SZAKDOLGOZAT

# Több processzoros ütemezés változtatható méretű feladatokkal

**Szilágyi Dániel**

**Témavezető:** Kis Tamás
egyetemi adjunktus
ELTE Matematika Intézet,
Operációkutatási Tanszék

**ELTE**
**2013**

# Tartalomjegyzék

# 1. Bevezető

A hétköznapi életben gyakran találkozunk olyan feladatokkal, amelyek elvégzéséhez mi dönthetjük el, hogy miként osztjuk be a munka elvégzéséhez szükséges erőforrásainkat, azaz gépeinket, energiánkat. Gondoljunk például egy költözésre. A bútorok és más ingóságok mozgatása összetett feladat, általában minél többen segítenek, annál könnyebben és gyorsabban folyik a munka. Leegyszerűsítve minden tárgyhoz hozzárendelhetünk különböző értékeket, annak függvényében, hogy milyen erőforrás használattal milyen gyorsan lehet mozgatni. Ezalatt azt értem, hogy egy bútor milyen sebességgel szállítható a kellő helyre, ha $1, 2, 3, ...$ ember mozgatja. Ez minden bútorra más és más értékeket adhat, a függvény azonban értelemszerűen csak egész helyeken értelmezett. Nem mindegy azonban, hogy a leegyszerűsített függvény konvex vagy konkáv. Például egy hatalmas ruhásszekrény cipelése egy embernek gyakorlatilag lehetetlen, kettőnek épp hogy lehetséges, hárman már jól haladnak vele, de 4 ember már mindenre elegendő, és ennyi a legalkalmasabb a munkára. Ebben az esetben a függvényünk konvex, érdemes tehát minél több (maximum 4) embert mozgósítani a feladathoz. Viszont mondjuk egy kisebb éjjeliszekrény cipelése már egy embernek is megoldható, 2 már kicsit felesleges. 3 vagy annál több pedig csak minimális segítséget nyújthat, így az ezt leíró erőforrás felhasználó függvény konkáv, minél kevesebb ember dolgozik, annál hatékonyabb a munkavégzésük.A dolgozat egy fontos témája a csak konvex és a csak konkáv végrehajtási idejű feladatok megoldása.

A feladatokat végző erőforrások legyenek processzorok. A processzorok száma legyen $m$, az elvégzendő feladatoké $n$, és minden feladatra van egy felső határa annak, hogy hány processzor dolgozhat rajta egyidejűleg. Minden időpillanatban változtatható, hogy a feladatokon hány processzor fusson, de egyszerre maximum $m$ darab működhet az összes feladaton, azaz egy processzor sem dolgozhat több feladaton. Ezt hívjuk változtatható processzorszámú feladatnak, röviden MPTS-nek. Ehhez képest az NPTS modellnél a feladatokhoz rendelt processzorszám, és ebből következően a feladatok végrehajtási ideje is előre adott. Párhuzamos ütemezést készítünk, azaz mind az $n$ feladat egymástól független, nincsenek előfeltételeik. A feladatokat elkezdhetjük a nulladik pillanatban. A feladatokat végző $m$ azonos processzor szintén független, és $n \leq m$; $n, m \in \mathbb{R}$ [4]. A processzorrendszeren párhuzamosan végzett feladatok lehető legjobb ütemezése, ahol semmilyen más tényező nem számít, mesterkélt probléma, a kapott eredmények azonban tökéletesen használhatóak a gyakorlatban, például számítógépes rendszerek működtetésénél. A processzorszám monoton csökkenő függvénye a végrehajtási idő nagysága . Az MPTS egy kiterjesztése az egyik legalapvetőbb ütemezési problémakörnek, a P|| $C_{max}$-nak. A P|| $C_{max}$-ban a $P$ a párhuzamosan működő erőforrások ütemezését jelöli, a $C_{max}$ pe-

dig azt, hogy a cél az összes feladat elvégzése során eltelt idő minimalizálása. Az egyetlen különbség az, hogy míg a P|| $C_{max}$-nál egy feladaton csak egy erőforrás dolgozhat, addig az MPTS-nél minden feladat egyszerre végezhető több erőforráson. Mindenféle rugalmasság, egyszerűsítés nagyban csökkentheti a végrehajtási időt, de jelentősen növeli a probléma megoldásának nehézségét. Néhány MPTS algoritmust megvizsgálva, mindegyik működött speciális esetekben is, de egyik sem kellően hatékonyan. Ilyen megfigyelésekre alapozva vizsgálunk meg néhány optimalizációs algoritmust és javító módszert az MPTS feladatokra. A célunk hozzárendelni a feladatokhoz a processzorokat olyan módon, hogy a végrehajtási idő minimális legyen. A processzorszám nagyságától függő végrehajtási időket konvexitásuk szerint bonthatjuk különböző esetekre. A kizárólag konvex végrehajtási idejű függvényekkel leírt feladatokat $O(n)$ idő alatt ütemezhetjük, a későbbiekben ehhez adunk algoritmust. A konkáv esetben, amennyiben az elvégzendő feladatok száma konstans, és a felhasznált processzorszám mindig egész minden feladatnál, akkor a probléma megoldható polinomiális időben. Összetettebb a helyzet, ha egy feladat végrehajtáshoz használt processzorok száma nem kell, hogy egész legyen (P-CNTN). A későbbiekben bizonyítjuk, hogy ez $O(n \max(m, n\dot{\log}^2 m))$ idő alatt ütemezhető. Bizonyított, hogy a legjobb ütemezés az eredeti, egész értékű (P-DSCR), és a nem egész értékű problémára összefügg egymással. Érdekes még a P-FIX eset, ami a P-DSCR azon leszűkítése, amikor egy feladatot végző processzorszám idő közben nem változhat. Azt is vizsgáljuk a későbbiekben, hogy $n = 2$ és $n = 3$ esetben egy optimális ütemezés a nem egész értékű feladatra konstans idő alatt átalakítható az eredeti feladat megoldására.

# 2. Minimális végrehajtási idő keresése

A konvexitás szerint szeparált feladatok megoldási módszereinek részletezése előtt vezessünk be néhány alapfogalmat. Minden $j$ feladathoz tartozik egy $p_j > 0$ munkamennyiség, ami a végrehajtásához szükséges, ez a feladat hossza. Amennyiben $r$ darab processzor $t$ időn át végzi a $j$ feladatot, akkor az ezen idő alatt elvégzett munka nagysága a feladaton $g_j(r) \cdot t$, ahol $g_j(r) \geq 0$ monoton növő végrehajtási sebesség függvény, $\forall r \in \{0, 1, \ldots m\}$, $g_j(0) = 0$. Az összes $j$-n elvégzett munka nagysága meg kell, hogy egyezzen $p_j$-vel $\forall j \in \{1, 2, \ldots n\}$. Legyen $C_j$ a kezdéstől számított azon időpont, amikor befejezzük $j$ feladatot. Minden feladatra egy ütemtervet kell adnunk, hogy melyik időintervallumban hány processzor dolgozik rajta. A célunk egy olyan ütemezés készítése a lehető leggyorsabban, ami teljesíti az eddigi feltételeket, és a legkésőbb befejezett feladat befejezési idejét, azaz a $C_{max}$ futásidőt minimalizálja. Jelölje ezt a minimumot, azaz az optimális megoldásnál kapott végrehajtási időt $C^*_{max}$. A folytatásban a problémát a P-DSCR és P-CNTN esetekben vizsgáljuk. A P-CNTN esetén a P-DSCR-ben használt $g_j(r)$ függvényt lineáris szakaszokkal kell kiegészítenünk az egészek között. A P-FIX feladat NP-nehéz, ezt 1989-ben Du és Leng bizonyították [10]. Ha minden feladat 1 vagy $k$ processzort használhat, akkor a probléma polinom időben megoldható. A P-FIX problémaköre hasonlít a ládapakolási feladatokéra, ahol az alapprobléma esetén 2 dimenzióban kell elhelyeznünk azonos méretű ládákba különböző méretű, téglalap alakú tárgyakat, ehhez minél kevesebb ládát használva. Az a feladat is NP-nehéz, de ismert, hogy a ládapakolási feladat bármely $\lambda$-approximációs algoritmusa polinomiális időben átalakítható a P-FIX feladat $\lambda$-approximációs algoritmusára. A $\lambda$-approximációs algoritmus azt jelenti, hogy az optimális $C^*_{max}$ végrehajtási időhöz képest $\lambda \cdot C^*_{max}$ idő alatt végzünk a feladatokkal.

## 2.1. Eddigi eredmények

A P-FIX problémakör eseteit többen vizsgálták. Chen és Lee [6] polinomiális idejű approximációs algoritmusa után Błażewicz, Drabowski és Węglarz [5] belátták, hogy ha minden feladat 1 vagy $k$ processzort használhat, akkor a probléma megoldható polinomiális idő alatt. Turek, Wolf és Yu [24] a ládapakolási feladatra alapozva fejlesztettek ki $\lambda$-approximációs algoritmust a P-FIX feladatra. Belátták, hogy bármely $\lambda$-approximációs két dimenziós ládapakolási feladat polinomiális idő alatt átalakítható egy $\lambda$-approximációs algoritmusává a P-FIX feladatnak. A ládapakolási feladat egyik dimenziója az időnek, a másik a processzoroknak felel meg az ütemezési feladatoknál [20]. Ezt felhasználva Ludwig [18] megalkotott egy 2-approximációs algoritmust a P-FIX feladatra. Ugyanezt a problémát az NPTS feladatra is vizsgálhatjuk. Az NPTS feladatnál azonban elvárás, hogy a processzorok sorszámai egy adott feladatnál folytonosak legyenek. Ezt nevezzük sorba állított NPTS ütemezésnek. Többen foglalkoztak ezzel a problémával, például Coffman és társai [7], akik kidolgozták az NFDH (a következő beillő, csökkenő nagyság szerint rendezve) módszert, valamint FFDH (az első beillő, csökkenő nagyság szerint rendezve) módszert. Ettől eltérően Lodi [17] a BFDH (leginkább odaillő, csökkenő nagyság szerint rendezve) módszert vizsgálta. Mindegyik módszer szintekre osztotta a feladatokat, és a szinteken belüli végrehajtási idő minimalizásásra törekedett [3]. Ezekhez hasonló a bal-alsó algoritmus, melyet Baker és társai [1] mutattak be a ládapakolási feadat azon esetére, amikor 1-szélességű ládákba pakolunk. Az algoritmusuk 3-approximációs volt erre a feladatra. Ezt az eredményt Sleator [22] $2,5$-approximációsra javította. Végül Steinberg tovább tökéletesítte a módszert, és 2-approximációs eljárást adott a problémára.

Az MPTS esetet Krishnamurti és Ma tanulmányozták [16], de azzal a kikötéssel, hogy minden feladatnak a 0 időpillanatban el kell kezdődnie. Ehhez persze szükséges az a feltétel, hogy több a rendelkezésre álló processzorok száma, mint a feladatoké. Ez leegyszerűsíti a problémát arra, hogy egy párosítást találjunk a processzorok és a feladatok között. Prasanna és Musicus [21] olyan P-FIX problémákat vizsgáltak, ahol a feladatokra megelőzési feltételek vannak. Abban a speciális esetben, ha a végrehajtási idő függvény minden feladatra $p^\alpha$, ahol $p$ a feladaton dolgozó processzorok száma, $0 < \alpha < 1$, akkor zárt formulát kapunk a párhuzamosan futó feladatok optimális ütemezésére. 1996-ban Drozdowski [9] publikált egy terjedelmes áttekintést a többprocesszoros feladatok ütemezésének összetettségéről és algoritmusairól.

