SZAKDOLGOZAT

Téglalappakolási algoritmusok

Winkler László

Témavezető: Dr. Hujter Mihály
egyetemi docens

BME Matematika Intézet,
Differenciálegyenletek Tanszék

BME
2009

Szakdolgozat-kiírás

Témavezető: Hujter Mihály, egyetemi docens
Téma: Téglalapok sűrű pakolásai alkalmazásokkal

Téglalapok sűrű pakolásainak szerepe van egyrészt raktározási, szállítási feladatoknál, másrészt ütemezési kérdéseknél, de elméleti jelentőségű feladatoknál is. Hazánkban többek között Csirik János, Dósa György, Galambos Gábor, Imreh Csanád, Iványi Antal, Vizvári Béla értek el eredményeket ezen a területen. Számos izgalmas kérdés felderítetlen maradt. A diplomamunka készítőjének feladata lenne egyrészt áttekinteni a kutatási ág jelenlegi helyzetét, másrészt egy napjainkban is vizsgált részterületen egy-két speciális kérdést alaposan is meg kellene tárgyalnia. Megfelelő táblázatokat és ábrákat is várunk.

Irodalom:

http://www.jgytf.u-szeged.hu/tanszek/sztt/pub/sztt.html
http://publishing.eur.nl/ir/repub/asset/11700/Two-dimensional_rectangle_packing.pdf
http://compalg.inf.elte.hu/~tony/Informatikai-Konyvtar/
http://www.inf.u-szeged.hu/~cimreh/csimreh.ps
http://www.springerlink.com/content/2lc5cv4jv3br976w/

Tartalomjegyzék

1. Bevezetés

Téglalapok sűrű pakolásainak számos helyen van szerepe. Szállítási vagy raktározási problémák matematikai modelljeként adódik ilyen feladat, valamint munkagépek ütemezésénél is találkozhatunk vele. Az elméleti jelentőségű feladatok mellett pedig még a fentieken túl számos alkalmazási területen fordulnak elő.

A 2-dimenziós téglalappakolási probléma nem más, mint hogy egy adott szélességű alul zárt felül nyitott téglalap alakú sávban kell elhelyeznünk téglalapokat úgy, hogy a felhasznált magasság minimális legyen. Ebben a dolgozatban olyan pakolásokkal foglalkozunk, ahol forgatni nem lehet a téglalpokat, oldalaik a sáv oldalaival párhuzamosak, és nem fedhetik egymást. A sokféle alkalmazás és feladattípus miatt nem lehet mindent tárgyalni, ezért ebben a dolgozatban két feladattípust mutatunk be részletesen. A probléma illetve a feladat megfogalmazása után közöljük azokat az algoritmusokat, vagy heurisztikus módszereket, melyeknél a megoldások minnél közelebb lesznek az optimumhoz, mert az optimum meghatározása NP-teljes feladat. A második fejezetben egy offline téglalappakolási problémát nézünk meg, a harmadikban pedig egy speciális online feladatot vizsgálunk majd.

Offline problémának nevezzük azt, amikor adott a T_i $(i = 1, ..., n)$ téglalapok egy $\mathcal{T} = \{T_1, T_2, ..., T_n\}$ halmaza. A $\mathcal{T}$ halmaz elemeit pedig a fent leírt feltételek mellett kell minnél kisebb magasság felhasználásával elhelyezni az adott sávban. Míg az online problémánál adott a T_i téglalapok egy $\mathcal{L}$ listája, ahol kezdetben $i = 1$ és T_i-t úgy kell elhelyezni, hogy a T_j $(j = i+1, i+2, ..., n)$ téglalapokat még nem ismerjük. Ezután $i = 2$, azaz megkapjuk a második téglalapot, és az előző módon járunk el.

Példa. Egy tipikus online pakolási feladat a TETRISZ nevű számítógépes játék. Az alap változatban adott egy zárt m egység széles téglalap alakú sáv. A sáv tetejéről esnek le egyesével olyan elemek, melyek 4 darab 1×1-es összefüggő nényzetből állnak, egészen addig, amíg a sáv aljába vagy egy korábban elhelyezett elembe nem ütköznek. Mindig csak az aktuálisan elhelyezendő és a rákövetkező egységet ismerjük. Ezeket vágás nélkül kell úgy forgatni és elhelyezni, hogy minnél több telített sort rakjunk ki. Egy ilyen sor azonnal eltűnk, és a felette lévők lejjebb esnek. Ha az elemek elérik a sáv tetejét, vége a játéknak.

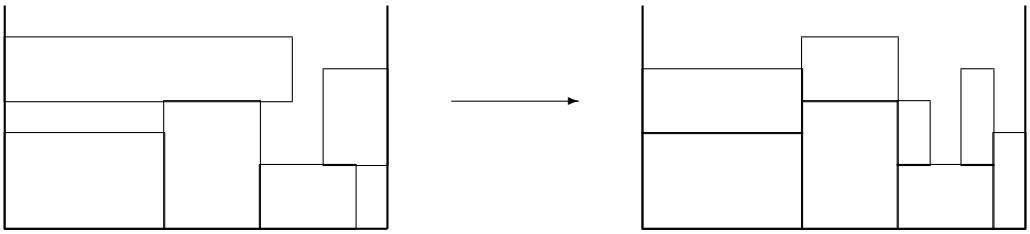
Egy érdekes feladatot fogunk vizsgálni a második részben, az úgynevezett relaxált 2-dimenziós téglalappakolási feladatot ($R2D$). Ennek megoldására heurisztikus mód-

szereket ismertetünk, hatékonyságbecsléseket adunk, valamint kísérleti eredményekkel igazoljuk, hogy az új algoritmusok sok esetben a korábbiaknál jobb megoldást adnak. (Itt megoldáson az optimum érték egyre jobban való közelítését értjük.) A harmadik fejezetben olyan online feladattal foglalkozunk, ahol a fent említett $\mathcal{L}$ lista elemeit 2 nem állandó paraméterrel adjuk majd meg. A feladatban egy téglalap magassága növelhető, szélessége csökkenthető, viszont területe állandó. Itt is ismertetünk néhány algoritmust, és megvizsgáljuk a hatékonyságukat. Az utolsó részben pedig összefoglaljuk a dolgozat témáit és fő mondanivalójukat.

2. Heurisztikus módszerek az $R2D$ feladatra

2.1. A relaxált feladat ismertetése

A kétdimenziós ládapakolási feladat egyfajta relaxációjával fogunk foglalkozni. Képzeljük el, hogy adott n darab $w_i \times h_i$ $(i = 1, ..., n)$ méretű téglalapot egy adott m egység szélességű sávon már elhelyeztünk úgy, hogy az oldalaik a sáv oldalaival párhuzamosak legyenek, és ne fedjék egymást; de most megengedjük, hogy a téglalapok azon részei, ahol nem érintkezik másik téglalap „tetejével" függőlegesen „leessenek". Ez azt jelenti, hogy eltoljuk ezeket a részeket a sáv aljának irányába mindaddig, amíg egy másik letett téglalap tetejébe, vagy a sáv aljába ütköznek. A téglalapok előbb említett részeinek ilyen módon történő elmozdítását a téglalapok lefelé igazításának hívjuk. Kérdés, hogy így mennyi a minimálisan felhasználandó magasság az elemek elhelyezéséhez. Ezt a feladatot jelöljük $R2D$-vel.



