Gépütemezési feladatok moldable esetben

Tardos Zsófia
alkalmazott matematikus Msc

Témavezető: *Kis Tamás*
ELTE TTK Operációkutatási Tanszék

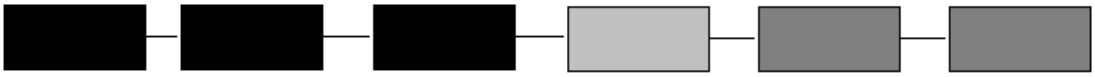
Budapest, 2012.

Tartalomjegyzék

1. Bevezető

Szakdolgozatom témája a párhuzamos szuperszámítógépek processzorainak minél jobb kihasználására vonatkozó algoritmusok bemutatása és elemzése. Párhuzamos szuperszámítógép alatt olyan rendszert értünk, amelyben több processzor áll összeköttetésben: a processzorok egy-egy részhalmaza tud egyszerre együtt egy feladaton dolgozni, miközben más részhalmazok más feladatot oldanak meg párhuzamosan. Ezeknek a processzor-rendszereknek sokféle architektúráját el lehet képzelni: ebben a szakdolgozatban olyan rendszert feltételezünk, ahol a processzorok sorban egymás mellett helyezkednek el: a szomszédos gépek állnak összeköttetésen egymással, így ezek tudnak együtt egy feladaton dolgozni. Azt is feltesszük, hogy egy-egy processzor egyszerre csak egy feladattal tud foglalkozni (azaz nincs olyan, hogy egy feladat pl. fél processzort vesz igénybe). A feladatokat függetlennek fogjuk tekinteni: nincsenek olyan kritériumaink, hogy bizonyos feladatok elvégzéséhez szükség van arra, hogy más feladatokat már korábban elvégezzünk.

Az 1. ábrán látható egy példa: egy hat processzorból álló rendszer egy adott pillanatban, ahol az első három processzor együtt dolgozik egy feladaton, a negyedik egyedül egy másikon, az ötödik és a hatodik pedig együtt egy harmadikon.



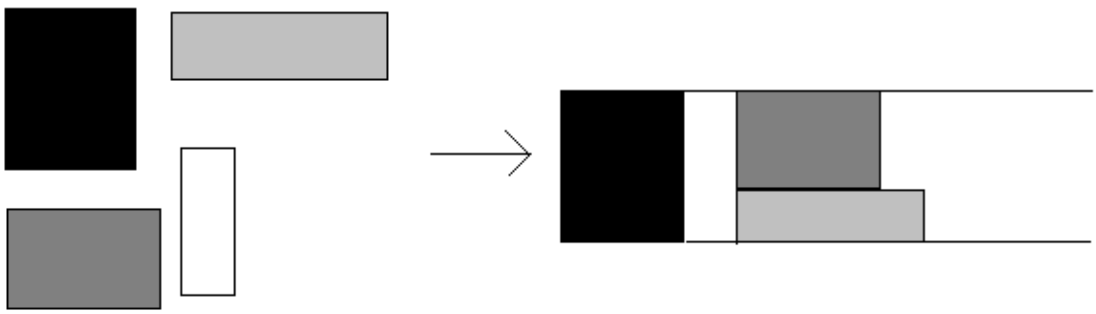
1. ábra. Párhuzamos feladatmegoldás

Azt, hogy mit tekintünk optimálisnak, többféle célfüggvény alapján is definiálhatjuk. Leggyakoribb cél, hogy minimalizáljuk a legkésőbb befejeződő feladat befejezési idejét, azaz azt, hogy mikorra készülünk el az egész munkával. Ha nem jelezzük külön, hogy más célfüggvényről van szó, akkor optimalitás alatt az ezzel a célfüggvénnyel definiáltat fogjuk érteni. Egy másik lehetőség, hogy előre megadunk egy határidőt,

ameddig be kellene fejezni a feladatokat, és a cél az, hogy a határidő után a processzorok minél kevesebb munkát végezzenek összesen.

A szuperszámítógépekre való hatékony beosztással foglalkozó kérdéseknek három különböző csoportját különböztethetjük meg aszerint, hogy milyen módon kell döntenünk arról, hogy egy feladatot hány processzoron szeretnénk megoldani.

1. *Rögzített méretű feladatok:* Adott, hogy egy-egy feladat hány processzoron mennyi ideig fut, nekünk csak azt kell eldöntenünk, hogy melyik feladatot melyik meghatározott számú processzoron, és mikor futtassuk. Ezt a feladatot elképzelhetjük a következő módon: adottak téglalapjaink (ezek a feladatok), amelyeknek magassága megfelel az általa használt processzorok számának, szélessége pedig a futási idejének. Adott egy valamilyen magasságú sávunk (a magassága annak felel meg, hogy hány processzorunk van összesen.) A feladat az, hogy a téglalapokat minél optimálisabban bepakoljuk egy adott kezdési időponttól kezde a sávba. A 2. ábrán látható erre a feladatra egy példa: 4 feladatot szeretnénk futtatni egy három processzorból álló rendszeren.



2. ábra. Rögzített méretű feladatok optimális ütemezése

2. *Moldable eset:*A szakdolgozat legnagyobb részében ezzel az esettel fogunk foglalkozni. Itt nem adjuk meg előre, hogy egy-egy feladatot hány processzoron futtatjuk, hanem megadunk egy táblázatot, amelyben az szerepel, hogy egy-egy feladat megoldása mennyi ideig tart annak függvényében, hogy hány processzort

használ. A következő két természetes monotonitási feltételezéssel élünk:

A feladatok elvégzése szükséges idő a használt processzorok számának függvényében monoton csökken.

Az egy-egy feladat elvégzéséhez szükséges munka (futási idő szorozva a processzorok számával) a használt processzorok számának függvényében monoton nő

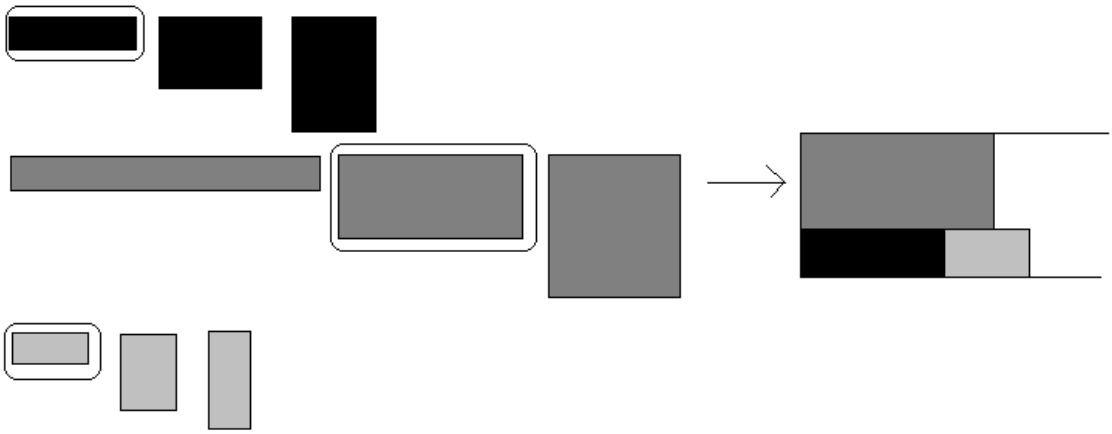
A második feltételezésünk abból fakad, hogy ha megosztunk egy feladatot több processzor között, akkor azoknak el kell végezniük ugyanannyi számítást összesen, mintha csak kevesebb processzort használnánk, viszont egymás között többet kell kommunikálniuk.

Ezt a feladatot a következő módon képzelhetjük el: most egy-egy feladathoz több téglalap is tartozik, (amelyeknek magassága a felhasznált processzorok számát, szélessége az elvégzési időt jelenti továbbra is), ezek közül kell egyet-egyet kiválasztanunk, és behelyeznünk a kiválasztottakat a sávba. (Ez a téglalapos interpretáció azért is nagyon kényelmes, mert az egyes feladatokhoz tartozó munka megfelel a téglalap területének, azaz a második monotonitási kritérium megfelel annak, hogy minél magasabb egy téglalap, annál nagyobb a területe.)

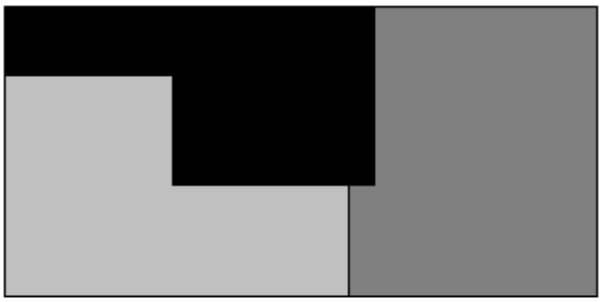
A 3. ábrán erre a típusra látunk egy példát: 3 feladatot szeretnénk futtatni egy 3 processzorból álló rendszeren.

3. *Malleable eset:* Ebben az esetben az egy feladat által használt feladatok számát idő közben változtathatjuk (pl. egy darabig egy processzort haszmálunk, aztán felszabadul egy másik processzor, úgyhogy kettőn futtatjuk, végül megint visszatérünk arra, hogy csak egy processzort használunk a megoldásához). Egy ilyen típusú beosztásra látunk példát a 4. ábrán. Ezzel a feladattal a dolgozatban nem foglalkozunk.

Ezek a feladatok mind NP-teljes feladatok (a PARTÍCIÓ feladatot vissza lehet vezetni arra a feladatra, amikor a feladatokat 1-1 processzoron futtathatjuk, és két



3. ábra. Optimális ütemezés moldable esetben



4. ábra. Mallaeble beosztás

processzorra szeretnénk beosztani őket. (PARTÍCIÓ feladat: egész számok egy halmazáról kell eldönteni, hogy ketté lehet-e őket osztani úgy, hogy a két részhalmazban lévő számok összege megegyezzen. Ez egy NP-teljes-feladat). Ezért aztán az ilyen jellegű feladatoknál csak az lehet a célunk, hogy minél hatékonyabb és minél jobb approximációs algoritmusokat találjunk.

A szakdolgozat a következő módon fog felépülni: a 2. fejezetben bemutatunk egy 2,5-approximációs algoritmust a rögzített feladatok esetére, majd ezt az eredményt felhasználva bemutatunk egy 2,5-approximációs algoritmust a moldable-esetre. A 3. fejezetben egy olyan 2-approximációs algoritmust ismertetünk, amely nem használ visszavezetést a rögzített méretű feladatokra (azaz nem különül el az a fázis, amikor döntünk a feladatok processzorszámáról, és az, amikor behelyezzük őket a sávba).

Az itt szereplő 7. állítás bizonyítása önálló eredmény. A 4. fejezetben bemutatunk egy duál-approximációs szemléletet felhasználó megoldási módot, ami 1,5-approximációt ad. Ezt az algoritmust impletáltuk: az 5. fejezetben bemutatjuk a tesztelés eredményeit. A 6. fejezetben megvizsgáljuk azt a speciális esetet, ha minden feladat egyforma. A 7. fejezetben foglalkozunk a határidő utáni munka minimálizálására vonatkozó optimalizálással, ebben a fejezetben önálló eredmények szerepelnek. A 8. fejezetben összefoglaljuk az eredményeket, és megemlítünk néhány nyitott kérdést.

1.1. jelölések

n: feladatok száma

m: processzorok száma

$p : (p_1, p_2, ..., p_n)$: az i. koordináta azt jelöli, hogy az i. feladatot hány processzorral végeztetjük

$t_i(j)$: az i. feladat futási ideje j processzoron

$w_i(j)$: az i. feladat j processzoron való futása esetén fennálló munka: $t_i(j)j$

$\gamma_i(d)$: minimum ennyi processzor kell ahhoz, hogy az i. feladat d időn belül befejeződjön.

C_{max}^{opt}: befejezési idő egy optimális beosztás esetén

C_{max}^{alg}: befejezési idő az algoritmus által adott beosztás esetén

$\omega(p) := \max(\frac{\sum w_i(p_i)}{m}, \max t_i(p_i))$

$\omega := \min_p \omega(p)$

$h(p)$: egy adott p processzorszámkiosztás esetén a legtovább tartó feladat futási ideje

2. A moldable eset megoldása rögzített feladatok esetére való visszavezetéssel

2.1. 2,5-approximációs algoritmus rögzített méreű feladatok esetén

Vizsgálódásunkat kezdjük először a rögzített méretű feladatok esetével. Adott n feladatunk, és m processzorunk. Minden feladathoz adott egy p_i processzorszám és egy t_i idő. Tegyük fel, hogy létezik olyan beosztás, amire a befejezési idő d. Bemutatunk egy olyan $O(n \log n)$ futásidejű algoritmust [7], ami olyan beosztást talál, ahol a befejezési idő (jelöljük C_{max}^{alg}-gal) $\leq 2,5d$.

Az algoritmus a következő:

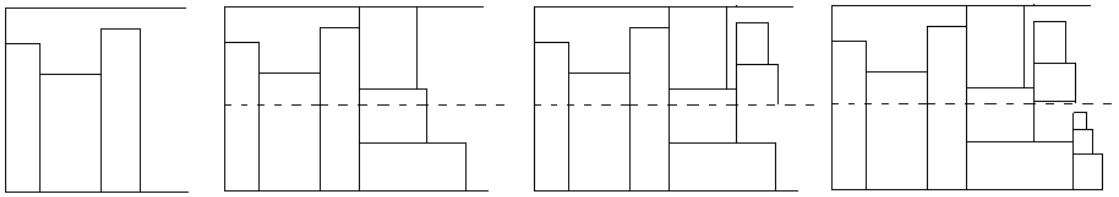
1. Kiválogatjuk azokat a feladatokat, amikre $p_i > \frac{m}{2}$. Ezeket a feladatokat egymás után behelyezzük.

2. t_i szerinti csökkenő sorrendbe rendezzük a feladatokat.

3. Ebben a sorrendben egymás fölé (azaz egymással párhuzamosan futtatva) behelyezünk annyi feladatot, amennyit tudunk (mivel itt már minden feladatra $p_i \leq \frac{m}{2}$, ezért ebben a lépésben legalább két feladatot helyezünk el párhuzamosan).

4. Kialakítunk két $\frac{m}{2}$ processzorból álló polcot (ha m páratlan, akkot a "fél" processzorokat kitöröljük). Mindig abba a polcba helyezünk egy réteget, amelyikben épp kevesebb ideig tart a munka. A feladatok nem lóghatnák át egyik polcból a másikba, vagyis az egy lépésben behelyezett feladatokhoz rendelt processzorok összege $\leq \frac{m}{2}$

Ezen algoritmus működésére látunk egy példát az 5. ábrán.

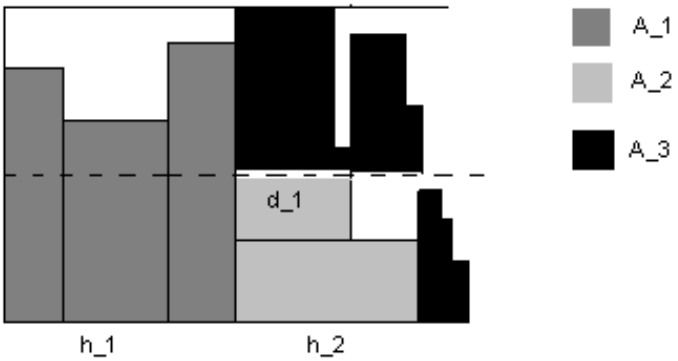


5. ábra. Az algoritmus lépései

A 2. lépésben n feladatot sorba kell rendeznünk, ennek időigénye: $O(n \log n)$. Ez dominálja a egész algoritmus futásidejét. (Hiszen az algoritmus többi részének futásideje csak $O(n)$.)

1. Tétel. *Az algoritmus 2,5-approximációs.*

Biz. Vezessük be a következő (6. ábrán szereplő) jelöléseket: jelölje A_1 azon feladatok összterületét, amelyeket az 1. lépésben helyeztünk be, A_2 az ezután következő réteg alsó térfélbe eső részét, A_3 pedig a maradék területet. Jelölje h_1 az első fázisban bepakolt feladatok összidejét, h_2 a második rétegben először bepakolt feladat futási idejét, d_1 pedig a második rétegben a két térfél között átlógó feladat, vagy ha ilyen nincs, akkor a felső térfél legalsó feladatának futásidejét. Jelöljük az A_2 területét adó feladatok összidejét D-vel, és a két polc befejezési idejének különbségét e-vel.



6. ábra. Az 1. Tétel bizonyításában használt jelölések

Az lesz a módszerünk, hogy C_{max}^{alg}-ot az összterület segítségével becsüljük felül, amelyet pedig felülről becsül C_{max}^{opt}: ebből fogjuk levezetni a tételben szereplő becslést.

