Alkalmazott Matematikai Lapok **24** (2007), 117-130.

ÁTPAKOLÁST HASZNÁLÓ SZEMI-ON-LINE LÁDAPAKOLÁSI ALGORITMUSOK

BALOGH JÁNOS, GALAMBOS GÁBOR

A cikkben az egydimenziós szemi-on-line ládapakolási feladattal foglalkozunk. Ennél a feladatnál az elemek on-line elpakolásával egyidejűleg megengedünk bizonyos műveleteket a már elpakolt elemeken is. Minden pozitív egész k értékre definiálunk egy algoritmust, amely minden lépésben legfeljebb k elemet pakol át. Bebizonyítjuk, hogy az algoritmus-sorozat aszimptotikus versenyképessége $\frac{3}{2} + \frac{1}{6k-1}$.

1. Bevezetés

A ládapakolási feladat jól ismert kombinatorikus optimalizálási probléma. Ennek egydimenziós esetében adott méretük szerint a (0,1] intervallumba eső tárgyak (elemek) egy $L = \{x_1, x_2, \ldots, x_n\}$ listája, és rendelkezésre áll végtelen sok, egység kapacitású láda. A feladat minden x_i elemet hozzárendelni pontosan egy ládához úgy, hogy a ládabeli elemek méreteinek összege nem haladhatja meg a láda kapacitását. A cél minimalizálni a felhasznált ládák számát (azon ládákét, amelyekben legalább egy elem van a teljes lista elpakolása után). Jól ismert tény, hogy optimális pakolást találni NP-nehéz feladat [5]. Ezért indokolt polinomiális idejű, elfogadható közelítést adó algoritmusok vizsgálata. Ezek egy osztályát alkotják az on-line algoritmusok, amelyek végrehajtásakor az elemeket érkezésük sorrendjében pakoljuk el úgy, hogy semmit nem tudunk a lista további elemeiről. (Sem az elemek száma, sem a később érkező elemek mérete sem ismert.) Az egyszer elpakolt elemek az algoritmus során többet nem mozgathatók. Az off-line algoritmusok teljes információval rendelkeznek a listáról, amelyet a stratégiájuk kialakításánál figyelembe is vesznek. Az úgynevezett szemi-on-line algoritmusok az on-line és az off-line között helyezkednek el [1]. Ezen algoritmusoknál a következő műveletek legalább egyike megengedett:

- elemek átpakolása [3, 4, 8, 9],
- néhány következő elem megvizsgálása ("előrenézése", [6, 7]),
- vagy néhány elem (elő)rendezése [2].

A ládapakolási algoritmusok hatékonysága különböző módszerekkel mérhető. Jelen cikkünkben az aszimptotikus versenyképességi vizsgálatra szorítkozunk. Jelölje $A(L)$ egy A algoritmus által valamely L lista elemeinek elpakolásánál felhasznált ládák számát, és $OPT(L)$ egy optimális pakolás által felhasználtakét. Legyen

$$R_m(A) := \max\left\{\frac{A(L)}{m} | OPT(L) = m\right\}.$$

Ekkor

$$R_\infty(A) := \limsup_{m\to\infty} R_A(m)$$

– amit $R(A)$-val fogunk jelölni – mutatja az A algoritmus *aszimptotikus versenyképességét* (AVK). Az on-line ládapakolási algoritmusok AVK-jára ismert legjobb alsó korlát $1,5401$ [13], míg a legjobb ismert AVK-jú algoritmus, a Harmonic++ Seiden nevéhez fűződik [12]. Az általa elemzett algoritmusra bebizonyította, hogy: $R(Harmonic++) \leq 1,58889$.

Időrendi sorrendben az első szemi-on-line ládapakolási algoritmust Galambos adta meg [2] az on-line ládapakolási feladat azon megszorítására, amikor csak korlátos számú láda lehet egyszerre nyitva. (Egy láda lezárt, ha már nem pakolhatunk bele később.) Ez az algoritmus két bufferládát használ az elemek átmeneti tárolására. Ezt javítva Galambos és Woeginger [3] definiált egy, 3 bufferládát használó algoritmust, amelynek AVK-ja $1,6910$.

Átpakolást használó szemi-on-line ládapakolási algoritmusokat vizsgáltak Gambosi és szerzőtársai [4]-ben. Megadtak egy lineáris idejű és $1,5$-es AVK-jú, valamint egy $O(n\log n)$ időkomplexitású $4/3$-os AVK-jú algoritmust. Utóbbi eredmény javítása Ivkovič és Lloyd $5/4$-os AVK-jú algoritmusa [9]. Nagyon fontos azonban megjegyezni, hogy mindegyik említett szemi-on-line algoritmusban lépésenként nem csak konstans számú elem elpakolása történik. Valamennyi, [4]-beli és [9]-beli algoritmusnál „parányi" elemek „kötegeinek" átpakolása egységnyi költségű műveletnek van definiálva, azaz olyan elemek átpakolását engedik meg egy lépésben, amelyekre a tárgyak méreteinek összege egy adott korlát alatt marad.

Jelen dolgozatban is olyan szemi-on-line ládapakolási algoritmusokkal foglalkozunk, amelyek megengedik az átpakolást. Az általunk vizsgált algoritmusokban bármely elem minden egyes átpakolása (újrapakolása) egységnyi költségű, és a listaelemenként átpakolható elemek maximális száma egy előre megadott k értékkel korlátozható. Ezek a *k-átpakolásos szemi-on-line ládapakolási algoritmusok*.

A ládapakolás irodalma iránt érdeklődő olvasónak pl. Coffmann és szerzőtársainak 1999-es áttekintő tanulmánya ajánlható [1], amely további szemi-on-line ládapakolási algoritmusokat is tárgyal.

