ csomagra nézve.

Az optimális $p$ érték meghatározásához az alábbi lemma ad segítséget.

**7. lemma.** [32] *Az optimális p értékre teljesül, hogy van olyan nyugta, amelyet közvetlenül valamely csomag érkezésekor küldünk.*

*Bizonyítás:* Vegyünk egy tetszőleges $I$ bemeneti sorozatot és tekintsünk egy olyan $p$ értéket, amelyre nem teljesül a lemmában megadott tulajdonság. Jelölje $k$ az $\text{ÉBRESZT}_p$ algoritmus által küldött nyugták számát. Az algoritmus definíciója alapján adódik, hogy minden nyugta teljes késedelme $p$, így a teljes költség $k(1+p)$.

Mivel egyetlen nyugtát sem küldtünk valamely csomag érkezési időpontjában, ezért létezik olyan kicsi $\varepsilon > 0$, hogy az $\text{ÉBRESZT}_{p-\varepsilon}$ algoritmus esetén minden nyugta pontosan azokat a csomagot nyugtázza, mint az $\text{ÉBRESZT}_p$ algoritmusnál. Másrészt ezen algoritmus esetén a teljes költség $k(1+p-\varepsilon)$, így $p$ nem lehetett optimális. □

Tehát az alábbi algoritmus meghatározza az optimális p értéket.

**OPTKeres Algoritmus**

- *Inicializálás* Legyen $p^* = 1$ és $Z = \infty$.
- *Keresés* Minden $1 \leq i < j \leq n$ esetén
  - Legyen $p := \sum_{k=i}^{j} (a_j - a_i)$.
  - Hajtsuk végre az $\text{ÉBRESZT}_p$ algoritmust a bemeneten, legyen a nyugták száma $k$.

– Ha $k(1+p) < Z$ akkor $p^* := p$ és $Z := k(1+p)$.

- Return $p^*$

A keresés $\Theta(n^2)$ iterációt hajt végre és minden iterációban a második lépés időigénye $\Theta(n)$ a többi konstans, tehát az algoritmus időigénye $\Theta(n^3)$.

A tanuló algoritmus majdnem 20 százalékkal kisebb költséget adott valós bemeneteken végrehajtott tesztekben (ld. [32]), mint az ÉBRESZT algoritmus.

## 9.2. A lapletöltési probléma

A már bemutatott lapozás problémájának az általánosítása a lapletöltési probléma. A világhálón a böngészők is használnak egy memóriát, amelyben a letöltött lapokat tárolják, hogy amennyiben egy oldalt rövid időn belül többször meg akar nézni a felhasználó, akkor 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 probléma a megfelelő cserélési stratégiák megkeresése. A különbség a lapozás problémájához képest, hogy nem minden lap mérete egyforma, továbbá az egyes lapok letöltési költsége is különböző. Tehát a problémát a következőképpen fogalmazhatjuk meg.

Adott egy $k$ méretű memória. A bemenet a letöltendő lapok sorozata. Minden $p$ lapnak van egy *lapmérete* $s(p)$ és egy *letöltési költsége* $c(p)$. 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 probléma online, 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 probléma és a speciális eseteinek megoldására több eljárást is javasoltak. Az alábbiakban a Young által kifejlesztett (ld. [53]) $k$-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 $H$ jelöli. Amennyiben egy $g$ fájlt kell letöltenünk, a következő lépéseket hajtjuk végre:

**Háziúr**$(H,g)$

**ha** $g$ nincs a memóriában
**akkor** amíg nincs elég hely
{legyen $\Delta = \min_{f \in H} cr(f)/s(f)$
legyen minden $f \in H$-ra $cr(f) = cr(f) - \Delta \cdot s(f)$
tegyünk ki olyan lapokat, akikre $cr(f) = 0$}

tegyük be $g$-t a $H$ memóriába, legyen $cr(g) = c(g)$

**egyébként** (ha benne volt) állítsuk át $cr(g)$ értékét valamelyik $cr(g)$ és $c(g)$ közötti értékre

Az algoritmus enyhén $k$-versenyképes, de ennél erősebb állítás is igaz. A lapletöltési problémára egy ALG online algoritmust enyhén $C$ versenyképesnek nevezünk a $(k,h)$ erőforrás kiterjesztéses modellben, ha van olyan $B$ konstans, hogy $ALG_k(I) \leq C \cdot OPT_h(I) + B$ minden bemenetre teljesül, ahol $ALG_k(I)$ az algoritmus által elvégzett összes letöltés költsége $k$ méretű memória mellett, $OPT_h(I)$ pedig a minimális elérhető teljes letöltési összeg $h$-méretű memória mellett. A HÁZIÚR algoritmusra teljesül a következő állítás.

**36. tétel.** [53] *A* HÁZIÚR *algoritmus $k/(k-h+1)$-versenyképes a lapletöltési problémára, a $(k,h)$ erőforrás kiterjesztéses modellben.*

*Bizonyítás:* Tekintsük kérések egy tetszőleges sorozatát, jelölje ezt a bemenetet $I$. 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 offline algoritmus memóriájában aktuálisan szereplő lapjainak halmazát $OPT$. Tekintsük a következő potenciálfüggvényt.

$$\Phi = (h-1)\sum_{f\in H} cr(f) + k\sum_{f\in OPT}(c(f) - cr(f)).$$

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 $f \in H$-ra a $cr(f)$ értéket.

  Ekkor minden $f \in H$ esetén a $cr(f)$ érték csökkenése $\Delta \cdot s(f)$, így összességében $\Phi$ a

  $$\Delta((h-1)s(H) - ks(OPT \cap H))$$

  értékkel csökken, ahol $s(H)$ és $s(OPT \cap H)$ rendre a $H$ illetve az $OPT \cap H$ halmazokban levő lapoknak a méreteinek az összege. Abban az időpontban mikor ez a lépés bekövetkezik OPT már elhelyezte a $g$ lapot $OPT$-ban de a lap még nincs a $H$ halmazban. Következésképp $s(OPT \cap H) \leq h - s(g)$. Másrészt ez a lépés azért hajtódik végre, mert nem fért el a $g$ lap a $H$ memóriában, tehát $s(H) > k - s(g)$, és így, mivel a lapok méretei egészek, ezért $s(H) \geq k - s(g) + 1$. Következésképpen azt kapjuk, hogy a $\Phi$ 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 $H$ memóriából.

  Mivel HÁZIÚR csak akkor rak ki egy $f$ lapot a memóriából, ha arra $cr(f) = 0$, ezért ebben a részben nem változik $\Phi$.

- HÁZIÚR elhelyezi a $g$ lapot a $H$ memóriában és beállítja a $cr(g) = c(g)$ értéket.

  Ekkor HÁZIÚR költsége $c(g)$. Másrészt $g$ nem volt eddig benne a $H$ memóriában így $cr(g) = 0$ teljesült. Továbbá elsőként OPT helyezte el a lapot, így $g \in OPT$ teljesül. Következésképpen a $\Phi$ függvény értékének csökkenése ebben a lépésben $-(h-1)c(g)+kc(g) = (k-h+1)c(g)$.

- HÁZIÚR átállítja a $g \in H$ 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 $\Phi$ nem növekedhet.

