Felsőbb matematika villamosmérnököknek – Kombinatorikus optimalizálás

Közelítő algoritmusok, ütemezési problémák

2020. május 5.

Bevezetés

A mai órán olyan feladatokról lesz szó, amelyeknek az optimális megoldására nincs hatékony algoritmus. (A hatékony itt nem feltétlenül polinomidejűt jelent.) Sok esetben nincs szükség azonban optimumra, elég, ha az algoritmus egy optimumhoz kellően közeli megoldást talál. Ezt a lehetőséget kihasználva esetleg tudunk olyan algortimust tervezni, ami igen gyorsan fut, mégis jól használható eredményt ad.

Bevezetés

A mai órán olyan feladatokról lesz szó, amelyeknek az optimális megoldására nincs hatékony algoritmus. (A hatékony itt nem feltétlenül polinomidejűt jelent.) Sok esetben nincs szükség azonban optimumra, elég, ha az algoritmus egy optimumhoz kellően közeli megoldást talál. Ezt a lehetőséget kihasználva esetleg tudunk olyan algortimust tervezni, ami igen gyorsan fut, mégis jól használható eredményt ad.
Feltesszük, hogy az algoritmusunk célja egy $f(x)$ célfüggvény minimalizálása egy, az I inputtól függő X_I megoldáshalmazon. Például, ha az I input egy négyzetes mátrix, a megoldáshalmaz a bástyaelhelyezések halmaza, és a célfüggvény értéke a bástyaelhelyezéshez tartozó mátrixelemek összegének (-1)-szerese, akkor az optimális megoldás egy maximális súlyú párosítás. Vagy ha az I input egy G gráf, $u, v \in V(G)$ és egy ℓ hosszfüggvény G élein, az X_I megoldáshalmazt pedig a G-beli uv-utak alkotják, akkor $f(x)$ lehet az x út hossza. Itt az optimális megoldás egy legrövidebb uv-út lesz.

Def: Tfh a Π problémában az f (nemnegatív) függvényt kell minimalizálni az I inputhoz tartozó X_I megoldáshalmazon. Tfh az A algoritmus tetszőleges inputra egy $A(I) \in X_I$ megoldást talál. Ekkor az A algoritmus additív hibája C, ha $f(A(I)) \leq min\{f(x) : x \in X_I\} + C$ teljesül minden lehetséges I inputra. Ekkor C az A közelítő algoritmus abszolút hibája.

Def: Tfh a Π problémában az f (nemnegatív) függvényt kell minimalizálni az I inputhoz tartozó X_I megoldáshalmazon. Tfh az A algoritmus tetszőleges inputra egy $A(I) \in X_I$ megoldást talál. Ekkor az A algoritmus additív hibája C, ha $f(A(I)) \leq min\{f(x) : x \in X_I\} + C$ teljesül minden lehetséges I inputra. Ekkor C az A közelítő algoritmus abszolút hibája.
Az A algoritmus multiplikatív hibája α, ha minden I inputra $f(A(I)) \leq \alpha \cdot min\{f(x) : x \in X_I\}$ teljesül. Ekkor A egy α-közelítő algoritmus Π-re és A relatív hibája α.

Def: Tfh a Π problémában az f (nemnegatív) függvényt kell minimalizálni az I inputhoz tartozó X_I megoldáshalmazon. Tfh az A algoritmus tetszőleges inputra egy $A(I) \in X_I$ megoldást talál. Ekkor az A algoritmus additív hibája C, ha $f(A(I)) \leq min\{f(x) : x \in X_I\} + C$ teljesül minden lehetséges I inputra. Ekkor C az A közelítő algoritmus abszolút hibája.
Az A algoritmus multiplikatív hibája α, ha minden I inputra $f(A(I)) \leq \alpha \cdot min\{f(x) : x \in X_I\}$ teljesül. Ekkor A egy α-közelítő algoritmus Π-re és A relatív hibája α.

Megj: Ha a Π problémában maximalizálni (és nem minimalizálni) kell az f célfüggvényt, akkor $f(A(I)) \geq \max\{f(x) : x \in X_I\} - C$ ill.
$f(A(I)) \geq \alpha \cdot max\{f(x) : x \in X_I\}$ segítségével definiáljuk az abszolút ill. relatív hibát.

Példák: (1) Ismert olyan algoritmus, ami tetsz. G síkgráfot legfeljebb 5 színnel kiszínez, de ha G-ben nincs háromszög, akkor legfeljebb 4-gyel, ha pedig G páros, akkor $\leq$ 2-vel. Mivel minden síkgráfhoz elég 4 szín, ezért az algoritmus abszolút hibája 2.

Példák: (1) Ismert olyan algoritmus, ami tetsz. G síkgráfot legfeljebb 5 színnel kiszínez, de ha G-ben nincs háromszög, akkor legfeljebb 4-gyel, ha pedig G páros, akkor ≤ 2-vel. Mivel minden síkgráfhoz elég 4 szín, ezért az algoritmus abszolút hibája 2.
(2) Vizing tétele szerint ha G egyszerű gráf, akkor élkromatikus száma legfeljebb eggyel több a maximális fokszámánál: $\chi'(G) \leq \Delta(G) + 1$. Ismert olyan algoritmus, ami bármely ilyen G-t kiszínez kiszínez legfeljebb $\Delta(G) + 1$ színnel. A triviális $\chi'(G) \geq \Delta(G)$ egyenlőtlenség miatt ennek az algoritmusnak az abszolút hibája 1.

Példák: (1) Ismert olyan algoritmus, ami tetsz. G síkgráfot legfeljebb 5 színnel kiszínez, de ha G-ben nincs háromszög, akkor legfeljebb 4-gyel, ha pedig G páros, akkor ≤ 2-vel. Mivel minden síkgráfhoz elég 4 szín, ezért az algoritmus abszolút hibája 2.
(2) Vizing tétele szerint ha G egyszerű gráf, akkor élkromatikus száma legfeljebb eggyel több a maximális fokszámánál: $\chi'(G) \leq \Delta(G) + 1$. Ismert olyan algoritmus, ami bármely ilyen G-t kiszínez kiszínez legfeljebb $\Delta(G) + 1$ színnel. A triviális $\chi'(G) \geq \Delta(G)$ egyenlőtlenség miatt ennek az algoritmusnak az abszolút hibája 1.