A következőkben k-átpakolásos szemi-on-line ládapakolási algoritmusok egy sorozatát adjuk meg. Minden $k \in N^+$-re definiálunk algoritmust, amelyet angol nevének rövidítésével – *Uniform Fit with k-repacking* – *UF-k* jelöljük, és amelyre $R(UF\text{-}k) = \frac{3}{2} + \frac{1}{6k-1}$ teljesül. Az algoritmus versenyképessége a $k = 1$ és $k = 2$ esetekre irreleváns, mivel a legjobb ismert on-line algoritmus jobb AVK-val rendelkezik. Az első érdekes eredmény a $k = 3$ paraméterhez tartozik, mivel erre

$R(UF\text{-}3) = 1,5588\ldots$ teljesül, amely jobb, mint a ma ismert legjobb on-line algoritmus [12] AVK-ja. Ennek másik értéke, hogy alatta van az úgynevezett *Harmonikus Fit típusú* on-line ládapakolási algoritmusok $1,583...$-as alsó korlátjának [11]. A $k = 5$ eset szintén javítás, mivel az *UF-5* algoritmus jobb AVK-val rendelkezik, mint a legjobb ma ismert on-line alsó korlát ($R(UF\text{-}5) = 1,53448...$). Ez az első eredmény, amely azt bizonyítja, hogy egy lépésenként konstans számú elem átpakolását megengedő, szemi-on-line algoritmus versenyképessége jobb lehet bármely on-line algoritmusénál. Ez lényegében azt jelenti, hogy a rendelkezésünkre álló információt jól ki lehet használni szemi-on-line algoritmusok versenyképességének javítására.

2. Az UF-k algoritmus és bonyolultsága

Ezt a fejezetet néhány jelöléssel és definícióval kezdjük. Ha x az input lista egy tetszőleges elemét jelöli, akkor szintén x-szel hivatkozunk ennek méretére ott, ahol a szövegkörnyezetben ez nem érthető félre, különben $s(x)$-szel. Egy megnyitott B *láda szintje* $szint(B)$-vel jelölt, és a tartalmazott elemek méreteinek összege, $0 < szint(B) \leq 1$. Egy elemet *kis elemnek* nevezünk, ha mérete a $\left(0, \frac{1}{2}\right]$ intervallumba esik, ha $x \in \left(\frac{1}{2}, 1\right]$, akkor *nagy elemnek*. *Nagy ládának* mondunk egy B ládát, ha tartalmaz egy nagy elemet.

Legyen k egy tetszőleges, rögzített pozitív egész szám. Osszuk fel a $(0, 1]$ intervallumot $3k + 2$ diszjunkt részintervallumra a következő módon:

$$\left(0, \frac{1}{6}\right] = \bigcup_{j=1,\ldots,k} \left(\frac{j-1}{6k}, \frac{j}{6k}\right] =: \bigcup_{j=1,\ldots,k} I_j,$$

$$\left(\frac{1}{6}, \frac{1}{3}\right] =: I_{2k},$$

$$\left(\frac{1}{3}, \frac{1}{2}\right] = \bigcup_{j=1,\ldots,k} \left(\frac{2k+j-1}{6k}, \frac{2k+j}{6k}\right] =: \bigcup_{j=1,\ldots,k} I_{2k+j},$$

$$\left(\frac{1}{2}, \frac{2}{3}\right] = \bigcup_{j=1,\ldots,k} \left(\frac{3k+j-1}{6k}, \frac{3k+j}{6k}\right] =: \bigcup_{j=1,\ldots,k} I_{3k+j},$$

$$\left(\frac{2}{3}, 1\right] =: I_{6k}.$$

Minden egyes intervallumhoz egy-egy ládaosztályt rendelünk hozzá, amelyet rendre $\mathcal{B}_j$-vel jelölünk. Egy B láda a $\mathcal{B}_j$ ládaosztályba kerül, ha a megnyitásakor benne elhelyezett első elem mérete az I_j intervallumba esik. Ha $B \in \mathcal{B}_j$, $3k + 1 \leq j \leq 4k$, akkor a B láda az algoritmus futása alatt átkerülhet egy másik ládaosztályba is. Az átsorolás akkor történik meg, ha olyan lépés hajtódik végre, amelynek során a lista éppen feldolgozás alatt lévő kis elemét, vagy ha korábban saját osztályába elpakolt kis elemet pakolunk át a $\mathcal{B}_j$ ládaosztályba tartozó nagy B ládába, és ezután már $szint(B) \notin I_j$.

Ennek az a következménye, hogy egy lista elpakolása során az egyidőben nyitott ládák száma maximum $3k+2$, de ez a szám dinamikusan változhat: ládákat nyithatunk meg, és ládák üresedhetnek ki. $A(L)$ számításakor csak azokat a ládákat tekintjük megnyitottnak, amelyekben a teljes lista elpakolása után is maradt elem.

Két ládaosztályt, $\mathcal{B}_j$ és $\mathcal{B}_l$ $(1 \le j \le 3k$ és $3k < l \le 6k)$, *komplemenseknek* nevezünk, ha az osztályokhoz rendelt I_j és I_l intervallumok tetszőleges elemeinek összege nem nagyobb, mint 1. Egy $\mathcal{B}_i$ ládaosztály komplemenseinek halmazát $C(\mathcal{B}_i)$-vel jelöljük $(1 \le i \le 6k)$. Vegyük észre, hogy $C(\mathcal{B}_{3k}) = \emptyset$ és $C(\mathcal{B}_{6k}) = \emptyset$. Továbbá, egy $\mathcal{B}_j$ $(1 \le j \le 3k-1)$, illetve egy $\mathcal{B}_l$ $(3k+1 \le l \le 4k)$ ládaosztály legkisebb indexű komplemense $\mathcal{B}_{3k}$, illetve $\mathcal{B}_1$, legnagyobb indexű komplemense $\mathcal{B}_{l^*}$, ahol $l^* = \min\{4k, 6k-j\}$, illetve $\mathcal{B}_{6k-l}$.