Végignéztük az algoritmusok lehetséges lépéseit és a $\Phi$ 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 $\Phi$ értéke $(k-h+1)$-szer annyival csökken, mint amennyi az algoritmusnál fellépő költség.
- A többi esetben $\Phi$ 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 $\Phi(v)$ és $\Phi(0)$ a potenciálfüggvény kezdeti és végső értékei. Mivel a potenciálfüggvény nemnegatív, ezért adódik, hogy $(k-h+1)\mathrm{HÁZIÚR}_k(I) \leq kOPT_h(I) + \Phi_0$, azaz azt kapjuk, hogy a HÁZIÚR algoritmus enyhén $k/(k-h+1)$ versenyképes a $(k,h)$ erőforrás kiterjesztéses modellben. □

Megjegyezzük, hogy ennél kisebb versenyképességű hányados nem érhető el, még a legegyszerűbb speciális esetben sem, amint azt az alábbi tétel mutatja.

**37. tétel.** [44] *Ha minden lap mérete és letöltési költsége 1, akkor nincs olyan online algoritmus, amelynek a $(k,h)$ erőforrás kiterjesztéses modellben a versenyképességi hányadosa kisebb, mint $k/(k-h+1)$.*

## 9.3. Online forgalomirányítás

A számítógépes hálózatok esetén az egyes kommunikációs csatornák túltelítettsége a számítógépes forgalom nagymértékű lassulásához és információk elvesztéséhez vezethet. Ezért a számítógépes hálózatok elméletének egyik legalapvetőbb feladata a forgalom szabályozása. A forgalom szabályozásának témakörébe tartozik a forgalomirányítás is, mely 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 online problémáról van szó. Az alábbiakban a forgalomirányítás problémájának két online modelljét mutatjuk be, amely modellek nem csak a számítógépes hálózatok modellezésére alkalmasak, hanem általánosabb forgalomirányítási problémák tanulmányozására is.

**A matematikai modellek**

A hálózatot egy gráf reprezentálja 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 online, 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 rendelt 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.

Az alábbiakban ismertetjük az exponenciális algoritmust, amelyet a [4] cikkben mutattak be. 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 minden $e$ él esetén 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.

**EXP algoritmus a nyereség modellre:**

Egy $j$ kérés elbírálása:

1. Legyen $c_e(j) = \mu^{l_e(j)}$, ahol $\mu$ egy a feladat paramétereitől függő konstans.
2. Vegyük azt a $P_j$ utat $s_j$ és $t_j$ között, amelyre

$$C(P_j) = \sum_{e \in P_j} \frac{r_j}{u(e)} c_e(j)$$

minimális.

3. Ha $C(P_j) \leq 2mb_j$, akkor elfogadjuk a kérést és $P_j$-n foglaljuk le a sávszélességet, ha nem, akkor visszautasítjuk.

**Megjegyzés:** Amennyiben az algoritmusból csak az 1 és 2 lépéseket hajtjuk végre és minden kérést elfogadunk, akkor a terhelést kiegyensúlyozó modellre kapjuk meg az exponenciális algoritmust.

Az algoritmus versenyképességére ad korlátot az alábbi tétel. Megjegyezzük, hogy mivel az eddigi példákkal ellentétben itt maximalizálási feladatról van szó, ezért a versenyképességet egy 1-nél kisebb szám adja meg, és minél nagyobb ez az érték annál jobbnak tekintjük az algoritmust.

**38. tétel.** [4] *Az* EXP *algoritmus* $1/O(\log\mu)$*-versenyképes, ha* $\mu = 4mPB$*, ahol* $B$ *felső korlátja a nyereségeknek, továbbá minden kérésre és élre teljesül*

$$\frac{1}{P} \leq \frac{r(j)}{u(e)} \leq \frac{1}{\log_2 \mu}.$$

*Bizonyítás:* 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 hasonlón 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.

A tétel állítását három lemmán keresztül igazoljuk. A lemmák közül csak az első lemmát bizonyítjuk be, a másik két lemma bizonyítását az olvasó megtalálhatja a [4] cikkben.

**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 $r_j/u(f) \leq 1/\log_2\mu$ és a $j$ kérés elfogadása után az $f$ élen túllépjük a megengedett sávszélességet, ezért adódik, hogy $l_f(j) > 1 - 1/\log_2\mu$. Másrészt ekkor az EXP algoritmus második lépésében számolt $C(P_j)$ értékre

$$C(P_j) = \sum_{e\in P_j} \frac{r_j}{u(e)} c_e(j) \geq \frac{r_j}{u(f)} c_f(j) \geq \frac{r_j}{u(f)} \mu^{1-1/\log_2\mu}.$$

Szintén a tétel feltételei alapján $\frac{r_j}{u(e)} \geq \frac{1}{P}$, továbbá $\mu^{1-1/\log_2 m} = \mu/2$, így a fenti egyenlőtlenség alapján azt kapjuk, hogy

$$C(P) \geq \frac{1}{P}\frac{\mu}{2} = 2mB.$$

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. □

**9. lemma.** *Az* OPT *algoritmus által kapott megoldásra teljesül*

$$\sum_{j\in A^*} b_j \leq \frac{1}{2m}\sum_{e\in E} c_e(v).$$

**10. lemma.** *Az* EXP *algoritmus által kapott megoldásra teljesül*

$$\frac{1}{2m}\sum_{e\in E} c_e(v) \leq (1+\log_2\mu)\sum_{j\in A} b_j.$$

A fenti lemmák alapján most már könnyen igazolható a tétel állítása.

Az EXP algoritmus nyeresége $\mathrm{EXP}(I)=\sum_{j\in A} b_j$, az OPT algoritmus nyeresége legfeljebb $\sum_{j\in A\cup A^*} b_j$. Következésképpen a fenti lemmák alapján azt kapjuk, hogy

$$OPT(I) \leq \sum_{j\in A\cup A^*} b_j \leq (2+\log_2\mu)\sum_{j\in A} b_j \leq (2+\log_2\mu)EXP(I),$$

amely egyenlőtlenség igazolja a tétel állítását. □

# 10. fejezet

# Online tanuló algoritmusok

## 10.1. Online gépi tanuló algoritmusok

Mind az online algoritmusok, mind a gépi tanulás témaköre olyan feladatokkal foglalkozik, amikor jelenidejű döntéseket kell hozni, pusztán a múlt ismeretében. Bár az előbb említett két terület különbözik egymástól, mások a hangsúlyos illetve a tipikusan vizsgált feladatok, a gépi tanulás elmélete számos olyan eredményt tartalmaz, amely szépen illeszkedik az online algoritmusok elméletéhez. E fejezetben a gépi tanulás elméletének néhány olyan modelljét tárgyaljuk, ezen belül olyan eredményeket említünk, amelyek az online algoritmusok szemszögéből nézve is érdekesnek tűnnek. Emiatt aztán nem is adhatunk itt teljes áttekintést, csak annyi a célunk, hogy némi betekintést adjunk az olvasónak a terület érdekesebb ötleteibe, eljárásaiba, problémáiba. Néhány egyszerű, de mégis intiutív eredményt tárgyalunk, néhány állításnak a bizonyítását is közöljük. Az érdeklődő, a témában elmélyedni kívánó olvasó számára a megadott irodalom olvasását ajánljuk, ahol a közölt algoritmusok, azoknak további változatai, más hasonló algoritmusok, és még sok egyéb érdekes eredmény is megtalálható.