Állítás: Gráf minimális méretű lefogó ponthalmazához van polinomidejű 2-közelítés.

Példák: (1) Ismert olyan algoritmus, ami tetsz. G síkgráfot legfeljebb 5 színnel kiszínez, de ha G-ben nincs háromszög, akkor legfeljebb 4-gyel, ha pedig G páros, akkor ≤ 2-vel. Mivel minden síkgráfhoz elég 4 szín, ezért az algoritmus abszolút hibája 2.
(2) Vizing tétele szerint ha G egyszerű gráf, akkor élkromatikus száma legfeljebb eggyel több a maximális fokszámánál: $\chi'(G) \leq \Delta(G) + 1$. Ismert olyan algoritmus, ami bármely ilyen G-t kiszínez kiszínez legfeljebb $\Delta(G) + 1$ színnel. A triviális $\chi'(G) \geq \Delta(G)$ egyenlőtlenség miatt ennek az algoritmusnak az abszolút hibája 1.

Állítás: Gráf minimális méretű lefogó ponthalmazához van polinomidejű 2-közelítés.

Biz: Válasszunk G-ből pként diszjunkt éleket amíg tudunk. Ha k él választása után nem tudunk többet választani, akkor mindezen k él $2k$ végpontja lefogó ponthalmaz. Másrészt a megtalált k él lefogásához szükség van legalább k csúcsra, így $f(A(G)) = 2k \geq 2 \cdot \tau(G) = 2 \cdot \min\{f(x) : x \in X_G\}$. □

Egy nemapproximálható feladat 14:48

Állítás: Ha $P \neq NP$, akkor nincs olyan polinomidejű A algoritmus, ami tetszőleges G gráfra C abszolút hibával talál közelítést G leghosszabb köréhez.

Egy nemapproximálható feladat 14:48

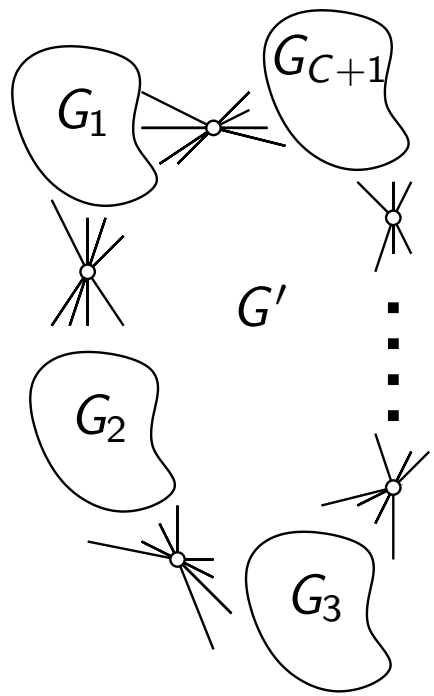
Állítás: Ha $P \neq NP$, akkor nincs olyan polinomidejű A algoritmus, ami tetszőleges G gráfra C abszolút hibával talál közelítést G leghosszabb köréhez.

Biz: Megmutatjuk, hogy ha van polinomidejű, C additív hibájú közelítés, akkor ebből kaphatunk olyan polinomidejű algoritmust, ami tetszőleges G inputgráfról polinomidőben eldönti, van-e Hamilton-útja. Márpedig ha $P \neq NP$, akkor nincs ilyen algoritmus.

Egy nemapproximálható feladat 14:48

Állítás: Ha $P \neq NP$, akkor nincs olyan polinomidejű A algoritmus, ami tetszőleges G gráfra C abszolút hibával talál közelítést G leghosszabb köréhez.

Biz: Megmutatjuk, hogy ha van polinomidejű, C additív hibájú közelítés, akkor ebből kaphatunk olyan polinomidejű algoritmust, ami tetszőleges G inputgráfról polinomidőben eldönti, van-e Hamilton-útja. Márpedig ha $P \neq NP$, akkor nincs ilyen algoritmus. „Varrjuk össze" G $C+1$ diszjunkt példányát az ábra szerint egy G' gráffá. Ha G-nek van Hamilton-útja, akkor az így kapott gráfnak lesz Hamilton-köre. Ha nincs, akkor G' bármely köre G minden példányából kihagy legalább egy csúcsot, ezért G' leghosszabb köre legalább $C+1$ csúccsal kevesebbet tarttalmaz, mint G csúcsszáma. Egy C additív hibával közelítő algoritmus outputjából azonnal látszik, hogy van-e G-nek Hamilton-köre. □



A halmazfedési probléma közelítése 20:45

Halmazfedési probléma **Input:** Az U n-elemű alaphalmaz az $S_1, S_2, \ldots, S_k \subseteq U$ részhalmazai, és a $c(S_i)$ nemnegatív költségek.
Cél: U minimális összköltségű fedése S_i-kkel.

A halmazfedési probléma közelítése 20:45

Halmazfedési probléma **Input:** Az U n-elemű alaphalmaz az $S_1, S_2, \ldots, S_k \subseteq U$ részhalmazai, és a $c(S_i)$ nemnegatív költségek. **Cél:** U minimális összköltségű fedése S_i-kkel.

Mohó algoritmus Egymás után választjuk a fedésbe az $S^1, S^2, \ldots$ részhalmazokat. Mindig azt a halmazt választjuk, amelyik fajlagosan a legolcsóbban fedi az eddig le nem fedett pontokat, azaz azt az S^i-t, amire $c(S^i)/m$ minimális, ahol m az S^i által lefedett, de az eddig választott halmazok által le nem fedett U-beli elemek száma: $m = |S \setminus (S^1 \cup S^2 \cup \ldots \cup S^{i-1})|$.

