

HORVÁTH MARKÓ

Buszvezető ütemezés

TDK DOLGOZAT

Alkalmazott Matematikus MSc

Operációkutatás szakirány

TÉMAVEZETŐ:

Dr. Kis Tamás

Operációkutatási Tanszék



2013. Január 9.

Tartalomjegyzék

1. Bevezető	4
2. A probléma ismertetése és modellezése	5
2.1. Áttekintés	5
2.2. A vizsgálni kívánt probléma	5
2.3. A workpiece gráf	6
2.4. Az IP-modell	6
2.5. Oszlogenerálás	7
2.6. Az erőforrás korlátos legrövidebb út	8
3. Az erőforrás korlátos legrövidebb út probléma	9
3.1. Egészértékű programozási modell	9
3.2. Lagrange-relaxáció	9
3.3. Szubgradiens algoritmus	10
3.4. Redukciós tesztek	11
3.5. Branch and Bound	12
4. A megoldás implementációja, tesztelése	14
4.1. I. Algoritmus	14
4.2. II. Algoritmus	14
4.3. Tesztelés kisebb inputon	14
4.4. Tesztelés nagyobb inputon	15
4.5. A Branch and Bound algoritmusban szereplő tesztek	15
4.6. Konklúzió	16
5. Összefoglalás, kitekintés	17
Függelék	18
Táblázatok	18
Ábrák	19
Hivatkozások	24

Kivonat

A dolgozatban egy tömegközlekedésből származó ütemezési problémával, a vezető ütemezéssel foglalkozunk. A célunk, hogy egy adott napra elkészítsük a vezetők beosztását a megadott járatokhoz. A vezetők munkaidejére alsó és felső korlátok adottak, továbbá biztosítanunk kell számukra egy hosszabb ebédszünetet. A problémát egészértékű programozási feladatként fogalmazzuk meg, melynek megoldása során oszlopgenerálást alkalmazunk. Az oszlopgenerálás szubproblémája erőforrás korlátos legrövidebb út keresésére vezet vissza, melynek megoldására kétféle módszert dolgozunk ki.

A feladat megoldására kidolgozott két algoritmust implementáljuk, és a BKV menetrendjeiből elkészített inputokon teszteljük.

1. Bevezető

A tömegközlekedésben gyakran felmerülnek ütemezési kérdések. A járművezetők beosztásainak elkészítése egy összetett folyamat része, melyet a probléma bonyolultsága több szinten kell kezelni. Az első szinten helyezkedik el az úgynevezett jármű ütemezési feladat (*Vehicle scheduling*), melynek során a menetrendben szereplő utakhoz kell járműveket rendelni. A cél az operatív költségek minimalizálása, melybe beletartoznak például a járművek használatának költségei. A második szinten helyezkedik el az általunk vizsgálni kívánt vezető ütemezési feladat (*Driver scheduling*). Ekkor rendelkezésre állnak a jármű ütemezés során elkészített járatok. A célunk egy adott napra, a vezetőket hozzárendelni az egyes járatokhoz, bizonyos feltételeknek megfelelően. Tipikus megkorlátozások például a vezetők munkaidejére vonatkozó alsó-felső korlátok, illetve a műszak közbe beiktatott szünetek. Az efféle feltételek országonként, területenként változnak, de legtöbb esetben biztosítani kell a vezető számára egy hosszabb ebédszünetet, esetleg több, rövidebb szünetet. A szünetek a műszakot több részre vágják, melyek hosszára szintén bevezethetők különböző korlátok. A következő szinten helyezkedik el a vezető beosztási feladat (*Driver rostering*), ahol a vezetők hosszabb távú beosztását kell elkészíteni. Az előző szinten elkészített napi beosztásokból kell a vezetők számára összeállítani néhány hétre előre a munkarendjüket. Itt is különféle korlátozási feltételek lehetnek, mint például a heti, havi munkaidőre vonatkozó alsó-felső korlátok, vagy az egymást követő napokon teljesítendő munkaórák száma.

A dolgozatban tehát a vezető ütemezéssel fogunk foglalkozni. A 2. fejezet elején egy rövid szakirodalmi áttekintést nyújtunk, majd részletesen ismertetjük a vizsgálni kívánt problémát. A feladatot egy egészértékű programozási feladatként fogalmazzuk meg, melynek megoldása során oszlopgenerálást alkalmazunk. A kihívást az új oszlop legenerálása jelenti, mely egy erőforrás korlátos legrövidebb út keresésére vezet vissza. Ezzel a problémával a 3. fejezetben részletesen foglalkozunk.