A következőkben vázlatosan ismertetjük algoritmusunkat: az UF-k alapvetően a Harmonikus Fit (HF) típusú algoritmusoknál alkalmazott szabály szerint [10] pakolja el minden osztály elemeit. Az egy osztályba eső elemeken belül ez Next Fit (NF) szabály szerinti pakolást jelent. Ez konkrétan azt jelenti, hogy ha az érkező elem mérete $x \in I_j$ és az utoljára megnyitott $\mathcal{B}_j$-beli B ládára $szint(B) + x \le 1$, akkor x-et elpakoljuk B-be. Különben nyitunk egy új $\mathcal{B}_j$ osztálybeli ládát, és x-et ebben a ládában helyezzük el. Fontos, hogy a fenti, HF eljárásnál alkalmazott szabálytól annyiban térünk el, hogy az új láda megnyitásakor nem zárunk be ládát az adott osztályban, tehát azok tartalma később is hozzáférhető lesz. Az algoritmusunk alapvetően ugyan egy Harmonikus Fit típusú szabályt alkalmaz, de osztópontjaink a $[0,1]$ intervallum egy ekvidisztáns (uniform) beosztásából származnak.

A HF szabály alkalmazásával minden olyan ládát, amely a $\mathcal{B}_i$, $i \le 3k$, ládaosztályokba esik – legfeljebb az osztályok utoljára megnyitott ládáitól eltekintve – egészen biztos, hogy legalább 2/3 szintig telepakolunk. Így, ha a lista nem tartalmaz más intervallumból elemeket, akkor a célként kitűzött 3/2-es aszimptotikus versenyképességet már el is érnénk. Mivel azonban a $\mathcal{B}_{3k+1}, \ldots, \mathcal{B}_{4k}$ osztályba tartozó ládákba a HF szabály egyetlen nagy elemet helyez, ezért ezekre a ládákra finomítanunk kell az algoritmust: ha a lista elemeinek méretei ezt lehetővé teszik, akkor gondoskodnunk kell arról, hogy ezek is fel legyenek töltve a megfelelő szintig. Két esetben térünk el a HF szabálytól.

2.1. Algoritmus. UF-k algoritmus

while van még elem az L listában **do** **input** a következő x;
  if $x \in I_l$, $3k \le l \le 6k$ **then**
    $x_{NF} \to \mathcal{B}_l$;
    if $x \notin I_{6k}$ **and** $x \notin I_{3k}$ **then**
      call Átpakolás(j, $6k-j$);
    endif ;
  endif ;
  else **if** $x \in I_j$, hogy $1 \le j \le 3k-1$ **then**
    call Feltöltés(x);
  endelse ;
endwhile ;

– **1. Szabály (Feltöltés)**: Ha egy olyan $x \in I_j$ kis elem érkezik, amelyre $1 \leq j \leq 3k-1$ teljesül, akkor megvizsgáljuk $\mathcal{B}_j$ komplemens osztályait indexeik szerint növekvő sorrendben oly módon, hogy az első nem üres $\mathcal{B}_l$ osztály esetében annak utoljára megnyitott B ládájába elhelyezzük x-et. Legyen ez a láda B. Az világos, hogy x elpakolása előtt $szint(B) \in I_l$. Ha az elpakolás után $szint(B) + x \notin I_l$, akkor B-t átsoroljuk abba a $\mathcal{B}_t$ ládaosztályba, amelyre $szint(B) + x \in I_t$.

  Ha a komplemensek mindegyike üres, akkor a kis elem elhelyezésére az NF szabályt használjuk.

2.1. Eljárás. Procedure Feltöltés(x)

$j := \lceil 6k \cdot x \rceil$; **comment:** x az aktuálisan elpakolandó elem mérete, j annak az osztálynak az indexe, melyre $x \in I_j$;
$l^* = \min\{4k, 6k - j\}$; ($\mathcal{B}_j$ legnagyobb indexű komplemense $\mathcal{B}_{l^*}$)
do $l = 3k+1$ **to** l^*
  if $\mathcal{B}_l \neq \emptyset$ **then**
    $x \to \mathcal{B}_l$;
    if $szint(B) \in I_p$, $p \neq l$ **then**
      $B \to \mathcal{B}_p$;
      if $p < 6k$ **then**
        call Átpakolás(p,$6k-p$);
      endif ;
    endif ;
    return ; (nincs osztályváltás vagy Átpakolásból tértünk vissza)
  endif ;
enddo ;
$x \to_{NF} \mathcal{B}_j$ osztály ládájába; (nem talált komplemenst)

– **2. Szabály (Átpakolás)**: Ha egy olyan nagy elem érkezik, amelyre $x \in I_l$, ahol $3k+1 \leq l \leq 4k$ teljesül, akkor az elem elpakolásához először nyitunk egy $\mathcal{B}_l$ osztálybeli nagy ládát, és abban elpakoljuk x-et. Ezután indexeik szerint csökkenő sorrendben megvizsgáljuk, hogy van-e olyan $\mathcal{B}_j$-beli láda, amely nem üres, ahol $\mathcal{B}_j \in C(\mathcal{B}_l)$. Ha találunk ilyen ládát, akkor ebből a ládából egy elemet áthelyezünk a nagy ládába. Amennyiben a nagy láda szintje az elpakolás után intervallumot vált, akkor a Feltöltési szabályban leírtakhoz hasonlóan járunk el. Figyelnünk kell arra, hogy ebben az esetben az a láda, amely a kis elemet tartalmazta, kiürülhet.