A halmazfedési probléma közelítése 20:45

Halmazfedési probléma **Input:** Az U n-elemű alaphalmaz az $S_1, S_2, \ldots, S_k \subseteq U$ részhalmazai, és a $c(S_i)$ nemnegatív költségek. **Cél:** U minimális összköltségű fedése S_i-kkel.

Mohó algoritmus Egymás után választjuk a fedésbe az $S^1, S^2, \ldots$ részhalmazokat. Mindig azt a halmazt választjuk, amelyik fajlagosan a legolcsóbban fedi az eddig le nem fedett pontokat, azaz azt az S^i-t, amire $c(S^i)/m$ minimális, ahol m az S^i által lefedett, de az eddig választott halmazok által le nem fedett U-beli elemek száma: $m = |S \setminus (S^1 \cup S^2 \cup \ldots \cup S^{i-1})|$.

Példa

$c(1234) = 10$	$c(124) = 6$	$c(134) = 4$	$c(12) = 3$	$c(1) = 1$
$2, 5$	2	$\frac{4}{3}$	$\frac{3}{2}$	$\boxed{1}$
$\frac{10}{3}$	3	$\boxed{2}$	3	$-$
10	6	$-$	$\boxed{3}$	

A halmazfedési probléma közelítése 20:45

Halmazfedési probléma **Input:** Az U n-elemű alaphalmaz az $S_1, S_2, \ldots, S_k \subseteq U$ részhalmazai, és a $c(S_i)$ nemnegatív költségek. **Cél:** U minimális összköltségű fedése S_i-kkel.

Mohó algoritmus Egymás után választjuk a fedésbe az $S^1, S^2, \ldots$ részhalmazokat. Mindig azt a halmazt választjuk, amelyik fajlagosan a legolcsóbban fedi az eddig le nem fedett pontokat, azaz azt az S^i-t, amire $c(S^i)/m$ minimális, ahol m az S^i által lefedett, de az eddig választott halmazok által le nem fedett U-beli elemek száma: $m = |S \setminus (S^1 \cup S^2 \cup \ldots \cup S^{i-1})|$.

A halmazfedési probléma közelítése 20:45

Halmazfedési probléma **Input:** Az U n-elemű alaphalmaz az $S_1, S_2, \ldots, S_k \subseteq U$ részhalmazai, és a $c(S_i)$ nemnegatív költségek. **Cél:** U minimális összköltségű fedése S_i-kkel.

Mohó algoritmus Egymás után választjuk a fedésbe az $S^1, S^2, \ldots$ részhalmazokat. Mindig azt a halmazt választjuk, amelyik fajlagosan a legolcsóbban fedi az eddig le nem fedett pontokat, azaz azt az S^i-t, amire $c(S^i)/m$ minimális, ahol m az S^i által lefedett, de az eddig választott halmazok által le nem fedett U-beli elemek száma: $m = |S \setminus (S^1 \cup S^2 \cup \ldots \cup S^{i-1})|$.

A Mohó algoritmus analízise Legyen OPT az optimális fedés összköltsége. Ha $a_1, a_2, \ldots, a_n$ sorrendben fedünk, akkor a_i fedésének fajl. költsége $\leq OPT/(n-i+1)$. Így az output összktge $\leq OPT \cdot (1/n + 1/(n-1) + \ldots + 1/1) \leq OPT \cdot (1 + lnn)$.

A halmazfedési probléma közelítése 20:45

Halmazfedési probléma **Input:** Az U n-elemű alaphalmaz az $S_1, S_2, \ldots, S_k \subseteq U$ részhalmazai, és a $c(S_i)$ nemnegatív költségek. **Cél:** U minimális összköltségű fedése S_i-kkel.

Mohó algoritmus Egymás után választjuk a fedésbe az $S^1, S^2, \ldots$ részhalmazokat. Mindig azt a halmazt választjuk, amelyik fajlagosan a legolcsóbban fedi az eddig le nem fedett pontokat, azaz azt az S^i-t, amire $c(S^i)/m$ minimális, ahol m az S^i által lefedett, de az eddig választott halmazok által le nem fedett U-beli elemek száma: $m = |S \setminus (S^1 \cup S^2 \cup \ldots \cup S^{i-1})|$.

A Mohó algoritmus analízise Legyen OPT az optimális fedés összköltsége. Ha $a_1, a_2, \ldots, a_n$ sorrendben fedünk, akkor a_i fedésének fajl. költsége $\leq OPT/(n-i+1)$. Így az output összktge $\leq OPT \cdot (1/n + 1/(n-1) + \ldots + 1/1) \leq OPT \cdot (1 + lnn)$.

Megj.: Nem létezik $konst \cdot lnn$-approximációnál lényegesen jobban közelítő polinomidejű algoritmus.

Input: $J_1, J_2, \ldots$ munkák p_i megmunkálási idők, gépek m száma.
Feladat: A munkák gépekre ütemezése. Egy S ütemezésre $S_t(i)$ és $S_m(i)$ adja meg, hogy J_i-t mikor és melyik gépen kell elkezdeni. $C_i^S = S_t(i) + p(i)$ a J_i befejezési időpontja, $C_{max}^S = \max_i C_i^S$ pedig az S ütemezés átfutási ideje.
Megkötések: A gépek egyformák (ebben a tárgyalásban). Egy gépen egyszerre egy munka végezhető. Minden munkát megszakítás nélkül kell elvégezni.
Lehetséges feltételek J_i-kre: r_i (rendelkezésre állási idő), w_i (súly), d_i (határidő), $\prec$ (precedencia)
Optimalizálandó mennyiségek: C_{max}^S, $(\sum_i C_i^S)/n$
Tömör jelölés: $\alpha|\beta|\gamma$, ahol
$\alpha \in \{1, Pm, P\}$ (1 gép, m gép, sok párhuzamos gép)
$\beta \in \{p = 1, r_j, \prec\}$ (feltételek megadása)
$\gamma \in \{C_{max}, \sum_i C_i\}$ (célfv megadása)
PI: (1) $1||C_{max}$: 1 gépen kell mihamarabb végezni. $OPT = \sum_i p_i$.
(2) $1|\prec|C_{max}$. Nem mindegy a sorrend, de itt is $OPT = \sum_i p_i$.