A párhuzamosan futó feladatokkal való modellezést a nehezen kezelhetően nagy mátrixoknál is használhatjuk. Dongarra és társai [8] azon mátrixok Cholesky-faktorizációjához használták a párhuzamos ütemezést, ahol a mátrix egésze nem

fért el a memóriában, ezzel lelassította a rajta futó processzor működését. Azonban ha blokkonként már elfér több memóriakártyán, akkor a számítások sebessége fontosabbá válik, mint a késések, amit a több processzoros rendszer okozhat. Ezzel a módszerrel a processzorszámnak a lineárisnál is jobb függvénye lehet a végrehajtási sebesség.

Fontos része még az ütemezéselméletnek az on-line ütemezés, ami azt jelenti, hogy egyes feladatokról csak az előtte elvégzett feladatok befejezése után kapunk információkat, lehetetlenné téve ezáltal az előre teljesen eltervezett ütemezés kialakítását, egy változó feladathalmaz végrehajtását megoldó algoritmust kell alkalmaznunk. Az egymástól független on-line feladatok esetét Feldmann, Sgall és Teng [12] elemezték. Kaoval kiegészülve [11] azt az esetet is kutatták, amikor a feladatok között megelőzési feltételek is lehetnek. Mindkét esetben feltették, hogy a feladatok hossza végig állandó, de ahogy eddig is, a rajtuk dolgozó processzorszám növelésével csökken a végrehajtási idejük.

## 2.2. A minimális végrehajtási idő definíciója

Először a P-CNTN esetet vizsgálva vezessünk be fontos fogalmakat. Ekkor lényeges, hogy úgy válasszuk meg az egész $r$ processzorszámok közötti nem egész értékekre a végrehajtási sebesség függvényt, hogy megmaradjanak a P-DSCR probléma alaptulajdonságai, úgy, mint a monotonitás vagy a konvexitás. Ez teljesül, ha a $g_j$ végrehajtási sebesség függvényt egy olyan $f_j$ függvénnyé terjesztjük ki, ami az eredeti függvény egész pontjai közötti értékei között lineárisan változik. Így

$$f_j(r) = \alpha_r g_j(\lfloor r \rfloor) + ((1 - \alpha_r) \cdot g_j(\lceil r \rceil)), \tag{1}$$

ahol $\alpha_r = \lceil r \rceil - r$, $0 \leq r \leq m$. Így definiálva $f_j$-t, az egész pontokban valóban megegyezik a megfelelő $g_j$ értékekkel, közöttük pedig egyenletesen változik. Ezeket a szakaszokat nem csak a két végpontjuk felvett értékének konvex kombinációjaként írhatjuk fel, hanem linearitásuk miatt az $r$ változó elsőfokú függvényeként is. Ehhez csak a megfelelő $b_{j,s}$ és $d_{j,s}$ együtthatókat kell megtalálnunk, amikre

$$f_j(r) = b_{j,s} r + d_{j,s}, \tag{2}$$

$\forall r \in [s-1, s]$, $s \in \{1, 2, ...m\}$, $j \in \{1, 2, ...n\}$ és $b_{j,0} = d_{j,0} = 0$. A $b_{j,s}$ és $d_{j,s}$ együtthatók adott $g_j(r)$-re $O(mn)$ idő alatt kiszámolhatóak, mivel ezek $m \cdot n$ adott pontpáron átmenő egyenesek egyenletei. Mivel egy feladatra két azonos egészrészű $r$-re a $b_{j,s}$ és $d_{j,s}$ értékek külön-külön megegyeznek, ezért $f_j(r)$-t $\lceil r \rceil$-szel is megadhatjuk:

$$f_j(r) = b_{j,\lceil r \rceil} r + d_{j,\lceil r \rceil}. \tag{3}$$

Mivel $f_j(r) = g_j(r)$, ezért a P-DSCR probléma $f_j$ és $g_j$ függvényekkel megegyezik. Újra a P-CNTN problémát tekintve az $f_j$ függvényekkel, legyen $C^0_{max}$ a P-CNTN feladat minimális $C_{max}$ értéke, azaz a megoldása. Mivel $C^*_{max}$ a P-DSCR megoldása volt, aminek kiterjesztése a P-CNTN, ezért $C^0_{max} \leq C^*_{max}$. Az összes lehetséges erőforrás felosztást jelölje $R$, amire így:

$$R = \left\{ r = (r_1, r_2, ...r_n) \mid r_j \geq 0, \sum_{j=1}^{n} r_j \leq m \right\}, \tag{4}$$

valamint

$$U = \{ u = (u_1, u_2, ...u_n) \mid u_j = f_j(r_j), \forall j \in \{1, 2, ...n\}, r \in R \} \tag{5}$$

jelölje az összes feladaton egyszerre végzett munkák sebességvektorát. Legyen $p = (p_1, p_2, \ldots p_n)$ a munkamennyiség vektor.

**2.1. Tétel.** *[25], [26] Legyen $n \leq m$, $convU$ az $U$ halmaz konvex burka, azaz $U$ elemeinek összes konvex kombinációja. $u := \frac{p}{C}$ egy n-dimenzióbeli egyenes, amelynek koordinátáit az $u_j = \frac{p_j}{C}$ paraméteres egyenletből kapjuk, ahol $j \in \{1, 2, \dots n\}$. Ekkor a minimális végrehajtási idejét a P-CNTN problémának a következő képlet írja le:*

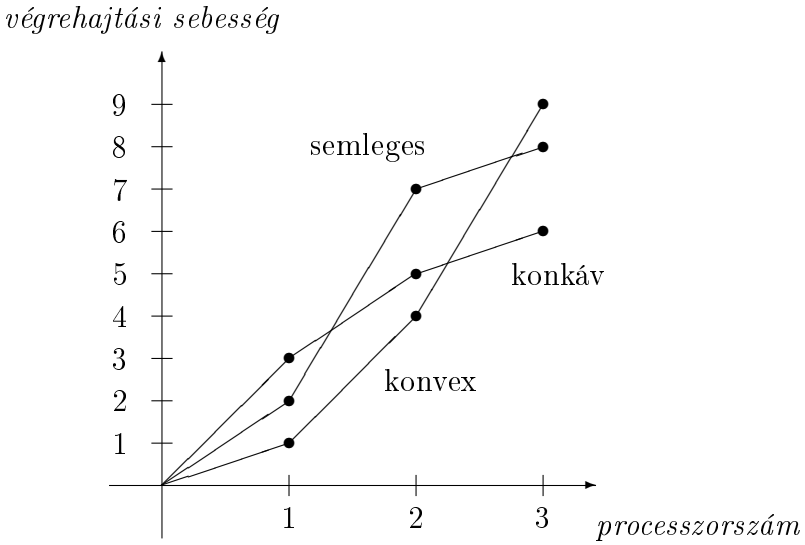
$$C^0_{max} = min\left\{C \mid C > 0, \frac{p}{C} \in convU\right\}. \tag{6}$$

*Bizonyítás:* Amennyiben találunk olyan $C$-t, amire igaz, hogy $\frac{p}{C} \in convU$, akkor az ütemezés biztosan előállítható $U$-beli elemek konvex kombinációjaként. Ez pedig akkor lesz a leghasznosabb, ha a kombináció a konvex burok peremén van, hiszen ekkor szorozhatjuk a legnagyobb $\frac{1}{C}$-vel, azaz a legkisebb $C$-vel $p$-t, hogy még éppen a konvex burkon belül maradhassunk. Így a lehetséges $r$ processzorszétosztások közül néhány konvex kombinációjaként megkaptuk a lehető leggyorsabb ütemezést, mivel nagyobb $C$ esetén lassabb megoldást kapunk, kisebb esetén pedig nem lennénk már a konvex burokban, azaz nem megengedett processzorszétosztást kéne alkalmaznunk. □

A bizonyításban említett konvex kombinációt jelölje $u^0$. Ekkor $C^0_{max} = \frac{p_j}{u^0_j}$, $\forall j \in \{1, 2, ...n\}$. Speciális esetben ezt az $u^0$ konvex kombinációt az $U$ elemeinek több konvex kombinációjaként is előállíthatjuk, ezért jelölje $r^0$ azt az optimális erőforrás szétosztást, amire $\sum_{j=1}^{n} r^0_j = m$ és $f_j(r^0_j) = u^0_j$, $j \in \{1, 2, \dots n\}$. A továbbiakban először azt az esetet vizsgáljuk, amikor minden $f_j$ függvény konvex, majd amikor mindegyik konkáv.

# 3. Konvex sebességfüggvények

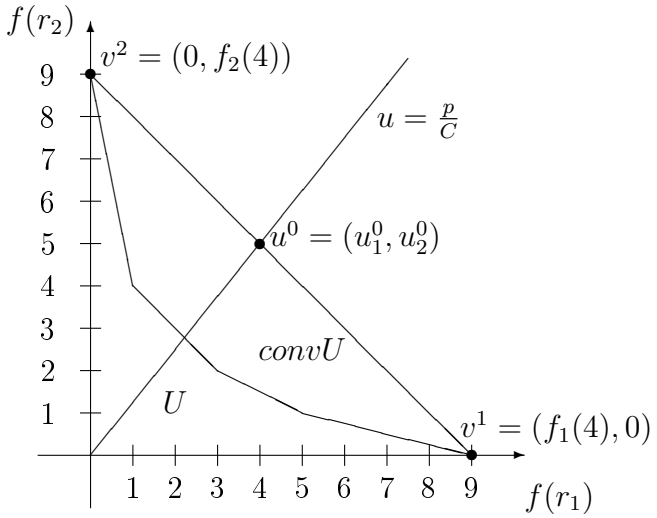
Minden $f_j$ függvény lehet konvex, konkáv, esetleg egyik sem. Ez utóbbi függvényeket nevezhetjük semlegesnek. Ezt szemlélteti a 3.1. ábra.




3.1. Szakaszonként lineáris függvények konvexitása

Az első esetben legyen minden $f_j$ függvény konvex, és mivel szakaszonként lineáris függvényekről beszélünk, ezért a maximum $m$ töréspont miatt $O(mn)$ idő alatt ellenőrizhető is a konvexitás. Mivel a függvény konvex, ezért a bevezetett $U$ halmaz (5) valódi részhalmaza a *convU* halmaznak, azaz a *convU* halmaz bővebb. Vezessük be a $v^i$ pontokat a következő módon, koordinátáiknak megadásával: $v^j_j = f_j(m)$, és $v^i_j = 0$, ha $i \neq j \ \forall i \in \{1, 2, \ldots n\}$, $j \in \{1, 2, \ldots n\}$. Könnyen látható, hogy a $v^1, v^2, ...v^n$ pontok a *convU* konvex burok egy lapjának csúcsai, és az $u^0$ pont is ezen a lapon helyezkedik el. Így $u^0$ előáll a $v^1, v^2, \ldots v^n$ pontok konvex kombinációjaként, azaz $u^0 = \sum_{i=1}^{n} \lambda_i v^i$, ahol $\lambda_i \geq 0 \ \forall \ i \in \{1, 2, \ldots n\}$ és $\sum_{i=1}^{n} \lambda_i = 1$.