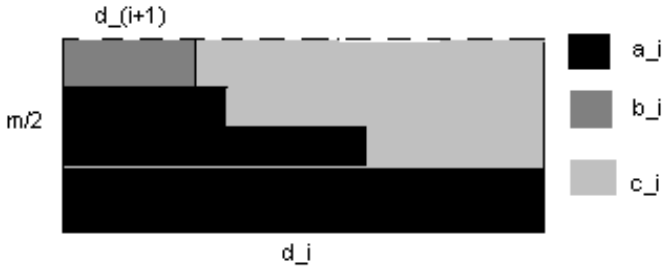
$$2C_{max}^{alg} = 2h_1 + h_2 + D + e$$

Az első tagot könnyen tudjuk becsülni:

$$h_1 \frac{m}{2} \leq A_1$$
$$2h_1 \leq \frac{4A_1}{m}$$

1. Állítás. $D \leq \frac{4A_3}{m} + d_1$

Biz. Jelöljük a negyedik lépésben bepakolt rétegek futási idejét sorra d_2, d_3, ... d_k-val. Minden egyes rétegnél a következő (7. ábrán szereplő) jelöléseket vezetjük be: a_i: a feladatok összterülete, b_i: Az adott rétegben üresen maradó processzorokon a réteg kezdési időpontjától kezdődő d_{i+1} hosszú terület, c_i: az adott polcrészben fennmaradó terület. (Ezeket a definíciókat terjesszük ki értelemszerűen $i = 1$-re is.) Az utolsó, k. rétegben b_n legyen üres.



7. ábra.

Így $A_3 = \sum a_i$. Jelöljük $\sum b_i$-t B-vel és $\sum c_i$-t C-vel.

Világos, hogy $D\frac{m}{2} = A_3 + B + C$. B-t és C-t kell tehát felülről becsülnünk. Figyeljük meg, hogy $b_i \leq a_{i+1}$, mivel az $(i+1)$-edik réteg első feladatának processzorszáma nagyobb, mint ahány processzort lefed a b_i terület (különben befért volna az i. rétegbe a feladat), a hosszuk pedig megegyezik. Mivel $b_n = 0$, így következik, hogy $A_3 \geq B$.

Következik C becslése: c_i legnagyobb szélessége $d_i - d_{i+1}$, kivéve c_n-t, ahol d_n. Azaz $C \leq \frac{m}{2} \sum_{i=1}^{n-1}(d_i - d_{i+1}) + d_n = \frac{d_1 m}{2}$

Ezekből $D \leq 2\frac{A_3 + A_3 + \frac{d_1 m}{2}}{m}$, vagyis $D \leq 4\frac{A_3}{m} + d_1$ □

A h_1-re és D-re kapott becsléseinket felhasználva kapjuk, hogy

$$2C_{max}^{alg} \leq \frac{4A_1}{m} + h_2 + \frac{4A_3}{m} + d_1 + e$$

A $\frac{d_1 m}{2} \leq A_2$, azaz $\frac{4A_2}{m} \leq 2d_1$ egyenlőtlenséget felhasználva:

$$2C_{max}^{alg} \leq 4\frac{A_1 + A_2 + A_3}{m} + h_2 + e - d_1$$

$\frac{A_1 + A_2 + A_3}{m}$-nál nem lehet kisebb C_{max}^{opt}:

$$2C_{max}^{alg} \leq 4C_{max}^{opt} + h_2 + e - d_1$$

Ha a negyedik fázisban volt olyan pillanat, amikor a felső térfélen tovább tartott a munka, mint az alsóban, akkor teljesül, hogy $e \leq d_1$, hiszen nincs d_1-nél hosszabb réteg. Azaz ekkor $2C_{max}^{alg} \leq 4C_{max}^{opt} + h_2$. $h_2 \leq C_{max}^{opt}$, így megkaptuk a kívánt $C_{max}^{alg} \leq 2,5C_{max}^{opt}$ becslést.

Ha pedig egyszer sem volt olyan pillanat, amikor a felső térfélen tovább tartott a munka, mint az alsóban, akkor az azt jelenti, hogy az összes feladatot sikerült $h_1 + h_2$ idő alatt befejeznünk. $h_2 \leq C_{max}^{opt}$, és $h_1 \leq C_{max}^{opt}$ szintén, mivel az első fázisban berakott feladatokat nem tudjuk közös processzorokon futtatni. Azaz ebben az esetben $C_{max}^{alg} \leq 2C_{max}^{opt}$. □

Valójában a következő erősebb [5]-ben szereplő állítás is kijön a bizonyításból:

2. Állítás. $C_{max}^{alg} \leq 2,5 \max(\frac{\sum w_i}{m}, \max t_i)$

2.2. Rögzített méretű feladatok pakolására való visszavezetés

A moldable probléma megoldására egy kézenfekvő ötlet lehet az, hogy két fázisban oldjuk meg a problémát: először rögzítsük, hogy melyik feladathoz hány processzort

rendelünk, majd utána a már rögzített méretű feladatokat pakoljuk be minél hatékonyabban. Ezt a módszert alkalmazzák a [5]-ben szereplő, alább bemutatott algoritmusban. Egy adott processzorszám-kiosztás esetén jelöljük p-rel azt a vektort, amelynek i. koordinátája megadja, hogy az i. feladatot hány processzoron futtatjuk.
Vezessük be a következő jelöléseket:

$$h(p) : \text{a leghosszabb ideig tartó feladat futási ideje}$$
$$\omega(p) := \max(\tfrac{\sum w_i(p_i)}{m}, h(p))$$
$$\omega := \min_p \omega(p)$$

Az világos, hogy $C^{opt}_{max} \geq \omega$ Ha ismernénk, hogy melyik $\overline{p}$ processzorszám-kiosztásra minimalizálódik az $\omega(p)$, akkor erre a $\overline{p}$-re alkalmazhatnánk a rögzített méretű feladatok bepakolására szóló előző algoritmust: a 2. állítás szerint erről tudjuk, hogy olyan beosztást talál, amelynek a befejezési ideje $\leq 2,5\omega$, és így a moldable problémára is találnánk egy 2,5-approximációs algoritmust. Azaz meg kell találnunk $\omega(p)$ minimumhelyét.

$$\min \omega(p) = \min_p \max(\tfrac{\sum p(i)t_i(p(i))}{m}, h(p)) =$$
$$\min_{\tau\in\mathbb{R}} \max(\min_p(\tfrac{\sum_{i\in feladatok}(p(i)t_i(p(i)))}{m} \mid h(p) \leq \tau), \tau)$$

A területek monotonitása alapján:

$$\min_p(\tfrac{\sum_{i\in feladatok}(p(i)t_i(p(i)))}{m} \mid h(p) \leq \tau) = \sum \gamma_i(\tau)t_i(\gamma_i(\tau)) := W(\tau).$$

Azaz

$$\min w(p) = \min_{\tau\in\mathbb{R}} \max(W(\tau), \tau) \ .$$

Ezt a minimalizációs feladatot fogjuk megoldani bináris kereséssel. Ehhez a következő állításokat kell belátnunk:

1. $\exists\overline{\tau}$, hogy $\exists(i,j) : t_i(j) = \overline{\tau}$ és $\min\max_\tau(W(\tau), \tau) = \max(W(\overline{\tau}), \overline{\tau})$.

2. Ha $(W(\tau) > \tau)$, akkor $\exists\overline{\tau} \geq \tau$, hogy $\min_\tau \max(W(\tau), \tau) = \max(W(\overline{\tau}), \overline{\tau})$.

3. Ha $(W(\tau) \leq \tau)$, akkor $\exists \overline{\tau} \leq \tau$, hogy $\min_\tau \max(W(\tau), \tau) = \max(W(\overline{\tau}), \overline{\tau})$.

3. Állítás. *$\exists \overline{\tau}$, hogy $\exists (i,j) : t_i(j) = \overline{\tau}$ és* $\min \max(W(\tau), \tau) = \max(W(\overline{\tau}), \overline{\tau})$

Biz. Az világos, hogy felvétetik valahol a $\min_\tau \max(W(\tau), \tau)$, mert elég a $[0, max_i(t_i(1))]$ intervallumon keresnünk, ahol két folytonos függvény maximumának minimuma felvétetik. Tegyük fel, hogy a $\dot{\tau}$ helyen felvétetik a minimum. Ekkor a $\bar{\tau} = \max_i \mathrm{t}_i(\gamma_i(\dot{\tau}))$ helyen is felvétetik a minimum, és ekkor $\exists (i,j) : t_i(j) = \overline{\tau}$. □

4. Állítás. *Ha $(W(\tau) > \tau)$, akkor $\exists \overline{\tau} \geq \tau$, hogy* $\min_\tau \max(W(\tau), \tau) = max(W(\overline{\tau}), \overline{\tau})$.

Biz. Tegyük fel, hogy $\exists \tau' < \tau$ és τ'-nál felvétetik a minimum. A területek monotonitása következtében ekkor $W(\tau) \leq W(\tau')$. Azaz $\tau' < \tau < W(\tau) \leq W(\tau')$, így $\max(\tau', W(\tau')) = W(\tau') \geq W(\tau) = \max(\tau, W(\tau))$, így τ-nál is felvétetik a minimum. □

5. Állítás. *Ha $(W(\tau) \leq \tau)$, akkor $\exists \overline{\tau} \leq \tau$, hogy* $\min \max(W(\tau), \tau) = \max(W(\overline{\tau}), \overline{\tau})$

Az állítást pont ugyanúgy bizonyíthatjuk, mint az előzőt.

A bináris keresés $O(\log mn)$ időt vesz igénybe, de előtte még növekvő sorrendbe kell rendeznünk a lehetséges τ, azaz a $t_i(j)$ értékeket, amit $O(mn \log n)$ idő alatt tehetünk meg.

Azaz találtunk egy 2,5-approximációs algoritmust $O(mn \log n)$ futási idővel.

3. Kis és nagy feladatok szétválasztásán alapuló algoritmus

Bár a rögzített méretű feladatokra csak 2,5-approximációs algoritmust ismerünk, a moldabe-esetre léteznek jobban közelítő algoritmusok is. Ebben a fejezetben bemutatunk egy [2]-ben szereplő 2-közelítő algoritmust, aminek a lényege az lesz, hogy minden lépésben meghatározunk egy τ értéket és ezen érték függvényében szétválasztjuk a feladatokat kicsikre és nagyokra: (az lesz kis feladat, ami 1 processzoron is befejeződik ezen értéknél röviebb idő alatt.) A kis feladatokat és a nagy feladatokat két fázisban külön processzorokon az alábbiakban részletezett mohó módon oldjuk meg. Bináris kereséssel megkeressük, hogy melyik τ értékre a legjobb ez a mohó módon elért eredmény.

Jelöljük adott τ-ra a kis feladatok befejeződési idejét $C^K_{max}(\tau)$-val.

Az algoritmus egy fázisa adott τ-val:

1. Kettébontjuk a feladatokat: $i \in K(\tau)$, ha $t_i(1) \leq \tau$, és $i \in N(\tau)$, ha $t_i(1) > \tau$.

2. Beosztjuk a nagy feladatokat: minden nagy feladat külön processzorra kerü: $\gamma_i(\tau)$ darabra, a kezdőidőpontjuk 0. Ha $\sum \gamma_i(\tau) > m$, akkor itt leállunk, és $C^K_{max}(\tau)$-nek ∞ értéket adunk. Különben $m - \sum \gamma_i(\tau)$-t jelöljük k-val.

3. A kis feladatokat a maradék k processzorra osztjuk be, méghozzá úgy, hogy minden kis feladatot 1 processzorra teszünk. Ezt a következő módon valósítjuk meg: csökkenő sorrendbe tesszük a feladatokat $t_i(1)$ szerint. Először az első k feladatot helyezzük el a k processzoron (mindegyikre egyet-egyet), majd mindig arra a processzorra tesszük a soron következő elemet, amelyiken a legrövidebb ideig futnak éppen a feladtok. Ha $k = 0$, és van kis feladat, akkor $C^K_{max}(\tau) = \infty$.

A τ-hoz tartozó befejezési idő: $C_{max}(\tau) = \max(C^K_{max}(\tau), \max_{i \in N} t_i(\gamma_i(\tau))$.

Célunk tehát egy olyan τ-t találni, amire $C_{max}(\tau)$ minimális. Érdemes megint megfigyelnünk, hogy elég τ-t a $t_i(j)$ értékek között keresni hasonló megfontolásból

mint az előző algoritmusban. Azaz

$$C_{max}(\tau) = \max(\tau, C^K_{max}(\tau)).$$

Jelöljük a lehetséges $t_i(j)$ értékek halmazát X-szel.

Két dolgot kell tennünk: egyrészt meg kell mutatnunk, hogy hogy tudunk megtalálni egy olyan $\overline{\tau}$-t, ahol ez a minimum felvétetik. Másrészt be kell bizonyítanunk, hogy $C_{max}(\overline{\tau}) \leq C^{opt}_{max}$

2. Tétel. $C_{max}(\overline{\tau}) \leq 2C^{opt}_{max}$

Biz. Adjunk egy alsó korlátot C^{opt}_{max}-ra! Jelöljük a τ értékkel futtatott algoritmussal kapott legrövidebb ideig futó nagy feladat futási idejét $l_N(\tau)$-val, és a kis feladatok által legrövidebb ideig használt processzor leállási idejét $l_K(\tau)$-val. Legyen $l(\tau) := min(l_N(\tau), l_K(\tau))$ (Azaz minden processzor legalább $l(\tau)$ ideig fut.) (Ha $k < 1$, akkor $l_K(\tau)$ legyen végtelen). □

6. Állítás. *Tetszőeges τ esetén $l(\tau) \leq C^{opt}_{max}$.*

Biz. Az algoritmus egy olyan beosztást adott, amelyben minden processzor legalább $l(\tau)$ ideig fut. Azaz, ahhoz, hogy találjunk egy olyan beosztást, ami kevesebb, mint $l(\tau)$ idő alatt befejeződik, csökkentenünk kell az összmunkát. De ez lehetetlen, mert sem a nagy feladatokat nem tehetjük kevesebb processzorra, mert akkor nem fejeződnének be $l(\tau)$ idő alatt, sem a kicsiket, mivel azokat az algoritmus már eredetileg is csak egy-egy processzorra tette. □

Azaz ahhoz, hogy bebizonyítsuk a tételt, elég, ha mutatunk olyan $\hat{\tau}$-t, amelyre $C_{max}(\tau) \leq 2l(\hat{\tau})$.

Jelöljük τ^*-gal a következő értéket: $\max\{\tau \,|\tau \in X\}$ és $C_{max}(\tau) = \min C_{max}(\tau)\}$

1. eset: τ^* esetén az algoritmus olyan beosztást ad, hogy létezik olyan processzor (ennek indexét jelöljük y-nal), amin 1-nél több kis feladat fut, és amelyen a befejezési

$id\H{o} = C_{max}(\tau^*)$

Az alábbi állítást látjuk be:

7. Állítás. $C_{max}(\tau^*) \leq 2l(\tau^*)$.

Biz. Egyrészt $C_{max}(\tau^*) \leq 2l_K(\tau^*)$: vizsgáljuk azt a pillanatot, amikor behelyeztük az utolsó feladatot az y. processzorra. Ha lenne olyan processzor, amire kis feladatokat pakolhatunk, és ahol a feladatok befejezési ideje kisebb, mint $\frac{C_{max}(\tau^*)}{2}$, akkor az y. processzor utolsó feladatát ide pakoltuk volna. (Hiszen a feladatokat csökkenő sorrendben pakoljuk be, úgyhogy az y. processzor utolsó előtti feladata nem fejeződhet be $\frac{C_{max}(\tau^*)}{2}$-nél korábban.)

Másrészt be kell bizonyítanunk, hogy $C_{max}(\tau^*) \leq 2l_N(\tau^*)$.

Vezessük be a következő jelölést: $\tau^{\oplus}$ a legkisebb olyan szám, amelyre teljesül, hogy $\tau^{\oplus} \in X$, és $\tau^{\oplus} > \tau^*$.