Az SPT sorrend optimalitása

Tétel: Az $1||\sum_i C_i$ feladatra optimális megoldás a munkák p_i szerint növekvő sorrendben történő ütemezése.

Az SPT sorrend optimalitása 40:21

Tétel: Az $1||\sum_i C_i$ feladatra optimális megoldás a munkák p_i szerint növekvő sorrendben történő ütemezése.

Biz: Ha $J_1, J_2, \ldots, J_n$ sorrendben ütemezünk, akkor $\sum_{i=1}^{n} C_i^S = \sum_{i=1}^{n} \sum_{j=1}^{i} p_j = \sum_{1 \le j \le i \le n} p_j = \sum_{j=1}^{n} \sum_{i=j}^{n} p_j = \sum_{j=1}^{n} (n+1-j)p_j = np_1 + (n-1)p_2 + \ldots + 2p_{n-1} + 1p_n$, és ez akkor minimális, ha $p_1 \le p_2 \le \ldots \le p_n$. □

Az SPT sorrend optimalitása 40:21

Tétel: Az $1||\sum_i C_i$ feladatra optimális megoldás a munkák p_i szerint növekvő sorrendben történő ütemezése.

Biz: Ha $J_1, J_2, \ldots, J_n$ sorrendben ütemezünk, akkor $\sum_{i=1}^{n} C_i^S = \sum_{i=1}^{n} \sum_{j=1}^{i} p_j = \sum_{1 \le j \le i \le n} p_j = \sum_{j=1}^{n} \sum_{i=j}^{n} p_j = \sum_{j=1}^{n} (n+1-j)p_j = np_1 + (n-1)p_2 + \ldots + 2p_{n-1} + 1p_n$, és ez akkor minimális, ha $p_1 \le p_2 \le \ldots \le p_n$. □

Tétel: Az $1||\sum_i w_i C_i$ feladatra optimális megoldás a munkák p_i/w_i szerint növekvő sorrendben történő ütemezése.

Tétel: Az $1||\sum_i C_i$ feladatra optimális megoldás a munkák p_i szerint növekvő sorrendben történő ütemezése.

Biz: Ha $J_1, J_2, \ldots, J_n$ sorrendben ütemezünk, akkor $\sum_{i=1}^{n} C_i^S = \sum_{i=1}^{n} \sum_{j=1}^{i} p_j = \sum_{1 \le j \le i \le n} p_j = \sum_{j=1}^{n} \sum_{i=j}^{n} p_j = \sum_{j=1}^{n} (n+1-j)p_j = np_1 + (n-1)p_2 + \ldots + 2p_{n-1} + 1p_n$, és ez akkor minimális, ha $p_1 \le p_2 \le \ldots \le p_n$. □

Tétel: Az $1||\sum_i w_i C_i$ feladatra optimális megoldás a munkák p_i/w_i szerint növekvő sorrendben történő ütemezése.

Biz: Ha $J_1, J_2, \ldots, J_n$ sorrendben ütemezünk, akkor $\sum_{i=1}^{n} w_i C_i^S = \sum_{i=1}^{n} w_i \sum_{j=1}^{i} p_j = \sum_{1 \le j \le i \le n} w_i p_j = \sum_{j=1}^{n} \sum_{i=j}^{n} w_i p_j$, és ha $p_i/w_i > p_{i+1}/w_{i+1}$ (azaz $p_i w_{i+1} > p_{i+1} w_i$), akkor J_i és J_{i+1} cseréjétől a célfüggvény értéke csökken. Egymást követő munkák cseréjével a célfüggvény értékét csökkentve előbb-utóbb a tételben szereplő ütemezés adódik. □

A $P2||C_{max}$ probléma

Állítás: A $P2||C_{max}$ probléma reménytelen.

Állítás: A $P2||C_{\max}$ probléma reménytelen.

Biz: Tudjuk, hogy az alábbi PARTÍCIÓ probléma NP-teljes. $a_1, a_2, \ldots, a_n$ input esetén kell eldönteni, hogy a 0 előállítható-e $0 = \pm a_1 \pm a_2 \pm \ldots \pm a_n$ formában. A PARTÍCIÓ fenti inputja mellett n job és $p_i = a_i$ process time. Ekkor $C_{\max} = \frac{1}{2}\sum_{i=1}^{n} p_i$ pontosan akkor teljesük, ha a két gép folyamatos munka mellett egyszerre tud végezni. Ez pedig pontosan akkor van így, ha a PARTÍCIÓ problémában a 0 előállíható a kívánt alakban. Ha tehát volna optimálisan ütemező polinomidejű algoritmus, akkor polinomidőben tudnánk megoldani egy NP-teljes problémát. □

LS algoritmus Az m gépen, $J_1, J_2, \ldots, J_n$ sorrendben végezzük el a munkákat, minden munkát az elsőnek felszabaduló gépen.

LS algoritmus Az m gépen, $J_1, J_2, \ldots, J_n$ sorrendben végezzük el a munkákat, minden munkát az elsőnek felszabaduló gépen.

Tétel: A $Pm||C_{\max}$ feladatra LS egy $(2 - \frac{1}{m})$-approximáció.

LS algoritmus Az m gépen, $J_1, J_2, \ldots, J_n$ sorrendben végezzük el a munkákat, minden munkát az elsőnek felszabaduló gépen.

Tétel: A $Pm||C_{\max}$ feladatra LS egy $(2 - \frac{1}{m})$-approximáció.

Biz: Jelölje $C^*_{\max}$ az optimális átfutási időt. Világos, hogy $C^*_{\max} \geq \frac{1}{m} \sum_{i=1}^{n} p_i$ ill. $C^*_{\max} \geq \max_i p_i$. Legyen t az LS szerint utolsónak befejezett J_k munka kezdési időpontja. Ekkor 0-tól t-ig mind az m gép dolgozik, tehát $t = C^{LS}_{\max} - p_k \leq \frac{1}{m} \sum_{i \neq k} p_i$, ezért $C^{LS}_{\max} = t + p_k \leq \frac{1}{m} \sum_{i \neq k} p_i + p_k = \frac{1}{m} \sum_{i=1}^{n} p_i + (1 - \frac{1}{m}) p_k \leq C^*_{\max} + (1 - \frac{1}{m}) C^*_{\max} = (2 - \frac{1}{m}) C^*_{\max}$. □

LS algoritmus Az m gépen, $J_1, J_2, \ldots, J_n$ sorrendben végezzük el a munkákat, minden munkát az elsőnek felszabaduló gépen.