2.2. Eljárás. Procedure Átpakolás(l, j)

comment: Megpróbálunk átpakolni egy kis elemet egy $C(\mathcal{B}_l)$ osztálybeli ládából $\mathcal{B}_l$ utolsó ládájába ($l \geq 3k+1$). j adja meg azt, hogy a $C(\mathcal{B}_l)$ melyik elemétől kell kezdeni a keresést.

```
do t = j to 1
   if B_t ≠ ∅ then
     x ∈ B ∈ B_t → B_l
     if szint(B) ∈ I_s, l + 1 ≤ s ≤ 6k then
       B → B_s;
       if s < 6k then
         call Átpakolás(p, s);
       endif ;
     endif ;
   endif ;
enddo ;
```

Az algoritmus és a két fenti szabály pontos leírását a táblázatokban található pszeudokódok tartalmazzák. A leírás során az $x_{NF} \to \mathcal{B}_t$, illetve $x \to \mathcal{B}_t$ jelölést használjuk akkor, ha az aktuális tárgyat a Next Fit szabállyal pakoljuk el a $\mathcal{B}_t$ ládaosztályba, illetve ha berakjuk a $\mathcal{B}_t$ ládaosztály utolsó ládájába. Hasonlóan, a $B \to \mathcal{B}_t$ jelölést használjuk annak rövidítésére, hogy a B ládát átsoroljuk a $\mathcal{B}_t$ ládaosztályba.

A következő három állítás az algoritmus bonyolultságának vizsgálatához nyújt segítséget.

2.1. Lemma. *Az UF-k algoritmus véges.*

Bizonyítás. A főprogram pontosan n-szer fut le és a Feltöltés eljárás listaelemenként legfeljebb egyszer hajtódik végre.

A főprogramból vagy a Feltöltés eljárásból hívott Átpakolás eljárás ugyan rekurzívan meghívhatja önmagát, de a rekurzív hívás csak akkor következik be, ha az aktuálisan használt láda szintje intervallumot vált. Ennek következménye, hogy egy Átpakolás eljárásban a ládamagasság intervallumát jelző (első) paramétere, l, a rekurzióban egymást követő Átpakolás eljárások során szigorúan monoton nő. A rekurziót indító Átpakolás eljárásban $l \geq 3k + 1$, az utolsónak hívottban legfeljebb $4k$. Ezért a rekurzív hívások száma listalemenként legfeljebb $k - 1$, ami adja, hogy az Átpakolás eljárás legfeljebb k-szor hívódik meg listalemenként.

Mivel az Átpakolás eljárás listaelemenként csak a főprogram és a Feltöltés eljárások legfeljebb egyikéből hívódik meg, így állításunk bizonyított. □

2.1. Következmény. *Ha k egy rögzített, pozitív egész szám, akkor az UF-k algoritmus legfeljebb k elemet pakol át lépésenként.*

2.2. Lemma. *Az UF-k algoritmus futása során ládaosztályok tartalmának vizsgálata ($B_i \neq \emptyset$ összehasonlítás) összesen legfeljebb $3kn$-szer történik meg.*

Bizonyítás. Azt fogjuk belátni, hogy listalemenként legfeljebb $3k$-szor történik meg az állításban említett összehasonlítás.

A főprogramban nincs, a Feltöltés eljárásban egy listaelem elhelyezésekor legfeljebb k-szor történhet ilyen vizsgálat.

A főprogramból vagy a Feltöltés eljárásból hívott Átpakolás eljárás első paramétere, l, második paramétere $6k-l$. $l = 3k+i$ alakú, ahol $1 \leq i \leq k$. A $6k-l$-nél nem nagyobb indexű ládaosztályok száma $2k+1-i$. Az Átpakolás eljárások egymást követő rekurzív hívásai során a második paraméterek alkotta sorozat – nem szigorú értelemben ugyan, de – monoton csökkenő sorozatot alkot. Mivel egy adott listelem elhelyezésekor az Átpakolás eljárás legfeljebb $k+1-i$-szer hívódik (beleértve a rekurzív hívásokat is), így – multiplicitásokkal számolva – ezekben összesen legfeljebb $(2k+1-i)+(k+1-i) = 3k+2-2i$ kérdéses vizsgálat történik.

Ha az adott listaelemnél az első Átpakolás eljáráshívás Feltöltés eljárásból történt, akkor $i \geq 2$ (hiszen ott a nagy láda ekkor osztályt is kellett, hogy váltson), és a Feltöltés eljárásban legfeljebb $i-1$ darab ilyen vizsgálat volt, összesen tehát legfeljebb $3k-1$.

Ha az Átpakolás a főprogramból hívódott, akkor a főprogramban egyetlen kérdéses összehasonlítás sem történt, ebben az esetben $i \geq 1$, így a kérdéses vizsgálatok száma összesen legfeljebb $3k$. □

A három állítás következményeként kimondható az alábbi tétel:

2.1. Tétel. *Az UF-k algoritmus időbonyolultsága mind az input elemek mind az átpakolható elemek számában lineáris, pontosan* $T(UF\text{-}k) = O(kn)$.

