LÁDAPAKOLÁSI ÉS ÜTEMEZÉSI FELADATOK ELMÉLETI, ÉS SZÁMÍTÓGÉPPEL SEGÍTETT VIZSGÁLATA

Doktori (PhD) értekezés

Szerző:

Benkő Attila

Témavezető:

Dr. Dósa György

DOI: 10.18136/PE.2014.566

PANNON EGYETEM

Matematika Tanszék

Informatikai Tudományok Doktori Iskola

2014.

LÁDAPAKOLÁSI ÉS ÜTEMEZÉSI FELADATOK
ELMÉLETI, ÉS SZÁMÍTÓGÉPPEL SEGÍTETT
VIZSGÁLATA

Értekezés doktori (PhD) fokozat elnyerése érdekében

Írta: Benkő Attila

Készült a Pannon Egyetem Informatikai Tudományok Doktori Iskolája keretében.

Témavezető: Dr. Dósa György

Elfogadásra javaslom: (igen / nem)

(aláírás)

A jelölt a doktori szigorlaton ..... %-ot ért el.

Veszprém

.....

a Szigorlati Bizottság elnöke

Az értekezést bírálóként elfogadásra javaslom:

Bíráló neve: ..... (igen / nem)

(aláírás)

Bíráló neve: ..... (igen / nem)

(aláírás)

A jelölt az értekezés nyilvános vitáján ..... %-ot ért el.

Veszprém

.....

a Bíráló Bizottság elnöke

A doktori (PhD) oklevél minősítése .....

.....

Az EDT elnöke

Tartalomjegyzék

Ábrák jegyzéke

Táblázatok jegyzéke

1. Kivonat

A dolgozat első részében egy kombinált feladattal foglalkozunk, amely egyrészt ládapakolással van kapcsolatban, másrészt szállítási, vagy ütemezési feladat is egyben. A modell későbbi vizsgálata érdekében először a dolgozat elején egy általános bevezetőt adunk meg a ládapakolással kapcsolatban és ismertetjük a leglényegesebb eredményeket. Ehhez hasonlóan bevezetjük mindazon fogalmakat, módszereket, amelyek a kombinált feladat vizsgálatához szükségesek. Ezek után az új feladat online változatának megoldására egy új, rugalmas algoritmuscsaládot is ismertetünk. Az offline esetről pedig megmutatjuk hogy bizonyos speciális esetekben polinomiális algoritmussal megoldható, de az általános esetben viszont nem, pontosabban, nem létezik rá APTAS, feltéve hogy $P \neq NP$. A dolgozatban tárgyalt eredményeink az [5, 7, 8, 9] publikációinkon alapulnak.

A disszertáció második részében egy félig online ütemezési feladattal foglalkozunk. A vizsgált modell a következő: Két hasonló gépünk van, ami azt jelenti, hogy az egyik gép gyorsabb mint a másik, s a sebességarány a gyors és a lassú gép között, egyébként a gépek, a sebesség különbségétől eltekintve egyformák, tehát mindkettő gép bármely munkát el tudja végezni. Ha egy munkát már elkezdtünk elvégezni az egyik gépen, akkor a munkának az elvégzését nem lehet megszakítani, tehát a nem megszakításos esettel foglalkozunk. Ezen belül egy olyan félig online modellt vizsgálunk, amikor a gépeken ütemezett munkákat egy bizonyos korlátozott módon újra lehet rendezni. Ennek az átrendezésnek több változatát vizsgálták már, mi azzal az esettel foglalkozunk, amikor valamely online ütemező algoritmus lefutása *után* van lehetőségünk korlátozott mértékben újrarendezésre, nevezetesen legfeljebb K munkát újraütemezhetünk, ahol $K \geq 1$, rögzített.

A modellt a Chen et al. [14] cikkben definiáltuk, és optimális algoritmusokat adtunk meg a következő esetekre: $s \geq 2$ esetén az optimális versenyképességi arány $\rho = \frac{s+2}{s+1}$, minden $K \geq 1$ esetén. Továbbá, ha $K \geq 2$, akkor az optimális algoritmus $1 \leq s \leq \frac{1+\sqrt{5}}{2}$ esetén $\rho = \frac{(s+1)^2}{s^2+s+1}$-versenyképes, és $\frac{1+\sqrt{5}}{2} \leq s \leq 2$ esetén $\rho = \frac{s^2}{s^2-s+1}$-versenyképes. Ugyanebben a cikkben megadtunk $K = 1$ esetére egy $\rho = \frac{s+2}{s+1}$-versenyképes algoritmust

is $1 \leq s \leq 2$ esetére, amely azonban nem optimalis.

Ezzel az esettel (tehát amikor $K = 1$ és $1 \leq s \leq 2$) foglalkozunk részletesebben a dolgozatban, megadunk egy hatékonyabb algoritmust. Ezt az algoritmust a Wang et al. [78] cikkben publikáltuk. Az algoritmus $1 \leq s \leq \sqrt{2}$ esetén $\rho = \frac{2(s+1)}{s+2}$, $\sqrt{2} < s \leq 2$ esetén pedig továbbra is $\rho = \frac{s+2}{s+1}$ versenyképes.

2. Abstract

In the first part of the dissertation we discuss a combined problem that is connected to bin packing problems and delivery or scheduling problems, at the same time. For further examination of the model, first we provide a general introduction about bin packing and we review the most important results. Then, we describe all of the notions and methods that are required to the examination of the combined problem. After this, in order to solve the online version of the new task, we define a new dynamic algorithm family. We prove that the offline case can be solved optimally by polynomial algorithms in special cases, but in the general case there is no APTAS, if $P \neq NP$. The results that are discussed in the dissertation are based on the following papers: [5, 7, 8, 9].

In the second part of the dissertation, we are discuss a semi online scheduling problem. The examined model is the following: we have two uniform machines which means that one is faster than the other, we note the speed ratio between the fast and the slow machine with s. There is no further difference between the machines, i.e. each machine can execute any job. If we have already started to execute a job on a machine, then the execution of the job cannot be interrupted. We are examining such a semi online model, when the scheduling of the jobs on the machines can be rearranged restrictively. Several versions of this rearrangement was already examined, we are discussing the case when the rearrangement can be done only after the execution of the online algorithm, i.e. when the sequence of the jobs is over, and we are informed about this fact. Then, at most K jobs can be rearranged, where $K \geq 1$ is a fixed integer. We have defined this semi online model in the following paper: Chen et al. [14]. We have also provided optimal algorithms for the following cases:

- if $s \geq 2$, then the optimal approximation ratio is $\rho = \frac{s+2}{s+1}$, for all $K \geq 1$.
- if $K \geq 2$, then the optimal approximation ratio is $\rho = \frac{(s+1)^2}{s^2+s+1}$, for all $1 \leq s \leq \frac{1+\sqrt{5}}{2}$.
- if $K \geq 2$ and $\frac{1+\sqrt{5}}{2} \leq s \leq 2$, then the optimal approximation ratio is $\rho = \frac{s^2}{s^2-s+1}$.

In this paper we also provided an algorithm with a $\rho = \frac{s+2}{s+1}$ approximation ratio for the case of $K = 1$ and $1 \leq s \leq 2$, but this algorithm is not optimal.

Thus in the dissertation we discussing this remained case (when $K = 1$ and $1 \leq s \leq 2$) in details. We provide a more efficient algorithm. We have published this algorithm in the following paper: Wang et al. [78]. The approximation ratio of the algorithm is $\rho = \frac{2(s+1)}{s+2}$ for all $1 \leq s \leq \sqrt{2}$ and $\rho = \frac{s+2}{s+1}$ for all $\sqrt{2} < s \leq 2$.

3. Abstrakte

Im ersten Teil der Dissertation werden wir diskutieren über einen kombinierten problem mit bin packing Problem und Lieferung oder Probleme mit dem Zeitplan. Für die weitere Prüfung des Modell, haben wir zuerst eine allgemeine Einführung zum Thema bin Verpackung und überprüfen wir die wichtigsten Ergebnisse. In ähnlich mit diesem beschreiben wir alle Begriffe und Methoden, die erforderlich sind, um die Prüfung der kombinierten Aufgabe. Nach dieser, zur Lösung die online Version der neuen Aufgabe, definieren wir eine neue Dynamik Algorithmus Familie. Wir zeigen, dass die offline Fall gelöst werden können durch polynomiale Algorithmen in besonderen Fällen, aber im allgemeinen Fall nicht, genau, es gibt keine APTAS, wenn $P \neq NP$. Die Ergebnisse, die in der Dissertation ist auf der Basis der folgenden Dokumente [5, 7, 8, 9].

Im zweiten Teil der Dissertation, wir diskutieren über einen semi online planen einer Aufgabe. Das untersuchte Modell ist die folgende: Wir haben zwei ähnliche Maschinen was bedeutet, dass man schneller ist als die anderen, wir beachten das Drehzahlverhältnis zwischen die schnelle und die langsame Maschine mit s, andere als die Ungleichheit, sie sind die gleichen, so dass jede Maschine ausgeführt werden kann. Wenn wir haben bereits damit begonnen haben, einen Job auf einer Maschine, dann die Ausführung des Jobs kann nicht unterbrochen werden daher haben wir es mit der kontinuierlichen Fall. In dieser sind wir Prüfung eines solchen semi online Modell, bei der Planung der Aufträge an den Maschinen können neu gestaltet werden sehr restriktiv auszulegen. Mehrere Versionen dieser Neuordnung war bereits geprüft wurden, wir sprechen über den Fall, wenn die Umstellung durchgeführt werden kann erst nach der Durchführung einer online Algorithmus, nämlich, wenn in den meisten K Arbeitsplätze gestaltet werden kann wenn $K \geq 1$ ist eine feste ganze Zahl.

Wir haben das Modell in den folgenden Artikel: Chen et al. [14] und damit haben wir einen optimalen Algorithmen für die folgenden Fälle:

- ist $s \geq 2$ dann die optimale Angleichung Verhältnis ist $\rho = \frac{s+2}{s+1}$, für alle $K \geq 1$.

- ist $K \geq 2$ dann würde die optimale Angleichung Verhältnis ist $\rho = \frac{(s+1)^2}{s^2+s+1}$, für alle $1 \leq s \leq \frac{1+\sqrt{5}}{2}$.

- ist $K \geq 2$ und $\frac{1+\sqrt{5}}{2} \leq s \leq 2$ dann die optimale Angleichung Verhältnis ist $\rho = \frac{s^2}{s^2-s+1}$.

In diesem Artikel haben wir einen Algorithmus mit einer $\rho = \frac{s+2}{s+1}$ Angleichung Verhältnis für alle $K = 1$ und $1 \leq s \leq 2$, aber dieser Algorithmus ist nicht optimal.

Wir reden über diesen Fall (wenn $K = 1$ und $1 \leq s \leq 2$) Details in der Dissertation und wir bieten eine effizientere Algorithmus. Wir veröffentlichen diesen Algorithmus in das folgende Dokument: Wang et al. [78]. Die Angleichung Verhältnis des Algorithmus ist $\rho = \frac{2(s+1)}{s+2}$ für alle $1 \leq s \leq \sqrt{2}$ und $\rho = \frac{s+2}{s+1}$ für alle $\sqrt{2} < s \leq 2$.

4. Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani témavezetőmnek, Dr. Dósa György docens úrnak többéves, folyamatos útmutatásáért és javaslataiért. Nagyon nagy hálával tartozom Dr. Győri István professzor úrnak folyamatos buzdításáért.

Köszönetet szeretnék mondani szüleimnek, Benkő Lajosnak és Benkő Lajosné Szabó Szilviának a kitartó ösztönzésért és támogatásért, amellyel tanulmányaim sikeres elvégzéséhez hozzájárultak.

Köszönöm továbbá feleségemnek, Benkőné Joó Szilviának a lelki támogatást és biztatást.

5. Bevezető

A disszertáció bevezetőjében áttekintjük azokat a modelleket, amelyekkel a későbbiekben foglalkozni fogunk, valamint a fejezetekben használt alapfogalmakat és főbb algoritmusokat is, és az ezekre vonatkozó lényegesebb eredményeket. Az ütemezési és ládapakolási feladatok témakörében rengeteg cikket publikálnak évről évre. Bizonyos problémák és algoritmusok ma már klasszikusnak számítanak, de az újabb feladatok meghatározásakor előveszik ezek megfelelő változatait. Gráfokkal, vagy játékelmélettel kapcsolatos ládapakolási, illetve ütemezési feladatok is kerülnek elő, tehát a téma szerteágazó.

5.1. Diszkrét programozás

A diszkrét programozási feladatok megoldása során valamely diszkrét halmazon értelmezett függvény optimumát keressük meg [47]. Gyakran 0-1 vektorokkal azonosíthatjuk a halmaz elemeit, amelyek kombinatorikai struktúrákat reprezentálnak, például gráf feszítőfáit. Néhány nevezetes diszkrét programozási (vagy más néven kombinatorikus optimalizálási) feladat a következő: az utazó ügynök feladat, minimális feszítőfa keresésének feladata, vagy a hátizsák feladat.

Diszkrét programozás esetén adott egy M véges halmaz, ahol $M = \{\alpha_1, \ldots, \alpha_n\}$ és legyen f egy numerikus függvény az M halmaz elemein értelmezve. Ekkor a feladat az, hogy megtaláljuk azt az $\alpha_j \in M$ elemet, ahol az f függvénynek abszolút minimuma (vagy maximalizálási feladat esetén maximuma) van. A diszkrét programozás csak a nemtriviális feladatokkal foglalkozik, amelyek eleget tesznek az alábbi két követelménynek:

1) Az $n = |M|$ számnak kellően nagynak kell lennie ahoz, hogy a feladatot ne lehessen megoldani manuálisan az $f(\alpha_i)$ értékek egyenkénti kiértékelésével.

2) A feladatnak nem szabad regulárisnak lenni. A feladat akkor reguláris, ha:

a) minden $\alpha_i \in M$ elemre van nem-üres szomszédja (definiálva van tehát egy szomszédság-struktúra, az $\alpha_i \in M$ elem szomszédjait $S(\alpha_i, M)$ jelöli, ezek az M halmaz azon elemei, amelyeket valamilyen mérték szerint közel vannak az adott α_i ponthoz) és

ha $|S(\alpha_i, M)| << |M|$.

b) az f függvény lokális szélsőértéke definiált egy egyszerű algoritmussal.

c) legalább egy lokális szélsőérték megegyezik a globális szélsőértékkel.

5.2. Abszolút, illetve aszimptotikus approximációs arányok

Tekintsünk egy minimalizálási optimalizálási feladatot. Ekkor az A algoritmus abszolút approximációs aránya a legkisebb olyan C szám, amelyre bármely I input esetén $A(I) \leq C \cdot OPT(I)$ teljesül. Itt $A(I)$ jelöli az I input esetén az A algoritmus által kapott megoldás értékét, amely egy valós szám, $OPT(I)$ pedig valamely optimális algoritmus megoldásának értékét. Ha egy, az inputtól független D nemnegatív additív konstans használata a definícióban megengedett, tehát minden I inputra az alábbi egyenlőtlenségnek kell teljesülnie: $A(I) \leq C \cdot OPT(I) + D$, akkor az így kapott legkisebb megfelelő C konstans az aszimptotikus approximációs arány.

Online ütemezési feladatok esetén abszolút approximációs arány helyett *versenyképességi arányt* szokás mondani.

Maximalizálási feladatok esetén az előbbi definíciókat kicsit át kell fogalmazni: Ekkor az A algoritmus abszolút approximációs aránya a legnagyobb olyan C szám, amelyre bármely I input esetén $A(I) \geq C \cdot OPT(I)$ teljesül. Az aszimptotikus approximációs arány esetében pedig minden I inputra az alábbi egyenlőtlenségnek kell teljesülnie: $A(I) \geq C \cdot OPT(I) + D$, ahol D az inputtól független konstans.

5.3. Ládapakolás

A klasszikus, egydimenziós ládapakolási feladat a következő. Adott egy úgynevezett $L = (p_1, p_2, \ldots, p_n)$ lista, amelyen alapesetben egyszerűen egy tárgy-halmazt értünk. Itt p_i jelöli a listában szereplő i-edik tárgy méretét, ahol feltesszük, hogy $0 < p_i \leq 1$. Ezeket a tárgyakat kell belepakolni a lehető legkevesebb számú ládába. Bármely ládába legfeljebb 1 összméretű tárgy pakolható.

5.4. Ládafedés

Ládapakolás esetén tehát a ládában lévő tárgyak összmérete kisebb, vagy egyenlő, mint a láda mérete. Ettől eltérően, ha a ládákba pakolt tárgyak összmérete (a láda zárásakor, vagyis a pakolás befejeztével) nagyobb, vagy egyenlő kell hogy legyen mint a láda mérete, akkor pedig ládafedésről beszélünk. A ládafedési feladatok esetén általában a fedett ládák számát maximalizáljuk.

5.5. Ütemezési modellek és algoritmusok

Az ütemezési feladatok esetében feltesszük, hogy adott valahány, valamilyen tulajdonságokkal rendelkező gép, amelyekkel valahány munkát el kell végeztetni úgy, hogy egy adott célfüggvény minimális (vagy pedig maximális) értékét megtaláljuk. Tehát a gépekről, a feladatokról és az ütemezési feladat célfüggvényéről adunk meg információkat ahhoz hogy a modellt definiáljuk.

A gépek száma pozitív egész szám, ami előre rögzített. Egyforma párhuzamos gépek esetén arról beszélünk, hogy a munkák végrehajtása minden gépen egységesen ugyanannyi időbe kerül. A munkák száma szintén pozitív egész szám. A munkák végrehajtási idővel rendelkeznek.

Néhány ütemezési feladatot a későbbiekben bővebben is bemutatunk. Most azért szerepel itt (is), mert a ládapakolási feladat „kombinatorikus értelemben vett duálja"-ként lehet említeni az egyik legegyszerűbb ütemezési feladatot.

Például a $P_m||C_{max}$ feladatban a gépek száma m (pozitív egész szám) és a gépek egyforma párhuzamos gépek. A feladatok száma n, a munkák végrehajtási ideje pedig p_i, ahol $1 \leq i \leq n$. A munka elvégzését nem lehet megszakítani, ha már elkezdtük elvégezni. A munkák ütemezésén azt értjük, hogy a munkákat szétosztjuk a gépek között, minden gépnek csak a hozzárendelt munkát kell elvégeznie, de azt el kell végeznie és nincs várakozási idő két munka elvégzése között. Egy adott gép terhelésének a géphez rendelt munkák végrehajtási idejének összegét értjük. A gép átfutási ideje egyenlő az utolsó munka befejezési időpontjával, feltéve, hogya gépek a 0 időpontban kezdtek el dolgozni.

Az ütemezés teljes átfutási ideje pedig egyenlő a gépek átfutási idejének maximumával. A teljes átfutási időt minimalizáljuk C_{max} esetén. A jelölés tehát három mezőből áll, most a középső mező üres. A harmadik mezőben a célfüggvényről (ami most C_{max}) adtunk meg információt. Az első mező most P_m, tehát m számú úgynevezett párhuzamos gép van.

Az egyik legelső algoritmus a $P_m||C_{max}$ feladatra, a Graham által megadott [36, 37] LPT (= Longest Processing Time) algoritmus. Ez a következőképpen működik. Az első lépésben a munkákat a végrehajtási idejük szerinti csökkenő sorrendbe rendezzük (LPT sorrend), aztán az ütemezést ebben a sorrendben fogjuk végezni. A soron következő munkát mindig úgy ütemezzük, hogy a lehetséges legkorábbi időpontban kezdjük végrehajtani, tehát mindig valamelyik minimális terhelésű gépre kerül. Az LPT algoritmus legrosszabb eset aránya $\frac{4}{3} - \frac{1}{3m}$, vagyis az algoritmus által adott megoldás értéke legfeljebb ennyiszerese az optimumnak. A legrosszabbhoz közeli esetek azonban csak igen kis százalékban fordulnak elő. Dósa bizonyította [21], hogy a legrosszabb eset csak egy nagyon speciális feladathalmaz esetén fordul elő.

Az LS (List Scheduling, vagyis lista szerinti ütemezés) algoritmus esetén pedig egy adott L lista szerinti sorrendben érkező munkákra végezzük az ütemezést, tehát itt hiányzik az LPT első lépése, a sorbarendezés. Az LS algoritmus legrosszabb eset aránya $2 - \frac{1}{m}$, m gép esetén.

5.6. Offline és online modellek

Ládapakolási feladat esetén, offline esetben a teljes tárgyhalmazt előre ismerjük, és a tárgyakat tetszőleges sorrendben pakolhatjuk a ládákba.

Online esetben azonban az L lista szerinti sorrendben kell a tárgyakat pakolnunk, és a listából mindig csak az éppen következő elemet ismerjük. Azt sem tudjuk, hogy lesznek-e még további tárgyak, sok vagy kevés, kicsi vagy nagy tárgyak következnek. Ha elfogynak a tárgyak a listáról, arról természetesen értesülünk, de csak az utolsó tárgy pakolása után. Ütemezési feladatok esetén hasonló a helyzet: offline esetben az inputról

minden információ előre adott. Online esetben pedig az ütemezendő munkák egy L lista szerint egyenként érkeznek, és azonnal ütemeznünk kell őket, a későbbi munkák ismerete nélkül.

5.7. Alsó korlátok, optimális algoritmus

Online feladatok esetén általában nem lehetséges az, hogy valamely algoritmus minden esetben optimális megoldást készítsen. Egy online feladatnak az **alsó korlát**ja bármely olyan ρ konstans, amelyre nem létezik olyan online algoritmus, amelynek a versenyképességi aránya ennél kisebb lenne. Egy online algoritmust akkor nevezünk **optimális**nak, ha az algoritmus versenyképességi aránya egyenlő a feladat alsó korlátjával.

5.8. Félig online (semi online) modellek

Félig online esetben a pakolás/ütemezés előtt már ismert néhány, de nem minden adat. Ilyen ismeret lehet például az, ha előre tudjuk, hogy csökkenő méretben jönnek a tárgyak, vagy az, ha korlátozott mértékben a pakolás után átrendezhetjük a tárgyakat a még nyitott ládákban. Ezért ezek a feladatok az online és az offline esetek „között” helyezkednek el.

A félig online problémákra alkalmazott algoritmusokat félig online algoritmusoknak nevezzük.

Hatékonyságukat versenyképességi analízissel vizsgáljuk. Egy félig online algoritmus optimális, ha a versenyképességi aránya megegyezik a feladat alsó korlátjával. A félig online feladatok esetén érdemes azt is megvizsgálni, hogy valamely további félig online feltétel csökkenti-e egy optimális algoritmus versenyképességi arányát, vagy sem.

A félig online ütemezési feladatok terén a témában az első cikk Kellerer, Kotov, Speranza és Tuza 1997-ben írt cikke [55], amely három félig online változatot vizsgál.

Az első esetben előre ismert a munkák méreteinek összhossza. A második esetben egy puffert használhatunk valahány munka tárolására (ez azt jelenti, hogy néhány munkát

félretehetünk és később dönthetünk az ütemezésükről). A harmadik esetben pedig egyszerre két, egymástól független ütemezést készíthetünk, és végül a jobbikat választjuk. A vizsgálat eredménye az lett, hogy mindhárom esetben egy-egy optimális félig online algoritmus versenyképességi aránya $\frac{4}{3}$. A második fejezet elején részletesebben is áttekintünk néhány félig online ütemezési modellt és eredményt.

5.9. Helykorlátos ládapakolási modellek

Ha egy ládába későbbiekben már nem kívánunk pakolni, azt mondjuk, hogy bezárjuk a ládát, egyébként a láda nyitott. A későbbiekben részletesen ismertetendő Next Fit algoritmus mindig csak egy nyitott ládát használ, ennek következtében persze nem túl hatékony általában. Általában, ha csak véges sok, mondjuk K számú láda lehet nyitva egyidőben, az ilyen modelleket helykorlátos, angolul „bounded space" modellnek nevezzük.

5.10. Abszolút, illetve aszimptotikus approximációs arányok, ládapakolás esetén

Ládapakolás esetén a standard metrika a legrosszabb esetben történő teljesítmény vizsgálatára az aszimptotikus approximációs arány. Az approximációs arány korábbi általános definícióját a ládapakolási feladatra aktualizálva a következőt kapjuk: Egy adott L lista és A algoritmus esetén legyen $A(L)$ azoknak a ládáknak a száma, amelyeket az L listában található tárgyak pakolásához használtunk. Jelölje $OPT(L)$ az L listában található tárgyak pakolásához szükséges optimális számú ládát, továbbá legyen $R_A(L) = \frac{A(L)}{OPT(L)}$. Ekkor az abszolút approximációs arány az alábbi módon definiálható:

$$R_A = \inf\{r \geq 1 : R_A(L) \leq r, \forall L\}.$$

Az aszimptotikus approximációs arány pedig ez lesz:

$$R_A^{\infty} = \inf\{r \geq 1 : N > 0, R_A(L) \leq r, \forall L : OPT(L) \geq N\}.$$

A polinomiális approximációs séma (angol rövidítése: PTAS) olyan algoritmuscsalád, amely minden $\epsilon > 0$ számhoz tartalmaz egy abszolút értelemben vett $(1+\epsilon)$-approximációs algoritmust, amely algoritmus az inputtól függően polinomiális lépésszámú. A teljesen polinomiális approximációs séma (FPTAS) azt jelenti, hogy az algoritmusoktól nem csak azt várjuk el, hogy az input méretében polinomiális lépésszámúak legyenek, hanem azt is, hogy $\frac{1}{\epsilon}$ -tól függően is legyenek ilyenek. Továbbá APTAS, illetve AFPTAS esetén olyan approximációs algoritmusok vannak megadva minden $\epsilon > 0$ értékre, amelyek csak aszimptotikus értelemben $(1+\epsilon)$-approximációs algoritmusok, egyébként az előbbiek teljesülnek rájuk.

