dc_2018_22

Hozzájárulás kombinatorikus optimalizálási problémák egy osztályához

Doktori értekezés tézisei

Békési József

Szegedi Tudományegyetem
Szeged, 2022

dc_2018_22

1. Bevezetés

Az MTA doktori értekezésem az elmúlt 15 évben elért kutatási eredményekből tartalmaz egy válogatást, amelyeknek a jelentős része az utóbbi 5 évből származik. A fő kutatási területem kombinatorikus optimalizálási algoritmusok, módszerek tervezése, fejlesztése, illetve ezek elemzése elméleti és tapasztalati módszerekkel. A kombinatorikus optimalizálás a diszkrét és alkalmazott matematika egyik fontos területe, amely szorosan kapcsolódik a számítástudományhoz is. Gyorsan fejlődő kutatási területről van szó, amely az utóbbi két évtizedben alkalmazási szempontból is nagyon fontossá vált. A diszkrét optimalizálási modelleket sikeresen alkalmazták olyan területeken, mint a gazdaság, környezettudományok, közlekedés, ipari termelés, stb. Kutatásaim során általában közvetlenül, vagy közvetett módon különböző gyakorlati problémákhoz kapcsolódó feladatokat vizsgáltam. A közlekedési alkalmazásokhoz kapcsolódóan foglalkoztam járműütemezési és útvonal optimalizálási, ipari alkalmazások témájában gépütemezési és pakolási feladatokkal. Emellett bemutatok elméleti eredményeket bizonyos párhuzamos számítógépes modellek témájából is. Az algoritmusok egy részét ipari projektek keretében valósítottuk meg, az eredmények valós adatokon történő tesztelések során keletkeztek. Más eredmények inkább alapkutatási jellegűnek tekinthetők. Az eredményeket igyekeztem egy szélesebb spektrumból, a kombinatorikus optimalizálás különböző területeiről válogatni, hogy ezzel is reprezentálni tudjam a teljes kutatási tevékenységemet. A disszertációban az elért eredményeket mutatom be, amelyek nagyobbik részben szerzőtársakkal közös publikációkon alapulnak. Mind a dolgozatban, mind a tézisekben azok az eredmények szerepelnek, amelyeknél a hozzájárulásomat jelentősnek tekintem és ezzel a szerzőtársaim is egyetértettek.

2. Ládapakolási problémák

A ládapakolás az egyik klasszikus probléma a kombinatorikus optimalizálás területén. Az egydimenziós változat az alábbiak szerint definiálható. Legyen $L = \{a_1, a_2, \ldots, a_n\}$ egy n elemű lista, amelynek minden eleméhez egy $s(a_i) \in (0, 1]$, $i = 1, \ldots, n$. méret tartozik. A feladat az, hogy az elemeket minimális számú egységnyi kapacitású ládához rendeljük, azzal a megkötéssel, hogy az egyes ládákhoz rendelt elemek összmérete legfeljebb 1 lehet. Közismert, hogy a probléma NP-nehéz (Johnson, 1973 [23]). Ezért sok közelítő algoritmust fejlesztettek ki az elmúlt 40 évben. A közelítő algoritmusok minősége központi kérdés az algoritmuselméletben. Számos módszer létezik az algoritmus jóságának mérésére: kísérleti vizsgálat, legrosszabb-eset, illetve

versenyképességi és valószínűségi elemzés. A ládapakolási probléma esetén is szokás vizsgálni az online változatot. Egy online algoritmus egyenként pakolja az elemeket. A aktuális elem érkezésekor semmit sem tud a lista fennmaradó részéről: sem az elemek száma, sem a méretük nem ismert. Nyilvánvaló, hogy az online algoritmusok nem rendelkeznek elegendő információval ahhoz, hogy olyan jó pakolást hozzanak létre, mint egy offline algoritmus, ami a teljes bemenetet ismeri. A ládapakolásban az online algoritmusok versenyképességi elemzésére két versenyképességi hányadost szokás használni. Legyen A egy online algoritmus és I egy tetszőleges bemenet. Jelöljük $A(I)$-vel, illetve $OPT(I)$-vel az A megoldását és az optimális megoldást. Az A algoritmus *abszolút versenyképességi hányadosát* az alábbiak szerint határozzuk meg:

$$R_A := \sup_I \left\{ \frac{A(I)}{OPT(I)} \right\}.$$

A másik mérőszám az *aszimptotikus versenyképességi hányados*, aminek a definíciója a következő:

$$R_A^\infty := \limsup_{k\to\infty} \left\{ \max_I \left\{ \frac{A(I)}{k} \middle| OPT(I) = k \right\} \right\}.$$

Az abszolút versenyképességi hányadosra ismert az optimális érték, ami $\frac{5}{3}$ [5]. Az aszimptotikus versenyképességi hányadosra ismert legjobb alsó korlát $\frac{1363-\sqrt{1387369}}{120} \approx 1.5427809064729$ [6].
A következő alfejezetekben az egydimenziós ládapakolási problémára kifejlesztett algoritmussal, illetve három speciális változattal kapcsolatos eredményeinket ismertetem.

2.1. Az Advanced Harmonic algoritmus

Az AH algoritmus
Az algoritmus osztópontok egy adott sorozatát használja, amelyek pontos definíciója a következő: $1 = t_0 > t_1 = \frac{1}{2} > t_2 > \cdots > t_b = \frac{1}{3} > \cdots > t_M > t_{M+1} = 0$. Minden j-re a $(t_j, t_{j-1}]$ intervallumba eső elemeket j osztályú elemeknek nevezzük. Az elemek egy osztályát *hatalmasnak* hívjuk ha $j = 1$, *nagynak* ha $1 < j \leq b$, *kicsinek* ha $b < j \leq M$, és *aprónak* ha az elemek mérete legfeljebb t_M. Az algoritmus az elemeket online módon részládákba pakolja, amelyek mindig azonos osztályú elemeket tartalmaznak, de bizonyosak kombinálhatók, egy láda legfeljebb két részládából tevődhet össze. Legyen $\gamma_j = \lfloor \frac{1}{t_{j-1}} \rfloor$ $j \leq M$-re. Minden j nagy vagy kicsi osztályra és minden i-re ($1 \leq i \leq \gamma_j$) adott egy nemnegatív α_{ij} paraméter, ahol $0 \leq \alpha_{ij} \leq 1$. α_{ij} jelöli az i számosságú ládák arányát a j osztály ládái között, amelyekre

a $\sum_i \alpha_{ij} = 1$ egyenlőség tejesül minden j-re. Azokat a részládákat, amelyek i elemet tartalmaznak a j osztályból, a j osztály *i-típusú* részládáinak nevezzük. Az algoritmus pakolási szabályai a részládákra vonatkozóan a következők:

- Egy hatalmas elemet egyedül pakolunk a részládába.
- Kis elemek esetén a következő érkező elemet mindig a meglévő részládába pakoljuk, ha van elég hely. Ha az összes meglévő részláda tele van, új részládát nyitunk és az arányok alapján eldöntjük annak típusát.
- Az apró elemek osztályánál hasonló módszereket alkalmazunk, de itt a részládák típusait az elemek teljes méretétől függően határozzuk meg.
- A nagy elemek a pakolása során az osztályában 4 részláda típust különböztetünk meg:
  - Egy kettes típusú szabályos részláda, amely már tartalmaz 2 elemet az osztályból.
  - Egy kettes típusú deklarált részláda, amely jelenleg egy elemet tartalmaz, de AH úgy döntött, hogy két elem lesz itt. Csak egy állandó számú ilyen részláda van.
  - Egy egyes típusú szabályos részláda, amely egy elemet tartalmaz, és már nem tud többhöz jutni a későbbiekben.
  - Egy egyes típusú ideiglenes részláda, amely egy elemet tartalmaz és feltehetőleg még egyhez juthat a későbbiekben.
- Az egyes típusú szabályos és az egyes típusú ideiglenes részládák teljes számának arányát rögzítjük, de az ezen típusokon belüli arányokat nem.
- Amennyiben meg kell növelnünk a kettes típusú részládák számát az egyes típusú ideiglenes részládák egyikének kettes típusra való cseréjével, akkor az új elemet egy olyan ládába pakoljuk, amelynek az eleme a legnagyobb ezen ládák között.
- Ha egy új egyes típusú részládára van szükség és ezért egy új ládát nyitunk meg, akkor ez a részláda egyes típusú ideiglenes részláda lesz.

A részládák ládákba pakolása az összeméretük alapján történik. A részládák összméretét a következőképpen értelmezzük:

- Az elem mérete a részládában, amennyiben csak egy eleme van.

- Az ilyen típusú részládákba illeszthető elemek összméretének felső határa, ha egynél több elemet tartalmazhat.
- A nagy elemekre vonatkozó speciális szabályok: egy egyes típusú ideiglenes részláda mérete megegyezik az elem méretével, egy kettes típusú deklarált mérete annyi, mint egy kettes típusú szabályosé.

A részládák pakolási szabályai a következők:

- Két részláda belefér egy ládába, amennyiben az összméretük legfeljebb 1.
- Megpróbálunk egy $\frac{1}{2}$-nél nagyobb méretű részládát összeilleszteni egy legfeljebb $\frac{1}{2}$ méretű részládával, mindig a legnagyobb méretűvel, amivel összefér.