Lényegében két feladatot mutatunk be, az egyik esetén a tanuló algoritmus szakértői tanácsok alapján tanul, a második, ennél általánosabb esetben pedig példákon keresztül tanul az algoritmus.

### 10.1.1. Előrejelzés szakértői tanácsokból

Tekintsük tehát azt a modellt, amikor szakértői tanácsok alapján tanul az algoritmus, e modellnek tekintélyes irodalma van a gépi tanulás irodalmán belül. Bemutatjuk egy algoritmusnak egy egyszerűbb, és egy módosított változatát, amelyek viszonylag jó versenyképességi hányadost képesek produkálni.

Kezdjük egy egyszerű, könnyen érthető feladattal. Egy tanuló algoritmussal igyekszünk megjósolni, előrejelzést adni arról, hogy aznap esni fog-e vagy sem. Annak érdekében hogy a helyes előrejelzést adhassuk, kikérjük $n$ szakértő tanácsát. Minden nap, mindegyik szakértő ad egy előrejelzést, ami vagy igen vagy nem, ezek alapján az algoritmus maga is hoz egy döntést, hogy igen, tehát szerinte esni fog az eső aznap, vagy pedig nem, aznap nem fog esni. Miután az algoritmus meghozta a saját előrejelzését kiderül, hogy az előrejelzése helyes volt-e, vagy hibás, vagyis tényleg esett-e azon a napon. Az egyszerűség kedvéért semmilyen

feltevéssel sem élünk a szakértők hozzáértését, vagy függetlenségét illetően, ezáltal nem is várhatjuk, hogy az előrejelzésünk teljesen pontos legyen. Ilyen helyzetben az algoritmusunk jóságával szemben támasztott természetes elvárásunk az lehet, hogy legyen legalább megközelítőleg olyan jó, mint a legjobb szakértő. Más szavakkal, azt kívánjuk elérni, hogy az algoritmus bármely $k$ lépés alatt legfeljebb csak $c$-szer annyi rossz döntést hozzon, mint az első $k$ lépés alatt a legjobbnak bizonyult szakértő.

Az algoritmus végrehajtása során egy próbának nevezzük a következő három lépés egymásutánját: (1) az algoritmus begyűjti a szakértők előrejelzéseit, (2) meghozza a saját döntését, és (3) kiderül, hogy igaz, vagy hamis döntést hozott. (A következőkben feltesszük, hogy egy-egy előrejelzés a $\{0,1\}$ halmazból való, de megjegyezzük, hogy ennél általánosabb, vektorértékű vagy valós értékű előrejelzéseket is vizsgáltak.)

### Egy egyszerű algoritmus

A jelen probléma egy alapváltozata annak, amit szakértői tanácsok által történő előrejelzésnek nevezünk. Most tehát nem foglalkozunk a feladat olyan kiterjesztéseivel, amikor a szakértők az előrejelzéseiket valamilyen valószínűséggel adják meg, vagy ennél bonyolultabb módon adnak előrejelzést arra nézve hogy mi fog bekövetkezni. Az ST (Súlyozott Többség) Algoritmus mindegyik szakértő számára fenntart egy $w_i$ súlyt, és az algoritmus a döntését az alapján hozza meg, hogy a szakértői tanácsoknak az igen vagy a nem oldalára esik-e a súlyozott többsége.

**ST Algoritmus**

1. Kezdetben minden szakértőhöz a $w_i = 1$ súlyt rendeljük, ahol $1 \leq i \leq n$.

2. Legyenek $\{x_1,...,x_n\}$ a szakértők által adott előrejelzések, adjunk igen előrejelzést, vagyis legyen 1 a válaszunk, ha a súlyozott többség ezen az oldalon van, (az igen válasz van túlsúlyban), vagyis

$$\sum_{i:x_1=1} w_i \geq \sum_{i:x_i=0} w_i,$$

   egyébként pedig legyen döntésünk nem, azaz 0.

3. Amikor a helyes válasz, $l$ megérkezik, mindegyik téves előrejelzést adó $i$ szakértőt a neki megfeleltetett $w_i$ súlynak a megfelezésével büntetjük. Pontosabban, legyen $w_i := w_i/2$, ha $x_i \neq l$, ellenkező esetben a $w_i$ súlyt nem változtatjuk.

**39. tétel.** [43] *A* ST *Algoritmus által elkövetett hibás előrejelzések $M$ száma legfeljebb* $2.41(m+\lg n)$*, ahol $m$ a legjobb szakértő által elkövetett hibás előrejelzések eddigi száma.*

*Bizonyítás:* Jelölje $W$ a szakértők összes súlyát, ekkor kezdetben $W = n$ teljesül. Ha az algoritmus hibát vét, ez csak úgy lehet, hogy a szakértők összes súlyának legalább a feléhez

tartozott rossz előrejelzés, emiatt az algoritmus 3. lépésében az összes súly legalább $1/4$-ével csökkent. Ezáltal, ha az algoritmus $M$ hibás választ adott, akkor teljesül a következő egyenlőtlenség:

$$W \leq n \cdot (3/4)^M .$$

Másrészt, amennyiben a legjobb szakértő $m$ hibát vétett, akkor e szakértő súlya pontosan $(1/2)^m$, ezáltal

$$W \geq (1/2)^m .$$

A két egyenlőtlenséget összevetve $(1/2)^m \leq W \leq n \cdot (3/4)^M$ adódik, amiből már könnyen adódik az alábbi levezetést:

$$\begin{aligned} n \cdot 2^m &\geq (4/3)^M , \\ \log_2 (n \cdot 2^m) &\geq \log_2 (4/3)^M , \\ \log_2 n + m &\geq M \cdot \log_2 (4/3) , \end{aligned}$$

vagyis

$$M \leq \frac{1}{\log_2 (4/3)} (\log_2 n + m) \leq 2.41 \cdot (\log_2 n + m) . \qquad \square$$

Kétféleképpen kaphatunk az előbbinél hatékonyabb algoritmust. Az egyik lehetőség a véletlenítés. Ezen itt azt értjük hogy az előbbi súlyozott többség választása helyett a súlyokat valószínűségeknek tekintjük, és egy-egy szakértő javaslatát akkora valószínűséggel választjuk, mint amekkora a hozzá rendelt súly. A másik lehetőség pedig az lehet, hogy a súlyok felezése helyett valamilyen más $\beta$-val szorzunk az algoritmus során, ahol $0 < \beta < 1$. Ezeket a változtatásokat elvégezve algoritmusunk a következőképpen néz ki:

**VST Algoritmus:**

1. Kezdetben legyen $w_i = 1$, $1 \leq i \leq n$.
2. Legyenek $\{x_1, ..., x_n\}$ a szakértők által adott előrejelzések, ekkor adjunk $x_i$ választ $w_i/W$ valószínűséggel, ahol $W = \sum_i w_i$.
3. Amint a helyes válasz, $l$ megérkezik, a hibás előrejelzést adó szakértőkhöz rendelt súlyokat megszorozzuk $\beta$-val, és menjünk újra a 2. lépésre.