Tétel: A $Pm||C_{\max}$ feladatra LS egy $(2 - \frac{1}{m})$-approximáció.

Biz: Jelölje $C^*_{\max}$ az optimális átfutási időt. Világos, hogy $C^*_{\max} \geq \frac{1}{m} \sum_{i=1}^{n} p_i$ ill. $C^*_{\max} \geq \max_i p_i$. Legyen t az LS szerint utolsónak befejezett J_k munka kezdési időpontja. Ekkor 0-tól t-ig mind az m gép dolgozik, tehát $t = C^{LS}_{\max} - p_k \leq \frac{1}{m} \sum_{i \neq k} p_i$, ezért $C^{LS}_{\max} = t + p_k \leq \frac{1}{m} \sum_{i \neq k} p_i + p_k = \frac{1}{m} \sum_{i=1}^{n} p_i + (1 - \frac{1}{m}) p_k \leq C^*_{\max} + (1 - \frac{1}{m}) C^*_{\max} = (2 - \frac{1}{m}) C^*_{\max}$. □

Tétel: A $Pm||C_{\max}$ feladatra LS ütemezést p_i szerint csökkenő (LPT) sorrendben végezve egy $\frac{4}{3}$-approximációs algoritmust kapunk. □

A ládapakolási probléma 53:08

Def: A ládapakolási(bin packing) problémában adott $a_1, a_2, \ldots, a_n$ méretű tárgyakat kell a lehető legkevesebb ládába elpakolni, ahol egy ládában a tárgyak összmérete legfeljebb 1 lehet.

Def: A ládapakolási(bin packing) problémában adott $a_1, a_2, \ldots, a_n$ méretű tárgyakat kell a lehető legkevesebb ládába elpakolni, ahol egy ládában a tárgyak összmérete legfeljebb 1 lehet.

Megj.: A ládapakolási feladat felfogható olyan inverz ütemezési problémaként, ahol $C_{\max} \leq 1$ feltétel mellett minimális számú géppel kell elvégezni a $J_1, \ldots, J_n$ munkákat.

A ládapakolási probléma 53:08

Def: A ládapakolási(bin packing) problémában adott $a_1, a_2, \ldots, a_n$ méretű tárgyakat kell a lehető legkevesebb ládába elpakolni, ahol egy ládában a tárgyak összmérete legfeljebb 1 lehet.

Megj.: A ládapakolási feladat felfogható olyan inverz ütemezési problémaként, ahol $C_{\max} \leq 1$ feltétel mellett minimális számú géppel kell elvégezni a $J_1, \ldots, J_n$ munkákat.

FF algoritmus A tárgyakat érkezési sorrendben pakoljuk a ládákba, mindegyiket a legelső olyanba, amelyikbe belefér.

FFD algoritmus A tárgyakat méret szerint csökkenő sorrendben pakoljuk a ládákba, mindegyiket a legelső olyanba, amelyikbe belefér.

A ládapakolási probléma 53:08

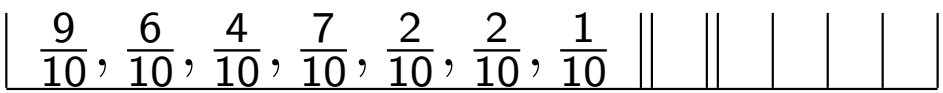
Def: A ládapakolási(bin packing) problémában adott $a_1, a_2, \ldots, a_n$ méretű tárgyakat kell a lehető legkevesebb ládába elpakolni, ahol egy ládában a tárgyak összmérete legfeljebb 1 lehet.

Megj.: A ládapakolási feladat felfogható olyan inverz ütemezési problémaként, ahol $C_{\max} \leq 1$ feltétel mellett minimális számú géppel kell elvégezni a $J_1, \ldots, J_n$ munkákat.

FF algoritmus A tárgyakat érkezési sorrendben pakoljuk a ládákba, mindegyiket a legelső olyanba, amelyikbe belefér.

FFD algoritmus A tárgyakat méret szerint csökkenő sorrendben pakoljuk a ládákba, mindegyiket a legelső olyanba, amelyikbe belefér.

Példa: FF:



A ládapakolási probléma 53:08

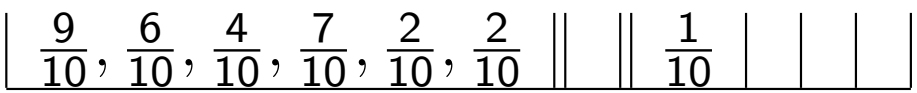
Def: A ládapakolási(bin packing) problémában adott $a_1, a_2, \ldots, a_n$ méretű tárgyakat kell a lehető legkevesebb ládába elpakolni, ahol egy ládában a tárgyak összmérete legfeljebb 1 lehet.

Megj.: A ládapakolási feladat felfogható olyan inverz ütemezési problémaként, ahol $C_{\max} \leq 1$ feltétel mellett minimális számú géppel kell elvégezni a $J_1, \ldots, J_n$ munkákat.

FF algoritmus A tárgyakat érkezési sorrendben pakoljuk a ládákba, mindegyiket a legelső olyanba, amelyikbe belefér.

FFD algoritmus A tárgyakat méret szerint csökkenő sorrendben pakoljuk a ládákba, mindegyiket a legelső olyanba, amelyikbe belefér.