τ^* definíciója miatt $C_{max}(\tau^*) < C_{max}(\tau^{\oplus})$. $\square$

8. Állítás. $C_{max}(\tau^{\oplus}) = \tau^{\oplus}$

Biz. Vizsgáljuk meg, hogy mi a különbség a $K(\tau^*)$ - $N(\tau^*)$ és a $K(\tau^{\oplus})$-$N(\tau^{\oplus})$ felosztások között! Azok a feladatok, amik 1 processzoron $\tau^{\oplus}$ ideig tartanak, $N(\tau^*)$-ba és $K(\tau^{\oplus})$-ba tartoznak. A többi feladat vagy mindkettőben kicsi, vagy mindkettőben nagy. Jelöljük azon feladatok számát, amelyek átkerülnek a nagyok halmazából a kicsikébe, z-vel! A $K(\tau^{\oplus})$ feladatainak beosztásához rendelkezésre álló processzorok száma legalább $2z$-vel több, mint a $K(\tau^*)$ feladataihoz rendelkezésre álló processzorok száma (hiszen a nagy feladatok mindig legalább 2 processzorra kerülnek.) Így $C^K_{max}(\tau^*)$ csak úgy lehet kisebb $C^K_{max}(\tau^{\oplus})$-nál, ha $C^K_{max}(\tau^{\oplus}) = \tau^{\oplus}$ (mivel annál nem adhat rosszabb eredményt az algoritmus, ha az eredetileg is kis feladatokat ugyanúgy helyezzük el, mint τ^* esetében, és az újonnan átkerült feladatokat egy-egy processzorra tesszük (azokra a processzorokra, amikre korábban tettük őket, amikor nagy feladatok voltak)). Ha pedig $C^K_{max}(\tau^{\oplus}) \leq C^K_{max}(\tau^*)$, akkor mivel $C_{max}(\tau^*) < C_{max}(\tau^{\oplus})$, következik, hogy $C_{max}(\tau^{\oplus}) = \tau^{\oplus}$. $\square$

Tehát a 7. állításhoz annyit kell még bebizonyítanunk, hogy $\tau^{\oplus} \leq 2l_N(\tau^*)$. Ehhez még az alábbi állításra lesz szükségünk:

9. Állítás. *Ha az i. feladat $p_i > 1$ processzoron $t_i(p_i)$ ideig tart, akkor p_{i-1} processzoron legfeljebb $2t_i(p_i)$ ideig tart.*

Biz.

A területek monotonitása következtében $p_i t_i(p_i) \geq (p_i - 1)t_i(p_i - 1)$, amiből $t_i(p_i - 1) \leq \frac{p_i}{p_i - 1} t_i(p_i)$, vagyis $t_i(p_i - 1)) \leq 2t_i(p_i)$ □

Legyen a τ^* esetén legrövidebb ideig futó nagy feladat indexe u. A területek monotonitása alapján tudjuk, hogy $t_u(\gamma_u(\tau^*)) \geq \frac{\tau^*}{2}$. Tegyük fel, hogy $t_u(\gamma_u(\tau^*)) < \frac{\tau^{\oplus}}{2}$. Ekkor a 8. állítás alapján $t_u(\gamma_u(\tau^*) - 1) < \tau^{\oplus}$. Másrészt $t_u(\gamma_u(\tau^*) - 1) > \tau^*$. Ez viszont $\tau^{\oplus}$ definíciója alapján ellentmondás. Vagy is adódik, hogy $\tau^{\oplus} \leq 2l_N(\tau^*)$, így $C_{max}(\tau^*) \leq C_{max}(\tau^{\oplus})$. Ezzel bebizonyítottuk a 7. állítást. □

2. eset: τ^* esetén az algoritmus olyan beosztást ad, hogy nem létezik olyan processzor, amin 1-nél több kis feladat fut, és amelyen a befejezési idő $= C_{max}(\tau^*)$

10. Állítás. *A 2. esetben $C_{max}(\tau^*) = \tau^*$*

Biz. Ha valamelyik nagy feladat éri el $C_{max}(\tau^*)$-t, akkor adódik az állítás. Ha pedig az i. kis feladat hossza $C_{max}(\tau^*)$, akkor $\tau^* \leq C_{max}(\tau^*) = t_i(1) \leq \tau^*$. □ Vezessük be a következő jelölést: $\tau^{\ominus}$ legyen az a legnagyobb szám, melyre $\tau^{\ominus} \in X$, és $\tau^{\ominus} < \tau^*$.

11. Állítás. $C_{max}(\tau^*) \leq 2l(\tau^{\ominus})$

Biz. Azt tudjuk, hogy $C_{max}(\tau^{\ominus}) \geq C_{max}(\tau^*)$. Ez csak úgy teljesülhet, ha $C_{max}(\tau^{\ominus}) = C^K_{max}(\tau^{\ominus})$, mivel $\tau^{\ominus} < \tau^* = C_{max}(\tau^*)$.
Az előző esetben beláttuk, hogy $C^K_{max}(\tau^*) \leq 2l_K(\tau^*)$. Ez a bizonyítás nem használt ki τ^*-ról semmit: ugyanígy igaz $C^K_{max}(\tau^{\ominus}) \leq 2l^K(\tau^{\ominus})$ is. Így $C_{max}(\tau^*) \leq 2l_K(\tau^{\ominus})$.

Az állításhoz szükséges másik egyenlőtlenség: $C_{max}(\tau^*) = \tau^* \leq 2l_N(\tau^{\ominus})$. Ez ugyanúgy bizonyítható, mint az előző esetben a $\tau^{\oplus} \leq 2l_N(\tau^*)$ egyenlőtlenség. □

[5]-ben mutatnak egy példát arra, hogy az algoritmus által számolt $C_{max}(\tau^*)$ és C_{max}^{opt} aránya tetszőlegesen megközelítheti a 2-t: legyen n darab feladatunk, és $2n-1$ darab processzorunk. A feladatok legyenek egyformák: $t_i(j) = 1/j \forall i$. Ekkor ha τ-t kisebbnek választjuk 1-nél, akkor minden feladat nagy lesz, és minden feladat legalább 2 processzorra kerül. Azaz nem lesz elég processzorunk, vagyis a befejezési idő végtelen. Ezek szerint az algoritmus τ-t 1-nek választja. Ekkor minden feladat kicsi lesz, és mindegyik 1-1 (különböző) processzorra fog kerülni. Azaz az algoritmus által talált befejezési idő: 1. Az optimális megoldás pedig az lenne, ha minden feladatot $2n-1$ processzorra tennénk. Ekkor a befejezési idő: $\frac{n}{2n-1}$

Hogyan találunk egy $\bar{\tau}$-t, ahol $C_{max}(\tau)$ minimuma felvétetik?

Megint bináris kereséssel szeretnénk keresni, amihez az alábbi állításokra lesz szükségünk:

12. Állítás. *Ha egy adott τ-hoz $\exists\ (i,j) : t_i(j) = \tau$, azaz a befejezési idő felvétetik olyan processzoron, amin egy feladat fut, akkor $\exists\ \grave{\tau} \leq \tau$, amire felvétetik az optimum.*

Biz. Mivel $C_{max}(\tau) = \max(\tau, C_{max}^K(\tau)$, ezért ha $\exists\ (i,j) : t_i(j) = \tau$, akkor $C_{max}(\tau) = \tau$. Ha pedig $\grave{\tau} > \tau$, akkor $C_{max}(\grave{\tau}) \geq \grave{\tau} > \tau = C_{max}(\tau)$. □

13. Állítás. *Ha egy adott τ esetén a befejezési idő felvétetik olyan processzoron, amelyen több feladat is fut, akkor $\exists\ \grave{\tau} \geq \tau$, amire felvétetik az optimum.*

Biz. Ha csökkentjük τ-t, akkor ami eddig kis feladat volt, az az is marad, és a kis feladatok rendelkezésére álló processzorok száma sem nőhet, azaz $C_{max}^S(\tau) \leq C_{max}^S(\grave{\tau})$. □Ezen állítások alapján bináris kereséssel megkereshetjük az optimális τ-t. Az előző algoritmushoz hasonlóan ez az algoritmus is $O(mn \log n)$ időt vesz igénybe.

4. Kétpolcos algoritmus

4.1. Duál-approximációs szemlélet

Ebben a fejezetben egy $(1,5+\epsilon)$-approximációs algoritmust mutatunk be, amely duál-approximáción alapul. Mielőtt erre az aloritmusra rátérnénk ebben a fejezetben először bemutatunk egy duál-approximációs algoritmust [3] egy a moldable-esetnél jóval egyszerűbb feladatra.

Legyen most n feladatunk és m processzorunk. A feladatok most csak 1 processzoron futtathatók, az i. feladat 1 processzoron való futási ideje t_i. Célunk, hogy olyan beosztást találjunk, amelynek a befejezési ideje minél kisebb. Ennek a feladatnak az optimális megoldása is NP-nehéz, mert a PARTÍCIÓ feladat erre is visszavezethető. Ezért megelégszünk egy $(1+\delta)$-közelítő algoritmussal valamilyen adott δ-ra.

Ezt a feladatot megfeleltethetjük a következő egydimenziós ládapakolási feladatnak: adott n tárgyunk, t_i méretekkel, valamint adottak azonos méretű ládáink. A célunk az, hogy minél kevesebb ládába bepakoljuk a feladatokat. (Néhány feladat akkor pakolható be együtt egy ládába, ha azok összsúlya nem haladja meg a láda méretét.)

A megfeleltetés a következő: ha a processzoros feladat megoldható d időn belül, akkor a ládapakolási feladat megoldható m darab d-méretű ládával.

Mit értünk a ládapakolási feladat esetébe duál-approximáción? Képzeljünk el egy olyan helyzetet, amikor a ládák mérete nem teljesen fix, kicsit kilóghatunk, de a célfüggvény értékét szeretnénk pontosan eltalálni: (például gondolhatunk egy vállalatra, ahol a dolgozók valamekkora túlterhelése megengedett, de fontos, hogy a lehető legkevesebb számú alkalmazottat kelljen alkalmazni.) A célunk a következő: tegyük fel, hogy megoldható a ládapakolási feladat m darab ládával. Olyan elrendezést szeretnénk találni, ahol csak m darab ládát használunk, viszont megengdjük, hogy a ládában található tárgyak összmérete a láda méretének $(1+\epsilon)$-szorosa legyen.

Világos, hogy ha ilyen algoritmus van, akkor bináris kereséssel tudunk keresni $(1+\delta)$-közelítő algoritmust a processzoros feladatra: d kezdő minimális értékének

válasszuk a $\max(\max t_i, \frac{\sum t_i}{m})$ értéket. Kezdő maximális értéknek pedig választhatjuk ennek a kétszeresét, ugyanis egy [2]-ban bizonyított állítás szerint a processzoros feladat optimális megoldásának befejezési ideje nem haladhatja meg ezt az értéket.) Adott d értékre megoldjuk az ϵ-duál-approximációs feladatot d-méretű ládákkal. Ha m ládába be tudunk pakolni, akkor ezek szerint $(1+\epsilon)d$ idő alatt megoldható a processzoros feladat, így most legyen az új d $\frac{d+d_{min}}{2}$, és d_{max}-ot pedig változtassuk erre a d-re, amivel ezt a feladatot elvégeztük. Ha pedig a ládapakolási feladatot nem tudjuk m ládával megoldani, vagy valamilyen más okból kiderül, hogy d idő alatt biztosan nem lehet megoldani a processzoros feladatot, akkor d_{min}-t változtassuk d-re és legyen az új $d := \frac{d+d_{max}}{2}$.

Hogyan oldjuk meg tehát az ϵ-duál-approximációs feladatot? Az alábbi megfigyelést érdemes tennünk:

Tegyük fel, hogy az ϵd-nél nagyobb feladatokhoz találtunk ϵ-duál-approximációs megoldást maximum m darab processzorral. Az ϵd-nél nem nagyobb feladatokat pakoljuk be mohón: tetszőleges sorrendbe állítva kezdjük őket bepakolni: bármelyik olyan ládába tehetjük az éppen soron következőt, amelyikben a tárgyak összmérete nem haladja meg d-t. Ha nincs ilyen láda, akkor nyissunk új ládát.

14. Állítás. *Ha ezzel a mohó algoritmussal sikerül minden feladatot bepakolnunk maximum m ládába, akkor találtunk egy ϵ-duál-approximációs megoldást a ládapakolási feladatra maximum m ládával. Ha pedig m-nél több ládára van szükség, abból az következik, hogy a processzoros feladat nem oldható meg d idő alatt.*

Biz. Az állítás első fele triviális: a kis feladatok bepakolása után is teljesülni fog, hogy minden ládában az összméret $\leq (1+\epsilon)d$. Az állítás második fele abból fakad, hogy ha egy kis feladatot miatt meg kell nyitnunk egy $m+1$. ládát, az azt jelenti, hogy m láda mindegyikében legalább d összméret található. Azaz $\frac{\sum t_i}{m} > d$, és ebből következik, hogy a processzoros feladat nem oldható meg d időn belül. □

Most már csak azt a feladatot kell megoldanunk, hogy az ϵd-nél nagyobb tárgyakat

bepakoljuk m darab $(1+\epsilon)d$-méretű ládába. Ezt a feladatot dinamikus programozással fogjuk megoldani.

Bontsuk fel az $(\epsilon d, d]$ intervallumot $s = \left\lfloor \frac{1}{\epsilon^2} \right\rfloor$ részre: $(\epsilon d = l_1, l_2]$, $(l_2, l_3]$, ... $(l_s, l_{s+1}]$. Vezessük be az x^j s-hosszú vektorokat: ennek i. koordinátája jelölje azt, hogy a j. ládában hány datab olyan tárgy van, amelynek mérete az $(l_i, l_{i+1}]$ intervallumba esik. Egy ládapakolás akkor lesz megengedett, ha $\sum_i x_i^j l_i \leq d \ \forall j$, hiszen egy ládába nem kerülhet $\left\lfloor \frac{1}{\epsilon} \right\rfloor$-nál több tárgy, azaz, ha $\sum_i x_i^j l_i \leq 1 \ \forall j$, akkor a tárgyak összmérete egyik ládában sem fogja meghaladni $(1+\epsilon)d$-t.

Jelöljük $y(b_1, b_2, ..., b_s)$-sel a megengedett megoldáshoz szükséges ládák minimális számát abban az esetben, ha b_i darab tárgy mérete esik az $(l_i, l_{i+1}]$ intervallumba. Ekkor a következő módon tudjuk alkalmazni a dinamikus programozást: aszerint választunk szét eseteket, hogy az első ládába melyik intervallumból hány feladatot teszünk:

$$y(b_1, b_2, ..., b_s) = 1 + \min(y(b_1 - x_1^1, b_2 - x_2^1, ..., b_n - x_n^1) \,|\sum x_i^1 \leq d)$$

A dinamikus programozáshoz n^s értéket kell kiszámolnunk. Egy érték kiszámításához pedig $\left\lfloor \frac{1}{\epsilon} \right\rfloor^s$ értéket kell összehasonlítanunk. Tehát az algoritmus futásideje $O(\frac{n}{\epsilon}^{\left\lfloor \frac{1}{\epsilon^2} \right\rfloor})$.

4.2. A kétpolcos algoritmus bemutatása

Rátérünk a moldable feladat megoldására. Tegyük fel, hogy tudjuk, hogy az optimális beosztás nem tart tovább, mint d. A kétpolcos algoritmusban [6] az előző fejezetben bemutatott duál-approximációs algoritmushoz hasonlóan most egy olyan beosztást fogunk keresni, ahol pontosan m darab processzort használhatunk, viszont megengedjük, hogy a processzorok d-nél tovább dolgozzanak: olyan megoldást fogunk keresni, ahol a feladatok befejezési ideje legfeljebb $1,5d$. Ezt úgy képzeljük el, hogy kialakítunk két polcot, az nagy d-hosszú, a kicsi $\frac{d}{2}$-hosszú. Azokat a feladatokat, amiket nem sikerül

bepakolnunk a nagy polcba, a kis polcban fogjuk elhelyeni. Az algoritmus abban is hasonlít az előző algoritmushoz, hogy a kis feladatokat először nem veszi számításba, csak utólag pakolja be őket. Az algoritmus $O(nm)$ idő alatt talál egy $1,5d$ idő alatt befejeződő beosztást.

Természetesen a gyakorlatban nem ismerjük C_{max}^{opt}-ot, ezért az előző algoritmushoz hasonlóan bináris keresést végzünk: egy adott d-vel végrehajtjuk az algoritmust, és megnézzük, hogy sikerült-e m processzorra bepakolni a feladatokat. Ha nem, akkor felfelé lépünk d-vel, ha igen, akkor lefelé.

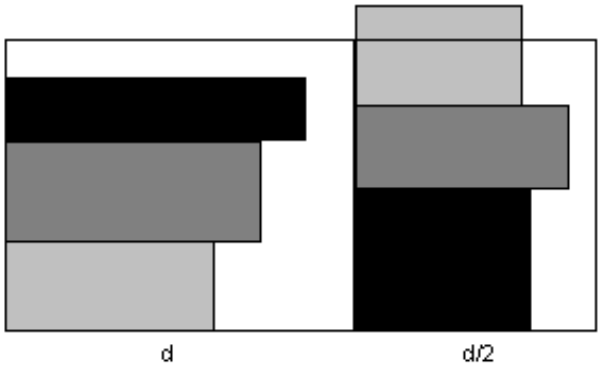
Tegyük fel, hogy d idő alatt meg lehet oldani a feladatokat.Az algoritmus a következő 4 lépésben talál egy $1,5d$ alatt befejeződő megoldást:

1. Kiválogatjuk azokat a feladatokat, amik 1 processzoron is legfeljebb $\frac{d}{2}$ idő alatt elkészülnek. Ezeket nevezük kicsi feladatoknak: velük az algoritmus során nem fogunk foglalkozni egészen az utolsó lépésig.