3. Az UF-k algoritmus aszimptotikus versenyképessége

Az UF-k algoritmus AVK-jának vizsgálatakor két esetet különböztetünk meg attól függően, hogy az algoritmusnak egy konkrét L listainputon való lefuttatása után

- **A.**: a $\mathcal{B}_{3k+1}, \ldots, \mathcal{B}_{4k}$ ládaosztályok mindegyike üres,
- **B.**: a $\mathcal{B}_{3k+1}, \ldots, \mathcal{B}_{4k}$ ládaosztályok valamelyike nem üres.

Az **A.** esetre a következő állítás mondható ki:

3.1. Lemma. *Ha egy L listainputon az UF-k algoritmus leállásakor a $\mathcal{B}_{3k+i}$, $i \in \{1, \ldots k\}$ ládaosztályok mindegyike üres, akkor* $UF\text{-}k(L) \leq \frac{3}{2}OPT(L)+(2k+1)$.

Bizonyítás. Ebben az esetben minden láda legalább $\frac{2}{3}$ szintig tele van, a csak *kis* elemeket tartalmazó ládaosztályok $(\mathcal{B}_1, \ldots, \mathcal{B}_{3k})$ legfeljebb $1-1$ ládájától eltekintve. Az ilyen ládaosztályok száma pedig maximum $2k+1$. Ebből a tétel állítása azonnal következik. □

A **B.** eset vizsgálatához az irodalomból jól ismert eszközt, az úgynevezett súlyfüggvény-technikát alkalmazzuk (lásd pl. [3]). A súlyfüggvény definiálásákor az algoritmus AVK-jának elemzésében a következő állítás fontos szerepet fog játszani.

3.2. Lemma. *Ha az UF-k algoritmus egy L inputon való lefutása után a $\mathcal{B}_{3k+i}$, $i \in \{1, \dots k\}$ legalább egyike nem üres, és i^* a minimális ilyen tulajdonságú index, akkor a $\mathcal{B}_{3k+i^*}$ összes komplemens osztálya, kivéve $\mathcal{B}_1$-et, üres.*

Bizonyítás. A bizonyítás indirekt módon történhet. Tegyük föl, hogy algoritmusunk elpakolta az adott listát, $\mathcal{B}_{3k+i^*}$-nak valamely komplemens osztálya nem üres, és az nem a $\mathcal{B}_1$ osztály. Legyen x a legnagyobb indexű ilyen komplemens osztály utolsó ládájának legfelső eleme. Mivel $\mathcal{B}_{3k+i^*}$ nem üres, így tartalmaz legalább egy nemüres ládát. Legyen ez B.

Két eset lehetséges. Vagy az x érkezett hamarabb, mint ahogy B-be utoljára elemet pakoltunk, vagy fordítva. Legyen y a B legfelső eleme, azaz az utoljára B-be helyezett elem. (Azaz az x kis elem vagy a B ezen formájának kialakulása, B tartalmának utolsó változásának időpillanata előtt érkezett, vagy fordítva).

- Az első esetben az algoritmus rápakolta volna a kis x elemet egy nagy ládára, hiszen akkor volt legalább egy, a feltételeknek megfelelő láda.
- A második esetben a szóban forgó nagy, $\mathcal{B}_{3k+i^*}$-beli ládába bepakolásra került volna még elem y bepakolása után, hiszen volt akkor legalább egy átpakolható „kis elem”.

Mindkét eset ellentmondáshoz vezet. □

Legyen i^* a 3.2. Lemmabeli, és legyen $j^* = \max\{i^*, 2\}$. A **B.** esetben használt súlyfüggvény pontos definíciója a következő:

$$w(x) := \begin{cases} \frac{6k}{6k-1}x, & x \in \{I_1 \cup \dots \cup I_{j^*-1}\} \\ \frac{j^*-1}{6k-1}, & x \in \{I_{j^*} \cup \dots \cup I_{3k-i^*}\}, \\ \frac{1}{2}, & x \in \{I_{3k-i^*+1} \cup \dots \cup I_{3k}\}, \\ 1 - \frac{6k}{6k-1}(\frac{3k+i^*-1}{6k} - x), & x \in \{I_{3k+1} \cup \dots \cup I_{3k+i^*-1}\} (= \emptyset, \text{ha} i^* = 1), \\ 1, & x \in \{I_{3k+i^*} \cup \dots \cup I_{6k}\}. \end{cases}$$

Mielőtt megadjuk az UF-k algoritmus aszimptotikus versenyképességét, kimondunk néhány további lemmát.

3.3. Lemma. *Ha az algoritmus lefutása után létezik legalább egy olyan i index, $i \in \{1, \dots, k\}$, hogy a $\mathcal{B}_{3k+i}$ osztály tartalmaz legalább egy nemüres ládát, és $3k + i^*$ a legkisebb ilyen index, akkor létezik egy olyan $0 \le M_2 < \infty$ konstans, hogy bármely L listára $w(L) = \sum_{x \in L} w(x) \ge UF\text{-}k(L) - M_2$.*

Bizonyítás. Azt fogjuk belátni, hogy – ládaosztályonként legfeljebb egy-egy ládától eltekintve – minden ládában az elemek összsúlya legalább 1. (Egy láda súlyán a benne elhelyezett elemek súlyainak összegét értjük.)