Amint azt reméltük, a módosítások által az algoritmus hatékonysága javul. Bizonyítás nélkül közöljük az alábbi tételt:

**40. tétel.** [43] *A* VST *algoritmus által elkövetett hibás előrejelzések $M$ számára teljesül az alábbi egyenlőtlenség:*

$$M \leq \frac{m \ln(1/\beta) + \ln n}{1 - \beta},$$

*ahol $m$ a legjobb szakértő által elkövetett hibás előrejelzések eddigi száma.*

**Megjegyzés:** Az előbbi képletben, (a logaritmus értékét enyhén felülről becsülve) $\beta = 1/2$ esetén a következőt kapjuk: $M < 1.39m + 2\ln n$, ha pedig például $\beta = 3/4$, akkor $M < 1.15m + 4\ln n$. Általában is, ahogy a $\beta$ szorzót növeljük, a multiplikatív konstans tetszőlegesen meg tudja közelíteni felülről az 1-et, ennek azonban az additív konstans (minden határon túl való) növekedése az ára.

A feladatnak sok változata és elnevezése van, mint például univerzális kódolás, univerzális előrejelzés, ezeket és más részleteket az érdeklődő olvasó megtalálhatja a [8] cikkben, illetve az ebben idézett művekben.

### 10.1.2. Online tanulás példák alapján

Előbb azzal foglalkoztunk, amikor szakértők tanácsából tanul az algoritmusunk. Most egy ennél általánosabb esetet vizsgálunk, amikor az algoritmus példákból tanul. A példák általában $n$-dimenziós $0-1$ vektorok, vagyis legyen a példák halmaza $X = \{0,1\}^n$. A tanulási folyamat most is próbákból áll. Egy-egy ilyen próba a következő három lépés egymásutánja: Először egy $x \in X$ példát kap az algoritmus, erre az algoritmus ad egy előrejelzést, hogy szerinte az $x$ példa esetén a 0 vagy 1 válasz a helyes, (más szóval az $x$ példa pozitív vagy negatív), és végül az algoritmus tudomására jut a helyes válasz, $l \in \{0,1\}$. Minden tévedés, vagyis minden hamis előrejelzés, más szóval, amikor az előrejelzés nem egyezik meg a helyes $l$ értékkel, hibás választ jelent. A célunk az hogy minél kevesebb hibás választ adjon az algoritmus. A példák a versenyképességi elemzéshez hasonlóan itt is valamilyen előre eltervezett módon jönnek, vagyis egy intelligens ellenfél szándékosan olyan példákat ad, amire az algoritmus lehetőleg sok hibát fog produkálni. A most leírt modellt Hibakorlát (Mistake Bound) tanulási módszernek nevezzük.

Általában további feltételekkel is kell élnünk a modellünk esetén, hiszen az offline optimummal (ami 0 hiba!) történő összehasonlítás esetén az algoritmus nem lehetne konstans versenyképes, hiszen nem várható el tőle hogy semmi hibát se vétsen. A következő, további feltevésekkel élünk hát: (1) az $l$ helyes válasz az $x$ példából valamilyen függvény eredményeként jöjjön ki, (2) az offline algoritmus, amivel az online algoritmusunkat versenyeztetjük, valamilyen előre ismert algoritmus-osztály tagja legyen, és (3) az ellenfél viselkedésében feltételezünk valamilyen véletlenszerűséget. E három feltétel alkalmazásához most szükségünk lesz a koncepció osztály fogalmára. Egy $C$ koncepció osztályon egyszerűen az $X$ halmazon értelmezett Boole-függvények egy csoportját értjük, a Boole-függvények reprezentációjával együtt. Például, a $\{0,1\}^n$ halmazon értelmezett diszjunkciók osztálya azokat a függvényeket jelenti, amelyek leírhatók az $x_1,\ldots,x_n$ bináris változók diszjunkciójaként. A DNF-formulák osztálya az összes Boole-függvényt jelenti. Tetszőleges $c \in C$ Boole-függvény esetén jelölje $s(c)$ a függvény minimális DNF formulájának a hosszát.

A Hibakorlát Modell esetén feltesszük, hogy az algoritmus tanulása során, az $x$ példából úgy adódik az $l$ helyes válasz, hogy $x$-et behelyettesítjük egy $c$ koncepcióba. A tanuló algoritmus persze nem tudja előre, hogy a $C$ osztály melyik rögzített $c$ eleméről van szó, ezt kell neki kitalálnia, a lehető legkevesebb hibával. Ha egy algoritmus a tanulása során legfeljebb $poly(n,s(c))$ hibát vét tetszőleges $c \in C$ esetén, és a futási ideje is legfeljebb $poly(n,s(c))$ , akkor azt mondjuk hogy az algoritmus képes felismerni a C osztály bármely elemét a hibakorlát modellben. Továbbá, ha a vétett hibák száma csak legfeljebb $poly(s(c)) \cdot poly(\log n)$,

vagyis az algoritmus hatékony abban az értelemben, hogy az irreleváns változók nem növelik a hibák számát, akkor azt mondjuk, hogy az algoritmus attribútum hatékony.

Sok különféle algoritmust fejlesztettek ki a sokféle koncepció osztály esetén, ilyen koncepció osztályok például a következők: diszjunkciók, $k$-DNF formulák, döntési listák, és egyebek. Mindjárt rátérünk egy elegáns és praktikus algoritmus, a Winnow algoritmus ismertetésére, amely egy diszjunkciót legfeljebb $O(r \log n)$ hibával képes felismerni, ahol $r$ a diszjunkcióban ténylegesen szereplő változók száma. Ez az algoritmus tehát attribútum hatékony.

Előbb jegyezzük még meg, hogy a koncepció osztályok között többféle redukciós lehetőség ismert. Például, a diszjunkciók, konjunkciók, $k$-CNF formulák, $k$-DNF formulák, (valamely rögzített $k$-ra) mind átalakíthatók monoton diszjunkciókra, (ilyen utóbbi például ez is: $x_1 \vee x_5 \vee x_9$). Emiatt az előbbi osztályok helyett elég a monoton diszjunkciók osztályát felismerő algoritmusokkal foglalkozni.

Most először lássunk egy egyszerű algoritmust, amely monoton diszjunkciókat ismer fel. Kezdetben feltételezzük, hogy a keresett diszjunkció a következő: $h = x_1 \vee x_2 \vee \ldots \vee x_n$. Ezután, ha valamely $x$ példa helyes kiértékelése 0, (ezekre mondtuk azt is hogy a példa negatív), viszont $h$ kiértékelése az algoritmus által 1, akkor ez csak úgy lehet, hogy fölösleges tagok vannak még $h$-ban, ezért tehát eltávolítjuk a $h$-ból azokat az $x_i$ változókat, amelyeknek az értéke az $x$ példában 1. Jegyezzük meg, hogy pozitív példa esetén az algoritmus soha nem vét hibát, hiszen minden szükséges tag megmarad $h$-ban, csak kezdetben még túl sok van belőlük. Ezáltal minden hiba esetén legalább egy tag eltávozik $h$-ból, vagyis az algoritmus legfeljebb $n$ hibát vét. Ez az algoritmus ezek szerint képes felismerni a diszjunkciót, de nem attribútum hatékony, mert ha a diszjunkció kevés tagot tartalmaz, akkor is akár $n$ hibát is véthet az algoritmus.