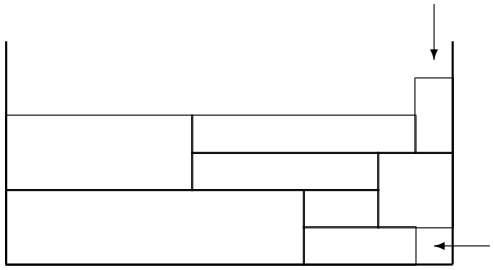
Lefelé igazítás az R2D feladatban

Az $R2D$ feladat felfogható úgy is, mint egy egyetlen erőforrással korlátozott ütemezési feladat. A téglalapok egy-egy munkát jelentenek, az i-edik munka w_i ideig tart, és az egyetlen erőforrásból h_i egységnyit vesz igénybe. Az erőforrásból minden pillanatban bármekkora mennyiség rendelkezésre áll, és a munkákat egy m egységnyi időhorizonton belül kell elvégezni. A kérdés az, hogy melyik munkát mikor kezdjük el, ha azt szeretnénk, hogy az egy-egy időpontban felhasznált erőforrásmennyiség maximuma minimális legyen az időintervallumra vonatkozólag. Ezután a feladatra téglalappakolási feladatként és ütemezési feladatként is fogunk hivatkozni.

Alkalmazás. Egy telepen szállításhoz kisebb tartályokat kell feltölteni valamilyen folyadékkal, egy nagyobb raktározó tartályból, amelyből tetszőlegesen sok egyforma csövön

keresztül áramoltathatjuk át az anyagot. A kis tartályok meghatározó adatai: 1) az egységnyi idő alatt történő maximális tölthetőségi sebesség (h_i), 2) a térfogatuk, amiből h_i ismeretében kiszámolható w_i, azaz hogy mennyi idő alatt tölthető fel az adott tartály. A munka kezdetétől a szállításig eltelt idő alatt (m) végezni kell az adott tartályokkal. A cél pedig az, hogy az egyes időpillantokban használt töltőcsövek számának maximuma minimális legyen.

Az alábbi illusztráció [8] azt a gyakran előforduló esetet szemlélteti, amikor a kétdimenziós (C_2), illetve a relaxált feladat optimális értéke különböző. A sávszélesség 12, és a következő 8 téglalapot kell lehelyezni: (8,2), (5,2), (3,1), (2,1), (5,1), (6,1), (2,2) és (1,2). A relaxált feladat esetén ezek elférnek 4 egységnyi magasságon belül, viszont a kétdimenziós esetben ez a magasság nem elég. Ezek után, ha ezt külön nem mondjuk, csak az $R2D$ feladattal foglalkozunk.



C_2 és $R2D$

2.2. Jelölések

Legyen $\mathcal{T} = \{t_i(w_i, h_i),\ i = 1, ..., n\}$ az adott n darab, $w_i \times h_i$ méretű téglalapból álló halmaz. Ezeknek megfelelően az i-edik munka elvégzésének időtartama w_i, az elvégzéshez szükséges (időben állandó mennyiségű) erőforrás nagysága h_i. Feltesszük, hogy $w_i \leq m$, ahol m a sáv szélessége, valamint $\sum_{j=1}^{n} w_i > m$, különben a munkákat egymás után is el lehetne végezni. A munkák egy ütemezésén a $\mathcal{T}$ halamaz valamely $\mathcal{P} = \{P_0, P_1, ..., P_{m-1}\}$ partícióját értjük, ahol P_i azon téglalpok halmaza, amelyeknek a bal széle a sáv bal szélétől i egység távolságra van, vagyis azon munkák halmaza, amelyekhez az i-edik időpontban

kezdtünk hozzá. A munkák sorrendje egy-egy rögzített időpontban az ütemezési feladat szempontjából közömbös. Legyen

$$S_i = \{t_\alpha \mid t_\alpha \in P_k,\ k \leq i < k + w_\alpha\},$$

vagyis S_i azon munkák halmaza, amelyek már elkezdődtek, vagy éppen elkezdődnek, de még nem fejeződtek be az i-edik időpontban. Legyen

$$H_i = \sum_{t_\alpha \in S_i} h_\alpha\ ,\ i = 0, 1, ..., m-1,$$

vagyis H_i az i-edik időpontban felhasznált erőforrás mennyisége.

1. Definíció. *A $\mathcal{P}$ ütemezés erőforrás-igénye az R2D feladat esetében*

$$C(P) = \max_{0 \leq i \leq m-1} H_i$$

2. Definíció. *A $\mathcal{P}^*$ ütemezés optimális, ha teljesül a $C(\mathcal{P}^*) \leq C(\mathcal{P})$ egyenlőtlenség a $\mathcal{T}$ halmaz tetszőleges $\mathcal{P}$ ütemezése esetén.*

Mivel véges sokféleképpen tudjuk a $\mathcal{T}$ halmaz elemeit m részre partícionálni, ilyen optimális ütemezés biztosan létezik, esetleg több is lehet.

A $C(\mathcal{P}^*)$ értéket ezentúl C_{R2D}-vel fogjuk jelölni, ami tehát csak $\mathcal{T}$-től és az m számtól függ. Ha ugyanezzel a $\mathcal{T}$ téglalaphalmazzal és m sávszélességgel a kétdimenziós feladatot oldjuk meg, akkor az elhelyezéshez minimálisan szükséges magasságot, vagyis az optimális megoldást jelölje C_2.

1. Tétel. [1] *C_2 és C_{R2D} jelöljék rendre a kétdimenziós és a relaxált kétdimenziós téglalap-pakolások optimális megoldásait. E két érték között a következő egyenlőtlenség áll fenn:*

$$C_{R2D} \leq C_2$$

2.3. Alsó becslések

Mivel az optimális ütemezés megkeresésének feladata már egydimenziós esetben is NP-teljes, gyakran heurisztikus módszerekkel próbálják az előbbi feladatot megoldani. Egy ilyen megoldás értékelésében nagy segítséget jelent, ha „elég jó” alsó becsléssel rendelkezünk az optimum értékét illetően, mert így tudjuk megbecsülni, hogy a heurisztikus

módszer által szolgáltatott megoldás, mennyire közelíti meg az optimumot, ugyanis az természetesen az alsó becslés és a heurisztikus megoldás értéke között van.

Legyen L egy alsó becslés, azaz legyen tetszőleges $\mathcal{T}$ téglalaphalmaz esetén $L(\mathcal{T}) \leq C_{R2D}$.