Ezeket láthatjuk a 3.2. ábrán, $n = 2$ esetén. A koordinátarendszerben a két feladat végrehajtási sebességét ábrázoljuk egymáshoz képest, attól függően, hogy az $m = 4$ processzorból melyik feladaton hány dolgozik. Az így kapott grafikon egy konvex törött vonal, az ez alatti terület lesz $U$. A nem egész processzorszámokra a fent említett lineáris függvényeket használjuk, így jön létre a törött vonal. A terület konvex burkát, azaz *convU*-t a törött vonal két végpontját összekötő szakasz és a koordinátatengelyek határolják, mivel ezek a szakaszok még megkaphatóak a $v^i$ pontok konvex kombinációjaként, de ezen a területen kívül semmilyen más pont nem.



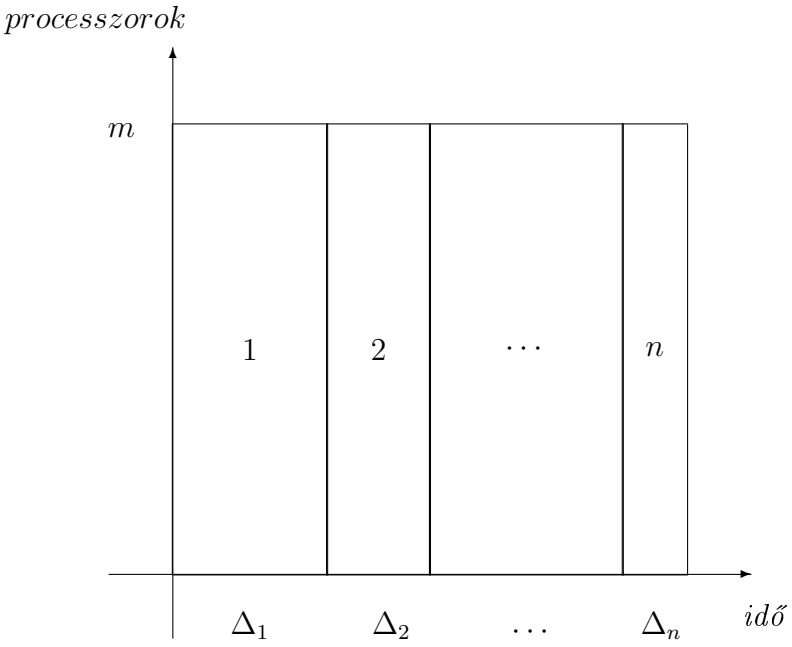

3.2. Konvex függvények minimális végrehajtási ideje

A törött vonal előállításához a 3.3. táblázat nyújt segítséget, ahol leolvasható a két feladat végrehajtási sebessége a processzorszám függvényeként. A processzorszám soronként növekszik, a két oszlop a két feladatnak felel meg. Miután kiszámoljuk az $u = \frac{p}{C}$ egyenes egyenletét, ábrázoljuk is, majd a keletkezett egyenes, és a $(v^1, v^2)$ szakasz metszéspontja adja a keresett $u^0$ pontot.

Tudjuk, hogy $u_j^0 = \frac{p_j}{C_{max}^0} = \sum_{i=1}^{n} \lambda_i v_j^i$, ami átírva $p_j = \sum_{i=1}^{n} \lambda_i C_{max}^0 \cdot v_j^i$. Legyen $\Delta_j = \lambda_j C_{max}^0 = \frac{p_j}{v_j^j} = \frac{p_j}{f_j(m)}$, $\forall j \in \{1, 2, ...n\}$. Mivel $\sum_{j=1}^{n} \Delta_j = C_{max}^0$ és $p_j = \Delta_j f_j(m)$ $\forall j \in \{1, 2, ...n\}$, ezért optimális ütemezést kapunk, ha $\forall j$-re $\Delta_j$ időn keresztül a $j$ feladaton mind az $m$ processzort használjuk, hiszen ennyi idő alatt elvégzik azokat, az előbbiek miatt a leggyorsabb módon. A 3.4. ábrán láthatunk egy optimális ütemezést tetszőleges feladat- és processzorszámra, a kiszámított $\Delta_j$ értékek segítségével. Ezek alapján látszik, hogy ha minden $f_j$ függvény konvex, akkor az optimális ütemezést $O(n)$ idő alatt megtalálhatjuk a P-CNTN esetben. Mivel a megoldásban kapott processzorszétosztás egész értékű volt, ezért ez az optimális ütemezés a P-DSCR esetben is. Ha az $f_j$ végrehajtási sebesség függvény nem konvex, akkor jóval nehezebb megtalálni az optimális ütemezést. Ezek közül most azt az esetet vizsgáljuk meg, amikor a függvényeink mind konkávok.

| | | |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 1 | 1 |
| 2 | 2 | 3 |
| 3 | 4 | 5 |
| 4 | 9 | 9 |
| | $f_1$ | $f_2$ |

3.3. Végrehajtási sebességek a processzorszám alapján




3.4. Az optimális ütemezés a konvex feladatra

# 4. Konkáv sebességfüggvények

## 4.1. Approximációs algoritmusok

Ebben a részben azt az esetet tárgyaljuk, amikor az összes $f_j$ függvényünk konkáv. Mivel ez lényegesebben összetettebb és nagyobb futásidejű probléma, ezért különböző approximációs algoritmus tárgyalásával kezdjük, amik kellően jól közelítik az optimális processzorszétosztást, és lényegesen rövidebb a futásidejük az optimális megoldást megtaláló leggyorsabb algoritmusoknál.

A végrehajtási sebesség függvények konkávok, ebből következik, hogy $\frac{f_j(r)}{r} \geq \frac{f_j(r+1)}{r+1}$ $\forall r \in \{1, 2, ...m-1\}$, azaz egy feladathoz több processzort használva romlik azok egyenkénti hatékonysága. Természetesen az $f_j(r) \leq f_j(r+1)$ összefüggés továbbra is fennáll $\forall r \in \{1, 2, \dots m-1\}$ esetén.

Az első approximációs algoritmus a P-FIX azon speciális esetében ad ütemezést, amikor nemcsak hogy nem változhat idő közben a megkezdett feladatokon a processzorszám, de ez az érték mindegyik feladatra 1. Azaz egy feladaton egyszerre csak egy processzor dolgozhat. Már ez a feladat is $NP$-nehéz $m \geq 2$ esetén. Ez egyúttal a ládapakolási feladat egy speciális esete is, az első megoldás listás ütemezést használ, azaz a feladatokat tetszőleges sorrendben olvassa be egy listáról, majd ezeket a megfelelő processzorokhoz rendeli. Ez $2 - \frac{1}{m}$-approximációs megoldást nyújt a speciális P-FIX feladatra (a későbbi 4.1. Állítás). $F_r$ az $r$ processzor aktuális feladatbefejezési idejét mutatja. Az algoritmus a következő:

**Listás algoritmus**

1. $i := 1$, $F_r = 0$ $\forall r \in \{1, 2, \dots m\}$.
2. Válasszuk ki a legkisebb $F_r$ értékű (egyenlőségnél egy tetszőleges, az egyenlőségben szereplő) $r$ processzort a processzorok közül.
3. Rendeljük az $i$. feladathoz az $r$ processzort, és $F_r$-t frissítsük: $F_r = F_r + \frac{p_i}{f_i(1)}$.
4. Ha $i \neq n$, akkor növeljük: $i = i + 1$, és térjünk vissza a 2.-es lépéshez. Ha $i = n$, akkor befejeződött az algoritmus, és a végrahajtási idő megegyezik a kapott $F_r$ értékek legnagyobbikával.

Az algoritmus tehát listaszerűen olvassa be a feladatokat, és mindig az aktuálisan leghamarabb végző processzorhoz rendeli őket. A processzorok feladatbefejezési idejeit a 3. lépésben a kellő futási időkkel növeli. Végül ha már minden feladatot beolvastunk, akkor már csak azt kell megnéznünk, hogy melyik fejeződik be legkésőbb, ez adja meg a végrehajtási időt.

**4.1. Állítás.** *Ez az algoritmus* $2-\frac{1}{m}$*-approximációs, ha egy feladaton egyszerre csak egy processzor dolgozhat [14].*

*Bizonyítás:* Először tegyünk két triviálisnak tűnő alsó becslést $C^*_{max}$-ra:

1. $C^*_{max} \geq \sum_{j=1}^{n} \frac{p_j}{m}$.
2. $C^*_{max} \geq \max p_j$.

Az első állítás teljesül, mivel nem végezhetünk hamarabb, mint az összes feladat összhossza, leosztva a processzorok számával. Az egyenlőség akkor teljesül, ha $C^*_{max}$-ig minden processzor folyamatosan dolgozik. A második állítás pedig még kézenfekvőbb, hiszen nem végezhetünk hamarabb az összes feladattal, mint bármelyikőjük (itt a leghosszabb) hossza. Legyen $l$ az utoljára befejeződött feladat. Az $l$ elkezdéséig minden processzor folyamatosan dolgozik, különben már korábban is elkezdhettük volna $l$-et. Vagyis ha $T_l$ jelöli $l$ kezdési idejét, akkor:

$$C_{max} = C_l = T_l + p_l \leq \sum_{j=1, j\neq l}^{n} \frac{p_j}{m} + p_l = \sum_{j=1}^{n} \frac{p_j}{m} + (1 - \frac{1}{m}) \cdot p_l \leq (2 - \frac{1}{m}) \cdot C^*_{max}$$

ahol az első egyenlőtlenség azért teljesül, mert az utoljára befejezett feladat elkezdéséig minden processzor folyamatosan dolgozik, a második pedig a fenti 1. és 2. triviális alsóbecslés következménye. Ezzel az állítást igazoltuk.

□

Az egyszerű listás ütemezés egy hatékonyabb továbbfejlesztése az LPT (a feladat hossza szerint nem növekvő sorrendbe rendező) algoritmus, aminél így a listáról épp beolvasott feladat az addig be nem olvasottak közül a leghosszabb. Az LPT algoritmus $O(n \cdot \log n + n \cdot \log m)$ idő alatt elvégezhető. Bizonyítottan $\frac{4}{3} - \frac{1}{3m}$ approximációs algoritmus a leszűkített P-FIX problémára, ami "nagy" $m$ esetén $\frac{4}{3}$-hoz tartó approximációt eredményez. A $\frac{4}{3} - \frac{1}{3m}$ approximációnál könnyebben bizonyítható, de kicsit gyengébb állítás a következő:

**4.2. Állítás.** *A P-FIX azon esetében, amikor minden feladaton csak egy processzor dolgozhat, az LPT szerinti listás ütemező algoritmus* $\frac{4}{3}$*-approximációs. [14]*

*Bizonyítás:* Feltehető, hogy az utoljára végződő feladat kezdődik utoljára, ha nem így lenne, akkor az összes később kezdődő feladatot vegyük ki a feladatok közül, ezáltal $C_{max}$ nem változik, de $C^*_{max}$ csökkenhet, ez tehát csak ronthat a közelítésünkön. Legyen az említett feladat hossza $p_j$. Mivel az összes többi feladat már előbb kezdődik mint $j$, ezért ő a legrövidebb. Vizsgáljunk két esetet:

1. $p_j \leq \frac{C^*_{max}}{3}$.

   A processzorok $C_{max} - p_j$-ig biztosan mind dolgoznak, mert csak akkor szabadul fel egy $j$-nek. Azaz ennél az értéknél $C^*_{max}$ sem lehet kisebb, és ezután $p_j$ hosszal befejeződik $C_{max}$. Azaz $C_{max} \leq C^*_{max} + p_j \leq C^*_{max} + \frac{C^*_{max}}{3} = \frac{4 \cdot C^*_{max}}{3}$, ezzel az állítást erre az esetre beláttuk.

2. $p_j > \frac{C^*_{max}}{3}$.