Ezek után ismertetjük a WINNOW algoritmust (ld. [42]), amely a monoton diszjunkciókat az előzőnél lényegesen hatékonyabb módon ismeri fel. Ha a (monoton) diszjunkció csak $r$ változót tartalmaz, a Winnow algoritmus legfeljebb $O(r \log n)$ hibát vét, amíg a diszjunkciót azonosítja. Hasonlóképp az ST algoritmushoz, ez is fenntart változónként egy-egy súlyt.

**Winnow algoritmus**

1. Kezdetben legyen minden súly 1, azaz $w_1 = w_2 = \ldots = w_n = 1$.

2. Ha a következő példa $x$, legyen $l = 1$, ha

$$w_1 x_1 + w_2 x_2 + \ldots + w_n x_n \geq n,$$

   egyébként legyen $l = 0$.

3. Ha az előrejelzés hibás, akkor

   (a) Ha pozitív lenne a helyes válasz, de az előrejelzés $l = 0$ volt, akkor minden olyan $i$-re ahol $x_i = 1$, megkétszerezzük $w_i$ értékét.

   (b) Ha negatív lenne a helyes válasz, de az előrejelzés $l = 1$ volt, akkor minden olyan $i$-re ahol $x_i = 1$, megfelezzük $w_i$ értékét.

4. Menjünk újra a 2. lépésre.

Az algoritmusra teljesül a következő tétel, amelyet bizonyítás nélkül közlünk.

**41. tétel.** [42] *A Winnow Algoritmus a Hibakorlát modellben a diszjunkciók felismerésekor legfeljebb* $2+3r(1+\log n)$ *hibát vét, ahol a keresett diszjunkció r változót tartalmaz.*

# 11. fejezet

# A versenyképességi elemzés változatai

## 11.1. A módszerek áttekintése

A versenyképességi elemzés több módosítását és változatát használták online algoritmusok vizsgálatára. Ebben a fejezetben röviden összefoglaljuk a legismertebb módszereket, majd bemutatunk néhány konkrét eredményt. A versenyképességi elemzés módosításait két csoportra oszthatjuk. Az egyik csoportba azok a változatok kerülnek, amelyekben az online algoritmusokat egészítjük ki további tulajdonságokkal (például extra információkkal), amelyek lehetővé teszik a versenyképességi hányados csökkentését. A másik osztályba az olyan eredmények tartoznak, ahol az online algoritmusok hatékonyságát csak a lehetséges bemenetek egy részén vizsgáljuk.

**Extra erő az online algoritmusnak**

Az online algoritmusokat elsősorban az alábbi extra tulajdonságokkal szokták kiegészíteni:

- Előrenéző tulajdonság: az algoritmus a további bemenet valamely részét látja.
- Erőforrás kiterjesztés: az offline algoritmus kisebb mennyiségű erőforrást használhat (példát láttunk a weblapletöltési problémánál).
- Az algoritmus apró módosításokat hajthat végre a régebbi döntéseken (pl ládapakolásnál átpakolhat).
- Az algoritmus több megoldást építhet és ezekből a jobbat vesszük figyelembe.

**Megszorítás az ellenfélnek**

Számos olyan eredmény ismert, amelyekben egy online algoritmus hatékonyságát a bemeneteknek csak bizonyos részhalmazán vizsgálják. Az alábbiakban összefoglaljuk a legjellemzőbb részhalmazokat.

- Rendezett bemenet: a bemenet nem lehet tetszőleges, hanem valamely szabály alapján rendezve van, például monoton csökkenő méretű tárgyak ütemezésnél vagy pakolásnál.

- Függőségi gráf: a lapozási probléma esetén olyan modell, amelyben a bemenet nem lehet tetszőleges sorozat, hanem egy gráf (a program függőségi gráfja) pontjait kapjuk, és minden pont után csak egy szomszédja jöhet.

- Félig átlagos elemzés: az ellenfél generálhatja a bemeneti sorozat tartalmát, de maga a bemenet a definiált sorozat egy véletlen permutációja lesz (egyenletes eloszlás alapján).

- Alkalmazkodó függvény: Olyan problémáknál használható, amelyben valamely erőforrás (pl. gépek száma vagy memória mérete) a feladat egy paramétere. Ekkor egy bemenet $\alpha$ sorozat, ha $OPT_{\alpha n} = OPT_{n'}$ minden $n' \geq \alpha n$ esetén, azaz $\alpha n$ mennyiség felett tovább növelve az erőforrások mennyiségét az optimális megoldás célfüggvényértéke már nem változik. Az alkalmazkodó függvény minden $\alpha$ esetén $\alpha$ sorozatok mellett vizsgálja a versenyképességi hányadost.

- Adott összméret: Ütemezési modellekben néha jobb versenyképességi hányadost lehet elérni, ha az algoritmus előre tudja az ütemezendő munkák megmunkálási időinek összegét.

- Ismert korlátok: Ütemezési feladatoknál előfordulhat, hogy előre tudunk az ütemezendő munkák végrehajtási idejeire valamilyen alsó vagy felső korlátot, vagy esetleg mindkettőt.

## 11.2. Előrenéző algoritmusok

### 11.2.1. Lapozás

A lapozási probléma esetén több előrenéző változatot is vizsgáltak, mi itt az [1] cikkben bemutatott erős előrenézés esetére mutatjuk be az alapvető eredményeket. Ebben az esetben egy $l$-előrenéző algoritmus azt a legrövidebb prefixet látja, amely l különböző lapra vonatkozó kérést tartalmaz. Az *LRU* algoritmusnak az alábbi kiterjesztése egy természetes ötlet a probléma megoldására.

**LRU(l) algoritmus:** Amennyiben a memória tele van, és egy lapot be kell tennünk, akkor az előre látott szakaszban nem igényelt lapok közül a legrégebben használtat dobjuk ki.

Az alábbi, bizonyítás nélkül közölt tételek mutatják, hogy a versenyképesség szempontjából az $LRU(l)$ algoritmus a legjobb erős előrenéző algoritmus, ami megadható.

**42. tétel.** [1] *Az LRU(l) algoritmus versenyképességi hányadosa $k-l$, ha $l \leq k-2$.*

**43. tétel.** *[1] Nincs olyan erősen l-előrenéző online algoritmus, amelynek versenyképességi hányadosa kisebb, mint $k-l$, ha $l \leq k-2$.*

### 11.2.2. Nyugtázás

Érdekes kérdés, hogy lehet-e a nyugtázási feladat esetén 2-nél kisebb versenyképességet elérni, ha az algoritmus valamennyi extra információt kap. Igazolható, hogy tetszőleges $N$ konstansra nem elegendő információ az elkövetkező $N$ csomag érkezési idejét ismerni ahhoz, hogy 2 versenyképesnél jobb hányadosú algoritmust kapjunk. Az alábbiakban a LIP (Lookahead Interval Planning) algoritmust (ld. [31]) mutatjuk be, amely egy adott $c$ hosszú intervallumban előre látja a csomagok érkezési idejét.