3. Definíció. *Az $\frac{L(\mathcal{T})}{C_{R2D}}$ törtek infimumát az L alsó becslés elméleti hatékonyságának nevezzük, és $H(L)$-lel jelöljük, vagyis*

$$H(L) = \inf\left\{\frac{L(\mathcal{T})}{C_{R2D}}\right\}$$

2. Tétel. [1] *Az R2D feladat esetében a felhasznált sávmagasság legalább akkora, mint a téglalapok magassága, valamint legalább akkora, mint a téglalapok m-ed része:*

$$C_{R2D} \geq L_1 := \max\left\{\frac{\sum_{i=1}^{n} w_i \cdot h_i}{m},\ \max\{h_i,\ i = 1, ..., n\}\right\}$$

Bizonyítás. Ha a téglalapokat teljesen egyenletesen sikerülne elosztani, akkor lenne az összterület m-edrésze a magasságuk; másrészt bármelyik téglalap magassága alsó korlát a felhasznált sávmagassághoz. □

3. Tétel. [1] *Az L_1 alsó becslés elméleti hatékonysága legalább $\frac{1}{3}$, vagyis $H(L_1) \geq \frac{1}{3}$.*

Bizonyítás. A [2]-ben szereplő $C_2 \leq \frac{2T}{m} + h_{max}$ becslésből, ahol T a téglalapok összterülete, h_{max} pedig a legnagyobb magasság, látható hogy $C_3 \leq 3L_1$, ebből $\frac{L_1}{C_{R2D}} \geq \frac{L_1}{C_2} \geq \frac{1}{3}$ adódik. □

4. Tétel. [1] *Rendezzük a téglalapokat magasságaik szerinti nem növekvő sorrendbe, vagyis $h_1 \geq h_2 \geq ... \geq h_n$. Legyen továbbá*

$$k = \left\lceil \frac{\sum_{t_i \in \mathcal{T}} w_i}{m} \right\rceil.$$

Ekkor C_{R2D} legalább akkora, mint a sorrendben az utolsó k számú téglalap magasságának az összege, vagyis

$$C_{R2D} \geq L_2 := h_{n-k+1} + ... + h_n.$$

Bizonyítás. A téglalapok elhelyezéséhez legalább k „sor” szügséges, azaz lesz olyan időpont, amikor legalább k számú munka egyszerre folyamatban van. □

Példa. Legyen $m = 6, n = 4$, és álljon a T halmaz két 3×3-as, egy 3×2-es és egy 4×3-as téglalapból. Ekkor $L_1 = 6$, míg $L_2 = 8$.

2.4. Néhány heurisztikus módszer

2.4.1. A BL algoritmus

Az optimális ütemezés megkeresésének NP-teljessége miatt gyors, közel optimális ütemezéseket adó algoritmusokat vezetünk be az $R2D$ feladat megoldására. Graham 1966-ban közölte az LPT (Longest Processing Time) algoritmust, amelyet egyforma párhuzamos gépek ütemezésére dolgozott ki [3]. Azóta is széles körben alkalmazzák, és sok más új algoritmus alapszik rajta. Az egyik ilyen a [4]-ben ismertetett LPT(k)' algoritmus, vagy az alábbi, a két dimenziós feladatra általánosított eljárás, melynek neve BL (Bottom Left) algoritmus [2]. Ez a következőket végzi. Először valamilyen sorrendbe rendezzük a téglalapokat. A soron következőt mindig úgy helyezzük el, hogy mindig a lehető leglejjebb legyen, és az így szóba jövő helyek közül a legelső időpontra ütemezzük a téglalapot, azaz balra igazítjuk. Ebből úgy kaphatunk $R2D$-re vonatkozó algoritmust, hogy az elemeket lefelé igazítjuk. Egy másik lehetőség, hogy rögtön az elhelyezésük után lefelé igazítjuk őket, és csak ezután helyezzük el a következőt.

BL algoritmus

1. Legyen m az időintervallum hossza, $P_i = \emptyset,\ H_i = 0,\ (i = 0, ..., m-1)$.
2. Rendezzük a téglalapokat valamilyen sorrendbe, és legyen $j = 1$.
3. Legyen $R(i) = \max_{i \le k \le i+w_j-1} H_k$, $(i = 0, \ldots, m - w_j)$.
   Legyen $i_0 = \arg\min R(i)\ :\ 0 \le i \le m - w_j$.
4. Legyen $P_{i_0} = P_{i_0} \cup \{t_j\}$, és legyen $H_k = H_k + h_j,\ (k = i_0, ..., i_0 + w_j - 1)$,
   vagyis ütemezzük a j-edik munkát az i_0-adik időpontra.
5. Legyen $j = j + 1$. $j \le n$ esetén menjünk a 3. lépésre, egyébként vége.

Az algoritmus elméleti hatékonyságával kapcsolatban [1] a következőket közli. Legyen $C(BL)$ a kétdimenziós algoritmus által kapott megoldás értéke. Ha a téglalapokat szélességeik szerinti növekvő, vagy magasságaik szerinti csökkenő sorrendben helyezzük el, akkor a $\frac{C(BL)}{C_{R2D}}$ arány tetszőlegesen nagy lehet. Ezek alapján érvényes a következő.

5. Tétel. [1] *Tetszőleges $M > 0$ valós számhoz létezik olyan $\mathcal{T}_1$ téglalaphalmaz, melynek elemeit szélességeik szerinti növekvő sorrendben elhelyezve $\frac{C(BL)}{C_{R2D}} > M$ teljesül. Hasonlóképpen létezik olyan $\mathcal{T}_2$ halmaz, melynek elemeit magasságaik szerinti csökkenő sorrendbe rendezve $\frac{C(BL)}{C_{R2D}} > M$ teljesül.*

Más a helyzet csökkenő szélesség szerinti sorrend esetén. Ha a téglalapokat valamenynyiük elhelyezése után igazítjuk lefelé, akkor a $\frac{C(BL)}{C_{R2D}}$ arány legrosszabb esetben 3 lehet, amely egy éles becslés.

2.4.2. Az $FFDH$ algoritmus

A kétdimenziós $FFDH$ (First Fit Decreasing Height) algoritmus a téglalapokat magasságaik szerint csökkenő sorrendben helyezi el egy-egy szintre egymás mellé balra igazítva, a következő téglalapot pedig mindig a legelső olyan szintre helyezi el, ahová befér. Ha már nem fér be a korábbi szintek egyikére sem, akkor közvetlenül a már létezők felett egy új szintet nyit, és ide rakja az aktuális téglalapot. Legyen i a szintek száma, r_α az α-adik szinten lévő téglalapok szélességeinek az összege, és tegyük fel, hogy az α-adik szint s_α magasságban van.

$FFDH$ algoritmus

1. Legyen $i = 1, r_1 = 0, s_1 = 0$.
2. Rendezzük a téglalapokat magasságaik szerint nem növekvő sorrendbe, és legyen $j = 1$.
3. $r_\alpha + w_j > m$ $(\alpha = 1, \ldots, i)$ esetén legyen $r_{i+1} = 0,\ s_{i+1} = s_i + h_j,\ j = j + 1$.
4. Legyen $\alpha_0 = \min\{\alpha \mid r_\alpha + w_j \leq m\}$. A j-edik téglalap az α_0-adik szintre kerül, $r_{\alpha_0} + w_j$.
5. Legyen $j = j + 1$. $j \leq n$ esetén menjünk a 3. lépésre, egyébként vége.

Az algoritmus megfelelő módosítással $R2D$ algoritmusnak is tekinthető. Így érvényes a [1]-ben közölt elméleti hatékonyság-becslés, mely szerint $\frac{C(FFDH)}{C_{R2D}} \leq 2,7$. Ha az egyes

szinteket nem mindig balról jobbra, hanem alternálva balról jobbra és jobbról balra töltjük fel, akkor az elméleti hatékonyság ugyan nem változik, de gyakran jobb megoldást kapunk.

2.4.3. A MIX algoritmus

A MIX nevű algoritmus a BL és $FFDH$ algoritmusok keveréke. Valamilyen sorrendbe rakja a téglalapokat. A soron következőt arra a legalsó szintre helyezni, ahová befér. Ennek a szintnek a magasságában elhelyez annyi téglalapot, amennyi csak lehetséges, balról jobbra feltöltve ezt a megkezdett szintet. Majd ezt a műveletet iterálja.