2.1. Tétel. ([4]) *Az AH algoritmus aszimptotikus versenyképességi hányadosa legfeljebb* 1.57828956.

2.2. Alsó korlát kötegelt ládapakolási problémára

A *kötegelt ládapakolási probléma* (Batched Bin Packing Problem, BBPP) fogalmát Gutin és munkatársai vezették be [21]. Az elemek kötegekben érkeznek és egyszerre egy köteg pakolható egy adott időben. Minden köteg különféle méretű elemeket tartalmazhat és üres is lehet. Ha $K \geq 2$ köteg van, akkor K-BBPP-ről beszélünk. Egy *kötegelt algoritmus* a köteget teljesen el kell hogy pakolja, mielőtt a következő köteg megérkezik. Nyilvánvaló, hogy ha minden köteg egy elemet tartalmaz, akkor a klasszikus online problémáról beszélünk, és ha csak egy köteg jön, akkor a probléma az általános (offline) egydimenziós ládapakolási feladat. Tekintsük az L bemeneti sorozatot, amely egy *kötegelt sorozat*, azaz $L = \{B_1, B_2, \ldots, B_K\}$, ahol a B_j elemek egy halmaza, $1 \leq j \leq K$. Általában a K darab kötegből sorozatot jelöljük $\mathcal{B}(K)$-val. Legyen A egy kötegelt algoritmus, akkor a BBPP esetében az aszimptotikus versenyképességi hányados (ACR) definíciója a következő:

$$R^{\infty}_{A,K} := \limsup_{N\to\infty} \left\{ \frac{A(L)}{\mathrm{OPT}(L)} : L \in \mathcal{B}(j), \quad j \leq K, \quad \mathrm{OPT}(L) = N \right\}.$$

Gutin és szerzőtársai az [21] cikkben egy 1.3871...-es alsó korlátot adtak tetszőleges online 2-BBPP algoritmus aszimptotikus versenyképességi hányadosára. Nyilvánvaló, hogy

$$R^{\infty}_{A,i} \leq R^{\infty}_{A,j} \leq \ldots, \quad \text{ha} \quad 1 \leq i < j < \infty.$$

A [3] cikkben a 3-BBPP-t vizsgáltuk és a következőkben bemutatom az elért eredményeinket.
Az alsó korlát bizonyításához használt konstrukció a következő:

- Az első köteg (B_1) $n_1 = 6jn$ darab azonos méretű pici elemet tartalmaz, amelyeket a_1-gyel jelölünk. Legyen $j \geq 4$ fix egész szám, ekkor a köteg minden elemének mérete $s(a_1) = 1/6j = \varepsilon$.

- A második kötegben a $B_{2,k}$ listák egyikét adjuk. A $B_{2,k}$ lista $n_{2,k} = \frac{6j}{j-k}n$ darab $a_{2,k}$ elemet tartalmaz, melyek mérete $s(a_{2,k}) = \frac{1}{3} + k\varepsilon - \frac{\varepsilon}{3} = \frac{1}{3} + \varepsilon\frac{3k-1}{3}$, ahol $1 \leq k \leq j-1$.

- A $B_{2,k}$ köteget a $B_{3,k}$ köteg követi. $B_{3,k}$-ban $n_{3,k} = \frac{6j}{j-k}n$ darab a_3 elem található, melyek mérete $s(a_3) = \frac{1}{2} + \frac{\varepsilon}{3}$.

Az alsó korlát bizonyításához a fenti három kötegtípust használjuk, amelyekből listákat képezünk a következő módon: $(B_1, B_{2,1}, B_{3,1})$, $(B_1, B_{2,2}, B_{3,2})$, $\ldots$, $(B_1, B_{2,j-1}, B_{3,j-1})$. Jelölje $(i_1, i_2, i_3)_1$, $(i_1, i_2, i_3)_{2,k}$, illetve $(i_1, i_2, i_3)_{3,k}$ egy láda típusát a B_1, $B_{2,k}$, illetve a $B_{3,k}$ kötegek elpakolása után. Ha egy láda az $(i_1, i_2, i_3)_{2,k}$ állapotban van, akkor nem pakolt az algoritmus bele $B_{3,k}$ elemet, vagyis bármilyen $(i_1, i_2, i_3)_{2,k}$ típusú ládára $i_3 = 0$. Egy (i_1, i_2, i_3) hármast *valós pakolási mintának* (vagy *lehetséges pakolási mintának*) nevezünk, ha

$$i_1 s(a_1) + i_2 s(a_{2,k}) + i_3 s(a_{3,k}) \leq 1.$$

A lehetséges pakolási minták halmazát V-vel jelöljük. Definiáljuk az alábbi részhalmazokat is:

$$V_t = \{v \in V \mid \ i_t > 0 \text{ és } i_r = 0, \text{ ha } r < t\}, \ t = 1, 2, 3.$$

Nyilvánvalóan $V_t \cap V_r = \emptyset$ ha $t \neq r$.
A [3] cikkben megfogalmazottak alapján elegendő olyan algoritmusokat vizsgálni, amelyek $B_1, B_{2,k}, B_{3,k}$-t úgy pakolják, hogy a pakolási minták az alábbi ládatípusokat tartalmazzák:

$$(i_1, 0, 0)_1, \quad (i_1, 1, 0)_{2,k}, \quad (i_1, 2, 0)_{2,k}, \quad (0, 1, 1)_{3,k}, \quad (0, 2, 0)_{2,k}, \quad (0, 0, 1)_{3,k}.$$

Egy tetszőleges online algoritmus aszimptotikus versenyképességi hányadosának kiszámítását a 3-BBPP-re lineáris program segítségével végezzük. A lineáris program feltételei az elemek számára és az algoritmusok alsó korlátainak lehetséges értékeire vonatkoznak. Az jelölés egyszerűsítése érdekében

$R^{\infty}_{A,3}$ helyett R-t fogunk használni. Az első feltétel az első köteg elemeinek számára vonatkozik.

$$\sum_{(i_1,i_2,i_3)\in V_1} i_1 x^k_{i_1,i_2,i_3} = \sum_{i_1=1}^{6j} i_1 x^k_{i_1,i_2,i_3} = n_1 \tag{1}$$

ahol $x^k_{i_1,i_2,i_3}$ jelöli az (i_1, i_2, i_3) típusú ládák számát a $B_1, B_{2,k}, B_{3,k}$ kötegek pakolása során. Hasonló feltételeket adhatunk a $B_{2,k}$ kötegekben szereplő elemek számára.

$$\sum_{i_1=2j-2k+1}^{4j-k} x^k_{i_1,1,0} + 2\sum_{i_1=1}^{2j-2k} x^k_{i_1,2,0} + x^k_{0,1,1} + 2x^k_{0,2,0} = n_{2,k}, \quad k = 1, 2, \ldots, j-1. \tag{2}$$

A következő $j-1$ egyenlet a $B_{3,k}$ kötegek elemeinek számára vonatkozik:

$$x^k_{0,1,1} + x^k_{0,0,1} = n_{3,k}, \quad k = 1, 2, \ldots, j-1. \tag{3}$$

Három alsó korlátra vonatkozó feltétel is szükséges.

$$\sum_{i_1=1}^{6j} x^k_{i_1,i_2,i_3} \le R \cdot \mathrm{OPT}(B_1) \tag{4}$$

$$\sum_{i_1=1}^{6j} x^k_{i_1,i_2,i_3} + x^k_{0,1,1} + x^k_{0,2,0} \le R \cdot \mathrm{OPT}(B_1, B_{2,k}) \tag{5}$$

$$\sum_{i_1=1}^{6j} x^k_{i_1,i_2,i_3} + x^k_{0,1,1} + x^k_{0,2,0} + x^k_{0,0,1} \le R \cdot \mathrm{OPT}(B_1, B_{2,k}, B_{3,k}) \tag{6}$$

Az alsó korlát kiszámításához azt a lineáris programozási feladatot tekintjük, amely tartalmazza a fenti $2(j+1)$ darab feltételt ((1) - (6)) és a célfüggvényben az R-t szeretnénk minimalizálni. A [3] cikkben tetszőleges j értékre kiszámoltuk a fenti LP feladat megoldását, ami így j-t a végtelenbe tartatva megadta a kívánt értéket.

2.2. Tétel. ([3]) *Ha A egy kötegelt algoritmus a 3-BBPP-re, akkor*

$$R^{\infty}_{A,3} \ge \frac{32\,W\left(-1, -\frac{9}{4}\mathrm{e}^{-\frac{23}{8}}\right) + 36}{24\,W\left(-1, -\frac{9}{4}\mathrm{e}^{-\frac{23}{8}}\right) + 33} \approx 1.51211383...$$

ahol $W(-1, x)$ a Lambert függvény negatív ága.

A számítás során megkaptuk a különböző j értékhez tartozó alsó korlátokat is, valamint azt a d értéket, ami meghatározza, hogy a feltételek közül mennyit szükséges feltétlenül szerepeltetni az LP-ben a kívánt érték eléréséhez.

j	d	R
5	1	1.480075901
10	3	1.494928787
20	6	1.503357743
50	15	1.508573181
100	30	1.510335641
200	60	1.511221192
500	152	1.511757013
1000	305	1.511935384

1. táblázat. R értékei különböző j-k esetén

2.3. Elemszám–korlátos ládapakolás

Az elemszám-korlátos ládapakolási probléma esetén a bemenet a ládapakolási feladatnak megfelelő, de van egy globális $k \geq 2$ paraméter, amelyet elemszám korlátnak neveznek. A cél az, hogy az elemeket minimális számú ládába pakoljuk úgy, hogy mindegyikben az elemek összmérete legfeljebb 1 lehet és a száma nem haladhatja meg a k-t.