A TDK munka fő célja egy olyan program elkészítése volt, amely az inputként megadott járatokhoz elkészíti az adott feltételeknek megfelelően a vezetők beosztásait. Ezért a modellezésen, és megoldás kidolgozásán túl, fő eredményeként a megoldás implementációit emelném ki. Az első implementáció során az egészértékű programozási feladatot, és az oszlopgenerálás árazási feladatát is egy általános MIP-megoldó segítségével oldottam meg. Egy hatékonyabb algoritmus reményében készítettem el a második implementációt, ahol a 3. fejezetben bemutatott módszerekkel oldottam meg az árazási feladatot. A két implementáció teszteléséhez a BKV menetrendjeiből készítettem input fájlokat. Az implementációk részletezéséről, a futási eredményekről, és azok értékeléséről a 4. fejezetben lesz szó.

2. A probléma ismertetése és modellezése

2.1. Áttekintés

A buszvezető ütemezés régóta vizsgált probléma. A bevezetőben már említettük, hogy ebben a feladatban az adott járatokhoz kell a vezetőket beosztani bizonyos feltételeknek megfelelően. A feltételekre Wren és Rousseau több példát is említ ([10]), de tipikus megszorítás a vezetők munkaidejére vonatkozó alsó-felső korlát. Rendszeresen felmerülnek olyan megkötések, melyek a vezetők pihenőidejére vonatkoznak. Többnyire biztosítani kell a vezetők számára egy hosszabb ebédszünetet, vagy akár több, rövidebb szünetet is. Ezek a korlátok területenként, országonként változnak, és függnak tömegközlekedési vállalattól is.

A feladat megoldására alkalmazható többek között lineáris programozás ([4],[11]), evolúciós algoritmus ([8]), tabu keresés ([9]) is. A lineáris programozási modell esetén a megoldásra gyakran oszlopgenerálást alkalmaznak ([3]).

2.2. A vizsgálni kívánt probléma

Tegyük fel, hogy van egy központi állomásunk, és a buszok vagy villamosok e központi állomás és más állomások között ingáznak a nap folyamán. Budapesti példánál maradva egy ilyen központi állomás lehet a Mexikói út, amely a 3-as és 69-es villamosjáratok végállomása, vagy akár a Széll Kálmán tér, ami jelenleg az egyik legfontosabb közlekedési csomópontja Budapestnek. Többek között 14 busz-, és 4 villamosjárat végállomása ez.

Tegyük fel, hogy egy jármű (továbbiakban busz) napi menete abból áll, hogy a központi állomásról fordulókat teljesít. Egy forduló (*workpiece*) alatt azt értjük, hogy elindul a központi állomásról majd bizonyos út megtétele után oda visszatér. Tulajdonképpen a fordulók indulási és érkezési időpontjai lesznek a problémánk inputja. Egy vezető *beosztását* úgy adjuk meg, hogy megmondjuk melyik fordulókat kell teljesítenie. Ez magában foglalja azt, hogy a vezetők kizárólag a központi állomáson válthatják egymást.

Két forduló között persze elképzelhető valamennyi szünet. Ha ez a szünet kevesebb, mint a percet vesz igénybe, akkor *átzállásnak* nevezzük, ha pedig legalább e_l , legfeljebb e_u percig tart, akkor *ebédszünetnek* nevezzük. Más szünetet nem engedünk meg. Egy vezető beosztását akkor nevezzük *megengedettnek*, ha legalább L , legfeljebb U órát ölel fel (melybe beleértjük a fordulók teljesítését és a szüneteket is), és pontosan egy ebédszünetet tartalmaz. Természetesen egy beosztás megengedettségéhez az is hozzátartozik, hogy nem szerepelhet benne két olyan forduló, melyek időben fedik egymást. Az olyan példáinkban, melyek egy teljes nap fordulóit tartalmazzák, például a következő korlátokat tekinthetjük: $a = 30$ (perc), $e_l = 30$ (perc), $e_u = 60$ (perc), $L = 7$ (óra), $U = 9$ (óra).