2. Készítünk két polcot: az első polc d-hosszúságú lesz, a másik polc $d/2$-hosszúságú. Ebbe a két polcba szeretnénk bepakolni a feladatokat. Első körben, ha az i. feladatról úgy döntünk, hogy az első polcra tesszük, akkor ott a lehető legkevesebb processzorra: $\gamma_i(d)$ darabra helyezzük el. Ha a másodikra tesszük, akkor pedig $\gamma_i(d/2)$ processzorra kerül. Egyelőre csak arra fogunk figyelni, hogy az első polcban ne használjunk több, mint p processzort, a másik polc esetleg túltelített lesz: ezt majd a 3. lépésben fogjuk kijavítani. Ezen a kritériumon belül viszont azt a szétosztást fogjuk választani, ahol az összmunka a lehető legkisebb. Ennek a szétosztásnak a megtalálásához egy hátizsákfeladatot fogunk megoldani:

$$\min \sum x_i \gamma_i(d) t_i(\gamma_i(d)) + \sum (1 - x_i) \gamma_i(d/2) t_i(\gamma_i(d/2))$$
$$\sum x_i \gamma_i(d) \leq p$$
$$x \in 0, 1$$

   A 8. ábrán láthatunk egy olyan beosztást, amit ebben a lépésben kapunk.



8. ábra. A hátizsák feladat megoldása utáni állapot

3. Átalakítjuk a polcrendszert. Most már 3 részleg lesz: m_3 darab processzorból képezünk egy harmadik polcot, ennek a hossza $3/2d$ lesz. A maradék processzorokon pedig fenntartjuk az eredeti két polcot. (Kezdetben m_3=0.)

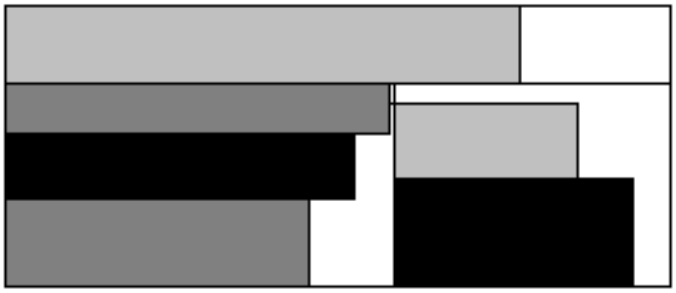
   Addig végezzük a következő 3 műveletet, amíg lehetséges.

   A) Ha találunk egy feladatot az első polcban, ami több, mint 1 processzoron fut, és nem tart tovább, mint $3/4d$, akkor eggyel kevesebb processzoron futtatjuk, és betesszük a 3. polcba.

   B) Ha találunk két feladatot, amelyeket 1-1 processzoron futtatunk, és egyik sem tart tovább, mint $3/4d$, akkor beteszük őket egymás mellé egy processzorra a 3. polcba.

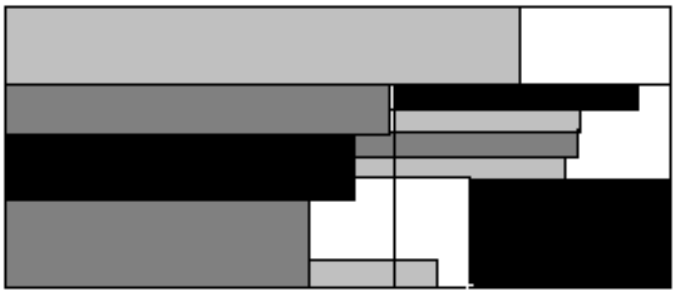
   C) Ha az első polcban k darab üres processzor van, és a második polcban találunk egy i. feladatot, amelyre $\gamma_i(3/2d) \leq k$, akkor azt elhelyezzük $\gamma_i(3/2d)$ processzoron, és annak függvényében, hogy így d idő alatt befejeződik-e vagy sem, betesszük az első vagy a harmadik polcba.

   Figyeljük meg, hogy a 3. polcban minden processzor legalább d ideig dolgozik. Be fogjuk bizonyítani, hogy a 3. lépés végére olyan beosztást kapunk, ahol a második polcban lévő feladatok nem használnak több processzort, mint $m - m_3$. A 9. ábrán látható egy olyan beosztás, amelyet ezután a lépés után kapunk.



9. ábra. A 3. lépés utáni állapot

4. A kis feladatokat is elhelyezzük a polcokon: az a második polcban minden feladatot kitolunk úgy, hogy pontosan $3/2d$-nél végződjön. Az első polcon lévő feladatok végpontja és a második polcon lévő feladatok kezdőpontja közötti területekre tesszük be a kis feladatokat: a kis feladatokat tetszőleges sorrendbe állítjuk, és mindig arra a processzorra teszünk be egy feladatot, ahol éppen a legnagyobb a hely. A 4. lépés utáni állapotot a 10. ábrán szemléltetjük.



10. ábra. A kis feladatok bepakolása utáni állapot

A következő kérdéseket kell megválaszolnunk:

1. Miért fogjuk tudni elhelyezni az összes kis feladatot a negyedik lépésben?

2. Hogyan valósítjuk meg O(nm) idő alatt a hátizsák-feladat megoldását?

3. Miért lehet mindig megtenni az A) lépést? (Azaz nem lehet-e, hogy túl hosszú lesz egy feladat a harmadik polcon?)

4. Mennyi ideig tart a harmadik lépés?

5. Miért jutunk el a 3. lépés végére egy olyan állapotba, ahol $m - m_3 \geq m_2$?

15. Állítás. *A kis feladatokat el tudjuk helyezni a 4. lépésben.*

Biz. Figyeljük meg, hogy milyen optimális célfüggvényértéket találhattunk a hátizsák-feladat megoldása után! Jelöljük a hátizsák-feladat optimális célfüggvényértékét $\dot{w}$-vel, a kisfeladatok egy processzoron való végzésével kapott összmunkát pedig W_s-sel.

16. Állítás. $\dot{w} \leq md - W_s$

Biz.

Feltettük, hogy d idő alatt el lehet végezni a feladatokat. Vegyünk egy ilyen beosztást. Itt a kis feladatok összmunkája legalább W_s. Azaz a többi feladat összmunkája legfeljebb $md - W_s$. Eszerint a beosztás szerint konstruáljunk egy szétosztást a két polcba. Azokat a feladatokat, amik itt hosszabbak, mint $1/2d$ tegyük az első polcba (ezek közül nem lehetett kettő egy processzoron, így a polcos beosztásban is be fognak férni az első polcba). A maradék feladatokat pedig tegyük a második polcba. Az így kapott összmunka csak csökkenhetett, azaz a hátizsákfeladatnak létezik olyan megoldása, ahol a célfüggvényérték legfeljebb $md - W_s$.

□

Látható, hogy a harmadik lépésben az összmunka értéke csak csökkenhet, mivel nincs olyan feladat, ami több processzorra kerül, mint eredetileg.

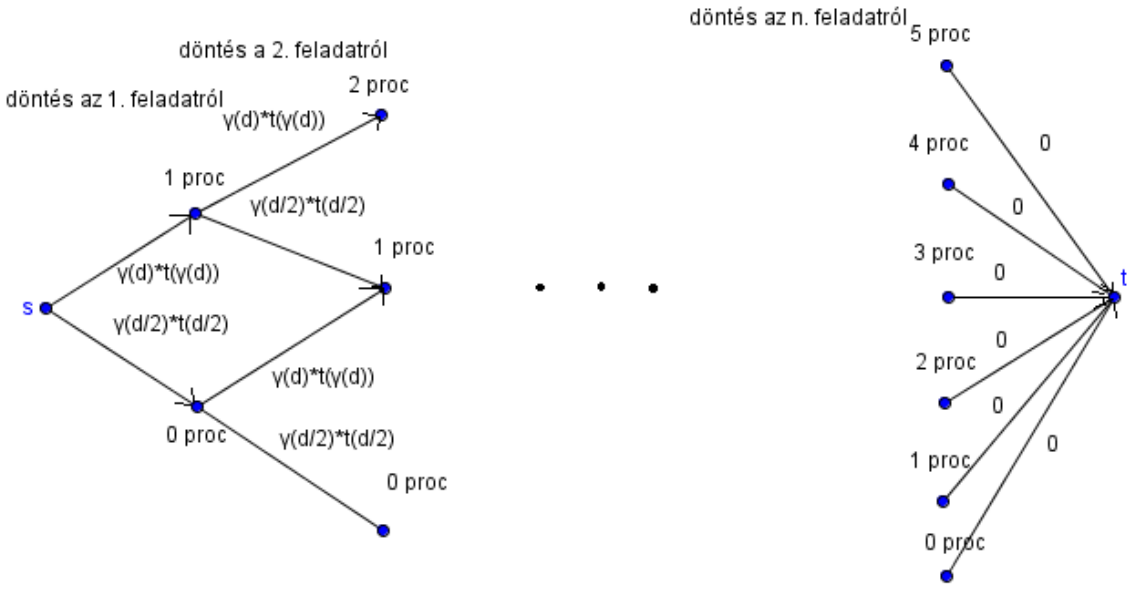
Ezek után már egyszerűen adódik, hogy be tudjuk pakolni a kis feladatokat: akkor történhetne csak baj, ha már minden polcon kevesebb, mint $d/2$ üres hely lenne, és még maradna bepakolandó feladat. Ez azt jelenti, hogy ekkor már a polcokon lévő összmunka nagyobb, mint md. Másrészt az előző állítás szerint, mielőtt megkezdtük

a kisfeladatok bepakolását, a polcokon lévő összmunka kisebb volt, mint $md - W_s$, így mivel még most még nem pakoltunk be minden kis feladatot, a polcokon lévő összmunka kisebb, mint md. Ezzel ellentmondásra jutottunk, azaz a kis feladatok bepakolása mindenképpen sikerül. □

17. Állítás. *A hátizsák-feladat megoldható $O(nm)$ időn belül.*

Biz.Képezzük a következő, a hátizsák-feladat döntéseit ábrázoló irányított gráfot: a csúcsok azt fogják jelenteni, hogy az aktuális állapotban hány processzort használunk már a nagy polcban. Az élek pedig azt jelentik, hogy mennyi munkát jelent, ha az illető feladatot a nagy, illetve a kis polcba tesszük be. (A 11. ábrán egy ilyen ötprocesszoros rendszerhez tartozó gráfot látunk.)

Ha egy döntéssel olyan állapotba jutnánk, hogy már több, mint p processzort fel-



11. ábra. A hátizsák-feladat döntéseit ábrázoló gráf

használnánk a nagy polcban, akkor azt nem jelenítjük meg a gráfban. Azaz egy olyan aciklikus irányított gráfot kapunk, amelyben kevesebb, mint $2nm$ él van. Egy aciklikus

irányított gráfban a legrövidebb utat $O(e)$ időben meg lehet találni. Azaz a hátizsák-feladatot meg tudjuk oldani $O(nm)$ időben. □

18. Állítás. *Az A) lépésben olyan feladat kerül a 3. polcba, amelynek befejési ideje* $\leq \frac{3}{2}d$.

Biz.A 8. állítás szerint, ha egy feladat rövidebb, mint $\frac{3d}{4}$, akkor egy processzorral kevesebben futtatva rövidebb, mint $\frac{3d}{2}$. □

19. Állítás. *A 3. lépésben* $O(n)$ *ideig tart.*

Biz. Az világos, hogy az A), B), C) műveleteket összesen kevesebb, mint $2n$-szer végezhetjük el, hiszen, ha az első polcban volt egy feladat, és azon változtatunk, akkor ezzel a harmadik polcba kerül, ahol már nem változtatunk a feladatokon. Ha pedig egy feladat a második polcban van, akkor azon maximum kétszer változtatunk: egyszer, amikor betesszük az első polcba, és egyszer, amikor onnan is áthelyezzük. □

Jelöljük a második polcban használt processzorok számát m_2-vel!

3. Tétel. *Ha az A), B), C) lépések közül egyik sem hajtható végre, akkor* $m_2 \leq m - m_3$

Biz.

Tegyük fel, hogy az A), B), C) lépések közül egyik sem hajtható végre, és mégis $m_2 > m - m_3$. Úgy fogunk ellentmondásra jutni, hogy ki fog derülni, hogy ebből az következne, hogy ebben a pillanatban a feladatok összmunkája nagyobb, mint $md - W_s$.

Jelöljük az első, második és harmadk polcon lévő feladatok összmunkáját rendre W_1, W_2 és W_3-mal!

20. Állítás. *Ha az A), B), C) lépések közül egyik sem hajtható végre, és* $m_2 > m - m_3$, *akkor* $W_2 \geq \frac{d}{4}(m - m_3 + 1)$

Biz.

Egy feladat a második polcban legalább $d/4$ ideig kell, hogy tartson, mivel egy processzoron legalább $d/2$ ideig tart (különben kis feladat lenne), ha pedig nem egy

processzoron futna egy $d/4$-nél rövidebb program, akkor eggyel kevesebbre is tehettük volna, hiszen, akkor is $d/2$-nél rövidebb idő alatt futna. Ebből, és a $m_2 > m - m_3$ egyenlőtlenségből következik, hogy $W_2 \geq \frac{d}{4}(m - m_3 + 1)$.

□

21. Állítás. *Ha az A), B), C) lépések közül egyik sem hajtható végre, akkor* $W_1 > 3/4d(m - m_3 - q)$.

Biz.

Ha sem az A), sem a B) műveletek nem hajthatók végre, akkor maximum egy olyan processzor lehet, ami nem üres, és a rajta lévő feladat nem tart hosszabb ideig, mint $3/4d$. Ha nincs ilyen processzor, akkor automatikusan adódik az állítás.

Tegyük fel, hogy van egy ilyen i. feladat.

22. Állítás. *Nem lehet, hogy ez az egyetlen feladat az első polcon.*

Biz. Jelöljük az első polcban lévő üres processzorok számát q-val! *1. eset: q=0*

Ekkor $m-m_3 = 1$, és teljesülnek az alábbi egyenlőtlenségek. $W_1 > \frac{d}{2}$ és $W_2 > \frac{d}{2}$ (mivel a második polcban legalább két processzort használunk). $W_1+W_2 > d = (m-m_3)d$. Ez nem lehet, mivel $W \leq md$ így $W_1 + W_2 = W - W_3 \leq (m - m_3)d$.

2. eset: q>0 A második polcon lévő egyik feladat sem tehető át az első polcbeli üres polcokra. Ha csak egy feladat van a második polcban, arra felírható a $W_2 > \frac{3}{2}dq$ egyenlőtlenség, hiszen ez a feladat q processzoron $\frac{3}{2}d$-nél hosszabb ideig futna, azaz q processzoron a munkája nagyobb lenne, mint $\frac{3}{2}dq$, most pedig q-nál több pocesszoron fut a második polcban. Az első polcban csak 1 processzor dologozik, azaz $q = m-m_3-1$. Így $(m - m_3)d \geq W_1 + W_2 > \frac{d}{2} + \frac{3}{2}dq = d/2 + \frac{3}{2}d(m - m_3 - 1) = \frac{3(p-m_3)-2}{2}d$. Ebből pedig következik, hogy $p-m_3 \geq (3(p-m_3)-2)$, azaz $m-m_3 \leq 1$. Ez ellentmondásban van azzal, hogy feltettük, hogy $q > 0$. □

Tehát kell lennie legalább még egy feladatnak az első polcban. Tudjuk, hogy az i. feladatot nem lehetett utána tenni, azaz a kettejük futási idejének összege nagyobb, mint $3d/2$. Az i. feladat csak 1 processzoron fut, azaz elmondható, hogy a nemüres processzorokon lévő futási idők átlaga legalább $3/4d$, amiből következik az állítás. □

Most már rátérhetünk a tétel bizonyítására: az eddigi lemmák alapján $W_1 + W_2 \geq \frac{d}{4}(m - m_3 + 1) + 3/4d(p - m_3 - q)$. Ha $q = 0$, akkor készen vagyunk, mert akkor $W_1 + W_2 \geq \frac{d}{4}(m - m_3 + 1) + 3/4d(m - m_3) > (m - m_3)d$, ami lehetetlen.