Vegyük a következő bemenetet egy adott Π ládapakolási problémára. Legyen $\theta \geq 2$ fix pozitív egész szám. Adott elemek θ darab listája, ahol az L_i lista $(1 \leq i \leq \theta)$ azonos méretű elemeket tartalmaz, az elemek mérete s_i, ahol $s_1 < s_2 < \cdots < s_\theta$. Egy $N > 0$ nagy egész számra az L_i listának $\alpha_i \cdot N$ eleme van, ahol $0 < \alpha_i \leq 1$ egy racionális szám paraméter $i = 1, \ldots, \theta$-re (N-et úgy választjuk hogy $\alpha_i \cdot N$ egész szám legyen). Az elemeket nemcsökkenő sorrendben adjuk egy online algoritmusnak és ezt bármikor befejezhetjük. Vagyis van θ lehetséges bemenet, amelyek egymás után jöhetnek és vizsgáljuk egy tetszőleges online algoritmus viselkedését ebben az esetben. A következőkben bemutatunk ilyen típusú alsó korlát konstrukciókat különböző k értékekre.

Tekintsük a $k = 5$ esetet. Legyen $\theta = 4$, $\alpha_1 = \frac{1}{2}$, $\alpha_i = 1$, $i = 2, 3, 4$-re. Legyen $0 < \delta < \frac{1}{2000}$, $s_1 = \frac{1}{42} - \delta > 0$, $s_2 = \frac{1+\delta}{7}$, $s_3 = \frac{1+\delta}{3}$, és $s_4 = \frac{1+\delta}{2}$. Nézzük a $k = 7, 8, \ldots, 11$ eseteket. Legyen $\theta = 4$, az elemek méretei legyenek ugyanazok mint a $k = 5$ esetben, továbbá $\alpha_1 = \frac{k-6}{6}$ (vagyis $\alpha_1 < 1$) és $\alpha_i = 1$ $i = 2, 3, 4$-re.

2.3. Tétel. ([9]) *A következő értékek alsó korlátok az aszimptotikus versenyképességi hányadosra vonatkozóan.*

- $\frac{3}{2} = 1.5$ $k = 5$-*re.*
- $\frac{k^2+24k}{k^2+10k+24}$ $k = 7, 8$-*ra. Ez az érték* $217/143 \approx 1.5174825$ $k = 7$-*re és* $\frac{32}{21} \approx 1.5238095$ $k = 8$-*ra.*
- $\frac{10.5}{62/9} = \frac{189}{124} \approx 1.5241935$*,* $k = 9$-*re.*
- $\frac{k^2+84k}{k^2+48k+36}$ $k = 10, 11$-*re. Ez az érték* $235/154 \approx 1.525974$ $k = 10$-*re és* $\frac{209}{137} \approx 1.525547$ $k = 11$-*re.*

A következőkben tekintsük a $14 \leq k \leq 18$ eseteket. Legyen $\theta = 4$, $\alpha_i = 1$ $i = 1, 2, 3, 4$-re. Legyen $0 < \delta < \frac{1}{2000}$, $s_1 = \frac{1}{18} - 3\delta > 0$, $s_2 = \frac{1+\delta}{9}$, $s_3 = \frac{1+\delta}{3}$ és $s_4 = \frac{1+\delta}{2}$.

2.4. Tétel. ([9]) *A* $\frac{45k}{29k+6}$ *érték alsó korlát az aszimptotikus versenyképességi hányadosra, ahol* $14 \leq k \leq 18$*.*

Végül legyen $k = 19, 20, \ldots, 35$. Legyen $\theta = 5$, $\alpha_1 = \frac{k-18}{18}$, $\alpha_i = 1$ $i = 2, 3, 4, 5$-re. Legyen $0 < \delta < \frac{1}{10000}$, $s_1 = \frac{1}{342} - \delta$, $s_2 = \frac{1+\delta}{19}$, $s_3 = \frac{1+\delta}{9}$, $s_4 = \frac{1+\delta}{3}$ és $s_5 = \frac{1+\delta}{2}$.

2.5. Tétel. ([9]) *A 2. táblázatban megadott értékek alsó korlátok az aszimptotikus versenyképességi hányadosra a* $k = 19, 20, \ldots, 35$ *esetben.*

k	korábbi alsó korlát	új alsó korlát	
5	1.47058 [19]	3/2	=1.5 [9]
7	1.5 [28]	217/143	≈1.51748 [9]
8	1.5 [28]	32/21	≈1.52380 [9]
9	1.5 [28]	189/124	≈1.52419 [9]
10	1.50943 [19]	235/154	≈1.52597 [9]
11	1.51724 [19]	209/137	≈1.52554 [9]
14	1.52595 [19]	315/206	≈1.52912 [9]
15	1.52912 [19]	75/49	≈1.53061 [9]
16	1.52567 [19]	72/47	≈1.53191 [9]
17	1.52312 [19]	765/499	≈1.53306 [9]
18	1.52459 [19]	135/88	≈1.53409 [9]
19	1.52678 [19]	30799/20072	≈1.53442 [9]
20	1.52912 [19]	2365/1541	≈1.53471 [9]
21	1.52941 [19]	13251/8633	≈1.53492 [9]
22	1.52914 [19]	10417/6786	≈1.53507 [9]
23	1.53004 [19]	49795/32434	≈1.53527 [9]
24	1.53086 [19]	152/99	≈1.53535 [9]
25	1.53162 [19]	54175/32284	≈1.53539 [9]
26	1.53231 [19]	3523/2294	≈1.53574 [9]
27	1.53296 [19]	2439/1588	≈1.53589 [9]
28	1.53356 [19]	1897/1235	≈1.53603 [9]
29	1.53412 [19]	70789/46079	≈1.53625 [9]
30	1.53465 [19]	6105/3974	≈1.53623 [9]
31	1.53514 [19]	84103/54742	≈1.53635 [9]
32	1.53560 [19]	39104/25449	≈1.53656 [9]
33	1.53603 [19]	23925/15568	≈1.53680 [9]
34	1.53644 [19]	289/188	≈1.53723 [9]
35	1.53682 [19]	76195/49569	≈1.53715 [9]

2. táblázat. Új alsó korlátok az aszimptotikus versenyképességi hányadosra. A második oszlop a korábbról ismert alsó korlátokat, a harmadik a javított értékeket tartalmazza.

2.4. NF-alapú tárkorlátos ládapakolási algoritmus

Zheng és szerzőtársai [30] egy gyakorlati problémát modelleztek az egydimenziós online ládapakolási probléma egy speciális változatával, és kidolgoztak egy félig online algoritmust a megoldásra. Algoritmusukban egy puffert

használtak az elemek ideiglenes tárolására, ezáltal lehetőségük volt arra, hogy előre tekintsenek az érkező elemek listájában. A gyakorlati probléma miatt az úgynevezett 2-paraméteres esetet vizsgálták, azaz amikor $\max_{a \in L} s(a) \leq 1/2$. Az algoritmusuk minden lépésben a puffer legnagyobb elemeit helyezi a új nyitott ládába a Next Fit szabály (NF) használatával: a puffer tartalmának pakolásakor az algoritmus új, üres ládát nyit, és – néhány egyszerű szabályt követve – iteratív módon a puffer elemeit ebbe a ládába helyezi, amíg belefférnek, majd bezárja a ládát. Ez azt jelenti, hogy legfeljebb 1 láda van nyitva a pakolás során. Azokat az algoritmusokat, amelyek konstans számú nyitott ládát használnak, *tárkorlátos algoritmusoknak* nevezzük. Ha legfeljebb 1 nyitott láda van, akkor az algoritmust *NF-alapú félig online algoritmusnak* nevezzük. Az NF-alapú algoritmusok hatékonyságát elemezve bizonyították, hogy az ACR értéke $\frac{13}{9}$ tetszőleges 1-nél nem kisebb pufferméretnél. $\frac{4}{3}$-os alsó határt adtak azokra a tárkorlátos algoritmusokra, amelyek NF-alapúak.
Zhang és szerzőtársai szintén ezt a problémát vizsgálták [29]. Két algoritmust ismertettek, az első 2 méretű puffert használt, és bebizonyították, hogy az algoritmus ACR értéke 1.4375... 3 méretű puffer használatával tovább javították a felső korlátot. Algoritmusuk ACR-je így 1.4243... volt. Végül megadtak egy alsó korlátot is, ami 1.4230..., ez szintén jobb a korábbról ismertnél.
A [13] cikkben az állandó méretű puffert használó NF-alapú, félig online algoritmusokat vizsgáltuk az általános r-paraméteres esetben. Olyan L listákat tekintettünk, ahol $\max_{a \in L} s(a) \leq \frac{1}{r}$ egy adott r egész számra ($r \geq 1$). Az alsó korlát bizonyításához az úgynevezett Sylvester sorozatot használtuk, aminek a definíciója adott $k > 1$ és $r \geq 1$-re a következő:

$$m_1^r = r + 1, \qquad m_2^r = r + 2, \qquad m_j^r = m_{j-1}^r (m_{j-1}^r - 1) + 1, \text{ ha } j = 3, \ldots, k.$$

m_j^r	$r = 1$	$r = 2$	$r = 3$	$r = 4$	$r = 5$
$j = 1$	2	3	4	5	6
$j = 2$	3	4	5	6	7
$j = 3$	7	13	21	31	43
$j = 4$	43	157	421	931	1807
$j = 5$	1807	24493	176821	865831	3263443

3. táblázat. Az általánosított Sylvester sorozat néhány tagja $k = 5$-re

Definiáljuk a következő értékeket:

$$h_\infty(r) = 1 + \sum_{i=2}^{\infty} \frac{1}{m_i^r - 1}.$$

A $h_\infty(r)$ sorozat első néhány tagja: $h_\infty(1) \approx 1.69103, h_\infty(2) \approx 1.42312$, $h_\infty(3) \approx 1.30238$.

