Eötvös Loránd Tudományegyetem

Természettudományi Kar

Páros munkák ütemezése

Markó Anna Erzsébet

Alkalmazott matematikus MSc

Operációkutatás specializáció

Belső témavezető: **Kis Tamás**

Operációkutatási Tanszék, ELTE TTK

Mérnöki és Üzleti Intelligencia Kutatólaboratórium, HUN-REN SZTAKI

Külső témavezető: **Györgyi Péter**

Mérnöki és Üzleti Intelligencia Kutatólaboratórium, HUN-REN SZTAKI

Budapest, 2025

Köszönetnyilvánítás

Szeretném megköszönni témavezetőmnek, Györgyi Péternek, a szakdolgozat elkészítésében nyújtott segítséget.

Tartalomjegyzék

1. Bevezetés

A páros munkák ütemezése feladatot Shapiro vezette be 1980-ban [13] egy gyakorlati probléma által. Egy radarkövetési rendszert modellezett, amely meghatározott periódusonként impulzusokat bocsát ki, majd azok visszaverődését fogadja. Az adás és a vétel nem történhet ugyanazon időben. Ezenkívül minden impulzusra adott, hogy pontosan mennyi idő múlva érkezik a visszavert jel. Ez a probléma a következőképpen modellezhető: ütemezzünk két részből álló munkákat, amelyek feladatrészeinek feldolgozása között rögzített időnek kell eltelnie, olyan gépeken, amelyek egyszerre egy munkán tudnak dolgozni.

A problémát elsősorban a legkésőbbi befejezési idő minimalizálásának szempontjából vizsgálták. Ennek a feladatnak számos gyakorlati alkalmazása van: a már említett radarkövetési rendszereken túl alkalmazzák például torpedók teljesítményének javítására [14], vegyipari termelésnél [1] vagy betegek kemoterápiás kezelésekhez való beosztására [11].

A probléma ezen célfüggvény melletti komplexitását átfogóan először Orman és Potts vizsgálta 1997-ben [12], akik meghatározták számos variáns komplexitását. Ezen eredmények sorát az elmúlt évtizedekben mások is bővítették [5],[8]. Bár a feladat számos variánsa NP-nehéz még egészen speciális esetekben is, több változatra is sikerült hatékony approximációs algoritmusokat kidolgozni [1], [2].

Az elmúlt években a kutatások fókusza más célfüggvények vizsgálatára is kiterjedt. Ezek közül az egyik a befejezési idők összegének minimalizálása. A feladat komplexitásáról átfogó képet adott Chen és Zhang 2021-es cikkükben [6]. Ezen eredmények sorát tovább bővítette Fischer és Györgyi 2023-ban [8], akik több variánsra adtak approximációs algoritmust is.

A szakdolgozat során páros munkák egygépes ütemezését vizsgáljuk, nagyrészt a befejezési idők összegének minimalizálására koncentrálva, ezen belül is különös tekintettel a $1|Coup - Task(a_j = a, b_j = b)|\sum C_j$ és a $1|Coup - Task(l_j = l)|\sum C_j$ variánsra.

Az első fejezetben definiáljuk a feladatot, bevezetjük a szakdolgozat során használt jelöléseket, definíciókat.

A második fejezet a problémakör komplexitását járja körül a legkésőbbi befejezési idő minimalizálása, illetve a befejezési idők összegének minimalizálása mellett. Belátjuk a $1|Coup - Task(a_j = a, b_j = b)|\sum C_j$ és a $1|Coup - Task(l_j = l)|\sum C_j$ feladatok

NP-nehézségét Chen és Zhang munkájára támaszkodva [6].

A harmadik fejezetben egészértékű programozási modelleket ismertetünk, amiket a későbbiek során felhasználunk a már említett variánsok elemzésére.

A negyedik és ötödik fejezet során vizsgáljuk a matematikai programozási modellek hatékonyságát, approximációs algoritmusok gyakorlatban való teljesítményét, illetve lehetséges javításait.

2. A feladat definiállása

Egygépes ütemezési problémákban adottak munkák, amelyeket egy gépen akarunk ütemezni különböző feltételek mellett valamilyen célfüggvényt optimalizálva. A feladat tehát meghatározni egy ütemezést, ami megadja az összes munkára, hogy mikor kell elkezdeni a feldolgozását. A gép egyszerre egy munkán tud dolgozni.

Általánosan ütemezési problémák leírására az $\alpha|\beta|\gamma$ jelölést használjuk, ahol α jelöli a gépek számát, illetve típusát, β jelöli a munkákra vonatkozó korlátokat, γ pedig a célfüggvényt. Ennek megfelelően egygépes feladatok leírására az $1|\beta|\gamma$ jelölés használatos.

A szakdolgozat során a következő jelöléseket használjuk:

- n - a munkák száma
- $J = \{j_1, j_2, \ldots, j_n\}$ - a munkák halmaza
- p_j - a j munka feldolgozási ideje
- S_j - a j munka feldolgozása kezdetének ideje egy adott ütemezésben
- C_j - a j munka feldolgozása befejezésének ideje egy adott ütemezésben
- σ - az $S_1, S_2 \ldots, S_n$ által meghatározott ütemezés
- C_{max} - egy adott ütemezésben utoljára feldolgozott munka befejezési ideje - az ütemezés befejezési ideje.

Ha egy ütemezés optimális adott célfüggvény szerint, az ütemezést leíró értékeket a jobb felső indexben *opt*-tal jelöljük: S_j^{opt}, C_j^{opt}, σ^{opt}.

Egy ütemezés során - a $[0, C_{max}]$ intervallumon belül - azokat az időintervallumokat, amikor a gép nem dolgozik, *holtidők*nek nevezzük.

A szakdolgozat során kétfajta célfüggvénnyel találkozunk, ezek a C_{max} és a $\sum C_j$. Ha a célfüggvény az előbbi, akkor a *legkésőbbi befejezési idő minimalizálás*áról, míg ha az utóbbi, akkor a *befejezési idők összegének minimalizálás*áról beszélünk.

Az ütemezési feladatok egy speciális típusa a *páros munkák ütemezése*. Ebben a feladatban két feladatrészből álló munkákat akarunk ütemezni úgy, hogy minden feladatrészpárra adott, hogy a feldolgozásaik között pontosan mennyi időnek kell eltelnie. A feladattípushoz kapcsolódó jelölések:

- a_j - a j munka elsőnek feldolgozandó feladatrésze. Használjuk ennek feldolgozási idejére is.
- b_j - a j munka másodiknak feldolgozandó feladatrésze. Használjuk ennek feldolgozási idejére is.
- l_j - a j munka várakozási ideje - ennyi időnek kell eltelnie a két feladatrész feldolgozása között

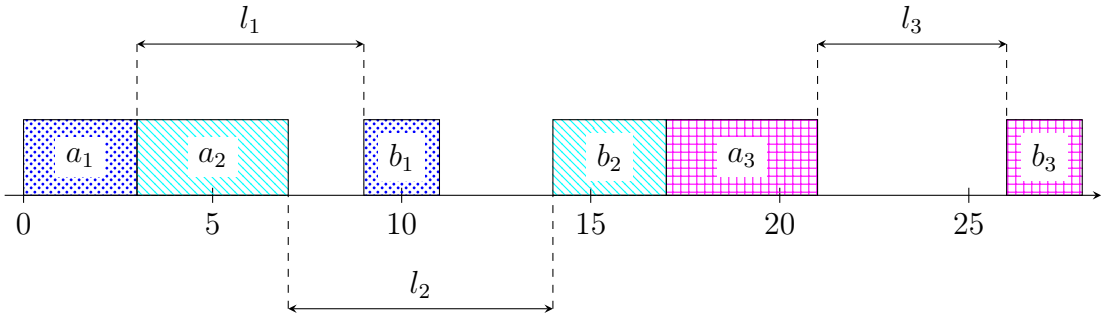
Az egygépes páros munkák ütemezése feladatot általánosan az $1|Coup - Task|\gamma$ hármassal jelöljük. Amennyiben a feladatot az a_j, l_j és b_j paraméterek valamilyen megkötése mellett vizsgáljuk, ezt a $Coup - Task$ után zárójelben jelöljük.

2.1. Példa. Nézzünk egy σ megengedett ütemezést a következő értékek mellett és számoljuk ki a C_{max} és $\sum C_j$ célfüggvényértékeket.

j	a_j	l_j	b_j
1	3	6	2
2	4	7	3
3	4	5	2

1. táblázat. Példa feladat paraméterei

Ütemezzük sorban a munkákat, először az 1-t, 2-t, majd a 3-t, a lehető leghamarabb. Tehát legyen $S_1 = 0$. $a_2 < l_1$ és $b_1 < l_2$, így a_2 ütemezhető az első munka várakozási idejében. Az is igaz, hogy $l_1 + b_1 < a_2 + l_2$, ezért a_2-t ütemezhetjük rögtön az a_1 feldolgozása után: $S_2 = a_1 = 3$. $a_1 + l_1 + b_1 = 11$, így a b_1 feldolgozása 11-kor ér véget. A b_2 feldolgozása $S_2 + a_2 + l_2 = 14$ -kor kezdődik. Mivel $a_3 > 3$, így nem ütemezhető a b_1 és b_2 feldolgozása közötti holtidőben. Ugyanígy nem ütemezhető a_2 és b_1 között sem. Tehát leghamarabb a b_2 feldolgozása után ütemezhető, így $S_3 = S_2 + a_3 + l_2 + b_2 = 17$.



1. ábra. Egy megengedett ütemezés.

Nézzük, hogyan alakulnak a célfüggvényértékek ezen ütemezés mellett:

$C_{max} = C_3 = S_3 + a_3 + l_3 + b_3 = 28$
$\sum C_j = C_1 + C_2 + C_3 = S_1 + S_2 + S_3 + \sum_{j=1}^{3}(a_j + l_j + b_j) = 56$

Tehát egy megengedett ütemezés a három munkára: $\sigma = (S_1, S_2, S_3) = (0, 3, 17)$, ami mellett a legkésőbbi befejezési idő 28, míg a befejezési idők összege 56.

Vezessünk be néhány definíciót, amit a továbbiakban használni fogunk.

A munka *hossza* az az idő, ami a feldolgozásának kezdetétől a befejezéséig szükséges. Egy j munka hossza tehát $a_j + l_j + b_j$.

Két munkát *szigorúan egymás után ütemezett*nek hívunk, ha a hamarabb ütemezett munka befejezése után kezdődik a később ütemezett munka feldolgozása.

Azt mondjuk, két munka *egymásba fonódik*, ha nem szigorúan egymás után vannak ütemezve.

Két munka *közvetlenül egymás után ütemezett*, ha feldolgozásaik kezdetének idejei között nem kezdődik meg új munka feldolgozása.

Egy ütemezésen belül *blokk*ot alkotnak azok a $(j_1, j_2, \ldots, j_k)$ közvetlenül egymás után ütemezett munkák, amelyekre igaz, hogy $\forall(j_h, j_{h+1})$, $h \in \{1, \ldots, k-1\}$ párra igaz, hogy a j_{h+1} munka feldolgozása hamarabb elkezdődik, mint hogy a j_h munka feldolgozása befejeződött volna.

3. A feladat komplexitása

Ez a fejezet a feladat különböző variánsainak komplexitását mutatja be. Először összefoglaljuk a $1|Coup - Task|C_{max}$ feladat komplexitásáról szóló legfontosabb eredményeket. Ezt követően áttérünk a $1|Coup - Task|\sum C_j$ feladatra, ahol a komplexitási eredmények összefoglalása mellett a dolgozat 5. és 6. fejezetében tárgyalt $1|Coup - Task(a_j = a, b_j = b)|\sum C_j$ és $1|Coup - Task(l_j = l)|\sum C_j$ feladatok NP-nehézségére adunk bizonyítást.

3.1. A legkésőbbi befejezési idő minimalizálása

Orman és Potts 1997-ben belátták, hogy az $1|Coup - Task(a_j = l_j = b_j)|C_{max}$, $1|Coup - Task(l_j = l, b_j = b)|C_{max}$ és $1|Coup - Task(a_j = b_j = p)|C_{max}$ feladatok NP nehezek. A $1|Coup - Task(a_j = b_j = p)|C_{max}$ feladat NP-nehézségéből következik az általánosabb $1|Coup - Task(a_j = a, b_j = b)|C_{max}$ feladat NP-nehézsége. Az $1|Coup - Task(l_j = l, b_j = b)|C_{max}$ feladat NP-nehézségéből pedig a lenti állítás segítségével következik az $1|Coup - Task(a_j = a, l_j = l)|C_{max}$ feladat NP-nehézsége.

3.1. Állítás. Az $1|Coup - Task|C_{max}$ feladat minden példányának tetszőleges ütemezéséből, ahol a paraméterek a_j, l_j, b_j, polinomidőben előállítható egy megengedett ütemezés a feladat azon példányára, ahol minden j-re felcseréljük az a_j és b_j értékeket, úgy, hogy a célfüggvény értéke mindkét esetben ugyanaz.