   Mivel $j$ a legrövidebb feladat, ezért minden $i$ feladatra $p_i > \frac{C^*_{max}}{3}$. Ekkor minden processzor maximum 2 feladaton dolgozhat az optimális ütemezéskor, így $2 \cdot m \geq n$. Az első $m$ feladatot LPT sorrendben kezdjük el a 0 időben végrehajtani a processzorokon, majd amikor egy processzor felszabadul, akkor rendeljük az LPT sorrendben következő feladathoz. Ezzel a leghosszabb feladatoknak nem lesz csak párja, ami ugyanazon a processzoron futna, a rövidebbeket pedig a lehető leghatékonyabban, a kisebbeket a nagyobbakkal párosítva ütemeztük, így ennél gyorsabb ütemezés nincs is ebben az esetben, így $C_{max} = C^*_{max}$, természetesen erre is teljesül a fenti egyenlőtlenség.

□

Ezek után nézzük az algoritmus hatékonyságát az eredeti, egy feladaton több processzor együttes munkáját is megengedő problémára. Az optimális ütemezés megtalálása továbbra is $NP$-nehéz. Az algoritmus alapgondolata az, hogy az LPT algoritmus elkészítése után iterálva módosítsuk azt úgy, hogy a feladatokhoz több processzort rendelünk, ezzel csökkentve a végrehajtási időt. Ehhez minden $i$ feladathoz vezessünk be egy $s_i$ értéket, ami az $i$-n aktuálisan dolgozó processzorszámot jelöli. Az algoritmus során folyamatosan növeljük a processzorszámot bizonyos feladatokon. Azonban csak akkor engedünk meg egy processzorszám növelést, ha az azonnali csökkenést okoz a végrehajtási időben. Az algoritmus a következő:

**Javított LPT algoritmus**

1. $s_i := 1$, $\forall\ 1 \leq i \leq n$. Valamint $a := m$, ahol $a$ az $m - \sum_{s_i > 1} s_i$ értéket veszi majd fel az algoritmus során. A feladatokra végezzük el az LPT ütemezést.

2. Ha $a = 0$, akkor térjünk az 5. lépésre. Különben válasszunk ki egy olyan $i$ feladatot, amelyre $\frac{p_i}{f_i(s_i)} \geq \frac{p_j}{f_j(s_j)}$, $\forall j\ 1 \leq j \leq n$. $h := \frac{p_i}{f_i(s_i)}$.

3. Ha $a = 1$ és $s_i = 1$, vagy $C_{max} \neq h$, akkor térjünk az 5. lépésre. Különben folytassuk a 4.-kel.

4. Ha $s_i = 1$, akkor $a := a - 2$, különben $a := a - 1$. $s_i := s_i + 1$. Készítsük el a következő $C'$ ütemezést: először határozzuk meg azokat a $j$ feladatokat, melyekre $s_j > 1$, és ezekhez a megfelelő $s_j$ darabszámú processzort rendeljük (az összdarabszám nem nagyobb $m$-nél). Majd a maradék feladatokra végezzük el az LPT algoritmust. Ha $C'_{max} > h$, akkor $s_i := s_i - 1$ és térjünk az 5. lépésre. Különben $C := C'$ és térjünk vissza a 2. lépésre.

5. A kapott ütemezés lesz a Javított LPT algoritmus végső ütemezése, ezt nevezzük $A$-nak.

A célunk az, hogy belássuk, hogy a Javított LPT algoritmus $\frac{2}{1+\frac{1}{m}}$ approximációs algoritmus erre a problémára. Ennek bizonyításához belátunk több lemmát, de először vezessünk be néhány új jelölést és kifejezést. Legyen $S$ egy ütemezés. Jelölje $k_i(S)$ az $i$ feladatot végző processzorok számát $S$-ben. Ha egy $i$ feladaton $k_i(S) > 1$ processzor dolgozik, akkor azt úgy is kezelhetjük, mintha $k_i(S)$ darab részfeladat lenne, mindegyik $\frac{p_i}{f_i(k_i(S))}$ végrehajtási idővel, mindegyiken egy processzort dolgoztatva. Ezeket a részfeladatokat nevezzük hasított feladatoknak. A többi esetben, amikor $k_i(S) = 1$, azon $i$ feladatokat hasítatlan feladatoknak hívjuk. Ezen két fajta feladat ütemezése esetén figyelni kell arra, hogy a hasított feladatok mindig egy időben kezdődjenek. Az aktuális megoldásunkban , azaz a Javított LPT algoritmusban először a hasított feladatokhoz rendeljünk processzorokat, majd a hasítatlan feladatokhoz az LPT sorrendjüket alkalmazva. Az algoritmusban használt $a$ változó garantálja, hogy maximum $m$ hasított feladat lesz. Így az, hogy egyszerre kezdődnek nem sérti meg az ütemezési feltételeket. A lemmáink előtt pedig két definíció:

**4.3. Definíció.** *Legyen $AVG(S)$ az $S$ ütemezésben az $m$ darab processzor átlagos munkavégzési ideje az egész folyamat alatt. Azaz $AVG(S) := \frac{1}{m} \cdot \sum_{i=1}^{m} \frac{p_i \cdot k_i(S)}{f_i(k_i(S))}$*

**4.4. Definíció.** *Legyen $h(S)$ a maximális végrehajtási idejű feladat az $S$ ütemezésben. Azaz $h(S) := \max\left\{\frac{p_i}{f_i(k_i(S))}\right\}$.*

Most pedig következzen az algoritmus hatékonyságáról korábban feltetteket igazoló lemmasor:

**4.5. Lemma.** $S_{max} \geq \max\{h(S), AVG(S)\}$.

*Bizonyítás:* A befejezési idő nem lehet kevesebb sem az átlagos processzor munkavégzési időnél, sem a leghosszabb feladat hosszánál, így ez triviális. □

**4.6. Lemma.** *Ha $A_{max} \neq A^*_{max}$, akkor $k_i(A) \leq k_i(OPT)$, $\forall 1 \leq i \leq n$, ahol $A$ a Javított LPT algoritmus által kapott ütemezés, $OPT$ pedig az optimális ütemezés.*

*Bizonyítás:* Bizonyítsuk az állítást indirekten. Tegyük fel, hogy $\exists i$: $k_i(A) > k_i(OPT)$. Mivel a Javított LPT algoritmus során folyamatosan 1-esével növeltük az $i$ feladatnál használt processzorok számát, ezért volt egy pillanat, amikor $s_i = k_i(OPT)$ teljesült. Majd az $i$ feladat kapott még egy processzort az algoritmus ugyanezen szakaszán, mivel a 3. lépés állítása igaz rá. Ez azt is jelenti egyúttal, hogy az aktuális ütemezés befejezési ideje ebben a szakaszban $\frac{p_i}{f_i(k_i(OPT))}$. Mivel az aktuális befejezési idő az algoritmus során végig csak csökkenhet, ezért $A_{max} \leq \frac{p_i}{f_i(k_i(OPT))}$. A 4.5. Lemma szerint mivel az $i$ feladaton $k_i(OPT)$ darab processzor dolgozik az optimális megoldásban, ezért $A_{max}* \geq \frac{p_i}{f_i(k_i(OPT))}$. Azaz $A_{max} \leq A^*_{max}$, amivel ellentmondáshoz jutottunk, ezzel bizonyítva a lemmát. □

**4.7. Lemma.** *Ha $A_{max} \neq A^*_{max}$, akkor $AVG(A) \leq AVG(OPT)$*

*Bizonyítás:* Az $f_i$ függvények konkáv tulajdonsága miatt, a bevezetésben leírtak miatt $\frac{f_j(r)}{r} \geq \frac{f_j(r+1)}{r+1}$, $\forall r \in \{1, 2, ...m-1\}$, így a reciprokaikra pozitív értékek lévén megfordul az egyenlőtlenség: $\frac{r}{f_j(r)} \leq \frac{r+1}{f_j(r+1)}$ . Mivel a 4.6. Lemma szerint $k_i(A) \leq k_i(OPT)$, ezért $\frac{k_i(A)}{f_i(k_i(A))} \leq \frac{k_i(OPT)}{f_i(k_i(OPT))}$ $\forall i$ feladatra. Az egyenlőtlenségek mindkét oldalát $p_i$-vel megszorozva, majd ezeket átlagolva kapjuk a lemma állítását. □

**4.8. Lemma.** *Legyen $S$ egy olyan ütemezés, ahol a hasított és hasítatlan feladatok az algoritmus 4. lépésében leírtak alapján követik egymást. A hasított feladatok száma, $x$ nem nagyobb $m$-nél. Legyen $y$ a végrehajtási ideje az $m-x+1$. legnagyobb hasítatlan feladatnak (ha nincs ennyi hasítatlan feladat, akkor $y := 0$). Ekkor*

$$S_{max} \leq \max\left\{h(S), AVG(S) + y \cdot (1 - \frac{1}{m})\right\}$$

.

*Bizonyítás:* Az $S$ ütemezés befejezési ideje egybeesik egy 0 időpontban elkezdett $i$ feladat befejezési idejével, vagy egy később elkezdett befejezési idejével. Az előbbi esetben $S_{max} = h(S)$, ahol $h(S)$-ben az $i$ feladat szerepel, így igaz a lemma. Az utóbbi esetben pedig $i$ egy hasítatlan feladat, amire $\frac{p_i}{f_i(1)} \leq y$. Ez azért van, mert a legnagyobb $m - x$ hasítatlan feladatot a 0 időpontban kezdjük el. Az $i$ feladat az $S_{max} - \frac{p_i}{f_i(1)}$ időpontban kezdődik. Az LPT algoritmus nem engedi, hogy bármelyik

processzor is leálljon ez előtt az időpont előtt. Ezért $S_{max} - \frac{p_i}{f_i(1)} \leq AVG'$, ahol $AVG'$ az ekkor épp futó feladatok átlagos befejezési ideje. De $AVG' \leq AVG(S) - \frac{p_i}{f_i(1) \cdot m}$, mert az $AVG'$-ben vett feladatokat végző processzorok nem állhatnak le, mielőtt elvégeznék a feladatokat, valamint az $i$ feladaton dolgozó processzor $i$-n $\frac{p_i}{f_i(1)}$ ideig dolgozik, ezzel $\frac{p_i}{f_i(1) \cdot m}$-mel növelve a processzorok átlagos munkavégzési idejét. A két egyenlőtlenséget összevonva: $S_{max} \leq AVG(S) + \frac{p_i}{f_i(1)} \cdot (1 - \frac{1}{m}) \leq AVG(S) + y \cdot (1 - \frac{1}{m})$, ezzel beláttuk a lemmát. □

**4.9. Lemma.** *Legyen $C$ a Javított LPT algoritmus során kialakuló ütemezések közül egy. Legyen $z$ a végrehajtási ideje egy hasított feladatnak $C$-ben. Ekkor $C_{max} \leq 2 \cdot z$.*

*Bizonyítás:* Legyen $i$ a $z$ végrehajtási időhöz tartozó feladat. Ekkor $z = \frac{p_i}{f_i(k_i(C))}$. Az algoritmus 3. lépése miatt egyértelmű, hogy $C_{max} \leq \frac{p_i}{f_i(k_i(C)-1)}$, mivel valamikor hozzáadtak még egy processzort az $i$ feladat elvégzéséhez, hogy az akkori $C_{max}$ tovább csökkenhessen. Az $f$ függvények konkávsága miatt tetszőleges $j$ processzorszámot egy feladaton $j+1$-re növelve, a végrehajtási idő maximum a $\frac{j}{j+1}$-szeresére csökkenhet, és mivel $j \geq 1$, ezért ez maximálisan 2-szeres gyorsítást jelent. Azaz $\frac{p_i}{f_i(k_i(C)-1)} \leq \frac{2 \cdot p_i}{f_i(k_i(C))} = 2 \cdot z$. A két egyenlőtlenséget összevonva kapjuk a lemma állítását. □

**4.10. Lemma.** *Legyen $C$ a Javított LPT algoritmus során kialakuló ütemezések közül egy. Legyen $x$ a hasított feladatok száma, $y$ pedig az $m - x + 1$. legnagyobb hasítatlan feladat végrehajtási ideje. Ekkor:*

1. $y \leq \frac{p_i}{f_i(k_i(C))}$, $\forall i \mid k_i(C) > 1$.
2. $C_{max} \leq \max\left\{h(C), \frac{2 \cdot AVG(C)}{1 + \frac{1}{m}}\right\}$.