5.11. A ládapakolás klasszikus algoritmusai

D. S. Johnson doktori dolgozata több, mára klasszikus ládapakolási algoritmust tartalmaz. Ezek egyike a „**Next Fit**" (röviden NF) algoritmus [52].

Ennek során a tárgyak egy adott L lista szerinti sorrendben érkeznek. Egyszerre csak egy láda lehet nyitva. Ha a soron következő tárgy ebbe belefér, akkor beletesszük, egyébként a ládát bezárjuk, és nyitunk egy másikat, majd ebbe az újonnan megnyitott ládába pakoljuk bele a tárgyat. Ezt a műveletsort addig ismételjük, amíg a lista szerinti sorrendben érkező tárgyak el nem fogynak. Mivel egyszerre csak egy ládát tart nyitva, ezért ez helykorlátos algoritmus, ahol $K = 1$. A „Next Fit" algoritmus implementációja lineáris idő alatt lefut, vagyis a bemeneten lévő tárgyak számának emelkedésével lineáris módon emelkedik a futási idő. Az optimális megoldással összehasonlítva NF-re a következő teljesül: $NF(L) \leq 2OPT(L) - 1$, tetszőleges L lista esetén. A korlát éles, vagyis a 2-es szorzó nem javítható, vagyis az NF algoritmus aszimptotikus approximációs aránya 2.

Johnson másik klasszikus algoritmusa a „**First Fit**”. Ennek során az L lista szerinti sorrendben érkező tárgyakból először veszi az elsőt és beleteszi az egyedüliként megnyitott, de még üres ládába. A következő tárgy érkezésekor megvizsgálja, hogy bele tudja-e pakolni az első nyitott ládába, vagy sem. Ha nem fér bele az első ládába, akkor nyit neki egy másik ládát és abba beleteszi. Bármelyik későbbi, soron következő tárgy esetén is megvizsgálja, az első ládát, hogy a tárgy belefér-e. Ha igen, akkor az elsőbe teszi, ha nem fér bele az elsőbe, akkor megvizsgálja, hogy a második nyitott ládába belefér-e. Ha belefér, akkor beleteszi, ha pedig nem fér bele, akkor nyit neki egy harmadik ládát, és így tovább. Általánosan tehát, a soron következő tárgyat a legelső olyan ládába teszi, ahova belefér. Ha nem talál ilyen ládát, akkor a tárgyat egy új ládába teszi. Ezt a műveletsort ismétli addig, amíg el nem fogynak a tárgyak a listáról.

A First Fit algoritmus aszimptotikus approximációs aránya régtől ismert, és 1.7-tel egyenlő, 1971-ben ugyanis Ullman bizonyította [35, 75], hogy bármely L lista esetén: $FF \leq 1.7OPT + 3$, tehát itt még az additív konstans értéke 3. Az 1976-os [32] cikk szerint pedig

$$FF(L) \leq \lceil \frac{17}{10} OPT(L) \rceil, \tag{1}$$

teljesül, vagyis a '76-os cikk a becslést $FF \leq 1.7OPT + 0.9$-re javította. Az abszolút approximációs arányra először Simchi-Levy [73] adott meg felső becslést, miszerint az legfeljebb 1.75. Ennek az aránynak a becslését Boyar és társai [11] $\frac{12}{7} \approx 1.7143$ -re javították, majd Németh [67] $\frac{101}{59} \approx 1.7119$-re javította tovább ezt az értéket. Az additív konstanst pedig a [80] cikk tovább csökkentette a következő egyenlőtlenség szerint: $FF \leq 1.7OPT + 0.7$.

Végül Dósa és Sgall [25] bizonyította hogy az abszolút approximációs arány éles becslése $FF \leq \lfloor 1.7OPT \rfloor$. Ez azt jelenti, hogy ha az optimális algoritmusnak a tárgyak ládákba pakolásához OPT mennyiségű ládára van szüksége, akkor a „First Fit” algoritmusnak mindig legfeljebb $\lfloor 1.7 \cdot OPT \rfloor$ számú láda kell.

A „**First Fit Decreasing**” algoritmus esetén a tárgyak mérete monoton csökkenő az

L tárgylistában és a tárgyak ebben az L lista szerinti sorrendben érkeznek. Az aktuálisan érkező tárgyat mindig az első olyan ládába pakoljuk, ahova az belefér. A „First Fit Decreasing" algoritmus esetén Johnson az alábbi egyenlőtlenséget bizonyította:

$$FFD(L) \leq \frac{11}{9}OPT(L) + 4. \tag{2}$$

Az additív konstans értékével kapcsolatban 1991-ben Yue [81] igazolta, hogy az legfeljebb 1, majd 2013-ben [23] teljes bizonyítást adott az alábbi egyenlőtlenségre:

$$FFD(L) \leq \frac{11}{9}OPT(L) + \frac{6}{9}, \tag{3}$$

és megmutatta hogy a $\frac{6}{9}$ konstans már nem csökkenthető tovább.

A „**Worst Fit**" algoritmus esetén az először érkező tárgynak nyitjuk az első ládát és abba tesszük. A többi tárgy esetében pedig az aktuálisan érkező tárgyat mindig abba a nyitott ládába tesszük, amelyik az összes nyitott láda közül a legkisebb telítettségű, továbbá az aktuálisan érkező tárgy belefér. Amennyiben nincsen ilyen nyitott láda, akkor nyitunk egy új ládát és az aktuális tárgyat ebbe az újonnan megnyitott ládába tesszük. Johnson igazolta, hogy a Worst Fit algoritmus aszimptotikus approximációs aránya 2.

A „**Best Fit**" algoritmus esetén az először érkező tárgynak nyitjuk az első ládát és abba tesszük. A többi tárgy esetében pedig az aktuálisan érkező tárgyat mindig abba a nyitott ládába tesszük, amelyik az összes nyitott láda közül a legnagyobb telítettségű, továbbá az aktuálisan érkező tárgy bele is fér. Ha nincs ilyen láda, az új tárgy egy új ládába kerül. Dósa és Sgall friss eredménye [26] szerint, $BF(L) \leq \lfloor 1.7OPT(L) \rfloor$ teljesül, és a becslés éles.

Az „**Any Fit**" algoritmus az általánosítása a „... Fit" algoritmusoknak, amely esetén az aktuálisan érkező tárgyat beletehetjük bármelyik olyan nyitott ládába, amelyikbe belefér. Ha nincs ilyen nyitott láda, akkor nyitunk egy új ládát és az aktuális tárgyat ebbe az újonnan nyitott ládába tesszük.

A „**Harmonic Fit**" algoritmus esetén az egységintervallumot K darab I_k $(1 \leq k \leq$

K) részintervallumra osztjuk, ahol $I_k = (\frac{1}{k+1}, \frac{1}{k}]$, $1 \leq k < K$ és $I_K = (0, \frac{1}{K}]$. Az aktuálisan érkező tárgy k típusú, ha a mérete eleme az I_k intervallumnak. Hasonlóan a ládákat is K féle típusba soroljuk, ezután a k típusú ládába csak a k típusú tárgyakat tehetjük bele és mindegyik típusú ládából egyszerre csak egy lehet nyitva. Ekkor minden érkező tárgyat az alábbiak szerint pakoljuk: legyen k az aktuálisan érkező tárgy típusa. Ha van olyan k típusú nyitott láda, ahova az aktuális tárgy belefér, akkor pakoljuk oda, egyébként pedig zárjuk a nyitott k típusú ládát és nyissunk egy új, k típusú ládát és az aktuális tárgyat tegyük ebbe az újonnan nyitott ládába. 1985-ben Lee és Lee [61] bizonyította, hogy a $\lim_{K\to\infty} R^{\infty}_{H_K}$ határérték megközelítőleg 1.69103. Az online ládapakolási algoritmusok között a jelenlegi legjobb a „**Harmonic++**" algoritmus [72], amely 1.588 -as aszimptotikus versenyképességi aránnyal rendelkezik, a jelenlegi legjobb alsó korlátot pedig, amely megközelítőleg $1,5403$, a [4] cikk tartalmazza.

Az offline algoritmusokkal kapcsolatban Fernandez de la Vega és Lueker [77] adták meg az első APTAS-t, továbbá Karmarkar és Karp [54] fejlesztette ki az első AFPTAS-t (aszimptotikus teljes polinomiális idejű approximációs séma).

Csirik és társai 2001-es cikkben [15] mutatták meg, hogy létezik APTAS a klasszikus offline **ládafedés**i problémára, ebben egy LP relaxációt használtak és megfelelő kerekítési eljárást. A módszert finomítva, Jansen egy 2003-as cikkben [51] adott meg AFPTAS-t, csökkentve a műveleti változók számán, így javított a futási időn is.

5.12. Lokális kereső módszerek, metaheurisztikák

Lokális keresésen alapuló módszerek, más néven metaheurisztikák az utóbbi bő 30 évben jelentek meg.

A „**tabu keresés**" olyan lokális keresési módszer, amelynek alapja a mohó keresés (mindig abba az irányba halad tovább, amely lokálisan a legjobb). A memóriában tárolja a régebbi döntések listáját, ez a tabu lista (vagyis a tiltott állapotok listája). Ez a tabu lista véges méretű és a frissítése folyamatos. Abban az esetben, ha az összes jobb megoldás a tabu listán szerepel, akkor elfogad roszabb megoldást is. A tabu algoritmus

megadásakor meg kell adnunk a kezdőállapotot, a kilépési feltételt (ez vagy egy adott lépésszám, vagy ha nem sikerül a célfüggvényen javítani, vagy pedig csak meghatározott mértékben javul adott lépésszám után), a szomszédos állapotokat, a célfüggvényt, a tabu feltételt (olyan feltétel, amely megakadályoz bizonyos állapot átmeneteket) és a tabu listát. Fred W. Glover alkotta meg az eljárást 1986-ban [42], és formalizálta 1989-ben [40, 41]. A tabu keresés alkalmazható például az utazó ügynök probléma megoldására is. (Az utazó ügynök problémában [70] adott néhány város, és ezek páronkénti távolsága, és a feladat az, hogy találjuk meg a legrövidebb olyan utat, amely során az összes várost pontosan egyszer érintjük.)

Az „**evolúciós algoritmus**" (angolul: „Evolutionary Algorithm") [64] olyan sztochasztikus kereső eljárás, amely a probléma megoldását egy evolúciós eljárással keresi [10]. A természetből merítette az ötletet: az evolúcióban a legéletképesebb a túlélő, továbbá ezáltal a jó tulajdonságok öröklődnek tovább. Három fő osztálya létezik, ezek:

- evolúciós programozás
- evolúciós stratégiák
- genetikus algoritmus.

Az „**evolúciós programozás**"-t programkód kifejlesztésére használták fel Fogel és társai 1966-ban úgy, hogy a kódrészleteket mutálták és ezek közül választottak [30].

Később **„evolúciós stratégiák"**-at használtak, pl. Rechenberg 1973-ban repülőgépszárny optimalizáláshoz [68].

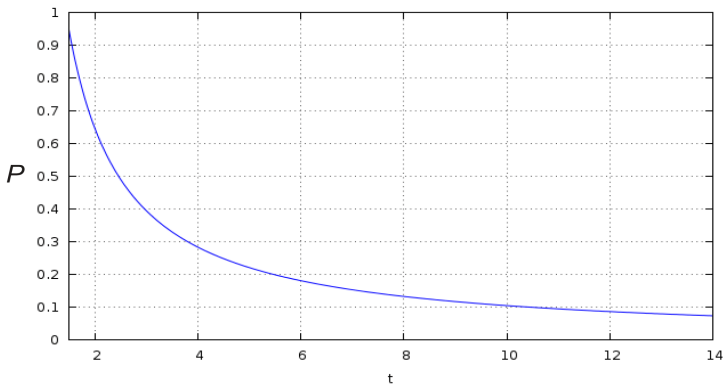
Ezekből fejlődött ki a „**genetikus algoritmus**" módszere, amelynek alapvető műveletei a szelekció, a keresztezés és a mutáció. Az eljárás ciklikusan egyidejűleg több megoldással (egyeddel) dolgozik. Fitnesz függvényt alkalmaz az egyed jóságának kifejezésére. A szelekció során szülőket válogat a populáció egyedei közül. Rekombinációval egy, vagy több szülő felhasználásával utódokat állít elő. Mutációval megváltoztatja az utódokat, majd visszahelyezi egy új populációba úgy, hogy lecseréli az előző populációt. A lecserélés történhet adott arány szerint is (mivel az előző populáció egyedeiből is tarthat meg), vagy

pedig a fitnesszfüggvény alapján dönti el, mik lesznek az új populáció egyedei. John Holland formalizálta először a következő generáció jóságát megjósolni képes keretrendszert számos megelőző kutatási anyag alapján 1975-ben [46].

Thomas Bäck és Frank Hoffmeister 1991-ben írt cikkében tárgyal olyan evolúciós stratégiát, amely felgyorsítja a keresést bizonyos célfüggvények esetén [3]. Brad L. Miller és David E. Goldberg 1996-ban írt cikkében a zajnak a kiválasztás műveletére gyakorolt hatását vizsgálta [65]. Edmund Burke és társai 2010-ben írtak a hiper-heurisztikus kutatásról, amely során a heurisztika kiválasztással és a heurisztika generálással is foglalkoztak [12]. A genetikus algoritmus módszere alkalmazható továbbá dinamikus rendszerek viselkedésének elemzésére is [57].

A „**szimulált hűtés**" egy másfajta lokális kereső módszer [60]. Az elnevezés eredete a szilárd anyagok fizikai hevítési eljárásából származik. Egy kristályos, szilárd anyaggal hőt közölnek, majd engedik, hogy nagyon lassan lehűljön, amíg el nem éri a legszabályosabb kristályrács konfigurációt (vagyis a legkisebb rácsenergia állapotot) és így mentes lesz a kristályhibáktól. Ha a hűtés időzítése elegendően lassú, akkor a szilárd anyag végső konfigurációjában jobb strukturális integritást eredményez. A szimulált hűtés kapcsolatot teremt ezen termodinamikai viselkedés és a diszkrét optimalizációs probléma globális minimumkeresése között, a következőképpen:

Az algoritmus egy megengedett megoldásból indul, eztán véletlenszerűen választ szomszédos állapotot (megengedett megoldást) és ha a kiértékelt célfüggvény jobb, akkor továbblép, ha pedig rosszabb, akkor csökkenő valószínűséggel elfogadja akár a rosszabb állapotot is, ez biztosítja azt, hogy a lokális optimumba nem fog beleragadni, onnan ki tud lépni. Tehát ez is egy véletlenen alapuló keresési technika, ami annyiban tér el a lokális kereséstől, hogy amíg a lokális keresés mohó módon mindig a lokálisan jobb irányba halad, addig a szimulált hűtés során egyre csökkenő valószínűséggel elfogadunk lokálisan rosszabb irányt is, azért hogy így elkerülve a lokális szélsőértékhelyekben rejlő csapdákat, eljuthassunk a globálisan legjobb helyre. A helyenkénti csökkenő valószínűség meghatározásához adott $P(temp) = e^{\frac{-(\beta-\alpha)}{temp}}$ függvényt használunk, ahol az α jelenti az



1. ábra. Példafüggvény szimulált hűtéshez: a keresés hűtési szakaszában ahogy a t idő halad, úgy exponenciálisan csökkenő valószínűséggel fogadjuk el azt a változást, ami ront a célfüggvényen.

aktuális helyet (ahol az algoritmus éppen tart), a β azt a helyet jelöli, ahova a következő lépést megtéve juthatunk és a $temp$ pedig a szimulált hőmérsékletet jelenti. Továbbá teljesül $P(temp) \to 0$, vagyis egyre kisebb (nullához tartó) valószínűséggel fogadunk el olyan változást, ami ront a célfüggvényen. Ezt a módszert Scott Kirkpatrick, C. Daniel Gelatt és Mario P. Vecchi dolgozta ki 1983-ban [56]. A szimulált hűtés módszere alkalmazható például fehérjék energiaszintjéhez tartozó globális minimumhely meghatározásához [79].

A „**hangyakolónia**” alapú lokális keresést a számítástudományban és az operációkutatásban is használják. Valószínűségszámítási technikákon alapuló módszer, amivel eléggé hatékonyan megoldhatóak olyan számítási feladatok, ahol gráfokban kell megkeresni a „jó” útvonalakat. Az alapötlet a természetben is előforduló hangyakolónia viselkedésén alapszik. A hangyák véletlenszerűen keresgélik a táplálékot, majd amikor az egyikük talál élelmet, akkor visszamegy a hangyakolóniába a talált élelemmel, miközben feromonnyomokat hagy maga után. A többi hangya pedig, amint találkozik egy olyan útvonallal, ami tartalmaz feromont, akkor végigmegy rajta, remélve, hogy talál még ott élelmet. Ha talál, akkor szintén feromonnyomokat bocsát ki, miközben hazaviszi a talált táplálékot és így egyre erősödik a feromonnyom illata. Ha már nincs több táplálék, akkor nem erősítik az útvonal illatát és így idővel egyre kevésbé lesz érezhető, míg végül teljesen

eltűnik. Ez az eltűnés voltaképpen lehetővé teszi azt, hogy a keresés során ne ragadjanak be egy lokális optimális megoldásba. Denebourg és kollégái vizsgálták először a hangyák viselkedését 1983-ban [16]. A hangyakolónia módszer alkalmazható például elektromos áramkörök tervezéséhez [83].

A „**raj intelligencia**" a decentralizált, önszerveződő (természetes, vagy mesterséges) rendszerek viselkedésén alapszik. A rendszer egyszerű egyénekből áll, akik egyszerű szabályokat követnek és habár nincs központi kontroll ami megmondaná az egyedeknek, hogyan viselkedjenek, mégis a köztük létrejövő interakciók egy egységesnek mondható intelligens viselkedést alkotnak. Ezért a rajintelligencia magába foglalja például a hangyakolónia alapú lokális keresést is. A rajintelligencia kifejezést először Gerardo Beni és Jing Wang ismertették 1989-ben [6]. A rajintelligencia módszer alkalmazható például rákellenes nanorobotok kifejlesztésénél a rákos sejtek eltávolítására [62].

5.13. Optimalizálás gráfokban

Mivel dolgozatunkban gráfokban történő optimalizálás is szóba kerül, nagyon röviden itt ennek bevezetésé-vel is foglalkozunk.

A $G = (V, E)$ rendezett párt **gráf**nak nevezzük [38], ahol a V halmaz a csúcspontokat, az E halmaz pedig a csúcspontokat összekötő éleket tartalmazza. A gráf rendje egyenlő a csúcsainak számával ($|V|$), a gráf mérete pedig egyenlő az éleinek számával ($|E|$). A gráf lehet irányított (az él egy adott csúcsból indul ki és egy adott csúcsba mutat), vagy irányítatlan, súlyozott (az élekhez vagy csúcsokhoz nemnegatív súlyokat rendelünk) vagy súlyozatlan, egyszerű (irányítatlan, csak egyszeres élekkel rendelkezik és nem tartalmaz kört), teljes (minden csúcsból megy él minden csúcsba) vagy többszörös élekkel is rendelkezhet két csúcspont kötött.

A gráfban lévő **út** a következő: egymáshoz csatlakozó élek véges sorozata. Ha a gráf súlyozott, akkor a gráfban lévő u és v csúcspontok közötti legrövidebb út az u és a v csúcspontokat úgy köti össze élekkel, hogy a két csúcspontot összekötő éleket tartalmazó élhalmaz éleinek összsúlya a lehető legkevesebb.

A gráfban lévő **fa** összefüggő és körmentes részgráfot alkot. A **minimális feszítőfa** pedig olyan fa a gráfban, amelynek élei a gráfban lévő összes csúcspontot összekötik olyan élekkel, amely éleknek az összsúlya a lehető legkevesebb. Irányított esetben azt a csúcsot, amibe nem megy él, de onnan megy tovább él, **gyökér**nek nevezzük. Azt a csúcsot pedig amelybe megy él, de onnan már nem indul ki további él, **levél**nek nevezzük. Azt az irányítatlan gráfot, amely egy, vagy több fából áll, **erdő**nek hívjuk.

A **gráfok színezése** azt jelenti, hogy színeket (számokat) rendelünk a gráf csúcsaihoz (vagy éleihez). A gráfot akkor színeztük ki „jól", ha az összekötött csúcsok (illetve érintkező élek) eltérő színűek és a színezéshez a lehető legkevesebb színt használtunk.

A gráfokban egy adott kezdőcsúcsból kiindulva megkereshetünk egy csúcsot úgy, hogy a kezdőcsúcsból átlépünk az egyik szomszédos (vagyis a kezdőcsúccsal összekötött) csúcsra és innen tovább egy másik szomszédos csúcsra, amíg meg nem találjuk a keresett csúcsot. Az alapján, hogy melyik szomszédos csúcsot választjuk, megkülönböztetünk szélességi és mélységi keresést is. **Mélységi keresés**nél addig nem lépünk vissza egy már érintett csúcsra, amíg el nem értünk egy levélig. **Szélességi keresés**nél pedig először az azonos szinteken lévő csúcsokat érintjük, majd ha már nincsen több csúcs az adott szinten, akkor megyünk eggyel tovább a következő szintre (az első szinten a gyökértől egy élnyi távolságra vannak a csúcsok, a másodikon pedig kettőre és így tovább). **Dijkstra** 1959-ben [17] kifejlesztett egy olyan mohó algoritmust, amivel irányított, vagy irányítatlan gráfokban lehet megkeresni a legrövidebb utakat egy adott csúcspontból kiindulva, nemnegatív élsúlyok mellett. Land és Doig 1960-ban [59] kifejlesztett egy **korlátozás és szétválasztás** (Branch and Bound) nevű algoritmust, amely egy általános kereteljárás, amelynek különböző konkrét alkalmazásai igen hatékonyan képesek megoldani kombinatorikus optimalizálási feladatokat (pl. az utazó ügynök problémát is).

5.14. NP-teljes feladatok

A **P osztály** a polinom időben megoldható problémákat tartalmazza, vagyis minden olyan problémát, amelyhez létezik olyan k konstans, hogy ha a probléma n hosszú be-

menetből áll, akkor az $O(n^k)$ idő alatt megoldható. Az **NP osztály** tartalmazza azokat a problémákat, amelyek polinom időben ellenőrizhetőek (vagyis a megoldás helyességét polinomiális időben le lehet ellenőrizni). Lásd: [33, 34]. Minden P-beli probléma az NP osztályba is beletartozik. Azonban az máig nyitott kérdés, hogy P osztály valódi részhalmaza-e az NP osztálynak [31]. Az **NP-teljes** problémák osztályába azok a problémák tartoznak, amelyek legalább olyan nehezek, mint bármely NP-beli probléma. Ha az NP-teljes problémák közül akár csak egy is megoldható polinomiális időben, akkor minden NP-beli probléma is megoldható polinomiális időben. Számos NP-teljes feladat létezik, ilyen például a gráfszínezés, minimális feszítőfa, 3-partíció probléma, ládapakolás és hátizsák feladatok.

Partíciós problémának nevezzük azt a feladatot, amikor el kell dönteni, hogy egy adott pozitív számokból álló véges halmaz két részre osztható-e úgy, hogy az első halmazban lévő számok összege egyenlő a második halmazban lévő számok összegével. A **3-partíciós probléma** esetén adott egy $3n$ darab számot tartalmazó halmaz, és feltesszük, hogy a számok mindegyike $\frac{1}{4}$ és $\frac{1}{2}$ közötti szám. Azt kell eldönteni, hogy kiválasztható-e n darab hármas úgy, hogy bármely hármas összege éppen 1. A 3-partíciós problémára nincsen pszeudo-polinomiális idejű algoritmus, kivéve, ha $P = NP$ [33]. A partíciós probléma általánosítása a **ládapakolási probléma**.

A bevezetés végén megemlítjük még, hogy minden bizonyítás végén, a bizonyítás lezárásaképpen a következő jelölést használjuk: ∎ Hasznos szimbólum, jelentése: „Bizonyítás vége, mert ezt kellett bizonyítani”[1].

Eredményeinket az alábbi tudományos közleményekben publikáltuk:

A doktori értekezés témájához kapcsolódik a következő három SCI folyóiratcikk [8, 14, 78], egy magyar nyelvű cikk [5], valamint a következő két konferencia kiadvány: [9, 19], és a [7] kézirat.

[1]Eredetileg Euklidész használta bizonyításai lezárásaképpen a „*οπερ εδει δειξαι*" (hóper édei deîxai) kifejezést, rövidítve: $O.E.\Delta$. Az 1600-as évektől latinul is használták: „*quod erat demonstrandum*", rövidítve: Q.E.D. Ezzel jelölték, hogy a bizonyítás teljessé vált, mert elérték a szükséges eredményt.