A $\frac{C(MIX)}{C_{R2D}}$ arány tetszőlegesen nagy lehet, ha a tégalalpok sorrendje magasságaik szerint csökkenő. Csökkenő szélesség esetén ez az arány legfeljebb 4, ez azonban nem éles becslés. Itt is sok esetben jobb megoldás kapható, ha a szinteket alternálva töltjük fel.

MIX algoritmus

1. Legyen m az időintervallum hossza, $P_i = \emptyset,\ H_i = 0,\ (i = 0, ..., m-1)$.
2. Rendezzük a téglalapokat valamilyen sorrendbe, és rakjuk őket egy $\mathcal{L}$ listába, $j := 1$.
3. $s := \min_{0 \le i \le m-w_j} \max_{i \le k \le i+w_j-1} H_k$, vagyis s a következő szint magassága.
4. $i_0 := \min\{i : \max_{i \le k \le i+w_j-1} H_k = s,\ i \in \{0, ..., m-w_j\}\}$.
5. $i_0 > -\infty$ esetén ütemezzük a j-edik munkát az i_0-adik időpontra,
   vagyis legyen $P_{i_0} = P_{i_0} \cup \{t_j\}$, és legyen $H_k = H_k + h_j,\ (k = i_0, ..., i_0 + w_j - 1)$.
   $j := j + 1,\ j > n$ esetén vége, egyébként menjünk újra a 4. pontra.
6. $i_0 = -\infty$ esetén erre a szintre már nem fért be téglalap.
   $j := j + 1,\ j \le n$ esetén menjünk a 3. pontra, egyébként vége.

2.5. Numerikus eredmények

A következő [1]-ből származó táblázat azt mutatja, hogy bevezetett alsó becslések mennyire közelítik meg a BL és $FFDH$ algoritmusok által adott heurisztikus megoldások értékeit. A tesztfeladatban a sávszélességet $m = 19$-nek választottuk, a téglalapok

szélességeit az [1,5], a magasságukat az [1,9] intervallumból választottuk egyenletes eloszlás szerint. A téglalapok száma 10, 20 és végül 50. A táblázat jobb oldalán a következők szerepelnek: hányszor volt az alsó becslés értéke egyenlő a jobbik heurisztikus megoldással ($= HEU$), és persze ekkor az optimummal is; hányszor volt az eredmény a jobbik heurisztikus megoldásnak legalább 0,9-szerese; illetve az alsó becslés/heurisztikus megoldás törtek átlaga százalékban kifejezve, 100 független kísérletet elvégezve.

$m = 19, w_i \in [1,5], h_i \in [1,9]$			
n	L_1	L_2	
10	53	88	$= HEU$
	77	99	$\geq 0,9HEU$
	93,80	98,78	$\frac{L_i}{HEU}$
20	7	12	$= HEU$
	76	79	$\geq 0,9HEU$
	92,47	93,05	$\frac{L_i}{HEU}$
50	13	13	$= HEU$
	100	100	$\geq 0,9HEU$
	97,06	97,06	$\frac{L_i}{HEU}$

Becslések pontossága

$n = 10$ esetén már az L_1 becslés is az esetek felében az optimális megoldás értékét adja, ez a tulajdonság a téglalapok számának emelkedésével viszont romlik. Érdekes módon elég nagy téglalapszám esetén ($n = 50$) az eseteknek csak töredékében szolgáltat L_1 pontos

becslést, viszont minden esetben legalább 0,9-szerese az optimumnak. Ez azt jelenti, hogy egy bizonyos téglalpszámon túl az L_1 becslésen az L_2 becslés már nem javít.

Az alábbi táblázat segítségével [1] a MIX és a korábbi algoritmusokat hasonlítjuk össze. A kísérletek száma 10000. A téglalapok szélességei és magasságai egyenletes eloszlás szerint kerültek ki az $[x_1, x_2]$ illetve az $[y_1, y_2]$ intervallumból. A BL esetén nem növekvő szélesség szerinti, a másik két algoritmus esetében nem növekvő magasság szerinti a téglalapok sorrendje. Az egyes sorokban lévő számok azt mutatják, hogy hányszor érte el a heurisztikus megoldás a jobb alsó becslést, ami L_2 (ez esetben a heurisztikus megoldás optimális); hányszor volt ennek legfeljebb az 1,05-szorosa; valamint a heurisztikus megoldás/alsó becslés arányok átlagát.

m	n	$[x_1, x_2]$	$[y_1, y_2]$	BL	$FFDH$	MIX	
29	27	[1,4]	[1,8]	427	5528	9711	$= L_2$
				427	5528	9711	$\leq 1,05 \cdot L_2$
				1,249	1,083	1,029	$\frac{C}{L_2}$
72	93	[1,8]	[1,9]	0	4950	9788	$= L_2$
				451	9583	9991	$\leq 1,05 \cdot L_2$
				1,113	1,031	1,011	$\frac{C}{L_2}$
40	50	[6,7]	[12,15]	8	3756	9587	$= L_2$
				2500	9976	9983	$\leq 1,05 \cdot L_2$
				1,152	1,018	1,073	$\frac{C}{L_2}$

Algoritmusok hatékonysága

Látható, hogy a MIX algoritmus általában jobb mint a BL és az $FFDH$. A $\frac{C}{L_2}$ arány minden esetben kisebb a MIX oszlopában. Több feladatosztály esetén nagy százalékban a legjobb heurisztikus megoldás biztosan optimális volt. Emellett más esetekben is elérhető a legjobb algoritmussal az optimum.

3. Alakítható téglalapok online pakolása

3.1. Bevezető

A fejlett offline algoritmusok a téglalapokat csökkenő sorrendbe rakják (amennyiben ez megengedett), és először a nagyobbak helyzeteit próbálják optimálisan meghatározni. Online feladatoknál nincs lehetőség a sorbarendezésre, mert az első téglalapot megkapva azt azonnal el kell helyezni, és csak azután kapjuk kézhez a következőt. A feladat a következő. Adott téglalapok egy listája: $\mathcal{L} = \{T_1, T_2, ...\}$, melyben minden elemet 2 paraméter határoz meg: $T_i = (w_i, h_i)$, $i = 1, 2, ...$, ahol w_i az i-edik téglalap maximális szélessége, h_i pedig a minimális magassága. Célunk, hogy $\mathcal{L}$ elemeit sorrendben bepakoljuk egy adott szélességű (m) alul zárt, felül nyitott téglalap alakú sávba úgy, hogy minimalizáljuk a totális magasságot, akárcsak korábban. Most azonban megengedett, hogy T_i szélességét csökkentsük, magasságát pedig növeljük úgy, hogy területe változatlan marad, azaz a $w_i \cdot h_i$ szorzat állandó minden i-re. Az esetleges átméretezés után helyezhetjük a téglalapokat a sávba. A továbbiakban feltesszük, hogy a sáv szélessége 1. Ezt minden további nélkül megtehetjük, mivel ha az eredeti feladat m hosszúságú sávra vonatkozik, akkor a következő transzformációt kell csak elvégezni: $(w_i, h_i) \rightarrow (\frac{w_i}{m}, \frac{h_i}{m})$.