- A $\mathcal{B}_1$ osztály ládái, legfeljebb 1 kivételtől eltekintve legalább $\frac{6k-1}{6k}$ szintig tele vannak. Itt minden elem súlya (saját) méretének $\frac{6k}{6k-1}$-szerese, így a legfeljebb egy ládától eltekintve minden láda súlya legalább 1.
- A $\mathcal{B}_2, \ldots, \mathcal{B}_{2k}$ osztályok mind üresek.
- A $\mathcal{B}_{2k+1}, \ldots, \mathcal{B}_{3k}$ osztályok közül amelyek nem üresek, azok ládái – osztályonként legfeljebb $1-1$ kivételtől eltekintve – pontosan két elemet tartalmaznak, mindkettő súlya $\frac{1}{2}$.
- A $\mathcal{B}_{3k+1}, \ldots, \mathcal{B}_{3k+i^*-1}$ osztályok üresek.
- A $\mathcal{B}_{3k+i^*}, \ldots, \mathcal{B}_{4k}$ és a $\mathcal{B}_{6k}$ osztályok ládái – kivétel nélkül – pontosan egy „nagy elemet" tartalmaznak. Legyen B egy ilyen láda, és legyen x a benne lévő nagy elem. Két esetet különböztetünk meg. Ha $x > \frac{3k+i^*-1}{6k}$, akkor ennek súlya önmagában 1, így $w(B) \geq 1$. Ha $x \in \left(\frac{1}{2}, \frac{3k+i^*-1}{6k}\right]$, akkor mivel $szint(B) > \frac{3k+i^*-1}{6k}$, így a láda tartalmaz legalább $\frac{3k+i^*-1}{6k} - x$ összmennyiségű kis elemet, amelyek összsúlya legalább $\frac{6k}{6k-1}\left(\frac{3k+i^*-1}{6k} - x\right)$, így $w(B) \geq 1$ ekkor is. Azaz, mindkét esetben, minden láda összsúlya legalább 1.

Figyelembe véve, hogy a kivétel ládák száma az összes – nem üres – ládaosztályban legfeljebb $k+1$, így az $M_2 = k+1$ konstanssal a lemma állítása teljesül. □

3.4. Lemma. *Ha az algoritmus lefutása után létezik legalább egy olyan i index, $i \in \{1, \ldots, k\}$, hogy a $\mathcal{B}_{3k+i}$ osztály tartalmaz legalább egy, nemüres ládát, és S az L elemeinek egy olyan részhalmaza, melyre $\sum_{x \in S} x \leq 1$, akkor*

$$w(S) := \sum_{x \in S} w(x) \leq \frac{3}{2} + \frac{1}{6k-1}$$

teljesül.

Bizonyítás. A kulcs ennek a résznek a bizonyításához (is) a 3.2. Lemma. Ha az S halmazban nincs elem az $\left(\frac{1}{2}, \frac{2}{3}\right]$ intervallumból, akkor a súlyfüggvény definíciójából látható, hogy minden S-beli x elem súlya $\leq \frac{3}{2}x$. Ebből következően

$$w(S) := \sum_{x \in S} w(x) \leq \sum_{x \in S} \frac{3}{2}x = \frac{3}{2} \sum_{x \in S} x \leq \frac{3}{2} \cdot 1 = \frac{3}{2}.$$

Ha az S halmaz tartalmaz elemet az $\left(\frac{1}{2}, \frac{2}{3}\right]$ intervallumból, akkor pontosan egy elemet tartalmaz onnan, és ez a legnagyobb méretű elem a ládában. Jelöljük ezt x_1-gyel. Feltételünk szerint ekkor létezik olyan i pozitív egész, hogy $x_1 \in I_{3k+i}$. Két esetet különböztetünk meg, x_1 mérete alapján.

1. Az első eset, amikor $x_1 > \frac{3k+i^*-1}{6k}$, azaz $i \geq i^*$. Az S-beli második legnagyobb elem, x_2 mérete alapján két lehetőségünk van.

a) Az első lehetőség, hogy x_2 nem eleme az I_j, $j \in \{3k-i+1, \dots, 3k\}$ intervallumok egyikének sem, azaz a maradék az x_1-től különböző S-beli elemek az I_j, $j \in \{1, \dots, 3k-i\}$ intervallumba esnek. Az ilyen kis x elemek súlya legfeljebb $\frac{6k}{6k-1}x$, így

$$w(S) = w(x_1) + \sum_{x \in S, x \neq x_1} w(x) \leq 1 + \frac{6k}{6k-1} \cdot \frac{1}{2} = 1 + \frac{3k}{6k-1} < \frac{3}{2} + \frac{1}{6k-1}.$$

b) A második lehetőség, hogy x_2 az I_j, $j \in \{3k-i+1, 3k\}$ intervallumok valamelyikébe esik. Ez az intervallum azonban kizárólag az I_{3k-i+1} intervallum lehet, mert bármely más intervallumbeli elemhez hozzáadva x_1-et 1-nél nagyobb értéket kapnánk. Viszont ekkor S x_1-en és x_2-n kívüli elemeinek összmérete legfeljebb $1-x_1-x_2 \leq 1 - \frac{3k+i-1}{6k} - \frac{3k-i}{6k} = \frac{1}{6k}$ lehet. (Ez azt is jelenti, hogy ekkor az S halmaz x_1-en és x_2-n kívül legfeljebb csak az I_1 intervallum elemeit tartalmazhatja.) Így

$$w(S) = \sum_{x \in S} w(x) = w(x_1) + w(x_2) + \sum_{x \in S, x \neq x_1, x \neq x_2} w(x) \leq$$
$$\leq 1 + \frac{1}{2} + \frac{6k}{6k-1} \cdot \frac{1}{6k} = \frac{3}{2} + \frac{1}{6k-1}.$$

2. A második eset, amikor $x_1 \leq \frac{3k+i^*-1}{6k}$, azaz $x \in I_{3k+i}$, $i < i^*$. Az S-beli második legnagyobb elem, x_2 mérete alapján szintén két lehetőségünk van.