6. Ládapakolás (illeve fedés) szállítással

E fejezetben bevezetünk egy újfajta feladatot, amelyben ládapakolás és szállítás egyszerre történik. Korábban is foglalkoztak már hasonló feladattal, lásd Iványi és Kaitar [50] cikke, amely egy hússzállítással kapcsolatos gyakorlati problémát, illetve annak megoldását tárgyalja. Ebben a cikkben a szállítási és pakolási offline költség minimalizálása a cél. Ez a feladat az alább ismertetendő 1. modell azon speciális esete, amikor a C listában a ládák minden lehetséges pakolásához megtalálható egy láda a pakolási költsége plusz a szállítási költség (a megfelelő csúcsok közötti minimális Hamilton-kör hosszának b-szerese).

Hasonló, de ütemezés és szállítás kombinációját tárgyaló cikk például [84].

Először tehát néhány lehetőséget veszünk sorra, amelyek mindegyikében valamiképp a ládapakolás és a szállítás kombinálható. Ezek rokon feladatok, de ezek közül majd csak egyet fogunk a későbbiekben vizsgálni. Az alább ismertetett hét modell megtalálható a [1] publikációban. Az új feladat megoldása során egy új megoldási módszert, illetve lehetséges megközelítést is bemutatunk. Lássunk tehát néhány lehetőséget arra, ahogy definiálhatunk egy olyan feladatot, amely ládapakolási/fedési feladat és szállítási feladat egyszerre.

I. A tárgyak online módon, egy $L = (p_1, \ldots, p_n)$ lista szerint érkeznek, ahol a t-edik tárgy mérete p_t minden $1 \leq t \leq n$ indexre. A tárgyakat ezen L lista szerint pakoljuk egyenként, az egyszerűség kedvéért feltesszük, hogy a tárgyak egységnyi idő elteltével érkeznek egyesével, tehát a t-edik tárgyat a t-edik időpontban pakoljuk bele valamely ládába. Továbbá adott egy másik lista: C, ami a fedett ládákra vonatkozó szállítási költségeket tartalmazza. Ez a C lista pedig a c_t értékeket tartalmazza, ahol a c_t érték a t időpontban zárt és fedett ládákra vonatkozó szállítási költséget jelenti.

A modell lehet online, vagy offline. Az online esetben nem ismerjük előre a listákat. Az offline esetben minden információt előre tudunk a C és az L listákról, de a pakolásnak a listák szerint kell történnie. Ez azt jelenti, hogy minden olyan esetben, amikor egy új tárgy érkezik, azonnal pakolnunk kell azt az új tárgyat egy olyan ládába, ahova az belefér, vagy nyitnunk kell egy új ládát. Ekkor lehetőségünk van arra is, hogy bezárjunk

néhány ládát, ha szeretnénk. Ha bezárjuk a ládát éppen az után, miután felfedtük a következő tárgy méretét és belepakoltuk a ládába, akkor azonnal szállítjuk a lezárt ládát és kifizetjük a c_t szállítási költséget. Lehetőségünk van arra is, hogy ebben az időpontban több ládát is lezárjunk. Az is megengedett, hogy egyetlen láda se legyen lezárva, vagyis az éppen érkező tárgyat beletesszük abba a ládába, ahova az belefér, és nyitva hagyjuk az összes ládát. Ezt azért tesszük, mert ha idővel csökken a szállítási költség, akkor arra időzítjük a ládák lezárását és szállítását. Amikor vége a sorozatnak, az összes nyitott ládát szállítani kell az aktuális (végleges) szállítási áron. A cél az, hogy a teljes szállítási költséget minimalizáljuk. Megjegyezzük, hogy abban az esetben, ha $c_t = 1$ minden t esetén, akkor az online esetben a tisztán online ládapakolási feladatot kapjuk, ami azt jelenti, hogy ez a feladat az általánosítása az online ládapakolási feladatnak.

II. Hasonló fedési feladat adható meg a következő módon: a t-edik tárgy a t időben érkezik. Amikor az I_t tárgy megérkezik, akkor azt a tárgyat azonnal bele kell pakolnunk egy ládába. A t-edik időpontban az is lehetséges, hogy ha akarjuk, akkor szállítjuk a fedett ládákat, ekkor c_t jelenti az aktuális, ideiglenes hasznot, amit a láda szállítása (és a benne lévő tartalom értékesítése) után kaptunk. Csak a fedett ládákat engedjük meg szállítani, és a haszon aktuális értékét kapjuk meg. A célunk ebben az esetben az, hogy a szállított ládák után kapott hasznot maximalizáljuk. Abban az esetben, ha a pakolások és szállítások után még marad olyan láda, amit nem zártunk le, akkor azok után nem kapunk hasznot, mert nem szállíthatjuk azokat. Ez a feladat is az általánosítása a tisztán online ládafedési feladatnak abban az esetben, ha $c_t = 1$, minden t esetén.

III. Abban az esetben, ha a költség/haszon függvény lineáris (vagyis $C(t) = At + B$, ahol: $A, B \geq 0$), akkor egy speciális, ám még mindig érdekes változatát kapjuk az előbb definiált feladatoknak, ugyanis ekkor a költség/haszon függvény jövőbeli változása előre ismert. További érdekes változat az, ha a $C(t)$ függvény szakaszonként konstans függvény. Például C_1 hasznot kapunk (vagy C_1 költséget fizetünk), ha a láda fedett és szállított egy előre meghatározott T idő előtt és csak $C_2 < C_1$ hasznot kapunk abban az esetben, ha ez az előre meghatározott T idő már letelt.

IV. Az I. feladathoz hasonló (de azzal nem azonos) feladat lehet a következő: bármely t időpillanatban veszünk egy ládát c_t áron, és ezt a ládát használhatjuk arra, hogy az aktuális tárgyat belepakoljuk és a következő tárgyakat is, ha belefférnek. Kezdetben nincsen nyitott láda, továbbá egy ládát minden bizonnyal meg kell vennünk c_1 áron, de utána már eldönthetjük azt, hogy akarunk-e venni még ládát az aktuális áron, vagy az aktuális tárgyat a már megnyitott láda, vagy ládák egyikébe pakoljuk, ha az belefér valamelyikbe. A célunk az, hogy a lehető legkisebb költséggel pakoljuk az összes tárgyat a ládákba. Abban az esetben, ha $c_t = 1$ minden t esetén, akkor a tisztán online ládapakolási feladatot kapjuk.

V. A következő egy ládafedési feladat. Minden fedett láda után kapunk hasznot, de a ládafedéshez szükséges idő növekedésével csökken a ládaszállításakor kapott haszon. Bevezetünk tehát egy $G(k) \geq 0$ monoton növekvő büntetőfüggvényt, ahol $k \geq 0$. Ha valamely ládába pakolt tárgyak minimális és a maximális indexeit kivonjuk egymásból, akkor megkapjuk a büntetőfüggvény k argumentumát. A célfüggvény a fedett ládák száma minusz az összes büntetés. Ekkor tehát célunk az, hogy minél több ládát fedjünk és ezzel egyidőben az összes büntetést minimalizáljuk. A két cél nyilván „egymás ellen dolgozik". Abban az esetben, ha a $G(k) = 0$, akkor a tisztán online ládafedési feladatot kapjuk.

VI. A tárgyak ismét online módon érkeznek. Amint egy tárgy megérkezik, azt azonnal pakolnunk kell az egyik ládába, kivéve, hogy van egy K méretű pufferunk, amelynek segítségével legfeljebb K darab tárgy pakolása késleltethető. Tehát itt tárolhatunk ideiglenesen legfeljebb K darab tárgyat, ezeket végül majd szintén el kell pakolni, és amely tárgy egyszer már kikerült a pufferből, az oda már nem tehető vissza.

Az eljárás végén, ha már nem érkezik több tárgy, akkor az esetleg még pufferben maradt tárgyakat is bele kell pakolnunk a ládákba. A tárgyak várakozási idejét büntethetjük is: ha zárunk egy ládát és szállítjuk, akkor $G(k) \geq 1$ költséget kell fizetnünk a szállított láda után, ha $k \leq K$ tárgy van jelenleg a pufferben. A cél az, hogy az összes tárgyat úgy pakoljuk a ládákba, hogy az összköltség minimális legyen. Ha a

$G(k) = 1$, akkor az online ládafedési feladatot kapjuk. Ha $G(0) = 1$ és G monoton növekvő, akkor az előbbinek egy általánosítását kapjuk, ahol a puffer egyrészt segíti a „jó" pakolás elvégzését, másrészt ennek használata pénzbe kerül.

VII. Végezetül definiálunk még egy fedési feladatot: egyszerre legfeljebb csak K darab ládát tartunk nyitva. Ha fedtünk egy ládát, akkor azt azonnal szállítjuk. Itt a pakolási eljárás késése az egyszerre nyitva tartott ládák számával arányosan növekszik. Vagyis ha pontosan $1 \leq k \leq K$ darab láda van nyitva abban a pillanatban, amikor valamely nyitott láda fedetté válik, akkor a $G(k) \geq 0$ monoton csökkenő haszonfüggvény alapján számított hasznot fogunk kapni a fedetté vált láda zárása és szállítása után. A cél az, hogy maximalizáljuk az összes hasznot. A továbbiakban részletesen csak ezzel a modellel fogunk foglalkozni.

A következő fejezetekben (ládafedés szállítással, offline és online esetek) tárgyalt eredményeink az [5, 7, 8, 9] publikációinkon alapulnak.

6.1. Valós alkalmazás

Egy kisebb konzervgyárban kézzel pakolják a gyümölcsöt dobozokba, a gyümölcs egy ablakon keresztül érkezik, egy rakodómunkás van az ablak túloldalán, aki ezt a feladatot végzi. *Minden ládába legalább adott s összsúlyú gyümölcsöt kell pakolni*, de nyilván nem érdemes többet pakolni, mint amennyit muszáj. (Tehát ez egy fedési feladat.) Természetes azt feltételezni, hogy a pakoló munkás nem tud túl sok ládát egyszerre kezelni, továbbá minél több ládával dolgozik, ez annál több időt igényel tőle, és a feladat egyre bonyolultabb lesz számára. Egy láda lehet: nyitott, vagy zárt. Valamely ládát akkor tekintünk nyitottnak, ha még pakolni kívánunk bele. A láda zárt, ha már nem pakolunk bele többet (mert már van benne s összsúlyú tárgy). Tehát a következő észrevételeket tehetjük: Egyrészt, igyekeznie kell minél jobb fedést generálnia (vagyis mindegyik láda legyen telepakolva, de ne túlságosan: a ládába pakolt gyümölcsök összsúlya legyen legalább s, de azt ne nagyon haladja meg), ami nem könnyű feladat, minél kevesebb nyitott láda van, annál nehezebb. Másrészt mivel a gyorsaság is számít,

érdeke, hogy egyszerre lehetőleg kevés számú láda legyen nyitva, vagyis kevés lehetőség közül kelljen választania. Az előbbi két érdek egymásnak ellentmond, a dobozba pakoló személy célját ezért úgy szimulálhatjuk, hogy minden fedett ládáért pénzt kap (emiatt érdemes sok ládát telepakolnia, vagyis jó fedést készítenie), de másrészt, ahogy a nyitott ládák száma növekszik, egyre kevesebb pénzt kap egy-egy fedett ládáért (vagyis érdemes kevés nyitott ládával dolgoznia).

6.2. Alapfogalmak

A továbbiakban először megismételjük a lényegesebb fogalmakat, amelyeket a feladat megoldása során használunk. Majd bemutatunk néhány természetesen adódó algoritmust az előbb definiált problémára.

Ládapakolás: Adott p_i (ahol: $i = 1, \ldots, n$) méretű tárgyakat kell pakolunk minimális számú ládákba úgy, hogy a ládákba pakolt tárgyak összmérete nem lehet nagyobb, mint a ládák kapacitása, amit egységnyinek veszünk.

Ládafedés: A ládát fedettnek tekintjük, ha a ládába pakolt tárgyak összmérete legalább akkora, mint a láda kapacitása.

Ládaszállítás: A fedett ládát azonnal szállítjuk (több tárgy ebbe a szállított ládába már nem pakolható).

Haszonfüggvény: A szállított ládák után kapott haszon: $G : \{1, \ldots, K\} \to \Re$, ahol k a fedett láda szállításakor az egyszerre megnyitott ládák száma ($1 \leq k \leq K$). A $G(k)$ függvényről feltesszük, hogy pozitív, monoton nem növekvő függvény. Ilyen célfüggvény például a $G(k) = 10.1 - 0.1 \cdot k$ függvény, ami $K = 3$ esetén: $G(1) = 10, G(2) = 9.9, G(3) = 9.8$, vagyis itt a fedett láda után kapott haszon éppen hogy csak egy kicsit csökken, ahogy a nyitott ládák száma nő.

Ládafedés szállítással: Az új feladat, amivel a disszertációban foglalkozunk. Ez a feladat a ládafedési feladat általánosítása. Ebben az új feladatban nem csupán a ládák pakolásának jósága alapján minősítjük a feladatot megoldó algoritmusunkat, hanem az a célunk, hogy a szállított ládák után kapott haszon maximális legyen.

6.3. Jelölések

A következő jelöléseket fogjuk alkalmazni.

L: lista, ami a pakolandó tárgyak méretét sorban tartalmazza.

p_i: az i-edik pakolandó tárgy mérete.

$G(k)$: haszonfüggvény, ahol: k az aktuálisan nyitott ládák száma.

I: input, az adott A algoritmushoz tartozó adatok, amelyek az induláshoz szükségesek (pl.: L és $G(k)$).

DNF: Duál Next Fit algoritmus, mindig csak egy nyitott ládába pakolja a tárgyakat. Amint megtelt egy láda, azt bezárja, és nyit egy újat a következő tárgynak.

$H(K)$: Harmonic algoritmus, mindig csak a hasonló mérettartományba tartozó tárgyakat pakolja egymáshoz.

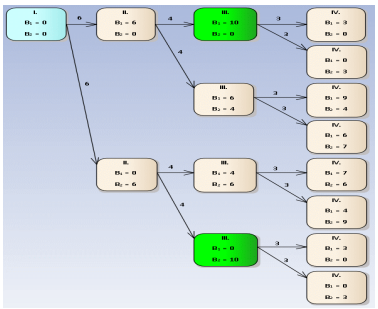
$SH(K)$: Smart Harmonic algoritmus, a $H(K)$ javított változata, később definiáljuk.

$Mask(\alpha, \beta, K)$: Maszk meta-algoritmus, egy új, általunk kifejlesztett algoritmuscsalád.

6.4. Offline eset

Először a feladat offline változatával foglalkozunk, vagyis előre ismert az L lista, amely szerint a tárgyak érkeznek. Azonban továbbra is fennáll, hogy a tárgyakat e lista szerinti sorrenden kell pakolni!

Közismert tény, hogy mind a ládapakolási, mind a ládafedési feladat megoldása NP-nehéz, vagyis az offline optimális megoldás megtalálásához exponenciálisan sok lépés is szükséges lehet. Sok esetben azonban nem áll rendelkezésünkre ennyi idő, viszont megelégszünk a feladatnak elég jó, közel-optimális megoldásával is. Az online esetben (amikor a tárgyak egyesével jönnek, és amint megjelenik a következő tárgy, azonnal kénytelenek vagyunk azt valamely ládába pakolni, nem ismerve az input további részét) általában még nehezebb a feladat, tehát még olyan jó megoldást sem tudunk konstruálni, mint az offline esetben.



2. ábra. Pakolási lehetőségek 3 tárgy és 2 láda esetén.

Az offline eset megoldását szemléltetjük az alábbi feladaton: pakoljuk a tárgyakat legfeljebb 2 ládába, ahol a ládaméret 10 egység. Legyen $L = \{6, 4, \ldots\}$, vagyis az első két tárgy mérete 6, ill. 4.

A 2. ábra megadja azt az irányított, összefüggő és körmentes (fa-)gráfot, ami az összes lehetséges pakolási lépést tartalmazza. A gráf gyökere (az első csúcs, amelyből kiindul él, de oda nem vezet él) a kiindulási helyzetnek felel meg, ahol még egyetlen tárgyat sem pakoltunk és a ládák üresek. Innen azért ágazik tovább két csúcs, mert az első érkező tárgyat (p_1) vagy az első (B_1), vagy pedig a második (B_2) ládába pakoljuk. Tehát két lehetőség van, az algoritmus a tárgyat az első ládába, vagy a második ládába pakolja. A gráf levelei (azok a csúcsok, amelyekből nem indul ki él) adják meg azt, hogy a második tárgy (p_2) az első, vagy a második ládába kerül. Mivel a megoldások kiértékelésénél a gráf leveleit kell összehasonlítanunk, ezért ha meg szeretnénk találni a legjobb pakolást, akkor *gráf alapú kereső algoritmust* alkalmazhatunk a ládapakolási algoritmusok lépéseit tartalmazó fa-gráfon.

Mivel azonban a feladat mérete nagyon nagy lehet, vagyis akár 5000 tárgy és 100 láda is előfordulhat, a gráf nagyon nagy méretű lenne.

Ha a tárgyak nem sorba rendezetten helyezkednek el az L listában, az offline optimum értékének kiszámítása **NP nehéz**, mert e feladatnak részfeladata a *3*-Particio probléma. Amennyiben G konstans függvény, akkor a klasszikus (NP nehéz) ládafedési problémát kapjuk, ha G nem konstans, akkor a feladat megoldása még nehezebb lehet.

Az alábbi tétel segítségével azonban elég pontosan jellemezhető az általunk definiált „ládafedés szállítással" feladat. Lényegében az derül ki, hogy az offline feladatot *hatékonyan meg tudjuk oldani*, még az általunk definiált általános esetben is, *ha az összes tárgyméret „elég nagy"*. Viszont általában az offline feladat megoldása is nehéz, nem remélhető, hogy polinomiális futásidejű algoritmussal, az optimum értékének legalább a $\frac{6}{7}$-ét megkapjuk, tetszőleges input esetén. Más szóval, nem létezik APTAS a feladatra, feltéve hogy $P \neq NP$.

Az offline optimum meghatározásával kapcsolatban tehát az alábbiakat állíthatjuk:

1. tétel. *Legyenek K és b adott pozitív egészek, $G : \{1, \ldots, K\} \to \Re^+$ tetszőleges haszonfüggvény és $c > 0$ adott valós konstans. Ekkor:*
(i) Ha az input összes elemére teljesül az, hogy az elemek mérete legalább c, akkor bármely adott L listára az offline optimum értéke polinomiális időben meghatározható.
(ii) Továbbá ha az inputra még az is igaz, hogy legfeljebb b különböző méretű elem van benne és egyik elem mérete sem kisebb, mint c, akkor bármely adott L-re az offline optimum lineáris időben is meghatározható.

Bizonyítás. Tekintsünk egy tetszőleges pakolási eljárást, ami megadja a feladat megoldását, ezzel együtt tehát a B_1,...,B_m ládákat is. Mivel bármely pillanatban legfeljebb K láda lehet egyszerre nyitva, ezért létezik egy $\lambda : \{1, \ldots, m\} \to \{1, \ldots, K\}$ cimkézés úgy, hogy nincs két egyforma cimkéjű láda egyidőben nyitva. Miután adott ez a hozzárendelés, minden olyan tárgy, amit a B_j ládába pakoltunk kaphat egy $\lambda(B_j)$ cimkét. Innentől fogva minden lehetséges pakolási eljárás meghatároz egy sorozatot $\{1, \ldots, K\}^n$-ben; és fordítva is, minden ilyen sorozat egyértelműen meghatároz egy pakolási eljárást. Annak ellenére, hogy a sorozatok lehetséges maximális száma K^n, belátjuk, hogy OPT kiszámítható polinomiális (vagy akár lineáris) számú lépésben.

Jegyezzük meg, hogy egy megoldás értéke kiszámítható a hozzá tartozó sorozatból. Valóban, lépésről lépésre megadható a nyitott ládák tartalma, ezáltal azonosítható a pillanat, amikor egy láda fedetté válik. Megnézhető, hogy abban az adott pillanatban mennyi az egyszerre megnyitott ládák száma és végül ezáltal kiszámítható a G-hez tartozó

összérték.

Ráadásul minden egyes lépésnél az aktuális helyzethez tartozó szükséges információ jellemezhető az előzőekben lezárult ládákhoz tartozó G értékek összegével és a nyitott ládák jelenlegi tartalmával (telítettségi szintjével). Fő észrevételünk az, hogy minden egyes lépésben ezen jellemzők lehetséges száma az n polinomjával felülről korlátos az (i) feltétel teljesülése esetén, és ez a szám konstanssal korlátozható felülről, az (ii) feltétel esetén. Nem áll szándékunkban az előbb említett felső korlátokat pontosan meghatározni, csak egy durva felső becslést adunk, de ez még mindig elegendő lesz a tétel bizonyításához.

(i) bizonyítása. Legyen $m = \left\lfloor \frac{1}{c} \right\rfloor$. Ekkor bármely $m+1$ tárgy teljesen fed egy ládát, ezért legfeljebb $M = \sum_{i=1}^{m} \binom{n}{i}$ különböző részhalmaz lehet a tartalma bármely nyitott, de még nem fedett ládának. Igazak a következőek is: $m \leq \frac{1}{c}$ és $M = (1+o(1))\binom{n}{m} = \frac{(1+o(1))n^{\frac{1}{c}}}{\left\lfloor \frac{1}{c} \right\rfloor !}$, hiszen $n \to \infty$ (c és m rögzítettek).

Az eljárás bármelyik lépésében, ha i számú láda van nyitva, akkor a tartalmukat legfeljebb $\binom{M}{i} = \frac{(1+o(1))n^{\frac{i}{c}}}{(i!\left\lfloor \frac{1}{c} \right\rfloor !^i)}$ különböző lehetséges módon választhatjuk meg, $\binom{K}{i}$ módon lehet kiválasztani az aktuális cimkéket és $i!$ módon lehet ezeket a ládákhoz rendelni. Így a legfeljebb K számú nyitott ládához legfeljebb $M^* := \sum_{i=0}^{K} \binom{M}{i}\binom{K}{i} i! \leq (1 + o(1)) \sum_{i=0}^{K} \binom{K}{i} \left(\frac{n^{\frac{1}{c}}}{\left\lfloor \frac{1}{c} \right\rfloor !} \right)^i \leq ((\left\lfloor \frac{1}{c} \right\rfloor)!^{-K} + o(1))n^{\frac{K}{c}}$ -féleképpen tartozhat valamely címke és részhalmaz, minden egyes lépésben. Mivel K és c konstans, az n kitevője felülről korlátos és a szorzója meglehetősen kicsi.

Ezen megfigyelések alapján egy hatékony, dinamikus programozás jellegű algoritmust készíthetünk. A t-edik lépés kezeli a p_t tárgyat. $t = 1, 2, \ldots, n$-re ki fogjuk számítani az aktuálisan nyitott ládák összes lehetséges tartalmát és ezen tartalmakhoz tartozó maximálisan elérhető hasznot (amely az előzőekben már elszállított ládákhoz tartozó G értékek összege).

Kezdetben az összes láda üres. Ha k_t láda van nyitva valamely konfigurációban a p_t tárgy kezelése után, akkor k_t választási lehetőségünk van, hogy a p_{t+1} tárgyat a nyitott ládák egyikébe tegyük; továbbá ha $k_t < K$, akkor van egy olyan lehetőségünk is, hogy p_{t+1} pakolásához egy további ládát nyissunk. Ez lehet hogy növeli, vagy nem növeli a

konfiguráció jelenlegi hasznát attól függően, hogy valamely nyitott láda fedetté válik, vagy sem. Így nem több, mint M^* különböző konfiguráció fordulhat elő. Néhányuk lehet, hogy a t lépés többféle konfigurációjából is adódik; mindegyikre választunk egyet, amelyikre a legnagyobb hasznot kapjuk az adott konfigurációval a p_{t+1} tárgy elhelyezése után és az új konfigurációhoz megjegyezzük az elődjét.

Miután az összes eloszlást megadtuk p_n-hez, az offline optimumot a legnagyobb haszonnal értük el. A fenti számításból következik, hogy az algoritmus $O(n^{1+\frac{K}{c}})$ futásidejű, ahol az O-ban elrejtett konstans meglehetősen kicsi.

(ii) bizonyítása. Legyen ismét $m = \left\lfloor \frac{1}{c} \right\rfloor$, ami a nyitott, de még nem szállított ládában lévő tárgyak számának egy felső korlátja. Ha egy nyitott láda pontosan j tárgyat tartalmaz ($1 \leq j \leq m$), a láda jelenlegi telítettsége meghatározható a tárgyak számának eloszlásából, amely egyike a legfeljebb $\binom{b-1+j}{j}$ különböző kombinációnak a lehetséges különböző b számú értékből (ami j rendezetlen elem m számosságú halmazából történő ismétléses választásával adódik). Ezért egy nyitott láda telítettsége nem vehet fel több értéket 0 és 1 között, mint: $\sum_{j=1}^{m} \binom{b-1+j}{j} < \binom{b+m}{m}$.