2.3. A workpiece gráf

A feladat modellezése és megoldása miatt elkészítjük a következő irányított gráfot. Minden w fordulónak felveszünk egy irányított $w'w''$ *workpiece-élt*. Az él hossza a forduló időtartama lesz. Ha a w_1 forduló után közvetlenül elvégezhető a w_2 forduló egy átszállás vagy egy ebédszünet közbeiktatásával, vagy esetleg szünet nélkül, akkor behúzzuk a $w'_1w'_2$ irányított élt. Ennek az élnak a hossza a szünet időtartama lesz, továbbá ha ez egy ebédszünetnek felel meg, akkor az élt *ebéd-élnak* nevezzük. Miután ezt minden fordulópárra megtettük, felveszünk egy s és t pontot. Minden w fordulóhoz bekötjük az sw' és $w''t$ éleket, melyeknek hosszuk 0 lesz. A 1. ábrán egy 10 fordulóból álló inputból készítettük el a workpiece gráfot.

A workpiece gráf tehát egy aciklikus gráf, melyben s egyértelmű forrás, és t egyértelmű nyelő. Jegyezzük meg, hogy minden megengedett beosztásnak megfelel a workpiece gráfban egy st út a következőképpen. Ha a beosztás sorra a $w_1, w_2, \dots, w_k$ fordulókba áll, akkor létezik az $(s, w'_1, w''_1, w'_2, w''_2, \dots, w'_k, w''_k, t)$ út a gráfban. És ez az út pontosan ott használ ebéd-élt, ahol a beosztásban ebédszünet következik. Hasonlóan egy st útnak is megfelel egy beosztás, de az már koránt sem biztos, hogy megengedett. Ennek megfelelően egy st utat megengedettnak nevezünk, ha neki megengedett beosztás felel meg.

Nyilvánvaló, hogy ha adottak a fordulók indulási és érkezési időpontjai, akkor a workpiece gráf egyértelműen elkészíthető. Megjegyezzük továbbá, hogy a workpiece gráf a súlyokkal és a (workpiece, ebéd) címkékkel a probléma szempontjából minden fontos adatot tartalmaz (kivéve az L és U értékeket). A továbbiakban ezért nem foglalkozunk a gráf elkészítésével, hanem egyszerűen a gráfot és az L, U értékeket tekintjük a problémánk inputjának.

Érdemes itt megjegyezni, hogy a megoldásunk a fent leírt feladatnál általánosabb problémák megoldására is alkalmas lesz. Ha például nem követeljük meg, hogy a vezetők csak a központi állomáson válthassák egymást, akkor nem kell a megoldásunkon, hanem elegendő a workpiece gráf elkészítésén változtatni. Ha például az A és B végállomásokon is megengedünk átszállásokat, akkor A és B közötti fordulókba készítjük el a gráfot. Átszálló- illetve ebéd-éleket kizárólag akkor húzunk be, ha az átszállás illetve ebédszünet azonos állomáson történik. Hasonlóan hozzávehetők a modellhez azon fordulók, amikor a garázsba ki, illetve garázsba visszaviszik a járműveket.

2.4. Az IP-modell

Jelölje w a fordulók számát, d a megengedett beosztások számát. Jelölje A az a mátrixot, melynek oszlopai a megengedett beosztásoknak felelnek meg, sorai pedig a fordulóknak, és $A_{ij} = 1$, ha a j . beosztás tartalmazza az i . fordulót, 0 egyébként. Tehát $A \in \{0, 1\}^{w \times d}$ a forduló-beosztás incidencia mátrix. Jelölje x_j azt, hogy a j . beosztáshoz rendelünk-e vezetőt ($x_j = 1$) vagy sem

($x_j = 0$). Az következő egészértékű programozási feladatot írjuk fel:

$$\begin{aligned} \min \sum_{j=1}^d x_j \\ Ax \geq 1 \\ x \geq 0 \\ x \in \mathbb{Z}^d \end{aligned} \tag{1}$$

Rögtön kezdjük néhány megjegyzéssel a modellel kapcsolatban. $Ax \geq 1$ azt jelenti, hogy minden fordulót szeretnénk legalább egyszer fedni. Azért nem $Ax = 1$ szerepel, mert szinte biztosan nem tudunk olyan beosztást készíteni, ahol minden forduló pontosan egyszer lenne fedve. Például ha csak egyetlen buszunk van, akkor erre nagyon könnyű példát adni. Ebből már látható, hogy a problémánknak megfelelő tökéletes megoldást nem fog garantálni a modell, de ha csak kevés olyan fordulónk van, amire több vezető is be van osztva, azzal most megelégszünk. Mivel x nemnegatív, valamint az A mátrix 0-1 értékű, ezért $x^* \leq 1$ minden optimális x^* megoldásra. Ezek után látható, hogy a célfüggvény a vezetők számát minimalizálja.