a) Ha $x_2 > \frac{3k-i^*}{6k}$, akkor $w(x_2) = \frac{1}{2}$, és S-ben az összes többi (x_1-től és x_2-től különböző) x elem összmérete $1-x_1-x_2 < 1 - \frac{3k+i-1}{6k} - \frac{3k-i^*}{6k} = \frac{i^*-i+1}{6k}$. Így

$$w(S) = \sum_{x \in S} w(x) = w(x_1) + w(x_2) + \sum_{x \in S, x \neq x_1, x \neq x_2} w(x) \leq$$
$$\leq 1 - \frac{6k}{6k-1} \cdot \left(\frac{3k+i^*-1}{6k} - x_1 \right) + \frac{1}{2} + \frac{6k}{6k-1} \cdot \frac{i^*-i+1}{6k}.$$

Mivel $x_1 \leq \frac{3k+i-1}{6k}$, így ennek a kifejezésnek az értéke

$$\leq \frac{3}{2} - \frac{3k+i^*-1-(3k+i-1)}{6k-1} + \frac{i^*-i+1}{6k-1} \leq$$
$$\leq \frac{3}{2} - \frac{i^*-i}{6k-1} + \frac{i^*-i+1}{6k-1} = \frac{3}{2} + \frac{1}{6k-1}.$$

b) Az utolsó lehetőség, hogy $x_1 \leq \frac{3k+i^*-1}{6k}$, de $x_2 < \frac{3k-i^*}{6k}$. Ekkor x_2-t is beleértve S bármelyik, x_1-től különböző x elemére $w(x) \leq \frac{6k}{6k-1}x$ teljesül. Így az előző eset első ágához hasonló módon $w(S) < \frac{3}{2} + \frac{1}{6k-1}$.

□

A 3.1., 3.3. és 3.4. Lemmákból következik az alábbi

3.1. Tétel. *Bármely pozitív, de rögzített $k \geq 1$ egész esetén*

$$R(\mathit{UF}\text{-}k) \leq \frac{3}{2} + \frac{1}{6k-1}.$$

Bizonyítás. A 3.1. Lemmával beláttuk, hogy az **A.** esetbeli minden L listára létezik olyan M_1 konstans, hogy

$$\mathit{UF}\text{-}k(L) \leq \frac{3}{2} OPT(L) + M_1.$$

A **B.** esetben minden L listára a 3.4. Lemma miatt a lista $w(L)$ súlyára, amely a benne elhelyezett elemek súlyainak összegeként definiált, a következő teljesül:

$$w(L) = \sum_{B \in OPT(L)} \sum_{x \in B} w(x) = \sum_{B \in OPT(L)} w(B) \leq \left(\frac{3}{2} + \frac{1}{6k-1} \right) OPT(L).$$

Ugyanakkor a 3.3. Lemmából adódik, hogy az ilyen L listákra létezik olyan M_2 konstans, hogy

$$\mathit{UF}\text{-}k(L) \leq w(L) + M_2,$$

amiből adódik, hogy

$$\mathit{UF}\text{-}k(L) \leq \left(\frac{3}{2} + \frac{1}{6k-1} \right) OPT(L) + M_2.$$

Az $M := \max\{M_1, M_2\}$ választással adódik, hogy bármely L listára

$$\mathit{UF}\text{-}k(L) \leq \left(\frac{3}{2} + \frac{1}{6k-1} \right) OPT(L) + M.$$

□

A most következő lemma azt igazolja, hogy az algoritmusunkra bizonyított felső korlát éles.

3.5. Lemma. *Bármely pozitív, de rögzített $k \geq 1$ egész esetén*

$$R(\mathit{UF}\text{-}k) \geq \frac{3}{2} + \frac{1}{6k-1}.$$

Bizonyítás. Minden $t \in N^+$-ra konstruálunk egy $N = \left(4 + \frac{1}{6k-1}\right)n$ elemszámú L listát, ahol $n = 2t(6k-1)$.

Álljon L négy (rész)lista konkatenációjából, azaz legyen $L = L_1 L_2 L_3 L_4$, ahol L_1: $2t$-szer L_1^*, ahol L_1^*: $6k-1$ darab $\frac{1}{6k} - 2\varepsilon$ után 1 darab $(12k+1)\varepsilon$ méretű elem, ahol ε tetszőleges pozitív érték, melyre $\varepsilon < \min\left\{\frac{b}{12k+1}, \frac{b}{n+3}, \frac{1}{2t(12k+1)+n}\right\}$ teljesül, L_2: n darab ε méretű elem,

L_3: n darab $\frac{3k-1}{6k}+\varepsilon$ méretű elem,
L_4: n darab $\frac{1}{2}+\varepsilon$ méretű elem.

Az UF-k algoritmus úgy működik ezen az inputon, hogy L_1^* elemeit (NF szabály szerint) egy-egy $\mathcal{B}_1$-beli ládába pakolja. Ezek magassága pontosan $1-\frac{1}{6k}+3\varepsilon$, így a következő L_1^* első eleme már nem fér rájuk. Ezért L_1 elemeinek elpakolásához (NF szabály szerint) az algoritmus $2t=\frac{n}{6k-1}$ darab ládát használ fel. Ezután L_2 elemeit – mivel azok is $\mathcal{B}_1$-ből valók és ε méretük definíciója miatt befférnek az utolsó, L_1 érkezésekor létrehozott ládába – bepakolja ebbe – az NF szabály szerint –, majd az algoritmus L_3 elemeit elpakolja az NF szabállyal $\mathcal{B}_{3k}$-ba, $n/2$ darab ládát felhasználva. L_4 minden elemének érkezésekor az algoritmus nyit egy új ládát $\mathcal{B}_{3k+1}$-ben, belepakolja az elemet, majd átpakol egy L_2-beli ε méretű elemet ebbe a ládába. Így ez újabb n darab ládát jelent.