2.6. Tétel. ([13]) *Tekintsük az r-paraméteres tárkorlátos esetet konstans méretű pufferrel. Ha A egy tetszőleges NF-alapú félig online algoritmus, akkor $R_\infty(A) \geq h_\infty(r)$.*

A következőkben egy súlyfüggvényt definiálunk, ami szokásos technika a ládapakolási algoritmusok elemzésénél.

$$W(x) = \begin{cases} x + \dfrac{1}{m_i(m_i - 1)}, & \text{if } \frac{1}{m_i} < x \leq \frac{1}{m_i - 1} \\ \dfrac{m_i + 1}{m_i} x, & \text{if } \frac{1}{m_{i+1} - 1} < x \leq \frac{1}{m_i} \end{cases}$$

Egy láda súlya a benne lévő elemek súlyának az összege, illetve általában egy halmaz súlya az elemei súlyának az összege.

A következőkben egy ládát *jó ládának* nevezünk, ha a benne lévő elemek súlyának összege legalább egy, és egy halmaz *jó részhalmaz*, ha elemei súlyának az összege nagyobb vagy egyenlő egynél és a méretek összege legfeljebb egy. Természetesen egy jó láda tartalmaz egy jó részhalmazt. Egy 3 nagyságú puffert veszünk és három virtuális ládát fogunk alkalmazni – egy kapacitással – a pufferben lévő elemek előfeldolgozására, mielőtt a ládába helyeznénk őket.

Az NFFD-B3 algoritmus

(1) Töltsük fel a puffert a lista következő elemeivel, amíg azok belefférnek.

(2) Rendezzük a pufferben lévő elemeket nem növekvő sorrendbe, és pakoljuk az elemeket három virtuális ládába – amelyek jelölése $VBIN_i$, $i = 1, 2, 3$ – az FFD szabályt használva. Azok az elemek, amelyek nem férnek el egyik virtuális ládában sem a pufferben maradnak.

(3) Ellenőrizzük a virtuális ládák tartalmát. Azoknak, amelyek jó ládák, nyissunk új üres ládát és pakoljuk bele a tartalmukat, majd zárjuk be azt. Lépjünk az (5) pontra.

(4) Keressünk egy jó részhalmazt a $VBIN_i$, $i = 1, 2, 3$ ládák tartalmában, nyissunk új üres ládát és pakoljuk bele a tartalmukat, majd zárjuk be azt.

(5) Ha van még elpakolatlan elem lépjünk (1)-re.

(6) Ürítsük ki a virtuális ládák tartalmát új ládákba, majd zárjuk be ezeket és lépjünk ki.

2.7. Tétel. ([13])

$$R_\infty(\textit{NFFD-B3}) = h_\infty(r).$$

3. Gépütemezési problémák

A gépütemezési problémákat széles körben vizsgálták az utóbbi évtizedekben, sok változata létezik, melyekkel számos cikk foglalkozott. Általánosan úgy fogalmazhatjuk meg a problémát, hogy adottak valamilyen feladatok, amelyek az általános esetben több tevékenységből is állhatnak és ezeket gépekhez szeretnénk rendelni. A tevékenységek a gépeket adott időtartamig használják, célunk az indulási idők meghatározása úgy, hogy valamilyen adott célfüggvény szerint az ütemezés optimális legyen. A diszkrét optimalizálás területén általában fontos szerepük van az online problémáknak, illetve algoritmusoknak. Ekkor az algoritmus inputja részenként érkezik, és végleges döntéseket kell hozni az input további részeinek teljes vagy részleges ismerete nélkül. Első esetben online, második esetben félig online problémáról beszélünk. Amennyiben az input teljesen ismert az algoritmus számára, akkor beszélünk offline algoritmusról. A gépütemezési problémák esetén is mindhárom változatot széles körben vizsgálták. A következő eredmények egy speciális online változatra vonatkoznak [22].

3.1. Alsó korlátok eltérő sebességű gépekre

A problémánál adott gépek egy halmaza, amelyek sebessége eltérő lehet, továbbá adottak feladatok, vagy munkák a feldolgozási idejükkel. A feladatok egy *ütemezése* minden munkát valamelyik géphez rendeli. A feladat végrehajtásához szükséges idő megegyezik a megadott feldolgozási idő és a gép sebességének hányadosával. Cél a maximális befejezési idő minimalizálása.

A probléma online verziójában a munkák egyenként jelennek meg. Amikor egy munka megjelenik, az online algoritmusnak egy visszavonhatatlan döntést kell hoznia, és a feladatot hozzárendelheti egy géphez. Ez a döntés végleges és a jövőbeli munkák ismerete nélkül történik, az algoritmus még azt sem tudja, hogy léteznek-e jövőbeli munkák vagy sem. Az online algoritmus *R-versenyképes*, ha minden input esetén egy olyan ütemezést készít, amelyre a befejezési idő legfeljebb az optimális R-szerese.

Az alsó korlát egy olyan példán alapul, ahol a feldolgozási idők egy mértani sorozatot alkotnak. Ezt a sorozatot már a korábbiakban is vizsgálták. [14, 18, 17]. Léteztek eredmények 2 és 3, illetve tetszőleges számú gépre. A sebességeket úgy kell megválasztani, hogy bármely online algoritmus csak a leggyorsabb gépeket használhassa, ezek az úgynevezett aktív gépek. Az alsó korlátot ezután ezen gépeken az ütemezés lehetséges mintáinak elemzésével kapjuk. Általánosítva ezt 3-nál több gépre, a lehetséges minták kiküszöbölésére számítógépes elemzést használva több alsó korlátot bizonyítottunk különböző gépszámokra. A korábbi és az általunk bizonyított alsó korlátokat, valamint a legjobb ismert algoritmusok korlátait az 4. táblázatban foglaljuk össze.

	alsó korlátok		algoritmusok	
m	korábbi	saját	legjobb	LS
2	$\phi \approx 1.618$	–	–	$\phi \approx 1.618$
3	2	–	–	2
4	–	**2.141391**	–	2.2248
5	–	**2.314595**	–	2.4143
6	2.2880	**2.347312**	–	2.5812
7	–	**2.439957**	–	< 2.7321
8	–	**2.439957**	–	< 2.8709
9	2.4380	**2.462775**	–	≤ 3
10	–	**2.483120**	–	< 3.1214
11	–	**2.502672**	–	< 3.2361
∞	2.5648		5.8284	$\Theta(\log m)$

4. táblázat. Korábbi és saját alsó korlátok; LS a List Scheduling, vagyis a mohó algoritmus, m a gépek száma ([22]).

3.2. Egységnyi végrehajtási idejű páros munka ütemezési algoritmus elemzése

A páros munka ütemezési feladatot (CTP) a következőképpen definiáljuk: adott n munka, mindegyik két részfeladatból áll. A két részfeladatot adott sorrendben kell végrehajtani és a végrehajtásuk között pontos késési időt kell betartani. A i. munkát ($i = 1, \ldots, n$) az (a_i, l_i, b_i) pozitív egész számokkal adjuk meg, ahol az értékek az első részfeladat feldolgozási idejét, a késleltetési időt, illetve a második részfeladat feldolgozási idejét jelentik. A késleltetés idő alatt a gép tétlen állapotban van és más munkák feldolgozhatók ebben az időintervallumban. A cél az, hogy az n feladatot egyetlen gépen ütemezzük oly módon, hogy ne legyen két részfeladat átfedésben és a legutoljára ütemezett munka befejezési ideje a lehető legkisebb legyen. Az általános esetre a szokásos három részből álló jelölést használjuk – Graham és szerzőtársai vezették be [20] – ami a következő: $1|\mathit{Coup\text{-}Task}, \mathit{exact}\ l_i|C_{\max}$.

Legyen $S(n)$ n páros munkából álló lista és legyen A egy közelítő algoritmus. Jelöljük $C_{\max}(\mathrm{A}, S(n))$-nel és $C_{\max}(\mathrm{OPT}, S(n))$-nel az A algoritmus, illetve egy optimális algoritmus által kiszámolt maximális befejezési időt $S(n)$-re. Az A algoritmust *ρ-közelítő algoritmusnak* nevezzük ($\rho \geq 1$) ha

$$C_{\max}(\mathrm{A}, S(n)) \leq \rho\, C_{\max}(\mathrm{OPT}, S(n))$$

minden bemeneti listára. A lehető legkisebb ilyen ρ értéket az A algoritmus

legrosszabb-eset hányadosának nevezzük és ρ_{A}-val jelöljük.
Ageev és Baburin [1] a következő algoritmust definiálta az egységnyi hosszú részfeladatokat tartalmazó esetre (UET), azaz amikor $a_i = b_i = 1$, $i = 1, \ldots, n$.

A DNF algoritmus

(1) Rendezzük a munkákat a késleltetési idejük szerint nemcsökkenő sorrendbe, azaz $l_1 \leq l_2 \leq \ldots \leq l_n$.

(2) Hozzuk létre a σ_1 ütemezést a következő módon:

(2.1) Kezdjük az 1-es munkát a 0. pozíción.