Bizonyítás. Vegyünk egy megengedett ütemezést a $1|Coup - Task(a_j, l_j, b_j)|C_{max}$ feladatra C_{max} legnagyobb befejezési idővel. Ekkor az az ütemezés, ahol az j munka kezdési ideje $C_{max} - C_j$, megengedett a $1|Coup - Task(b_j, l_j, a_j)|C_{max}$ feladatra és a legnagyobb befejezési idő szintén C_{max}. □

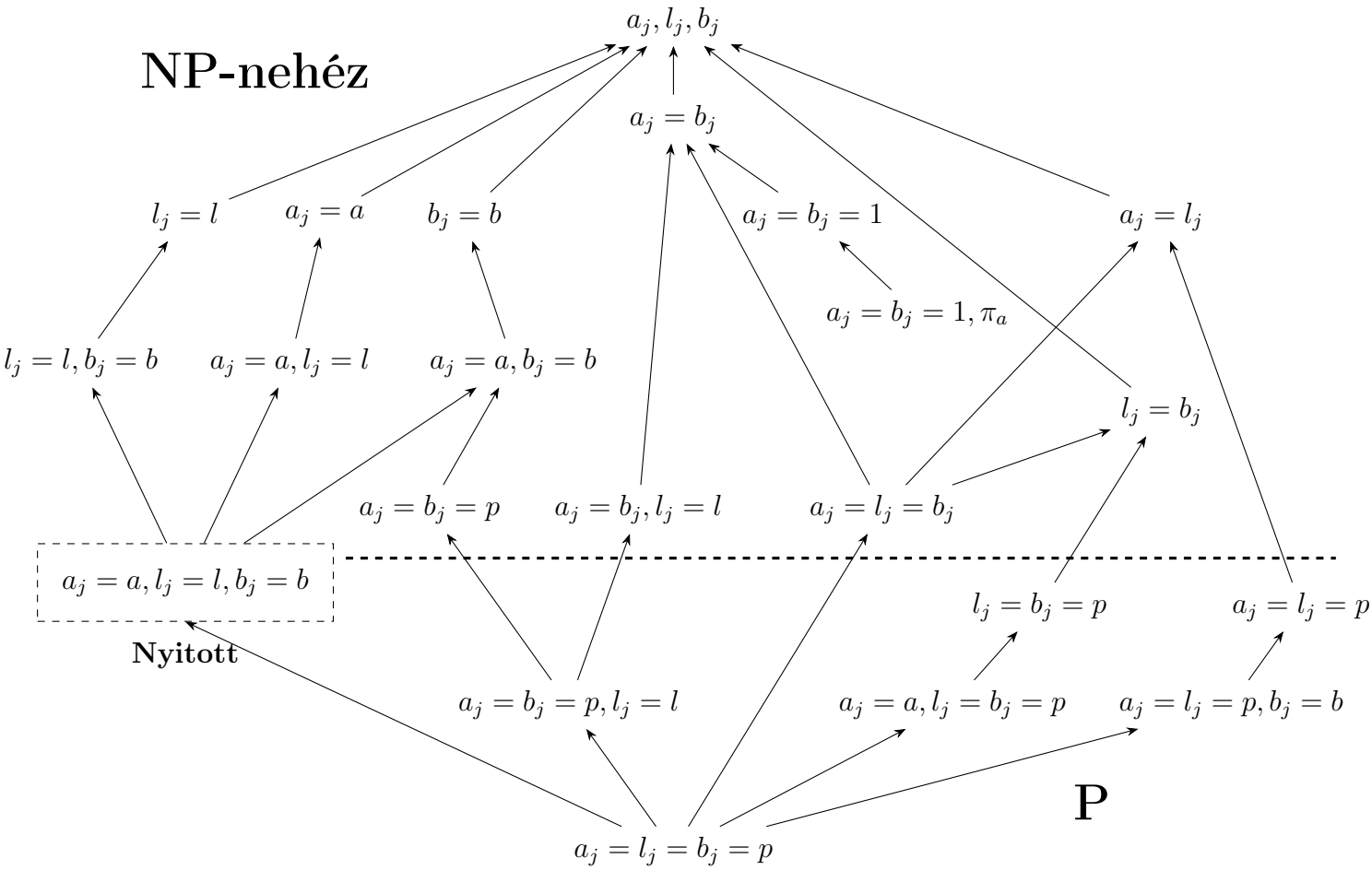
2004-ben Yu [16] belátta, hogy a $1|Coup - Task(a_j = b_j = p)|C_{max}$ feladat a $p = 1$ esetben is NP-nehéz. Condotta és Shakhlevich [5] 2012-ben megmutatták, hogy azzal a további szigorítással is NP-nehéz marad a feladat, ha kikötjük, milyen sorrendben kell a munkák feldolgozását elkezdeni - ezt a variánst a $1|Coup - Task(a_j = b_j = 1, \pi_a)|C_{max}$ hármassal jelöljük.

2023-ban Fischer és Györgyi [8] újabb esetre bizonyította az NP-nehézséget: az $1|Coup - Task(a_j = b_j, l_j = l)|C_{max}$ feladatra.

Orman és Potts [12] 1997-es cikkükben adtak polinomiális algoritmust az $1|Coup - Task(a_j = l_j = p)|C_{max}$ és $1|Coup - Task(a_j = b_j = p, l_j = l)|C_{max}$ feladatokra, amiből következik, hogy a kevésbé általános, illetve a 3.3. Állítás miatt ekvivalens feladattípusok is P-ben vannak.

Azokat a feladatokat, ahol $a_j = a$, $l_j = l$ és $b_j = b$ identikus feladatnak hívjuk. A $1|Coup - Task(a_j = a, b_j = b, l_j = l)|C_{max}$ feladat bonyolultsága nyitott. Rögzített (a, l, b) értékek mellett létezik $O(\log n)$ algoritmus [3].

Az egyes feladattípusok komplexitási viszonyát bonyolultsági gráfokon ábrázolhatjuk. A bonyolultsági gráf egy irányított gráf, melynek csúcsai megfelelnek az egyes problématípusoknak. Minden él egy kevésbé általános feladattípust jelölő csúcsból mutat egy általánosabb feladatot jelölő csúcs felé. Az eredményeket a 2. ábrán foglaltuk össze.



2. ábra. Bonyolultsági gráf a C_{max} célfüggvény esetén. A szaggatott vonal feletti variánsok NP-nehezek, az alatta lévők polinom időben megoldhatóak. Az $a_j = a$, $l_j = l$, $b_j = b$ variáns bonyolultsága nyitott.

3.2. A befejezési idők összegének minimalizálása

A problématípusok komplexitás szerinti osztályozásával a $\sum C_j$ célfüggvény mellett először Chen és Zhang [6] foglalkozott 2021-ben. Belátták az $1|Coup - Task(a_j = b_j = p)|\sum C_j$, $1|Coup - Task(l_j = l, b_j = b)|\sum C_j$, $1|Coup - Task(a_j = a, l_j = l)|\sum C_j$ és $1|Coup - Task(a_j = l_j = b_j)|\sum C_j$ feladatok NP-nehézségét. 2023-ban Fischer és Györgyi [8] bizonyították az $1|Coup - Task(a_j = b_j, l_j = l)|\sum C_j$, illetve az $1|Coup - Task(a_j = 1, b_j = 1, \pi_a)|\sum C_J$ feladatok NP-nehézségét.

Az $1|Coup - Task(a_j = l_j = p)|\sum C_j$, $1|Coup - Task(b_j = l_j = p)|\sum C_j$, $1|Coup - Task(a_j = b_j = p, l_j = l)|\sum C_j$ feladatokra Chen és Zhang adott polinomiális algoritmust [6].

Az identikus feladat bonyolultsága a $\sum C_j$ célfüggvény szerint is nyitott.

Ezek alapján az $1|Coup - Task|\sum C_j$ bonyolultsági gráfja megegyezik az $1|Coup - Task|C_{max}$ feladat bonyolultsági gráfjával, ami a 2. ábrán látható.

Az $1|Coup - Task(a_j = a, b_j = b)|\sum C_j$ feladat NP-nehézsége következik az $1|Coup - Task(a_j = b_j = p)|\sum C_j$ feladat NP-nehézségéből, míg a $1|Coup - Task(l_j = l)|\sum C_j$ feladat NP-nehézsége az $1|Coup - Task(a_j = a, l_j = l)$ feladat NP-nehézségéből. A szakdolgozat későbbi fejezeteiben kiemelt szerepet kapnak ezek a problémák, így lássuk most az NP-nehézséget bizonyító két tétel bizonyítását.

Mindkét bizonyítás azon alapul, hogy az NP-teljes 3-partíció problémát visszavezetjük az adott probléma eldöntési verziójára. Egy ütemezési probléma eldöntési verziójának hívjuk azt a feladatot, ahol a kérdés nem az optimumra irányul, hanem hogy létezik-e adott célfüggvényértékű ütemezés.

3.2. Definíció. 3-partíciós probléma

Input: $\mathcal{S} = \{e_1, e_2, \ldots, e_{3m}\} \subset \mathbb{Z}$, $T = \frac{\sum_{i=1}^{3m} e_i}{m}$, $\forall i \in \{1, 2, \ldots 3m\} : \frac{T}{4} < e_i < \frac{T}{2}$.
Kérdés: Partícionálható-e az $\mathcal{S}$ halmaz $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_m$ diszjunkt halmazokra úgy, hogy $\forall j \in \{1, 2, \ldots m\}$-re S_j-ben az elemek összege T.

3.3. Tétel. *[9] A 3-partíciós probléma NP-teljes.*

A tétel bizonyításához először lássunk egy lemmát. Vezessük be a következő jelöléseket: jelölje B_j^n a j-ik blokkban lévő feladatok számát, B_j^l a blokk hosszát és B_j^w a blokkban lévő feladatok befejezési ideje és a blokk kezdete közötti eltelt idők összegét.

3.4. Lemma. Minden optimális ütemezés bármely két B_i, B_j blokkjára igaz, hogy ha B_i hamarabb van ütemezve, mint B_j, akkor $\frac{B_i^n}{B_i^l} \geq \frac{B_j^n}{B_j^l}$.

Bizonyítás. Tegyük föl, hogy egy optimális ütemezésben léteznek B_i és B_j blokkok, hogy B_i hamarabb van ütemezve, mint B_j és $\frac{B_j^n}{B_j^l} > \frac{B_i^n}{B_i^l}$. Ekkor létezik két szomszédos blokk $B_{i'}$ és $B_{j'}$, amik közül $B_{i'}$ van hamarabb ütemezve és $\frac{B_{j'}^n}{B_{j'}^l} > \frac{B_{i'}^n}{B_{i'}^l}$. Cseréljük fel ezt a két blokkot, és nézzük, hogyan változik a célfüggvény. A befejezési idők a többi blokkban nem változnak, így elég ezt a két blokkot vizsgálnunk. Legyen $\{B_1 \ldots, B_{i'-1}\}$ a $B_{i'}$ előtt ütemezett blokkok halmaza. Az eredeti ütemezésben ebben a $B_{i'}$ és $B_{j'}$ blokkokban a befejezési idők összege:

$$\sum_{k=1}^{i'-1} B_k^l(B_{i'}^n + B_{j'}^n) + B_{i'}^w + B_{i'}^l B_{j'}^n + B_{j'}^w$$

A csere után:

$$\sum_{k=1}^{i'-1} B_k^l(B_{i'}^n + B_{j'}^n) + B_{i'}^w + B_{j'}^l B_{i'}^n + B_{j'}^w$$

$$\frac{B_{j'}^n}{B_{j'}^l} > \frac{B_{i'}^n}{B_{i'}^l} \implies B_{i'}^w + B_{i'}^l B_{j'}^n + B_{j'}^w > B_{i'}^w + B_{j'}^l B_{i'}^n + B_{j'}^w.$$

Ez ellentmond az eredeti ütemezés optimalitásának. □

3.5. Megjegyzés. A bizonyításból az is következik, hogy két B_i és B_j blokk felcserélésével az ütemezés optimális marad, ha $\frac{B_i^n}{B_i^l} = \frac{B_j^n}{B_j^l}$.

3.6. Tétel. *[6] Az* $1|Coup - Task(a_j = b_j = p)|\sum C_j$ *NP-nehéz.*

Bizonyítás. Belátjuk, hogy a 3-partíciós probléma polinomiálisan visszavezethető a $1|Coup - Task(a_i = b_i = p)|\sum C_j$ eldöntési verziójára.

Legyen $\mathcal{S} = \{e_1, e_2, \ldots, e_{3m}\} \subset \mathbb{Z}$, $T = \frac{\sum e_i}{m}$ a 3-partíciós probléma egy példánya. Legyen $\omega = 40m(2m+1)T$. Definiáljuk az ütemezési feladat egy példányát:

$n = \omega + 6m$

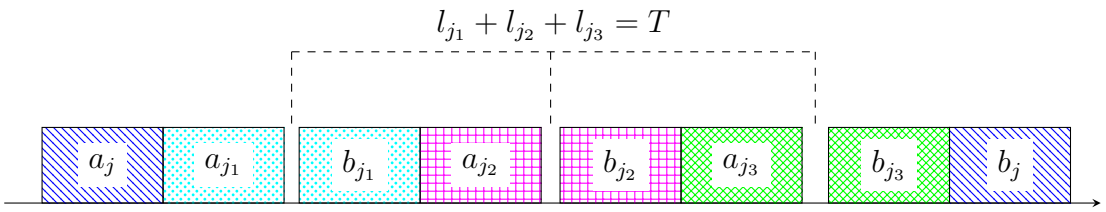
$a_j = b_j = 2T \quad l_j = e_j \qquad j = 1, \ldots, 3m$

$a_j = b_j = 2T \quad l_j = 13T \qquad j = 3m+1, \ldots, 4m$

$a_j = b_j = 2T \quad l_j = 2T - 1 \qquad j = 4m+1, \ldots, 6m+\omega.$

Legyen $y = \omega + m(29T-2)\omega + (6T-1)\dfrac{\omega(\omega+1)}{2}$. Belátjuk, hogy akkor és csakis akkor létezik a 3-partíciós problémának megoldása, ha létezik olyan ütemezés, ahol $\sum C_j < y$.

$\Longrightarrow$: Tegyük fel, hogy a partíciós problémának létezik megoldása. Jelölje $\{\mathcal{S}_i\}_{i=1}^m$ a megfelelő partícionálást: $\sum_{e_j \in \mathcal{S}_i} e_j = T$, $\forall i = 1, \ldots, m$. Az első $4m$ munkát ütemezzük m blokkba: minden $j \in \{3m+1, \ldots, 4m\}$ munka várakozási idejében ütemezzük közvetlenül egymás után azokat a munkákat, melyek várakozási ideje az $\mathcal{S}_i = \{e_{i_1}, e_{i_2}, e_{i_3}\}$ partícióban van, ahol $i = j - 3m$. Ez a három munka hossza összesen $4T + e_{i_1} + 4T + e_{i_2} + 4T + e_{i_3} = 13T$, így ütemezhetőek a j munka várakozási idejében. Tehát egy blokk a következőképpen fog kinézni, ahol a j munka várakozási idejében a j_1, j_2, j_3 munkákat ütemezzük:



3. ábra. A négy munka által alkotott blokk, ahol a holtidő összege T.