*Bizonyítás:* Az 1. állításhoz azt kell belátnunk, hogy $y$ kisebb az összes hasított feladat végrehajtási idejénél. Legyen $i$ a legkisebb végrehajtási idejű feladat a hasított feladatok közül. Legyen $z$ ennek végrehajtási ideje, azaz $\frac{p_i}{f_i(k_i(C))}$. A 4.9. Lemmából következik, hogy $C_{max} \leq 2 \cdot z$. Ha indirekten feltesszük, hogy $z < y$, akkor $C_{max} \geq y + z$ egyenlőtlenség teljesülni fog, mivel az $y$ végrehajtási időhöz tartozó hasítatlan feladat egy legalább $z$ végrehajtási idejű hasított feladat befejezése után kezdődik, vagy egy legalább $y$ végrehajtási idejű hasítalan feladat után. Ebből azt kapjuk, hogy $2 \cdot z \geq z + y$, azaz $z \geq y$, amivel ellentmondáshoz jutottunk, ezzel belátva az 1. állítást. Azt 1. állítás következményeként az $y$-hoz tartozó feladat

végrehajtási ideje maximum akkora, mint az $x$ hasított feladat bármelyikének végrehajtási ideje. Szintén tudjuk, hogy $y$ nem nagyobb az $m - x$ legnagyobb hasítatlan feladat bármelyikének végrehajtási idejénél a definíció szerint. Összességében így van $m + 1$ feladatunk legalább $y$ végrehajtási idővel, így a processzorok munkavégzési idejére $AVG(C) \geq y + \frac{y}{m}$, átrendezve $y \leq \frac{AVG(C)}{1+\frac{1}{m}}$. Ebből és a 4.8. Lemmából következik, hogy $S_{max} \leq \max\left\{h(S), AVG(S) + \frac{AVG(C)}{1+\frac{1}{m}} \cdot (1 - \frac{1}{m})\right\}$, amiből következik a 2. állítás. □

**4.11. Tétel.** *Legyen $A$ a Javított LPT algoritmus által kapott ütemezés. Ekkor:*

$$A_{max} \leq \frac{2}{1 + \frac{1}{m}} \cdot A^*_{max}$$

.

*Bizonyítás:* Feltehető, hogy $C_{max} \neq C^*_{max}$, különben az állítás triviálisan igaz. Ekkor $h(A)$ mérete szerint vizsgáljunk két fő esetet:

1. $C_{max} \neq h(A)$.
   A 4.5. Lemma alapján $A_{max} \geq \max\{h(A), AVG(A)\}$. Ebben az esetben ez azt jelenti, hogy $A_{max} > h(A)$. A 4.10. Lemma 2. állítása miatt $A_{max} \leq \frac{2 \cdot AVG(A)}{1+\frac{1}{m}}$. A 4.7. Lemma szerint $AVG(A) \leq AVG(OPT)$, és a 4.3. Lemma miatt az optimális ütemezésre $AVG(OPT) \leq C^*_{max}$. Ezeket összevonva kapjuk a $A_{max} \leq \frac{2}{1+\frac{1}{m}} \cdot A^*_{max}$ egyenlőtlenséget, ami a bizonyítandó állítás volt.

2. $C_{max} = h(A)$.
   Ebben az esetben vizsgáljuk meg azt, hogy az algoritmus melyik lépéséből érkeztünk az 5. lépésre, azaz miért fejeződött be az algoritmus. Legyen $b$ egy $h(A)$ végrehajtási idejű feladat, ilyen a feltétel miatt van. Ekkor $h(A) = \frac{p_b}{f_b(k_b(A))}$. Legyen az $A$ ütemezésben lévő hasított feladatok száma $x$.

   (a) Az algoritmus azért állt le, mert $a = 0$ lett a 2. lépésben vagy $a = 1$ és $k_b(A) = 1$ (azaz $b$ egy hasítatlan feladat). Az első esetben $m$, a másodikban $m - 1$ a hasított feladatok száma. Legyen $z$ a minimális végrehajtási idejű $l$ hasított feladat végrehajtási ideje, azaz $z = \frac{p_l}{f_l(k_l(A))}$. Mivel legalább $m-1$ hasított feladatunk van legalább $z$ végrehajtási idővel, valamint még egy feladat $h(A)$ végrehajtási idővel, ezért $AVG(A) \geq z \cdot (1 - \frac{1}{m}) + \frac{h(A)}{m}$. Mivel $A_{max} = h(A)$, ezért átírhatjuk az egyenlőtlenséget $z$-re rendezve: $z \leq \frac{AVG(A) - \frac{A_{max}}{m}}{1-\frac{1}{m}}$. A 4.9. Lemma alapján $2 \cdot z \geq A_{max}$. Ezt felhasználva $A_{max}$-ra rendezve az egyelőtlenséget:

$A_{max} \leq \frac{2 \cdot AVG(A)}{1+\frac{1}{m}} \leq \frac{2 \cdot A^*_{max}}{1+\frac{1}{m}}$, ahol az első egyenlőtlenség a 4.5. Lemmából, a második a 4.7. Lemmából következik, ezzel beláttuk a tétel állítását ebben az esetben is.

(b) Az algoritmus azért állt le, mert a 3. lépésében $C_{max} \neq h(A)$. Ezt már vizsgáltuk az 1. fő esetnél.

(c) Az algoritmus azért állt le, mert a 4. lépésben kapott $C'$ ütemezésre $C'_{max} > h(A)$. A hasított feladatok száma $C'$-ben a 4. lépés növelése miatt: $x' = x + 1$. Legyen $y$ a $m - x' + 1$. legnagyobb végrehajtási idejű hasítatlan feladat. A 4.6. Lemmából: $k_b(OPT) \geq k_b(A)$. Ha ez egyenlőséggel teljesül, akkor a 4.5. Lemma miatt $A^*_{max} \geq \frac{p_b}{f_b(k_b(A))} = h(A) = A_{max}$, azaz $A^*_{max} = A_{max}$, azaz teljesül a tétel egyenlőtlensége. Ha pedig $k_b(OPT) > k_b(A)$, akkor $k_b(OPT) \geq k_b(A) + 1 = k_b(C')$. Ebből és a 4.7. Lemma alapján $AVG(C') \leq AVG(OPT) \leq A^*_{max}$. Az $y$ nagysága alapján tekintsünk két alesetet:

i. Az $y$ érték nem nagyobb, mint bármely $C'$-beli hasított feladat végrehajtási ideje.
A 4.10. Lemma 2. állítása alapján $C'_{max} \leq \max\left\{h(C'), \frac{2 \cdot AVG(C')}{1+\frac{1}{m}}\right\}$. Mivel $C'_{max} > h(A) \geq h(C')$, ezért az előbbi egyenlőtlenség csak úgy teljesülhet, ha $C'_{max} \leq \frac{2 \cdot AVG(C')}{1+\frac{1}{m}} \leq \frac{2 \cdot A^*_{max}}{1+\frac{1}{m}}$. Ugyanakkor $A_{max} = h(A) < C'_{max}$, így ezen két egyenlőtlenség összevonásával kapjuk a bizonyítandó állítást.

ii. Van olyan hasított feladat, melynek végrehajtási ideje kisebb, mint az $y$ érték.
Legyen $z$ a minimális hasított feladat végrehajtási idő $C'$-ban, és a hozzá tartozó feladat $l$. Ekkor $y > z$, valamint $z$ nem nagyobb, mint $x'$ hasított, és $m - x'$ hasítatlan feladat végrehajtási ideje. Ezeken kívül van még az $y$ végrehajtási időhöz tartozó feladat, így a processzorok átlagos munkavégzése alulról becsülhető: $AVG(C') \geq \geq z + \frac{y}{m} \geq z \cdot (1 + \frac{1}{m})$. Ha $b \neq l$, akkor az $l$ feladathoz az $A$ és a $C'$ ütemezésben azonos processzorszám tartozik. Emiatt ha alkalmazzuk a 4.9. Lemmát az $A$ ütemezésre, azt kapjuk, hogy $A_{max} \leq 2 \cdot z$. Ha $b = l$, akkor $A_{max} = \frac{p_b}{f_b(k_b(A))} \leq \frac{p_b}{f_b(k_b(A)+1)}$, mivel $k_b(C') = k_b(A) + 1$. Így mindkét esetben azt kaptuk, hogy $A_{max} \leq 2 \cdot z$. A fenti egyenlőtlenségekből következik, hogy $A_{max} \leq \frac{2 \cdot AVG(C')}{1+\frac{1}{m}} \leq \frac{2 \cdot C^*_{max}}{1+\frac{1}{m}}$, amivel az utolsó esetre is beláttuk a tétel állítását.

□

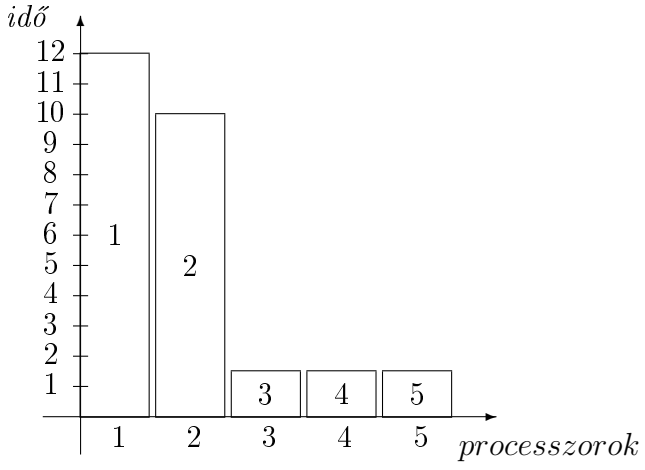
**4.12. Tétel.** *A Javított LPT algoritmus futásideje* $O(n \cdot \log n + n \cdot m \cdot \log m)$ *[13], [14].*

*Bizonyítás:* A Javított LPT algoritmus a listás algoritmuson alapul, melynek futásideje $O(n \cdot \log n + n \cdot \log m)$. Ehhez képest az új algoritmus az 1. lépésben elkészíti a listás ütemezést, a 2. lépésében $O(\log n)$ idő alatt megtalálja a leghosszabb futásidejű feladatot, és végül a 4. lépés miatt az egész folyamatot maximum $m$-szer a 2. lépéstől újrakezdi, ami így az eredeti listás ütemezés $O(n \cdot \log n)$ tagját nem változtatja, a 4. lépés miatt pedig az összes feladat közül meg kell keresni az aktuálisan leghamarabb végző processzort, maximum $m$-szer, amihez $O(n \cdot m \cdot \log m)$ időre van még szükség, azaz összesen $O(n \cdot \log n + n \cdot m \cdot \log m)$ a futásidő, ezt kellett bizonyítanunk. □