Vizsgáljuk most a $q > 0$ esetet:

Egyfelől

$$W_2 \leq (m - m_3)d - W_1 < (m - m_3)d - 3/4d(m - m_3 - q) = 1/4(m - m_3)d + 3/4qd$$

Másfelől adunk W_2-re egy alsó becslést is. Jelölje a második polcban található feladatok számát k! Használjuk azt a becslést, ami abból következik, hogy egyik feladat sem pakolható át az első polcba az üres polcokra:

$$W_2 > kq\tfrac{3}{2}d!$$

A két W_2-re vonatkozó egyenlőtlenségből: $(m - m_3) + 3q > 6qk$, azaz $(m - m_3) > 6qk - 3q$, amiből

$$m - m_3 > 3q(2k - 1)$$

Felírhatunk egy másik alsó becslést is W_2-re: használhatjuk alsó becslésként azt a munkát, amit akkor kapnánk, ha eggyel kevesebb processzoron futtatnánk a feladatokat.

$$W_2 > \sum((\gamma_i(d/2) - 1)\tfrac{d}{2} \mid \text{az } i. \text{ feladat a második polcban van}) =$$
$$(\sum(\gamma_i(\tfrac{d}{2}) \mid \text{az } i. \text{ feladat a második polcban van}) - k)\tfrac{d}{2} \geq (m - m_3 + 1 - k)\tfrac{d}{2}$$

Ebből és a $W_2 \leq 1/4(m - m_3)d + 3/4qd$ egyenlőtlenségből: $(m - m_3) + 3q > 2(m - m_3 + 1 - k)$ azaz

$$m - m_3 < 3q + 2k - 2,$$

A két $m - m_3$-ra adott becslés alapján: $3q(2k-1) < 3q + 2k - 2$, azaz $3q(2k-2) < 2k - 2$, tehát $2(3q-1)(k-1) < 0$, ami nem lehet, mivel $k > 0$ és $q > 0$. Azaz a feltevésünk hamis.

□

4. Tétel. *A kétpolcos algoritmus* $(1,5+\epsilon)$*-approximációs megoldást ad a moldable ütemezési problémára.*

Biz. Bináris kereséssel megkeressük ϵ-pontossággal azt a minimális d-t, amire az algoritmus be tudja pakolni a feladatokat (ezt jelöljük d_{alg}-gal). Ezt a következő módon tesszük meg: világos, hogy ha a d kisebb, mint $2/3 \sum t_i(1)$, akkor nem lehet bepakolni a feladatokat a két polcba. Ha pedig $d = \sum t_i(p)$, akkor már a nagyobbik polcba is be tudjuk pakolni a feladatokat (mindegyiket p processzorra tesszük). E két érték között bináris kereséssel kereshetjük d_{alg}-t pontossággal. Az aktuális d-vel megoldjuk a hátizsákfeladatot. Ellenőrizzük, hogy egyáltalán van-e megoldása a hátizsákfeladatnak, és ha van, akkor az összmunka nem haladja-e meg $dm - W_s$-t (ez kell ahhoz, hogy a kis feladatokat be tudjuk pakolni). Ha nincs megoldás, vagy túl nagy az összmunka, akkor fölfelé mozdítjuk d-t. Ha pedig tovább tudunk haladni, akkor megnézzük, hogy az átcsoportosítások után sikerül-e a második polcba bepakolni a feladatokat. Ha marad ki feladat, akkor megint felfelé mozdítjuk d-t, ha pedig sikerült bepakolni őket, akkor lefelé. □

5. Az algoritmus tesztelése

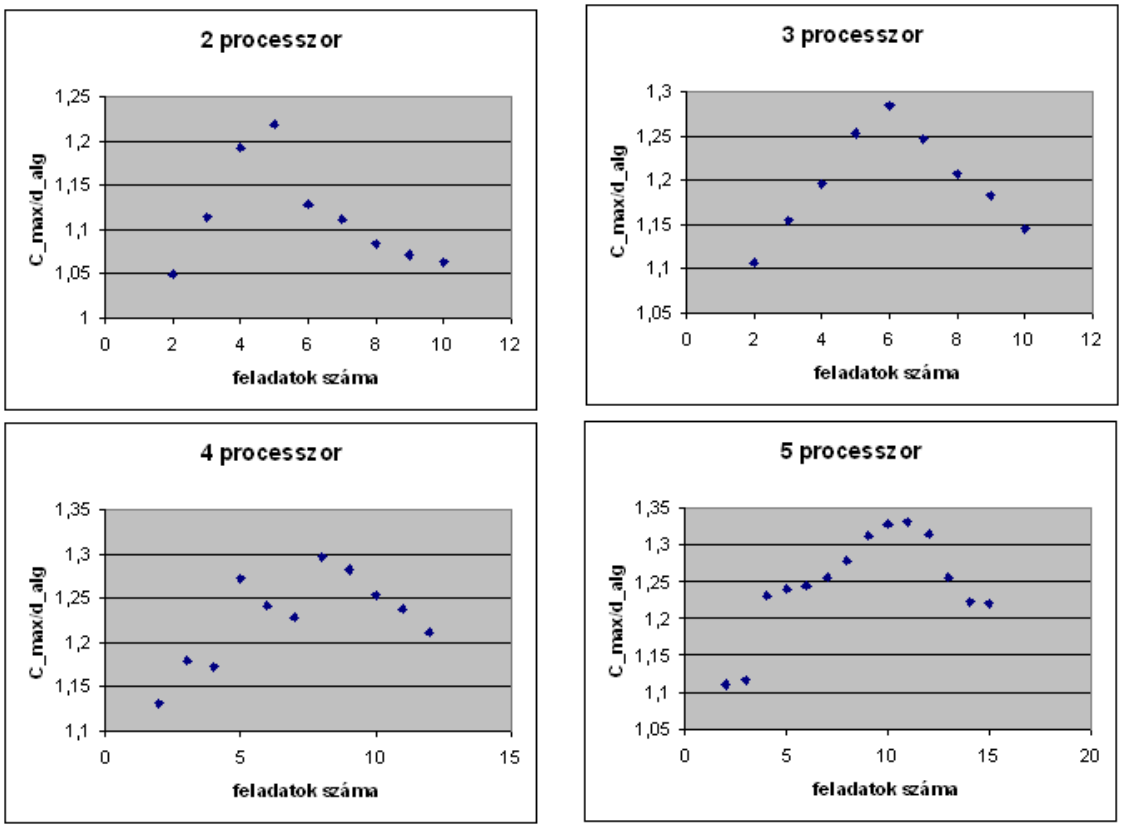
A kétpolcos algoritmust a fent bemutatott bináris kereséssel implementáltuk ($\epsilon = 0,1$). Azzal a természetes módosítással éltünk, hogy az algoritmus végén a második polcban található feladatokat előre toljuk, amennyire lehet.

Az algoritmus implementálása után lehetőségünk nyílt az eredmények tesztelésére. Arra a kérdésre szeretnénk választ kapni, hogy az algoritmus által adott befejezési idő mennyire közelíti meg az optimális befejezési időt. Erre kérdésre persze nem tudunk pontos választ adni, mivel nem ismerjük az optimális befejezési időt. Becsülni tudjuk az arányt a következő módon: $d_{alg} - \epsilon$ egy alsó becslés C_{max}^{opt}-ra, hiszen azért ezt a d-t választotta az algoritmus, mert ha nála ϵ-nal kisebbnek választotta a nagyobbik polc méretét, akkor már nem tudta beütemezni a feladatokat a két polcba. Megvizsgálhatjuk tehát, hogy hogyan viselkedik a $\frac{C_{max}^{alg}}{d_{alg}}$ arány, és ez egy felső becslést ad $\frac{C_{max}^{alg}}{C_{max}^{opt}}$-ra az ϵ hibatagtól eltekintve.

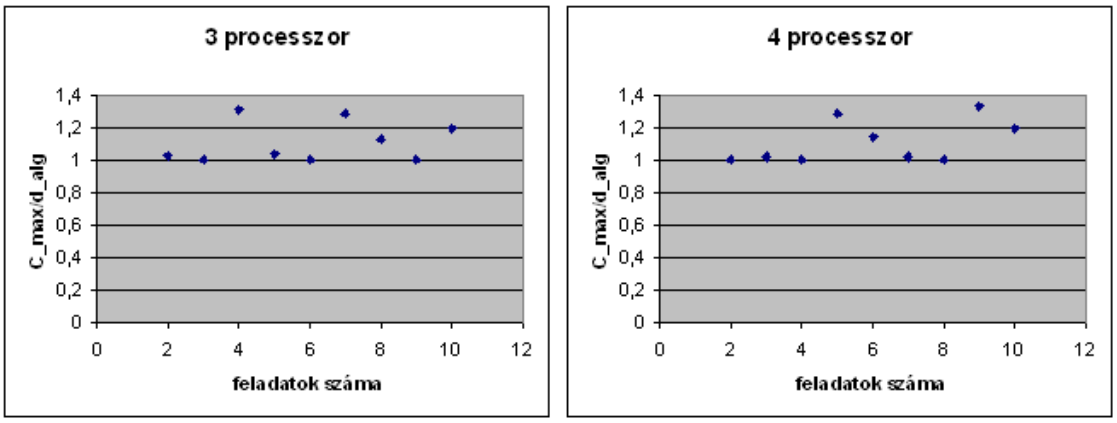
Az $2 \leq m \leq 5$ eseteket vizsgáltuk a feladatok különböző száma mellett. Egy adott (m, n) párra 50 kísérletet végeztünk el. A feladatok méretét úgy generáltuk, hogy $t_i(1)$-et egyenletes eloszlás szerint sorsoltuk, majd $t_i(j)$-t olyan határok között sorsoltuk egyenletes eloszlás szerint, amik a monotonitási feltételek teljesülését garantálják a $t_i(k)$ $k < j$ feladatokkal szemben. Minden vizsgált (m, n) párra az 50 kísérletben kapott $\frac{C_{max}^{alg}}{d_{alg}}$ arány átlagát jegyeztük fel. A tesztelés eredményei a függelékben találhatóak.

A kapott eredmények (12. ábra) alapján azt látjuk, hogy amíg $m > \frac{n}{2}$, addig a $\frac{C_{max}^{alg}}{d_{alg}}$ arány a feladatok számának növelésével monoton növő tendenciát mutat, majd onnantól kezdve, hogy $m = \frac{n}{2}$, monoton csökkenőt.

Felmerül a kérdés, hogy tudunk-e olyan feladatosztályt mutatni, amelyekre nem ez a tendencia figyelhető meg. A válasz igen: ha minden feladatot egyformának választuk, akkor az $\frac{C_{max}^{alg}}{d_{alg}}$ arány viselkedésében valamiféle periodicitás figyelhető meg (13. ábra).



12. ábra. $\frac{C_{max}^{alg}}{d_{alg}}$ arány a feladatok számának függvényében 2,3,4 és 5 processzor mellett



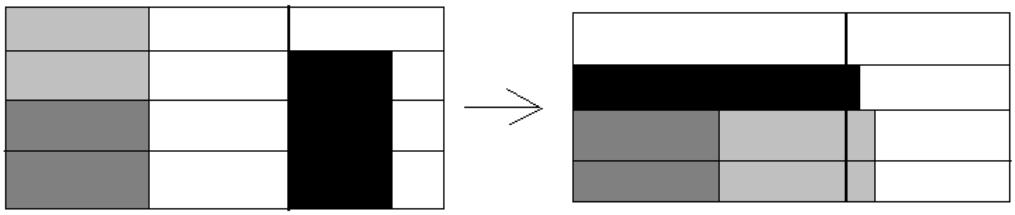
13. ábra. $\frac{C_{maxalg}}{d_{alg}}$ arány egyforma feladatok esetén

Ezt a viselkedést meg is tudjuk könnyen indokolni. Az indoklást a $p = 4$ esetre mutatjuk be. Az ϵ hibatagtól eltekintünk.

$n = 2$: az algoritmus d_{alg}-ot t_2-nek választja, és mindkét feladatot 2 processzoron

futtatja, így $d_{alg} = C_{opt}^{max}$.

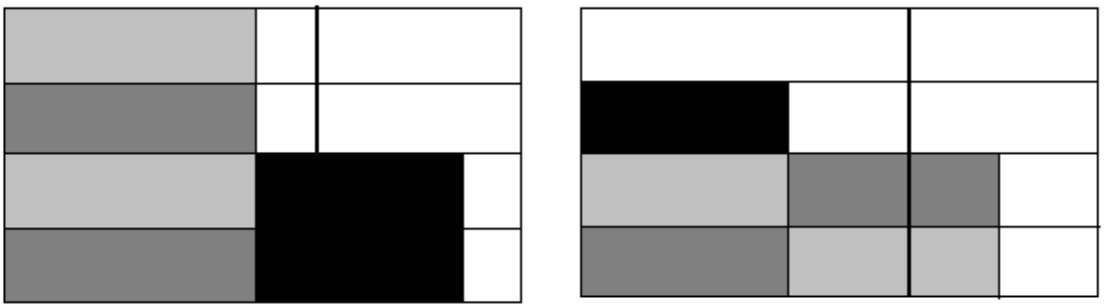
$n = 3$: két eset lehetséges: lehetséges, hogy az algoritmus d_{alg}-ot t_1-nek választja (pl. $t_1 = 10$, $t_2 = 8$, $t_3 = \frac{16}{3}$, $t_4 = 4$: itt ha $d = 10 - \epsilon$, akkor a nagy polcba bekerül két feladat két processzoron, a kis polcba pedig egy feladadat négy processzoron, de ezek összmunkája $4 \cdot 8 + 4 \cdot 4 > 4(10 - \epsilon)$, vagyis $d_{alg} > 10 - \epsilon$). A feladatok lehetnek olyanok is (pl. $t_1 = 10$, $t_2 = 6$, $t_3 = 4$, $t_4 = 3$), hogy $d_{alg} < t_1$. Ilyenkor biztos, hogy a nagy polcban a feladatok $\frac{3}{4}d_{alg}$-nál nem hosszabbak, ugyanis ekkor a három feladaton végzett összmunka meghaladná $4d_{alg}$-ot. Így ilyenkor a 14. ábrán látható beosztást kapjuk.



14. ábra. Egyforma feladatok, $n = 3$, $m = 4$, $d_{alg} < t_1$

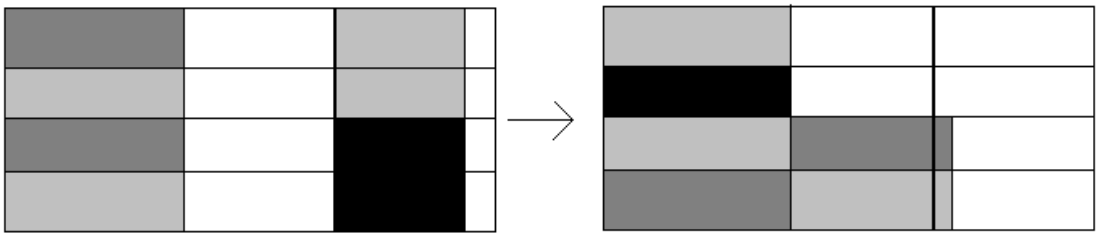
$n = 4$: az algoritmus d_{alg}-ot t_1-nek választja, és minden feladatot 1 processzoron futtat, így $d_{alg} = C_{opt}^{alg}$.

$n = 5$: most $t_1 < d_{alg}$. Aszerint, hogy $t_1 > \frac{3}{4}d_{alg}$ vagy $t_1 \leq \frac{3}{4}d_{alg}$ a 15. ábrán látható kétféle beosztást kaphatjuk. (Látható, hogy mindkét beosztásban a $\frac{C_{max}^{alg}}{d_{alg}}$ arány nagy lehet.)



15. ábra. Egyforma feladatok, $n = 3$, $m = 4$, $d_{alg} < t_1$

$n = 6$: ahhoz hogy a hátizsákfeladat megoldása után az összterületek mérete ne haladja meg $4d_{alg}$-ot, $4t_1 + 2 \cdot 2t_2 \leq 4d_{alg}$-nek kell teljesülnie. Azaz $d_{alg} \geq t_1 + t_2$. $d_{alg} = t_1 + t_2$ viszont meg is felel. Ekkor $t_1 \leq \frac{3}{4} d_{alg}$ teljesül. Azaz $\frac{C_{max}^{alg}}{d_{alg}} = \frac{2t_1}{t_1+t_2}$

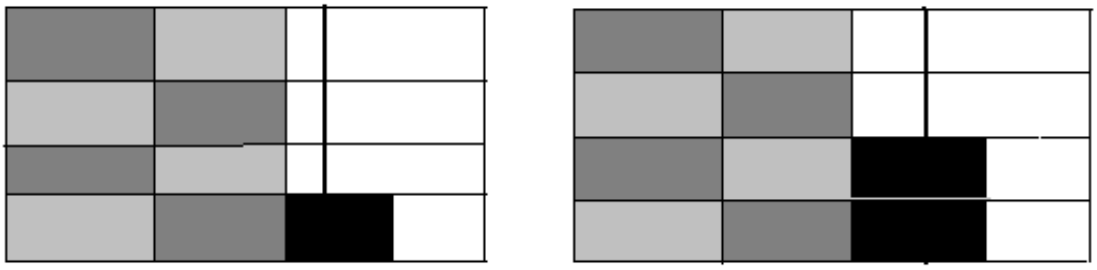


16. ábra. Egyforma feladatok esete, $n = 6$, $m = 4$

$n = 7$: az előző esethez hasonló, csak most az algoritmus még nagyobb, $2t_1$-hez közeli d_{alg}-ot talál, így $\frac{C_{max}^{alg}}{d_{alg}} = \frac{2t_1}{t_1+t_2}$ kicsi.