Ez összességében azt jelenti, hogy i ($i = 1, \ldots, K$) cimkézett és nyitott láda esetén kevesebb, mint $\sum_{i=1}^{K} \binom{b+m}{m}^i \binom{K}{i} i!$ különböző eset van bármely t-re. Ezért a fent bemutatott algoritmus esetén, minden egyes lépésben korlátos számú telítettség eloszlást vizsgáljunk meg. Következésképpen a p_n-hez tartozó legjobb G érték $O(n)$ időben határozható meg. Eszerint n lépés alatt visszakereshető az a sorozat $\{1, \ldots, K\}^n$-ben, ami felveszi az optimumot. Ez a sorozat meghatároz egy optimális pakolási eljárást. ∎

Ezek után megmutatjuk, hogy nem létezik asszimptotikus polinomiális idejű approximációs séma ($APTAS$) az offline problémára (az triviális hogy PTAS nem lehetséges). Ezzel kapcsolatban az alábbi állítást fogalmazzuk meg:

2. tétel. *Tetszőleges $K \geq 2$ választás esetén, kellően megválasztott G haszonfüggvény esetén létezik olyan L listák osztálya, amely esetén nem létezik olyan polinomiális idejű algoritmus, ami $\frac{6}{7}$-nél jobb (abszolut vagy aszimptotikus) approximációs aránnyal rendelkezik, feltéve hogy $P \neq NP$.*

Bizonyítás. Megmutatjuk, hogy a következő választás megfelelő (az utolsó rész csak akkor alkalmazható, ha $K \geq 3$): $G(1) = 1$, $G(2) = \frac{1}{2}$, $G(3) =, \ldots, = G(K) = 0$.

Bemutatjuk a tárgyak egy általános osztályát, ami a következő részsorozatokból áll. Először, az (a) fázisban sok kicsit tárgy érkezik, melyek összmérete 2. A (b) fázisban érkezik két nagy tárgy, mindkettő 1 méretű. E két fázisban jövő tárgyak együttesét tekintjük egy *csomag*nak (vagy „batch"-nak) a sorozatban. A sorozat n darab ugyanolyan csomagból áll. Feltételezzük, hogy létezik egy felosztása a sok kicsi tárgynak két egyenlő összméretű részre (vagyis $1 + 1$ összméretű részekre), de a kicsi tárgyak sorozatának semelyik kezdő részének az összmérete nem egyenlő 1-el. Az (a) fázisra ez azt jelenti, hogy a klasszikus Partició problémára, a „pozitív esetek nehezen felismerhető osztályával" állunk szemben (hard-to recognize class of positive instances), vagyis azzal az esettel amikor az optimális megoldás értéke 1, de ezt „nehéz előállítani".

Állítjuk, hogy $OPT \geq n(3G(1) + G(2)) = 3.5n$, vagyis az optimális megoldás értéke legalább $3.5n$. Tekintsük ennek érdekében a következő offline megoldást. Kezdetben két láda nyitott és a kicsi tárgyakat ebbe a két ládába pakoljuk úgy, hogy az (a) fázis végére mindkét láda telítettsége 1 lesz, azaz mindkét láda fedett lesz. Ekkor kapunk $G(2) = 0.5$ hasznot az első fedett láda után és $G(1) = 1$ hasznot a második láda fedése után. Ebben a pillanatban nincsen nyitott láda. Ekkor a két nagy tárgy érkezik egyesével, beletesszük az első tárgyat egy nyitott ládába és szintén beletesszük egy nyitott ládába a második tárgyat is. Ezen két láda fedéséért $2G(1) = 2$ hasznot kapunk. A sorozat n csomagot tartalmaz és minden csomagnál $3G(1) + G(2)$ lesz a kapott haszon. (Ez egyébként az optimális offline megoldás az adott inputra, de erre a tényre nincs szükségünk a bizonyítás során.)

Másrészt tekintsük egy tetszőleges polinomiális idejű A algoritmust. Először tegyük fel, hogy az A algoritmus sohasem nyit meg egyszerre kettőnél több ládát.

Mivel a Partíció probléma megoldása NP nehéz, A nem képes két ládát fedni a kicsi tárgyakkal, emiatt erre nem képes egyetlen csomag esetén sem. Így ha nincsen nyitott láda az (a) fázis elején, akkor legalább egy láda nyitvamarad annak a fázisnak a

induláskor	befejezéskor	maximális haszon
0	1	G(1)
0	2	G(1)
1	0	2G(1)
1	1	2G(1)
1	2	2G(1)
2	0	2G(1) + G(2)
2	1	2G(1) + G(2)
2	2	2G(1) + G(2)

1. táblázat. A lehetséges átmenetek az (a) fázis után.

induláskor	befejezéskor	maximális haszon
0	0	2G(1)
1	0	2G(1)
1	1	2G(2)
2	0	G(1) + G(2)
2	1	2G(2)

2. táblázat. A lehetséges átmenetek a (b) fázis után.

végén. Ezért a pakolás nyitott, de nem üres ládászám szerinti jellemzéséhez megadjuk a következő lehetséges átmeneteket (lásd 1. táblázat).

A táblázatbeli számokat könnyen lehet ellenőrizni, például ha az (a) fázis két nyitott ládával kezdődik[2], akkor az első fedés csak $G(2)$ hasznot hoz. Egy hasonló táblázat készíthető a (b) fázishoz is (lásd 2. táblázat).

Ennek igazolásához azt kell észrevenni, hogy az 1 méretű tárgy mindig fed egy ládát, ezért legfeljebb egy láda maradhat nyitva a fázis végén; és ha valóban egy láda maradt nyitva, akkor mindkét nagy tárgy a második ládát fedte.

A fenti két táblázat felhasználásával megadható a lehetséges összes átmenet a csomag feldolgozása során. Itt jegyezzük meg, hogy az eljárás tárgyak nélkül kezdődik (vagyis egyetlen láda sem lehet nullától eltérő telítettségű), így azzal a feltételezéssel, hogy harmadik láda sohasem volt megnyitva, legfeljebb egy láda maradhat nyitva minden csomag

[2]Ez a helyzet csak akkor áll elő, ha elhagyjuk azt a feltételezést, hogy az A algoritmus csak legfejlebb két ládát nyithat meg egyszerre (emiatt persze $K \geq 3$ szükségszerűen teljesül), de ezt az esetet azért tettük bele ebbe a táblázatba, hogy a bizonyítás későbbi részében (az általános esetben) hivatkozni tudjunk rá.

induláskor	közben	befejezéskor	maximális haszon
0	1	0	$3G(1) = 3$
0	2	0	$2G(1) + G(2) = 2.5$
0	1	1	$G(1) + 2G(2) = 2$
0	2	1	$G(1) + 2G(2) = 2$
1	0	0	$4G(1) = 4$
1	1	0	$4G(1) = 4$
1	2	0	$3G(1) + G(2) = 3.5$
1	1	1	$2G(1) + 2G(2) = 3$
1	2	1	$2G(1) + 2G(2) = 3$

3. táblázat. Egy csomag közbülső állapottal az (a) és a (b) fázis között.

végén. Így kapjuk a 3. táblázatot.

A legfontosabb észrevételünk most arra irányul, hogy legfeljebb mekkora hasznot kaphatunk egy csomag kezelése során.

Ez négy lehetséges kombinációt jelent minden egyes csomag esetén (0-val, vagy 1-el kezdve és 0-val, vagy 1-el befejezve), kivéve azt, hogy az első csomagra csak két esetünk van (0-val kezdve). Néhány kombináció (nevezetesen $0-2-0$ és $1-2-0$) irreleváns, mert ezeknél vannak jobb alternatívák (például $0-2-0$ helyett jobb lenne a $0-1-0$ kombináció.)

Így a $\{0,1\} \times \{0,1\}$-hez tartozó négy lehetséges átmenet haszna 3, 2, 4, 3.

Ezek után, annak érdekében hogy pontosan lássuk hogy mi történik a csomagok feldolgozása során, a folyamatot egy él-súlyozott, irányított gráf segítségével vizsgáljuk a továbbiakban. A $G = (V, E)$ gráf $2n+1$ csúcsot és $4n-2$ élet tartalmaz, ahol: $V = \bigcup_{i=0}^{n} V_i$, $V_0 = \{v_{0,0}\}$, $V_i = \{v_{i,0}, v_{i,1}\}$, ahol $i = 1, \ldots, n$. A V_{i-1} és V_i csúcshalmazokat teljesen összekötjük, a kisebb indextől a nagyobb felé irányuló élekkel, és az élek súlya az alábbiak szerint legyen adott:

$$w(v_{i-1,0}v_{i,0}) = 3 \ (1 \leq i \leq n),$$
$$w(v_{i-1,0}v_{i,1}) = 2 \ (1 \leq i \leq n),$$
$$w(v_{i-1,1}v_{i,0}) = 4 \ (2 \leq i \leq n),$$
$$w(v_{i-1,1}v_{i,1}) = 3 \ (2 \leq i \leq n).$$

Ekkor $C_A(L,G)$ nem lehet nagyobb, mint a V_0-ból V_n-be vezető irányított utak közül a legnagyobb súlyúnak a súlya. Ezt a maximális súlyt jelölje l. Azt állítjuk, hogy $l = 3n$. Ebből már következik ugyanis, hogy

$$\frac{C_A(L,G)}{OPT} \leq \frac{l}{OPT} \leq \frac{3n}{3.5n} = \frac{6}{7}.$$

Most tehát, az $l = 3n$ egyenlőség igazolása érdekében, legyen P a $V_0 \to V_n$ utak közül azon maximális súlyú útnak a súlya, amelyre a $v_{i,0}$ típusú csúcsok száma a lehető legnagyobb. Azt állítjuk, hogy P csúcshalmaza pontosan $\{v_{i,0} | 0 \leq i \leq n\}$, ebből ugyanis az állításunk már azonnal következik majd.

Figyeljük meg, hogy a $v_{n,0}$ biztosan P-hez tartozik. Ez az állítás azért teljesül, mert $w(v_{n-1,0}v_{n,0}) > w(v_{n-1,0}v_{n,1})$ és $w(v_{n-1,1}v_{n,0}) > w(v_{n-1,1}v_{n,1})$. Most ellentmondásként tegyük fel, hogy $v_{i,0} \notin P$ valamely $0 < i < n$ esetén, és legyen i a legnagyobb ilyen tulajdonságú index. Ekkor természetesen $v_{i+1,0} \in P$.

Legyen $j \in \{0,1\}$ az az érték, amire $v_{i-1,j} \in P$. Most azt vehetjük észre, hogy

$$w(v_{i-1,j}v_{i,0}) + w(v_{i,0}v_{i+1,0}) = w(v_{i-1,j}v_{i,1}) + w(v_{i,1}v_{i+1,0}),$$

függetlenül attól, hogy mi a j aktuális értéke. Ezért ha $v_{i,1}$-et kicseréljük $v_{i,0}$-val P-ben, akkor nem csökken az összsúly, viszont ezáltal növekszik azon csúcsok száma P-ben, amelyeknek 0 a második indexe. Ez ellentmond a P választásának, ezáltal az $l = 3n$ egyenlőséget igazoltuk.

Ezek után, hogy teljessé váljon a tétel bizonyítása, már csak azt kell igazolnunk, hogy egyszerre három, vagy több nyitott láda lehetőségének megengedése (vagyis $K \geq 3$) nem javítja az approximációs arányt. Ennek bizonyítása az előző bizonyítás kibővített változata. Az elérhető lehetséges haszon maximuma az (a) fázisban több, mint két nyitott ládával kezdve a csomag pakolását, megint $2G(2) + G(1)$, ami ugyanaz, mintha csak két nyitott ládával kezdtünk volna, függetlenül attól, hogy ennek a fázisnak a végén hány láda maradt nyitva. Továbbá, ha kezdéskor a nyitott ládák száma 0 vagy 1, akkor a keletkező haszon számításánál irreleváns, hogy az (a) fázis végén kettő, vagy legalább három láda

induláskor	közben	befejezéskor	maximális haszon
≥ 2	0	0	$4G(1) + G(2) = 4.5$
≥ 2	1	0	$4G(1) + G(2) = 4.5$
≥ 2	1	1	$2G(1) + 3G(2) = 3.5$
≥ 2	2	0	$3G(1) + 2G(2) = 4$
≥ 2	2	1	$2G(1) + 3G(2) = 3.5$
≥ 2	2	2	$2G(1) + G(2) = 2.5$
≥ 2	3	1	$2G(1) + 2G(2) = 3$
≥ 2	3	≥ 2	$2G(1) + G(2) = 2.5$
≥ 2	≥ 4	≥ 2	$2G(1) + G(2) = 2.5$

4. táblázat. A haszon felső határa egy csomagban legalább két nyitott ládával kezdve.

marad nyitva, vagyis az előző táblázatban szereplő adatok érvényesek maradnak a jelenlegi esetekre is.

A (b) fázisban a különbség az, hogy több, mint egy láda maradhat nyitva a (b) fázis befejezése után. Ez lehetővé teszi, hogy a következő csomag kezdésekor több mint egy nyitott ládával kezdjünk, ami magasabb hozamot hozhat ebben a következő csomagban. Azonban mindjárt kiderül, hogy ha így teszünk, akkor annak nagy árát fizetjük az előző elszállított ládánál, emiatt összességében nem fogunk extra hasznot kapni. Bármely esetben, minden nagy tárgy a (b) fázisban fedni fogja azt a ládát, ahova pakoltuk, így a megnyitott ládák száma nem tud növekedni. Az elvileg elérhető maximális haszon azokban az esetekben, amelyeket még nem vizsgáltunk, a következő táblázatba lett összegyűjtve (lásd 4. táblázat).

Előzőleg, egy nyitott ládával a kezdéskor és 0, 1, vagy 2-vel a befejezéskor azt kaptuk a haszon maximumaira hogy az 4, 3, vagy 2, egy csomag feldolgozásakor. Amint az a 4. táblázatból látható, két kezdő ládával az előbbi három szám, vagyis az elért haszon felnövekedhet 4.5, 3.5, vagy 2.5-re. Az extra haszon 0.5, de ahhoz hogy ezt lehetővé tegyük, kell lennie legalább 1 veszteségnek az előző csomagban, vagyis ez az ára annak, hogy több, mint egy ládával zárunk.

Ezek után a korábbi gráfot kell kiegészítenünk, ahol a csúcsok és élek azt reprezentálják, hogy hány nyitott láda van a batch feldolgozása előtt, illetve után, illetve az átmeneteket: egy-egy él az átmenet során a haszon növekedésének felel meg. A gráf

redukálható a következő észrevételek felhasználásával:

i, a kezdő ládák számát 2-re csökkentve (ha az ennél nagyobb volt), a profitot nem változtatja

ii, hasonlóan, a nyitva lévő ládák záró számát (nyitva maradt ládák száma a csomag végén) 2-re csökkentve, a haszon nem fog csökkenni,

iii, az átmenet során, ha az (a) és (b) fázis közötti ládaszámot 2-re csökkentjük, ha az ennél nagyobb volt, a haszon nem csökken.

Ezáltal az optimális út az általános esetben ($K \geq 3$) nem használ olyan éleket, amelyeket a korábbi esetben ($K \leq 2$) ne használt volna, vagyis az optimális útnak nem kell használnia a páros gráf más részeit, mint amelyek 0, 1 és 2 szintekhez tartoznak. Végül, bármely irányított út, ami a második szintről tartalmaz csúcsokat, módosítható olyan úttá, ami eggyel kevesebb csúcsot tartalmaz a második szinten, anélkül, hogy a kapott haszon csökkenne. Ezt ismételve kapjuk, hogy a haszon nem növekedhet több, mint két nyitott láda esetén, ezáltal a tétel bizonyítása teljessé vált. ∎

3. állítás. Legyen $K \geq 2$ *rögzített egész esetén. Ekkor kellően megválasztott* G *haszonfüggvény esetén létezik olyan* L *lista (inputok olyan osztálya), amelyre nem létezik olyan polinomiális idejű algoritmus, ami jobb abszolút approximációs aránnyal rendelkezik, mint* $\frac{1}{2}$*, feltéve hogy* $P \neq NP$*.*

Bizonyítás. A bizonyítás alapvető gondolata megegyezik az előző tétel bizonyításában foglaltakkal, csak most minden sokkal egyszerűbb. Legyen $G(1) = 1$, $G(2) = 1$, továbbá $G(3) = 0,\ldots,G(K) = 0$. Az általános példa a vizsgálandó esetben: tárgyak egy L listája, amely megint egy pozitív példája a Partició problémának (vagyis L szétoszthető két egyforma részre). Ekkor egy optimális algoritmus egészen megtölt két ládát és megkapja a 2 hasznot érte. Másrészt mivel a Partició probléma NP-teljes, nincs olyan polinomiális idejű A algoritmus, ami a tárgyak jobb felosztását bármely input esetén megtalálja. Ha A nyit két ládát, akkor nem tudja telíteni mindkettőt, és csak 1 hasznot kap összesen. Ha csak egy ládát nyit, akkor pedig nem tudja a ládát pontosan 1-ig telíteni, ezért miután szállította az első zárt ládát, A nem tud újabb ládát fedni. Következésképpen bármely A

csak 1 hasznot kap összesen a legrosszabb esetekben és az abszolút approximációs arány csak $\frac{1}{2}$ lesz. ∎

6.5. Online eset

Az online algoritmusok hatékonyságát rendszerint versenyképességi analízissel mérik. Ez azt jelenti, hogy egy A online algoritmus által kapott $C_A(I)$ célfüggvény-értéket (ahol I-vel jelölik az inputot) összehasonlítják az offline optimum $OPT(I)$ értékével.

Az offline modellben előre ismerjük a bemenetre vonatkozó összes információt, de a tárgyakat az adott L lista szerinti sorrendben kell a ládákba pakolnunk. Természetesen offline optimális megoldásnak léteznie kell. Bármelyik pillanatban, amikor egy új tárgy érkezik, véges sok lehetőség közül lehet választani. A mindent összevetve is véges számú lehetőség között léteznie kell olyan megoldásnak, ami a legjobb értékét adja a célfüggvénynek. Természetesen az offline-optimum nemcsak a tárgykészlettől függ, de az adott L listától is és a $G(k)$ haszon-függvénytől is.

Bármely véges L tárgy-lista és G haszon-függvény esetén legyen $C_A(L,G)$ egy A algoritmus által kapott megoldási értéke. Ezt hasonlítjuk tehát össze az offline-optimális megoldással, amit $OPT(L,G)$-vel jelölünk. Természetesen: $C_A(L,G) \leq OPT(L,G)$. Ekkor azt mondjuk, hogy:

Az A algoritmus ρ-versenyképes $(0 \leq \rho \leq 1)$: Ha $\frac{C_A(L,G)}{OPT(L,G)} \geq \rho$ teljesül bármely L, G esetén.

Az A algoritmus versenyképességi aránya: Az előbbi ρ számok szuprémuma.

Nyilvánvaló, hogy ez a szuprémum létezik, tehát a versenyképességi arány jól definiált. Másrészt, tegyük fel, hogy az L lista és G haszon-függvény esetén tetszőleges A online algoritmusra $\frac{C_A(L,G)}{OPT(L,G)} \leq \eta$ teljesül valamilyen η számmal.

A feladat felső korlátja: A η számok bármelyike.

A következőekben bemutatjuk néhány klasszikus algoritmus természetes adaptációját.

6.6. Néhány (természetesen adódó) algoritmus

Először tekintsük a Duál Next-Fit (röviden DNF) algoritmust. A DNF mindig csak egy ládát tart nyitva, és amint a láda megtelik, elszállítjuk, és új tárgy érkezése esetén új ládát nyitunk. Az algoritmus formális leírása az alábbi:

DNF Algoritmus:

1. Kezdetben nincs nyitott láda.
2. A listán következő tárgyat mindig a nyitott ládába pakoljuk, ha van nyitott láda; ellenkező esetben nyitunk számára egy új ládát.
3. Amint fedetté válik egy láda, bezárjuk és elszállítjuk.
4. Ha már nem jön több tárgy, az algoritmus megáll, ellenkező esetben a 2. lépésre megyünk.

Ezen algoritmus alkalmazása esetén csak egy lehetőség van a soron következő tárgy pakolásához. Azonban ez az algoritmus mégis optimális tud lenni abban a speciális esetben, ha az elszállított ládák után kapott haszon nulla, ha több mint egy láda van egyszerre nyitva.

1. lemma. *Legyen $K \geq 2$ és tegyük fel, hogy $G(k) = 0$ bármely $2 \leq k \leq K$ esetén. Ebben az esetben a DNF algoritmus optimális.*

Továbbá a DNF optimális abban az esetben is, ha a tárgyméretek majdnem egyformák, a következő Lemma szerint:

2. lemma. *Tegyük fel, hogy $\frac{1}{m} < p_i < \frac{1}{m-1}$ teljesül minden tárgyméretre, ahol $m \geq 2$ rögzített egész szám. Ekkor a DNF algoritmus optimális.*

Bizonyítás. Minden fedett láda pontosan $m-1$ tárgyat tartalmaz. Emiatt a legjobb választás, hogy egyszerre csak egy láda van nyitva. ∎

3. lemma. *A DNF algoritmus versenyképességi aránya $\rho(DNF) \geq \frac{1}{2}$, tetszőleges G haszonfüggvény esetén.*

fedett ládák átlagos telítettsége	fedett ládák száma (haszon)
1.126	210
1.116	210
1.133	208
1.128	207
1.131	208

5. táblázat. A DNF algoritmus alkalmazásával elért haszon, $I = [0.1; 0.4]$ esetén.

	az elért haszon	átlagos ládatelítettség
I_1	172	1.093
I_2	346	1.258
I_3	424	1.230

6. táblázat. A DNF algoritmus alkalmazásával elért haszon a három különböző input esetén.

Bizonyítás. Az állítás abból a tényből következik, hogy $G(k)$ nem növekszik a k változó függvényeként, továbbá az egyes ládákba pakolható tárgyak összmérete legfeljebb 2, mert a pakolandó tárgyak mérete legfeljebb 1. ∎

Az alábbiakban teszteljük a DNF algoritmus hatékonyságát: a tárgyméreteket vegyük az $I = [0.1; 0.4]$ intervallumból (egyenletes eloszlással), a tárgyak száma legyen 1000 és a fedett és szállított ládák után járó haszon pedig 1 egység (vagyis itt $K = 1$ és G(1) = 1). Öt futtatás eredménye megtalálható a 5. táblázatban.

Az alábbiakban ismét teszteljük a DNF algoritmus hatékonyságát három különböző esetben, ahol az intervallumok a következők: $I_1 = [0.1; 0.3]$, $I_2 = [0.3; 0.6]$, $I_3 = [0.3; 0.8]$, a tárgyak száma továbbra is 1000 és a fedett és szállított ládák után járó haszon $G(k) = 1$. Három futtatás eredménye megtalálható a 6. táblázatban.

A 6. táblázat adataiból láthatjuk, hogy a három intervallum közül az I_1 intervallum esetén tudta a DNF algoritmus a legjobban felülről megközelíteni az 1 egységnyi ládatelítettséget. Az elért haszon mégis az I_3 intervallum esetén a legnagyobb, hiszen az I_3 intervallum eleve nagyobb méretű tárgyakat tartalmaz, mint az I_1 (ezért ebből adódik, hogy több ládát lehetett telíteni) és mindkét esetben a pakolandó tárgyak darabszáma változatlanul 1000.

Természetesen a tárgyméretekre általában az előző feltétel nem teljesül. Ezért a DNF alkalmazásakor veszteség keletkezik, mert számos láda „túl lesz pakolva". Ezen azt értjük, hogy a ládát úgy fedjük, hogy a ládába pakolt tárgyak összmérete jóval nagyobb, mint 1, tehát jóval több, mint ami éppen elegendő lenne. *Ha ezt a veszteséget el akarjuk kerülni, lehetővé kell tennünk, hogy egyszerre több láda legyen nyitható, ezáltal a ládaméret jobban közelíthetővé válik.*

Tekintsük a következő klasszikus algoritmus, a $Harmonic(\kappa)$ algoritmus, vagy röviden $H(\kappa)$ adaptációját, ahol: $\kappa \geq 1$ egész. Az algoritmus elve az, hogy csak hasonló méretű tárgyak kerülhetnek egy ládába. Egyszerre legfeljebb κ darab láda lehet nyitva. A tárgyakat méreteik alapján osztályokba csoportosítjuk, minden láda egy osztályt reprezentál. Ha a következő tárgy mérete az $S_k = (\frac{1}{k+1}, \frac{1}{k}]$ intervallumba esik, akkor a k-adik ládatípusba kerül, ahol: $k = 1, \ldots, \kappa - 1$. A legkisebb tárgyak, vagyis az $S_\kappa = (0, \frac{1}{\kappa}]$ intervallumba esők, pedig a κ-adik típusú ládába kerülnek.

$H(\kappa)$ Algoritmus:

- Ha a következő tárgy mérete az S_k intervallumba esik, helyezzük a k-adik típusú ládába, ha van ilyen nyitott láda.
- Ha nincs ilyen nyitott láda, akkor nyitunk számára egy ilyen típusút.
- Amint egy láda megtelik (fedetté válik), elszállítjuk.
- Ha nincs további tárgy, az algoritmus megáll.

Az alábbiakban teszteljük a $H(\kappa)$ algoritmus hatékonyságát, az intervallumok a következők: $I_1 = [0.1; 0.3]$, $I_2 = [0.3; 0.6]$, $I_3 = [0.3; 0.8]$, a tárgyak száma továbbra is 1000 és a fedett és szállított ládák után járó haszon $G(k) = 10.1 - k * 0.1$, vagyis G enyhén csökken. Három futtatás eredménye megtalálható a 7. táblázatban.

A 7. táblázat alapján megfigyelhető, hogy a $H(\kappa)$ algoritmus jobban teljesít, mint a DNF, ha κ „nem túl nagy" és a profit-függvény „nem csökken túl gyorsan". (I_1 és I_2 esetén $H(2)$ jobb, I_3 esetén is majdnem olyan jó).