A maradék $\omega + 2m$ munkát ütemezzük az m-ik blokk után egyenként úgy, hogy a gép a munkák várakozási idején kívül folyamatosan dolgozzon. A $j = 3m+1, \ldots, 4m$ munkák hossza $17T$. Szigorúan egymás után vannak ütemezve, így a befejezési idők számtani sorozatot alkotnak.

$$\sum_{j=3m+1}^{4m} C_j = 17T\frac{m(m+1)}{2}.$$

Minden $j = 3m+1, \ldots, 4m$ munka várakozási idejében ütemeztünk 3 darab $j \in \{1, \ldots, 3m\}$ munkát, így az első $3m$ munka befejezési idejeinek összegét felülről becsülhetjük:

$$\sum_{j=1}^{4m} C_j < 3\left[17T\frac{m(m+1)}{2}\right].$$

A maradék $2m+\omega$ munkát vizsgáljuk két részben. Először nézzük a $j = 4m+1, \ldots, 6m$ munkák befejezési idejeinek összegét. Mindegyik megmunkálási ideje összesen $6T-1$

és az első $17mT$-kor kezdődik, így:

$$\sum_{4m+1}^{6m} C_j = 34m^2T + (6T-1)m(2m+1).$$

Az utolsó ω munka feldolgozása a $17mT + 2m(6T-1) = m(29T-2)$ időpontban tud elkezdődni, és hosszuk szintén $6T-1$, tehát:

$$\sum_{6m+1}^{6m+\omega} C_j = \omega m(29T-2) + (6T-1)\frac{\omega(\omega+1)}{2}.$$

Összegezzük most a kapott befejezési időket:

$$\begin{aligned}
\sum C_j &< 34Tm(m+1) + 34m^2T + (6T-1)m(2m+1) + m(29T-2)\omega + (6T-1)\frac{\omega(\omega+1)}{2} = \\
&= 34Tm(2m+1) + (6T-1)m(2m+1) + m(29T-2)\omega + (6T-1)\frac{\omega(\omega+1)}{2} = \\
&= m(2m+1)(40T-1) + m(29T-2)\omega + (6T-1)\frac{\omega(\omega+1)}{2} = \\
&= \omega + m(29T-2)\omega + (6T-1)\frac{\omega(\omega+1)}{2} - m(2m+1) = \\
&= y - m(2m+1).
\end{aligned}$$

Ezzel beláttuk, hogy ha a partíciós problémának létezik megoldása, akkor a célfüggvényérték kisebb, mint y.

$\Longleftarrow$: Most tegyük fel, hogy a 3-partíciós problémának nincs megoldása. Belátjuk, hogy ekkor $\sum C_j \geq y$.

A $j = 3m+1, \ldots, 4m$ munkák kivételével a többi munka várakozási ideje túl rövid ahhoz, hogy az alatt egy másik munka feladatrészén dolgozzon a gép. Ez azt jelenti, hogy ezek a munkák vagy különálló blokkot alkotnak, vagy egy $j \in \{3m+1 \ldots, 4m\}$ munka várakozási idejében vannak ütemezve. Ezek várakozási ideje $13T$, így legfeljebb 2 darab $2T-1$ várakozási idejű munka ütemezhető egy alatt. Tehát van legalább ω munka, amelyek várakozási ideje $2T-1$ és különálló blokkot alkotnak. Vegyünk egy optimális ütemezést. Ekkor az 5.3.Lemma, illetve a 5.4.Megjegyzés alapján feltehetjük, hogy legalább ω darab $2T-1$ várakozási idejű munka közvetlenül egymás után van ütemezve. Jelölje ezt az ω egymás után ütemezett munkát B.

1. eset: Minden B-n kívüli B_i blokkra $\dfrac{B_i^n}{B_i^l} \geq \dfrac{1}{(6T-1)}$. Ekkor tehát B az ütemezés végén

áll. Nézzük leghamarabb mikor kezdődhet meg B feldolgozása.

$$\sum_{j=1}^{3m}(a_j+l_j+b_j)+\sum_{j=3m+1}^{4m}(a_j+b_j)+\sum_{j=4m+1}^{6m}(a_j+l_j+b_j) = 13mT+4mT+2m(6T-1) = 29mT-2m.$$

Mivel a 3-partíciós feladatnak nincs megoldása, ezért a $j = 3m+1\ldots,4m$ munkák között kell, hogy legyen olyan, aminek a várakozási idejében nem dolgozik folyamatosan a gép. Emiatt a B feldolgozása leghamarabb $m(29T-2)+1$ időpontban tud elkezdődni, amiből következik, hogy:

$$\sum_{j=6m+1}^{6m+\omega} C_j \geq \omega\left[m(29T-2)+1\right]+(6T-1)\frac{\omega(\omega+1)}{2} = y \implies \sum C_j \geq y.$$

2. eset: Létezik legalább egy olyan B_i blokk, amire $\frac{B_i^n}{B_i^l} < \frac{1}{(6T-1)}$. Készítünk egy új (nem megengedett) ütemezést, amire a célgüggvény értéke kisebb, mint az eredeti optimális ütemezésé, és belátjuk, hogy még ez is nagyobb, mint y. Nézzük a B után ütemezett blokkokat, legyen az első ezek közül B_k. Ezekben a blokkokban kell, hogy legyen legalább egy olyan munka, aminek a várakozási ideje $13T$. Jelölje η_1 a B_k blokkban lévő $j \in \{1,\ldots,3m\}$ munkák számát, illetve legyen λ ezen munkák hosszának összege. Jelölje η_2 a $j \in \{3m+1,\ldots,4m\}$ munkák számát, η_3 a $j \in \{4m+1,\ldots,6m\}$ munkák számát. A blokkban lévő azon holtidők összege, ami nem egy $j \in \{1,\ldots,3m\} \cap \{4m+1,\ldots,6m\}$ munka várakozási ideje, legyen L. Tehát a blokk hossza felírható, mint $\lambda_1+\eta_2 4T+\eta_3(6T-1)+L$. Ez alapján:

$$\frac{B_k^n}{B_k^l} = \frac{\eta_1+\eta_2+\eta_3}{\lambda_1+4T\eta_2+(6T-1)\eta_3+L} < \frac{1}{(6T-1)}.$$

A $j \in \{1,\ldots,3m\}$ munkák hossza kisebb, mint $6T-1$, ezért:

$$\frac{\eta_1}{\lambda_1} > \frac{1}{(6T-1)}.$$

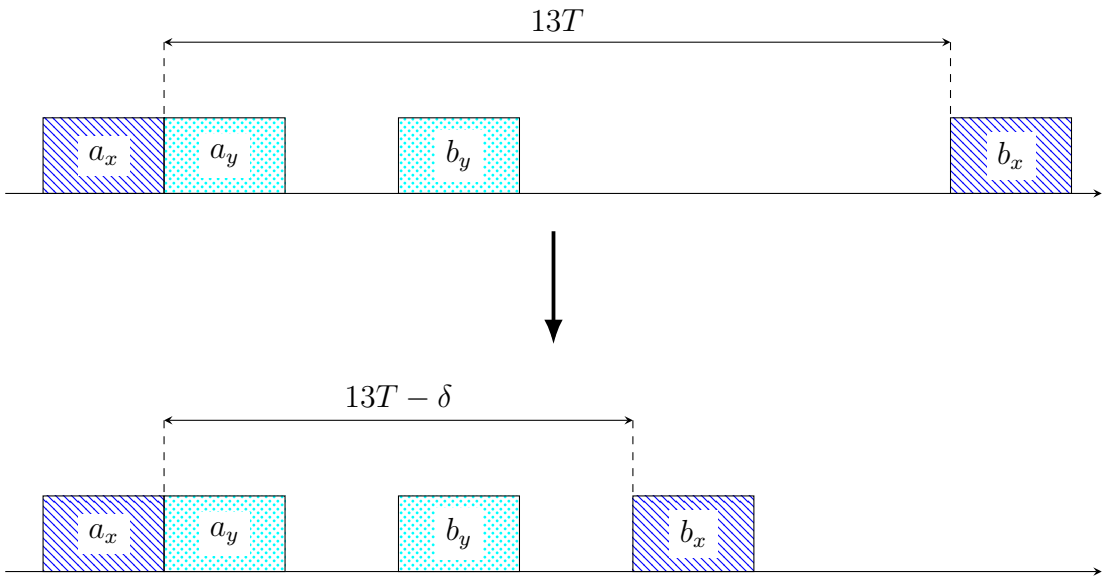
A $j \in \{4m+1,\ldots,6m\}$ munkák hossza pontosan $6T-1$:

$$\frac{\eta_3}{\eta_3(6T-1)} = \frac{1}{(6T-1)}.$$

Az előző három összefüggés alapján:

$$\frac{\eta_2}{4T\eta_2+L} < \frac{1}{(6T-1)} \implies L > \eta_2(2T-1) > 0.$$

Legyen $\delta = L-\eta_2(2T-1) > 0$. A B_k pár feladatrészét, beleértve az utolsót, ütemezzük



4. ábra. Egy B-t követő blokk lehetséges átalakítása.

hamarabbra úgy, hogy a blokkban lévő azon holtidők összege, ami nem egy $j \in \{1, \ldots, 3m\} \cap \{4m+1, \ldots, 6m\}$ munka várakozási ideje, δ-val csökkenjen. Ezzel bár az ütemezés nem lesz megengedett, a célfüggvény csökken.

Hajtsuk végre ezt az átalakítást a megfelelő értékek mellett az összes B-t követő blokkon. Egy lehetséges átalakítás a következőképpen nézhet ki:

Az átalakítások után minden B-t követő B_j blokkban a $\frac{B_j^n}{B_j^l}$ érték eléri a $\frac{1}{(6T-1)}$-t. Ütemezzük ezeket a blokkokat B elé, a lemma miatt ezzel a célfüggvény tovább csökkenhet, de biztosan nem nő. A holtidőket minden blokkban úgy csökkentettük, hogy értékük pozitív maradjon. Ez azt jelenti, hogy az új ütemezésünkben B leghamarabb $m(29T-2)+1$-kor tud elkezdődni. Jelölje az új befejezési időket C'_j, $\forall j \in \{1, \ldots, 6m+\omega\}$. Az új ütemezést úgy hoztuk létre, hogy van olyan befejezési idő, ami csökkent, és semelyik sem növekedett. Ezek értelmében:

$$\sum C_j > \sum C'_j > [m(29T-2)+1]\,\omega + (6T-1)\frac{\omega(\omega+1)}{2} = y.$$

Ezzel ezt az irányt is beláttuk. □

3.7. Tétel. *[6] Az* $1|Coup - Task(a_j = a, l_j = l)|\sum C_j$ *NP-nehéz.*

Bizonyítás. Belátjuk, hogy a 3-partíciós probléma polinomiálisan visszavezethető a $1|Coup - Task(l_j = l, b_j = b)|\sum C_j$ eldöntési verziójára.
Legyen $\mathcal{S} = \{e_1, e_2, \ldots, e_{3m}\} \subset \mathbb{Z}$, $T = \frac{\sum e_i}{m}$ a 3-partíciós probléma egy példánya.

Legyen $\min_i e_i = a$. Legyen $A = 12m^2T$ és $\omega = 9m^2A$. Definiáljuk az ütemezési feladat egy példányát:

$n = 5m + \omega$

$a_j = a \quad l_j = T \quad b_j = e_j \qquad\qquad i = 1, \ldots, 3m$

$a_j = a \quad l_j = T \quad b_j = A \qquad\qquad j = 3m+1, \ldots, 5m+\omega.$

Jelölje a $j = 3m+1, \ldots, 5m+\omega$ munkák hosszát $\alpha = a + T + A$. Legyen $y = 3m(2T + a) + (3m - 1 + \omega)m(2T + A + a) + \frac{(m+\omega)(m+\omega+1)}{2}\alpha$. Belátjuk, hogy a 3-partíciós problémának pontosan akkor létezik megoldása, ha létezik ütemezés, amire $\sum C_j < y$.

$\Longrightarrow$: Tegyük fel, hogy ha a partíciós problémának létezik megoldása. Belátjuk, hogy ekkor létezik ütemezés, amire $\sum C_j < y$. Jelölje $\{\mathcal{S}_j\}_{j=1}^m$ a megfelelő partícionálást: $\sum_{e_i \in \mathcal{S}_j} e_i = T$, $\forall j = 1, \ldots, m$. Minden e_i elemnek megfeleltethető az a i munka, ahol $b_i = e_i$, ezen kívül minden $\mathcal{S}_j$ partíciónak feleltessük meg a $3m+j$ munkát. Ütemezzük a munkákat a partíciók szerint blokkokba: a partíciónak megfelelő munka várakozási idejében ütemezzük a partíció elemeinek megfelelő munkák második feladatrészét.

Legyen $\mathcal{S}_j = \{e_x, e_y, e_z\}$ egy partíció, ekkor az elemeknek megfelelő munkák x, y, z, a partíciónak megfelelő munka j.

$a_x = a \quad l_x = T \quad b_x = e_x$

$a_y = a \quad l_y = T \quad b_y = e_y$

$a_z = a \quad l_z = T \quad b_z = e_z$

$a_j = a \quad l_j = T \quad b_j = A$

Mivel b_x, b_y és b_z értékek megfelelnek a $\mathcal{S}_j$ partíció elemeinek: $b_x + b_y + b_z = T$. Ahhoz, hogy ezek a feladatrészek ütemezhetőek legyenek a j munka várakozási idejében, azt kell még meggondolni, hogy az első feladatrészeik feldolgozása nem fogja metszeni egymást, illetve a_j-t sem. Ütemezzük sorban az l_j várakozási időben a b_x, b_y és b_z feladatrészeket, nézzük ekkor hova ütemeződnek a feladatok első feladatrészei.