Példa: FF:



A ládapakolási probléma 53:08

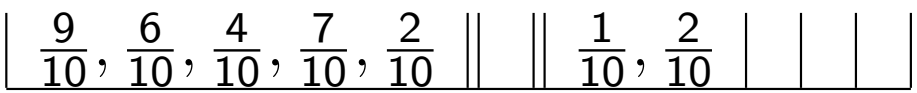
Def: A ládapakolási(bin packing) problémában adott $a_1, a_2, \ldots, a_n$ méretű tárgyakat kell a lehető legkevesebb ládába elpakolni, ahol egy ládában a tárgyak összmérete legfeljebb 1 lehet.

Megj.: A ládapakolási feladat felfogható olyan inverz ütemezési problémaként, ahol $C_{\max} \leq 1$ feltétel mellett minimális számú géppel kell elvégezni a $J_1, \ldots, J_n$ munkákat.

FF algoritmus A tárgyakat érkezési sorrendben pakoljuk a ládákba, mindegyiket a legelső olyanba, amelyikbe belefér.

FFD algoritmus A tárgyakat méret szerint csökkenő sorrendben pakoljuk a ládákba, mindegyiket a legelső olyanba, amelyikbe belefér.

Példa: FF:



A ládapakolási probléma 53:08

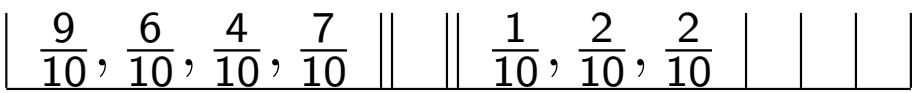
Def: A ládapakolási(bin packing) problémában adott $a_1, a_2, \ldots, a_n$ méretű tárgyakat kell a lehető legkevesebb ládába elpakolni, ahol egy ládában a tárgyak összmérete legfeljebb 1 lehet.

Megj.: A ládapakolási feladat felfogható olyan inverz ütemezési problémaként, ahol $C_{\max} \leq 1$ feltétel mellett minimális számú géppel kell elvégezni a $J_1, \ldots, J_n$ munkákat.

FF algoritmus A tárgyakat érkezési sorrendben pakoljuk a ládákba, mindegyiket a legelső olyanba, amelyikbe belefér.

FFD algoritmus A tárgyakat méret szerint csökkenő sorrendben pakoljuk a ládákba, mindegyiket a legelső olyanba, amelyikbe belefér.

Példa: FF:



A ládapakolási probléma 53:08

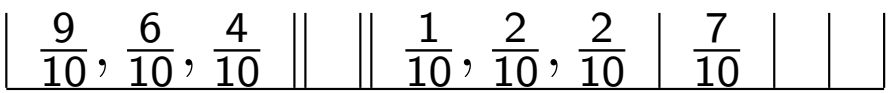
Def: A ládapakolási(bin packing) problémában adott $a_1, a_2, \ldots, a_n$ méretű tárgyakat kell a lehető legkevesebb ládába elpakolni, ahol egy ládában a tárgyak összmérete legfeljebb 1 lehet.

Megj.: A ládapakolási feladat felfogható olyan inverz ütemezési problémaként, ahol $C_{\max} \leq 1$ feltétel mellett minimális számú géppel kell elvégezni a $J_1, \ldots, J_n$ munkákat.

FF algoritmus A tárgyakat érkezési sorrendben pakoljuk a ládákba, mindegyiket a legelső olyanba, amelyikbe belefér.

FFD algoritmus A tárgyakat méret szerint csökkenő sorrendben pakoljuk a ládákba, mindegyiket a legelső olyanba, amelyikbe belefér.

Példa: FF:



A ládapakolási probléma 53:08

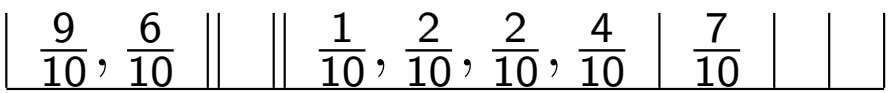
Def: A ládapakolási(bin packing) problémában adott $a_1, a_2, \ldots, a_n$ méretű tárgyakat kell a lehető legkevesebb ládába elpakolni, ahol egy ládában a tárgyak összmérete legfeljebb 1 lehet.

Megj.: A ládapakolási feladat felfogható olyan inverz ütemezési problémaként, ahol $C_{\max} \leq 1$ feltétel mellett minimális számú géppel kell elvégezni a $J_1, \ldots, J_n$ munkákat.

FF algoritmus A tárgyakat érkezési sorrendben pakoljuk a ládákba, mindegyiket a legelső olyanba, amelyikbe belefér.

FFD algoritmus A tárgyakat méret szerint csökkenő sorrendben pakoljuk a ládákba, mindegyiket a legelső olyanba, amelyikbe belefér.

Példa: FF:



A ládapakolási probléma 53:08

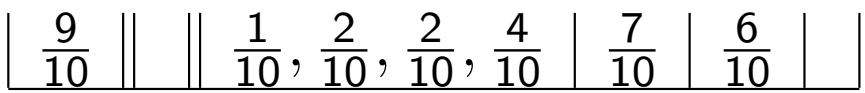
Def: A ládapakolási(bin packing) problémában adott $a_1, a_2, \ldots, a_n$ méretű tárgyakat kell a lehető legkevesebb ládába elpakolni, ahol egy ládában a tárgyak összmérete legfeljebb 1 lehet.

Megj.: A ládapakolási feladat felfogható olyan inverz ütemezési problémaként, ahol $C_{\max} \leq 1$ feltétel mellett minimális számú géppel kell elvégezni a $J_1, \ldots, J_n$ munkákat.

FF algoritmus A tárgyakat érkezési sorrendben pakoljuk a ládákba, mindegyiket a legelső olyanba, amelyikbe belefér.

FFD algoritmus A tárgyakat méret szerint csökkenő sorrendben pakoljuk a ládákba, mindegyiket a legelső olyanba, amelyikbe belefér.