	DNF	$H(\kappa = 2)$	$H(\kappa = 3)$
I_1	1720 (1.093)	1730 (1.101)	1720 (1.097)
I_2	3460 (1.258)	3789 (1.141)	3562.3 (1.190)
I_3	4240 (1.230)	4153.5 (1.248)	4046.3 (1.285)

7. táblázat. A DNF és a $H(\kappa)$ algoritmus alkalmazásával elért haszon összehasonlítása három (sorban: I_1, I_2, I_3) intervallum esetén.

	$SH(\kappa = 2)$	$SH(\kappa = 3)$
I_1	1740 (1.088)	1720 (1.088)
I_2	3795.5 (1.149)	3678.6 (1.162)
I_3	4353.3 (1.198)	4382 (1.190)

8. táblázat. Az $SH(\kappa)$ algoritmus alkalmazásával elért haszon három (sorban: I_1, I_2, I_3) intervallum esetén.

Megjegyezzük, hogy a $H(\kappa)$ okosabbá tehető oly módon (vagyis nyerünk egy új algoritmust, amit Smart Harmonic(κ)-nak, vagy röviden $SH(\kappa)$-nak neveztünk el, ami a $H(\kappa)$ „ügyes változata"), ha a következő szabályok szerint pakolja a tárgyakat:

$SH(\kappa)$ Algoritmus:

- Ha a következő elem le tudja fedni valamelyik ládát, akkor tegyük a tárgyat ezek közül a legkisebb telítettségű ládába, és szállítsuk el a ládát.
- Bármely más esetben az $SH(\kappa)$ algoritmus a $H(\kappa)$ algoritmus szabálya szerint működik.

Az alábbiakban teszteljük az $SH(\kappa)$ algoritmus hatékonyságát, az intervallumok változatlanul a következők: $I_1 = [0.1; 0.3]$, $I_2 = [0.3; 0.6]$, $I_3 = [0.3; 0.8]$, a tárgyak száma továbbra is 1000 és a fedett és szállított ládák után járó haszon $G(k) = 10.1 - k * 0.1$. Három futtatás eredménye megtalálható a 8. táblázatban.

A 8. táblázat alapján megfigyelhető, hogy az $SH(\kappa)$ algoritmus mindhárom intervallumban (megfelelően választott κ paraméter esetén) jobb eredményeket ért el, mint a DNF vagy a $H(\kappa)$ algoritmus.

A 9. táblázatban az előző algoritmusokat hasonlítottuk össze. Soronként a különböző algoritmusok által kapott megoldásokat közöljük, soronként más-más feladatosztályokra

	DNF	$H(2)$	$H(3)$	$SH(2)$	$SH(3)$
1	2080=100%	98.56%	97.4%	99.1%	97.51%
2	1770=100%	101.13%	100%	101.13%	101.13%
3	3460=100%	93.64%	83.12%	99.23%	100.55%
4	2080=100%	100%	101.2%	98.67%	101.75%
5	1770=100%	100.57%	100%	100%	101.13%
6	3460=100%	101.03%	95.65%	103.21%	108.37%

9. táblázat. Algoritmusok összehasonlítása ($\kappa = 2$ és $\kappa = 3$ esetén) az elért haszon alapján hat esetet figyelembe véve.

alkalmazva őket. A problémaosztályok az alábbiak:

A ládaméretet minden esetben 1-nek, a tárgyak számát pedig 1000-nek választottuk. A tárgyméretek soronként a következő intervallumokból kerülnek ki: $[0.1; 0.4]$, $[0.15; 0.25]$, $[0.1; 1]$, és ismét $[0.1; 0.4]$, $[0.15; 0.25]$, $[0.1; 1]$. A profit függvény az első három esetben: $G(k) = 11 - k$ (pl.: $G(1) = 10$, $G(2) = 9$, és így tovább), ami egy gyorsan csökkenő profit-függvény. A következő három esetben pedig: $G(k) = 10.1 - 0.1 * k$ (pl.: $G(1) = 10$, $G(2) = 9.9$, és így tovább), vagyis egy lassan csökkenő profit-függvényt választottunk.

Így kapunk hat lényegesen különböző feladatosztályt. A DNF algoritmus által kapott megoldás értékét mindig 100%-nak tekintjük és ehhez hasonlítjuk a többi algoritmus által kapott megoldásokat. Tíz futás átlagát írtuk a táblázat megfelelő rublikáiba a 9. táblázatban. (Az algoritmusokat megvalósító program C-nyelven íródott, a fordítás GCC-vel történt. A szükséges nyílt forráskódú GCC fordítóprogram szabadon letölthető a MinGW honlapjáról.)

Emlékezzünk vissza, hogy az első három esetben a haszon-függvény erősen csökkenő. Az első feladatosztály esetén a többi algoritmus nem képes a DNF algoritmust „legyőzni”, de a második és a harmadik esetben már van olyan algoritmus, amelyik legyőzi a DNF-et. A többi esetben a profit-függvény csak enyhén csökken, itt a DNF könnyen legyőzhető, de egy idő után a $H(\kappa)$ ill. $SH(\kappa)$ hatékonysága nem feltétlenül javul tovább κ növelésével, ezt szemlélteti a 10. táblázat.

Az $SH(\kappa)$ bizonyos esetekben tényleg „okosabb”, vagy „ügyesebb”, mint a $H(\kappa)$ algoritmus egyszerű változata (ugyanazzal a κ paraméterrel).

	DNF	$H(4)$	$SH(4)$
1	2080=100%	84.52%	89.57%
2	1770=100%	99.44%	99.44%
3	3460=100%	75.87%	96.68%
4	2080=100%	100.91%	104.32%
5	1770=100%	100%	100.57%
6	3460=100%	92.98%	108.11%

10. táblázat. Algoritmusok összehasonlítása ($\kappa = 4$ esetén) az elért haszon alapján hat esetet figyelembe véve.

Figyeljük meg, hogy a hatodik feladatosztály esetében az $SH(\kappa)$ lényegesen jobb, mint a $H(\kappa)$, ennek az lehet az oka, hogy a tárgyméretek eloszlása nagyobb lehetőséget ad az algoritmusnak arra, hogy „okos" legyen.

6.7. Egy új, rugalmas algoritmus-család

Most egy új algoritmus-családot definiálunk, amely kellően rugalmas ahhoz, hogy az előbbi algoritmusok bármelyikével sikeresen felvegye a versenyt. Ezt különböző stratégiai paraméterek beállításával érjük el, ahol a K paraméter jelenti az egyszerre megnyitható ládák maximális számát, a többi paraméter pedig az algoritmus pakolási politikáját szabályozza.

Az algoritmus egy elfogadás-elutasítás politikát folytat: a következő tárgyat elfogadja, és valamely ládába pakolja, ha a ládába pakolt tárgyak (az aktuális tárgy méretével) megnövelt összmérete az „elfogadó" tartományba kerül. Elutasítja a tárgynak a ládába történő pakolását, ha a megnövelt összméret az „elutasító" tartományba kerül.

Az elfogadás-elutasítás politika lehetővé teszi, hogy minél kevesebb teli láda legyen túlpakolva (vagyis minél több teli láda minél jobban közelítse felülről az egységnyi ládaméretet).

$Mask$ Algoritmus:

1. Ha a következő tárgy lefedi valamelyik ládát az elfogadó-tartományban, akkor pakoljuk a tárgyat abba a ládába, amelyik ezen ládák közül a legkisebb telítettségű.

Szállítsuk el a ládát és menjünk az 5-ös pontra.

2. Ha a következő tárgy pakolható valamelyik ládába (az elfogadó-tartományban, de a láda még nem lesz fedett), akkor pakoljuk azt egy ilyen ládába. Menjünk az 5-ös pontra.

3. Ha $k < K$, akkor nyissunk egy új ládát, az aktuális tárgy ebbe a ládába kerül, és menjünk az 5-ös pontra.

4. Ha $k = K$, akkor pakoljuk az aktuális tárgyat a legkisebb telítettségű ládába. Ha a láda fedett lesz, szállítsuk el. Menjünk az 5-ös pontra.

5. Ha nincs több tárgy, az algoritmus megáll, különben menjünk az 1-es pontra.

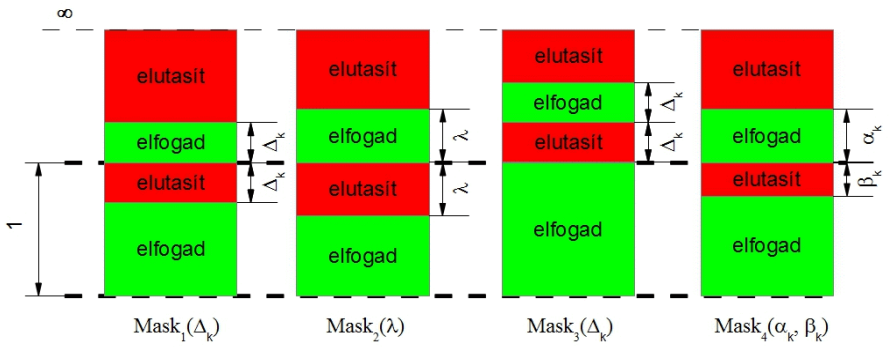
Megjegyzés: a $Mask$ algoritmusnak itt már rögtön az ügyes változatát definiáltuk, vagyis ha a következő tárggyal be tudunk fedni egy ládát, az elfogadási tartományon belül, ezek közül olyanba tesszük, amelyik a legkevésbé lesz túlpakolva.

Az elutasító, illetve az elfogadó tartományokat a λ egész, vagy a Δ, ill. α és β K dimenziós nemnegatív vektorok (paraméterek) definiálják, ahol minden komponens kisebb, mint 1.

A $Mask_1(\Delta_k, K)$ algoritmus esetén a k-adik láda elfogadó tartománya: $[0; 1-\Delta_k] \cup [1; 1+\Delta_k]$ (mindegyik ládához két elfogadó tartományt rendelünk). Az elutasító tartomány: $(1-\Delta_k; 1) \cup (1+\Delta_k; \infty)$. A $Mask_1$ algoritmus esetén a Δ_k paramétert a 0 és 0.5 közötti tartományból választjuk, különböző $1 \leq k \leq K$ esetén Δ_k különböző is lehet.

A $Mask_2(\lambda, K)$ algoritmus esetén a k-adik láda elfogadó tartománya: $[0; 1-\lambda] \cup [1; 1+\lambda]$. Az elutasító tartomány: $(1-\lambda; 1) \cup (1+\lambda; \infty)$. A $Mask_2$ algoritmus esetén a λ paramétert 0 és 0.5 közötti tartományból választjuk. A $Mask_2$ algoritmus egy speciális esete ($\Delta_k = \lambda$) a $Mask_1$ algoritmusnak, tehát itt Δ_k ugyanaz minden K-ra.

A $Mask_3(\Delta_k, K)$ algoritmus esetén a k-adik láda elfogadó tartománya: $[0; 1] \cup [1 + \Delta_k; 1+2*\Delta_k]$. Az elutasító tartomány: $(1; 1+\Delta_k) \cup (1+2*\Delta_k; \infty)$. A $Mask_3$ algoritmusnak különböző elfogadó és elutasító tartományai vannak, mint a $Mask_1$ algoritmusnak,



3. ábra. A maszk algoritmus típusai (ebben az esetben a paraméterek K dimenziós nemnegatív vektorok).

de hasonlóan Δ_k paramétert használ a tartományok beállítására. (Itt jegyezzük meg, hogy a $Mask_3$ algoritmus $(1; 1 + \Delta_k)$ elutasító tartománya habár azt segíti, hogy a ládát ne lehessen kissé túlpakolni, de az $[1+\Delta_k; 1+2*\Delta_k]$ elfogadó tartománya már megengedi a nagyobb túlpakolást is, ami azonban csak speciális esetekben lehet előnyös.)

A $Mask_4(\alpha, \beta, K)$ algoritmus esetén a k-adik láda elfogadó tartománya: $[0; 1 - \beta_k] \cup [1; 1 + \alpha_k]$. Az elutasító tartomány: $(1 - \beta_k; 1) \cup (1 + \alpha_k; \infty)$. A $Mask_3$ algoritmus tud $Mask_1$ algoritmusként működni (például: $Mask_1(0.1, 4)$ működésben megegyezik a $Mask_4(0.1, 0.1, 4)$ algoritmussal), azonban a $Mask_4$ algoritmus sokkal rugalmasabb, mint az előző változatok, mert két különböző (α az egységnyi ládaméret feletti, β pedig az egységnyi ládaméret alatti tartományok beállításához) paraméterrel állítható be az elfogadó és az elutasító tartomány.

A különböző $Mask$ algoritmusok tartományait a 4. ábra szemlélteti.

A 11. táblázatban az előző $Mask$ algoritmusokat hasonlítottuk össze (a $Mask_1$-et és a $Mask_2$-őt most azonosnak tekintjük). A ládaméretet minden esetben 1-nek, a tárgyak számát pedig 1000-nek választottuk. A tárgyméretek a következő intervallumból kerülnek ki: $I = [0.3; 0.8]$ továbbá $K = 3$, $\Delta_k = \alpha_k = [0.2, 0.2, 0.2]$ és $\beta_k = [0.25, 0.25, 0.25]$.

A 11. táblázatból jól látható, hogy a $Mask_4$ algoritmus jobb, mint a másik kettő,

	$Mask_1$	$Mask_3$	$Mask_4$
I	4599.7 (1.126)	3969.6 (1.325)	4648.5 (1.121)

11. táblázat. $Mask$ algoritmusok összehasonlítása az elért haszon (és átlagos ládatelítettség) alapján az I intervallumon.

ezért a továbbiakban a $Mask_4$ algoritmust tárgyaljuk és $Mask$ algoritmusként jelöljük.

Egy példát adunk ennek a $Mask$ algoritmusnak a működésére. Legyenek adottak: $K = 4$, ládaméret: 100, $\underline{\alpha} = [10; 20; 30; 40]$, $\underline{\beta} = [30; 20; 30; 40]$, továbbá a tárgyméretek a $[10; 40]$ intervallumból az alábbiak: $24; 35; 18; 22; 16; 29; 20; 17; 38; 14; 31; 28; 32$.

Az első lépésben csak egy nyitott láda van és ez a láda még üres. Az α_1 paraméter értéke miatt a tárgyakat szabad pakolni az első nyitott ládába, ha az így megnövekedett ládaméret kisebb, vagy egyenlő, mint: $100 - 10 = 90$. A β_1 paraméter értéke miatt pedig a tárgyakat szabad pakolni az első nyitott ládába, ha az így megnövekedett ládaméret nagyobb, mint 100 és kisebb, vagy egyenlő, mint: $100 + 30 = 130$. Tehát az elutasító tartományok: 90 és 100 között, továbbá $100 + 30 = 130$ és ∞ között helyezkednek el. Az elfogadó tartományok: 0 és 90 között, továbbá 100 és 130 között helyezkednek el az első ládára vonatkozólag.

Tegyük az első tárgyat: 24 ebbe a nyitott ládába. A következő tárgy: 35 még szintén az első ládába kerülhet, hiszen az így megnövekedett összméret: $24 + 35 = 59$. Tegyük a következő tárgyat: 18 a ládába, így a ládatartalom: $18 + 59 = 77$ lesz.

A következő tárgy: 22 nem kerülhet ebbe a ládába, mert a megnövekedett összméret már az elutasító tartományba esne, ezért egy új, üres, második ládát kell nyitnunk. A második láda elfogadó tartománya: 0 és $100 - 20 = 80$ között, továbbá 100 és $100 + 20 = 120$ között helyezkedik el az α_2 és β_2 paraméterek alapján.

Tegyük a következő tárgyat: 22 a második nyitott ládába. Most van egy 77 és egy 22 tartalmú nyitott ládánk. Tegyük a következő tárgyat: 16 a második ládába, így ennek a ládának a tartalma $22 + 16 = 38$ lesz. A következő tárgyat: 29 tegyük az első ládába, mert ebben az esetben az első láda összmérete: $77 + 29 = 106$ lesz és így már szállíthatjuk is ezt a ládát. Amikor fedtük az első ládát, egyszerre két nyitott ládánk volt (az első és

a második), ezért a fedett láda szállításáért járó haszon: $10 - 1 = 9$ lesz.

Most csak egy nyitott ládánk van, ennek a telítettsége pedig 38. Tegyük a következő, 20 méretű tárgyat ebbe a nyitott ládába, így a megnövekedett összméret: $38 + 20 = 58$ lesz. A következő tárgy mérete: 17, tegyük ezt a tárgyat is ebbe a nyitott ládába. Mivel a ládatartalom így is csak $58 + 17 = 75$ lesz, ezért a következő 38 méretű tárgyat is ebbe a ládába pakoljuk, mérete: $75 + 38 = 113$ lesz. A láda már fedett, szállítható, a szállítás után az összes haszon így $9 + 10 = 19$ lesz.

Egy új, üres ládát nyitunk az α_1 és β_1 paraméterekkel és a következő 14 méretű tárgyat a nyitott ládánkba tesszük. A következő tárgy mérete: 31 és szintén ebbe a ládába kerül, így a ládánk összmérete: $14 + 31 = 45$ lesz. A következő tárgy, amelynek mérete 28, szintén ebbe a nyitott ládába kerül. A ládánk tartalma már $45 + 28 = 73$ lesz. A következő 32 méretű tárgy ismét ebbe a ládába kerül, a ládatartalom ezáltal $73 + 32 = 105$ lesz, tehát ezt a ládát is sikerült fednünk, szállíthatjuk. Az összes haszon így: $19 + 10 = 29$ lett.

Ha a DNF algoritmust használtuk volna a $Mask$ algoritmus helyett, akkor a haszon $10+10 = 20$ és lett volna egy fedetlen (következésképpen nem szállított) $14+31+28 = 73$ tartalmú nyitott ládánk.

Tehát megállapíthatjuk, hogy esetünkben a $Mask$ algoritmus nagyobb hasznot ért el, azonban működése és így hatékonysága is nagyban függ az α_k és β_k paraméterektől.

Az alábbiakban teszteljük a $Mask$ algoritmus ($Mask_4$) hatékonyságát, az alábbi hat feladatosztály esetén:

1. eset: $I_1 = [0.1; 0.4]$ és $G(k) = 11 - k$.
2. eset: $I_2 = [0.15; 0.25]$ és $G(k) = 11 - k$.
3. eset: $I_3 = [0.1; 1]$ és $G(k) = 11 - k$.
4. eset: $I_1 = [0.1; 0.4]$ és $G(k) = 10.1 - 0.1 * k$.
5. eset: $I_2 = [0.15; 0.25]$ és $G(k) = 10.1 - 0.1 * k$.
6. eset: $I_3 = [0.1; 1]$ és $G(k) = 10.1 - 0.1 * k$.

A tárgyak száma minden esetben továbbra is 1000. Az összehasonlítás eredménye

	Act	$Mask^1$	$Mask^2$	$Mask^3$	$Mask^4$	$Mask^5$	$Mask^6$
1	100%	**100.2%**	100.1%	96.2%	91%	93.7%	88.9%
2	101.1%	91.5%	**101.1%**	94.6%	86.2%	88.8%	88.3%
3	100.6%	100%	101.5%	**106%**	100%	103.9%	100.6%
4	104.3%	105.9%	100.5%	104%	**106.6%**	102%	102.8%
5	101.1%	100.8%	101.1%	103.1%	102.6%	**103.5%**	102.3%
6	108.4%	105%	101.5%	108.3%	113.1%	110%	**114%**

12. táblázat. A $Mask$ algoritmusok összehasonlítása a DNF algoritmust tekintve 100%-nak, hat különböző feladatosztályt figyelembe véve.

megtalálható a 12. táblázatban. Az adott esetekhez tartozó korábbi legjobb eredményeket a táblázat Act oszlopába válogattuk össze.

A 12. táblázatból látható, hogy az új $Mask$ algoritmus a paraméterek megfelelő beállítása esetén nagyobb hasznot ért el, mint az előző algoritmusok (DNF, $H(2)$, $H(3)$, $H(4)$, $SH(2)$, $SH(3)$, $SH(4)$) bármelyike.

$Mask$ algoritmusonként az alábbi paramétereket választottuk:

$Mask^1$: $K = 2$, $\alpha = [0.15, 0.2]$, $\beta = [0.1, 0.3]$.

$Mask^2$: $K = 1$, $\alpha = [0.1]$, $\beta = [0.2]$.

$Mask^3$: $K = 2$, $\alpha = [0.15, 0.15]$, $\beta = [0.3, 0.3]$.

$Mask^4$: $K = 3$, $\alpha = [0.2, 0.2, 0.2]$, $\beta = [0.3, 0.3, 0.3]$.

$Mask^5$: $K = 3$, $\alpha = [0.1, 0.1, 0.1]$, $\beta = [0.25, 0.25, 0.25]$.

$Mask^6$: $K = 3$, $\alpha = [0.1, 0.2, 0.3]$, $\beta = [0.4, 0.5, 0.6]$.

Természetesen nem mindegyik $Mask$ jó minden esetben. Ez nem is lehet célunk. Azonban megfigyelhetjük a következőt: mindegyik feladatosztály esetén van olyan $Mask$ algoritmus (van olyan paraméter-beállítás), amelyik versenyképes a korábbi legjobb algoritmussal, sőt legyőzi azt. Ki kell még találnunk, hogyan tudjuk megadni egy adott feladatosztály esetén a paraméterek megfelelő beállítását. (Paraméter tanuló online algoritmusokkal, tehát a paraméterek megfelelő beállításának megtalálásával foglalkoznak például a következő cikkek: [66, 48].) Ezzel a kérdéssel foglalkozunk mi is a következő fejezetben.

6.8. Algoritmusok evolúciója

Megállapítottuk tehát, hogy a $Mask$ algoritmus paramétereinek helyes megválasztásával jól meg tudjuk oldani a feladatot. A $Mask$ felülmúlja a többi, korábban tárgyalt algoritmust. Az egyetlen problémát a paraméterek helyes megválasztása jelenti. Ebben a fejezetben egy új módszert adunk a ládafedési feladatunk megoldására, a módszert „Algoritmusok evolúciójának" (angolul *evolution of algorithm*, röviden EoA) nevezzük. Ez a módszer más nehéz online feladat megoldására is alkalmazható. Módszerünk nem azonos, azzal, amit „Evolúciós algoritmus"-nak (EA) neveznek.

Az algoritmus a paramétert online módon tanulja meg, futás közben, mint az irodalomban tárgyalt egyéb hasonló algoritmusok is.

Az evolúciós algoritmusok (EA) (vagy más lokális kereső módszerek), valamilyen offline feladat megoldására szolgálnak. Legyen S a feladat megengedett megoldásainak halmaza, ezek között egy szomszédsági struktúrát definiálunk. Az EA először meghatározza a feladat egy x_0 megoldását, majd kiindulva ebből az x_0-ból, ennek a környezetéből választ egy (másik) x_1 megoldást, ha bizonyos kritériumok teljesülnek, akkor x_0-át x_1-re cseréli és ezt a lépést iterálja.

Most másról van szó, először is, feladatunk nem *offline* hanem *online*. (Vagyis bizonyos értelemben az EoA az EA *online* megfelelője.) Nem az offline feladat *megengedett megoldásai* között, hanem az online feladat *megoldó algoritmusai* között hozunk létre szomszédsági struktúrát.

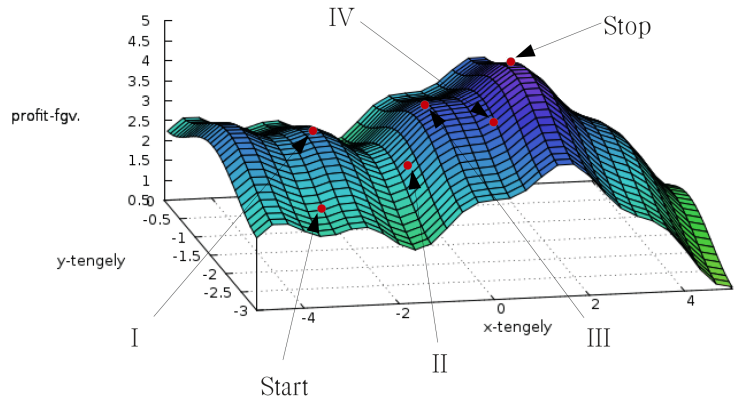
Megengedett megoldások helyett algoritmusokon lépegetünk. A módszer (meghatározunk egy kellően rugalmas algoritmus-családot valamely online feladat megoldására, *az algoritmusok között egy szomszédsági struktúrát definiálunk*, ezután egy lokális kereső módszer segítségével kiválasztjuk az algoritmusok legjobbikát) alkalmazható más (nehéz) feladatok megoldására is.

A főbb különbségeket az EA és EoA között az 13. táblázat szemlélteti.

A szomszédsági struktúra természetes módon határozható meg a különböző $Mask$ algoritmusok között: Egy rögzített $Mask(K;\alpha;\beta)$ algoritmusnak módosítjuk valamelyik

	EA	EoA
A feladat jellege:	offline	online
"amin lépkedünk":	megengedett megoldás	megoldó algoritmus

13. táblázat. Az EA és az EoA összehasonlítása.



4. ábra. Maximumkeresési feladat szemléltetése egy $\sin(x)^3 + \sin(y)^2 - ((\frac{x}{4})^2 - (\frac{y}{20})^2) + 3$ célfüggvénnyel adott példa esetén.

paraméterét: K-t 1-gyel növeljük, vagy csökkentjük, illetve az α, β vektorok valamelyik komponensét csökkentjük, vagy növeljük egy előre rögzített kicsi pozitív konstanssal, (úgy, hogy a megváltozott érték ismét pozitív legyen és kisebb, mint 1). A megfelelő szomszéd választásához szimulált hűtést alkalmaztunk, amely keresési módszer megválasztását az alábbiakban röviden indokoljuk (egy példát adunk az 5. ábrán).