$S_{a_x} = C_{a_j} - l_x - a_x = C_{a_j} - T - a$

$C_{a_x} = C_{a_j} - l_x = C_{a_j} - T$

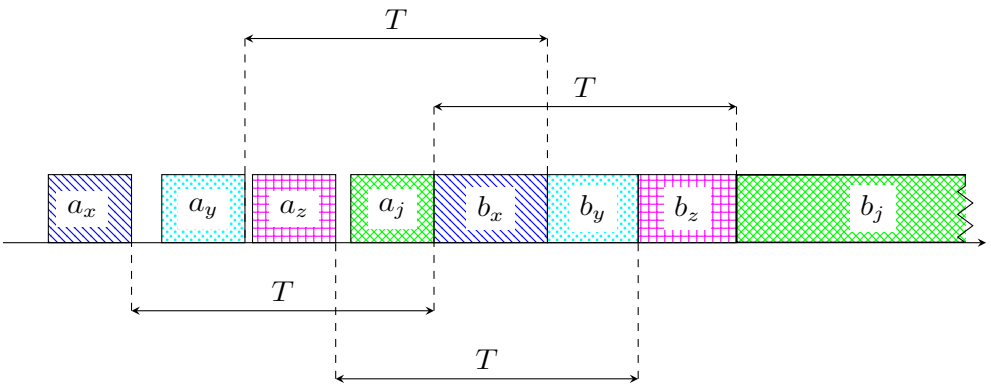
$S_{a_y} = C_{a_j} - l_x - a_x + b_x = C_{a_j} - T - a + b_x$

$C_{a_x} = C_{a_j} - l_x + b_x = C_{a_j} - T + b_x$

$$S_{a_z} = C_{a_j} - l_x - a_x + b_x + b_y = C_{a_j} - T - a + b_x + b_y$$
$$C_{a_z} = C_{a_j} - l_x + b_x + b_z = C_{a_j} - T + b_x + b_y.$$

Az a értékét az e_j-k minimumaként határoztuk meg, így az (S_q, C_q) $q \in \{a_x, a_y, a_z\}$ intervallumok nem fogják metszeni egymást. Ugyanezért az a_j ütemezését sem metszhetik, mivel $T - b_x - b_j \geq a = a_j$.

Tehát az $\mathcal{S}_j$-nek megfelelő blokk az 5. ábrán látható módon alakul.



5. ábra. Az $\mathcal{S}_j$ partíciónak megfelelő blokk.

Egy ilyen blokk hossza $a + 2T + A$, így a $j = 3m + 1, \ldots, 4m$ munkák befejezési idejeinek összege:

$$\sum_{j=3m+1}^{4m} C_j = \sum_{j=3m+1}^{4m} j(a + 2T + A) = (a + 2T + A)\frac{m(m+1)}{2} \tag{1}$$

Egy blokk kezdetétől a partíció elemeinek megfelelő munkák befejezéséig $a+2T$ idő telik el, ezért:

$$\sum_{j=1}^{3m} C_j < \sum_{j=1}^{m} j(a + 2T + A) - A < 3m(a + 2T) + (a + 2T + A)\frac{3m(m-1)}{2} \tag{2}$$

Az m blokk után ütemezzük egyenként a maradék $m + \omega$ munkát. Az m blokk feldolgozása $m(a + 2T + A)$ ideig tart, így az utolsó $m + \omega$ munka befejezési idejének összege:

$$\sum_{j=4m+1}^{5m+\omega} C_j = m(a+2T+A)+j\alpha = (m+\omega)m(a+2T+A)+\alpha\frac{(m+\omega)(m+\omega+1)}{2} \tag{3}$$

$(1),(2),(3) \implies \sum_{j=1}^{5m+\omega} C_j < y.$

$\Longleftarrow$: Megmutatjuk, hogy ha a 3-partíciós problémának nem létezik megoldása, akkor $\sum_{j=4m+1}^{5m+\omega} C_j \geq y$. Nem egy munkából álló blokk kétféle módon állhat elő: vagy csak $\mathcal{S}$ elemeinek megfelelő munkát tartalmaz, vagy legfeljebb három elemnek megfelelő munkát és egy j munkát, amire $b_j = A$. Az $\mathcal{S}$ elemeinek megfelelő munkák mindegyik hossza kisebb, mint $2T$, így egy B_p blokkra, ami k darab, kizárólag elemeknek megfelelő munkákból áll: $\frac{B_p^n}{B_p^l} > \frac{k}{2kT} = \frac{1}{2T}$. Egy B_q blokkra, ami különböző várakozási idejű munkákat tartalmaz: $\frac{B_q^n}{B_q^l} > \frac{2}{a+2T+A}$. A B_r egy darab $j \in \{3m+1,\ldots,5m+\omega\}$ munkát tartalmazó blokkokra $\frac{B_r^n}{B_r^l} = \frac{1}{\alpha}$.

$\frac{1}{2T} > \frac{2}{a+2T+A} > \frac{1}{\alpha}$, így a 5.3. Lemma értelmében egy optimális ütemezésben azok a j munkák lesznek utoljára ütemezve, melyekre $b_j = A$ és nem alkotnak más munkával blokkot. Bármely $j \in \{3m+1,\ldots,5m+\omega\}$ munka tud blokkot alkotni bármely két elemnek megfelelő munkával. Ugyanakkor optimális ütemezésben nem létezhet kettő olyan blokk, ami egy elemnek megfelelő, és egy olyan j munkából áll, amire $b_j = A$, hiszen ekkor a 5.3. Lemma miatt lenne két ilyen blokk egymás után ütemezve, amelyek közül, ha a később ütemezett blokkból kivennénk az elemnek megfelelő munkát és a hamarabb ütemezett blokkban ütemeznénk, a célfüggvény csökkenne. Ez azt jelenti, hogy egy optimális ütemezés végén legalább ω darab egy $j \in \{3m+1,\ldots,5m+\omega\}$ munkából álló blokk fog állni.

Ha egy blokkba beépítünk egy $j \in \{1,\ldots,3m\}$ munkát, az a blokk hosszát legalább b_j-vel növeli. Másrészt a $j > 3m$ munkák közül egy blokkban csak egy lehet, ezért az első $5m$ munka nem fejeződhet be hamarabb, mint $2m\alpha + \sum_{i=1}^{3m} b_j = m(2\alpha+T)$.

Viszont ahhoz, hogy a célfüggvény kisebb legyen, mint y, az utolsó $m+\omega$ munka ütemezése nem kezdődhet később, mint $m(2\alpha+T)$, máskülönben:

$$\sum_{j=5m+1}^{5m+\omega} C_j \geq \omega + \omega m(2\alpha+T) + \alpha\frac{\omega(\omega+1)}{2} =$$

$$= 9m^2A + \alpha\frac{(m+\omega)(m+\omega+1)}{2} - \alpha\frac{m(m+1)}{2} + \omega m(\alpha+T) >$$

$$> 3m(2T+a)+(3m-1)m(a+2T+A)+\alpha\frac{(m+\omega)(m+\omega+1)}{2}+\omega m(\alpha+T) =$$

$$= y$$

Tehát az első $5m$ munka feldolgozásának $m(2\alpha + T)$-ig be kell fejeződnie. Ez azt jelenti, hogy minden $j = \{1, \ldots, 3m\}$ munka olyan blokkban van ütemezve, ami tartalmaz $j > 3m + 1$ munkát. Ha a 3-partíciós problémának nincs megoldása, akkor kell lennie legalább $m + 1$ darab nem egy munkából álló blokknak. Emiatt:

$$\sum_{j=1}^{3m} C_j \geq A + 3\sum_{j=1}^{m}(j-1)A = A + 3A\frac{m(m-1)}{2}$$

Tehát az összes munka befejezési idejének összege:

$$\sum_{j=1}^{5m+\omega} C_j = \sum_{j=1}^{3m} C_j + \sum_{j=3m+1}^{5m} C_j + \sum_{j=5m+1}^{5m+\omega} C_j \geq$$

$$\geq A + 3A\frac{m(m-1)}{2} + \alpha\frac{2m(2m+1)}{2} + m(2\alpha+T)\frac{\omega(\omega+1)}{2} \geq$$

$$\geq A + 3A\frac{m(m-1)}{2} + \alpha\frac{m(3m+1)}{2} + \alpha\frac{m(m+1)}{2} + \omega m(a+2T+A) + \omega m\alpha + \alpha\frac{\omega(\omega+1)}{2} =$$

$$= A + 3A\frac{m(m-1)}{2} + \alpha\frac{m(3m+1)}{2} - m(3m-1)(a+2T+A) - 3m(2T+a) + y =$$

$$= A + (T+a)\frac{m(3m+1)}{2} - m(3m+2)(2T+a) + y$$

$$> y.$$

Ezzel ezt az irányt is beláttuk. □

4. Matematikai programozási modellek

Ahogy az előzőekben láttuk, a vizsgált ütemezési problémák számos esetben NP-nehéznek bizonyulnak. A matematikai programozási modellek lehetőséget biztosítanak a problémák strukturált megfogalmazására és kezelésére. A következőkben a legkésőbbi befejezési idő minimalizálására megfogalmazott egészértékű programokat ismertetünk, majd ezeket átalakítjuk úgy, hogy a befejezési idők összegének minimalizálását modellezzék.

A modellek leírása során legyen M egy felsőbecslés C_{max}-ra: $M = \sum_{j=1}^{n}(a_j + l_j + b_j)$. Ennyi a legkésőbbi befejezési idő akkor, ha a munkákat szigorúan egymás után ütemezzük úgy, hogy a gép csak a várakozási időkben nem dolgozik. Tehát ennyi idő alatt biztosan minden munka ütemezhető.

4.1. Időindexelt modell

Az egygépes páros munkák ütemezésére az első matematikai programot 2004-ben publikálták [7]. Az idősíkot diszkretizáljuk egységnyi időpontokra. Az időpontok halmazát jelölje $\mathcal{T} = \{1, \ldots, M\}$. Minden (j,t), $j \in \mathcal{J}$, $t \in \mathcal{T}$ párra bevezetünk egy $x_{j,t}$ bináris változót, aminek az értéke pontosan akkor 1, ha a j munka t-kor kezdődik. Jelölje τ_j azt a legutolsó időpillanatot, amikor a j munka még elkezdődhet: $\tau_j = M - (a_j + l_j + b_j) + 1, \forall j \in \mathcal{J}$. Minden (j,t) párra legyen $\xi_j^t = \{t - a_j + 1, t - a_j + 2 \ldots, t\} \cap \mathcal{T}$, $\zeta_j^t = \{t - (a_j + l_j + b_j) + 1, t - (a_j + l_j + b_j) + 2 \ldots, t - (a_j + l_j)\} \cap \mathcal{T}$. Egy t időpontban az a_j feladatrész feldolgozása folyik, ha $S_j \in \xi_j^t$, illetve a b_j feladatrész feldolgozása folyik, ha $S_j \in \zeta_j^t$. Ezek ismeretében lássuk az egészértékű programot a $1|Coup - Task|C_{max}$ feladatra:

$$\min C_{max}$$

feltéve, hogy:

$$C_{max} \geq \sum_{t=1}^{\tau_j}(t + a_j + l_j + b_j - 1)x_{j,t} \qquad \forall j \in \mathcal{J} \quad (1)$$

$$\sum_{t=1}^{\tau_j} x_{j,t} = 1 \qquad \forall j \in \mathcal{J} \quad (2)$$

$$\sum_{j=1}^{n} \sum_{\substack{q \in \xi_j^t \\ q \leq \tau_j}} x_{j,q} + \sum_{j=1}^{n} \sum_{\substack{q \in \zeta_j^t \\ q \leq \tau_j}} x_{j,q} \leq 1 \qquad \forall t \in \mathcal{T} \quad (3)$$

$$x_{j,t} \in \{0,1\} \qquad \forall j \in \mathcal{J}, \forall t \in \mathcal{T} \quad (4)$$

Az (1) feltétel biztosítja, hogy $C_{max} \geq C_j$, $\forall j \in \mathcal{J}$. A (2) feltétel ahhoz kell, hogy minden munka feldolgozása pontosan egyszer kezdődjön el, míg a (3) feltétel meggátolja, hogy egyszerre több munka feldolgozása folyjon a gépen.

A célfüggvény módosításával, illetve az első feltétel elhagyásával kapunk modellt a teljes befejezési idő minimalizálása feladatra. A j feladat befejezési ideje számolható, mint $\sum_{t=1}^{\tau_j} x_{j,t}(t + a_j + l_j + b_j - 1)$, így a módosított modell a következőképpen írható fel:

$$\min \sum_{j=1}^{n} \sum_{t=1}^{\tau_j} x_{j,t}(t + a_j + l_j + b_j - 1)$$

feltéve, hogy:

$$\sum_{t=1}^{\tau_j} x_{j,t} = 1 \qquad \forall j \in \mathcal{J} \quad (5)$$

$$\sum_{j=1}^{n} \sum_{\substack{q \in \xi_j^t \\ 1 \leq q \leq \tau_j}} x_{j,q} + \sum_{j=1}^{n} \sum_{\substack{q \in \zeta_j^t \\ 1 \leq q \leq \tau_j}} x_{j,q} \leq 1 \qquad \forall t \in \mathcal{T} \quad (6)$$

$$x_{j,t} \in \{0,1\} \qquad \forall j \in \mathcal{J}, \forall t \in \mathcal{T} \quad (7)$$

A gyakorlatban gyakran előfordul, hogy nem áll rendelkezésre tetszőlegesen sok idő. Ekkor az M-t lecserélve az ütemezésre fenntartott időre kereshetünk megengedett megoldásokat. Ugyanakkor, ha M nem egy megfelelő felsőbecslés C_{max}, nincs rá garancia, hogy létezik megengedett megoldás.

Az időindexelt modell hátránya, hogy a változók száma n munka esetén nM, ami $M >> n$ esetén nagyon lecsökkentheti az effektivitást.

4.2. Sherali és Smith modellje

A változók gyors növekedését Sherali és Smith[15] egy új modellel próbálták megoldani. Bármely i és j munka egymáshoz képest legfeljebb 5 féle képen lehet ütemezve:

1. eset: A i munka szigorúan j előtt van ütemezve:

2. eset: A j munka szigorúan i előtt van ütemezve:

3. eset: A két munka feldolgozása egymásba fonódik. Az i munka feldolgozása hamarabb kezdődik, és hamarabb is fejeződik be, mint a j munkáé.

4. eset: A két munka feldolgozása egymásba fonódik. Az j munka feldolgozása hamarabb kezdődik, és hamarabb is fejeződik be, mint az i munkáé.