(2.2) Az $i = 2, \ldots, n$ munkákra indítsuk az i. munkát a legkorábbi lehetséges helyen az előző munka első részfeladata után.

(3) Legyen k azon munkák száma, amelyek teljesen befejeződnek az n. munka indítása előtt.

(4) Hozzuk létre a σ_2 ütemezést a következőképpen (csak ha $k > 0$):

(4.1) Indítsuk a $k+1$. munkát a 0 pozíción.

(4.2) Az $i = k+2, \ldots, n, 1, \ldots, k$ munkákra indítsuk az i. munkát a legkorábbi lehetséges helyen az előző munka első részfeladata után.

(5) Legyen σ az algoritmus kimenete, ami a σ_1 és σ_2 ütemezések közül a jobbik.

Ageev és Baburin a következő tételt bizonyította.

3.1. Tétel. ([1]) *A DNF algoritmus legrosszabb-eset hányadosára teljesül a következő*

$$\rho_{\mathrm{DNF}} \leq \frac{7}{4}.$$

Az [11] cikkben egy új alsó korlátot igazoltunk az UET típusú problémákra, valamint egy konstrukcióval bizonyítottuk a $\frac{7}{4}$-es korlát élességét.

3.2. Tétel. ([11]) *Tegyük fel hogy* $\sum_{i=1}^{n} l_i > n(n-1)$. *Ekkor tetszőleges* $S(n)$ *listára*

$$C_{\max}(\mathrm{OPT}, S(n)) \geq 2n + \left\lceil \frac{1}{n}\Big(\sum_{i=1}^{n} l_i - n(n-1)\Big) \right\rceil.$$

A konstrukció a következő:
Legyen $n = 4m + 2$, $m \geq 2$-re. Definiáljuk az $S_I(n)$ probléma sorozatot a következőképpen

$$\begin{aligned} l_i &= 2m - 1, \quad i = 1, \ldots, 2m - 1, \\ l_{2m} &= 4m, \\ l_i &= 4m + 1, \quad i = 2m + 1, \ldots, 4m, \\ l_{4m+2} &= 8m + 1. \end{aligned}$$

3.1. Lemma. ([11]) *Az $S_I(n)$ probléma sorozatra*

$$C_{\max}(\mathrm{DNF}, S_I(n)) = \frac{7n - 6}{2}.$$

3.2. Lemma. ([11]) *Az $S_I(n)$ probléma sorozatra*

$$C_{\max}(\mathrm{OPT}, S_I(n)) = 2n.$$

3.3. Tétel. ([11]) *A DNF algoritmus legrosszabb-eset hányadosa $\frac{7}{4}$.*

3.3. First Fit típusú algoritmus a páros munka ütemezési feladatra

A következőkben a *First-Fit Decreasing* (*FFD*) közelítő algoritmust definiáljuk a páros munka ütemezési feladat egy speciális esetére, amely az alábbi:

$$1|\mathit{Coup\text{-}Task}, \mathit{exact}\ l_i \in \{L_1, L_2\},\ a_i = b_i = 1|C_{\max},$$

ahol L_1 és L_2 adott pozitív egész számok.

Az FFD algoritmust eredetileg D. S. Johnson [23] definiálta, mint ládapakolási algoritmust. A következőkben definiáljuk a megfelelő verzióját a problémánkra. Természetesen az algoritmust úgy adjuk meg, hogy tetszőleges számú különböző késleltetéssel oldja meg a problémát, de csak két különböző késés értékre elemezzük.

A First-Fit Decreasing algoritmus

(1) Rendezzük a munkákat a késleltetéseik szerint nemnövekvő sorrendbe. Ezután tegyük fel, hogy $l_1 \geq l_2 \geq \ldots \geq l_n$. Legyen $i = 1$.

(2) Ütemezzük a J_i munkát a legkorábbi időpontra, amelyre lehetséges ütemezést kapunk.

(3) $i = i + 1$. Ha $i \leq n$ akkor lépjünk (2)-re, egyébként álljunk meg.

3.4. Tétel. ([10]) *Legyen $I(n,k)$ egy olyan példa, amely $n = n_1 + n_2 = 9k$ munkát tartalmaz, ahol $n_1 = 3k$ és $n_2 = 6k$ a hosszú, illetve a rövid munkák száma. Tegyük fel, hogy $L_1 = 12k - 2$ és $L_2 = 9k - 2$. Ekkor*

$$\limsup_{k\to\infty} \frac{C_{\max}^{FFD}(I(n,k))}{\mathrm{OPT}(I(n,k))} = \frac{30}{19}.$$

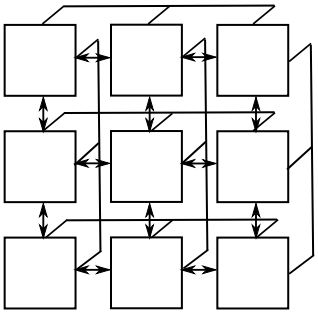
3.5. Tétel. ([10]) *Azon UET típusú problémák esetén, ahol csak két különböző késleltetésű munkánk van, az FFD algoritmus legrosszabb eset hányadosára a következő teljesül:*

$$1.57894\ldots = \frac{30}{19} \leq \rho_{FFD} \leq \frac{\sqrt{11}+3}{4} = 1.579156\ldots,$$

és az alsó korlát éles ha $L_1/n_1 \leq 4$.

4. A mátrix transzponálási probléma

Az alkalmazott tudományok gyorsan növekvő számítási igényei a számítógépes rendszereket fokozatosan a nagyobb számítási kapacitás felé vitték. Ugyanakkor hatékonyabb algoritmusokra volt szükség. Egy jó architektúra, vagy egy hatékonyabb algoritmus csökkentheti a feldolgozási időt a párhuzamos számítási környezetekben. A hardver oldalán a párhuzamos számítás hatékonysága erőteljesen függ attól, hogy milyen gyorsan tudunk adatokat küldeni a forrásprocesszorról a rendeltetési helyre. A különböző architektúrák között az úgynevezett hálószerűen összekapcsolt processzorokból álló rendszereket széles körben vizsgálták. A legegyszerűbb – egydimenziós – esetben a hálózat egy lineáris tömb, amelyben minden processzor kétirányú összeköttetéssel kapcsolódik szomszédaihoz. Magasabb dimenziókban a processzorok valódi tömböt alkotnak és egymással hálószerűen vannak összekapcsolva. Egyes esetekben a párhuzamos kommunikáció hatékonyságát buszokkal segítik. Amennyiben buszt használunk, azt mindig összekötjük néhány processzorral. Az így összekapcsolt processzorok nemcsak szomszédaikkal, hanem bármelyik, hozzájuk busszal kapcsolódó processzorhoz küldhetnek egy lépésben üzenetet. Fontos azonban, hogy egy lépésben csak egyetlen processzor használhat egy adott buszt. Általánosan az az elterjedt, hogy kétdimenziós hálózatok esetén sor- és oszlopbuszokat használunk. Ilyen architektúra esetén az egy sorban ill. az egy oszlopban elhelyezkedő processzorokat kötjük össze egy-egy busszal. Ezeket *sor-* ill. *oszlopbuszok*nak nevezzük.



1. ábra. Sor- és oszlop buszok

A kétdimenziós hálózatokon definiálható egy speciális permutációs üzenettovábbítási probléma a *mátrix transzponálási probléma* (*MTP*). E feladatnál minden i,j-re, ahol $1 \leq i,j \leq n$, az (i,j) indexű forrásban tárolt üzeneteket a (j,i) indexű célhoz kell eljuttatni. Kétdimenziós busz nélküli esetben Ding, Ho és Tsai [16] elemezte az *MTP*-nek azt a verzióját, amikor a processzorokban k darab mátrix elemeit tároljuk, és a feladat ezeknek a mátrixoknak a transzponálása. Ez az ún. $k-k$ verzió. Ha egy A algoritmus vizsgálatakor $T_A(k,n)$-nel jelöljük k darab $n \times n$-es mátrix transzponálásához szükséges lépések számát, akkor [16] alapján a következő alsó korlát igaz: tetszőleges *MTP*-t megoldó A algoritmusra, $T_A(k,n) \geq (1-1/\sqrt{2})kn \approx 0.293kn$. Kaufmann, Meyer és Sibeyn [24] definiált egy algoritmust, amely a megoldáshoz $0.301kn + O(n/k)$ lépést igényel. A [12] cikkben megmutattuk, hogy sor- és oszlopbuszok használatával javíthatjuk a mátrix transzponálási probléma megoldási algoritmusainak hatékonyságát. Először a kétdimenziós esetre bizonyítottunk egy alsó korlátot.