Egyszerűen ellenőrizhető, hogy bármely rögzített k-ra az UF-k algoritmusunk $n+\frac{n}{2}+\frac{n}{6k-1}$ ládát használ, és hogy $OPT(L)\le n+1$. Mivel ez tetszőleges t pozitív egészre teljesül, így ezzel állításunkat bizonyítottuk. □

A 3.1. Tétel és a 3.5. Lemma együttesen szolgáltatja a következő eredményt:

3.2. Tétel. *Bármely pozitív, de rögzített $k\ge 1$ egész esetén*

$$R(\textit{UF-k})=\frac{3}{2}+\frac{1}{6k-1}.$$

4. Összefoglalás

A 3.2. Tétel állítását megvizsgálva az UF-k algoritmusra vonatkozóan azt látjuk, hogy ha k tart a végtelenbe, akkor az algoritmus(ok) aszimptotikus versenyképessége, $R(UF-k)$ tart $1,5$-hez. Két konkrét esetet érdemes kiemelni: a $k=3$ esetben az UF-3 algoritmus aszimptotikus versenyképessége $1,5588...$, ami alatta van a legjobb ismert on-line algoritmusra bizonyított $1,58889$ aszimptotikus versenyképességnek. A $k=3$ eset másik érdekessége, hogy annak AVK-ja alatta van az úgynevezett Harmonikus Fit típusú on-line ládapakolási algoritmusokra bizonyított $1,583...$-as alsó korlátnak [11]. A $k=5$ esetben az UF-5 algoritmus AVK-ja $1,53448...$, amely jobb van Vliet $1,5401$-es on-line algoritmusokra vonatkozó alsó korlátjánál [13].

Számos nyitott kérdés felvethető, ezekből hármat említünk meg. Az első, hogy adjunk lépésenként 3-nál kevesebb elem átpakolását megengedő, $1,583...$-nál kisebb AVK-jú szemi-on-line ládapakolási algoritmust! A második, hogy adjunk lépésenként 5-nél kevesebb elem átpakolását megengedő, $1,5401$-nél kisebb AVK-jú algoritmust! Továbbá, a harmadik nyitott probléma kis k értékekre (pl. $k=1,2$) alsó korlát adása.

Köszönetnyilvánítás. A kutatást az OTKA (T 048377 és T 046822 számú projektek), valamint a MÖB-DAAD Magyar-Német Kutatócsere Program (21-es számú projekt) támogatta.

Hivatkozások

[1] E. G. Coffman – G. Galambos – S. Martello – D. Vigo: *Bin Packing Approximation Algorithms: Combinatorial Analysis.* In Handbook of Combinatorial Optimization (Eds. D.-Z. Du and P.M. Pardalos), pages 151-208. Kluwer Academic Publishers, (1999)

[2] G. Galambos: *A new heuristic for the classical bin packing problem.* Technical Report **82**, Institute fuer Mathematik, Augsburg, (1985)

[3] G. Galambos – G. J. Woeginger: *Repacking helps in bounded space on-line bin packing.* Computing, **49**: 329-338, (1993)

[4] G. Gambosi – A. Postiglione – M. Talamo: *Algorithms for the Relaxed Online Bin-Packing Model.* SIAM J. Computing, **30(5)**: 1532-1551, (2000)

[5] M. R. Garey – D. S. Johnson: *Computers and Intractability (A Guide to the theory of NP-Completeness).* W. H. Freeman and Company, San Francisco, (1979)

[6] E. F. Grove: *Online bin packing with lookahead.* SODA, 430–436, (1995)

[7] G. Gutin – T. Jensen – A. Yeo: *Batched Bin Packing.* Discrete Optimization, **2(1)**: 71-82, (2005)

[8] Z. Ivković – E. L. Lloyd: *A fundamental restriction on fully dynamic maintenance of bin packing.* Information Processing Letters, **59(4)**: 229-232, (1996)

[9] Z. Ivković – E. L. Lloyd: *Fully Dynamic Algorithms for Bin Packing: Being (Mostly) Myopic Helps.* SIAM J. Computing, **28(2)**: 574-611, (1998)

[10] C. C. Lee – D. T. Lee: *A Simple On-line Bin Packing Algorithm.* J. of ACM, **32**: 562-572, (1985)

[11] P. Ramanan – D. J. Brown – C. C. Lee – D. T. Lee: *On-line bin packing in linear time.* Journal of Algorithms, **10(3)**: 305-326, (1989)

[12] S. S. Seiden: *On the online bin packing problem.* Journal of the ACM, **49(5)**: 640-671, (2002)

[13] A. van Vliet: *An improved lower bound for online bin packing algorithms.* Information Processing Letters, **43(5)**: 277-284, (1992)

(Beérkezett: 2005. szeptember 29.)

BALOGH JÁNOS, GALAMBOS GÁBOR
SZEGEDI TUDOMÁNYEGYETEM,
JUHÁSZ GYULA PEDAGÓGUSKÉPZŐ KAR,
SZÁMÍTÁSTECHNIKA TANSZÉK,
6701 SZEGED, Pf. 396.
{balogh,galambos}@jgypk.u-szeged.hu

ALGORITHMS FOR THE ON-LINE BIN PACKING PROBLEM WITH REPACKING

János Balogh and Gábor Galambos

In the paper we deal with a semi-on-line bin packing problem. In this modification of the classical on-line bin packing problem some operations are allowed on the earlier packed elements while the algorithm packs the actual element in an on-line manner. We define an algorithm for every positive k value, which repacks at most k elements per step. We prove that the asymptotic competitive ratio of our algorithms is $\frac{3}{2} + \frac{1}{6k-1}$.