Az 5. ábrán látható maximum-keresési feladat esetén, ha nem alkalmaznánk szimulált hűtést, hanem csak egy egyszerű, mohó lokális keresést, akkor nem garantált, hogy elsőre meg tudjuk találni a globális maximumhelyet. A példafeladatban az algoritmusunk kezdőpontja először a *Start* ponttal adott. Ebből a *Start* pontból egyszerű lokális keresést alkalmazva egy lokális maximumhelybe (I) ragadunk. Válasszunk most egy másik kiindulási pontot, ezt jelöltük az ábrán a II-es ponttal. Innen az egyszerű lokális kereséssel ismét egy lokális maximumhelybe (III) ragad bele. A IV. kiindulási pont kiválasztása esetén a lokális keresés képes megtalálni a globális maximumhelyet és az algoritmus itt

áll meg. Tehát az egyszerű lokális kereső algoritmus eredményessége nagyban függ attól, hogy az algoritmusunk kiindulási pontját melyik pontnak választottuk.

A szimulált hűtés (simulated annealing) egy népszerű, sok feladat megoldására alkalmazott, hatékony metaheurisztika, közeli rokonságban áll a Tabu-kereséssel és a genetikus algoritmusokkal. Részletes ismertetésük megtalálható például itt: [45]. Szimulált hűtés esetén adott egy (nehéz) kombinatorikus optimalizálási offline feladat, amelynek ismerjük, vagy előállítjuk egy x megengedett megoldását, esetleg egy ilyen megoldást valamilyen heurisztikával állítunk elő. Feltételezzük, hogy bármely megengedett megoldásnak könnyen elő tudunk állítani (véletlenszerűen) valamilyen y „szomszédját", vagyis egy olyan másik megengedett megoldást, amely az előbbinek egy „kicsi" változtatásával adódik, vagyis kicsi környezetéből való.

Szimulált hűtés esetén, ezután a két megoldás közül az egyiket választjuk majd, megfelelő szabály szerint. Így, iterációnként egy-egy megengedett megoldásunk van, és ezekkel igyekszünk valamilyen optimális megoldást megközelíteni. A szimulált hűtés azt szimulálja, amikor valamilyen anyagot felmelegítünk, majd lassú hűtés folyamán a részecskék valamilyen alaphelyzetbe kerülnek. A kulcs most az, hogy hogyan választunk az előbbi két megengedett megoldás; x és y közül. Egyrészt, ha y jobb megoldás, mint x, vagyis maximalizálandó célfüggvény esetén nagyobb célfüggvény-érték tartozik hozzá, akkor mindenképpen a jobb megoldást, y-t választjuk. Azonban y-t akkor is elfogadjuk, ha csak egy „kicsivel" rosszabb megoldás mint x, de egyre csökkenő valószínűséggel, és egyre kisebb célfüggvényromlást engedünk meg.

Szimulált hűtés esetén is sok múlik bizonyos stratégiai paraméterek megfelelő beállításán, amelyek az algoritmus hatékonyságát jelentősen befolyásolják.

A $Mask$ algoritmus működése során tehát a paraméter vektorok minden egyes komponensét apránként[3] változtatva haladunk a keresési térben a célfüggvény szerinti maxi-

[3]Azt is be kell állítanunk, hogy az adott ponttól való legkisebb eltérés (vagyis a szomszéd pont távolsága) $\delta = 0.05$ legyen. Ez a δ egység adja meg a legkisebb eltérés mértékét, vagyis a keresés finomságát azzal a megkötéssel, hogy a K stratégiai paraméter csak egész értékeket vehet fel. Vagyis egy-egy komponens δ-val nő, vagy csökken, K pedig 1-gyel nő, vagy csökken.

haszon (ládatelítettség)	végső $Mask$ paraméterei
1847.5 (1.045)	$\alpha = [0.15, 0.2, 0.45]$, $\beta = [0.2, 0.35, 0.4]$, $K = 3$
1854.7 (1.050)	$\alpha = [0.4, 0.45, 0.25]$, $\beta = [0.1, 0.35, 0.3]$, $K = 3$
1827.0 (1.054)	$\alpha = [0.6, 0.55, 0.25]$, $\beta = [0.4, 0.25, 0.3]$, $K = 3$
1833.9 (1.051)	$\alpha = [0.3, 0.2, 0.35]$, $\beta = [0.3, 0.25, 0.3]$, $K = 3$
1817.9 (1.047)	$\alpha = [0.3, 0.3, 0.8]$, $\beta = [0.3, 0.4, 0.45]$, $K = 3$

14. táblázat. A $Mask$ algoritmus szimulált hűtés után kapott paraméterekkel futtatva elért haszon.

mumhely felé.

A 14. táblázatban összefoglaltuk a fenti feladat[4] esetén a kapott paramétereket öt futás eredményét figyelembe véve. Ebben az esetben egyszerű lokális keresést alkalmaztunk, és a lokális keresést javítottuk az EoA felhasználásával.

A 14. táblázatból jól látható, hogy a keresés során a stratégiai paraméterek vektorainak komponensei esetenként eltérőek többszöri futtatás esetén. Ezt azért tapasztaljuk, mert nem minden paraméterváltozás jelent azonnal különbséget a haszonfüggvény értékében.

A 15. táblázatban közlünk egy összehasonlító tesztet, ahol azt vizsgáltuk, hogy vajon az EoA algoritmus által jobb eredményeket kapunk-e, mint a korábbi algoritmusokkal (vagyis tényleg sikerül-e a $MASK$ algoritmus stratégiai paramétereit helyesen beállítani, "kézi-vezérlés", vagyis próbálkozás nélkül). A problémaosztályok ugyanazok voltak, mint korábban is. (Továbbá használunk egyszerű lokális keresést is minden esetben, ahol csak akkor lépünk át a szomszéd algoritmusra, ha az az előzőnél jobb eredményt produkál.)

Amint az a 15. táblázatból látható, az EoA módszerünk az adott feladat esetén tényleg működik, vagyis alkalmas arra, hogy a feladatot hatékonyan megoldó algoritmust konstruáljon. A más algoritmusok által kapott megoldásokat helyenként lényegesen sikerült javítani. A lokális keresés is sok esetben hatékonynak bizonyul, de ahogy várható volt ennél EoA (szimulált hűtés alkalmazása miatt) még hatékonyabb.

Más online feladatok megoldására, hasonló, paramétertanuló algoritmusokkal foglal-

[4] A feladat további adatai: a tárgyak a $[0.15; 0.25]$ intervallumból származnak, a tárgyak száma 1000 és a haszonfüggvény $G(k) = 10.1 - k * 0.1$.

	Act	*LS*	*EoA*
1	100.2%	100.6%	101%
2	101.1%	101.1%	101.7%
3	106%	106.2%	106.9%
4	106.6%	108.7%	109.6%
5	103.5%	103.6%	104.7%
6	114%	116.9%	116.9%

15. táblázat. Az eddigi legjobb eredmények (*Act*), a lokális keresés (angolul: *local search*, röviden: *LS*) és az Algoritmusok Evolúciója módszerek összehasonlítása.

koznak az [48, 49, 66] cikkek.

7. Kétgépes ütemezés újrarendezéssel

A továbbiakban két hasonló gép nem-megszakítható ütemezési feladatát vizsgáljuk arra az esetre, ha kizárólag egyetlen munka átrendezését engedjük meg a munkasorozat végén, a teljes átfutási időt minimalizálva (tehát először az összes munkát ütemezzük, és majd az ütemezés végén egyetlen munkát átütemezhetünk a másik gépre). Ez tehát egy félig-online ütemezési feladat. A fejezetben tárgyalt eredményeink a következő publikációinkon alapulnak: [14, 19, 78]. Először adunk egy rövid bevezetőt a félig online ütemezési feladatokkal kapcsolatban, ezen belül áttekintjük hogy az irodalomban milyen átrendezéses modellekkel foglalkoztak, és az ezekre vonatkozó eredményeket.

7.1. Ütemezésről általában

Tegyük fel, hogy adott valahány, általában m-mel jelölt számú gép, amelyeken adott n számú munkát kell elvégezni úgy, hogy közben valamilyen célfüggvényérték minimumát, vagy maximumát keressük. Ezen gépek mindegyike valamilyen tulajdonsággal rendelkezhet. A feladattípus jelölésére az

$$\alpha|\beta|\gamma$$

szimbólumot használjuk, ahol az α a gépekről, a β a feladatokról, a γ pedig a célfüggvényről ad információt [76].

Például a $P_m||C_{max}$ feladat (az egyik legegyszerűbb feladat, de már ez is az NP-teljes feladatosztályba tartozik) esetén m (ahol m pozitív egész) egyforma és párhuzamos gépünk van. Ez azt jelenti, hogy a munkák végrehajtása minden gépen egységesen ugyanannyi időt vesz igénybe. A munkák száma: n, a munkák végrehajtási ideje pedig: p_i (ahol $1 \leq i \leq n$). A gépeken a munkák megszakítása nem megengedett, tehát ha egy gép egy munkát már elkezdett, akkor annak a végrehajtását már nem szakíthatja félbe. A munkákat ütemezzük, ezen azt értjük, hogy szétosztjuk a munkákat a gépek között (minden gépnek végre kell hajtania a hozzá rendelt munkákat és csakis azokat,

amelyek hozzá lettek rendelve). Tehát a munkák ütemezése megfelel a munkák halmaza egy partíciójának.

Ha egy gép végzett egy munkával, akkor azonnal elkezd a következő munkán dolgozni (ha van még elvégzendő munkája), vagyis nincs várakozási idő. Egy géphez hozzárendelt munkák végrehajtási idejének összegét a gép terhelésének nevezünk. Minden munkának van kezdési és befejezési időpontja. Mivel nincs várakozási idő, ezért egy gép terhelése egyenlő a gépen végrehajtott utolsó munka befejezési idejével (ezt az időpontot a gép átfutási idejének is hívjuk). A gépek átfutási idejeinek maximuma egyenlő az ütemezés átfutási idejével. (Ismét feltesszük, hogy a gépek a 0 időpontban kezdenek dolgozni.)

A C_{max} célfüggvény esetén a teljes átfutási időt minimalizáljuk.

Ebben a fejezetben hasonló gépekkel foglalkozunk (ezeket Q-val jelöljük), ahol a gépek szintén párhuzamosan működnek, de ez a feladat annyiban tér el az előző feladattól, hogy itt minden gépnek van egy saját munkavégzési sebessége, ami akár el is térhet a többi gép munkavégzési sebességétől. Az i-edik gép sebességét s_i-vel jelöljük. Vagyis a j-edik munka elvégzéséhez szükséges idő az i-edik gépen: $\frac{p_j}{s_i}$, (a munka végrehajtási idejét osztjuk a gép sebességével).

Az átlapolást nem engedjük meg, vagyis nem dolgozhat egyetlen munkán egyszerre több gép. Ha az ütemezendő munkákról minden információ adott már az ütemezés megkezdése előtt, akkor az ütemezési feladatot offline-nak hívjuk. Az online esetben pedig egy adott L lista szerinti sorrendben érkeznek a munkák, és amikor egy munka megérkezik, akkor még semmit nem tudunk arról, hogy jön-e még további munka, (és ha igen, akkor az milyen).

Az online algoritmusok hatékonyságát versenyképességi analízissel vizsgálják a következőképpen: Legyen A egy online algoritmus, jelölje C_A az A algoritmus által kapott ütemezés teljes átfutási idejét és legyen C_{OPT} az optimum értéke (az offline ütemezés esetén). Ekkor az A algoritmus versenyképességi aránya a legkisebb olyan C valós szám, amelyre $C_A \leq CC_{OPT}$ teljesül a munkák tetszőleges sorozata esetén. Egy online feladatnak a ρ konstans alsó korlátja, ha nem létezik olyan online algoritmus, amelynek a

versenyképességi aránya ennél kisebb lenne. Továbbá egy online algoritmust optimálisnak nevezünk, amennyiben a versenyképességi aránya megegyezik a feladat alsó korlátjával.

A félig-online (semi online) ütemezések esetén pedig valamit előre tudunk a munkákról, vagy a félig online algoritmus kap valamilyen könnyítést, például puffer használatát. A félig online feladatokra alkalmazott algoritmusokat félig online algoritmusoknak nevezzük. Ezek hatékonyságát az online esethez hasonlóan a versenyképességi analízis segítségével vizsgáljuk. Egy félig online algoritmust optimálisnak nevezünk, ha a versenyképességi aránya megegyezik a feladat alsó korlátjával.

Az alábbiakban áttekintünk néhány félig online ütemezési modellt, és azokra vonatkozó eredményeket.

7.2. Az első online, illetve félig online modellek

R. L. Graham 1966-ban írt cikkében [36] található LPT (= Longest Processing Time) algoritmusa először műveleti idő szerinti csökkenő sorrendbe rendezi a munkákat, majd ebben a sorrendben ütemezi őket. A soron következő munka végrehajtását mindig a lehetséges legkorábbi időpontban kezdjük, vagyis a soron következő munka mindig valamelyik minimális terhelésű gépre kerül. Ha a munkák egy adott L lista szerinti sorrendben érkeznek, és ebben a sorrendben kell őket ütemezni, akkor Graham LS (= List Scheduling, vagyis: lista szerint ütemező) algoritmusát kapjuk. Ez az algoritmus tekinthető az első online algoritmusnak, a párhuzamos egyforma gépek ütemezése esetén. Az algoritmus versenyképességi aránya $2 - \frac{1}{m}$, m gép esetén.

Az első félig online modellek megjelenése Kellerer et al. 1997-es cikkében [55] található. A cikk a következő három ilyen modellt ismerteti:

- előre ismerjük a munkák méreteinek összhosszát,
- felhasználható egy puffer valahány munka ideiglenes tárolására,
- egyszerre két, egymástól független ütemezést készítünk és végül a jobbat választjuk.

Pufferes modellek esetén egy $l \geq 1$ méretű pufferben tárolhatunk l számú munkát, vagyis amennyiben a puffer nincs teljesen megtöltve, akkor egy beérkező munka esetén lehetőségünk van arra, hogy ideiglenesen eltároljuk a pufferben. Ha a puffer tele van, akkor vagy a pufferből kikerül egy munka és ütemezzük, ekkor a beérkező munka a pufferbe mehet, vagy a beérkező munkát ütemezzük. A lista végén a pufferben levő munkákat még ütemezni kell. A második változatot Zhang is vizsgálta 1997-es cikkében [82].

A [55] cikk vizsgálatainak eredménye az lett, hogy mindhárom előbb felsorolt esetben egy-egy optimális, félig online algoritmus versenyképességi aránya $\frac{4}{3}$.

Dósa 2007-ben írt értekezésében [22], az [55] cikk előbbi három feltételéből párosította valamelyik kettőt, kétféleképpen. Az első esetben előre ismert az ütemezendő feladatok méretének összege és egy munka ütemezését mindig tartalékoljuk. Erre megadott egy optimális $\frac{5}{4}$ versenyképességű algoritmust. Megmutatta továbbá, hogy a puffer méretének növelésével a versenyképességi arány nem javítható, vagyis a feladat bármely A megoldó algoritmusának a versenyképességi aránya legalább $\frac{5}{4}$, akkor is, ha a puffer mérete 1-nél nagyobb. A második esetben is előre ismert az ütemezendő feladatok méretének összege, itt viszont egyszerre két ütemezést készítünk és az ütemezés végén a jobbikat választjuk. Erre megadott egy optimális $\frac{6}{5}$ versenyképességű algoritmust. Mindkét esetben igaznak bizonyult, hogy a további félig online feltétel által csökken az elérhető versenyképességi arány.

További pufferes modellekkel foglalkozó publikációk egyebek mellett, például: [24, 28].

7.3. További félig online modellek

Számos félig online modell létezik még, az alábbiakban ismertetünk a teljesség igénye nélkül a modellek és az eredmények közül néhányat. A következő három pontban kétgépes ütemezési feladatokra vonatkozó eredményeket tekintünk át.

- He és Zhang 1999-ben írt cikkében [44] foglalkozik azzal az esettel, amikor előre

ismerjük a maximális és minimális munkaméret arányát, ekkor az optimális algoritmus versenyképességi aránya: $\frac{4}{3}$.

- Seiden 2000-es cikke [71] ismertet olyan félig online modellt, amelyben a munkák monoton csökkenő méretben érkeznek, ekkor az optimális algoritmus versenyképességi aránya: $\frac{7}{6}$.

- Azar 2001-ben publikált cikkében [2] található modellben ismerjük a teljes átfutási idő optimális értékét, ekkor az optimális algoritmus versenyképességi aránya: $\frac{4}{3}$, két gép esetén.

Dósa 2007-ben írt értekezésében foglalkozott a $P_3|online|C_{max}$ feladatnak azzal a „tightly grouped"-nek nevezett, félig online változatával is, amikor előre tudjuk, hogy a munkák végrehajtási ideje közel egyforma. Pontosabban előre tudjuk, hogy a munkák hossza a p és rp konstansok között van ($p > 0$, $r \geq 1$). Ha az r konstans túl nagy, akkor az előbbi információ nem sokat ér, mert megmutatta, hogy ha $r \geq 6$, akkor nincs olyan félig online algoritmus, amelynek versenyképességi aránya jobb lenne, mint valamely optimális online algoritmusé. Érdemes tehát azt vizsgálni, hogy mekkora az a maximális r méretarány, amelynél egy optimális félig-online algoritmus versenyképességi aránya jobb, mint az egy-szerű online eseté. Az $r \in (2, \frac{5}{2}]$ esetben megadott egy optimális algoritmust, melynek versenyképességi aránya $\frac{3}{2}$. Az $r \in (\frac{5}{2}, 3]$ esetben megadott egy közel optimális algoritmust, melynek versenyképességi aránya $\frac{4r+2}{2r+3}$. Az $r \in (3, 6)$ esetben megadott egy algoritmust, melynek versenyképességi aránya $\frac{5}{3}$-nál szigorúan kisebb (az LS algoritmusé pedig $\frac{5}{3}$). Tehát három gép és közel egyforma végrehajtási idők esetén lehetséges az LS algoritmusnál hatékonyabb algoritmust megadni, ha a méretarány kettő és hat közötti szám, tehát LS nem minden esetben optimális. Továbbá az optimális félig-online algoritmus versenyképességi aránya erősen függ az r paraméter értékétől.

7.4. Hasonló gépek ütemezése

Két hasonló gép online megszakítás nélküli ütemezésével foglalkozik Epstein et al. [29] cikke. Ebben a modellben a gépek száma kettő, az M_1 és az M_2 gépek működési sebessége pedig $s_1 = 1$, és $s_2 = s \geq 1$, valamint $\phi \approx 1.618$ pedig az aranymetszés aránya. A célfüggvény a teljes átfutási idő, amelyet minimalizálni kell. A cikk közöl egy optimális algoritmust, amelynek versenyképességi aránya $\frac{2s+1}{s+1}$ ha $1 < s \leq \phi$, valamint $\frac{s+1}{s}$ ha $s \geq \phi$. A versenyképességi arány maximuma éppen ϕ. Mint látni fogjuk, ennél lényegesen kisebb lesz az optimális versenyképességi arány, ha átrendezésre is lehetőségünk van, lásd: 5. ábra.

7.5. Ütemezés korlátos átrendezéssel, bármely időpontban

Az utóbbi időben több modellt definiáltak, amelyben korlátozott módon lehetőség van, néhány, már korábban ütemezett munkának az átrendezésére, lásd például [69].

Az egyik ilyen modellt Dósa, Wang, Han és Guo 2011-es cikkében [27] definiálta és nevezte el *REAR* modellnek, továbbá a feladathoz algoritmusokat is megadtak. A *REAR* modellben a munkák egyesével jönnek egy adott lista szerinti sorrendben. Bármelyik pillanatban, amikor egy új munka érkezik, akkor legfeljebb K darab már ütemezett munkát eltávolíthatunk az ütemezésből, és rögtön ezután az összes eltávolított munkát és az újonnan jött munkát (ez együtt legfeljebb $K+1$ munka) azonnal hozzá kell rendelnünk a gépekhez, még a következő munka érkezése előtt. Ez a modell a Dósa és Epstein 2010-es cikkében [24] található „online ütemezés pufferrel", röviden: *BUFF* modell nevű hasonló feladatnak egy relaxációja. (A *BUFF* modell esetén adott egy puffer, ahol legfeljebb K tárgyat tárolhatunk, vagyis ezek ütemezését így késleltetni tudjuk.)

A két modell összehasonlítása a [19] cikkben történik.

Tegyük fel, hogy az M_1 gép sebessége 1 és az M_2 gép sebessége pedig $s \geq 1$. A *versenyképességi arány tekintetében* azt kapjuk, hogy:

- Ha $s \geq \sqrt{3}$, vagy $K \geq 2$, akkor a *REAR* modell optimális versenyképességi aránya

megegyezik a *BUFF* modell optimális versenyképességi arányával, ami $\frac{s+2}{s+1}$ (vagyis $s \geq \sqrt{3}$, vagy $K \geq 2$ esetén a feladat alsó korlátja egyenlő az algoritmusok versenyképességi arányával, tehát optimálisak az algoritmusok a megfelelő modellekben).

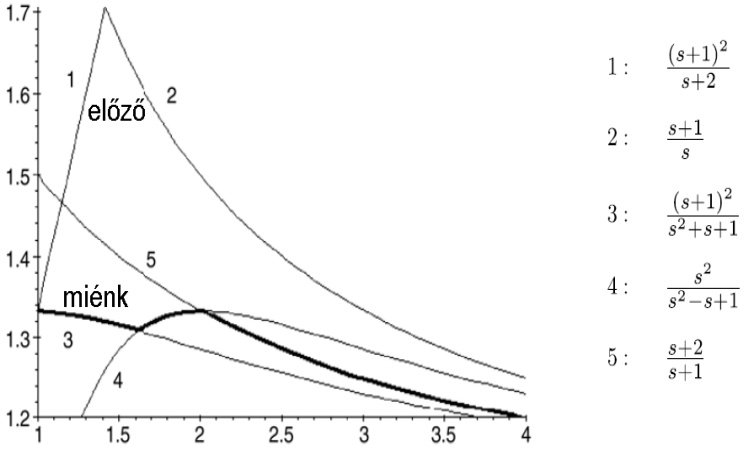
- Ha $q \leq s < \sqrt{3}$ (ahol $q \approx 1.618$, ami az aranymetszés aránya) és $K = 1$, akkor a *REAR* modell esetén $\frac{s+1}{2}$ alsó korlát.
- Ha $1 \leq s < q$ és $K = 1$, akkor a *REAR* modell esetén $\frac{(s+1)^2}{s^2+s+1}$ alsó korlát.
- Ha $s_0 < s < \sqrt{3}$ (ahol $s_0 \approx 1.3247$, ami az $s^3 - s - 1 = 0$ gyöke $s \in [1, 2]$ esetén) és $K = 1$, akkor a *REAR* modell esetén a felső korlát: $\frac{s+\sqrt{5s^2+8s+4}}{2(s+1)}$.
- Ha $1 \leq s \leq s_0$ és $K = 1$, akkor a *REAR* modell esetén az optimális arány: $\frac{(s+1)^2}{s^2+s+1}$.
- Ha $1 \leq s < \sqrt{2}$ és $K = 1$, akkor a *BUFF* modell esetén az alsó korlát: $\frac{(s+1)^2}{s^2+s+1}$ és a felső korlát ebben az esetben: $\frac{2(s+1)}{s+2}$.
- Ha $s \geq \sqrt{2}$ és $K = 1$, akkor a *BUFF* modell esetén az alsó és a felső korlát megegyezik: $\frac{s+2}{s+1}$, tehát a megadott algoritmus optimális.

Természetesen, ha $K = 0$ már ütemezett munkát engedünk meg átrendezni és a gyors gép sebessége 1 ($s = 1$), akkor a feladatot visszavezettük az egyik legalapvetőbb ütemezési feladatra: két azonos sebességű párhuzamos géphez történő munkák online hozzárendelése a teljes átfutási idő minimalizálásával.

7.6. Három másik modell: átrendezés a végén

Tan és Yu 2008-as cikke [74] három modellt definiál, mindhárom különbözik az előbb definiált modelltől. A három modell:

- Az ütemezés végén, bármelyik gép utolsó munkája újrarendezhető másik gépre, jelölése: P_E,



5. ábra. A vastag görbék jelentik a javított felső korlátokat, amelyek megegyeznek az alsó korlátokkal, tehát a felső korlátaink optimálisak.

- Az ütemezés végén, a sorozat utolsó legfeljebb K darab munkája újrarendezhető, jelölése: P_L,
- Az ütemezés végén, tehát a lista megérkezése és a munkák ütemezése *után*, bármely K munka újrarendezhető, jelölése: P_A.

Liu és társai 2009-es cikkben [63] ismertetik a *HSF-1* algoritmust, amelynek versenyképességi aránya (a P_A és a P_L modellben egyaránt): $\frac{s+1}{s}$, ha: $1 \leq s \leq \sqrt{2}$. Ebben a cikkben adják meg az *EX-RA* algoritmust is, amelynek versenyképességi aránya (a P_A modellben): $\frac{(s+1)^2}{s+2}$, ha: $\sqrt{2} \leq s \leq 2$. Az *EX-RA* algoritmus az $1 \leq s \leq \sqrt{2}$ esetben felülmúlja a *HSF-1* algoritmust. A továbbiakban a célunk az, hogy olyan algoritmust adjunk meg, amelyik mindkét előző algoritmusnál jobb versenyképességi aránnyal rendelkezik a megadott feltételek esetén.