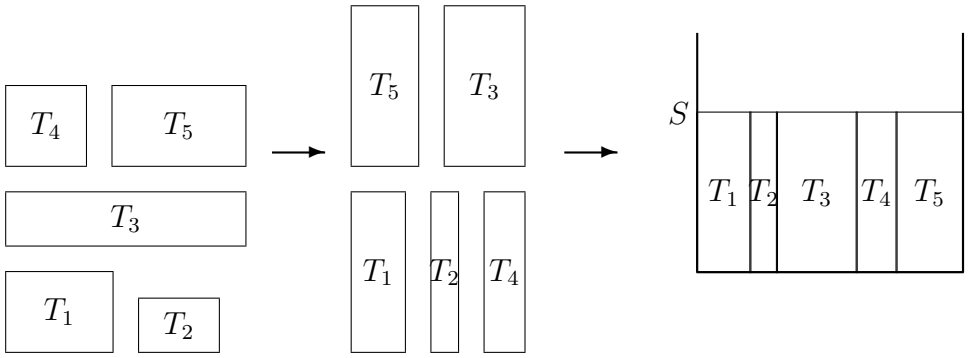
A feladat offline változata az, ahol ismert az egész $\mathcal{L}$ lista. A következő megállapítás mutatja, hogy ebben az esetben könnyű az optimális magasság meghatározása, amit jelöljön $OPT(\mathcal{L})$. Egy olyan algoritmussal, amely függőlegesen megnyújtja a téglalapokat $OPT(\mathcal{L})$ magasságig, és bepakolja őket sorban egymás mellé a sáv aljára, elérhető hogy a kihasznált magasság éppen $OPT(\mathcal{L})$ legyen.

6. Tétel. [5] *Az alakítható téglalapok sávpakolási feladának offline változatában az optimális megoldás, azaz a totális magasság*

$$OPT(\mathcal{L}) = max\{S, H\}$$

ahol $S = \sum_{i \in \mathcal{L}} w_i \cdot h_i$ *és* $H = max_{i \in \mathcal{L}} h_i$.

Bizonyítás. Először vegyük azt az esetet, amikor $S > H$. Ekkor mindegyik T_i nyújtható függőlegesen úgy, hogy a magasságuk S legyen. A megnyújtott téglalapokat helyezzük a sáv aljára egymás mellé. Így egy olyan pakolást kapunk, ahol a totális magasság S.



Másfelől a téglalapok területei változatlanok, így nem érhető el olyan pakolás, ahol a totális magasság kisebb mint S. Az másik eset hasonló az előzőhöz, azzal a különbséggel, hogy minden téglalapot H magasságúra nyújtunk, és ekkor még szabad hely is maradhat a sáv alján az $1 \times H$ területen. □

Az érdekes kérdés tehát az, hogy mit mondhatunk a probléma online változatáról. Az online esetben ugye egymás után kapjuk a téglalapokat egy listáról, és döntenünk kell, hogyan változtassuk meg az oldalaik méretét, és hová pakoljuk őket, anélkül hogy bármit is tudnánk a később jövő elemekről.

Alkalmazás. Egy akkumulátor-töltő állomásra egymás után érkeznek a feltöltésre szánt lemerült akkumulátorok. A töltéstároló képességük a téglalap területének felel meg, amely amperórában (Ah) van megadva. Nem lehet bármekkora erősségű árammal (A) tölteni őket. Az i-edik akkumulátor esetén ennek a felső korlátnak a w_i paraméter felel meg, h_i pedig a töltési időt (h) szimbolizálja. A $w_i \cdot h_i$ szorzat így valóban az akkumulátor energia tárolási kapacitása (Ah). Az áramerősség természetesen csökkenthető, de ekkor vele fordított arányosságban nő a töltési idő. A töltő berendezés rendelkezik egy maximális töltési képességgel (a sáv szélessége (m)). A feladat az, hogy eldöntsük, mikor kell egy beérkező akkumulátort belehelyezni a töltő berendezésbe ahhoz, hogy a teljes átfutási időt minimalizáljuk, és hogy tölthető-e a megengedett maximális áramerősséggel, azaz hogy a matematikai modellben hogyan szükséges átméretezni a neki megfelelő téglalapot.

4. Definíció. *Egy A online algoritmus* ***c-kompetitív*** *valamely $c > 0$-ra, ha minden véges $\mathcal{L}$ listára*

$$A(\mathcal{L}) \leq c \cdot OPT(\mathcal{L})$$

Két alapvető eljárás van egy online algoritmus hatékonyságának meghatározására.

5. Definíció. *Egy A algoritmus* ***abszolút versenyképességi hányadosa***

$$C_A = \sup_{\mathcal{L}} \left\{ \frac{A(\mathcal{L})}{OPT(\mathcal{L})} \right\},$$

ahol $A(\mathcal{L})$ jelöli az A algoritmus által megvalósított pakolás totális magasságát a téglalapok $\mathcal{L}$ listájára.

Általában, mint azt majd a későbbiekben láthatjuk, az *aszimptotikus versenyképességi hányados* nagyobb szabadságot nyújt egy algoritmusnak a hatékonyságát tekintve.

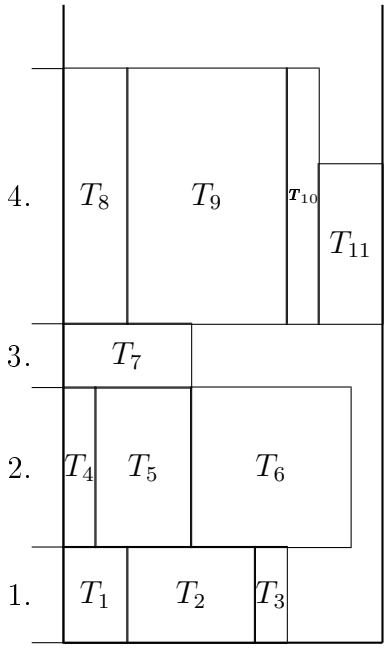
6. Definíció. *Egy A algoritmus* ***aszimptotikus versenyképességi hányadosa***

$$C_A^{\infty} = \lim_{k \to \infty} \sup_{\mathcal{L}} \left\{ \frac{A(\mathcal{L})}{OPT(\mathcal{L})} \mid OPT(\mathcal{L}) \geq k \right\}.$$

3.2. A polcpakolási algoritmusok családja

Az hogy az aszimptotikus vagy az abszolút versenyképességi hányadost használják a pakolási feladatoknál, alkalmazásfüggő. A probléma vizsgálatakor kiderül, hogy elérhető a felhasznált sávmagasságra vonatkozó magas optimális érték, ha tudjuk, hogy az $\mathcal{L}$ lista csak néhány nagyon magas téglalapból áll. Az aszimptotikus versenyképességi hányados használatával pontosabb képet kapunk az algoritmusról az olyan feladatoknál, ahol tudjuk, hogy csupán néhány téglalap érkezik majd, de csak akkor, ha magasságukra van egy felső korlátunk. Ha a C^{∞} mutatót szeretnénk használni a vizsgálat során, akkor általában fel szokás tenni (ha ez megengedett), hogy $h_i \leq 1$ minden i-re. Így jobban kezelhetőek a feladatok, mint a feltétel nélküli esetekben. Megmutatjuk, hogy ezen feltétel mellett minden $\varepsilon > 0$-ra létezik olyan algoritmus, amelynek aszimptotikus versenyképességi hányadosa $1 + \varepsilon$. Azonban mielőtt ismertetnénk ezt az algoritmust, definiáljuk az algoritmusok egy általános családját, a polc algoritmusokat. Ez a definíció egy átfogalmazása a [6]-belinek.