$n = 8$: $d_{alg} = 2t_1 = C_{max}^{alg}$

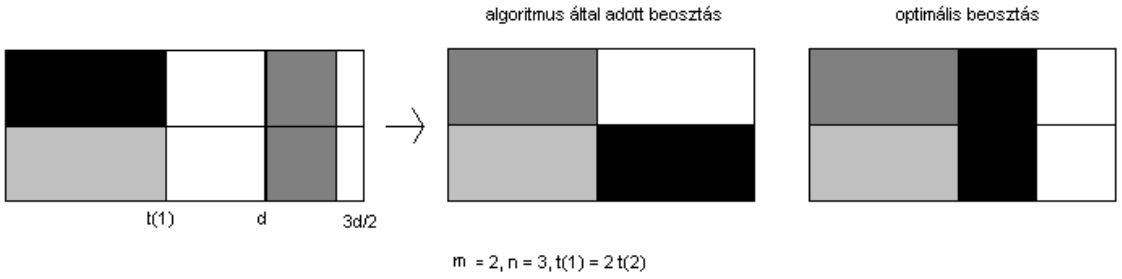
$n > 8$: $d_{alg} > 2t_1$, azaz minden feladat kicsi, így a 17. ábrán látható módon történik a beosztás. Így, ha $n = 4k$, akkor $d_{alg} = C_{opt}^{max}$, ha pedig $n = 4k + l$, akkor $d_a lg = kt_1 + \frac{4}{l} t_1$, azaz $\frac{C_{max}^{alg}}{d_{alg}} = \frac{(k+1)t_1}{kt_1 + \frac{4}{l} t_1}$



17. ábra. Egyforma feladatok esete, $n = 9$, $n = 10$

6. Egyforma feladatok

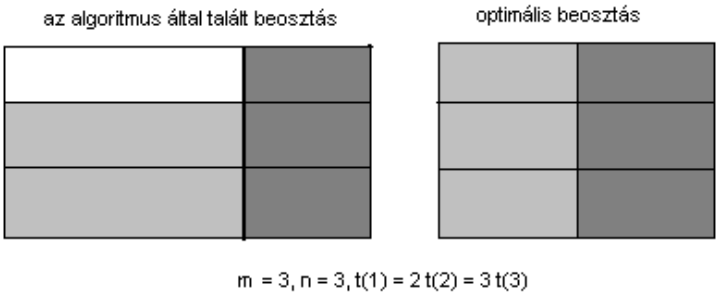
Az előző fejezetben láttuk, hogy az az eset, amikor minden feladat egyforma, teljesen máshogyan viselkedik, mint az általános eset. Felmerül a kérdés, hogy mennyivel könnyebb a feladat ezzel a megkötéssel, azaz tudunk-e gyorsabb és hatékonyabb algoritmust erre az esetre, mint a kétpolcos algoritmus. Mutatunk két példát arra, amikor a kétpolcos algoritmus által adott befejezési idő nagyban eltér az optimálistól: A 18. ábrán látható a példában $n > m$, és az algoritmus $\frac{4}{3}$-szor rosszabb eredményt ad az optimálisnál. 3 feladatot szeretnénk 2 processzorra beosztani, $t(1) = 2t(2)$. Az algoritmus a bináris kereséssel a $d = \frac{3}{2}t(1)$-et találja meg (ez a legkisebb olyan érték, amire a hátizsák-feladat által adott beosztásban az összmunka $\leq dm$). Utána elhelyez két feladatot a nagy polcba, a harmadikat pedig 2 processzoron a kis polcba. Ezek után a két 1 processzoron lévő feladatot egymás után teszi (hiszen $t(1) < \frac{3}{4}\frac{3}{2}t(1)$), a harmadik feladatot pedig átteszi az üres processzorra.



18. ábra. A feladatok egyformák, $n > m$, a kétpolcos algoritmus célfügvényértéke $\frac{4}{3}$-szor rosszabb az optimumnál.

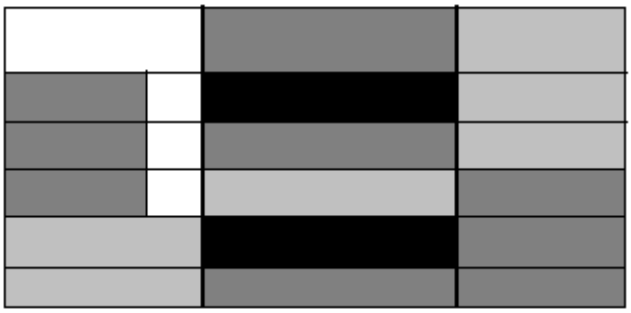
A 19. ábrán látható példában pedig $m > n$, és az algoritmus 3/2-szer rosszabb eredményt ad, mint az optimális: itt $t(1) = 2t(2) = 3t(3)$. Az algoritmus bináris kereséssel a $d_{alg} = t(2)$ értéket találja meg.

[1]-ben mutatnak egy algoritmust, ami az egyforma feladatok esetére konstans időben talál egy olyan beosztást, ami az $n > m$ esetben $\frac{6}{5}$-approximációs, egyébként



19. ábra. A feladatok egyformák, $m > n$, a kétpolcos algoritmus célfügvényértéke $\frac{3}{2}$-szor rosszabb az optimumnál.

$\frac{5}{4}$-aproximációs. Ez az algoritmus egy úgynevezett PPS (phase-by-phase) beosztást fog adni, ami a következőt jelenti: a feladatokat fázisokba csoportosítjuk, ahol az egy fázison belül szereplő feladatok kezdési ideőpontja megegyezik. Egy fázis nem kezdődhet hamarabb, mint ahol az előtte kezdődő végződik. A 20. ábrán láthatunk egy ilyen PPS beosztást.



20. ábra. PPS-beosztás

Az lesz a célunk, hogy ezek között a PPS-beosztások között találjunk egy optimálisat.

23. Állítás. *Az egyforma feladatoknak létezik olyan optimális PPS-beosztása, ahol egy fázison belül minden feladat ugyanannyi processzoron fut.*

Biz. Tekintsünk egy optimális PPS-beosztást. Minden fázison belül tekintsük azt a feladatot, amelyik a legkevesebb processzorra van beosztva. Minden más feladatot

osszunk be annyi processzorra, amennyi processzorra ez a feladat van beosztva. A fázisok hossza nem változik, azaz a befejezési idő sem, így találtunk egy optimális megoldást, ahol egy fázison belül minden feladat ugyanannyi processzoron fut. □

24. Állítás. *$O(m^3)$ időben tudunk optimális PPS-beosztást találni.*

Biz. Olyan optimális PPS-beosztást fogunk keresni, ahol egy-egy fázison belül minden feladat ugyanannyi processzorra lesz beosztva. Jelöljük x_i-vel azt, hogy hány darab olyan fázis szerepel a beosztásban, ahol a feladatok i processzoron futnak.

Érdemes megfigyelnünk, hogy leszűkíthetjük az optimumkeresést azokra a beosztásokra, amelyekre igaz, hogy $x_i < i$ ha $i \geq 2$ hiszen, ha van i darab olyan fázisunk, ahol a feladatok i processzoron futnak, akkor ezeket a fázisokat helyettesíthetjük egy olyan fázissal, ahol minden feladat 1 processzoron fut. A munkára vonatkozó monotonitás következtében az így kapott fázis nem hosszabb, mint az eredeti i darab fázis összhossza. Számoljuk meg, hogy maximum hány olyan feladat lehet, ami az általunk keresett optimális PPS-beosztásban nem egy processzorra kerül:

$$\sum_{2\leq i\leq m} x_i \left\lfloor \frac{m}{i} \right\rfloor \leq \sum_{2\leq i\leq m} (i-1) \left\lfloor \frac{m}{i} \right\rfloor \leq (m-1)m$$

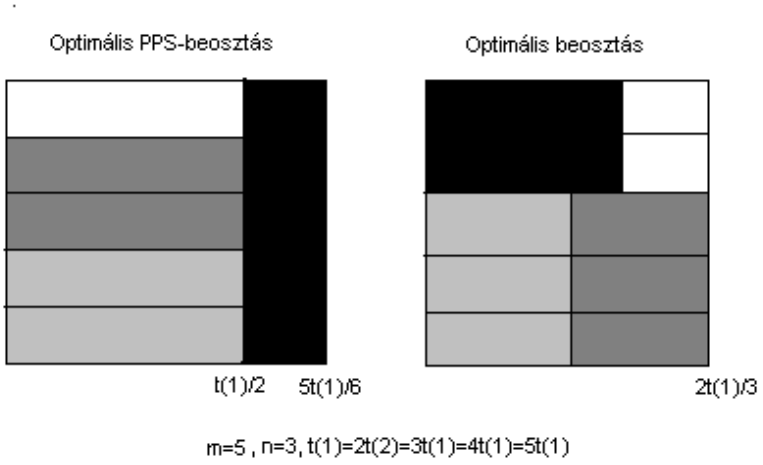
Azaz $m \left\lceil \frac{n-(m-1)m}{m} \right\rceil$ feladatot beosztunk 1-1 processzorra, és a maradék feladatokat kell még fázisokba osztanunk.

Így most is egy PPS feladatot kell megoldanunk, csak most kevesebb, mint m^2 feladatra. Ezt pedig dinamikus programozással fogjuk megoldani: a k. lépésben az első k feladathoz keresünk egy optimális PPS-beosztást, és kiszámoljuk ezeket az optimális befejezési időket (jelöljük ezeket $C^{alg}_{max}(k)$-val). Amikor az első k lépéshez tartozó optimális beosztást keressük, akkor az alapján bontjuk szét a lehetőségeket, hogy az utolsó fázisban hány feladat szerepel. Azaz a következő módon tudjuk kiszámolni $C^{alg}_{max}(k)$-t:

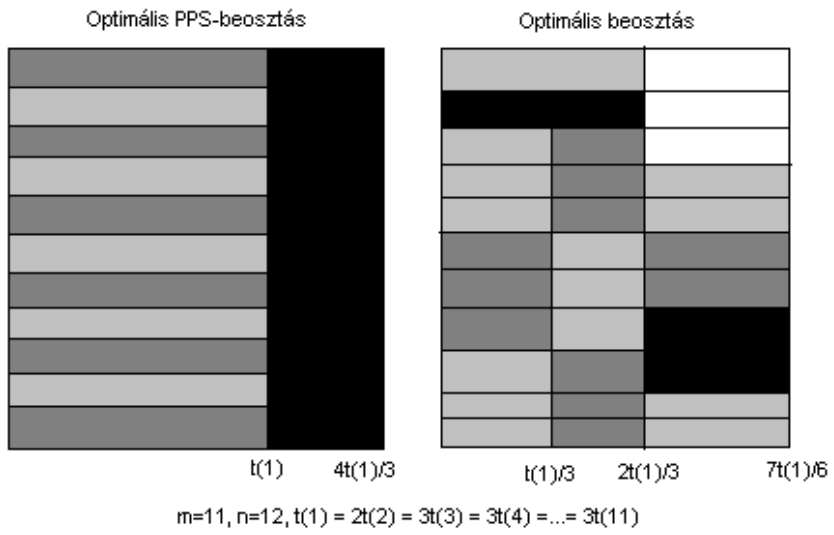
$$C^{alg}_{max}(k) = \min_{1\leq i\leq m} \left(C^{alg}_{max}(k-i) + t\lfloor \frac{m}{i} \rfloor \right)$$

Ez a dinamikus program m^2 feladatra $O(m^3)$ ideig tart. □[1]-ben mutatnak egy példát arra, hogy ha egy algoritmussal optimális PPS-beosztást keresünk, akkor ez az

algoritmus nem lehet jobb, mint $\frac{5}{4}$-approximációs (a 21. ábrán látunk egy 3 feladatból és 5 processzorból álló ütemezési feladatot ($t(1) = 2t(2) = 3t(3)$), ahol $C_{max}^{alg} = \frac{5}{4}C_{max}^{opt}$. Arra is adnak példát (22. ábra: 12 feladat, 11 processzor $t(1) = 2 = t(2) = 3t(3) = 3t(4) = ...3t(5)$), hogy ha feltesszük, hogy $n > m$, akkor sem lehet jobb az algoritmus, mint $\frac{8}{7}$-approximációs.



21. ábra. Az optimális PPS beosztás befejezési ideje $\frac{5}{4}$-szerese az optimumnak



22. ábra. $n > m$, az optimális PPS beosztás befejezési ideje $\frac{8}{7}$-szerese az optimumnak

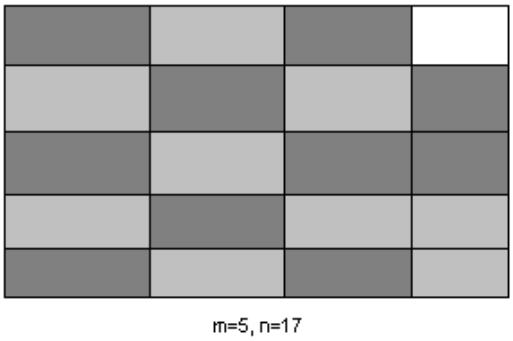
[1]-ben nem is erről az optimális PPS-beosztásról bizonyítják be az approximációs tulajdonságot, hanem az m és az n arányától függően mutatnak egy konstans időben

számolható PPS-beosztást, ami nem feltétlenül az optimális PPS-beosztás, viszont bebizonyítható róla, hogy ha a befejezési idő maximum $\frac{5}{4}$-szerese az optimálisnak, ha pedig feltesszük, hogy $n > p$, akkor maximum $\frac{6}{5}$-szöröse.

Bemutatjuk, hogy az $n > p$ esetben hogyan számolható ez a PPS-beosztás. A $p \leq n$ eset is hasonlóan működik:

1. eset: $n > 3m$

$\left\lfloor \frac{n}{m} \right\rfloor$ feladatot 1-1 processzorra teszünk, a maradékot pedig $\left\lfloor \frac{m}{n - m\left\lfloor \frac{n}{m} \right\rfloor} \right\rfloor$ processzoron futtatjuk. (Erre a beosztásra látunk példát a 23. ábrán.)



23. ábra. 1. eset

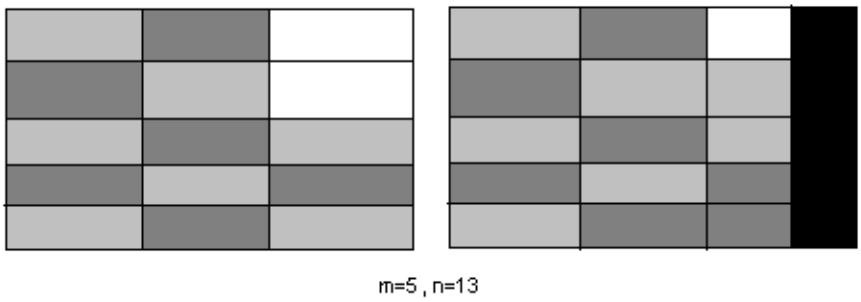
2. eset: $\left\lfloor \frac{5}{2} \right\rfloor < n \leq 3m$

Attól függően, hogy melyik gyorsabb: vagy minden feladatot 1 processzorra teszünk, vagy $2m$ feladatot 1 processzorra, $\lfloor \frac{m}{2} \rfloor$ feladatot 2 processzorra teszünk, a maradékot pedig $\lfloor \frac{m}{n - \lfloor \frac{5m}{2} \rfloor} \rfloor$ processzoron futtatjuk. (A két lehetséges beosztásra látunk példát a 24. ábrán.)

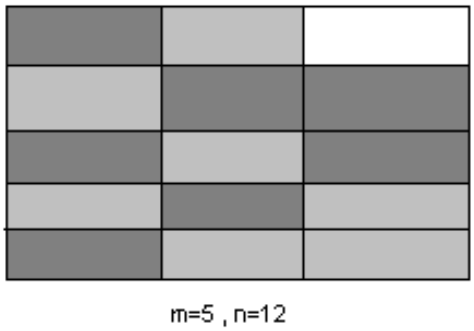
3. eset: $2m < n \leq \lfloor \frac{5}{2}m \rfloor$

$2m$ feladatot 1-1 processzorra teszünk, a maradékot pedig $\left\lfloor \frac{m}{n-2m} \right\rfloor$ processzoron futtatjuk. (Erre a beosztásra látunk pédát a 25. ábrán.)