Mi *REND* modellnek (a rövidítés az angol „rearrangement at the end" kifejezésből származik) neveztük azt a modellt, amit Tan és Yu 2008-as cikkében P_A-val jelöl. Chen, Lan, Benkő, Dósa és Han a 2011-es cikkben [14] a *REND* modellben ad meg két optimális online algoritmust két különböző esetre, a cikk javít a korábbi algoritmuson.

- Az első vizsgált eset amikor $s \geq 2$. Az optimális algoritmus csak legfeljebb 1 munkát rendez át, vagyis nem kapunk jobb arányt, ha K nagyobb. A cikkünkben megadott LC algoritmus (a rövidítés az angol „Largest Change" kifejezésből származik) versenyképességi aránya: $\frac{s+2}{s+1}$.

- Az $1 \leq s \leq 2$ esetben, ha K legalább 2 lehet, akkor szintén sikerült optimális algoritmust kidolgozni (tehát nem kaphatunk jobb versenyképességi arányt akkor sem, ha $K > 2$). A cikkünkben megadott SMF algoritmus (a rövidítés az angol „Slow Machine First" kifejezésből származik) versenyképességi aránya: $\frac{(s+1)^2}{s^2+s+1}$, ha: $s \in [1, \frac{1+\sqrt{5}}{2})$; és $\frac{s^2}{s^2-s+1}$, ha: $s \in [\frac{1+\sqrt{5}}{2}, 2)$.

Tehát azokat az eseteket, amikor $s \geq 2$, illetve ha $s \in [1, 2]$ és $K \geq 2$, már tisztáztuk, így marad az az eset, amikor $s \in [1, 2)$ és $K = 1$, amely esettel a következő alfejezetekben foglalkozunk.

7.7. A modell pontos definíciója

Ezek után az online, kétgépes, nem-megszakítható ütemezési feladatot vizsgáljuk, egyetlen munka átrendezését engedjük meg, és csak a munkasorozat végén. A teljes átfutási időt minimalizáljuk. Vagyis a P_A feladattal foglalkozunk, abban az esetben, amikor $K = 1$. Megadunk egy javított algoritmust, az $1 \leq s \leq 2$ esetre. A korábbi legjobb eredmény $s \leq \sqrt{2}$ esetén az [63] cikkben, illetve $s \geq \sqrt{2}$ esetén a [14] cikkben található. A javított algoritmus a [78] cikkünkben szerepel.

Bemenet: Adott az elvégzendő munkák listája, $J = \{j_1, j_2, \ldots, j_n\}$. A munkák száma n (amit előre nem ismerünk), a munkák végrehajtási ideje (processing time), vagy egyszerűen szólva mérete p_i, ahol: $1 \leq i \leq n$. Bármely munkát két gép (M_1 és M_2) valamelyike végzi el (tehát pontosan egyikük), ahol az M_1 gép munkavégzési sebessége 1 egység, az M_2-es gép sebessége pedig $s \geq 1$ Vagyis hasonló gépeket ütemezünk (angolban uniform machines).

Kimenet: Ütemezzük a munkákat az M_1 és M_2 gépeken úgy, hogy a két gép átfutási idejeinek maximumát, vagyis a teljes átfutási időt minimalizáljuk.

Újrarendezés: A vizsgált modellben lehetőségünk van arra, hogy *miután* az összes munkát ütemeztük már a gépekre, és értesültünk arról hogy a sorozatnak vége van, ekkor legfeljebb $K = 1$ már ütemezett munka ütemezését átrendezhetjük. A K szám előre adott konstans.

A munkák előzetes ütemezését *ütemezési fázisnak* nevezzük, az ezután történő átrendezést pedig *átrendezési fázisnak*.

Legyen t egy tetszőleges index, ahol $1 \leq t \leq n$. A következő jelölést vezetjük be, legyen $J_t = \{j_1, j_2, \ldots, j_t\}$. Legyen továbbá L_t^i az M_i gép terhelése, közvetlenül a j_t munka ütemezése után, ahol $i = 1, 2$, az ütemezési fázis során. Legyen továbbá

$$\Theta_t = \max\left\{\frac{l_t}{s}\ \frac{P_t}{s+1}\right\} \tag{4}$$

az a természetesen adódó alsó korlát, ahol l_t az első t munka közötti legnagyobb méretet jelöli, P_t pedig az eddigi összméretet. Amennyiben a t idő jelölése nélkül is egyértelmű hogy miről lesz szó, vagy t az aktuális munka indexe, ilyen esetekben egyszerűen csak az L_i és a Θ jelölések fogjuk használni a pontosabb L_t^i és a Θ_t helyett jelölések helyett.

Továbbá azt mondjuk, hogy az M_1 gép *alulterhelt*, ha $L_t^1 \leq \rho(s)\Theta_t$ teljesül, különben pedig *túlterheltnek* nevezzük. Ennek megfelelően az M_2 gép alulterhelt, ha $L_t^2 \leq s\rho(s)\Theta_t$, egyébként pedig túlterhelt. Az előbbi definíciókban valamely A online algoritmus vonatkozásában adtuk meg ezen fogalmakat, és $\rho(s)$ az algoritmus (egyértelműen meghatározott) versenyképességi arányát jelenti.

1. lemma. *Legyen A egy online algoritmus a feladatra. Ha az ütemezés az M_1 gépen túlterhelt, akkor $L_1 > \frac{\rho(s)}{s+1-\rho(s)}L_2$. Ha az ütemezés az M_2 gépen túlterhelt, akkor $L_2 > \frac{\rho(s)s}{s+1-\rho(s)}L_1$ teljesül a gépek terhelésének aktuális értékeire.*

Bizonyítás. Először is, ha $L_2 = 0$, ebből azonnal következik az első állítás, és hasonlóan, ha $L_1 = 0$, ebből pedig a második állítás. Tegyük fel ezek után, hogy az előbbiek egyike sem 0. Definíció szerint $\Theta_t = max\{\frac{l_t}{s}\ \frac{L_t^1+L_t^2}{s+1}\}$, ebből rögtön következik,

hogy

$$L_1 + L_2 \leq (s+1)\Theta.$$

Ha az M_1 gép túlterhelt, akkor:

$$L_1 > \rho(s)\Theta$$

teljesül, emiatt tehát az előző két egyenlőtlenséget összevetve

$$L_2 \leq (s+1)\Theta - \rho(s)\Theta$$

adódik. Ezért ebből az alábbi következik:

$$\frac{L_1}{L_2} > \frac{\rho(s)\Theta}{(s+1)\Theta - \rho(s)\Theta} = \frac{\rho(s)}{s+1-\rho(s)},$$

amiből kapjuk az első állítást.

Ha pedig az M_2 gép túlterhelt, akkor:

$$L_2 > s\rho(s)\Theta,$$

emiatt

$$L_1 \leq (s+1)\Theta - s\rho(s)\Theta,$$

amiből következik, hogy

$$\frac{L_2}{L_1} > \frac{s\rho(s)\Theta}{(s+1)\Theta - s\rho(s)\Theta} = \frac{s\rho(s)}{s+1-s\rho(s)},$$

ebből pedig adódik a második állítás. ∎

7.8. A javított online ütemező algoritmus a $K = 1$ és $1 \leq s \leq 2$ esetén

Korábban a [14] cikkben már megadtunk egy ütemező algoritmust a $K = 1$, $1 \leq s \leq 2$ esetre. Az algoritmus versenyképességi aránya $\rho'(s)$, amely az alábbi szerint definiálható:

$$\rho'(s) = \begin{cases} \frac{(s+1)^2}{s+2}, & 1 \leq s < \sqrt{2}, \\ \frac{s+2}{s+1}, & \sqrt{2} \leq s \leq 2. \end{cases}$$

Az algoritmus leírása, az algoritmus versenyképességi arányát megadó alsó és felső becslésekre vonatkozó tételek és azok bizonyításai (terjedelmi okok miatt) ebben az értekezésben nem kerülnek bemutatásra. Azonban megadunk az alábbiakban egy javított algoritmust, és ennek teljes vizsgálatát, mármint a versenyképességre vonatkozó tételeket a bizonyításokkal együtt. Az algoritmus versenyképességi aránya legyen $\rho(s)$, amely az alábbi szerint definiálható:

$$\rho(s) = \begin{cases} \frac{2(s+1)}{s+2}, & 1 \leq s \leq \sqrt{2}, \\ \frac{s+2}{s+1}, & \sqrt{2} < s \leq 2. \end{cases}$$

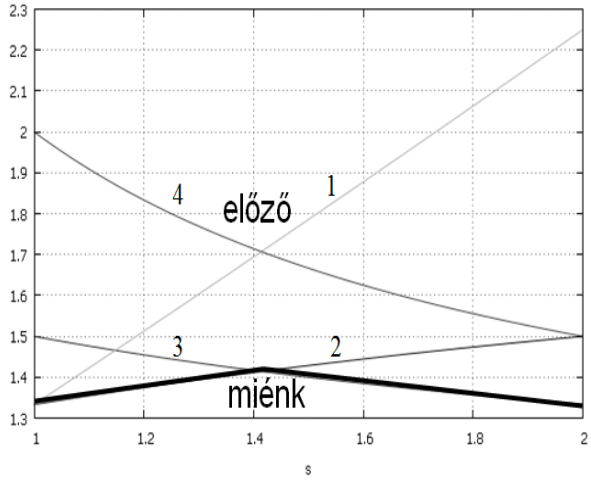
A korábbi és a javított versenyképességi arányok görbéit az alábbi ábrán mutatjuk be. Megjegyezzük, hogy $1 \leq s < \sqrt{2}$ esetén a korábbi $\frac{(s+1)^2}{s+2}$ arány a [63] cikkből származik, ők a $\sqrt{2} < s \leq 2$ intervallumra az $\frac{s+1}{s}$ felső korlátot adták meg.

Az algoritmus leírása az alábbi: adott két fázis, az első fázis az ütemezés, a második pedig az újrarendezés fázisa. Az ötlet az, hogy az ütemezési fázisban a gyors gépet végig alulterheltnek ütemezzük, és ha muszáj, akkor a lassú gépet terheljük túl. Az újrarendezési fázisban pedig a legnagyobb munkát áthelyezzük a lassú gépről a gyors gépre, de csak akkor, ha ez szükséges, vagyis ezáltal javul a teljes átfutási idő.

Az algoritmust JO algoritmusnak nevezzük, ami a „the slow machine Just Overloaded” rövidítése.

JO Algoritmus:

- **Ütemezési fázis**: amikor a p_t méretű j_t munka megérkezik:



1: $\frac{(s+1)^2}{s+2}$

2: $\frac{2(s+1)}{s+2}$

3: $\frac{s+2}{s+1}$

4: $\frac{s+1}{s}$

6. ábra. A javított felső korlátok az $I = [1; 2]$ intervallumon K=1 esetén.

1. Frissítjük a Θ_t alsó korlátot.
2. Legyen Δ a már korábban az M_1 géphez rendelt legnagyobb munka mérete. Ha $p_t + L^1_{t-1} - max\{\Delta, p_t\} \leq \rho(s)\Theta_t$, akkor a j_t munka az M_1 gépre kerül, (vagyis más szóval ez azt jelenti hogy ezt e lépést akkor hajtjuk végre, ha a j_t munkának az M_1 gépre ütemezésével, és eztán az M_1 gépről legfeljebb egy munkát áthelyezve az M_2 gépre az M_1 gép alulterhelt marad illetve azzá válik).
3. Minden más esetben a j_t munkát az M_2 gépre ütemezzük.

- **Újrarendezési fázis**: ha az input végetért.

1. Helyezzük át a legnagyobb munkát az M_1 gépről az M_2 gépre, feltéve, ha ez szükséges (vagyis, ha a munka áthelyezésével csökken a teljes átfutási idő).

2. lemma. *Az M_2 gép az ütemezési fázis teljes idejében alulterhelt.*

Bizonyítás. A bizonyításhoz teljes indukciót alkalmazunk. Nem nehéz belátni, hogy az első munkát az algoritmus az M_1 géphez rendeli hozzá, és ezután az M_2 gép alulterhelt a $t = 1$ időpillanatban. Tegyük fel, hogy az M_2 gép alulterhelt, miután a j_i munkát az

egyik géphez hozzárendeltük az ütemezési fázisban, ahol $i \geq 1$. Azt kell belátnunk, hogy az M_2 gép még mindig alulterhelt, miután a j_{i+1} munkát is ütemeztük.

Ha a j_{i+1} munkát az M_1 géphez rendeltük hozzá, akkor az M_2 gép terhelése nem változik, ezáltal M_2 alulterhelt marad. Tegyük fel tehát, hogy a j_{i+1} munka az M_2 géphez lett hozzárendelve. Ekkor

$$L^1_{i+1} = L^1_i,$$

továbbá

$$p_{i+1} + L^1_i - max\{p_{i+1}, \Delta\} > \rho(s)\Theta_{i+1}.$$

Ebből következik, hogy

$$L^1_{i+1} = L^1_i > \rho(s)\Theta_{i+1}$$

teljesül, ahol Δ az M_1 géphez korábban hozzárendelt legnagyobb munka mérete, p_{i+1} pedig a j_{i+1} munka mérete. Mivel

$$\Theta_{i+1} \geq \frac{L^1_{i+1} + L^2_{i+1}}{s+1},$$

valamint $\rho(s) \geq 1$, azt kapjuk, hogy

$$L^2_{i+1} \leq (s+1)\Theta_{i+1} - L^1_{i+1} < (s+1)\Theta_{i+1} - \rho(s)\Theta_{i+1} = (s+1-\rho(s))\Theta_{i+1} < s\rho(s)\Theta_{i+1},$$

ez pedig pontosan azt jelenti, hogy a második gép alulterhelt. (Az utolsó egyenlőtlenség a $\rho(s) \geq 1$ egyenlőtlenségből következik.) Tehát M_2 bármelyik munka ütemezése után alulterhelt marad az ütemezési fázis során. ∎

3. lemma. *Az ütemezési fázis bármely $t \geq 1$ időpontjában*

$$L^1_t \geq \frac{\rho}{s+1-\rho} L^2_t$$

teljesül, *ahol* L_t^i *az* M_i *gép terhelése.*

Bizonyítás. A lemmát ismét indukcióval bizonyítjuk.

Az algoritmus leírásából fakadóan az első munkát az M_1 géphez kell hozzárendelni. Tehát a $t = 1$ időpillanat után azt kapjuk, hogy

$$L_t^1 \geq \frac{\rho}{s+1-\rho} L_t^2,$$

hiszen ekkor még $L_t^2 = 0$. Tegyük fel, hogy a lemma állítása teljesül a $t = i$ időpillanatban, vagyis az i-edik munka ütemezése utáni pillanatban, ez pedig azt jelenti, hogy

$$L_i^1 \geq \frac{\rho}{s+1-\rho} L_i^2.$$

Most legyen $t = i + 1$. Ha a j_{i+1} munka az M_1 gépre kerül, akkor:

$$L_{i+1}^1 > L_i^1 \geq \frac{\rho}{s+1-\rho} L_{i+1}^2.$$

Ellenkező esetben a j_{i+1} munka az M_2 gépre kerül, ez viszont az algoritmus szabálya miatt azért történt, mert a $t = i + 1$ időpillanatban az M_1 gép túlterhelt lett (vagy már előtte is az volt). Az 1. lemmát alkalmazva tehát azt kapjuk, hogy

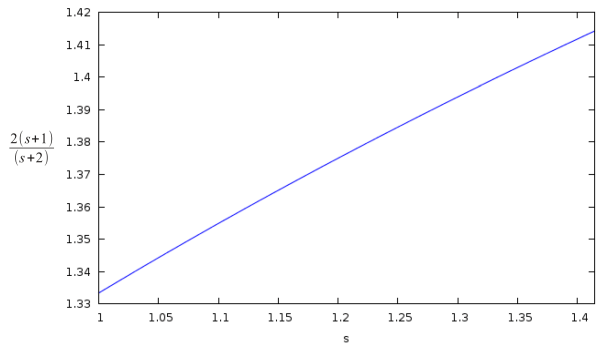
$$L_{i+1}^1 > \frac{\rho(s)}{s+1-\rho(s)} L_{i+1}^2.$$

Ez pedig pontosan az, amit jelen lemma állít. ∎

7.9. Az $1 \leq s \leq \sqrt{2}$ intervallum

Ebben az alfejezetben belátjuk, hogy a *JO* algoritmus versenyképességi aránya $\rho_1(s) = \frac{2(s+1)}{s+2}$, az $1 \leq s \leq \sqrt{2}$ intervallumban. Az előbbi felső korlátot, mint az s sebesség függvényét a következő ábra szemlélteti.

4. lemma *A JO algoritmus versenyképességi aránya* $\rho_1(s) = \frac{2(s+1)}{s+2}$, *az* $1 \leq s \leq \sqrt{2}$ *intervallumban.*



7. ábra. A $\frac{2(s+1)}{s+2}$ függvény az $I = [1; \sqrt{2}]$ intervallumon ábrázolva.

Bizonyítás. Legyen L_i (L_i') az M_i gép terhelése közvetlenül az ütemezési fázis után (az újrarendezés elvégzésével), ahol $1 \leq i \leq 2$.

Emlékeztetünk arra, hogy az algoritmus újrarendezési fázisában a következőt tesszük: a legnagyobb munkát áthelyezzük az M_1 gépről az M_2 gépre, de csak akkor, ha ezáltal a teljes átfutási idő csökken, különben semmiféle átrendezés nem történik.

Az újrarendezés során a teljes átfutási idő nem növekszik, emiatt teljesül a következő egyenlőtlenség:

$$\max\left\{L_1, \frac{L_2}{s}\right\} \geq \max\left\{L_1', \frac{L_2'}{s}\right\}.$$

Emiatt, ha mindkét gép alulterhelt az ütemezési fázis után, vagyis

$$\max\left\{L_1, \frac{L_2}{s}\right\} \leq \rho\Theta$$

teljesül, akkor természetes módon

$$\max\left\{L_1', \frac{L_2'}{s}\right\} \leq \rho\Theta$$

is teljesül, és a lemma bizonyításával végeztünk is.

Továbbá a 2. lemma szerint az M_2 gép végig alulterhelt az ütemezési fázis során,

emiatt egyedül annak az esetnek a vizsgálata maradt hátra, amikor az M_1 gép túlterhelt az ütemezési fázis végén. A következőkben bebizonyítjuk, hogy ha ebben az esetben az M_1 gépre ütemezett legnagyobb munkát áthelyezzük az M_2 gépre, akkor az M_2 gép nem válik túlterheltté. Az algoritmikus szabálynak megfelelően M_1 nem marad túlterhelt az áthelyezés után.

Legyen tehát az X munka a legnagyobb méretű munka az M_1 gépen az ütemezési fázis elvégzése után. Tegyük fel, hogy az M_2 gép túlterheltté válik, miután az X munkát áthelyezzük az M_1 gépről az M_2 gépre. Legyen I egy minimális ellenpélda, vagyis olyan input, amelyre a lemma állítása nem teljesül (ha van ilyen), és minimális a munkák számát illetően. Legyen x és y az M_1 és az M_2 gép terhelése éppen abban a pillanatban, mielőtt az X munka az M_1 gépre lett ütemezve.

Továbbá legyen az X munka ütemezése után, az M_1 és M_2 gépekre ütemezett munkák méreteinek összege rendre u és v. (Amennyiben X az utolsó munka, akkor természetszerűleg $u = 0$ és $v = 0$, különben legalább az egyik közülük pozitív.) Jelöljük az X munka méretét az alábbi módon: Δ. Ekkor a következő egyenlőtlenségeket írhatjuk fel:

$$x + y + u + v + \Delta \leq (s+1)\Theta, \tag{5}$$

$$x + u + \Delta > \rho\Theta, \tag{6}$$

$$y + v + \Delta > s\rho\Theta, \tag{7}$$

ahol Θ az alsó korlát végleges értéke. Az (5) egyenlőtlenség az alábbi tényből következik: a két gépre ütemezett munkák összmérete legfeljebb $(s+1)\Theta$. A (6) egyenlőtlenség abból következik, hogy az M_1 gép túlterhelt még az áthelyezés előtt. A (7) egyenlőtlenség pedig abból a feltételezésből következik, hogy az M_2 gép túlterheltté válik az áthelyezés után.

Ha az (5) egyenlőtlenséget kivonjuk a (6) és a (7) összegéből, akkor a következő egyenlőtlenséget kapjuk:

$$\Delta > (s+1)\Theta(\rho - 1). \tag{8}$$

Ekkor az (5) és a (8) egyenlőtlenségekből következik, hogy:

$$x + y + u + v \leq (s+1)\Theta - \Delta < (s+1)\Theta(2-\rho). \tag{9}$$

Ezekből pedig

$$x + y + u + v < (s+1)\Theta(2-\rho) = \rho\Theta$$

következik, alkalmazva ρ-ra a $\rho = \frac{2(s+1)}{s+2}$ helyettesítést.

Ez azt jelenti, hogy ha vannak olyan munkák, amelyek az X munka után érkeznek, akkor az összes ilyen munkát az M_1 gépre kell ütemezni az algoritmusunk szabályainak megfelelően, vagyis $v = 0$ és $u > 0$.

Ezáltal ellentmondást kapunk az ellenpéldánk minimalitására vonatkozólag, mivel ezen (az X munka után érkező) munkák törlése után a Θ alsó határ értéke csökken, vagy változatlan marad, a gyorsabb gép terhelése nem változik, így egy kisebb ellenpéldát kapnánk

Ezek szerint következik tehát, hogy az X munka szükségképpen az utolsó munka. Ekkor az (5) - (9) egyenlőtlenségek változatlanul érvényben maradnak, csak erősebb formában, hiszen tudjuk már hogy $u = v = 0$.

Menjünk tovább. Éppen mielőtt az X munka megérkezne, a 3. lemma alapján

$$x > \frac{\rho}{s+1-\rho} y$$

teljesül, vagy ezzel ekvivalens módon (miután itt is alkalmaztuk a $\rho = \frac{2(s+1)}{s+2} \Rightarrow \frac{\rho}{s+1-\rho} = \frac{2}{s}$ helyettesítést) azt kapjuk, hogy

$$x + y > \frac{s+2}{s} y, \tag{10}$$

illetve

$$\frac{s}{s+2}(x+y) > y. \tag{11}$$

Ekkor a (10), (11) valamint (7) egyenlőtlenségből nyerjük, hogy

$$\frac{s}{s+2}(x+y) + \Delta > s\rho\Theta,$$

emiatt

$$(x+y) + \frac{s+2}{s}\Delta > (s+2)\rho\Theta = 2(s+1)\Theta. \tag{12}$$

Összevetve a (12) egyenlőtlenséget (5)-tel, azt kapjuk, hogy

$$\left(\frac{s+2}{s} - 1\right)\Delta > (s+1)\Theta,$$

és emiatt az következik, hogy a $\Delta > \frac{s(s+1)}{2}\Theta \geq s\Theta$ egyenlőtlenség-lánc teljesül. Itt felhasználtuk azt is hogy $s \geq 1$.

Másrészt, az alsó korlát definíciója szerint tudjuk, hogy:

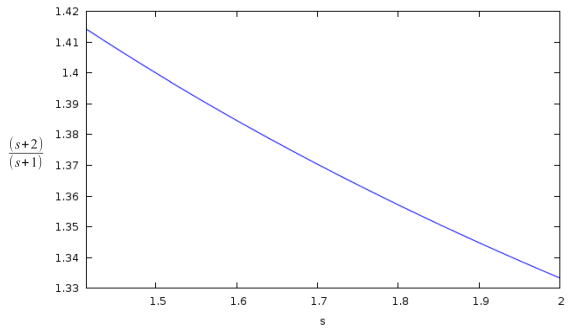
$$\Theta \geq \frac{\pi}{s} \geq \frac{\Delta}{s}$$

teljesül, ahol $\pi \geq \Delta$ a legnagyobb munka mérete, és emiatt $s\Theta \geq \Delta$. Ezáltal ellentmondáshoz jutottunk, vagyis az indirekt feltevésünk (az M_2 gép túlterheltté válik) nem igaz, vagyis következik, hogy az M_2 gép alulterhelt lesz az átrendezés után. ∎

7.10. A $\sqrt{2} < s \leq 2$ intervallum

Ebben az alfejezetben bizonyítjuk, hogy a *JO* algoritmus versenyképességi aránya $\rho(s) = \frac{s+2}{s+1}$, a $\sqrt{2} < s \leq 2$ intervallumon. A felső korlátot a következő ábra szemlélteti.

5. lemma. *A JO algoritmus versenyképességi aránya* $\rho(s) = \frac{s+2}{s+1}$, *a* $\sqrt{2} < s \leq 2$ *intervallumon.*



8. ábra. Az $\frac{s+2}{s+1}$ függvény az $I = [\sqrt{2}; 2]$ intervallumon ábrázolva.