Az algoritmus blokkokra bontja a bemenetet és minden blokkra a csomagokat az optimális offline megoldás alapján nyugtázza. A blokk mindig az első nyugtázatlan csomagnál kezdődik. Elsőként megvizsgáljuk, hogy van-e két egymást követő csomag: $a_i$ és $a_{i+1}$ a $c$ hosszú intervallumban, amelyekre $a_{i+1} - a_i \geq 1$ teljesül. Ha van ilyen pár, akkor a blokk az első ilyen $a_i$ csomagnál véget ér, egyébként a blokk hossza $c$. LIP meghatározza az optimális offline megoldást a blokk csomagjaira és ennek megfelelően küldi a nyugtákat.

**44. tétel.** [31] *LIP $1+1/c$-versenyképes.*

*Bizonyítás:* Vegyünk egy tetszőleges $I$ bemenetet és osszuk fázisokra. Legyen $S_1 = \{a_1, \ldots, a_{k(1)}\}$, ahol $k(1)$ az első olyan index, amelyre $a_{k(1)+1} - a_{k(1)} \geq 1$. A többi fázis hasonlóan definiálható $S_{j+1} = \{a_{k(j)+1}, \ldots, a_{k(j+1)}\}$, ahol $k(j+1)$ az első olyan index $k(j)$ után, amelyre $a_{k(j+1)+1} - a_{k(j+1)} \geq 1$. Az utolsó fázist az utolsó munka zárja. Ekkor van olyan optimális offline algoritmus, amely minden fázis utolsó csomagjánál nyugtát küld. (Ha a fázis utolsó csomagját a következő fázisban nyugtázza, akkor a késedelmi költség növekedése legalább 1.) Másrészt egy fázis utolsó csomagja szintén utolsó csomagja valamelyik blokknak is, így LIP is küld nyugtát a csomag érkezésekor.

Vegyünk egy tetszőleges $S_i$ fázist. Jelölje $r$ a benne szereplő blokkok számát. Vegyünk egy optimális megoldását a fázisnak. Ha kiegészítjük $r-1$ további nyugtával, akkor egy olyan megoldást kapunk, amely minden blokk végén nyugtát küld. Másrészt ezek közül LIP a legkisebb költségű megoldást adja meg, így $(r-1) + OPT(S_i) \geq LIP(S_i)$, azaz $LIP(S_i)/OPT(S_i) \leq 1 + (r-1)/OPT(S_i)$.

Mivel minden blokk ugyanabban a fázisban van, ezért az első $r-1$ blokk hossza $c$, így a fázis hossza legalább $(r-1)c$. Tegyük fel, hogy egy offline algoritmus $k$ nyugtát küld a fázisban. Ekkor az első $k-1$ nyugta mindegyike után egy legfeljebb 1 hosszú csomagmentes időintervallum van. Ebből adódik, hogy a teljes késedelem legalább $(r-1)c-(k-1)$. Következésképpen $OPT(S_i) \geq k + (r-1)c - (k-1) = (r-1)c + 1$. Tehát azt kaptuk, hogy $LIP(S_i)/OPT(S_i) \leq 1 + 1/c$. □

Másrészt 1-versenyképes algoritmus nem konstruálható, amint azt az alábbi, bizonyítás nélkül közölt állítás mutatja.

**45. tétel.** [31] *Tetszőleges $c$-hosszú intervallumra előrenéző algoritmus versenyképességi hányadosa $1 + \Omega(1/c^2)$.*

## 11.3. Függőségi gráf

A lapozás feladata esetén érdekes eredményeket értek el a programok függősségi gráfjának figyelembe vételével. Egy függőségi gráf esetén a gráf csúcsai a lehetséges lapok, és két csúcsot akkor köt össze él, ha előfordulhatnak egymás után a kérések listájában. Ez azt jelenti, hogy a bemenet egy séta a függőségi gráfon.

Ekkor értelemszerűen egy algoritmus versenyképessége függ a vizsgált függőségi gráftól, $c_{A,k}(G)$ jelöli a $G$ gráf mellett az $A$ algoritmus versenyképességi hányadosát. Továbbá $c_k(G)$ jelöli az elérhető legjobb versenyképességi hányadost a $G$ gráf mellett. A két legismertebb online algoritmust összehasonlították függőségi gráfokra, és az alábbi eredményt kapták.

**46. tétel.** [13] LRU *versenyképessége egyetlen gráf esetén sem nagyobb, mint FIFO versenyképessége.*

Nyilván a függőségi gráf ismeretében lehetséges olyan algoritmust kifejleszteni, amely használja a gráf struktúráját. Egy ilyen algoritmus a FAR algoritmus, amely egy bélyegző algoritmus, amely mindig azt a jelöletlen lapot rakja ki, amelynek a kért laptól a távolsága maximális a függőségi gráfban. Erre az algoritmusra teljesül a következő állítás:

**47. tétel.** [36] *Minden $G$ és $k$ esetén $c_{\mathrm{FAR},k}(G) = O(c_k(G))$.*

Gyakorlati szempontból a probléma az, hogy nem ismerjük előre az egyes alkalmazások mögött a függőségi gráfot, így azokat a gyakorlatban nem használhatjuk online lapozási algoritmusok kifejlesztésére. Erre javasolták a [26] cikkben azt az ötletes megoldást, hogy az online algoritmus futása közben tanulja a függőségi gráfot. Ezzel az megoldással sikerült egy olyan online algoritmust kifejleszteni, amely véletlen teszteseteken is az LRU algoritmushoz hasonlóan jó eredményt ért el.

## 11.4. Félig átlagos elemzés

A félig átlagos elemzés esetén az ellenfél generálhatja a bemeneti sorozat tartalmát, de a bemenet a definiált sorozat elemeinek egy véletlen permutációja lesz (általában egyenletes eloszlás alapján). Ilyen kérdéseket több online probléma esetén vizsgáltak, az alábbiakban csak egy problémát tekintünk, ahol jól látszik a véletlen sorrend jelentősége.

A konstans költségű kiszolgáló-elhelyezési feladatban adottak egy metrikus térben $s_1, \ldots, s_n$ kérések, amelyek a metrikus tér pontjai. Az algoritmusnak kiszolgálókat kell elhelyeznie a metrikus tér pontjaiba. A cél a kiszolgálás teljes költségének minimalizálása, amely költség az alábbi két részköltség összege:

- A kiszolgálók elhelyezési költsége: egy $f$ konstans szorozva a kiszolgálók számával.
- A kérések kiszolgálási költsége: $\sum_{i=1}^{n} min_{j=1,\ldots,k} d(s_i, p_j)$, ahol a kiszolgálók a $p_1, \ldots, p_k$ pontokban vannak. (Egy kérés kiszolgálásának a költsége a legközelebbi ponttól való távolsága.)

Az online kiszolgáló-elhelyezési feladatban a kérések egyenként jönnek és az egyes kérések érkezése után kell eldöntenünk, hogy veszünk - e új kiszolgálót. A feladat megoldására a [45] cikkben a következő egyszerű véletlenített algoritmust javasolták.

**Meyerson algoritmus:** Legyen a kérés távolsága a legközelebbi kiszolgálótól $d$, és legyen $p = \min\{d/f, 1\}$. Vegyünk a kérés helyén egy új kiszolgálót $p$ valószínűséggel.

Az algoritmus versenyképességét illetően a [45] dolgozatban az alábbi állítást igazolták.

**48. tétel.** [45] *Meyerson algoritmusa $O(\log n)$ versenyképes, ahol $n$ a kérések száma.*

Szintén igazolást nyert az alábbi állítás.