4. eset: $\left\lfloor \frac{3m}{2} \right\rfloor < n \leq 2m$

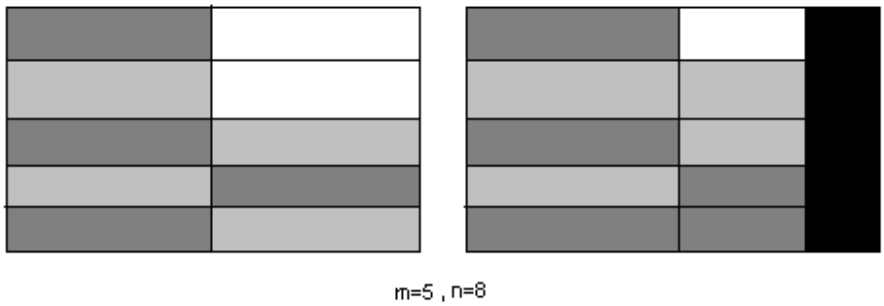


24. ábra. 2.eset



25. ábra. 3. eset

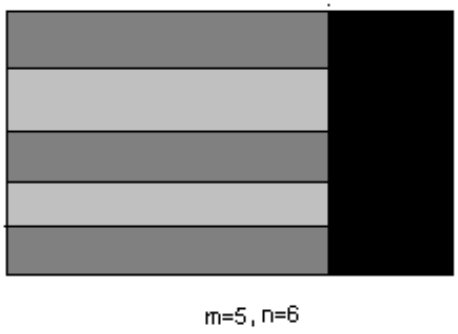
Attól függően, hogy melyik gyorsabb: vagy minden feladatot 1 processzorra teszünk, vagy p feladatot 1 processzorra, $\left\lfloor \frac{m}{2} \right\rfloor$ feladatot 2 processzorra teszünk, a maradékot pedig $\left\lfloor \frac{m}{n-\left\lfloor \frac{3m}{2} \right\rfloor} \right\rfloor$ processzoron futtatjuk.(A két lehetséges beosztásra látunk pédát a 26. ábrán.)



26. ábra. 4. eset

5. eset: $m < n \leq \left\lfloor \frac{3m}{2} \right\rfloor$

m feladatot 1-1 processzorra teszünk, a maradékot pedig $\left\lfloor \frac{m}{n-m} \right\rfloor$ processzoron futtatjuk. (Erre a beosztásra látunk pédát a 27. ábrán.)



27. ábra. 5. eset

25. Állítás. *A fenti algoritmus* $\frac{6}{5}$*-approximációs. [1]*

7. Határidő utáni munkavégzés minimalizálása

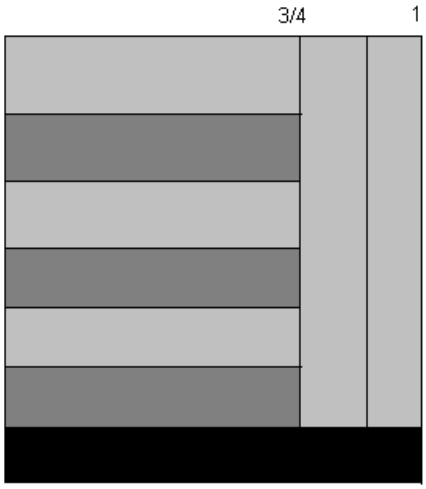
Előzőleg láttuk, hogy a kétpolcos algoritmus 1,5-approximációs, ha a cél a munka befejezésének minimalizálása. Ebben a fejezetben azt fogjuk vizsgálni, hogy az algoritmus milyen eredményt ad, ha a célfüggvényt megváltoztatjuk: tekintük C_{max}^{opt}-ot mint határidőt, és legyen a célunk az, hogy ezen határidőn túl a gépek összmunkája minimális legyen! Jelöljük W_{kint}^{alg}-tel a határidőn túl végzett összmunkát. A $\frac{W_{kint}^{alg}}{C_{max}^{opt}m}$ arányról szeretnénk valamit mondani. Világos, hogy $\frac{W_{kint}^{alg}}{C_{max}^{opt}m} \leq \frac{1}{2}$.
Mutatunk egy példát, ahol $\frac{W_{kint}^{alg}}{C_{max}^{opt}m} = \frac{2}{7}$: legyen 9 feladatunk és 7 processzorunk:

$$t_1(j) = 1 \ \forall j$$

$$t_i(1) = \tfrac{3}{4},\ t_i(2) = \tfrac{3}{8},\ t_i(3) = \tfrac{1}{4},\ t_i(4) = \tfrac{3}{16},$$

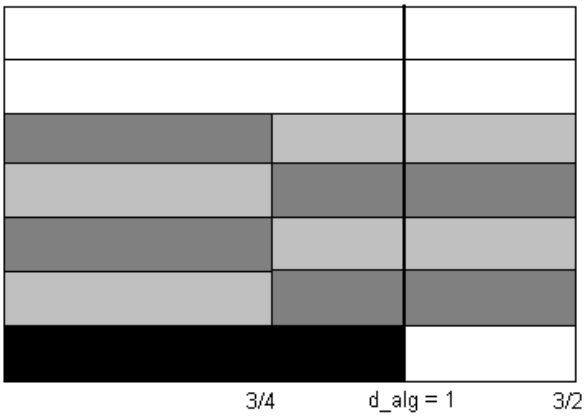
$$t_i(5) = \tfrac{3}{20},\ t_i(6) = \tfrac{1}{8},\ t_i(7) = \tfrac{1}{8}\ \forall i \in 2...9$$

A 28. ábrán mutatunk egy beosztást, amire a befejezési idő 1. Tehát $C_{max}^{opt} = 1$ (ennél kevesebb nem lehet az első feladat miatt.)



28. ábra. Optimális beosztás

Könnyen látható, hogy az algoritmus a 29. ábrán található beosztást adja megoldásnak ($d_{alg} = 1$, hiszen kisebb nem lehet az első feladat miatt, ha pedig $d = 1$, akkor a hátizsák-feladat megoldása után az összmunka dp lesz.) Így ebben a példában $\frac{W_{kint}^{alg}}{C_{max}^{opt}m} = \frac{2}{7}$.

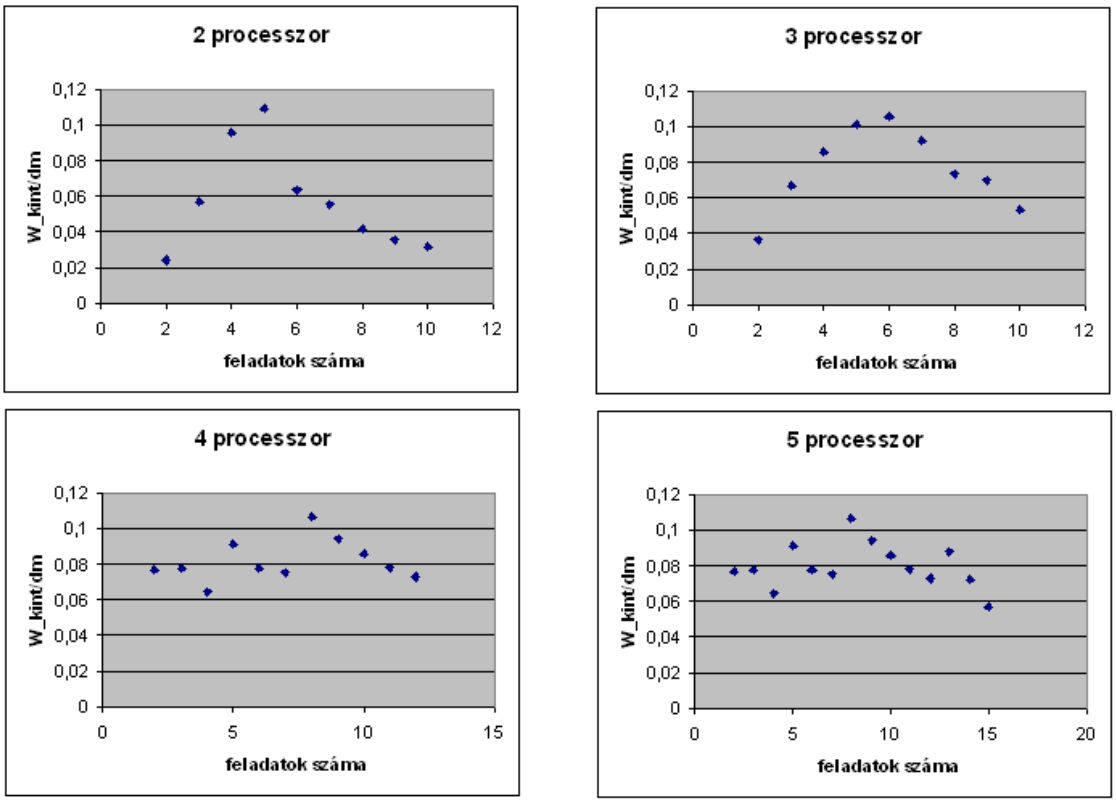


29. ábra. A kétpolcos algoritmus által adott beosztás

Ezzel tehát azt láttuk be, hogy $\max \frac{W_{kint}^{alg}}{C_{max}^{opt} m} \geq \frac{2}{7}$. $\max \frac{W_{kint}^{alg}}{C_{max}^{opt} m} \leq \frac{1}{2}$-nél jobb felső korlátot nem sikerült találnunk.

$\frac{W_{kint}^{alg}}{C_{max}^{opt} m}$ átlagos viselkedését numerikusan nem tudjuk vizsgálni, mivel C_{max}^{opt}-ot nem tudjuk hatékonyan kiszámítani. Ehelyett vizsgálhatjuk, hogy ahhoz a d_{alg}-hez képest, hogyan viselkedik a kilógó terület (jelöljük $W_{kint}^{alg}(d_{alg})$-vel). A $\frac{W_{kint}^{alg}(d_{alg})}{d_{alg} m}$ felső becslést ad $\frac{W_{kint}^{alg}}{C_{max}^{opt} m}$-re.

A tesztelés eredményei megtalálhatóak a függelékben. Látható, hogy a $\frac{W_{kint}^{alg}}{C_{max}^{opt} m}$ arány viselkedése hasonló a $\frac{C_{max}^{alg}}{d_{alg}}$ arány viselkedéséhez. (30. ábra)



30. ábra. A $\frac{C_{max}^{alg}}{d_{alg}}$ arány viselkedése n függvényében az $m = 2, 3, 4, 5$ esetekben

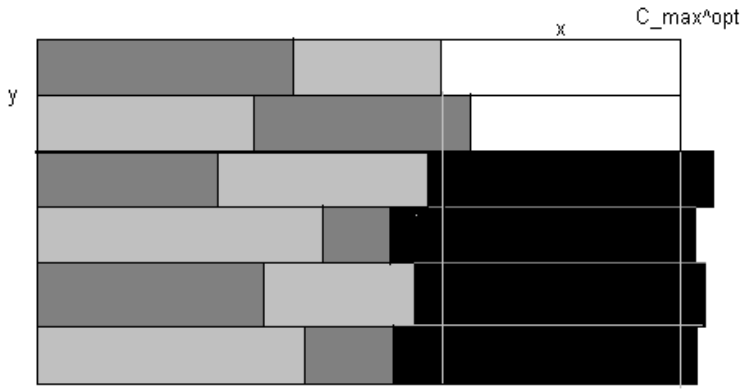
7.1. Egy speciális eset vizsgálata

Ebben a részben bemutatunk egy speciális esetet, amelyben a $\frac{W_{kint}^{alg}}{C_{max}^{alg}p}$ arányról többet tudunk mondani.

Ehhez először tekintsük az 5.1-ben szereplő egyszerűbb feladatot (minden feladat csak 1 processzoron futtatható). Jelölje W_{kint}^{alg} egy adott algoritmus által adott megoldásban a C_{max}^{opt} határidőn kívüli összmunkát.

26. Állítás. *Tegyük fel, hogy* $\frac{\sum t_i}{m} \geq 2 \max t_i$. *Ekkor ha tetszőleges sorrendben tesszük be a feladatokat, mindig arra a processzorra, amelyiken a legkevesebb ideig folyik a munka, akkor* $\frac{W_{kint}^{alg}}{C_{max}^{opt}m} \leq \frac{1}{8}$. *(*$O(mn)$ *futásidő)*

Biz. Az állításban szereplő feltétel miatt $C_{max}^{opt} \leq t_i \; \forall i$. Rendezzük a 31. ábrán látható



31. ábra. A 25. állítás bizonyításában szereplő jelölések

módon a processzorokat: felül legyenek azok, amiken nem lógnak túl a feladatok C_{max}^{opt}-n. Jelöljük ezen processzorok számát y-nal! Jelöljük a felső y processzor közül a munkát leghamarabb befejező befejezési időpontja és a C_{max}^{opt} közti időt x-szel! Ez azt jelenti, hogy az alsó $m-y$ processzoron az utolsó előtti feladatok befejeződési ideje mind $C_{max}^{opt}-x$ előtt van. (Különben nem hozzájuk kerülnének a kilógó feladatok, mert lenne processzor, ami előbb befejeződött volna náluk.) Azaz a következő két felső becslést kapjuk $\frac{W_{kint}^{alg}}{mC_{max}^{opt}}$-re:

$$\frac{W_{kint}^{alg}}{C_{max}^{opt}m} \leq \frac{yx}{mC_{max}^{opt}}$$
$$\frac{W_{kint}^{alg}}{C_{max}^{opt}m} \leq \frac{\left(\frac{C_{max}^{opt}}{2}-x\right)(m-y)}{mC_{max}^{opt}}$$

Az első egyelőtlenség abból adódik, hogy tudjuk, hogy a feladatok összterülete $\leq C_{max}^{opt}m$. A második egyenlőtlenség pedig abból következik, hogy tudjuk, hogy a feladatok futási ideje $\leq \frac{C_{max}^{opt}}{2}$, azaz semelyik feladat sem lóghat túl $\frac{C_{max}^{opt}}{2}-x$-nél jobban.

Jelöljük $\frac{x}{C_{max}^{opt}}$-t x'-vel, $\frac{y}{m}$-t y'-vel! Ezt kapjuk:

$$\frac{W_{kint}^{alg}}{C_{max}^{opt}m} \leq x'y'$$
$$\frac{W_{kint}^{alg}}{C_{max}^{opt}m} \leq \left(\tfrac{1}{2}-x'\right)(1-y')$$

A legrosszabb eset becsléséhez meg kell oldanunk ezt a feladatot:

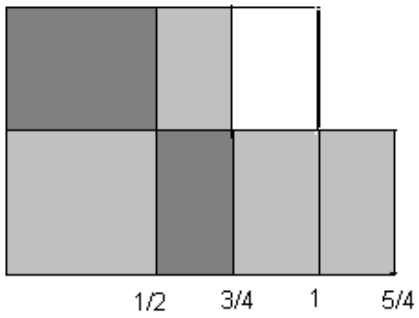
$$\max \min \left(x'y', \left(\tfrac{1}{2} - x'\right)(1 - y')\right)$$

$$0 \leq y' \leq 1$$

$$0 \leq x' \leq \tfrac{1}{2}$$

Könnyen látható, hogy ezen feladat optimuma $x' = \frac{1}{4}$, $y' = \frac{1}{2}$-nél van. Azaz, azt kaptuk, hogy $\frac{W_{kint}^{opt}}{C_{max}^{alg} m} \leq \frac{1}{8}$. □

Ez a becslés éles: a 32. ábrán látható példában teljesül is, hogy $\frac{W_{kint}}{dp} = \frac{1}{8}$ ($C_{max}^{opt} = 1$)



32. ábra. Példa $\frac{W_{kint}}{dp} = \frac{1}{8}$ teljesülésére

Látjuk, hogy ezen a példán a leghosszabb feladatok azok, amik végül kilógnak. Nézzük meg, mi történne, ha annyiban módosítanánk az algoritmust, hogy nagyság szerint csökkenő sorrendben helyeznénk be a kis feladatokat!