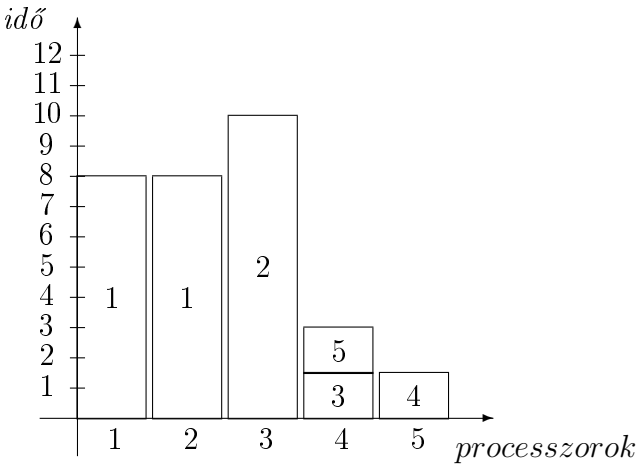
Nézzünk egy példát az algoritmus működésére, konkáv sebességfüggvények esetén. Legyen $m = n = 5$. Az egyes feladatok végrehajtási idejeit aszerint, hogy hány processzor dolgozik rajtuk (azaz a $\frac{p_j}{f_j(r)}$ hányadosokat) mutatja a 4.1. táblázat. A kezdeti LPT ütemezés a 4.2. ábra szerint alakul, minden feladat az aktuálisan legelőször felszabaduló processzorhoz kerül, most $m = n$ esetén ez azt jelenti, hogy minden feladat a 0 időpontban kezdődik. Az algoritmus lépésein végighaladva először kiválasztjuk a 2. lépésben az 1. feladatot, és $h$-t 12-re állítjuk. A 3. lépésben leellenőrizzük, hogy az aktuális befejezési idő egybeesik $h$-val, majd az 1. feladat processzorszámát 1-ről 2-re növeljük. A többi feladat új LPT ütemezésével az új befejezési idő 10 (4.3. ábra), ami jobb az eddiginél, visszatérhetünk a 2. lépésre. Innentől pedig az eddigi műveletsort ismételjük: a 2. feladat fog egy helyett két processzoron végrehajtódni, az új befejezési idő 8, ami ismét javulást jelent (4.4. ábra). Ezután ismét az 1. feladat processzorszámát növeljük, mely által kapott ütemezés $7,5$ ideig tart, ami újra jobb az előzőnél (4.5. ábra). Azonban ekkor a 2. lépés a 2. feladatot választja ki, mivel ennek a legnagyobb az aktuális végrehajtási ideje, azonban ez 7, míg $h = 7,5$, így a folyamat a 3. lépésről az 5.-re ugrik, és ott a 4.5 ábrán látható ütemezéssel leáll, így ez lesz az algoritmus által kapott processzorszétosztás.

| *processzorok* | | | | | |
|---|---|---|---|---|---|
| 1 | 12 | 10 | 1,5 | 1,5 | 1,5 |
| 2 | 8 | 7 | 1 | 1,5 | 1 |
| 3 | 6 | 6 | 1 | 1,5 | 1 |
| 4 | 5 | 6 | 1 | 1,5 | 1 |
| 5 | 5 | 6 | 1 | 1,5 | 1 |
| | 1. | 2. | 3. | 4. | 5. *feladatok* |

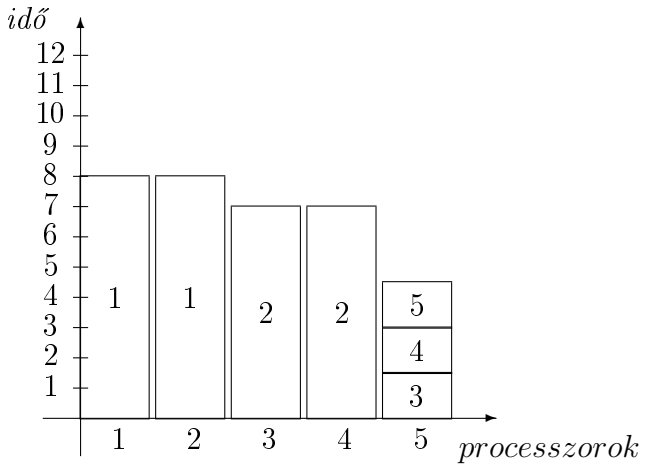
4.1. Végrehajtási idők a processzorszám szerint



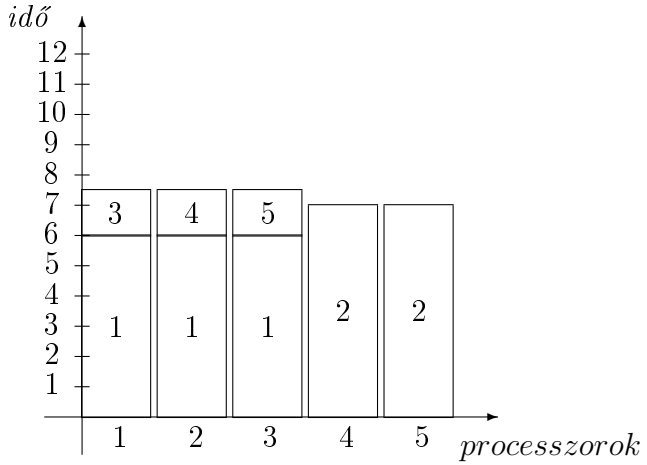

4.2. Kezdeti LPT ütemezés




4.3. Az ütemezés 2. állapota




4.4. Az ütemezés 3. állapota




4.5. A végső ütemezés

## 4.2. Az optimális ütemezés

A konkáv tulajdonságukat $O(mn)$ lépésben ellenőrizhetjük, ahogy a konvex esetben is tettük. Mivel az $f_j$-k konkávok és szakaszonként lineárisak, ezért az $U$ halmaz egy $n$-dimenziós konvex poliéder. Így $convU = U$ ebben az esetben.

### 4.2.1. A P-CNTN eset

Jelölje $\hat{f}_j(r)$ a szigorúan monoton növő részét az $f_j(r)$ függvénynek. Mivel $f_j(r)$ konkáv, ezért $\exists m_j \in \{1, 2, \dots m\}$, amire $\hat{f}_j(r) = f_j(r)$, ha $0 \leq r \leq m_j$, és $f_j(r) = f_j(m)$, ha $m_j \leq r \leq m$. Valamint az $f_j$ függvénynek nincs szakadási pontja az $(\hat{r}_j^0, m)$ intervallumon. Legyen $r^0$ a már definiált optimális erőforrás szétosztás. $X$ jelölje azoknak a $j$ feladatoknak a halmazát, melyekre $r_j^0 > m_j$. Könnyen belátható, hogy a következő $\hat{r}^0$ processzorszétosztás is optimális: $\hat{r}_j^0 = r_j^0$, ha $j \notin X$, és $\hat{r}_j^0 = m_j$, ha $j \in X$. Mivel az $\hat{f}_j$ függvény már injektív, ezért vehetjük az $\hat{f}_j^{-1}$ inverz függvényét. Legyenek az $\hat{f}(r) = u$ és az $\hat{f}^{-1}(u) = r$ egyenletekben szereplő $u$ és $v$ vektorok azok, melyekre $\hat{f}_j(r_j) = u_j$, $\forall j \in \{1, 2, \dots n\}$. Az optimális megoldás elkészítéséhez a 2.1. tételt használjuk. Az $u = \frac{p}{C}$ egyenes és a $U$ konvex burkának határának metszéspontja $(u^0)$ és a hozzá tartozó optimális processzorszétosztás $(r^0)$ az alább leírtak alapján található meg. Legyen $I$ azon pontok halmaza, melyek az $u_j = f_j(r_j) \mid j \in \{1, 2, \dots n\}, r_j \in \{1, 2, \dots m_j\}$ síkok és az $u = \frac{p}{C} \mid C > 0$ egyenes metszeteként állnak elő. Legyen $u^{min}$ az a pont, melyre a $C$ érték minimális. Ha $\min\left\{\frac{p_j}{f_j(m_j)} \mid j \in \{1, 2, \dots n\}\right\}$ érték a $j = j^*$ feladathoz tartozik, akkor ez az $u_j = f_{j^*}(m_{j^*})$ síknak és az $u = \frac{p}{C}$ egyenesnek a metszéspontja. Továbbá $\hat{f}^{-1}(u^{min} = r^{min})$ és $\sum_{j=1}^{n} r_j^{min} \leq m$, azaz $r^{min}$ egy optimális ütemezés, ahonnan könnyen megkaphatjuk az $r^0$ keresett ütemezést.

A megoldás további részében így feltehető, hogy $\sum_{j=1}^{n} r_j^{min} > m$. Legyen $\overline{u}$ egy olyan $I$-beli pont, melyre $\hat{f}^{-1}(\overline{u}) = \overline{r}$, valamint $\sum_{j=1}^{n} \overline{r}_j \geq m$, és nincs másik $I$-beli $u$ pont, amelyre $u_1 < \overline{u}_1$ és $\sum_{j=1}^{n} r_j \geq m$, ahol $r = \hat{f}^{-1}(u)$. Mivel $u^0$ és $\overline{u}$ ugyanazon az $u = \frac{p}{C}$ egyenesen fekszenek, és mivel $u^0$ közelebb van az origóhoz, mint $\overline{u}$, ezért $\forall n$-re: $u_j^0 \leq \overline{u}_j$, valamint $\hat{r}_j^0 \leq \overline{r}_j$. Ráadásul mivel nincs más $u \in I$ pont az $u^0$ és az $\overline{u}$ pontok között, ezért egyik $f_j$ függvénynek sincs töréspontja az $\hat{r}_j^0, \overline{r}_j$ intervallumon, ezért az $r_j^0, \overline{r}_j$ intervallumon sem. Ezért, és a régebben tárgyalt (3) egyenlet miatt

$$u_j^0 = f_j(r_j^0) = b_{j,\lceil \overline{r}_j \rceil} r_j^0 + d_{j,\lceil \overline{r}_j \rceil} = \frac{p_j}{C_{max}^0}, \tag{7}$$

$\forall j \in \{1, 2, ...n\}$. Ezt az egyenletet átrendezve kapjuk, hogy

$$r_j^0 = \frac{p_j}{b_{j,\lceil \overline{r}_j \rceil} \cdot Cmax^0} - \frac{d_{j,\lceil \overline{r}_j \rceil}}{b_{j,\lceil \overline{r}_j \rceil}}, \tag{8}$$

$\forall j \in \{1, 2, \dots n\}$. Mivel az $u^0$ pont $U$ konvex burkának határán fekszik, ezért $\sum_{j=1}^{n} r_j^0 = m$, ezt hozzávéve a (8) egyenlethez:

$$C_{max}^0 = \frac{\sum_{j=1}^{n} \frac{p_j}{b_{j,\lceil \overline{r}_j \rceil}}}{m + \sum_{j=1}^{n} \frac{d_{j,\lceil \overline{r}_j \rceil}}{b_{j,\lceil \overline{r}_j \rceil}}}, \tag{9}$$

valamint az $u^0$ és $r^0$ értékeket a (7) és (8) egyenletekből számolhatjuk. Mivel $p_j = u_j^0 C_{max}^0$ $\forall j \in \{1, 2, \dots n\}$ és $\sum_{j=1}^{n} r_j^0 = m$, ezért ezzel egy olyan optimális ütemezést kapunk a P-CNTN problémára, ahol minden feladat a $(0, C_{max}^0)$ intervallumon belül fut, és a $j$ feladat $r_j^0$ processzort használ $\forall j \in \{1, 2, \dots n\}$.