**49. tétel.** [45] *Nincs konstans versenyképes algoritmus az online kiszolgáló-elhelyezés problémájának megoldására.*

Ezzel szemben a fenti algoritmus konstans versenyképes, ha a félig átlagos elemzés alapján vizsgáljuk, mint azt az alábbi tétel mutatja.

**50. tétel.** [45] *Amennyiben a bemeneti sorozaton a végrehajtás előtt egy egyenletes eloszlás alapján választott permutációt is végrehajtunk, akkor ezen bemenetekre a Meyerson féle algoritmus konstans versenyképes.*

## 11.5. Rendezett bemenetek

Bizonyos algoritmusok sokkal jobb eredményt érnek el, ha tudjuk, hogy a bemenet valamely szabály alapján rendezve van. Az alábbiakban a ládapakolás és az ütemezés területéről foglalunk össze néhány ilyen eredményt.

### 11.5.1. Ládapakolás csökkenő méretű elemekkel

A ládapakolás esetén alkalmazott bizonyos algoritmusok lényegesen jobb eredményt adnak, amennyiben az elemek méret szerint csökkenő sorrendben érkeznek. Ezt mutatják az alábbi, bizonyítás nélkül közölt tételek.

**51. tétel.** [5] *Az* NF *algoritmus aszimptotikusan $\sum_{i=1}^{\infty} 1/a_i \approx 1.691$-versenyképes csökkenő sorozatok esetén, ahol $a_1 = 1$ és $a_{i+1} = a_i(a_i + 1)$ $i > 1$ esetén. (Az általános esetben 2-versenyképes)*

**52. tétel.** [17] *A* FF *algoritmus aszimptotikusan $11/9 \approx 1.22$-versenyképes csökkenő sorozatok esetén. (Az általános esetben 1.7-versenyképes.)*

### 11.5.2. Ütemezés

Az azonos gépek ütemezése esetén láttuk, hogy a LISTA algoritmus esetén a legrosszabb bemenetben kis megmunkálási idővel rendelkező munkák voltak elől és egy hosszú munkával zárult a bemenet. Az alábbi tétel mutatja, hogy az algoritmus lényegesen jobb versenyképességi hányadossal rendelkezik, ha csak olyan bemeneteket vizsgálunk, amelyekben a munkák a megmunkálási idő szerint monoton csökkenően rendezettek. Ebben az esetben az algoritmust LPT (longest processing time) algoritmusnak nevezzük.

**53. tétel.** [28] *A* LISTA *algoritmus* $4/3-1/(3m)$*-versenyképes csökkenő méretű munkák esetén.*

*Bizonyítás:* Az állítást indirekt igazoljuk. Ehhez tegyük fel, hogy léteznek olyan ellenpéldák, amelyekre az optimális ütemezés és az LPT algoritmus által kapott ütemezés költségeinek hányadosa nagyobb mint $4/3-1/(3m)$. Ezen ellenpéldák közül van olyan, amely minimális számú munkát tartalmaz.

Mivel a tekintett ellenpéldában a munkák száma minimális, ezért az utolsónak érkezett munka befejezési ideje megegyezik a maximális befejezési idővel. Amennyiben ez a tulajdonság nem teljesülne, akkor arra a $\sigma'$ bemenetre, amelyet a tekintett ellenpéldából az utolsó munkát elhagyva kapnánk $LPT(\sigma')=LPT(\sigma)$ és $OPT(\sigma)\geq OPT(\sigma')$ teljesülne.

A fentiek alapján azt kapjuk, hogy az utolsó munka kezdési ideje $LPT(\sigma)-p_n$. Másrészt eddig az időpontig az összes gép dolgozott, így

$$LPT(\sigma)-p_n\leq\frac{\sum_{i=1}^{n-1}p_i}{m}.$$

Továbbá $OPT(\sigma)\geq\frac{1}{m}(\sum_{j=1}^{n}p_j)$.
Következésképp

$$\frac{4}{3}-\frac{1}{3m}<\frac{LPT(\sigma)}{OPT(\sigma)}\leq\frac{p_n+\sum_{i=1}^{n-1}p_i/m}{OPT(\sigma)}=$$

$$\frac{p_n(1-1/m)}{OPT(\sigma)}+\frac{\sum_{i=1}^{n}p_i/m}{OPT(\sigma)}\leq\frac{p_n(1-1/m)}{OPT(\sigma)}+1.$$

A fenti egyenlőtlenség jobb és baloldalát véve a következő korlát adódik $OPT(\sigma)$ értékére:

$$OPT(\sigma)<3p_n.$$

Mivel $p_n$ a minimális végrehajtási idő, ezért a fenti egyenlőtlenség azt jelenti, hogy az optimális ütemezésben minden gép legfeljebb két munkát tartalmaz, azaz $n\leq 2m$. Másrészt ebben az esetben az LPT eljárás optimális, azaz ellentmondáshoz jutottunk, amivel a tétel állítását igazoltuk. □

Még megjegyezzük, hogy olyan bemenet, amelyre az LPT versenyképességi hányadosa éppen a lehető legrosszabb, vagyis $4/3-1/(3m)$, minden $m$ gépszám esetén csak egyetlen egy létezik [22] szerint, és ez a következő: Az első két munka hossza $2m-1$, a következő

kettő hossza $2m-2$, és így tovább, utoljára jön két munka $m+1$ hosszúsággal, és legutoljára még három $m$ hosszú munka. Bármely más bemenet esetén az $LPT(\sigma)/OPT(\sigma)$ hányados szigorúan kisebb mint a legrosszabb eset hányadosa.

# Irodalomjegyzék


[1] S. Albers, The Influence of Lookahead in Competitive Paging Algorithms, *In Proceedings of ESA93, LNCS 726*, Springer-Verlag, 1–12, 1993.

[2] S. Albers, B. von Stengel, R. Werchner, A combined BIT and TIMESTAMP algorithm for the list update problem, *Information Processing Letters*, **56**, 135–139, 1995.

[3] S. Albers, J. Westbrook, Self-organizing data structures, *In Online algorithms: The State of the Art LNCS 1442*, Springer-Verlag, 13–51, 1998.

[4] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, O. Waarts, On-line load balancing with application to machine scheduling and virtual circuit routing, *Journal of the ACM*, **44**, 486–504, 1997.

[5] B.S. Baker, E.G. Coffman, A Tight Asymptotic Bound for Next-Fit-Decreasing Bin-Packing, *SIAM Journal on Algebraic and Discrete Methods*, **2**, 147–152, 1981.

[6] B.S. Baker, J.S. Schwartz, Shelf algorithms for two dimensional packing problems, *SIAM Journal on Computing*, **12**, 508–525, 1983.

[7] Y. Bartal, S. Leonardi, A. Marchetti-Spaccamela, J. Sgall, L. Stougie, Multiprocessor scheduling with rejection, *SIAM Journal on Discrete Mathematics*, **13**, 64–78, 2000.

[8] A. Blum, Online Algorithms in Machine Learning, *In Online algorithms: The State of the Art LNCS 1442*, Springer-Verlag, 306–325, 1998.