4.1. Tétel. ([12]) *Legyen A egy tetszőleges algoritmus és jelölje $T_A^B(1,n)$ az MTP megoldásához szükséges lépések számát az $n \times n$-es kétdimenziós, buszokkal ellátott hálózaton. Ekkor*

$$\lim_{n\to\infty} \frac{T_A^B(1,n)}{n} \geq 2 - \frac{2}{5}\sqrt{15} \approx 0.450806\ldots$$

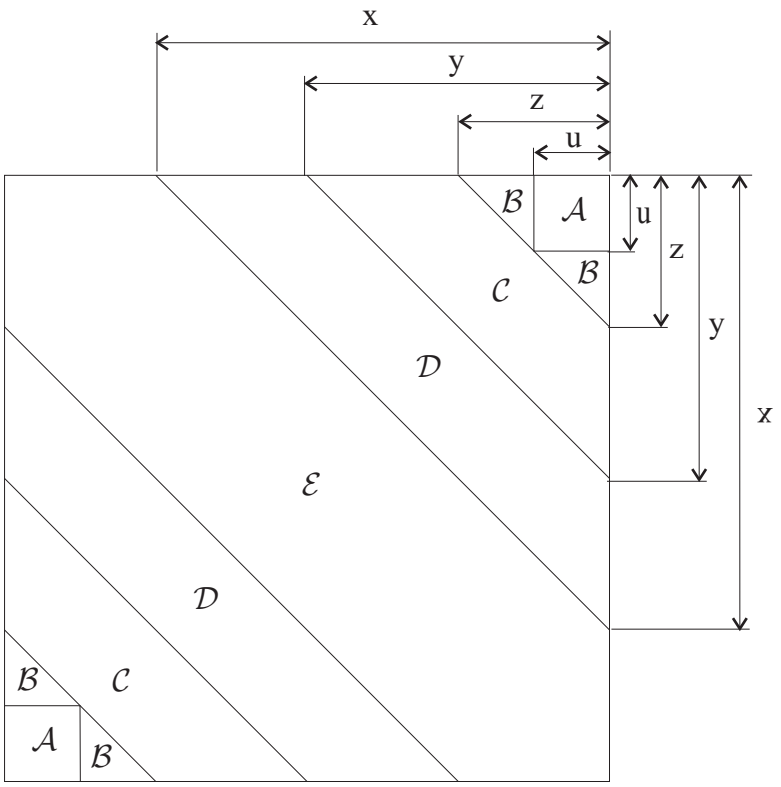
A [12] cikkben definiáltuk *LBA* algoritmust a mátrix transzponálási probléma megoldásra 2 dimenzióban.

Az LBA algoritmus

Előkészítő lépés: Legyen

$$x = n - \left\lfloor \frac{\lfloor \frac{n}{2} \rfloor}{2} \right\rfloor - 1, \quad y = \left\lceil \frac{n}{2} \right\rceil,$$

$$z = \left\lfloor \frac{\lfloor \frac{n}{2} \rfloor}{2} \right\rfloor + 1, \quad u = \left\lceil \frac{\left\lfloor \frac{\lfloor \frac{n}{2} \rfloor}{2} \right\rfloor + 1}{2} \right\rceil$$



2. ábra. Az $n \times n$-es hálózat felosztása az LBA-val

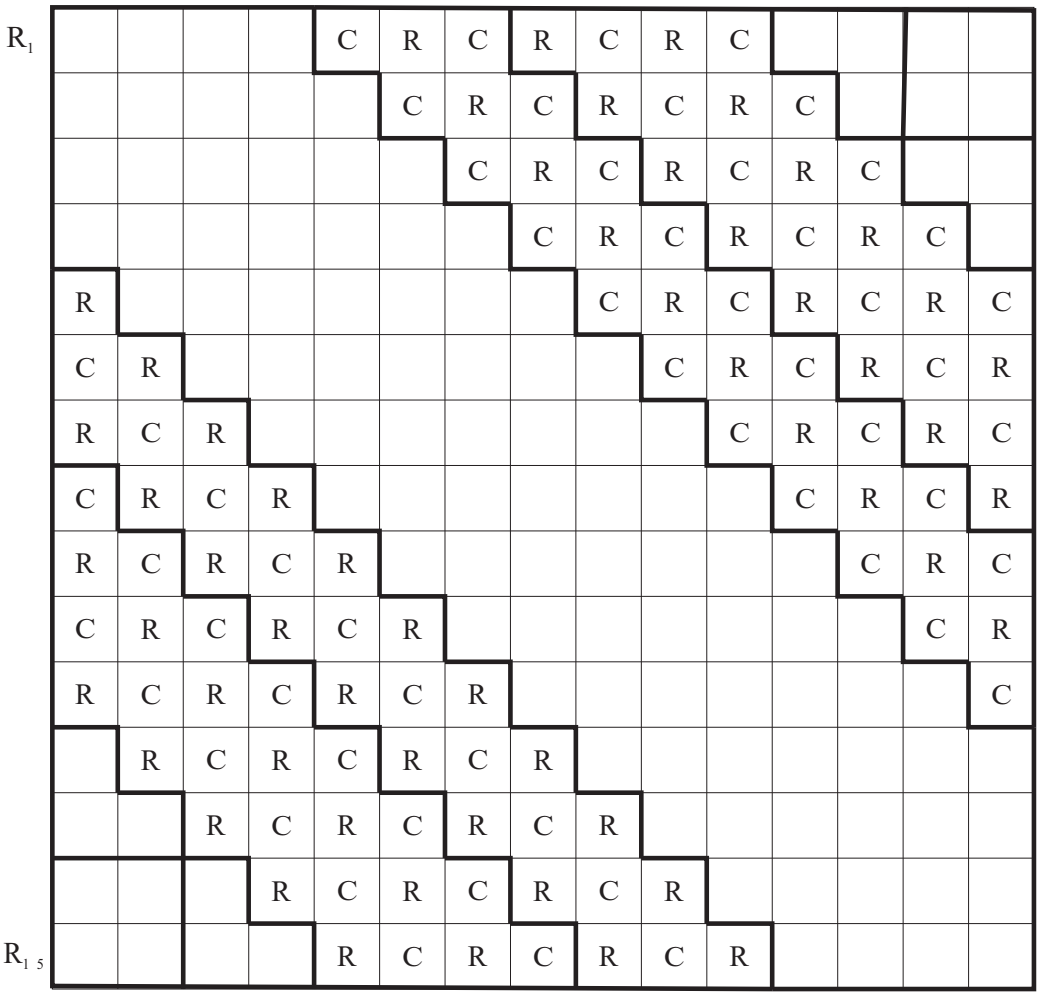
és osszuk fel a processzorokat öt diszjunkt részre. Legyenek ezek $\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D}, \mathcal{E}$ ahogy a 2. ábra mutatja.

1. lépés: Jelöljük meg a processzorokat a $\mathcal{C}$ és $\mathcal{D}$ területeken az R és C betűkkel, a 3. ábra szerint. Azok a processzorok, amelyek az R jelet kapták, először a sorbuszokat fogják használni, míg a C jelűek oszlopbuszokon indulnak. Vegyük észre, hogy ha egy processzor a C jelet kapta, akkor a párja R jelű és fordítva.

A következő 4 lépése az algoritmusnak (2-5. lépések) párhuzamosan hajtódik végre:

2. lépés: Továbbítsuk az $\mathcal{E}$ terület üzeneteit a kapcsolóvonalakon a mohó algoritmussal: a főátló alatt lévő elemek először a céloszlopukhoz sétálnak, ott elfordulnak, majd a célprocesszorukhoz mennek lépésről lépésre. A többi elem először a célsorhoz megy, balra fordul, majd eljut a célprocesszorhoz.

3. lépés: Ütemezzük a $\mathcal{C}$ és $\mathcal{D}$ területek üzeneteit a sor- és oszlopbuszokra a 1. lépésben meghatározott címkék szerint. Az R címkével ellátott processzorokra a "legtávolabbi legelőször" stratégiát alkalmazzuk, először a $\mathcal{D}$ terület elemeivel indítva. Vegyük észre, hogy a megfelelő oszlophoz való



3. ábra. A sor- és oszlopbuszok ütemezése egy 15×15-ös hálózaton

megérkezés után a $\mathcal{D}$ területről érkező üzenetek a kapcsolóvonalakon haladnak a céljuk felé. A $\mathcal{C}$ terület üzenetei ezután újból buszt fognak használni, méghozzá R jelű üzenet esetén oszlopbuszt. így ezen üzenetek továbbítása 2 buszműveletet igényel. A C-jelű üzenetek továbbítása hasonlóan történik.

4. lépés: Továbbítsuk a $\mathcal{B}$ terület üzeneteit a hálózat közepe felé a kapcsolóvonalakon, amíg az $(1, n-z+1), (z, n), (n-z+1, 1), (n, z)$ pozíciók üzenetei eljutnak a nekik megfelelő $(1, y+1), (n-y, n), (y+1, 1), (n, n-y)$ pozíciókba.

5. lépés: Ütemezzük az újonnan érkezett elemeket (amelyek a $\mathcal{B}$ területről származnak) a $\mathcal{C}$ területen a sor- és oszlopbuszokra, amikor megérkeznek. A felfelé (illetve lefelé) mozgó elemek sorbuszokra lesznek ütemezve, a balra (illetve jobbra) mozgó elemek pedig oszlopbuszokra. Amikor egy ilyen elem elküldésre kerül a buszon, mindig egy új elem kerül a pozíciójára, hogy a következő lépésben a buszt használhassa. Az első buszművelet után az elemek a második buszműveletre a processzorok bufferében várakoznak. A sorbanállás megrövidítése érdekében ezek az elemek folyamatosan mozoghatnak is a céljuk felé.

6. lépés: Amikor a fenti négy lépés befejeződött, akkor az algoritmus az $\mathcal{A}$ régió elemeit, és azokat az elemeket továbbítja a sor ás oszlopbuszok használatával, amelyek az 5. lépés végrehajtása után a pufferben várakoznak. Mivel ezek nem használnak közös buszt, ezért ezek továbbítása párhuzamosan megoldható.