Egy alap eljárás az, hogy képzeletbeli polcokat hozunk létre a sávban, és ezekre vagy ezekbe pakoljuk a téglalapokat. Gondolhatunk erre úgy, mint egyforma, vagy akár különböző magasságú polcokból álló polcrendszerre egy szobában, vagy könyvtárban. Ezen polcokkal tulajdonképpen létrehozunk egy zárt, 1 szélességű téglalapot a sávban. Az algoritmus miután eldöntötte, melyik polcra fogja helyezni az éppen soron következő téglalapot, átméretezi azt oly módon, hogy a magassága a polc magasságával egyenlő legyen. Az esetlegesen átméretezett elemet a polcon balra igazítja, természetesen úgy, hogy a már korábban az adott polcra elhelyezett téglalapokat ne fedje. Fontos továbbá, hogy a polcról nem nyúlhat bele a téglalap sem az alatta sem a felette lévő polcba.



Polc pakolás

Egy polc pakolásra vonatkozó algoritmusnak a következő két dologban kell döntenie egy téglalap érkezésekor. Az első az, hogy létrehozzon-e új polcot a sávban vagy sem. Ha

szükség van egy újra, akkor meg kell határoznia annak magasságát. Az új polc mindig közvetlenül az előző fölé kerül. Az alsó oldala megegyezik az előző felső oldalával, azaz nincsenek hézagok a polcok között. Az első polc a sáv alján foglal helyet. Az algoritmusnak ki kell választania, hogy melyik polcra tegye az aktuális téglalapot. Ha az i-edik téglalap minimális magassága (h_i) nagyobb, mint a polc magassága, akkor természetesen semmilyen módon nem pakolható az adott polcra. Mindenképp kell új polcot létrehozni.

Nézzünk egy ilyen polc algoritmust, amely csak egyetlen x paramétertől függ.

xS algoritmus

1. (Inicializálás) Hozzunk létre egy x magasságú polcot, ahol $x > 1$. Legyen ez az úgynevezett aktív polc, azaz amelyikkel az algoritmus épp dolgozik.

2. (Pakolási fázis) Az érkező téglalapot, amennyiben lehetséges, pakoljuk az aktív polcra. Persze miután x magasságúra nyújtottuk. Egyébként pedig hozzunk létre egy új x magasságú polcot közvetlenül az előző felett. Legyen ez az aktív polc, és pakoljuk be ide a téglalapot. (Az eddig tárgyalt feltételek mellett ez természetesen megtehető.)

7. Tétel. [5] *Az xS algoritmus aszimptotikus versenyképességi hányadosára vonatkozó egyenlőtlenség:*

$$C_{xS}^{\infty} \leq 1 + \frac{1}{x-1} \tag{1}$$

Bizonyítás. Tegyük fel, hogy megkaptuk a téglalapok $\mathcal{L}$ listáját, és az algoritmus k darab polcot hozott létre. A sáv szélessége továbbra is 1. Vizsgáljuk meg az elhelyezett téglalapok összterületét. Valahányszor új polcot hoz létre az algoritmus, az azt jelenti, hogy olyan téglalap érkezett, ami területét tekintve túl nagy volt ahhoz, hogy beférjen az aktív polc még szabad területére. Másfelől nincs olyan téglalap $\mathcal{L}$-ben, amelynek a területe 1-nél nagyobb, azaz egy már átméretezett téglalap szélessége nem lehet nagyobb mint $\frac{1}{x}$. Ezért ha új polcot kell létrehozni, akkor az előző polcnak legalább az $\left(1 - \frac{1}{x}\right)$-ed része fel van töltve. Így a téglalapok területösszege az utolsó polcot kivéve mindegyiken egyenként legalább $x-1$. Ez azt jelenti, hogy a sávban elhelyezett téglalapok összterülete nem lehet kevesbb mint $(k-1)(x-1)$. Ezért $\frac{xS(\mathcal{L})}{OPT(\mathcal{L})} \leq \frac{kx}{(k-1)(x-1)}$, amely $k \to \infty$ esetén tart a tételbeli $1 + \frac{1}{x-1}$ értékhez. □

Ha (1)-ben x értékét $\left(1+\frac{1}{\varepsilon}\right)$-nak választjuk, akkor az xS algoritmus aszimptotikus versenyképességi hányadosa tetszőlegesen megközelítheti az 1 értket.

Következmény. Minden $\varepsilon > 0$ valós számhoz létezik olyan polc algoritmus, melynek aszimptotikus versenyképességi hányadosa legfeljebb $1 + \varepsilon$.

Ebből az eredményből adódik, hogy a vizsgálatoknál az aszimptotikus versenyképességi hányadost használva könnyen kapunk közel optimális megoldást szolgáltató algoritmusokat. De mi a helyzet az abszolút versenyképességi hányadossal? A problémák mélyebb vizsgálatához alkalmasabb mérce lehet az abszolút versenyképességi hányados, azaz hogy konkrétan egy adott problémánál mennyire jó az algoritmus.

3.3. Algoritmusok és versenyképességük

3.3.1. Az NFS_r algoritmus

Ebben a fejezetben versenyképességen az abszolút versenyképességet értjük, és az $\mathcal{L}$ lista téglalapjainak magasságára ezúttal semmilyen felső korlátot nem teszünk fel. Az első algoritmus, amit vizsgálni fogunk, egy igen elterjedt változata a [7]-ben ismertetett NFS_r (Next Fit Shelf r) algoritmusnak. Az eredeti versenyképességi hányadosa 7,46. Az imént említett elterjedt változatot jelöljük most mi is NFS_r-rel. Ez az algoritmus csak az $r > 1$ paramétertől függ, és hasonló módon xS-hez aktív polcokat használ a pakolás során. A fő különbség az, hogy xS-sel ellentétben itt több polc is aktív, és minden k-ra létezik egyetlen aktív r^{k+1} magasságú polc.

NFS_r algoritmus

Ha érkezik egy $t_i = (w_i, h_i)$ téglalap, annak megfelelően választunk k-nak egy értéket úgy, hogy az $r^k < h_i \leq r^{k+1}$ feltételt kielégítsük. Módosítsuk a téglalap méreteit a következőképpen: $(w_i, h_i) \to \left(\frac{w_i \cdot h_i}{r^{k+1}}, r^{k+1}\right)$. Ha van egy olyan aktív r^{k+1} magasságú polc, ahová még befér a módosított téglalap, akkor tegyük be oda, egyébként hozzunk létre egy új r^{k+1} magasságú polcot, és ide tegyük be a téglalapot. Ezután már ez lesz az egyetlen aktív r^{k+1} magasságú polc.

8. Tétel. [5] *Az NFS_r algoritmus versenyképességi hányadosa*

$$C_{NFS_r} = \frac{2 + r^2}{r - 1} \tag{2}$$

Az $r = 2$ választásnál, (2) alapján igaz az NFS_r algoritmus leggyakrabban használt változatára az alábbi.

Következmény. Az NFS_2 algoritmus versenyképességi hányadosa 6.

3.3.2. A *DS* algoritmus

Több a [6]-ban közölt NFS_r algoritmushoz hasonló eljárást módosítottak úgy, hogy az a rugalmas téglalapok online pakolási feladatát oldja meg. A következő eljárás is egy ilyen ütemezési algoritmuson alapszik. Jelenleg ez a legkisebb felső korláttal rendelkező algoritmus erre a problémára.