Bizonyítás. Az állítást a 4. Lemmához hasonló módon bizonyítjuk, emiatt csak az alábbi esetet vizsgálata szükséges: az M_1 gép túlterhelt az ütemezési fázis elvégzése után és az M_2 gép szintén túlterhelt (túlterheltté válik) ha az M_1 gépre ütemezett legnagyobb méretű munkát az M_2 gépre ütemezzük át. Az egyéb esetekben végeztünk is a bizonyítással. A következő bizonyításban két esetet tárgyalunk. Az első esetben azt bizonyítjuk, hogy az áthelyezés után az M_2 gép nem lehet túlterhelt. A másik esetben azt bizonyítjuk, hogy ha az M_1 gép legnagyobb munkáját áthelyezzük az M_2 gépre, akkor az M_2 gép ugyan túlterheltté válhat, vagyis a terhelése lehet hogy nagyobb lesz mint $s\rho\Theta$, de ekkor is teljesül, hogy az M_2 gép terhelése csak legfeljebb $s\rho OPT$, ahol OPT az optimális érték.

Tegyük fel, hogy ez az állítás nem teljesül. Legyen I egy minimális ellenpélda az input elemszámára nézve. Legyen az X munka a legnagyobb méretű munka amely az M_1 gépre lett ütemezve az ütemezési fázis során. Legyen ennek a munkának a mérete: Δ. Legyen Y az a munka, amit az M_2 géphez utoljára rendeltünk hozzá az ütemezési fázis során (biztosan van ilyen munka, a feltevésünk miatt).

1. eset. Az Y munka megelőzi az X munkát a sorozatban. A 4. Lemmában alkalmazott tárgyalásmódhoz hasonlóan azt fogjuk bizonyítani, hogy $\Delta > s\Theta$. Ez ellentmondásban áll azzal a ténnyel, hogy $\Delta \leq s\Theta$. A bizonyítás részletei következnek az

alábbiakban.

Legyen x és y az M_1 és az M_2 gép terhelése pontosan az X munka ütemezése előtti pillanatban (ekkor $x, y \geq 0$ természetes módon teljesül és fontos megjegyezni, hogy az Y munka mérete beleszámít y-ba.) Legyen az X munka után érkező és az M_1 géphez rendelt munkák összmérete: u. Ekkor az újrarendezési fázis előtti terhelése a gépeknek pontosan $x + \Delta + u$, illetve y. Jelölje Θ az alsó korlát végső értékét. Ekkor teljesül a következő két egyenlőtlenség:

$$x + y + \Delta + u \leq (s+1)\Theta, \tag{13}$$

$$y + \Delta > \rho \cdot s \cdot \Theta. \tag{14}$$

A (13) egyenlőtlenségből következik, hogy:

$$x + y \leq x + y + u \leq (s+1)\Theta - \Delta. \tag{15}$$

Itt jegyezzük meg, hogy az X munka ütemezése előtt az M_1 gép túlterhelt volt (mivel az M_1 gép az ütemezési folyamat bármely pillanatában túlterhelt) és ekkor a gépek terhelése pontosan x és y. Ekkor a 3. Lemma alapján azt kapjuk, hogy

$$x > \frac{s+2}{s^2+s-1}y,$$

hiszen $\rho = \frac{s+2}{s+1}$. Ebből pedig következik, hogy

$$x + y > \frac{(s+1)^2}{s^2+s-1}y.$$

A (15) egyenlőtlenség alapján teljesül

$$y < \frac{s^2+s-1}{(s+1)^2}((s+1)\Theta - \Delta).$$

Emiatt, alkalmazva a (14) egyenlőtlenséget azt kapjuk, hogy:

$$\Delta > \rho s\Theta - y > \frac{s^2+2s}{s+1}\Theta - \frac{s^2+s-1}{s+1}\Theta + \frac{s^2+s-1}{(s+1)^2}\Delta = \Theta + \frac{s^2+s-1}{(s+1)^2}\Delta.$$

A fenti egyenlőtlenség alapján pedig következik, hogy

$$\frac{s+2}{(s+1)^2}\Delta > \Theta,$$

vagyis $\Delta > \frac{(s+1)^2}{s+2}\Theta > s\Theta$.

2. eset. Az Y munka az X munka után következik a sorozatban. Ez azt jelenti, hogy az összes munka, ami az Y munka után következik, az M_1 géphez lesz rendelve. Állítjuk, hogy az Y munka az utolsó a sorozatban. Ellenkező esetben ugyanis egy rövidebb sorozatot kapunk, azáltal, hogy eltávolítjuk azokat a munkákat, amik az Y munka után következnének. Ez pedig ellentmond annak a ténynek, hogy az I sorozat minimális.

Jelölje az Y munka végrehajtási idejét Λ. Legyen a gépek terhelése (közvetlenül az Y munka ütemezése után) rendre $L_1 = x + \Delta$ és $L_2 = y + \Lambda$. A 4. lemmában található levezetéshez teljesen hasonlóan kapjuk:

$$\begin{aligned} x + \Delta + y + \Lambda &\leq (s+1)\Theta, \\ x + \Delta &> \rho\Theta, \\ y + \Lambda + \Delta &> s\rho\Theta, \end{aligned}$$

emiatt ezekből az egyenlőtlenségekből pedig egyértelműen következik az a tény, hogy

$$\Delta > \Theta(s+1)(\rho - 1) = \Theta.$$

2.1. eset. Tegyük fel, hogy $\Lambda > \Delta$. Ekkor azt kapjuk, hogy

$$OPT \geq \max\left\{\Delta, \frac{\Delta+\Lambda}{s}\right\}.$$

Ennek az oka, hogy ha egy optimális megoldás esetén ha az X, vagy az Y munkát az M_1 géphez rendeljük hozzá, akkor: $OPT \geq \Delta$, különben mindkét munkát az M_2 géphez rendeljük hozzá, ekkor pedig: $OPT \geq \frac{\Delta+\Lambda}{s}$ teljesül. Ezáltal

$$OPT \geq \max\left\{\Theta, \Delta, \frac{\Delta+\Lambda}{s}\right\}$$

teljesül. Ebből pedig következik, hogy

$$y + \Lambda + \Delta \leq (s+1)\Theta - \rho\Theta + \Delta \leq (s+2-\rho)OPT = s\rho OPT, \tag{16}$$

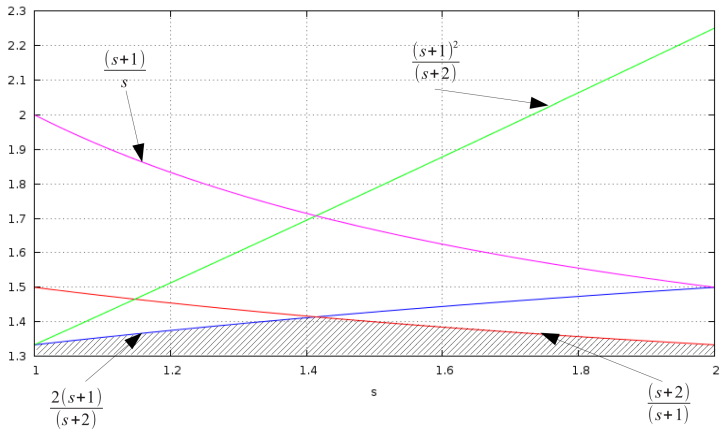
ahol az utolsó egyenlőtlenség $OPT \geq max\{\Theta, \Delta\}$ következtében teljesül, és az utolsó egyenlőség $\rho = \frac{s+2}{s+1}$ miatt teljesül. Ez azt jelenti, hogy az X munka átrendezése után a versenyképességi arányra vonatkozó felső korlát nem sérül.

2.2. eset. Tegyük fel, hogy $\Delta \geq \Lambda$. Ekkor az X munkának van a legnagyobb mérete az összes munka között. Két lehetséges esetet vizsgálunk meg a következőkben (ezek egyike szükségképpen teljesül, de csak az egyikük), aszerint, hogy optimális ütemezés esetén az X munka az M_1 gépre lesz ütemezve, vagy pedig az M_2 gépre kerül.

2.2.1. eset. Optimális ütemezés esetén az X munkát az M_1 géphez kell hozzárendelni. Ekkor $OPT \geq \Delta$ teljesül. A 2.1. eset vizsgálatához hasonlóan bizonyítható, hogy az X munka áthelyezés után az M_2 gép terhelése legfeljebb $s\rho OPT$ lesz.

2.2.2. eset. Ha optimális ütemezés során az Y munka az M_1 gépre kerül, akkor $OPT \geq max\{\Theta, \Lambda\}$. Ha mindkét munka (mármint X és Y is) az M_2 gépre kerül, akkor $OPT \geq max\{\Theta, \frac{\Lambda+\Delta}{s}\} \geq max\{\Theta, \Lambda\}$, mivel $s \leq 2$ és $\Delta \geq \Lambda$. Emiatt, $OPT \geq max\{\Theta, \Lambda\}$ mindenképpen teljesül.

Tudjuk, hogy az Y munka az utolsó a sorozatban és az M_2 géphez rendeltük hozzá. Tehát az algoritmusunk szabálya szerint ez azt jelenti, hogy az M_1 gép túlterhelt maradna, ha az Y munka az M_1 gépre kerülne, még akkor is ha ezután az X munkát az M_2 gépre



9. ábra. A felső korlát függvények összehasonlító ábrája.

helyezzük át.

Vagyis $x + \Lambda > \rho\Theta$, ekkor alkalmazzva a (16) egyenlőtlenséget, azt kapjuk, hogy:

$$y + \Delta < (s+1)\Theta - \rho\Theta = (s+1-\rho)\Theta.$$

Ebből következik, hogy:

$$y + \Delta + \Lambda < (s+1-\rho)\Theta + OPT \leq (s+2-\rho)OPT = s\rho OPT,$$

ahol az utolsó egyenlőtlenség $OPT \leq max\{\Theta, \Lambda\}$ következtében teljesül, és az utolsó egyenlőség $\rho = \frac{s+2}{s+1}$ miatt teljesül. Tekintettel arra, hogy az összes lehetséges esetet megvizsgáltuk, a bizonyítás teljes. ∎

A 4. lemma és az 5. lemma által a következő állítást bizonyítottuk be:

6. tétel: *A JO algoritmus versenyképességi aránya* $1 \leq s \leq \sqrt{2}$ *esetén* $\frac{2(s+1)}{s+2}$, *valamint a versenyképességi arány* $\sqrt{2} < s \leq 2$ esetén $\frac{s+2}{s+1}$.

Megjegyzés: A *JO* algoritmussal jelentősen javítottunk a lehetséges felső korláton ($K = 1$ és $1 \leq s \leq 2$ esetén), azonban az elért korlát még nem éles, legalábbis nem ismerjük az éles felső korlátot. Mivel az összes többi esetben (vagyis ha $K \geq 2$, vagy ha

$s \geq 2$) már ismert az éles arány, egyedül $K = 1$ és $1 \leq s \leq 2$ esetén maradt továbbra is nyitva valamely optimális algoritmus versenyképességi arányának meghatározása.

Hivatkozások

[1] Gy. Dósa, Zs. Tuza, Bin Packing/Covering with Delivery: some variations, theoretical results and efficient offline algorithms. *arXiv:* 1207.5672v1, 2011.

[2] Y. Azar, O. Regev, Online bin stretching, *Theor. Comp. Sci.*, **268** (2001), 17-41.

[3] T. Bäck, F. Hoffmeister, Extended Selection Mechanisms in Genetic Algorithms, *Richard K. Belew, Lashon B. Booker (Eds.): Proceedings of the 4th International Conference on Genetic Algorithms, San Diego, CA, USA, July 1991. Morgan Kaufmann 1991 ISBN 1-55860-208-9*, (1991), 92-99.

[4] J. Balogh, J. Békési, G. Galambos, G. Reinelt, Lower bound for the online bin packing problem with restricted repacking. *SIAM J. of Computing*, **38**(1) (2008), 398-410.

[5] A. Benkő, Gy. Dósa, Egy új feladat: Ládafedés szállítással, és ennek megoldása algoritmusok evolúciójával, *Alk. Mat. Lapok,* **27** (2010), 1-12.

[6] G. Beni, J. Wang, Swarm Intelligence in Cellular Robotic Systems, *Proceed. NATO Advanced Workshop on Robots and Biological Systems*, Tuscany, Italy, 1989.

[7] A. Benkő, Gy. Dósa, Zs. Tuza, A new tool "Evolution of Algorithms", for solving the hard combinatorial problem Bin Packing with Delivery, submitted, 2013.

[8] A. Benkő, Gy. Dósa, Zs. Tuza, Bin Covering with a general profit function: approximability results, *Central European Journal of Operations Research*, **21**(4) (2013), 805-816.

[9] A. Benkő, Gy. Dósa, Zs. Tuza, Bin Packing/Covering with Delivery, Solved with the Evolution of Algorithms, *Proceedings 2010 IEEE 5th International Conference on Bio-Inspired Computing: Theories and Applications*, BIC-TA 2010, art. no. 5645312 (2010), 298-302.

[10] I. Borgulya, Evolúciós algoritmusok, *Dialóg Campus Kiadó*, 2004.

[11] J. Boyar, G. Dósa, and L. Epstein, On the absolute approximation ratio for First Fit and related results, *Discrete Appl. Math.*, **160** (2012), 1914-1923.

[12] E. K. Burke, M. Hyde, G. Kendall et al., A classification of hyper-heuristic approaches, In M. Gendreau, J.-Y. Potvin (Eds), *Handbook of Metaheuristics, Springer*, 2010.

[13] M. Chen, Gy. Dósa, X. Han, C. Zhou, A. Benkő, 2D Knapsack: Packing Squares, *FAW-AAIM 2011 Conference,* LNCS **6681**, 176-184.

[14] X. Chen, Y. Lan, A. Benkő, G. Dósa, X. Han Optimal algorithms for online scheduling with bounded rearrangement at the end. *Theor. Comp. Sci.,* **412**(45) (2011), 6269-6278.

[15] J. Csirik, D. S. Johnson, C. Kenyon, Better approximation algorithms for bin covering, *Proceedings of the 12th ACM-SIAM Symposium on Discrete Algorithms* (SODA '01), (2001), 557-566.

[16] J. L. Denebourg, J. M. Pasteels et J. C. Verhaeghe, Probabilistic Behaviour in Ants: a Strategy of Errors?, *Journal of Theoretical Biology*, numéro 105, 1983.

[17] E. W. Dijkstra, A note on two problems in connexion with graphs, *Numerische Mathematik,* **1** (1959), 269-271.

[18] Gy. Dósa, Cs. Imreh, Online algoritmusok, Elektronikus jegyzet, Typotex kiadó, 2011.

[19] Gy. Dósa, A. Benkő, X. Han, Reassignment models on two related machines, *Proc. Conf. MAPSP 2011*, 256-258, 10th Workshop on Models and Algorithms for Planning and Scheduling Problems, Nymburk, Chech Republic, 2011, Institute of Theoretical Computer Science, Charles University.

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

[21] Gy. Dósa, Graham's example is the only tight one for $P||C_{max}$, *Annales Univ. Sci. Budapest*, **47** (2004), 207-210.

[22] Gy. Dósa, Optimális és közel-optimális online és félig online algoritmusok ütemezési feladatokra, *PhD értekezés*, Informatika Doktori Program, Operációkutatás és kombinatorikus optimalizálás alprogram, Szegedi Egyetem, 2009.

[23] Gy. Dósa, R. Li, X. Han, Zs. Tuza, Tight absolute bound for First Fit Decreasing bin-packing: FFD(L) <= 11/9 OPT(L) + 6/9, *Theor. Comp. Sci.*, online published, 2013.10.02.

[24] Gy. Dósa, L. Epstein, Online scheduling with a buffer on related machines. *J. Comb. Optim.,* **20**(2) (2010), 161-179.

[25] Gy. Dósa, J. Sgall, First Fit bin packing: A tight analysis, *Proceedings of STACS* (2013), 538-549.

[26] Gy. Dósa, J. Sgall, Optimal analysis of Best Fit bin packing, manuscript, 2013

[27] Gy. Dósa, Y. Wang, X. Han, H. Guo, Online scheduling with rearrangement on two related machines. *Theor. Comp. Sci.,* **412**(8-10) (2011), 642-653.

[28] M. Englert, D. Özmen, M. Westermann The power of reordering for online minimum makespan scheduling in: *Proc. 48th Symp. Foundations of Computer Science* (FOCS) (2008), 603-612.

[29] L. Epstein, J. Noga, S.S. Seiden, J. Sgall, G. J.Woeginger, Randomized Online Scheduling on Two Uniform Machines, *Journal of Scheduling*, **4**(2) (2001), 71–92.

[30] L. J. Fogel, A. J. Owens, M. J. Walsh, Artificial Intelligence through Simulated Evolution, *John Wiley*, 1966.

[31] L. Fortnow, The status of the P versus NP problem, *Comm. of the ACM* **52**(9) (2009), 78-86.

[32] M. R. Garey, R. L. Graham, D. S. Johnson, and A. C. Yao. Resource constrained scheduling as generalized bin packing. *J. Comb. Th. Ser. A*, **21** (1976), 257-298.

[33] M. R. Garey, D. S. Johnson, *A Guide to the Theory of NP-Completeness, Computers and Intractability*. New York, 1979.

[34] M. R. Garey, D. S. Johnson Strong NP-completeness results: Motivation, examples and implications *J. ACM* **25** (1978), 499-508.

[35] M. R. Garey, R. L. Graham and J. D. Ullman, Worst-case analysis of memory allocation algorithms. In *Proc. 4th Symp. Theory of Computing,* ACM, (1973), 143-150.

[36] R. L. Graham, Bounds for certain multiprocessing anomalies, *The Bell System Technical J.*, **45** (1966), 1563-1581.

[37] R. L. Graham, Bounds on multiprocessor timing anomalies, *SIAM J. Appl. Math.,* **17** (1969), 416-429.

[38] R. L. Graham, M. Grötschel, L. Lovász, Handbook of Combinatorics, *MIT Press*, 1995.

[39] R. L. Graham, E. L. Lawer, J. K. Lenstra, A. H. G. Rinnony Kan Optimization and approximation in deterministic sequencing and scheduling: A survey, *Ann. Discrete Math.* **5** (1979), 287-326.

[40] F. Glover, Tabu Search - Part 1, *ORSA J. on Computing* **1(2)** (1989), 190-206.

[41] F. Glover, Tabu Search - Part 2, *ORSA J. on Computing* **2(1)** (1990), 4-32.

[42] F. Glover and C. McMillan, The general employee scheduling problem: an integration of MS and AI, *Computers and Operations Research*, 1986.

[43] Y. He, X. Min, Online uniform machine scheduling with rejection, *Computing*, **65** (2000), 1-12.

[44] Y. He, G. C. Zhang, Semi online scheduling on two identical machines, *Computing*, **62** (1999), 179-187.

[45] D. Henderson, S. H. Jacobson, A. W. Johnson The Theory and Practice of Simulated Annealing. *Handbook of Metaheuristics* (2003), chapter 10, 287.

[46] J. H. Holland, Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence, *U Michigan Press*, 1975.

[47] B. Imreh, Cs. Imreh, Kombinatorikus optimalizálás, *Novadat*, 2005.

[48] Cs. Imreh, T. Németh, Parameter learning in online scheduling algorithms, 10th Workshop on Models and Algorithms for Planning and Scheduling Problems (MAPSP 2011), Nymburk, Chech Republic, 64-66.

[49] Cs. Imreh, T. Németh, Parameter learning algorithm for the online data acknowledgement problem, *Optimization Methods and Software*, **26**(3) (2011), 397-404..

[50] A. Ivani, L. Kaitar, Simultaneous solution of the travelling salesman problem and the packing problem. (Russian) *Vestnik Moskov. Univ. Ser. XV Vychisl. Mat. Kibernet.* 1989(1), 48-54. Translation in *Moscov Univ. Comput. Math. Cybernet.* 1989(1), 66-73.

[51] K. Jansen, R. Solis-Oba, An asymptotic fully polynomial time approximation scheme for bin covering, *Theoretical Computer Science,* **306(**1-3) (2003), 543-551.

[52] D. S. Johnson. Near-Optimal Bin Packing Algorithms. *PhD thesis*, Massachusetts Institute of Technology, Department of Mathematics, Cambridge, 1973.

[53] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM J. Comput.*, **3** (1974), 299-325.

[54] Karmarkar, N., Karp, R.: An efficient approximation scheme for the one-dimensional bin packing problem. In: Proceedings of the 23rd Annual Symposium on Foundations of Computer Science (FOCS'82), (1982), 312–320.

[55] H. Kellerer, V. Kotov, M. G. Speranza, Zs. Tuza, Semi on-line algorithms for the partition problem, *Op. Res. Lett.*, **21** (1997), 235-242.

[56] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi, Optimization by Simulated Annealing, *Science* **220**(4598) (1983), 671-680.

[57] D. S. Kovalenko, V. A. Kostenko, A generic algorithm for construction of recognisers of anomalies in behaviour of dynamical systems, *IEEE Fifth Int. Conf. on Bio-Inspired Computing: Theories and Applications* (BIC-TA), 2010, 258-262.

[58] Y. Lan, Gy. Dósa, X. Han, C. Zhou, A. Benko, 2D Knapsack: Packing Squares, *Theor. Comp. Sci.*, **508** (2013), 35-40.

[59] A. H. Land and A. G. Doig, An automatic method of solving discrete programming problems, *Econometrica* **28**(3) (1960), 497-520.

[60] P. J. M. van Laarhoven, E. H. L. Aarts, Simulated annealing: theory and applications. *Mathematics and its applications*, Kluwer Academic Publishers, 1987.

[61] C. C. Lee and D. T. Lee, A simple on-line packing algorithm, *J. ACM*, **32** (1985), 562-572.

[62] M. A. Lewis, G. A. Bekey, The Behavioral Self-Organization of Nanorobots Using Local Rules, *Proceedings of the 1992 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1992.

[63] M. Liu, Y. Xu, C. Chu, F. Zheng, Online scheduling on two uniform machines to minimize the makespan, *Theor. Comp. Sci.,* **410**(21-23) (2009), 2099-2109.

[64] K. Miettinen, M. M. Makela, P. Neittannmaki, J. Páriaux, (ed), Evolutionary algorithms in engineering and computer science. *John Wiley and Sons*, 1999.

[65] L. B. Miller, D. E. Goldberg, Genetic Algorithms, Selection Schemes and the Varying Effects of Noise, *Massachusetts Institute of Technology*, **4(2)** (1996), 113-131.

[66] T. Németh, Cs. Imreh, Parameter Learning Online Algorithm for Multiprocessor Scheduling with Rejection, *Acta Cybernetica* **19** (2009), 125–133.

[67] Z. Németh, A first fit algoritmus abszolút hibájáról (in Hungarian), *Eötvös Loránd Univ.*, Budapest, Hungary, 2011.

[68] I. Rechenberg, Evolutionsstrategie Optimierung technischer Systeme nach Prinzipien der biologischen Evolution, Problemata, *frommann-holzboog*, 1973.

[69] P. Sanders, N. Sivadasan, M. Skutella, Online scheduling with bounded migration, *Math. Oper. Res.,* **34**(2) (2009), 481-498.

[70] A. Schrijver, On the history of combinatorial optimization (till 1960), pp. 1-68, *Handbook of Discrete Optimization*, Elsevier, Amsterdam, 2005.

[71] S. Seiden, J. Sgall, G. Woeginger, Semi online scheduling with decreasing job sizes, *Operations Research Letters*, **27** (2000), 215-227.

[72] S. S. Seiden, On the online bin packing problem, *J. of the ACM*, **49** (2001), 2002.

[73] D. Simchi-Levi, New worst case results for the bin-packing problem, *Naval Res. Logist.*, **41** (1994), 579-585.

[74] Z. Tan, S. Yu, Online scheduling with reassignment, *Oper. Res. Lett.* **36**(2) (2008), 250-254.

[75] J. D. Ullman, The performance of a memory allocation algorithm, *Technical Report* 100, Princeton Univ., Princeton, NJ, 1971.

[76] B. Vizvári, Bevezetés a termelésirányítás matematikai elméletébe. *Egyetemi jegyzet*, ELTE (1992).

[77] Fernandez de la Vega, W., Lueker, G.S.: Bin packing can be solved within $1 + \varepsilon$ in linear time. Combinatorica **1** (1981), 349–355.

[78] Y. Wang, A. Benkő, X. Chen, Gy. Dósa, H. Guo, X. Han, C. Sik Lanyi, Online scheduling with one rearrangement at the end: Revisited. *Inform. Process. Lett.* **112** (2012), 641-645.

[79] S. R. Wilson, W. Cui, Applications of simulated annealing to peptides, *Biopolymers*, **29**(1) (1990), 225-235.

[80] B. Xia and Z. Tan, Tighter Bounds of the First Fit algorithm for the bin-packing problem, *Discrete Appl. Math.*, **158** (2010), 1668-1675.

[81] M. Yue, A simple proof of the inequality $FFD(L) \leq (11/9)OPT(L) + 1$, for all L, for the FFD bin-packing algorithm. *Acta Mathematicae Applicatae Sinica* **7**(4) (1991), 321-331.

[82] G. Zhang, A simple semi on-line algorithm for $p2//c_{max}$ with a buffer, *Inform. Process. Lett.*, **61** (1997), 145-148.

[83] J. Zhang, H. Chung, W. L. Lo, and T. Huang, Extended Ant Colony Optimization Algorithm for Power Electronic Circuit Design, *IEEE Transactions on Power Electronic.,* **24**(1) (2009), 147-162.

[84] W. Zhong, Gy. Dósa, Z. Tan, On the machine scheduling problem with job delivery coordination. *Eur. J. Op. Res.*, **182** (2007), 1057-1072.