Példa: FF:



A ládapakolási probléma 53:08

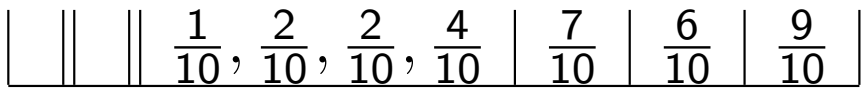
Def: A ládapakolási(bin packing) problémában adott $a_1, a_2, \ldots, a_n$ méretű tárgyakat kell a lehető legkevesebb ládába elpakolni, ahol egy ládában a tárgyak összmérete legfeljebb 1 lehet.

Megj.: A ládapakolási feladat felfogható olyan inverz ütemezési problémaként, ahol $C_{\max} \leq 1$ feltétel mellett minimális számú géppel kell elvégezni a $J_1, \ldots, J_n$ munkákat.

FF algoritmus A tárgyakat érkezési sorrendben pakoljuk a ládákba, mindegyiket a legelső olyanba, amelyikbe belefér.

FFD algoritmus A tárgyakat méret szerint csökkenő sorrendben pakoljuk a ládákba, mindegyiket a legelső olyanba, amelyikbe belefér.

Példa: FF: $k = 4$



Def: A ládapakolási(bin packing) problémában adott $a_1, a_2, \ldots, a_n$ méretű tárgyakat kell a lehető legkevesebb ládába elpakolni, ahol egy ládában a tárgyak összmérete legfeljebb 1 lehet.

Megj.: A ládapakolási feladat felfogható olyan inverz ütemezési problémaként, ahol $C_{\max} \leq 1$ feltétel mellett minimális számú géppel kell elvégezni a $J_1, \ldots, J_n$ munkákat.

FF algoritmus A tárgyakat érkezési sorrendben pakoljuk a ládákba, mindegyiket a legelső olyanba, amelyikbe belefér.

FFD algoritmus A tárgyakat méret szerint csökkenő sorrendben pakoljuk a ládákba, mindegyiket a legelső olyanba, amelyikbe belefér.

Példa:

A ládapakolási probléma 53:08

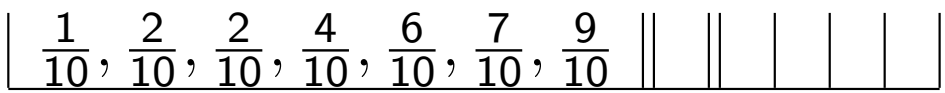
Def: A ládapakolási(bin packing) problémában adott $a_1, a_2, \ldots, a_n$ méretű tárgyakat kell a lehető legkevesebb ládába elpakolni, ahol egy ládában a tárgyak összmérete legfeljebb 1 lehet.

Megj.: A ládapakolási feladat felfogható olyan inverz ütemezési problémaként, ahol $C_{\max} \leq 1$ feltétel mellett minimális számú géppel kell elvégezni a $J_1, \ldots, J_n$ munkákat.

FF algoritmus A tárgyakat érkezési sorrendben pakoljuk a ládákba, mindegyiket a legelső olyanba, amelyikbe belefér.

FFD algoritmus A tárgyakat méret szerint csökkenő sorrendben pakoljuk a ládákba, mindegyiket a legelső olyanba, amelyikbe belefér.

Példa: FFD:



A ládapakolási probléma 53:08

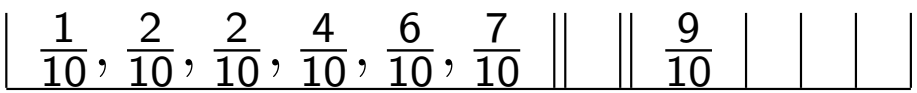
Def: A ládapakolási(bin packing) problémában adott $a_1, a_2, \ldots, a_n$ méretű tárgyakat kell a lehető legkevesebb ládába elpakolni, ahol egy ládában a tárgyak összmérete legfeljebb 1 lehet.

Megj.: A ládapakolási feladat felfogható olyan inverz ütemezési problémaként, ahol $C_{\max} \leq 1$ feltétel mellett minimális számú géppel kell elvégezni a $J_1, \ldots, J_n$ munkákat.

FF algoritmus A tárgyakat érkezési sorrendben pakoljuk a ládákba, mindegyiket a legelső olyanba, amelyikbe belefér.

FFD algoritmus A tárgyakat méret szerint csökkenő sorrendben pakoljuk a ládákba, mindegyiket a legelső olyanba, amelyikbe belefér.

Példa: FFD:



A ládapakolási probléma 53:08

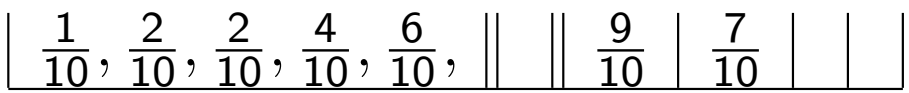
Def: A ládapakolási(bin packing) problémában adott $a_1, a_2, \ldots, a_n$ méretű tárgyakat kell a lehető legkevesebb ládába elpakolni, ahol egy ládában a tárgyak összmérete legfeljebb 1 lehet.

Megj.: A ládapakolási feladat felfogható olyan inverz ütemezési problémaként, ahol $C_{\max} \leq 1$ feltétel mellett minimális számú géppel kell elvégezni a $J_1, \ldots, J_n$ munkákat.

FF algoritmus A tárgyakat érkezési sorrendben pakoljuk a ládákba, mindegyiket a legelső olyanba, amelyikbe belefér.

FFD algoritmus A tárgyakat méret szerint csökkenő sorrendben pakoljuk a ládákba, mindegyiket a legelső olyanba, amelyikbe belefér.

Példa: FFD:



A ládapakolási probléma 53:08

Def: A ládapakolási(bin packing) problémában adott $a_1, a_2, \ldots, a_n$ méretű tárgyakat kell a lehető legkevesebb ládába elpakolni, ahol egy ládában a tárgyak összmérete legfeljebb 1 lehet.

Megj.: A ládapakolási feladat felfogható olyan inverz ütemezési problémaként, ahol $C_{\max} \leq 1$ feltétel mellett minimális számú géppel kell elvégezni a $J_1, \ldots, J_n$ munkákat.

FF algoritmus A tárgyakat érkezési sorrendben pakoljuk a ládákba, mindegyiket a legelső olyanba, amelyikbe belefér.