Az eddigiekből látszik, hogy ahhoz, hogy kiszámoljuk az $r^0$ és $u^0$ vektorokat, ahhoz meg kell találni az $\overline{r}$ processzorszétosztást. Ehhez először minden $j$ feladathoz végezzük el a következő felezéses kereső algoritmust $\overline{u}^j := \Phi$, $l = 0$ és $t = m_j$ kezdőértékekkel:

**BS felezéses kereső algoritmus**

1. Az algoritmus első iterációjában $r = m_j$-re, minden további iterációjában $r = \left\lfloor \frac{l+t}{2} \right\rfloor$-re számoljuk ki az $u^{(j)} = u_1^{(j)}, u_2^{(j)}, \dots u_n^{(j)}$ metszéspontját az $u_j = f_j(r)$ síknak és az $u = \frac{p}{C}$ egyenesnek, ahol $u_j^{(j)} = f_j(r)$ és $u_i^{(j)} = \frac{p_i \cdot f_j(r)}{p_j}$, ahol $i \neq j$.

2. Számoljuk ki az $r^{(j)} = \hat{f}^{-1}(u\ (j))$ értékeket a következő módon: először $r_j^{(j)} := r$. Minden $i \neq j$-re keressük meg az $s_i \in \{1, 2, \dots m_j\}$ töréspontot, melyre $f_i(s_i - 1) < u_i^{(j)} \leq f_i(s_i)$.

3. Ha $\sum_{i=1}^{n} r_i^{(j)} \geq m$, akkor $\overline{u}^{(j)} := u^{(j)}, t := r$, és $l$-et hagyjuk változatlanul. Ha $\sum_{i=1}^{n} r_i^{(j)} < m$, akkor $l := r$ és $t$-t hagyjuk változatlanul.

4. Ha $t - l < 1$, akkor ugorjunk az 5. lépésre. Ha nem, akkor lépjünk vissza az 1. lépésre.

5. A keresett $\overline{r}$ ütemezést a következő módon kapjuk: $\overline{r} = \hat{f}^{-1}(\overline{u})$, ahol $\overline{u}$ koordinátáit az $\overline{u}_l = \min\left\{\overline{u}_l^{(j)} \mid \overline{u}^{(j)} \neq \Phi, \forall j \in \{1, 2, \dots n\}\right\}$ egyenlőségből kapjuk.

**4.13. Tétel.** *A P-CNTN feladat $O(n \cdot \max\left\{m, n \log^2 m\right\})$ idő alatt megoldható [23].*

*Bizonyítás:* A BS felezéses kereső algoritmus iterációszáma nem haladja meg $O(\log m)$-et, mert az $r$ érték mindig az $l$ és $t$ értékek átlaga, amelyek különbsége a 3. lépés szabálya szerint így minden lépésben feleződik, 1 $O(\log m_j)$ lépés alatt lesz. Az 1. lépés $O(n)$ idő alatt megvalósítható, a 2. lépésnél $r_j^{(j)}$ megtalálásához felező algoritmust használva a $\{0, 1, \dots m_j\}$ értékeken $O(n \cdot \log m)$ idő szükséges.

Nem konstans lépésszámot az 5. lépés végrehajtása igényel, mivel a keresett $\overline{r}$ kiszámításához mind az $n$ koordinátájára el kell végeznünk az algoritmust, ezzel eddig összesen $O(n^2 \cdot \log^2 m)$ időnél tartunk. Valamint szükségünk van $O(m \cdot n)$ időre a szakaszonként lineáris $f_j$ függvények együtthatóinak kiszámolásához, így a két idő közül a nagyobbikra van szükségünk az algoritmus elvégzéséhez, ezzel beláttuk a tételt. □

### 4.2.2. A P-DSCR eset

Ebben az esetben a feladatokon csak egész számú processzor dolgozhat. Bizonyított, hogy ekkor $C^*_{max} = C^0_{max}$, azaz a megoldás összefügg a P-CNTN eset megoldásával. Ez $n = 2$ és $n = 3$ esetekben azt jelenti, hogy konstans idő alatt megkaphatjuk a P-DSCR feladat optimális ütemezését a P-CNTN feladat megoldásának $(r^0, u^0, C^0_{max})$ ismeretében.

**Az n=2 eset:**

Ha $r^0$ egész koordinátájú, akkor meg is van az ütemezés, egész idő alatt $r^0_1$ processzort dolgoztatunk az első, és $r^0_2$ processzort a második feladaton. Ha $r^0$ nem egész koordinátájú, akkor legyen $r^{(1)} = (\lfloor r^0_1 \rfloor, \lceil r^0_2 \rceil)$ és legyen $r^{(2)} = (\lceil r^0_1 \rceil, \lfloor r^0_2 \rfloor)$. Ekkor $r^0$ az $(r^{(1)}, r^{(2)})$ szakaszon van P-CNTN szakaszos linearitása miatt. Ebben az esetben $r^0 = \lambda \cdot r^{(1)} + (1 - \lambda) \cdot r^{(2)}$, ahol $\lambda = \lceil r^0_1 \rceil - r^0_1 = r^0_2 - \lfloor r^0_2 \rfloor$. Mivel se az $f_1$, se az $f_2$ függvénynek nincs töréspontja $r^{(1)}$ és $r^{(2)}$ között, ezért $u^0 = \lambda \cdot f(r^{(1)}) + (1 - \lambda) \cdot f(r^{(2)})$. P-DSCR esetén az optimális ütemezésnek két futási intervallumhossza van, $\Delta_1 = \lambda \cdot C^0_{max}$ és $\Delta_2 = (1-\lambda) \cdot C^0_{max}$. A $\Delta_1$ intervallum alatt az első és második feladathoz rendelt processzorszám így rendre $\lfloor r^0_1 \rfloor$ és $\lceil r^0_2 \rceil$, a $\Delta_2$ intervallumon pedig $\lceil r^0_1 \rceil$ és $\lfloor r^0_2 \rfloor$

**Az n=3 eset:**

Ekkor 3 esetet különböztetünk meg:

1. $r^0$ $\forall$ koordinátája egész,
2. $r^0$ egyik koordinátája egész, a másik kettő nem,
3. $r^0$ egyik koordinátája sem egész.

Fontos megjegyezni, hogy olyan eset nincs, amikor $r^0$ pontosan két koordinátája egész, mert ha van két egész koordinátája, abból következik, hogy a harmadik is az.

Az 1. eset triviális, hiszen ekkor a $[0, C^0_{max}]$ futási intervallumon kell végrehajtanunk mindhárom feladatot, $r^0_j$ processzort használva a $j$ feladathoz, $j \in \{1, 2, 3\}$.

A 2. esetben feltehetjük, hogy csak $r_j^0$ egész. Ekkor könnyen látható, hogy $r^0$ az $(r^{(1)}, r^{(2)})$ intervallumon van, ahol $r^{(1)} = (r_1^0, \lfloor r_2^0 \rfloor, m - r_1^0 - \lfloor r_2^0 \rfloor)$ és $r^{(2)} = (r_1^0, \lceil r_2^0 \rceil, m - r_1^0 - \lceil r_2^0 \rceil)$. Azaz $r^0 = \lambda \cdot r^{(1)} + (1-\lambda) \cdot r^{(2)}$, ahol $\lambda = \lceil r_2^0 \rceil - r_2^0$.

Az optimális ütemezésnél 2 futási intervallumot kapunk $\Delta_1 = \lambda \cdot C_{max}^0$ és $\Delta_2 = (1-\lambda) \cdot C_{max}^0$ hosszokkal. Az első feladathoz tartozó processzorszám mindkét esetben $r_1^0$. A $\Delta_1$ intervallumban a 2. feladat processzorszáma $\lfloor r_2^0 \rfloor$, a 3. feladaté $m - r_1^0 - \lfloor r_2^0 \rfloor$, a $\Delta_2$ intervallumban pedig $\lceil r_2^0 \rceil$ illetve $m - r_1^0 - \lceil r_2^0 \rceil$.

A 3. esetben $r^0$ egy $r^{(1)}$, $r^{(2)}$, $r^{(3)}$ csúcsokkal rendelkező háromszög belsejében van, ahol a csúcsok minden koordinátája egész, és az eddigi egészrészek által határolt, valamint a processzor kihasználás maximális, azaz:

1. $r_j^{(i)} \in \{\lfloor r_j^0 \rfloor, \lceil r_j^0 \rceil\}$ $i, j \in \{1,2,3\}$ és
2. $r_1^{(i)} + r_2^{(i)} + r_3^{(i)} = m$, $i \in \{1,2,3\}$.

Ezt a két feltételt egyszerre a következő négy pont teljesítheti:

$$a = (\lceil r_1^0 \rceil, \lceil r_2^0 \rceil, m - \lceil r_1^0 \rceil - \lceil r_2^0 \rceil),$$

$$b = (\lfloor r_1^0 \rfloor, \lfloor r_2^0 \rfloor, m - \lfloor r_1^0 \rfloor - \lfloor r_2^0 \rfloor),$$

$$c = (\lceil r_1^0 \rceil, \lfloor r_2^0 \rfloor, m - \lceil r_1^0 \rceil - \lfloor r_2^0 \rfloor),$$

$$d = (\lfloor r_1^0 \rfloor, \lceil r_2^0 \rceil, m - \lfloor r_1^0 \rfloor - \lceil r_2^0 \rceil).$$

Viszont $a$ és $b$ közül egyszerre csak az egyik elégítheti ki a feltételeket, mert ha mindkettő igaz lenne, akkor $r_3^0$ alsó és felső egészrésze között 2 lenne a különbség, ami nem lehetséges. Így ha $\lceil r_1^0 \rceil + \lceil r_2^0 \rceil + \lfloor r_3^0 \rfloor = m$, akkor $r^{(1)} := a$, ellenkező esetben $r^{(1)} := b$. Az $r^0$-t tartalmazó háromszög másik két pontja pedig $r^{(2)} = c$ és $r^{(3)} = d$. Emiatt $\exists\, \lambda_1 > 0$, $\lambda_2 > 0$, $\lambda_3 > 0$, melyekre $\sum_{i=1}^{3} \lambda_i = 1$ és $r_j^0 = \lambda_1 \cdot r_j^{(1)} + \lambda_2 \cdot r_j^{(2)} + \lambda_3 \cdot r_j^{(3)}$, $j \in \{1,2,3\}$. Ez a három ismeretlent tartalmazó, három lineáris egyenletből álló egyenletrendszer konstans idő alatt megoldható. Így végeredményként három futási intervallumot kaptunk, melyek $\Delta_i = \lambda_i \cdot C_{max}^0$, $i \in \{1,2,3\}$, és $\Delta_i$ intervallumban a $j$ feladathoz tartozó processzorszám $r_j^{(i)}$, ahol $i, j \in \{1,2,3\}$. Ezzel konstans idő alatt megoldottuk a P-DSCR problémát $n = 3$ esetben is.

### 4.2.3. Példa a P-DSCR eset megoldására n=3 esetén

Nézzük meg a gyakorlatban egy konkrét feladaton $n = 3$ esetén a megoldás menetét!