5. eset: A két munka feldolgozása egymásba fonódik. Az i munka feldolgozása hamarabb kezdődik, és később fejeződik be, mint a j munka feldolgozása vagy a j munka feldolgozása kezdődik hamarabb, és fejeződik be később. Ehhez teljesülni kell, hogy $l_i \geq a_j + l_j + b_j$ vagy $l_j \geq a_i + l_i + b_i$, így egyszerre biztosan csak az egyik eset fordulhat elő.

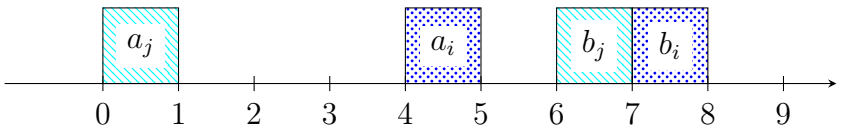
Minden i és j munkához, ahol $i < j$ bevezetünk öt darab $y_{i,j,k}$ bináris változót, $k = 1, \ldots, 5$. Az $y_{i,j,k} = 1$, ha az i és j munkák relatív pozíciója a k-ik esetnek felel meg. Jelölje $l_{i,j,k}$ a leghamarabbi, $r_{i,j,k}$ a legkésőbbi relatív időpontot, amikor a j munka feldolgozása el tud kezdődni az i munka feldolgozásának kezdetéhez képest, ha az i és j munkák relatív pozíciója k. Tehát, ha az i és j munkák relatív pozíciója k, akkor $l_{i,j,k} \leq S_j - S_i \leq r_{i,j,k}$. Ezt a legfeljebb $5\frac{n(n-1)}{2}$ darab feladattól függő konstansot a feladat előfeldolgozásánál kiszámoljuk.

4.1. Példa. Számoljuk az $r_{i,j,4}$ és $l_{i,j,5}$ értékeket a következő paraméterek mellett:

	a_i	l_i	b_i
i	1	2	1
j	1	5	1

2. táblázat. A feladat paraméterei

Számoljuk először $r_{i,j,4}$-et. Ekkor az $S_j - S_i < 0$, tehát az $r_{i,j,4}$-re egyszerűbb úgy tekinteni, hogy $S_i - S_j \geq -r_{i,j,4}$. Lássuk, hogy lehet ez a különbség a legkisebb:



6. ábra. $S_i - S_j$ minimalizálása $k = 4$ mellett

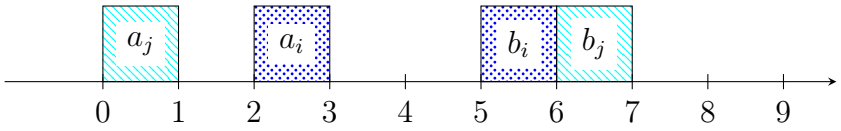
A b_i feldolgozása leghamarabb az $S_j + a_j + l_j + b_j$ időpontban kezdődhet. Az i munka feldolgozása a b_i feladrész feldolgozásának kezdeténél $a_i + l_i$-vel kezdődik hamarabb,

így:

$$S_i - S_j \geq a_j + l_j + b_j - (a_i + l_i) = 7 - 3 = 4.$$

Azt kaptuk tehát, hogy $r_{i,j,4} = -4$.

Nézzük most, hogyan számolható $l_{i,j,5}$. Mivel $a_i + l_i + b_i = 4 > l_j$, ezért az az eset állhat fenn, ahol a j munka feldolgozása kezdődik hamarabb és fejeződik be később. Tehát az előzőekhez hasonlóan $S_j < S_i$, így úgy érdemes az $l_{i,j,5}$ értékre tekinteni, hogy $S_i - S_j \leq -l_{i,j,5}$. Lássuk, hogy lehet ez a különbség a legkisebb:



7. ábra. $S_i - S_j$ maximalizálása $k = 5$ mellett

Az a_i munka feldolgozása legkésőbb a b_j munka feldolgozásának kezdete előtt az i munka hosszával kezdődhet, a b_j munka feldolgozásának kezdete pedig $S_j + a_j + l + j$, tehát:

$$S_i - S_j \leq a_j + l_j - (a_i + l_i + b_i) = 6 - 4 = 2.$$

Ezzel az $l_{i,j,5}$ értékét is meghatároztuk.

Az egészértékű program az $1|Cuop - Task|C_{max}$ feladatra az $l_{i,j,k}$ és $r_{i,j,k}$ értékek ismeretében:

$$\min C_{max}$$

feltéve, hogy:

$$C_{max} \geq S_j + (a_j + l_j + b_j) \qquad \forall j \in \mathcal{J} \quad (1)$$

$$\sum_{k=1}^{5} y_{i,j,k} = 1 \qquad \forall (i,j) : i < j, \quad i,j \in \mathcal{J} \quad (2)$$

$$S_j - S_i \geq \sum_{k=1}^{5} l_{i,j,k} y_{i,j,k} \qquad \forall (i,j) : i < j, \quad i,j \in \mathcal{J} \quad (3)$$

$$S_j - S_i \leq \sum_{k=1}^{5} r_{i,j,k} y_{i,j,k} \qquad \forall (i,j) : i < j, \quad i,j \in \mathcal{J} \quad (4)$$

$$S_j \geq 0 \qquad \forall j \in \mathcal{J} \quad (5)$$

$$y_{i,j,k} \in \{0,1\} \qquad \forall (i,j,k) : i < j, \quad i,j \in \mathcal{J}, k \in \{1,\ldots,5\} \quad (6)$$

Az (1)-es feltétel biztosítja, hogy $C_{max} \geq \max_{j \in \mathcal{J}} C_j$. A (2)-es feltétel értelmében bármely két munka relatív pozíciója pontosan egyféle lehet. A (3)-as és (4)-es feltételek biztosítják, hogy a gép egyszerre egy munkán dolgozzon. Az (5)-ös feltétel biztosítja, hogy a munkák feldolgozásának kezdeti időpontjai nem negatívak.

Az $1|Coup - Task|\sum C_j$ feladat megoldásához az (1)-es feltételhez nincs szükség, ezen felül elegendő a célfüggvényt módosítani a következőre: $\min \sum_{j \in \mathcal{N}} (S_j + a_j + l_j + b_j)$.

A (3)-as és (4)-es feltételnél adott i és j munka mellett elegendő az öt közül a lehetséges relatív pozícióknak megfelelő k értékeket figyelembe venni.

4.3. Lineáris rendezési modell

Az előző modellhez némiképp hasonló Békésiék 2014-es modellje [4], ami a feladatrészek egymáshoz való viszonyát veszi alapul. Ez két feladatrész esetében két fajta lehet annak függvényében, melyik feldolgozása kezdődik hamarabb. Egy ütemezés megad egy lineáris rendezést a feladatrészeken, a megengedett rendezések között keressük az optimálisat. Legyen $\mathcal{F} = \{1,\ldots,2n\}$ a feladatrészek halmaza: a j-ik feladathoz tartozzanak a $2j-1$ és $2j$ feladatrészek, ahol az előbbi felel meg a_j-nek, utóbbi b_j-nek. Minden $i,j \in \mathcal{F}$ feladatrész párhoz bevezetünk egy $x_{i,j}$ bináris változót, aminek értéke pontosan akkor 1, ha az i-ik feladatrész feldolgozása megelőzi a j-ik feladatrész feldolgozását. A $j \in \mathcal{F}$ feladatrész feldolgozási idejét jelölje p_j. Ekkor az egészértékű program:

$$\min C_{max}$$

feltéve, hogy:

$$C_{max} \geq S_{2j} + b_j \qquad \forall j \in \mathcal{J} \quad (1)$$

$$x_{2j-1,2j} = 1 \qquad \forall j \in \mathcal{J} \quad (2)$$

$$x_{i,j} + x_{j,i} = 1 \qquad \forall (i,j) : i < j, \quad i,j \in \mathcal{F} \quad (3)$$

$$x_{i,j} + x_{j,k} + x_{k,i} \leq 2 \qquad \forall(i,j,k) : i \neq j \neq k \neq i, \quad i,,j,k \in \mathcal{F} \quad (4)$$

$$S_{2j} = S_{2j-1} + a_j + l_j \qquad \forall j \in \mathcal{J} \quad (5)$$

$$S_j \geq S_i + p_i - M(1 - x_{i,j}) \qquad \forall(i,j) : i \neq j, \quad i,j \in \mathcal{F} \quad (6)$$

$$S_j \geq 0 \qquad \forall j \in \mathcal{F} \quad (7)$$

$$x_{i,j} \in \{0,1\} \qquad \forall(i,j) : i \neq j, \quad i,j \in \mathcal{F} \quad (8)$$

Az (1) feltétel biztosítja, hogy a C_{max} nagyobb bármely feladatrész befejezési idejénél. A (2) feltétel értelmében az ugyanazon munkához tartozó feladatrészek a megfelelő sorrendben kerülnek feldolgozásra - először a_j, majd b_j. A (3)-as és (4)-es feltételek biztosítják a lineáris rendezést a feladatrészeken, előbbi az antiszimmetriához, utóbbi a tranzitivitáshoz szükséges. Minden $j \in \mathcal{J}$ munka feladatrészeinek feldolgozása között pontosan l_j időnek kell eltelnie, ez következik az (5)-ös feltételből. Ha az i-ik feladatrész a j-ik előtt kerül ütemezésre, tehát $x_{i,j} = 1$, akkor a j-ik feladatrész leghamarabb az i-ik feladatrész befejezése után kezdődhet: ezt biztosítja a (6)-ös, 'big M' korlát.

A munkák befejezési idejeinek összege $\sum_{j \in \mathcal{J}} (S_{2j} + p_{2j})$, így a célfüggvény módosításával erre, illetve az (1) feltétel elhagyásával a modell használható a $1|Coup-Task|\sum C_j$ feladatra.

4.2. Megjegyzés. A C_{max} minimalizálása ekvivalens a holtidők minimalizálásával. Ezt a megközelítést használja Békésiék második modellje [4], ez viszont a $\sum C_j$ célfüggvényre nem alkalmazható.

Lássuk, hogyan alakul az egyes modelleknél a változók illetve feltételek száma az idősík nagyságának és a munkák számának függvényében:

Modell	Változók száma	Feltételek száma
Time-index modell	$O(Mn)$	$O(n+M)$
Sherali és Smith modellje	$O(n^2)$	$O(n^2)$
Lineáris rendezési modell	$O(n^2)$	$O(n^6)$

3. táblázat. A változók és feltételek növekedési üteme az egyes modelleknél

5. $1|Coup - Task(a_i = a, b_i = b)|\sum C_j$

A következőkben rátérünk a $1|Coup - Task(a_i = a, b_i = b)|\sum C_j$ feladat vizsgálatára. Elsőként áttekintjük, hogyan teljesítenek az előző fejezetben ismertetett matematikai programozási modellek. Ezt követően bemutatunk egy közelítő algoritmust, majd különböző módszerekkel javítjuk annak teljesítményét.

Az elemzésekhez szükséges programkódokat Python nyelven készítettem el, ezek eredményei kerülnek ismertetésre.

5.1. Egészértékű programozási modellek

Először megvizsgáljuk, hogyan egyszerűsödnek az előfeldolgozás során kiszámolandó értékek a vizsgált feladat esetében.

Az időindexelt modell előfeldolgozás részeként szükséges kiszámolni a τ_j, ξ_j^t és ζ_j^t értékeket:

$\tau_j = M - (a + l_j + b)$
$\xi_t^j = \{t - a + 1, t - a + 2, t\} \cap \mathcal{T}$
$\zeta_t^j = \{t - (a + l_j + b) + 1, t - (a + l_j + b) + 2, t\} \cap \mathcal{T}.$

Sherali és Smith modellénél az öt eset közül bármelyik előfordulhat, hiszen l_j tetszőlegesen nagy lehet. Nézzük, hogyan alakulnak az $r_{i,j,k}$ és $l_{i,j,k}$ értékek:

k	Feltételek	$l_{i,j,k}$	$r_{i,j,k}$
1	–	$a + l_i + b$	$M - (a + l_j + b)$
2	–	$a + l_i + b - M$	$-(a + l_j + b)$
3	$l_i \geq a,\ l_j \geq b$	$a + l_j \geq l_i + b:\ a$ $a + l_j < l_i + b:\ l_i - l_j + b$	l_i
4	$l_j \geq a,\ l_i \geq b$	$-l_j$	$a + l_i \geq l_j + b : -a$ $a + l_i < l_j + b : -(l_j + b - l_i)$
5	$l_i \geq a + l_j + b$	a	$l_i - l_j - b$
	$l_j \geq a + l_i + b$	$-(l_j - b - l_i)$	$-a$

4. táblázat. A Sherali és Smith modell paraméterei $a_j = a$, $b_j = b$ esetén.

A lineáris rendezési modellnél nincs szükség előfeldolgozásra.

Lássuk, hogyan teljesítenek az egyes modellek. Azt mértem, átlagosan hány másodperc alatt oldja meg az adott modell az ütemezési feladatot n függvényében. Az a és b értékeket minden bemenet során 1 és 10 között, míg az l_j értékeket 10 és 50 között generáltam. Minden n esetén húsz futtatás eredményét átlagoltam.

n	Időindexelt modell (s)	Sherali és Smith modellje (s)	Lineáris rendezési modell (s)
3	0.175762	0.092697	0.106644
4	0.215539	0.287981	0.391705
5	1.414203	1.785091	2.886351
6	4.970265	14.276776	10.626103
7	7.402631	92.310744	73.964368
8	20.54552	–	–
9	48.789491	–	–
10	297.800672	–	–

5. táblázat. Az egészértékű modellek teljesítménye a $1|Coup - Task(a_j = a, b_j = b)|\sum C_j$ variáns mellett.

Minden modellt addig teszteltem, amíg el nem értem egy olyan példányt, amelyen a fent ismertetett véletlenszerű paraméterek mellett már nem futott le sikeresen a kódom.

Látszik, hogy kevés munkából álló feladatok esetében az időindexelt modell teljesít a legjobban. Ez azonban nem jelenti azt, hogy a másik két modell haszontalan lenne, hiszen ahogy az előző fejezetben láttuk, ha növeljük az idősík nagyságát, ennél a modellnél nő a változók száma a leggyorsabban.