[9] J. Balogh, J. Békési, G. Galambos, New Lower Bounds for Certain Classes of Bin Packing Algorithms, *Proceeding of WAOA10, LNCS 6534*, 25–36, 2010.

[10] A. Borodin, R. El-Yaniv, *Online Computation and Competitive Analysis*, Cambridge University Press, 1998.

[11] Y. Cho, S. Sahni, Bounds for list schedules on uniform processors, *SIAM Journal on Computing*, **9**, 91–103, 1988.

[12] M. Chrobak, L. Larmore, An optimal algorithm for k-servers on trees, *SIAM Journal on Computing*, **20**, 144–148, 1991.

[13] M. Chrobak, J. Noga, LRU Is Better than FIFO, *Algorithmica*, **23(2)**, 180–185, 1999.

[14] J. Csirik, G. Woeginger, Shelf algorithms for on-line strip packing, *Information Processing Letters*, **63**, 171–175, 1997.

[15] J. Csirik, G. Woeginger, On-line packing and Covering problems, *In Online algorithms: The State of the Art LNCS 1442*, Springer-Verlag, 147–177, 1998.

[16] D. R. Dooly, S. A. Goldman, S. D. Scott: On-line analysis of the TCP acknowledgment delay problem, *Journal of the ACM*, **48(2)**, 243–273, 2001.

[17] Gy. Dósa, The Tight Bound of First Fit Decreasing Bin-Packing Algorithm Is $FFD(I) \leq (11/9)OPT(I)+6/9$, *Combinatorics, Algorithms, Probabilistic and Experimental Methodologies, LNCS 4614*, 1–11, 2007.

[18] Gy. Dósa, Z. Tan, New upper and lower bounds for online scheduling with machine cost, *Discrete Optimization*, **7(3)**, 125–135, 2010.

[19] Gy. Dósa, Y. He, Preemptive and non-preemptive on-line algorithms for scheduling with rejection on two uniform machines, *Computing*, **76(1)**, 149–164, 2006.

[20] Gy. Dósa, Y. He, Scheduling with machine cost and rejection, *Journal of Combinatorial Optimization*, **12**, 337–350, 2006.

[21] Gy. Dósa, Y. He, Better Online Algorithms for Scheduling with Machine Cost, *SIAM Journal on Computing*, **33(5)**, 1035–1051, 2004.

[22] Gy. Dósa, Graham's example is the only one tight one for P II Cmax, *Annales Univ. Sci. Budapest. Eötvös Sect. Math.* **47**, 207–210, 2004.

[23] A. Fiat, R.M. Karp, M. Luby, L. A. McGeoch, D.D. Sleator, N.E. Young, Competitive Paging Algorithms, *Journal of Algorithms*, **12**, 685–699, 1991.

[24] R. Fleischer, M. Wahl, On-line scheduling revisited, *Journal of Scheduling*, **3(6)**, 343–353, 2000.

[25] A. Fiat, Y. Rabani, Y. Ravid, Competitive k-server algorithms, *Journal of Computer and System Sciences*, **48**, 410–428, 1994.

[26] A. Fiat, Z. Rosen, Experimental Studies of Access Graph Based Heuristics: Beating the LRU Standard? *Proccedings of SODA97*, 63–72, 1997.

[27] A. Fiat, G.J. Woeginger (szerk.) *Online algorithms: The State of the Art LNCS 1442*, Springer-Verlag, 1998.

[28] R. Graham, Bounds for certain multiprocessor anomalies, *Bell System Technical Journal*, **45**, 1563–1581, 1966.

[29] Cs. Imreh, Online strip packing with modifiable boxes, *Operations Research Letters*, **66**, 79–86, 2001.

[30] Cs. Imreh, Competitive analysis, *Algorithms of Informatics Volume 1*, szerk A. Iványi, mondAt, Budapest, 395–428, 2007.

[31] Cs. Imreh, T. Németh, On time lookahead algorithms for the online data acknowledgement problem *Proceedings of MFCS07, LNCS 4708*, 288–297, 2007.

[32] Cs. Imreh, T. Németh, Parameter learning algorithm for the online data acknowledgment problem, *Optimization Methods and Software*, megjelenés alatt, DOI 10.1080/10556788.2010.544313

[33] Cs. Imreh, Online scheduling with general machine cost functions, *Discrete Applied Mathematics*, **157**, 2070–2077, 2009.

[34] Cs. Imreh, J. Noga, Scheduling with machine cost, *Proceedings of RANDOM-APPROX 99, LNCS 1671*, 168–176, 1999.

[35] S. Irani, Two results on the list update problem, *Information Processing Letters*, **38**, 301–306, 1991.

[36] S. Irani, A. Karlin, S. Phillips, Strongly competitive algorithms for paging with locality of reference, *SIAM Journal of Computing*, **25(3)**, 477–497, 1996.

[37] A. Iványi, Performance bounds for simple bin packing algorithms, *Annales Univ. Sci. Budapest, Sectio Combinatorica*, **5**, 77–82, 1984.

[38] A. Iványi, Tight worst-case bounds for bin packing algorithms, *Theory of Algorithms, Colloquia of Mathematical Society János Bolyai 44. kötete*, North-Holland, 233–240, 1985.

[39] D.S. Johnson, *Near-optimal bin packing algorithms*, PhD disszertáció, MIT, 1973

[40] D.S. Johnson, A. Demers, J.D. Ullman, R.M. Garey, R.L. Graham, Worst-case performance bounds for simple one-dimensional packing algorithms, *SIAM Journal of Computing*, **3**, 256–278, 1974.

[41] E. Koutsoupias, C. Papadimitriou, On the k-server conjecture, *Journal of the ACM*, **42**, 971–983, 1995.

[42] N. Littlestone, Learning Quickly When Irrelevant Attributes Abound: A New Linear-threshold Algorithm, *Machine Learning*, **2**, 285–318, 1988.

[43] N. Littlestone, M. K. Warmuth, The weighted majority algorithm, *Information and Computation*, **108(2)**, 212–261, 1994.

[44] M.Manasse, L.A. McGeoch, D.Sleator, Competitive algorithms for server problems, *Journal of Algorithms*, **11**, 208–230, 1990.

[45] A. Meyerson, Online Facility Location, *Proceedings of FOCS01*, 426–431, 2001.

[46] R. Motwani, P. Raghavan, *Randomized Algorithms*, Cambridge Univerity Press, 1995.

[47] M.J. Osborne, A. Rubinstein, *A course in game theory*, MIT Press, 1994.

[48] D. Sleator, R.E. Tarjan, Amortized efficiency of list update and paging rules, *Communications of the ACM*, **28**, 202–208, 1985.

[49] D. Shmoys, J. Wein, D. P. Williamson, Scheduling parallel machines online, *SIAM Journal on Computing*, **24**, 1313–1331, 1995.

[50] A. Vestjens, *On-line machine scheduling*, PhD disszertáció, Eindhoven University of Technology, 1997.

[51] A. van Vliet, *Lower and upper bounds for on-line bin packing and scheduling heuristics*, PhD disszertáció, Erasmus University, Rotterdam, The Netherlands, 1995.

[52] A.C. Yao, Probabilistic computations: Toward a unifed measure of complexity *Proceedings of FOCS 77*, 222–227, 1977.

[53] N. Young, On-line file caching, *Algorithmica*, **33**, 371–383, 2002