27. Állítás. *Ha az előző algoritmust úgy módosítjuk, hogy nagyság szerint csökkenő sorrendben helyezzük be a feladatokat, akkor* $\frac{W_{kint}^{opt}}{C_{max}^{alg} m} \leq \frac{1}{12}$. *($O(mn \log n)$ futásidő)*

Biz. Az állítás bizonyításához szükségünk lesz az alábbi észrevételre:

28. Állítás. *Ha egy ütemezési feladatra a 27. állításban szereplő algoritmus által adott* $\frac{W_{kint}^{opt}}{C_{max}^{alg} m}$ *arány maximális, akkor ebben a feladatban a határidőn túllógó feladatok hossza megegyezik.*

Biz. Vegyünk egy ütemezési feladatot, amire a $\frac{W_{kint}^{opt}}{C_{max}^{alg} m}$ arány maximális. Tegyük fel, hogy a határidőn túllógó feladatok hossza nem egyforma. Változtassuk minden túllógó

feladat méretét akkorára, mint a leghosszabb túllógó feladat mérete. Futassuk le most erre a feladatra az algoritmust. Azok a feladatok, amik eredetileg nem lógtak túl, ugyanoda kerülnek, mint az eredeti esetben (hiszen, továbbra is hosszabbak, mint a túllógó feladatok.) Azaz a túllógó feladatok ugyanakkor kezdődhetnek, mint az eredeti esetben. Így az új túllógó feladatok bepakolása után $\frac{W_{kint}^{opt}}{C_{max}^{alg}m}$ nő. Ez nem lehetséges, vagyis a feltevésünk hamis. □

Jelöljük a túllógó feladatok méretét z-vel, $\frac{z}{C_{max}^{opt}}$-ot pedig $z' - vel$. Ekkor, ugyanúgy, mint az előbb láttuk következik az alábbi két becslés:

$$\frac{W_{kint}^{alg}}{C_{max}^{opt}m} \leq \frac{yx}{C_{max}^{opt}m}$$
$$\frac{W_{kint}^{alg}}{C_{max}^{opt}m} \leq \frac{(z-x)(m-y)}{C_{max}^{opt}m}$$

Az előző számításhoz hasonlóan az adódik, hogy $\frac{W_{kint}^{alg}}{C_{max}^{opt}m}$ akkor maximális, ha $x' = \frac{z'}{2}$ és $y' = \frac{1}{2}$, és az összes processzor $C_{max}^{opt} - x$-ig töltött, majd a processzorok felénél még beteszünk egy-egy $C_{max}^{opt} - x$ időpontban kezdődő z hosszú munkát. Kérdés, hogy meg lehet-e oldani, hogy $C_{max}^{opt} - x$-ig kitöltsük a processzorokat úgy, hogy csak z-nél hosszabb feladatokat használhatunk. Azaz rögzített z mellett $\frac{W_{kint}^{alg}}{C_{max}^{opt}m} \leq \frac{z}{4}$, és $=$ akkor állhat fenn, ha ki tudjuk tölteni a $C_{max}^{opt} - x$ időt z-nél nagyobb feladatokkal.

1. eset: $z' > \frac{1}{3}$

Ekkor teljesül, hogy $t_i > C_{max}^{opt}$ $\forall i$. [4]-ben szereplő állítás szerint ekkor a fenti algoritmusra a befejezési idő megegyezik az optimális befejezési idővel, azaz $W_{kint}^{alg} = 0$.

2. eset: $z' \leq \frac{1}{3}$

Ebben az esetben $\frac{W_{kint}^{alg}}{C_{max}^{opt}m} \leq \frac{z}{4} \leq \frac{1}{12}$.

□

Nézzük meg, hogy a moldable ütemezési probléma esetén milyen állítások felelnek meg a 26. és 27. állításoknak!

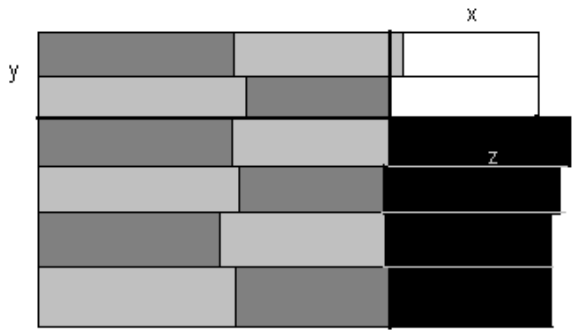
29. Állítás. *Ha a kétpolcos algoritmus által választott d_{alg}-hoz képest minden feladat kicsi (azaz $\frac{d_{alg}}{2} \leq t_i(1)$ $\forall i$), akkor $\frac{W_{kint}^{alg}}{C_{max}^{opt}m} \leq \frac{1}{8}$.*

Ez a becslés éles is: a 31. ábrán látható példa itt is működik. A bizonyítás ugyanúgy működik, mint a 27. állításé, azzal a különbséggel, hogy itt most azt tudjuk, hogy az algoritmus által adott beosztásban az összterület $\leq md \leq mC_{max}^{alg}$ a területek monotonitása következtében.

30. Állítás. *Ha a kétpolcos algoritmus által választott d_{alg}-hoz képest minden feladat kicsi, és a kis feladatokat méret szerinti csökkenő sorrendben pakoljuk be, akkor* $\frac{W_{kint}^{alg}}{C_{max}^{opt}m} \leq 0,10103$.

Biz. Most nem tehetjük fel, hogy a $\frac{W_{kint}^{alg}}{C_{max}^{opt}m}$-ot maximalizáló ütemezési feladatnál minden kilógó feladat hossza egyforma, ugyanis nem tudjuk a 28. állítás bizonyításához hasonlóan módosítani $t_i(1)$-eket, mivel itt figyelnünk kell arra, hogy ne sértsük meg a monotonitási feltételeket.

Jelöljük most z-vel a C_{max}^{opt} határidőn leghosszabban túllógó feladat méretét, és z' jelölje $\frac{z}{C_{max}^{opt}}$-ot. (33. ábra)



33. ábra. A 30. állítás bizonyításában szereplő jelölések

A 27. állításhoz hasonlóan most is fennáll az alábbi két egyenlőtlenség:

$$\frac{W_{kint}^{alg}}{C_{max}^{opt}m} \leq \frac{yx}{C_{max}^{opt}m}$$
$$\frac{W_{kint}^{alg}}{C_{max}^{opt}m} \leq \frac{(z-x)(m-y)}{C_{max}^{opt}m}$$

Vegyük észre, hogy $1 - x' \geq 2z'$, mivel a felső y processzoron legalább két feladat fut,

és ezek hosszabbak z-nél. Azaz a legrosszabb eset becsléséhez az alábbi feladatot kell megoldanunk:

$$\max \min(x'y', (z' - x')(1 - y'))$$
$$1 - x' \geq 2z'$$
$$0 \leq x', y', z' \leq 1$$

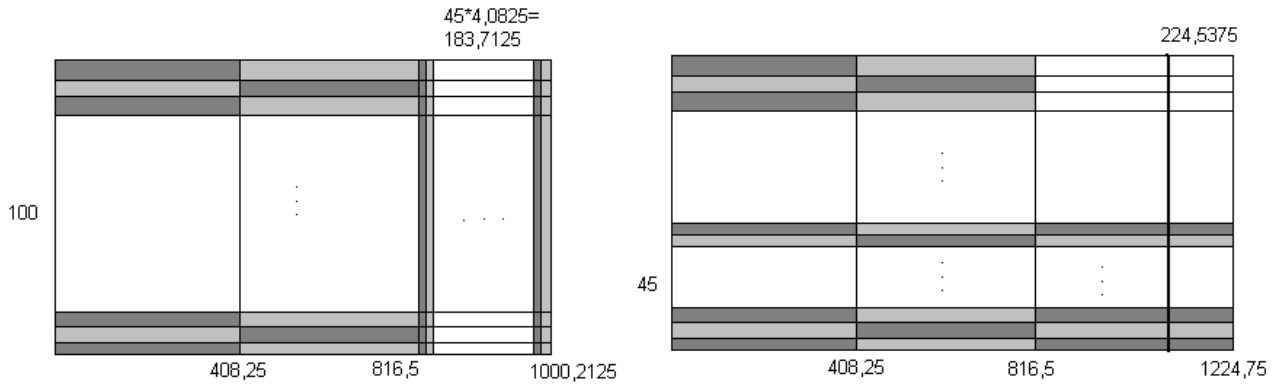
Ez ekvivalens az alábbi feladattal:

$$\max \min(x'y', (\tfrac{1-x'}{2} - x')(1 - y'))$$
$$0 \leq x', y' \leq 1$$

A fenti feladat maximuma ott vétetik fel, ahol $x'y' = (\frac{1-x'}{2} - x')(1 - y'))$. Azaz ekkor $y' = \frac{1-3x'}{1-x'}$. Így $x'y' = (\frac{1-x'}{2} - x')(1 - y')) = x\frac{1-3x}{1-x}$. Tehát az $f(x') = x'\frac{1-3x'}{1-x'}$ függvény maximumát kell megkeresni. $f'(x') = \frac{3x'^2-6x'+1}{(1-x')^2}$. A deriváltnak két zérushelye van, ebből egyre teljesül, hogy $0 \leq x', y' \leq 1$: $x' = 1 - \frac{\sqrt{2}}{\sqrt{3}}$. Ezen értéknél a függvénynek valóban maximumhelye van. Azaz a keresett maximumhely az $x' = 1 - \frac{\sqrt{2}}{\sqrt{3}}$, $y' = \frac{1-3x'}{1-x'}$-nél van. A maximumérték $\approx 1,0102$.

□

Mutatunk egy a 30. állítás bizonyításában szereplő $x' = 0,1835$, $y' = 0,5505$ értékeken alapuló példát (34. ábra), ahol $\frac{W_{kint}^{alg}}{C_{max}^{opt} m} > 0,101$. Legyen 100 processzorunk, és 245 feladatunk. Legyenek a feladatok olyanok, hogy bárhány processzoron is végezzük őket, ugyanannyi munkát jelentenek. $t_i(1) = 408,25 \forall i$, és $t_i(100) = 4,0825 \ \forall i$. Ekkor $C_{max}^{opt} = 2 \cdot 408,25 + 45 \cdot 4,0825 = 1000,2125$. A képtpolcos algoritmus által adott megoldásnál $W_{kint}^{alg} = (3 \cdot 408,25 - 1000,2125) \cdot 45 = 10104,1875$. Azaz $\frac{W_{kint}^{alg}}{m C_{max}^{opt}} \approx 0,10102$. A konstrukcióból látszik, hogy tetszőlegesen megtudjuk közelíteni a pontos felső becslést. Azaz a 30. állítás bizonyításában szereplő feladat optimumértéke éles becslést ad.



34. ábra. Példa $\frac{W_{kint}^{alg}}{C_{max}^{opt}m} > 0,101$ teljesülésére

8. Összefoglalás

A dolgozatban a moldable feladatra vonatkozó algoritmusok bemutatása után bemutattuk a kétpolcos algoritmus implementálásával kapott tesztelési eredményeinket. A $\frac{C_{max}^{alg}}{d_{alg}}$ és a C_{max}^{opt} határidőre vonatkozó $\frac{W_{kint}^{alg}(d)}{d_{alg}m}$ arány alapján tudtuk csoportosítani az ütemezési feladatokat: megfigyeltük, hogyha $m > 2n$, akkor a $\frac{C_{max}^{alg}}{d_{alg}}$ és a $\frac{W_{kint}^{alg}(d)}{d_{alg}m}$ arány n növelésével nő, utána pedig csökken. Megfigyeltük, hogyha a feladatok egyformák, akkor a $\frac{C_{max}^{alg}}{d_{alg}}$ arány periodikusan viselkedik.

Bemutattunk két eredményt a C_{max}^{opt} határidőre vonatkozó $\frac{W_{kint}^{alg}}{C_{max}^{opt}m}$ becslésével kapcsolatban arra az esetre nézve, ha a feladatokat 1 processzoron futtathatjuk, és feltesszük, hogy $t_i(1) \leq \frac{\sum t_i(1)}{m}$ $\forall i$, mutattunk egy $O(mn)$ futásidejű algoritmust, amire a C_{max}^{opt} határidőhőz képest $\frac{W_{kint}^{alg}}{C_{max}^{opt}m} \leq \frac{1}{8}$, és egy $O(mn \log n)$ futásidejű algoritmust, amelyre $\frac{W_{kint}^{alg}}{C_{max}^{opt}m} \leq \frac{1}{12}$. Az első algoritmusra vonatkozó $\frac{1}{8}$-os becslésről megmutattuk, hogy éles. A második becslés élességének bizonyítása, vagy élessé tevése nyitva maradt.

Ezeket a becsléseket fel tudtuk használni a moldable ütemezési feladatok azon speciális esetére, ha minden feladat kicsi, azaz $t_i(1) \leq \frac{d_{alg}}{2}$. Megmutattuk, hogy ekkor a kétpolcos algoritmus olyan megoldást ad, amelyre a C_{max}^{opt} határidőre vonatkozó $\frac{W_{kint}^{alg}}{C_{max}^{opt}m} \leq \frac{1}{8}$.

Erről a becslésről is megmutattuk, hogy éles. Ha a kétpolcos algoritmust úgy módosítjuk, hogy a kis feladatokat csökkenő sorrendben helyezzük be, (ezzel a lépésszám $O(mn \log n)$-re növekszik) akkor ugyanebben a speciális esetben $\frac{W_{kint}^{alg}}{C_{max}^{opt} m} \leq 0,10102$ becslést sikerült igazolnunk (a felső becslés egy irracionális kifejezés közelítő értéke). Megmutattuk, hogy a közelítő értékként kapott becslésünk pontosításával kapható felső becslésesekhez hogyan tudunk olyan példát konstruálni, hogy azt tetszőlegesen megközelítsük.

A fenti speciális esettel az a probléma, hogy a gyakorlatban nem jól használható, mivel csak úgy tudjuk eldönteni, hogy egy ütemezési feladat bele tartozik-e, ha lefuttatjuk a kétpolcos algoritmust, és megállapítjuk d_{alg}-ot. Ezért hasznos lenne olyan állításokat találni, amelyek segítségével d_{alg} kiszámítása nélkül is kiderülne bizonyos feladatosztályokról, hogy bele tartoznak-e a speciális esetbe.

Köszönetnyilvánítás

Nagyon köszönöm témavezetőmnek, Kis Tamásnak, hogy ötletek felvetésével, értékes tanácsokkal, együtt gondolkozással valamint dolgozatom átolvasásával segítette a munkámat. Ezen kívül köszönettel tartozom Dücső Márton barátomnak a programozásban nyújtott segítségéért.

9. Függelék: tesztelési eredmények

m	n	$\frac{C_{max}^{alg}}{d_{alg}}$	$\frac{W_{kint}^{alg}}{d_{alg}m}$	m	n	$\frac{C_{max}^{alg}}{d_{alg}}$	$\frac{W_{kint}^{alg}}{d_{alg}m}$
2	2	1,04854	0,02431	3	2	1,10711	0,0365
2	3	1,11316	0,05660	3	3	1,15487	0,0668
2	4	1,19129	0,0956	3	4	1,19506	0,0855
2	5	1,21836	0,1091	3	5	1,25275	0,1010
2	6	1,12809	0,0640	3	6	1,28433	0,1052
2	7	1,1109	0,0554	3	7	1,24674	0,0915
2	8	1,08391	0,0419	3	8	1,20623	0,0740
2	9	1,07184	0,03591	3	9	1,1836	0,0696
2	10	1,06367	0,0318	3	10	1,1456	0,0534
4	2	1,1319	0,0765	5	2	1,1103	0,0414
4	3	1,1790	0,0771	5	3	1,1161	0,0536
4	4	1,1731	0,0647	5	4	1,2295	0,0810
4	5	1,2726	0,0912	5	5	1,2402	0,1001
4	6	1,2406	0,0772	5	6	1,2429	0,0861
4	7	1,2283	0,0750	5	7	1,2544	0,0744
4	8	1,2960	0,1066	5	8	1,2776	0,0843
4	9	1,2815	0,0944	5	9	1,3115	0,0831
4	10	1,2529	0,0860	5	10	1,3263	0,0935
4	11	1,2379	0,0780	5	11	1,3302	0,0874
4	12	1,2108	0,0725	5	12	1,3144	0,1029
				5	13	1,2552	0,0880
				5	14	1,2231	0,0721
				5	15	1,2213	0,0570

Hivatkozások

[1] T. Decker, T. Lücking, B. Monien: A 5/4-approximation algorithm for scheduling identical malleable tasks. Theoretical Computer Science 361(2): 226-240. 2006.

[2] R.L Graham: Bounds for certain multiprocessing anomalies. Bell System Technical Journal 45 1966, 1563-1581.

[3] D.S. Hochbaum, D.B. Shmoys: Using dual approximation for scheduling problems: theoretical and practical results. Journal of the ACM, 34: 144-162. 1987

[4] Jordán Tibor: Ütemezéselmélet jegyzet, 2007.

[5] Walter Ludwig, Prasoon Tiwari: Scheduling mallaeble and nonmallaeble paralell tasks. SODA 1994: 167-176. 1994.

[6] G. Mounie, C. Rapine, D. Trystram: A 3/2-dual approximation algorithm for scheduling independent monotonic malleable tasks. SIAM J. on Computing, 37: 401412. 2000.

[7] D.K.D.B. Sleator: A 2.5 times optimal algorithm for packing in two dimensions. Information Processing Letters, vol. 10: 37-40. 1980.