5.2. Közelítő algoritmus

Ahhoz, hogy több munkából álló feladatokat is kezelni tudjunk, a továbbiakban egy közelítő algoritmussal dolgozunk. Bár ez a megközelítés nem garantálja az optimális megoldás megtalálását, lehetővé teszi a nagyobb méretű, sok munkából álló feladatok kezelését.

Fischer és Györgyi 2023-ban publikáltak [8] a problémára egy approximációs algoritmust. Az algoritmus először a várakozási idők szerint nem csökkenő sorrendbe rendezi a munkákat, majd ebben a sorrendben mohón ütemezi őket. Belátták, hogy az algoritmus általános esetben legalább 3-közelítő, tehát legfeljebb háromszor nagyobb célfüggvény értéket biztosít, mint az optimum, míg $a \geq b$ esetben legalább 2-közelítő.

Algoritmus 1: Közelítő alg. - $1|Coup - Task(a_i = a, b_i = b)|\sum C_j$

Input : $a,\ b,\ l_j, j = 1 \dots n$

Output: $S_j,\ j = 1 \dots n$

1 Rendezzük a munkákat l_j szerint nem csökkenő sorrendbe;

2 $S_1 := 0$;

3 **for** $j = 2 \dots n$ **do**

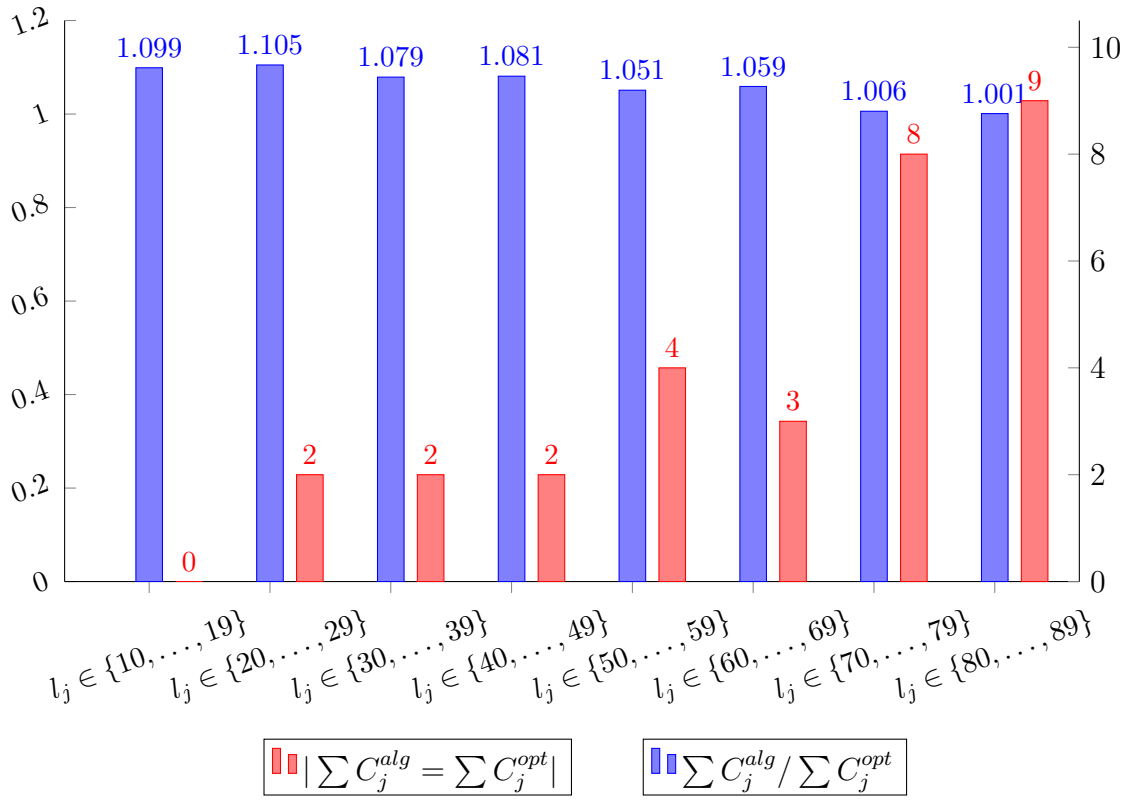
4 ütemezzük a j munkát a lehető leghamarabb

5 **return** $S_j,\ j = 1 \dots n$

A közelítő algoritmus által adott ütemezés megad egy felsőbecslést az idősík nagyságára. Ez felhasználható az időindexelt modell effektivitásának növelésére, ha M-et csökkentjük a kapott felsőbecslésre. Az approximációs algoritmus tesztelésére ezt a módszert alkalmaztam.

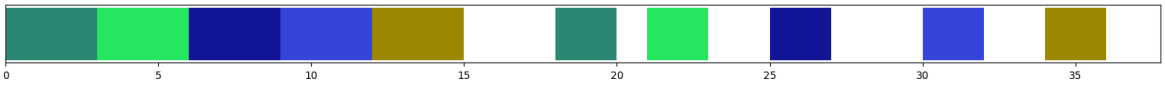
Először nézzük, hogyan teljesít az algoritmus, ha az l_j értékeket szűk tartományból generáljuk. Tíz munkából álló feladatokon vizsgáltam, hányszorosa az algoritmus által adott eredmény az optimumnak. Az a és b értékeket minden esetben 1 és 10 között generáltam véletlenszerűen. Az l_j értékeket az $\{i, i+1, \dots, i+9\}$ halmazokból generáltam $i \in \{10, 20 \dots, 80\}$ mellett. Mind a hat esetben tíz futtatást végeztem.

Jelölje az algoritmus által adott eredményt $\sum C_j^{alg}$. A következő ábrán láthatjuk az $\sum C_j^{alg} / \sum C_j^{opt}$ arányok átlagát az egyes esetekben, illetve hogy a tíz futtatásból, hányszor volt egyenlő a két érték.



8. ábra. A közelítő algoritmus teljesítménye, ha az l_j értékeket szűk intervallumból generáljuk.

Ahogy látszik, az algoritmus egyre jobban teljesít, ahogy a várakozási időket növeljük. Ez nem meglepő, hiszen az l_j értékek növekedésével az algoritmus egyre nagyobb blokkot tud létrehozni. Ha $\min_j l_j \geq na$ és $a \geq b$ akkor minden munka első feladatrésze holtidő és második feladatrész feldolgozása nélkül követi egymást az ütemezésben, ahogy a lenti ábrán látszik.



9. ábra. A közelítő algoritmus által kapott ütemezés, ha $\min_j l_j \geq na$ és $a \geq b$

Most belátjuk, hogy ez az ütemezés optimális.

5.1. Állítás. Ha van olyan megengedett ütemezés, ahol az első feladatrészek ütemezése holtidő, illetve második feladatrész feldolgozása nélkül követi egymást, akkor az optimális a $1|Coup - Task(a_j = a, b_j = b)|\sum C_j$ feladatra.

Bizonyítás. Tegyük fel, hogy létezik ilyen megengedett ütemezés. Nézzük, változik-e a célfüggvény, ha a j-iknek ütemezett munkát átütemezzük a j'-edik helyre, megtartva,

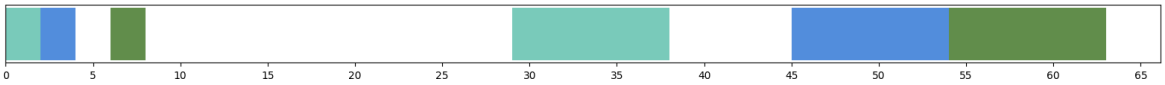
hogy az első feladatrészek ütemezése holtidő nélkül követi egymást (előfordulhat, hogy az ütemezés ezután nem megengedett). Ha $j > j'$, akkor ennek a munkának a befejezési ideje $j' - j$-vel csökken, viszont ugyanennyi munka befejezési ideje eggyel nő, tehát a célfüggvény nem változik. Ugyanígy, ha $j < j'$, akkor az adott munka befejezési ideje $j' - j$-vel nő, viszont ugyanennyi másik munka befejezése eggyel csökken. Tehát minden olyan ütemezés esetében, ahol az első feladatrészek ütemezése holtidő, illetve második feladatrész feldolgozása nélkül követi egymást, a célfüggvény egyenlő, mivel bármelyik két sorrend átrendezhető egymásba munkák egyesével való áthelyezésével.

Az optimalitás bizonyításához azt kell még meggondolni, hogy nem adhat kisebb célfüggvény értéket olyan ütemezés, ahol a munkák nem az állításban leírtak szerint vannak ütemezve. Vegyünk egy ilyen ütemezést. Minden munkát, ha szükséges, ütemezzünk annyival hamarabb, hogy pontosan a közvetlenül előtte ütemezett munka első részének befejezése után kezdődjön a feldolgozása. Ezzel a célfüggvény csökken. Előfordulhat, hogy ez az ütemezés nem megengedett, ekkor munkák áthelyezésével állítsunk elő egy megengedett ütemezést - feltettük, hogy létezik ilyen. Ezzel találtunk az eredetinél kisebb célfüggvényértéket szolgáltató ütemezést. □

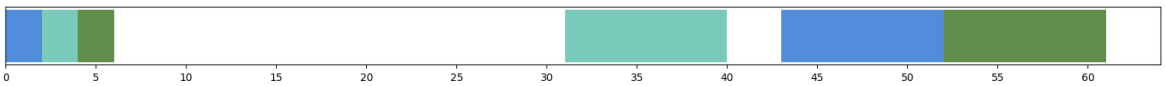
Ha $a < b$, akkor sajnos akkor sem kapunk feltétlenül optimális megoldást az algoritmus által, ha $\min_j \geq nb$. Lássunk erre egy példát.

5.2. Példa. Ütemezzünk három munkát a következő értékek mellett a közelítő algoritmussal, majd lássuk az optimális megoldást.

$a = 2, \quad b = 9 \quad l_1 = 27 \quad l_2 = 41 \quad l_3 = 46$



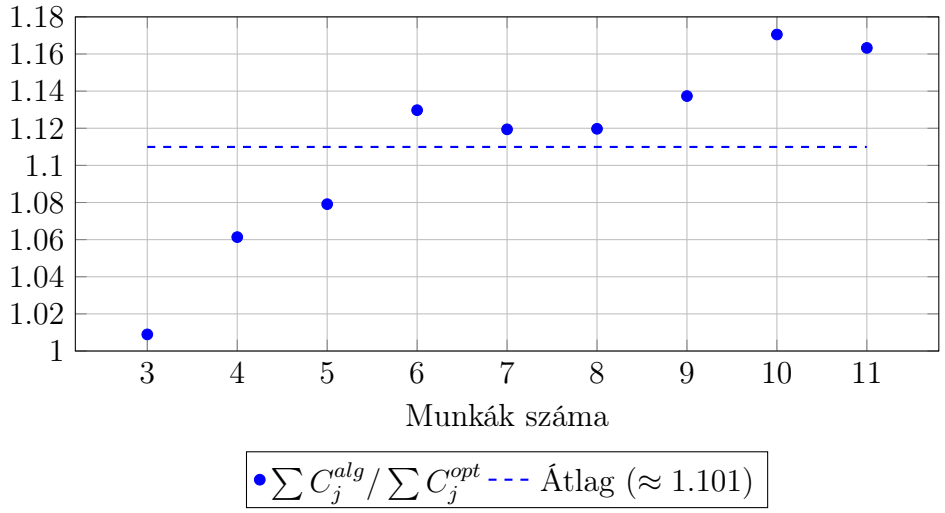
10. ábra. A közelítő algoritmus által kapott ütemezés



11. ábra. Az optimális ütemezés

Az algoritmus által kapott ütemezés célfüggvényértéke: $(a + l_1 + b) + (2a + l_2 + b) + (2a + l_2 + 2b) = 155$. Ezzel szemben az optimális célfüggvényérték: $(a + l_2 + b) + (2a + l_1 + b) + (3a + l_3 + b) = 153$.

Lássuk, hogyan teljesít az algoritmus, ha az l_j értékeket tágabb halmazból generáljuk. Itt kilenc esetet vizsgáltam a munkák száma szerint: $n = 3 \ldots, 11$. Az a és b értékeket itt is 1 és 10 között generáltam, az l_j értékeket pedig 10 és 50 között. Mind kilenc esetben tíz futtatást végeztem. Az átlagolt eredmények a következő ábrán láthatóak:



12. ábra. A közelítő algoritmus teljesítménye, ha az l_j értékeket a $\{10, \ldots, 50\}$ halmazból generáljuk.

Azt tapasztaljuk, hogy a munkák számának növekedésével az algoritmus gyengülő tendenciát mutat.

A következőkben a közelítő algoritmus javításával foglalkozunk.

5.3. Lokális keresés és tabu keresés

*Lokális keresés*nek hívjuk azt a módszert, ami egy kezdeti megoldásból kiindulva iteratívan javítja azt meghatározott operátorok révén addig, amíg el nem ér egy lokális optimumot.

A közelítő algoritmus csupán mohó módon ütemezi a munkákat egy adott sorrendben, így egyszerűen alkalmazható rá a lokális keresés módszere: módosítsuk a munkák sorrendjét, majd értékeljük ki az adott sorrendet mohón ütemezve a munkákat az új sorrend szerint. Ha találunk javulást, frissítsük a sorrendet.

Vezessünk be két operátort: az egyik legyen az *áthelyezés*, a másik a *csere* operátor.

Az áthelyezés operátor sorban veszi az összes munkát, áthelyezi azt minden lehetséges helyre a sorrendben, majd mohón ütemezi a munkákat az új sorrend szerint. Ha

talál olyan ütemezést, aminek a célfüggvényértéke jobb, mint az eredeti sorrend szerint ütemezve a munkákat, akkor visszaadja ezek közül a legjobb célfüggvényértéket biztosító sorrendet.