4.2. Tétel. ([12]) *Az LBA algoritmus kevesebb mint* $\frac{n}{2}+9$ *lépésben megoldja a mátrix transzponálási problémát.*

A 5. táblázatban láthatjuk, hogy az adott elemek melyik lépésben érkeznek a rendeltetési helyükre egy 10x10-es mátrix esetében. A szám melletti kód bemutatja az átvitel módját az utolsó lépésben, amely lehet séta (W), sorbusz (R) és oszlopbusz (C).

0	2W	4W	5W	5W	5C	5R	5C	4R	4C
4W	0	2W	5W	5W	5W	3C	5R	4C	4R
4W	4W	0	2W	5W	5W	5W	3C	3R	4R
5W	4W	4W	0	2W	5W	4W	5W	5C	4R
5W	5W	4W	4W	0	2W	4W	4W	5W	5C
5R	5W	5W	4W	4W	0	4W	4W	4W	5W
5C	3R	5W	4W	4W	4W	0	4W	4W	4W
5R	5C	3R	5W	4W	4W	2W	0	4W	4W
4C	4R	3C	5R	5W	4W	4W	2W	0	4W
4R	4C	4C	4C	5R	5W	4W	4W	2W	0

5. táblázat. Az LBA algoritmus lépései 10x10-es mátrixon

n	10	100	500	1000	4000
LB	5	45	226	451	1803
MTB	14	59	259	509	2009
Ratio	2.8	1.3111	1.1460	1.1286	1.1142

6. táblázat. Az alsó korlátok értékei és az LBA algoritmus lépéseinek száma különböző méretű (nxn) mátrixokon.

5. Útvonaltervező rendszer tömegközlekedéshez

Manapság az utazók körében elterjedt a számítógépes útvonaltervezők használata. Az egyéni utasok számára webhelyek és GPS navigációs készülékek széles választéka áll rendelkezésre. Általánosságban elmondható, hogy az ilyen típusú rendszereknél a úthálózatot egy gráf modellezi. A gráf csúcsai az utak találkozási pontjai, míg az élek a pontokat összekötő szakaszok. Ha az útszakasz hosszát az élekhez rendeljük, akkor súlyozott gráfot kapunk. A jól ismert Dijkstra algoritmust használhatjuk a két pont közötti legrövidebb út kiszámításához. A Dijkstra algoritmus hatékonysága meglehetősen jó, ám a valós úthálózatokat leíró gráfok mérete nagyon nagy lehet, különösen nagyobb földrajzi terület, például egy kontinens esetében. Mivel az utasok általában szinte azonnali választ várnak a kereséseikre, még a polinomiális futási idővel rendelkező Dijkstra algoritmus sem elég gyors időnként. Az

elmúlt két évtizedben számos gyorsítási lehetőséget vizsgáltak a probléma megoldása céljából.
Az útvonalkereső szolgáltatások nemcsak egyéni utazók, hanem tömegközlekedést használó utasok számára is elérhetőek. Ebben az esetben az úthálózatot általában nem szükséges gráffal leírni. Ennek oka az, hogy az útvonal előre meghatározott, vagy az utazás gyakran nem egy hagyományos úthálózaton történik, hanem kötött pályán (például vasúton), levegőben, vagy akár vízen. A tömegközlekedési útvonalak gráfjai általában a jármű megállóit tartalmazzák, az élek pedig a megállók közötti utazási lehetőségeket reprezentálják. Az élek súlya jelentheti az utazási időt, de mivel az egyes felhasználók különböző szempontok alapján választhatják ki az útvonalat, más modelleket is figyelembe lehet venni. A tömegközlekedési hálózatok általában nagyobbak lehetnek, mint a közúti hálózatok, mivel az utazás időtől függ, és ennek kezelését a modellben is megkövetelik. Ezért ebben az esetben a keresések felgyorsításának lehetőségei különösen fontosak, és ezeket szintén széles körben vizsgálták. A közúti gráfokon elérhető gyorsítási lehetőségek közül néhány használható itt is, de azok nem, amelyek az úthálózatok sajátosságait használják.
A [8] cikkben egy útvonaltervező rendszer, illetve annak a keresési algoritmusa került bemutatásra. A rendszer két ország, nevezetesen Magyarország és Szerbia két régiójának teljes tömegközlekedési hálózatára került kifejlesztésre. A modell alapját képező adatbázisok távolsági vonatokat, autóbuszokat és a nagyobb városok teljes helyi közlekedését tartalmazták. Az adatbázisok a régiókban működő szolgáltatóktól kapott menetrendeken alapultak. A rendszer modellezte a gyalogos forgalmat is, nem túl távoli megállóhelyek között. Figyelembe vette az egyéni felhasználói preferenciákat, mint például a gyaloglási távolságokat, a szállítási módokat és a célfüggvény tulajdonságait. A hálózatot ábrázoló gráf nagyon nagy volt, de néhány gyorsítási módszer segítségével sikerült létrehozni egy hatékony keresési algoritmust, amely képes volt kezelni a felhasználói igényeket. Az algoritmus az EU által finanszírozott Magyarország-Szerbia Interreg-IPA CBC Program keretében megvalósuló projektben került kifejlesztésre egy komplex útvonaltervezési alkalmazás részeként.

A hálózatok szerkezete

Az irodalomból ismert mindhárom gráftípust használtuk a közlekedési hálózat modellezésére. A memóriában az úgynevezett Station gráfot [27] és az arra épített Time-Dependent gráfot [15] tároltuk tartósan. A Time-Dependent gráf éleinek száma meglehetősen nagy volt, mivel ebben az időszakban hét különböző menetrend változat volt használatban. Ez azt jelentette, hogy

a különböző napokra és időszakokra vonatkozó keresésekre eltérő menetrendek vonatkoztak. Például a vasárnapi menetrend különbözött a hétköznapoktól. A Station gráfnak és a Time-Dependent gráfnak nem volt közvetlen szerepe keresésben, csak az előfeldolgozásnál. A tényleges keresés során a Dijkstra algoritmust egy Time-Expanded gráfon [26] valósítottuk meg. A Time-Expanded gráf nem volt állandóan a memóriában, mivel az mindig dinamikusan változott a kérések során. A 7. táblázat a különböző típusú gráfok méretét mutatja.

Gráf típus	Csúcsok száma	Élek száma
Station	12014	416163
Time Dependent	12014	6786662
Time Expanded	≈ 562000	≈ 4616000

7. táblázat. A gráfok mérete

A keresési algoritmus gyorsítása

A keresés gyorsítása érdekében elvégeztünk egy előfeldolgozási lépést, mielőtt elkészítettük a megfelelő Time-Expanded gráfot és végrehajtottuk rajta a Dijkstra algoritmust. A cél az volt, hogy meghatározzuk azokat a csomópontokat, amelyek tartalmazzák a felhasználói paramétereknek és a célfüggvénynek megfelelő legrövidebb utat. A módszer hasonló a TRANSIT algoritmusokhoz, amelyeket a [2, 7, 25] cikkekben ismertettek. Végül a Time-Expanded gráf sokkal kevesebb csúcspontot tartalmazott, és ez a keresés jelentős felgyorsulásához vezetett.
Az előfeldolgozáshoz a Station gráfot használtuk. Célunk az volt, hogy meghatározzuk az összes csúcsot, amelyek a forrás és a rendeltetési hely közötti maximálisan k hosszúságú úton fekszenek. Ebben az esetben a k paraméter azt jelentette, hogy hány átszállás lehetséges az utazás során. Itt a mélységi keresés algoritmus módosított változatát használtuk, amely nem vette figyelembe a forrástól a kívánt hosszúságnál távolabbi csúcsokat. Mivel bizonyos esetekben az eljárás végrehajtása hosszabb időt vett igénybe, a k paraméter értéke és a futási idő is korlátozott volt a keresésben. A tapasztalatok azt mutatták, hogy ezek a heurisztikák a gyakorlatban is jól működtek, vagyis a kapott csúcspontokra épített Time-Expanded gráf szinte minden esetben tartalmazta a teljes gráfban megtalálható legrövidebb utat.

A célfüggvény

Olyan célfüggvényt használtunk, amely a különböző célok súlyozott összegét tartalmazta. A súlyokat a felhasználók által megadott információk alapján, előre beállított minták felhasználásával határoztuk meg.
Az utazási él súlya a következő volt: $t_r w_r + f_r$, ahol t_r az utazás ideje, w_r az utazások tényleges súlya, és f_r az utazási költségek additív tényezője.
A várakozási él súlya a következő: $t_i w_i$, ahol t_i a várakozási idő, és w_i a várakozás tényleges súlya.
A gyalogos él súlya a következő: $t_a w_a + f_a$, ahol t_a a gyaloglás ideje, w_a a gyaloglás tényleges súlya, és f_a a kiegészítő tényező a gyalogosok számára.
A keresés során a felhasználók számára három lehetőség állt rendelkezésre a cél tekintetében. Ezek a leggyorsabb út, a minimális átszállással való utazás és a legkevesebb gyaloglás. Az egyes célok súlyainak paramétereit a 8. táblázat tartalmazza.