FFD algoritmus A tárgyakat méret szerint csökkenő sorrendben pakoljuk a ládákba, mindegyiket a legelső olyanba, amelyikbe belefér.

Példa: FFD:

A ládapakolási probléma 53:08

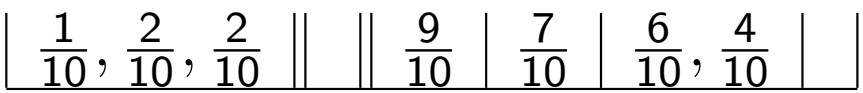
Def: A ládapakolási(bin packing) problémában adott $a_1, a_2, \ldots, a_n$ méretű tárgyakat kell a lehető legkevesebb ládába elpakolni, ahol egy ládában a tárgyak összmérete legfeljebb 1 lehet.

Megj.: A ládapakolási feladat felfogható olyan inverz ütemezési problémaként, ahol $C_{\max} \leq 1$ feltétel mellett minimális számú géppel kell elvégezni a $J_1, \ldots, J_n$ munkákat.

FF algoritmus A tárgyakat érkezési sorrendben pakoljuk a ládákba, mindegyiket a legelső olyanba, amelyikbe belefér.

FFD algoritmus A tárgyakat méret szerint csökkenő sorrendben pakoljuk a ládákba, mindegyiket a legelső olyanba, amelyikbe belefér.

Példa: FFD:



A ládapakolási probléma 53:08

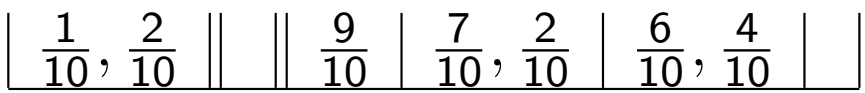
Def: A ládapakolási(bin packing) problémában adott $a_1, a_2, \ldots, a_n$ méretű tárgyakat kell a lehető legkevesebb ládába elpakolni, ahol egy ládában a tárgyak összmérete legfeljebb 1 lehet.

Megj.: A ládapakolási feladat felfogható olyan inverz ütemezési problémaként, ahol $C_{\max} \leq 1$ feltétel mellett minimális számú géppel kell elvégezni a $J_1, \ldots, J_n$ munkákat.

FF algoritmus A tárgyakat érkezési sorrendben pakoljuk a ládákba, mindegyiket a legelső olyanba, amelyikbe belefér.

FFD algoritmus A tárgyakat méret szerint csökkenő sorrendben pakoljuk a ládákba, mindegyiket a legelső olyanba, amelyikbe belefér.

Példa: FFD:



A ládapakolási probléma 53:08

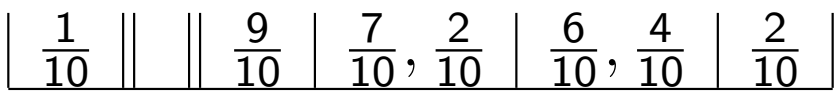
Def: A ládapakolási(bin packing) problémában adott $a_1, a_2, \ldots, a_n$ méretű tárgyakat kell a lehető legkevesebb ládába elpakolni, ahol egy ládában a tárgyak összmérete legfeljebb 1 lehet.

Megj.: A ládapakolási feladat felfogható olyan inverz ütemezési problémaként, ahol $C_{\max} \leq 1$ feltétel mellett minimális számú géppel kell elvégezni a $J_1, \ldots, J_n$ munkákat.

FF algoritmus A tárgyakat érkezési sorrendben pakoljuk a ládákba, mindegyiket a legelső olyanba, amelyikbe belefér.

FFD algoritmus A tárgyakat méret szerint csökkenő sorrendben pakoljuk a ládákba, mindegyiket a legelső olyanba, amelyikbe belefér.

Példa: FFD:



A ládapakolási probléma 53:08

Def: A ládapakolási(bin packing) problémában adott $a_1, a_2, \ldots, a_n$ méretű tárgyakat kell a lehető legkevesebb ládába elpakolni, ahol egy ládában a tárgyak összmérete legfeljebb 1 lehet.

Megj.: A ládapakolási feladat felfogható olyan inverz ütemezési problémaként, ahol $C_{\max} \leq 1$ feltétel mellett minimális számú géppel kell elvégezni a $J_1, \ldots, J_n$ munkákat.

FF algoritmus A tárgyakat érkezési sorrendben pakoljuk a ládákba, mindegyiket a legelső olyanba, amelyikbe belefér.

FFD algoritmus A tárgyakat méret szerint csökkenő sorrendben pakoljuk a ládákba, mindegyiket a legelső olyanba, amelyikbe belefér.

Példa: FFD: $k = 4$

		$\frac{9}{10}, \frac{1}{10}$	$\frac{7}{10}, \frac{2}{10}$	$\frac{6}{10}, \frac{4}{10}$	$\frac{2}{10}$

Def: A ládapakolási(bin packing) problémában adott $a_1, a_2, \ldots, a_n$ méretű tárgyakat kell a lehető legkevesebb ládába elpakolni, ahol egy ládában a tárgyak összmérete legfeljebb 1 lehet.

Megj.: A ládapakolási feladat felfogható olyan inverz ütemezési problémaként, ahol $C_{\max} \leq 1$ feltétel mellett minimális számú géppel kell elvégezni a $J_1, \ldots, J_n$ munkákat.

FF algoritmus A tárgyakat érkezési sorrendben pakoljuk a ládákba, mindegyiket a legelső olyanba, amelyikbe belefér.

FFD algoritmus A tárgyakat méret szerint csökkenő sorrendben pakoljuk a ládákba, mindegyiket a legelső olyanba, amelyikbe belefér.

Tétel: Ha a ládapakolási feladat egy inputjához k láda elegendő, akkor az FF algoritmus legfeljebb $\frac{17}{10}k + 2$, az FFD pedig legfeljebb $\frac{11}{9}k + 7$ ládát használ.
(Az FF lényegében $1,7$-, az FFD $1,22$-közelítő algoritmus.) □

Éljen!