DS algoritmus

Az első téglalap érkeztével létrehozunk egy h_1 magasságú polcot. Ez lesz az aktív polc. Ezek után ha érkezik egy téglalap, akkor a következők szerint járunk el:

1. Ha a téglalap minimális magassága nem nagyobb, mint az aktív polc magassága, és az esetleges átméretezés után, azaz a polc magasságáig nyújtva bepakolható az aktív polcra, akkor tegyük oda. Egyébként menjünk a 2. lépésre.

2. Hozzunk létre egy új polcot, melynek magassága pontosan kétszerese az aktív polc magasságának. Ezek után ez lesz az aktív polc, és menjünk az 1. lépésre.

9. Tétel. [5] *A DS algoritmus versenyképességi hányadosa 4.*

Bizonyítás. Megmutatjuk, hogy az algoritmus 4-kompetitív. Legyen $\mathcal{L}$ egy korlátlan hosszú lista, azaz nem tudjuk, hogy éppen melyik elem lesz az utolsó a sorban érkezők

közül. Jelölje H az utoljára aktív polc magasságát. Ha az algoritmus ezt a polcot létrehozta, az azt jelenti, hogy egy téglalapot nem tudott bepakolni a korábbi aktív polcra, ezért bizonyos, hogy az előző aktív polc alacsonyabb mint $OPT(\mathcal{L})$. Ennélfogva $H \leq 2 \cdot OPT(\mathcal{L})$. Másfelől a korábbi polcok magasságai $\frac{H}{2}, \frac{H}{4}, \frac{H}{8}, \ldots, \frac{H}{2^i}$. A pakolás totális magassága tehát kisebb mint $2H$, és $2H \leq 4 \cdot OPT(\mathcal{L})$. A felső korlát éles a téglalapok alábbi $\mathcal{L}_i$ listájára. Az első i darab téglalap legyen $\left(\frac{1}{4}, 2^j\right),\ j = 1, \ldots, i$ és az utolsó pedig $(\frac{1}{4}, 2^i + 1)$. Egyszerű számolás után látható, hogy a $\frac{DS(L_i)}{OPT(L_i)}$ hányados tart 4-hez, amint $i \to \infty$, ami jelzi a felső korlát élességét. □

1. Megjegyzés. A DS algoritmusnak jobb a versenyképességi hányadosa mint NFS_r-nek. Viszont az NFS_r-nek van néhány előnye. Nem hoz létre jóval magasabb polcokat mint a téglalapok magasságának a maximuma. Ezért jobban használható az olyan modelleknél, ahol erősen korlátolt a sávmagasság. Hiszen a DS mindenképpen létrehoz egy 2^k magasságú polcot a 2^{k-1} magasságú felett, ha oda már nem fér be a soron következő téglalap, abban az esetben is, ha oda nem rakható be. Ekkor pedig egy újabb polcot kell létrehozni r^{k+1} magassággal. Az előző polc pedig már sosem lesz aktív, így ez esetben van egy 2^k magasságú felesleges polc, amely beleszámít a felhasznált sávmagasságba.

2. Megjegyzés. Gyakran kaphatunk jobb eredményt, ha a probléma nem teljesen online. Ez azt jelenti, hogy van valamilyen információnk a téglalapokról. Ha tekintjük azt az esetet, amikor tudjuk, hogy az $\mathcal{L}$ lista elemei magasság szerinti nem növekvő sorrendben vannak, akkor létezik egyszerű algoritmus, amely 2-kompetitív. Könnyen belátható, hogy a következő ($FFS1$) algoritmus ilyen.

$FFS1$ algoritmus

Legyen h_1 az első téglalap magassága. Először hozzunk létre egy h_1 magasságú polcot. Ezután ha érkezik egy elem, vizsgáljuk meg, vajon létezik-e olyan polc, ahová berakható. Ha van ilyen polc, akkor pakoljuk be ezek közül a legalsóba. Persze miután a polc magasságával megegyező nagyságúra nyújtottuk. Ha pedig nincs ilyen polc, akkor hozzunk létre egy újabb h_1 magasságú polcot közvetlenül a legfelső felett, és ide helyezzük a téglalapot, az átméretezés után.

Az alakítható téglalapok online feladatának megoldására van azonban alsó korlát is. Míg a nem online problémánál elméletileg lehetne olyan algoritmust konstruálni, amely

meghatározza az optimális megoldást, addig az online feladatra nem lehet elméletben sem olyan algoritmust alkotni, amely megtalálja a legjobb megoldást. Igaz az offline esetben egy ilyen algoritmus futásideje az input növekedésének megfelelően exponenciálisan növekszik.

10. Tétel. [5] *Nem létezik olyan algoritmus az alakítható téglalapok online pakolására, amelynek versenyképességi hányadosa kisebb mint K, ahol $K \approx 1,73$, az*

$$e^{-(\frac{K+1}{K-2})} = K - 1$$

egyenlet megoldása.

Ez azonban csak egy körülbelüli érték. Azt tudjuk, hogy valójában 1,73 körül van. Viszont az $1,73 \cdot OPT(\mathcal{L})$ és a DS algoritmus által garantált $4 \cdot OPT(\mathcal{L})$ között óriási a rés. Az alsó korlát vizsgálata ezért nem túl érdekes. Ha tudnánk olyan algoritmusról, amely jobban megközelítené, akkor érdemes lenne foglalkozni azzal, hogy tulajdonképpen mennyi is az alsó korlát.

4. Összefoglalás

A téglalappakolás a kezdetektől jelen volt a történelemben. A férőhellyel való gazdálkodás, a raktározás, az ütemezés, az idővel való spórolás, és az ilyen fajta optimalizálási feladatok hétköznapi jelenségek voltak. Az 1980-as években kezdtek behatóbban, nagy erőkkel foglalkozni a matematikusok a problémával. Persze azelőtt is születtek kimagasló eredmények ezen a területen. A vizsgálatok a bin packing (ládapakolási) feladatokkal kezdődtek. Napjainkra a matematikában a geometriai optimalizálás igen szép, érdekes és sokat kutatott területévé nőtte ki magát. Az ebben a dolgozatban említett hivatkozások és feladatok csupán töredékei a szakirodalomnak.

A dolgozatban olyan téglalappakolásokkal foglalkoztunk, ahol egy adott szélességű alul zárt felül nyitott téglalap alakú sávban kell elhelyeznünk téglalapokat úgy, hogy a felhasznált sávmagasság minimális legyen, ahol adott a T_i $(i = 1, ..., n)$ téglalapok egy $\{T_1, T_2, ..., T_n\}$ halmaza $\mathcal{T}$ vagy listája $\mathcal{L}$. Kikötöttük, hogy nem forgathatók a téglalapok, és egymást nem fedhetik.

A 2. részben a kétdimenziós ládapakolási probléma egyfajta relaxációjával ismerkedhettünk meg. Definiáltuk a relaxált feladatot. Ez olyan téglalappakolási feladat, ahol megengedjük, hogy a sávba pakolt elemeknek azon részei, ahol nem nem érintkeznek alattuk lévő téglalap tetejével, függőlegesen leessenek. A jelölések bevezetése után bevezettük az algoritmikus alsó becsléseket, majd egyre hatékonyabb algoritmusokat adtunk a problémára. Ezek az algoritmusok először valamilyen módon sorba rendezik $\mathcal{T}$ elemeit, és a soron következőt mindig a lehető legalacsonyabb szintre helyezik el balra igazítva. Végül kísérletek alapján készült táblázatok segítségével vizsgáltuk meg az alsó becslések pontosságát, illetve a különböző algoritmusok hatékonyságát. Láthattuk, hogy némely esetben az alsó becsléseket elérhetik az algoritmusok, azaz optimális megoldást adnak. Sok esetben pedig az optimum 0,9-szeresénél jobb megoldást eredményeznek. Az algoritmusok összehasonlítása során pedig a MIX-nek nevezett eljárás bizonyult a legjobbnak.