Opciók	w_r	f_r	w_i	w_a	f_a
Leggyorsabb	1	30	0.8	1.5	30
Legkevesebb átszállás	1	1000	0.8	1.5	10
Legkevesebb gyaloglás	1	30	0.8	30	1000

8. táblázat. A súlyok paraméterei

Eredmények

Megvizsgáltuk hogy a vizsgált példák mekkora részében tartalmazza a csökkentett méretű gráf a legrövidebb út összes csúcsát. Ez biztosította ugyanis, hogy a gyorsított keresés optimális megoldást adjon. Az előfeldolgozás heurisztikában két paramétert módosítottunk. Az egyik a korábban már említett k paraméter, a másik a keresési idő. A k lehetséges értékei $s+1$ és $s+2$ voltak, ahol s a Station gráfban az adott két pont közötti legrövidebb út. Kétféle tesztelést végeztünk. Mindkettő 1000 kérdést tartalmazott, amelyek egyike a helyi közlekedésre (T1), a másik pedig a távolsági közlekedésre (T2) vonatkozott. Az alábbi táblázatban összefoglaljuk, hogy az előfeldolgozás milyen hatékonyságú volt a két esetben, különböző k értékek esetén. A táblázatban láthatjuk, hogy a csökkentett méretű gráf hányszor adott olyan megoldást, amely eltért az optimálistól, amit a teljes Time-Expanded gráf segítségével számoltunk ki.

Input típus	$k = s + 1$	$k = s + 2$
T1	13	0
T2	54	11

9. táblázat. Az előfeldolgozás optimalitási statisztikája ([8])

A 10. és 11. táblázatok az átlagos és a maximális keresési időket tartalmazzák ezredmásodpercben az említett 1000 kérdésnél a csökkentett és a teljes Time Expanded gráfra. A jelölések a táblázatban a következők:
RB: csökkentett gráf felépítésének ideje,
RS: keresési idő a csökkentett gráfon,
RC (=RB+RS):összes keresési idő a csökkentett gráfon,
FB: teljes gráf felépítésének ideje,
FS: keresési idő a teljes gráfon,
FC (=FB+FS): összes keresési idő a teljes gráfon.

	RB	RS	RC	FB	FS	FC
$T1, k = s + 1$	308	18	327	5327	166	5493
$T1, k = s + 2$	1394	406	1801			
$T2, k = s + 1$	256	55	312	5427	1137	6564
$T2, k = s + 2$	1186	316	1502			

10. táblázat. Átlagos futási idők ezredmásodpercben (ms)([8])

	RB	RS	RC	FB	FS	FC
$T1, k = s + 1$	1478	446	1529	5744	4130	9502
$T1, k = s + 2$	4959	2458	5453			
$T2, k = s + 1$	2491	788	3279	6290	10216	15650
$T2, k = s + 2$	3119	3938	6469			

11. táblázat. Maximális futási idők ezredmásodpercben (ms)([8])

6. Összefoglalás

A doktori disszertáció 4 különböző témában a PhD fokozat megszerzése után elért eredményekből mutatott be néhányat, amelyek nagyobbik részben szerzőtársakkal közös publikációkon alapulnak. A disszertációban és a tézisekben

azok az eredmények szerepelnek, amelyeknél a hozzájárulásomat jelentősnek tekintem és ezzel a szerzőtársaim is egyetértettek.

7. Köszönetnyilvánítás

Köszönettel tartozom szerzőtársaimnak Balogh Jánosnak, Dósa Györgynek, Galambos Gábornak, Leah Epsteinnek, Łukasz Jeżnek, Asaf Levinnek, Marcus Oswaldnak, Gerhard Reineltnek, Jarret Schwartznak, Jiří Sgallnak és Zhiyi Tannak. A hozzájárulásuk nélkül az eredményeket tartalmazó cikkek nem készülhettek volna el.

Hivatkozások

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

[2] L. Antsfeld, T. Walsh, Finding Optimal Paths in Multi-modal Public Transportation Networks using Hub Nodes and TRANSIT algorithm, In.: *L. Frommberger,K. Schill, B. Scholz-Reiter (eds.), Proceedings of the 3rd Workshop on Artifical Intelligence and Logistics*, Montpellier, France, 7–13, 2012.

[3] J. Balogh, J. Békési, G. Galambos, Gy. Dósa, Z. Tan, Lower bound for 3-batched bin packing, *Discrete Optimization*, 21, 14–24, 2016.

[4] J. Balogh, J. Békési, Gy. Dósa, L. Epstein, A. Levin, A New and Improved Algorithm for Online Bin Packing, In: *Yossi, Azar; Hannah, Bast; Grzegorz, Herman (szerk.) 26th Annual European Symposium on Algorithms (ESA 2018)*, Schloss Dagstuhl, Németország, 5:1–5:14, 2018.

[5] J. Balogh, J. Békési, Gy. Dósa, J. Sgall, R. van Stee, The optimal absolute ratio for online bin packing, *Journal of Computer and System Sciences* 102, 1–17, 2019.

[6] J. Balogh, J. Békési, Gy. Dósa, L. Epstein, A. Levin, A New Lower Bound for Classic Online Bin Packing, *Lecture Notes in Computer Science* 11926, 18–28, 2020.

[7] H. Bast, S. Funke, D. Matijevic. Transit ultrafast shortest-path queries with linear-time preprocessing. In *9th DIMACS Implementation Challenge*, 2006.

[8] J. Békési, Regional Multicriteria and Multimodal Route Planning System for Public Transportation: A Case Study, *Acta Cybernetica*, 23(3), 773–782, 2018

[9] J. Békési, Gy. Dósa, L. Epstein, Bounds for online bin packing with cardinality constraints, *Information and Computation*, 249, 190–204, 2016.

[10] J. Békési, Gy. Dósa, G. Galambos, A first fit type algorithm for the coupled task scheduling problem with unit execution time and two exact delays. *European Journal of Operational Research*, 297(3), 844–852, 2022.

[11] J. Békési, G. Galambos, M. Oswald, G. Reinelt, Improved analysis of an algorithm for the coupled task problem with UET jobs, *Operations Research Letters* 37(2), 93–96, 2009.

[12] J. Békési, G. Galambos, Matrix transpose on meshes with buses *Journal of Parallel and Distributed Computing*, 96, 194–201, 2016.

[13] J. Békési, G. Galambos, Tight bounds for NF-based bounded-space online bin packing algorithms, *Journal of Combinatorial Optimization* 35(2), 350–364, 2017.

[14] P. Berman, M. Charikar, M. Karpinski, On-line load balancing for related machines, *Journal of Algorithms* 35, 108–121, 2000.

[15] G. Brodal, R. Jacob, Time-dependent Networks as Models to Achieve Fast Exact Time-table Queries, *Electronic Notes in Theoretical Computer Science*, 92, 3–15, 2004.

[16] K.S. Ding, C.T. Ho, J.J. Tsay, Matrix Transpose on Meshes with Wormhole and XY Routing, *Proc. 6th Symposium on Parallel and Distributed Processing*, IEEE, 656–663, 1994.

[17] T. Ebenlendr, J. Sgall, A lower bound on deterministic online algorithms for scheduling on related machines without preemption. In: *Proc. 9th International Workshop in Approximation and Online Algorithms (WAOA 2011)*, Springer, Lecture Notes in Comput. Sci., vol 7164, 102–108, 2012.

[18] L. Epstein, J. Sgall, A lower bound for on-line scheduling on uniformly related machines. *Operations Research Letters* 26, 17–22, 2000.

[19] H. Fujiwara and K. M. Kobayashi, Improved lower bounds for the online bin packing problem with cardinality constraints, *Journal of Combinatorial Optimization*, 29(1), 67–87, 2015.

[20] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G.Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Discrete Optimization II, Proceedings of the Advanced Research Institute on Discrete Optimization and Systems Applications of the Systems Science Panel of NATO and of the Discrete Optimization Symposium co-sponsored by IBM Canada and SIAM Banff, Aha. and Vancouver*, pages 287–326. Elsevier, 1979.

[21] G. Gutin, T. Jensen, A. Yeo, Batched bin packing, *Discrete Optimization*, 2(1), 71–82, 2005.

[22] Ł. Jeż, J. Schwartz, J. Sgall, J. Békési, Lower bounds for online makespan minimization on a small number of related machines, *Journal of Scheduling* 16(5), 539–547, 2013.

[23] D.S. Johnson, Near-optimal bin-packing algorithms, *Doctoral Thesis, MIT, Cambridge*, 1973.

[24] M. Kaufmann, U. Meyer, J.F. Sibeyn, Matrix Transpose on Meshes: Theory and Practice, *Computers and Artificial Intelligence* 16, 107–140, 1997.

[25] J. Koszelew, Two Methods of Quasi-Optimal Routes Generation in Public Transportation Network, *International Conference on Computer Information Systems and Industrial Management Applications*,231–236, 2008.

[26] F. Schulz, D. Wagner, K. Weihe, Dijkstra's Algorithm On-Line: An Empirical Case Study from Public Railroad Transport, *ACM Journal of Experimental Algorithmics*, 5(12), 1–23, 2000.

[27] F. Schulz, D. Wagner, C. Zaroliagis, Using multi-level graphs for timetable information in railway systems, *Proceedings of the 4th Workshop on Algorithm Engineering and Experiments (ALENEX'02)*, Lecture Notes in Computer Science, Vol. 2409, Springer, 43–59, 2002.

[28] A. C. C. Yao, New algorithms for bin packing, *Journal of the ACM*, 27, 207–227, 1980.

[29] M. Zhang, X. Han, Y. Lan, H-F. Ting, Online bin packing problem with buffer and bounded size revisited, *Journal of Combinatorial Optimization*, 33(2), 530–542, 2017.

[30] F. Zheng, L. Huo, E. Zhang, NF-based algorithms for online bin packing with buffer and bounded item size. *Journal of Combinatorial Optimization*, 30(2), 360–369, 2015.