Algoritmus 2: Áthelyezés operátor

Input : a, b, l_j, $j = 1, \ldots, n$, sorrend
Output: legjobb megoldás, legjobb sorrend
for (i, j), $i \neq j$, $i, j = 1, \ldots, n$ **do**
    helyezzük át az i-ik munkát a j-ik helyre;
    ütemzzük a munkákat az új sorrend szerint;
    if *javulás* **then**
        frissítsük a legjobb sorrendet;
return *legjobb megoldás, legjobb sorrend*

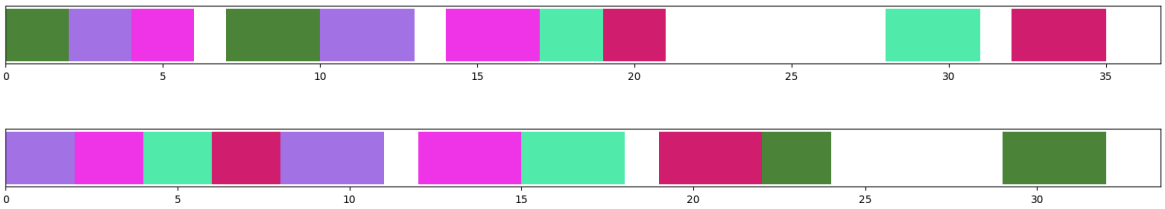
A csere operátor sorban minden lehetséges módon felcseréli két munka helyét a sorrendben, majd mohón ütemezi a munkákat az új sorrend szerint. Ezután ugyanúgy, mint az áthelyezés operátor, ha talál olyan ütemezést, aminek a célfüggvényértéke jobb, mint az eredeti sorrend szerint ütemezve a munkákat, akkor visszaadja ezek közül a legjobb célfüggvényértéket biztosító sorrendet.

Algoritmus 3: Csere operátor

Input : a, b, l_j, $j = 1, \ldots, n$, sorrend
Output: legjobb megoldás, legjobb sorrend
for (i, j), $i \neq j$, $i, j = 1, \ldots, n$ **do**
    Cseréljük fel az i és j munkák sorrendben elfoglalt helyét;
    ütemzzük a munkákat az új sorrend szerint;
    if *javulás* **then**
        frissítsük a legjobb sorrendet;
return *legjobb megoldás, legjobb sorrend*

Nézzünk egy-egy példát a két operátor működésére.

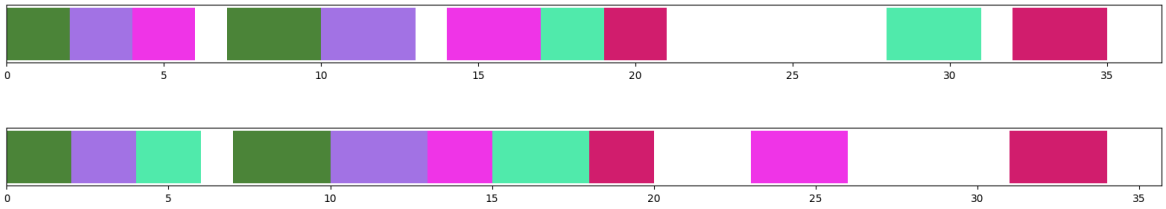
5.3. Példa. Lokális keresés az áthelyezés operátorral



13. ábra. Az áthelyezés operátor egy lépése

Az elsőnek ütemezett munkát átraktuk az utolsó helyre, majd ebben a sorrendben mohón újraütemeztük a munkákat.

5.4. Példa. Lokális keresés a csere operátorral



14. ábra. A csere operátor egy lépése

A másodiknak (rózsaszín) és negyediknek (világoszöld) ütemezett munkák sorrendjét felcseréltük, majd az új sorrend szerint mohón ütemeztük a munkákat.

A lokális keresés effektivitását növeli, ha váltakozva alkalmazzuk az operátorokat. Javítsunk az egyik operátorral, amíg van javító lépés, majd váltsunk a másik operátorra. Ezt a folyamatot ismételjük, míg egy lokális optimumba jutunk.

Algoritmus 4: Lokális keresés operátorok váltakozó alkalmazásával

Input : a, b, l_j, $j = 1 \dots n$
Output: S_j, $j = 1 \dots n$

```
while az ütemezés javítható do
    while a csere operátorral javítható az ütemezés do
        alkalmazzuk a csere operátort;
    while az eltolás operátorral javítható az ütemezés do
        alkalmazzuk az eltolás operátort;
return legjobb megoldás
```

A programozási eredmények ismertetése előtt vezessünk be még egy módszert a közelítő algoritmus lehetséges javítására.

Az előbb látott lokális keresés egy hibája, hogy ha lokális minimumba kerül, az algoritmus rögtön leáll. Ezt a problémát kezelhetjük a *tabu keresés*sel.

A tabu keresés során megengedünk olyan lépéseket, ami nem javítja a legjobb megoldásunkat, abban az esetben, ha egy lokális minimumba vagyunk. Hogy elkerüljük annak a lehetőségét, hogy a következő pár lépés során visszakerüljünk ugyanabba az állapotba, fenntartunk egy tabu listát - ebben tároljuk az utolsónak megtett lépéseket, amiket nem engedünk meg. Ennek nagyságát előre inicializáljuk, és ha megtelt, a legrégebben berakott lépést töröljük. Előfordulhat, hogy egy tiltott lépés egy új megoldásba

visz, aminek célfüggvényértéke jobb, mint az addig talált legjobb megoldásunk (tehát nem csak az aktuális megoldásunknál jobb, hanem a tabu keresés során talált eddigi legjobb): ebben az esetben megengedjük a tiltott lépést - ezt *aspiráns kritérium*nak nevezzük.

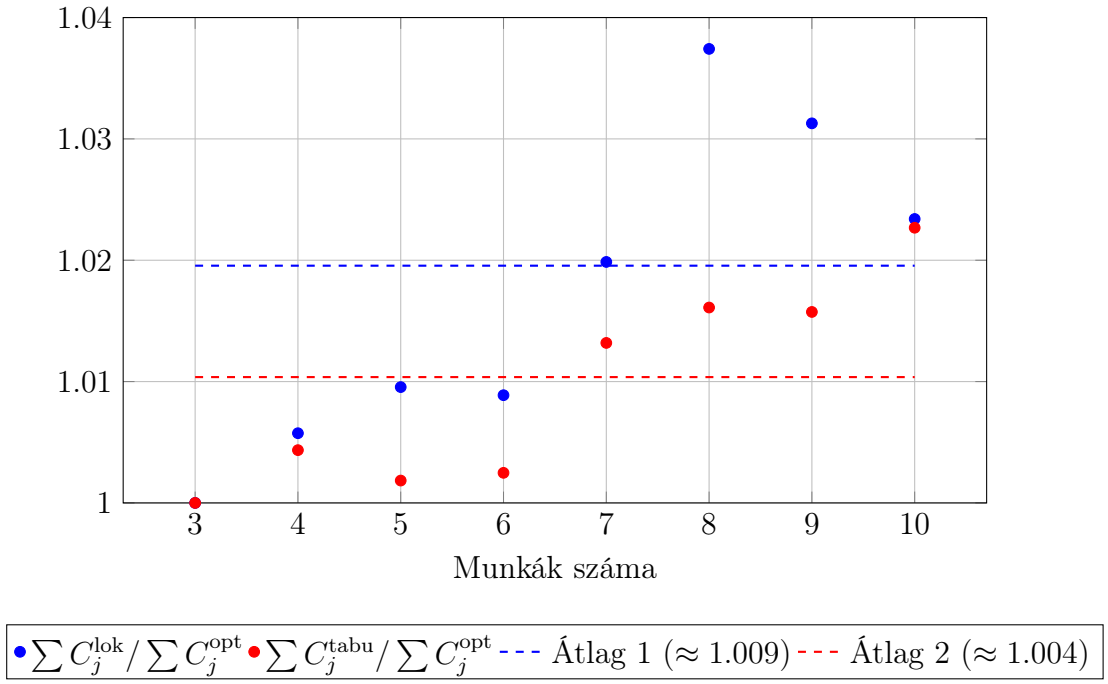
A tabu keresés implementálásánál ugyanúgy a csere és az áthelyezés műveleteket alkalmaztam új megoldás keresésére.

Algoritmus 5: Javítás tabu kereséssel

Input : a, b, l_j, $j = 1 \dots n$, *tabu_meret*, *max_iter*
Output: S_j, $j = 1 \dots, n$

```
1  inicializáljuk az aktuális sorrendet és költséget;
2  legjobb megoldás ← aktuális megoldás;
3  tabu lista ← üres;
4  for 1 to max_iter do
5      szomszédlista ← üres;
6      for (i, j), i ≠ j,    i, j = 1, . . . , n do
7          cseréljük fel az i és j munkák sorrendben elfoglalt helyét;
8          ütemzzük a munkákat az új sorrend szerint;
9          if a lépés nem szerepel a tabu listában then
10             vegyük fel a szomszédlistába;
11         else if az új megoldás jobb, mint a legjobb megoldás then
12             vegyük fel a szomszédlistába;
13         helyezzük át az i-ik munkát a j-ik helyre;
14         ütemzzük a munkákat az új sorrend szerint;
15         if a lépés nem szerepel a tabu listában then
16             Vegyük fel a szomszédlistába;
17         else if az új megoldás jobb, mint a legjobb megoldás then
18             Vegyük fel a szomszédlistába;
19     if a szomszédlista nem üres then
20         válasszuk ki a legjobb lépést;
21         frissítsük az aktuális megoldást és sorrendet;
22         if az új megoldás jobb, mint a legjobb megoldás then
23             frissítsük a legjobb megoldást;
24         adjuk hozzá a választott lépést a tabu listához;
25         if a tabu lista hossza meghaladja tabu_méret-t then
26             távolítsuk el a legrégebbi elemet;
27     else
28         alkalmazzunk egy véletlenszerű cserét vagy áthelyezést új megoldás
             előállításához;
29         adjuk hozzá az új lépést a tabu listához;
30         if a tabu lista hossza meghaladja tabu_méret értékét then
31             Távolítsuk el a legrégebbi lépést a tabu listából;
32 return legjobb megoldás
```

Nézzük hogyan teljesít a lokális keresés operátorok váltakozó alkalmazásával, illetve a tabu keresés algoritmus. Először hasonlítsuk össze a teljesítményüket az optimummal az $n = 3, \ldots, 10$ esetekben. Az a és b értékeket itt is 1 és 10 között, míg az l_j értékeket 10 és 50 között generáltam. Minden n esetében tíz inputon teszteltem. Jelölje a lokális keresés által kapott célfüggvényértéket $\sum C_j^{lok}$, a tabu keresés által kapott célfüggvényértéket pedig $\sum C_j^{tabu}$.



15. ábra. A lokális keresés és tabu keresés teljesítménye az optimumhoz képest.

Mindkét algoritmus közel optimális eredményeket szolgáltatott, viszont ebből messzemenő következtetéseket nem lehet levonni, hiszen kevés munkából álló feladatokon a közelítő algoritmus is nagyon jól teljesített.

Nézzük most, hogyan teljesítenek a javító algoritmusok a közelítő algoritmushoz képest. Itt is az a és b értékeket 1 és 10 között, az l_j értékeket 10 és 50 között generáltam. A tabu lista mérete 10, míg az iterációk száma 100 volt a tabu keresésnél. Húsz darab ötven munkából álló inputon futtattam mindhárom algoritmust. A lokális keresés átlagosan $12,18\%$-kot, míg a tabu keresés $12,71\%$-ot javított a közelítő algoritmus által adott célfüggvényértéken.

6. $1|Coup - Task(l_i = l)| \sum C_j$

Ebben a fejezetben a $1|Coup-Task(l_i = l)| \sum C_j$ feladat vizsgálatával foglalkozunk az előző fejezet felépítéséhez hasonlóan.

6.1. Egészértékű programozási modellek

Tekintsük át, hogyan teljesítenek a már ismertetett matematikai programozási modellek. Ehhez először vizsgáljuk, hogyan alakulnak az előfeldolgozások során kiszámolandó értékek.

Az időindexelt modell használatához a τ_j, ξ_j^t és ζ_j^t paramétereket szükséges meghatározni:

$\tau_j = M - (a_j + l + b_j)$
$\xi_j^t = \{t - a_j + 1, t - a_j + 2, t\} \cap \mathcal{T}$
$\zeta_j^t = \{t - (a_j + l + b_j) + 1, t - (a_j + l + b_j) + 2, t\} \cap \mathcal{T}$.

Sherali és Smith modellénél az öt eset közül csak az első négy fordulhat elő, mivel minden munka várakozási ideje egyenlő. Nézzük, hogyan alakulnak az $r_{i,j,k}$ és $l_{i,j,k}$ értékek:

k	Feltételek	$l_{i,j,k}$	$r_{i,j,k}$
1	–	$a_i + l + b_i$	$M - (a_j + l + b_j)$
2	–	$a_i + l + b_i - M$	$-(a_j + l + b_j)$
3	$l \geq b_i,\ l \geq a_j$	$a_j \geq b_i:\ a_i$ $a_j < b_i:\ a_i + b_i - a_j$	$a_i + l - a_j$
4	$l \geq a_i,\ l \geq b_j$	$-(a_j + l - a_i)$	$a_i \geq b_j : -a_j$ $a_i < b_j : -(a_j + b_j - a_i)$

6. táblázat. A Sherali és Smith modell paraméterei $l_j = l$ esetén.

Itt is azt mértem, átlagosan hány másodperc alatt oldja meg az adott modell az ütemezési feladatot a munkák számának függvényében. Minden $l \in 10, 20, \ldots, 50$ értékhez öt a_j és b_j értéket generáltam 1 és 10 között. Összesen tehát minden n-re 25 futtatást végeztem, majd ezek eredményét átlagoltam. A teszteléseket most is addig végeztem,

amíg el nem értem egy olyan példányt, amin az adott modell minden rendelkezésre álló memóriát felhasznált, és a futtatási környezet összeomlott.

n	Időindexelt modell (s)	Sherali és Smith modellje (s)	Lineáris rendezési modell (s)
3	0.134708	0.043588	0.094660
4	0.301731	0.257005	0.439868
5	0.540930	0.946389	1.814009
6	1.192092	7.191716	6.916243
7	3.215810	38.529877	42.574422
8	9.684820	227.18492	–
9	13.061183	–	–
10	22.456726	–	–
11	55.687853	–	–

7. táblázat. Az egészértékű modellek teljesítménye a $1|Coup - Task(l_j = l)|\sum C_j$ variáns mellett.