A 3. fejezetben megvizsgáltunk egy online téglalappakolási feladatot, ahol lehetőség van az elhelyezés előtt az elemek méretbeni módosítására. $\mathcal{L}$ elemeit két paraméter segítségével adtuk meg, ahol w_i a téglalap maximális szélessége, h_i pedig a minimális magassága. Egy téglalap magasságát növelhetjük, szélességét csökkenthetjük oly módon, hogy közben a területe változatlan marad. Az online algoritmusokat körülbelül 30 éve vizsgálják. Hatékonyságuk mérésének egyik fő módszere egyfajta „legrosszabb eset

elemzés", az úgynevezett kompetitív analízis. Egy online problémában az input részletenként érkezik. Az inputsorozaton adott egy rendezés, a sorozat tagjai eszerint a rendezés szerint egyesével jönnek, és az online algoritmusnak olyan döntések sorozatát kell hoznia velük kapcsolatban, amelyeknek hatása lesz a teljes hatékonyság végső minőségére. Mindezen döntéseket a már megérkezett inputrész alapján kell hozni anélkül, hogy bármilyen információja lenne a jövőre vonatkozóan. Az online feladatok három legelterjettebb modellje a téglalap vagy ládapakolási feladat, az online hipergráfok színezése és az úgynevezett k-szerver probléma. Bevezettük azt a fogalmat, hogy egy algoritmus *c-kompetitív*, amely azt jelenti, hogy az adott $\mathcal{L}$ listára az algoritmus által adott érték az optimumnak legfeljebb a c-szerese. Különböző algoritmusokat ismertettünk, melyeknek megvizsgáltuk a kompetitív tulajdonságát, azaz a versenyképességét. Mutattunk egy olyan eljárást az általános feladatra, amely 4-kompetitív. Jelenleg az általános problémára nincsen ismert algoritmus, amely ennél jobbat tudna garantálni. Közöltük továbbá, hogy nem lehet olyan algoritmust adni a dolgozatban tárgyalt online problémára, amelynek versenyképességi hányadosa jobb lenne, mint 1,73. Úgy mint magasabb dimenziós feladatoknál, itt is nagy rés van az eddig ismert legalacsonyabb felső korlát és az alsó határ között. Hatalmas előrelépés lenne, ha tudnánk olyan algoritmust adni, amely ezt a rést akárcsak egy kicsit is csökkenti.

Sok érdekes nyitott kérdés van még a témában. Vizsgálható más modellje is az alakítható téglalapok pakolásának. Egy a 3. részben tárgyalthoz hasonló az, amikor nem engedjük meg, hogy tetszőleges magasságúra nyújtsuk a téglalapot. Ez is egy igen érdekes feladatosztály. Általában két modellel szoktak foglalkozni ebben az esetben. Az elsőben adott egy k konstans, melyre $h_i \leq k$, $\forall T_i(w_i, h_i) \in \mathcal{L}$, és $h_i \leq k$ a téglalap módosítása után is. A másik lehetőség az, hogy egy adott k konstansra a h_i érték maximum a k-szorosára növelhető. Egy másik feladattípus az, amikor nem kötjük ki, hogy h_i meddig növelhető, de a nyújtásért bizonyos bűntetés jár. A büntetésre definiálására két alapvető lehetőség van. Az egyik az, hogy úgynevezett külső bűntetéseket rendelünk hozzá valamilyen módon a változtatásokhoz, és a teljes költség a büntetések és a felhasznált sávmagasság összege lesz. A másik lehetőség a belső bűntetés, amikor is az átméretezés hatására egy előre meghatározott módon nő a téglalap területe.

Jelenleg is folynak kutatások az irányban, hogy miként lehetne ezt a 4-es értéket lejjebb szorítani. A DS algoritmus lefutása után gyakran marad egy keskenyebb rész a sáv jobb szélén. A legújabb eredmények olyan algoritmusokról szólnak, amelyek ezt a

kihasználatlan területet hasznosítják valamilyen módon. Kérdés, hogy ez milyen hatással van az algoritmus futásidejére. Egy ilyen DS-hez hasonló eljárás az, amikor nem csak a legfelső, hanem a felső két vagy több polc is aktív, és megengedjük, hogy ha egy adott polcra nem férne be a soron következő téglalap, akkor átnyúlhasson az alatta lévő szintekre, amennyiben így már elférne. Viszont a legrosszabb esetben ezek az algoritmusok sem tudnak jobbat garantálni, mint a DS.

Ha beszélünk egy 2 dimenziós problémáról, akkor mindig az első kérdések között foglal helyet az, hogy mi a helyzet 3 dimenzióban. Szintén nagy irodalma van ennek a problémának is. A legtöbb 2 dimenziós modell kiterjeszthető eggyel magasabb dimenzióra. Ilyen feladat merül fel akkor, ha például kamionokat szeretnénk megpakolni szállításra váró dobozolt áruval, és az a kérdés, hogy hány járművel kell szállítani.

A tárgyalt problémák nagyon aktuálisak, mivel ebben a rohamosan fejlődő világban mindenütt szükség van optimalizálásra. Az energia és a nyersanyagok gazdaságos felhasználása, illetve az idő hatékony kihasználása mindennapos probléma. Egyre gyakoribbak az olyan feladatok, ahol nem teljes az információ. Nem ismerjük az összes optimalizálandó elemet, vagy még az elemek számáról sincs információnk. Ezek miatt az online problémák miatt fontosak a dolgozatban tárgyaltak, és az, hogy folynak a kutatások az egyre hatékonyabb, jobb algoritmusok megtalálásának irányába.

Hivatkozások

[1] Dósa György, Heurisztikus módszerek a relaxált kétdimentiós téglalappakolási feladatra, Alkalmazott Matematikai Lapok, 19 (1999), 133–153.

[2] B.S. Baker, E. G. Coffman, R. L. Rivest, Orthogonal packings in two dimenson, SIAM J. Comput. 4 (1980), 846–855.

[3] R. L. Graham, Bounds for certain multiprocessor anomalies, Bell System Tech. J., 45 (1966), 1563–1581.

[4] Dósa György, Vizvári Béla, Az LPT(k)' algoritmus egyforma párhuzamos gépek ütemezésére, Alkalmazott Matematikai Lapok, 21 (2004), 269–289.

[5] Imreh Csanád, Online stip packing with modifiable boxes, Operations Research Letters 29 (2001), 79–85.

[6] J. Csirik, G. Woeginger, Shelf algorithms for on-line strip packing, Inform. Process. Lett. 63 (1997), 171–175.

[7] B. S. Baker, J. S. Schwarz, Shelf algorithms for two dimensional packing problems, SIAM J. Comput. 12 (1983), 508–525.

[8] M. Hujter, „On the dynamic storage allocation problen", Manuscript, Computer and Automation Institute Hung. Acad. Sci. (1990).