A három feladatra $p_1 := 7$, $p_2 := 5$, $p_3 := 2$. A processzorszám legyen 5. A végrehajtási sebesség függvény minden feladatra legyen $g_j(r) = \sqrt{r}$, amely valóban konkáv. Ekkor az algoritmusunk:

1. Számítsuk ki a szakaszonként lineáris $f_j(r)$ függvények együtthatóit az $f_j(r) = f(r) = b_{\lceil r \rceil} \cdot r + d_{\lceil r \rceil}$ egyenletekből, ahol $0 \leq r \leq m, j \in \{1, 2, \ldots n\}$. Ezek a $(0,0)$, $(1,1)$, $(2,\sqrt{2})$, $(3,\sqrt{3})$, $(4,2)$, $(5,\sqrt{5})$ pontok közötti szakaszok egyenleteinek együtthatóit jelentik, azaz $b_1 = 1$, $d_1 = 0$, $b_2 = \sqrt{2} - 1$, $d_2 = 2 - \sqrt{2}$, $b_3 = \sqrt{3} - \sqrt{2}$, $d_3 = 3\sqrt{2} - 2\sqrt{3}$, $b_4 = 2 - \sqrt{3}$, $d_4 = 4\sqrt{3} - 6$, $b_5 = \sqrt{5} - 2$, $d_5 = 10 - 4\sqrt{5}$.

2. Keressük meg azt az $\overline{r}$ processzorszétosztást, melyre $f(\overline{r}) \in I$, $\sum_{j=1}^{n} \overline{r}_j \geq m$ és $\nexists u \in I \backslash \{f(\overline{r})\}$, amire $u_1 < \overline{u}_1$ és $\sum_{j=1}^{n} r_j \geq m$, ahol $r = f^{-1}(u)$. A kapott érték: $r = (3, 1.57, 0.49)$.

3. Számoljuk ki $C^*_{max}$-ot:

$$C^*_{max} = C^0_{max} = \frac{\sum_{j=1}^{n} \frac{p_j}{b_{\lceil \overline{r}_j \rceil}}}{m + \sum_{j=1}^{n} \frac{d_{\lceil \overline{r}_j \rceil}}{b_{\lceil \overline{r}_j \rceil}}} = 4,07.$$

   Az $r^0$ kiszámolására használjuk a (8) formulát, ezzel $r^0 = (2.96, 1.55, 0.49)$.

4. Mivel $r^0$ egyik koordinátája sem egész, ezért a 3. eset egyenleteit kielégítő $r^{(i)}$ pontokat kell kiszámolnunk, amik: $r^{(1)} = (2, 2, 1)$, $r^{(1)} = (3, 1, 1)$ és $r^{(1)} = (3, 2, 0)$.

5. Az ezekkel az együtthatókkal kapott lineáris egyenletrendszer:

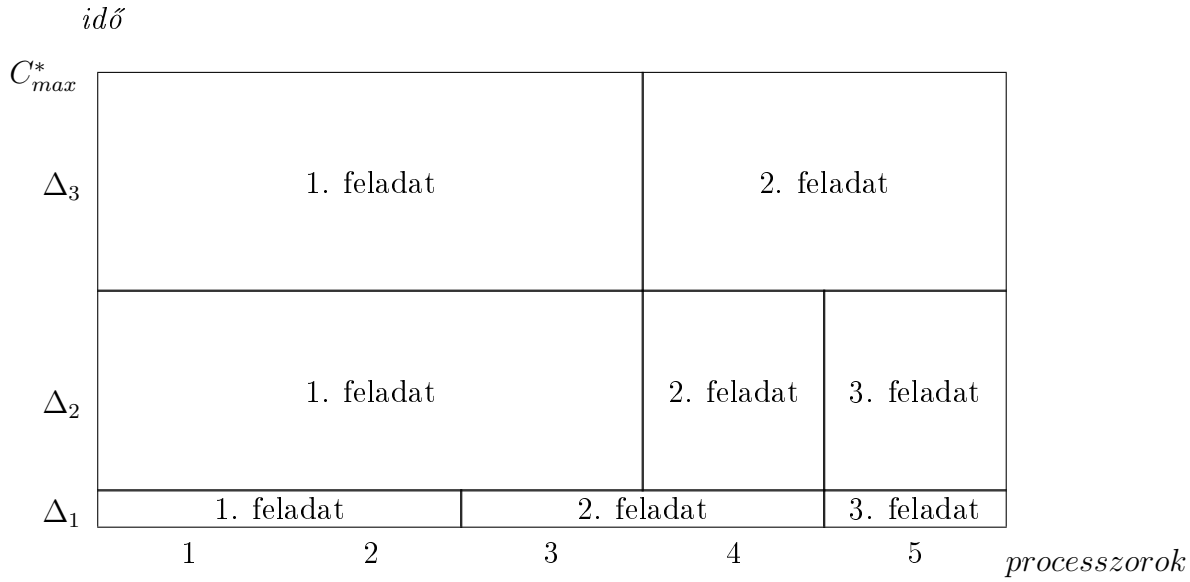
$$2.96 = 2\lambda_1 + 3\lambda_2 + 3\lambda_3,$$

$$1.55 = 2\lambda_1 + \lambda_2 + 2\lambda_3,$$

$$0.49 = \lambda_1 + \lambda_2,$$

   amiből $\lambda_1 = 0.04$, $\lambda_2 = 0.45$ és $\lambda_3 = 0.51$.

6. Végül számoljuk ki a keresett $\Delta_i$ értékeket: $\Delta_1 = \lambda_1 C^0_{max} = 0.17$, $\Delta_2 = \lambda_2 C^0_{max} = 1.83$, $\Delta_3 = \lambda_3 C^0_{max} = 2.07$. Ezt az optimális ütemezést ábrázolja a 4.6. grafikon.




4.6. Az optimális ütemezés a P-DSCR esetben

# 5. Nyitott problémák

Akár az MPTS, akár az NPTS esetre csak a független feladatokra vizsgáltunk megoldási módszereket. A probléma kiterjeszthető egymástól függő feladatokra is. Ekkor minden feladathoz megadhatunk más feladatokat, melyek ennek a feladatnak előfeltételei, csak ezek elvégzése után kezdhetjük el a feladatot (természetesen ezt megoldani csak az előfeltételek körmentes megadása esetén lehetséges).

Ennek további nehezítése az ún. on-line megelőzési feltételes modell. Ekkor bizonyos feladatok hosszáról, és egyáltalán létezéséről csak akkor szerzünk tudomást, amikor már az összes előfeltételét elvégeztük. Mint korábban említettük Feldmann, Sgall, Teng és Kao foglalkozott behatóan ezzel az esettel, de bőven találhatunk még megoldatlan problémákat ebben a témakörben.

Az összes problémánál csak egy fajta erőforrást használtunk, a processzorokat. Ennek egy kiterjesztése az az eset, ha más erőforrásokat is igénybe vehetünk, vagy megszorításokat alkalmazunk rájuk, például változtatható memória követelményekkel.

Végezetül szeretném megköszönni témavezetőmnek, Kis Tamásnak a segítségét és hasznos tanácsait, amik nélkül nem készülhetett volna el a dolgozatom ebben a formájában.

# Hivatkozások


[1] B. Baker, E. Coffman and R. Rivest: *Orthogonal packing in two dimensions*, SIAM journal on computing 9(4), 846-855., 1980.

[2] K. P. Belkhale and P. Banerjee: *Approximate algorithms for the partitionable independent task scheduling problem*, Proceedings of the 1990 International conference on parallel processing vol. I, 72-75., 1990.

[3] K. P. Belkhale and P. Banerjee: *A parallel algorithm for hierarchical circuit extraction*, International Conference on computer aided design, 236-239., 1990.

[4] J. Błażewicz, M. Machowiak, J. Węglarz, M. Kovalyov and D. Trystram: *Scheduling malleable tasks on parallel processors to minimize the makespan*, Annals of Operations Research, Kluwer Academic Publishers 129(16), 65-80., 2004.

[5] J. Błażewicz, M. Drabowski and J. Węglarz: *Scheduling multiprocessor tasks to minimize schedule lenght*, IEEE Transactions on computing 35, 389-393., 1986.

[6] J. Chen and C. Y. Lee: *General multiprocessor task scheduling*, Naval research logistics 46(1), 57-74., 1999.

[7] E. G. Coffman Jr., M. R. Garey, D. S. Johnson and R. E. Tarjan: *Performance bounds for level-oriented two-dimensional packing algorithms*, SIAM journal on computing 9(4), 808-826., 1980.

[8] J. Dongarra, L. Duff, D. Danny, C. Sorensen and H. van der Vorst: *Numerical linearalgebra for high performance computers*, Philadelphia, Society for Industrial and Applied Mathematics, 1999.

[9] M. Drozdowski: *Scheduling multiprocessor tasks - an overview*, European journal of operational research 94, 215-230., 1996.

[10] J. Du and J. Leung: *Complexity of scheduling parallel task systems*, SIAM journal on discrete mathematics 2(4), 473-487., 1989.

[11] A. Feldmann, M.-Y. Kao, J. Sgall and S.-H. Teng: *Optimal online scheduling of parallel jobs with dependencies*, Proceedings of the twenty-fifth annual ACM symposium on the theory of computing, 642-651., 1993.

[12] A. Feldmann, J. Sgall and S.-H. Teng: *Dynamic scheduling on parallel machines*, 32nd annual symposium on foundations of computer science, 111-120-, 1991.

[13] M. R. Garey and D. S. Johnson: *Computers and intractability, A guide to the theory of NP-completeness*, New York, W. H. Freeman and Company, 1979.

[14] R. L. Graham: *Bounds of multiprocessing timing anomalies*, SIAM journal on applied mathematics vol. 17, 263-269., 1969.

[15] R. L. Graham, E. L. Lawler, J. K. Lenstra and A. H. G. Rinnooy Kan: *Optimalization and approximation in deterministic sequencing and scheduling: A survey*, Annals of discrete mathematics vol. 5, 287-326., 1979.

[16] R. Krishnamurti and E. Ma: *The processor partitioning problem in special-purpose partitionable systems*, Proceedings of the 1988 International conference on parallel processing vol. I, 434-443., 1988.

[17] A. Lodi, S. Martello and M. Monaci: *Two-dimensional packing problems: a survey*, European journal of operational research 141(2), 241-252., 2002.

[18] W. T. Ludwig: *Algorithms for scheduling malleable and nonmalleable parallel tasks*, PhD. thesis, University of Wisconsin-Madison, Department of Computer Science, 1995.

[19] W. T. Ludwig, and P. Tiwari: *Scheduling Malleable and Nonmalleable Parallel Tasks*, Proceedings of the fifth annual ACM-SIAM symposium on Discrete algorithms, 167-176., 1994.

[20] G. Mounie, C. Rapine and D. Trystram: *Efficent approximation algorithms for scheduling malleable tasks*, Proceedings of the eleventh annual ACM symposium on Parallel algorithms and architectures, 23-32., 1999.

[21] G. N. S. Prasanna and B. R. Musicus: *The optimal control approach to generalized multiprocessor scheduling*, Algorithmica, 1995.

[22] D. Sleator: *A 2,5 times optimal algorithm for packing in two dimensions*, Information processing letters 10(1), 37-40., 1980.

[23] D. F. Stanat and D. F. McAllister: *Discrete mathematics in computer science*, Englewood Cliffs, Prentice-Hall, 1977.

[24] J. Turek, J. Wolf and P. Yu: *Approximate algorithms for scheduling parallelizable tasks*, 4th Annual ACM symposium on parallel algorithms and architectures, 323-332., 1992.

[25] J. Węglarz: *Project scheduling with continously-divisible, doubly constrained resources*, Management Science 27, 1040-1052., 1981.

[26] J. Węglarz: *Modelling and controll of dynamic resource allocation project scheduling systems*, S. G. Tzafestas, Optimization and control of dynamic operational research models, Amstredam: North-Holland, 1982.