Hasonlóan a $1|Coup - Task(a_j = a, b_j = b)|\sum C_j$ feladat esetében kapott eredményekhez, itt is azt tapasztaltam, hogy az időindexelt modell a leghatékonyabb. A fentebb vázolt véletlenszerűen generált paraméterek mellett, ezzel a modellel legfeljebb 11 munkából álló feladatokat sikerült megoldanom. Ezzel szemben a lineáris rendezési modellel csupán 7 munkából álló bemenetekre futott le a programom. Ez nem zárja ki, hogy egyes konkrét bemenetek esetén nagyobb munkákból álló feladatokat is meg tudnak oldani a modellek, csupán az általam generált bemenetek között mindig volt olyan, amit már nem tudott kezelni.

6.2. Közelítő algoritmus

A következő részben rátérünk egy közelítő algoritmus vizsgálatára, amivel lehetővé válik több munkákból álló feladatok kezelése.

A $1|Coup - Task(l_i = l)|\sum C_j$ variánsra szintén Fischer és Györgyi adtak köze-

lítő algoritmust 2023-as publikációjukban [8]. Belátták, hogy az algoritmus általános esetben 3-közelítő, míg az $a_j = b_j$ megszorítás mellett 1.5-közelítő.

Az algoritmus először rendezi a munkákat $a_j + b_j$ szerint nem csökkenő sorrendbe, majd ebben a sorrendben ütemezi azokat. Ha a soron következő j munka ütemezhető közvetlenül a_{j-1} feldolgozása után holtidő nélkül, a $C_{a_{j-1}}$ időpontban, akkor ide ütemezi. Ha ez nem megengedett, akkor megpróbálja úgy ütemezni b_{j+1}-t, hogy a b_{j+1} feladatrész feldolgozása közvetlenül b_{j-1} feldolgozása után kezdődjön holtidő nélkü, tehát a C_{j-1} időpontban. Ha ez sem lehetséges, akkor a j munkát a b_{j-1} feldolgozása után ütemezi holtidő nélkül, ekkor $S_j = C_{j-1}$.

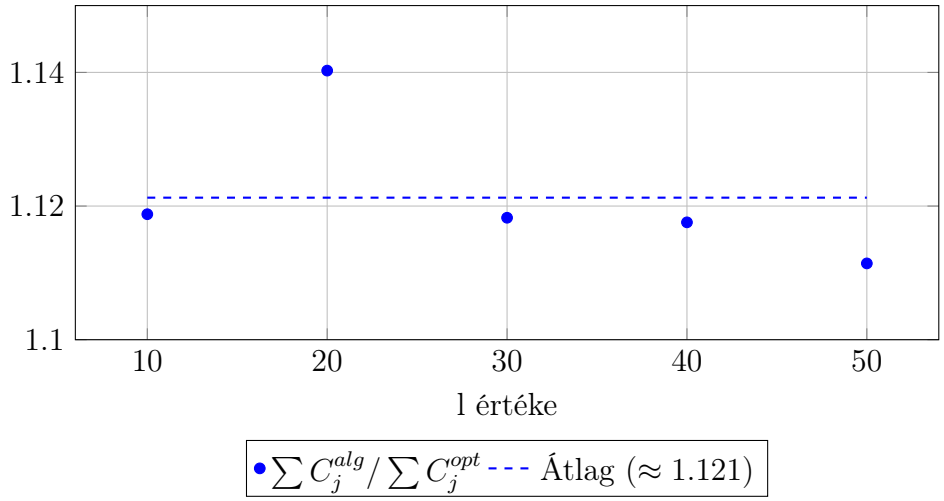
Algoritmus 5: Közelítő algoritmus - $1|Coup - Task(l_i = l)| \sum C_j$

Input : $(a_j, b_j) \quad j = 1 \dots n$, l
Output: $(s_j)_{j=1}^{n} \quad j = 1 \dots n$

```
1  Rendezzük a munkákat a_j + b_j szerint nem csökkenő sorrendbe;
2  s_1 := 0;
3  for j = 2 ... n do
4      if a_j ütemezhető holtidő nélkül közvetlenül a_{j-1} után anélkül, hogy a
         feldolgozása egybeesne egy már ütemezett feladatrész feldolgozásával then
5          Ütemezzük a_j-t közvetlenül a_{j-1} után;
6          s_j := s_{j-1} + a_{j-1}
7      else
8          if b_j ütemezhető holtidő nélkül közvetlenül b_{j-1} után anélkül, hogy az a_j
             feldolgozása egybeesne egy már ütemezett feladatrész feldolgozásával
             then
9              Ütemezzük b_j-t közvetlenül b_{j-1} után;
10             s_j := s_{j-1} + a_{j-1} + b_{j-1} − a_j
11         else
12             Ütemezzük a_j-t közvetlenül b_{j-1} után;
13             s_j := s_{j-1} + a_{j-1} + b_{j-1} + l;
```

Lássük először, hogyan teljesít az algoritmus az optimumhoz képest. Ahogy már láttuk, az időindexelt modell effektivitása növelhető, ha az M értéket egy megfelelő C_{max} felsőbecslésre cseréljük. A közelítő algoritmus által adott ütemezés révén meghatározhatunk egy ilyen felsőbecslést. Ezt a módszert használva 12 munkából álló feladatokon teszteltem a közelítő algoritmust.

Minden $l \in \{10, 20, \dots, 50\}$ érték esetében tíz futtatást végeztem, az a és b értékeket 1 és 10 között generáltam. Az átlagolt eredmények a következő ábrán láthatóak, ahol a közelítő algoritmus által szolgáltatott célfüggvényértéket $\sum C_j^{alg}$ jelöli.



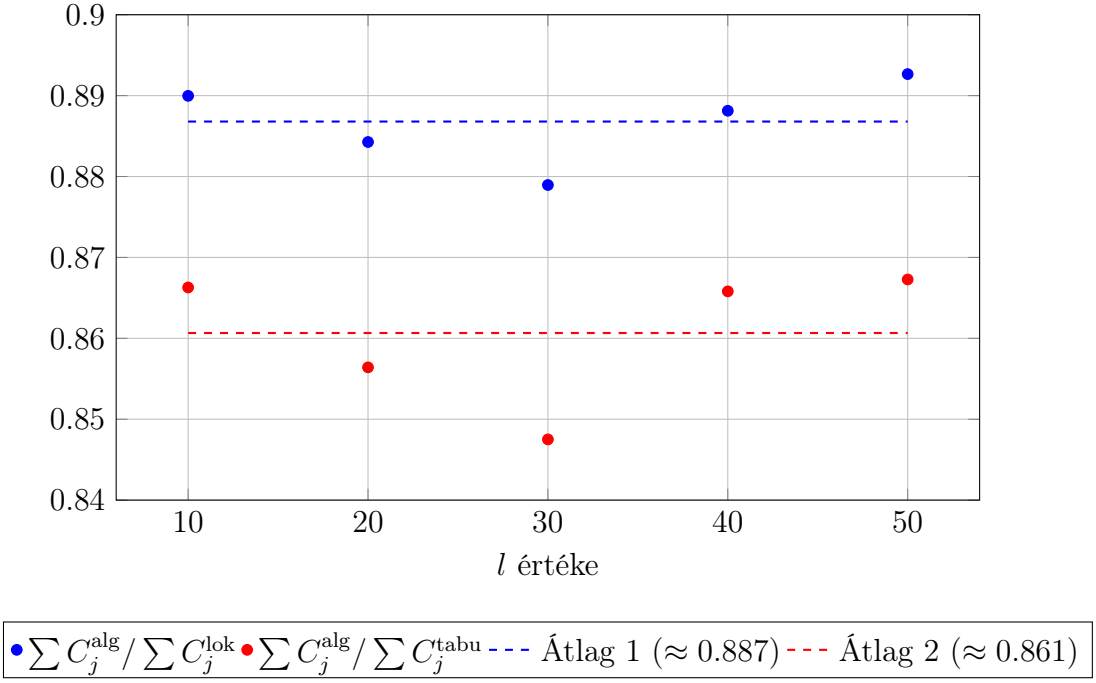
16. ábra. A közelítő algoritmus teljesítménye az optimumhoz képest.

6.3. Lokális keresés és tabu kersés

Az algoritmus által szolgáltatott eredmény itt is javítható az előző fejezetben ismertetett lokális keresés és tabu keresés módszerekkel. A lokális keresés operátorait módosítjuk úgy, hogy kiértékelő algoritmusként az $1|Coup - Task(l_j = l)| \sum C_j$ variánsra bevezetett közelítő algoritmust használják. Ugyanígy módosítsuk a tabu keresés ütemező algoritmusát is.

Lássuk, mennyit javítanak az előzőleg bevezetett közelítő algoritmuson a lokális keresés operátorok váltakozó alkalmazásával és tabu keresés módszerek. Itt is minden $l \in \{10, 20, \ldots, 50\}$ érték esetében tíz futtatást végeztem 50 munkából álló feladatokra. Az a és b értékeket 1 és 10 között generáltam. A tabu lista mérete 10, míg az iterációk száma 100 volt a tabu keresésnél. Az átlagolt eredmények a lenti ábrán láthatók. Jelölje a lokális keresés által kapott célfüggvényértéket $\sum C_j^{lok}$, a tabu keresés által kapott célfüggvényértéket pedig $\sum C_j^{tabu}$.

A lokális keresés átlagosan $12,33\%$-ot, míg a tabu keresés $13,93\%$-ot javított a közelítő algoritmus által adott eredményen.



17. ábra. A lokális keresés és tabu keresés javítása a közelítő algoritmushoz képest.

7. Összegzés

A dolgozat első felében ismertettük a páros munkák ütemezése feladat általános jellemzőit, áttekintettük a hozzá kapcsolódó legfontosabb bonyolultsági eredményeket, majd bemutattunk matematikai programozási modelleket. Ezeket átültettük a legkésőbbi befejezési idő minimalizálása célfüggvényről a befejezési idők összegének minimalizálására.

A dolgozat második felében a hangsúlyt a gyakorlati megoldási lehetőségekre helyeztük. Két speciális esetet vizsgáltunk: a $1|Coup - Task(a_j = a, b_j = b)|\sum C_j$ és a $1|Coup - Task(l_j = l)|\sum C_j$ problématípusokat. Mindkét esethez ismertettünk egy-egy approximációs algoritmust. Bár a korlátozott számítási kapacitás miatt csak kevés munkából álló feladatokon volt lehetőségünk az optimummal összehasonlítani ezek hatékonyságát, ezeken a bemeneteken azt tapasztaltuk, hogy az algoritmusok a gyakorlatban jelentősen jobban teljesítenek, mint amit az elméleti eredmények sugallnak.

A közelítő algoritmusokat a lokális keresés, illetve tabu keresés módszerekkel javítottuk, hogy sok munkából álló feladatokat is minél jobban tudjunk kezelni. A két módszer javításai között nem tapasztaltunk lényeges különbséget, azonban a tabu keresés paramétereinek változtatásával lehet, hogy jobb eredmények is elérhetők lennének – ezzel a dolgozat keretében nem foglalkoztunk.

Hivatkozások

[1] Ageev, A. A., Baburin, A. E. (2007). Approximation algorithms for UET scheduling problems with exact delays. Operations Research Letters, 35(4), 533–540.

[2] Ageev, A. A., Kononov, A. V.(2006). Approximation algorithms for scheduling problems with exact delays. In Approximation and online algorithms, volume 4368 of Lecture Notes in Computer Science (pp. 1–14.).

[3] Baptiste, P. (2010). A note on scheduling identical coupled tasks in logarithmic time. Discrete Applied Mathematics 158(5), 583–587.

[4] Békési, J., Galambos, G., Jung, M. N., Oswald, M., and Reinelt, G. (2014). A branch-and-bound algorithm for the coupled task problem. Mathematical Methods of Operations Research 80(1), 47–81.

[5] Condotta, A., Shakhlevich, N. V. (2012). Scheduling coupled operation jobs with exact time-lags. Discrete Applied Mathematics, 160, 2370–2388.

[6] Chen, B., Zhang, X. (2021). Scheduling coupled tasks with exact delays for minimum total job completion time. Journal of Scheduling, 24(2), 209–221.

[7] Elshafei, M., Sherali, H. D., and Smith, J. C. (2004). Radar pulse interleaving for multitarget tracking. Naval Research Logistics (NRL) 51(1), 72–94.

[8] Fischer, D., Györgyi, P. (2023). Approximation algorithms for coupled task scheduling minimizing the sumofcompletiontimes. Annals of Operations Research (2023) 328:1387–1408.

[9] Garey, M. R., Johnson, D. S. (1979). Computers and Intractability: A Guide to the Theory of NP-Completeness.

[10] Hwang, F. J., Lin, B. M. T. (2011). Coupled-task scheduling on a single machine subject to a fixed-job-sequence. Computers Industrial Engineering, 60(4), 690–698.

[11] Khatami, M., Salehipour, A., Cheng, T. C. E. (2020). Coupled task scheduling with exact delays: Literature review and models. European Journal of Operational Research, 282(1), 19–39.

[12] Orman, A. J., Potts, C. N. (1997). On the complexity of coupled-task scheduling. Discrete Applied Mathematics., 72(1-2):141–154.

[13] Shapiro, R. D. (1980). Scheduling coupled tasks. Naval Research Logistics Quarterly 27(3), 489–498.

[14] Simonin, G., Giroudeau, R., König, J. (2011). Complexity and approximation for scheduling problem for a torpedo. Computers Industrial Engineering, 61(2), 352–356.

[15] Sherali, H. D., Smith, J. C. (2005). Interleaving two-phased jobs on a single machine. Discrete Optimization 2 348–361.

[16] Yu, W., Hoogeveen, H., Lenstra, J. K. (2004). Minimizing makespan in a two-machine flow shop with delays and unit-time operations is NP-hard. Journal of Scheduling, 7, 333–348.

Nyilatkozat

Alulírott Markó Anna Erzsébet nyilatkozom, hogy a szakdolgozat elkészítése során a következő feladatok elvégzésére alkalmaztam a GPT-40 eszközt: 5. fejezet ábrák generálása (29., 35. o.).