Az A mátrix elkészítése bonyolult feladat lenne, hiszen minden megengedett beosztást el kellene készítenünk. Másrészt ha még el is készítenénk, akkor nagy méretű IP-feladathoz jutnánk, melynek közvetlen megoldása nem hatékony. Ezért (1)-t oszlopgenerálással oldjuk meg.

2.5. Oszlopgenerálás

Nagy méretű IP-feladatokra bevett szokás oszlopgenerálást alkalmazni ([1]), mely esetünkben a következőképpen néz ki. Először kiindulunk egy olyan beosztásokból álló D mátrixból, melyek együtt minden fordulót lefednek, azaz melynek létezik megengedett megoldása (1)-re. Egy ilyen D mátrixot akár a megengedett beosztások mohó keresésével is elkészíthetünk. Mi egy más megoldást választunk, melyre később visszatérünk.

Írjuk fel (1) LP-relaxáltját egy ilyen D mátrixra (2) és annak duálisát (3):

$$\min \{x \cdot 1 : Dx \geq 1, x \geq 0\} \tag{2}$$

$$\max \{y \cdot 1 : yD \leq 1, y \geq 0\} \tag{3}$$

Tegyük fel, hogy megoldottuk a relaxált problémát és kaptunk egy (x, y) primál-duál megoldaspárját (2)-(3)-nak. Az x -nek megfelel (1) egy megengedett $\hat{x}$ megoldása: $\hat{x} = (x, 0)$, ahol x a D -beli oszlopokhoz tartozik. Ha $yA \leq 1$ is teljesül, akkor $(\hat{x}, y)$ primál-, illetve duál-megengedett megoldások (2)-(3)-ra, és ezért optimális megoldások ($\hat{x} \cdot 1 = x \cdot 1 = y \cdot 1$).

Tehát $yA \leq 1$ teljesülését kell ellenőriznünk. Az y valójában egy költségfüggvényt ad a fordulón, és $ya \leq 1$ azt jelenti, hogy az a beosztás összköltsége y szerint legfeljebb 1. Legyen ξ az y szerint legdrágább megengedett beosztás költsége. Ha $\xi \leq 1$, akkor minden megengedett beosztás költsége legfeljebb 1, és így $yA \leq 1$ teljesül. Viszont ha $\xi > 1$, akkor $yA \leq 1$

nem teljesül. Ebben az esetben a ξ -nek megfelelő megengedett beosztást hozzávesszük D -hez, és megoldjuk a feladatot most már az új mátrixra. Ezt az eljárást folytatjuk amíg optimális megoldáshoz jutunk, vagy míg egy bizonyos iterációs számot el nem érünk. Mindkét esetben a legenerált mátrixra oldjuk meg az (1) IP-feladatunkat.

A fenti eljárás ismeretében megfigyelhetjük, hogy a kiinduló D mátrixnak nem kell feltétlenül megengedett beosztásokból állnia, ezért mi a $w \times w$ méretű egység mátrixból indulunk ki. Ugyanis az oszlopgenerálás befejeztével egy optimális megoldás valószínűleg már nem használja a kiinduló oszlopokat, hiszen ha valamely fordulót lefed egy megengedett beosztás, akkor a fordulónak megfelelő D -beli egységvektort már nem használja optimális megoldás. Ahhoz, hogy ezt biztosítsuk, a kezdeti oszlopokra nagyobb súlyt adhatunk a célfüggvényben.

A legfőbb kérdés ezek után az, hogyan találhatjuk meg a legdrágább megengedett beosztást. Ezt nevezzük az oszlopgenerálás szubproblémájának, vagy másképp árazási feladatnak. Ehhez a workpiece gráfot fogjuk használni. Az y duálváltozókat fogjuk a megfelelő workpiece-élekre írni, minden más élre 0 költséget írunk. Ekkor minden megengedett beosztás költsége y szerint egyenlő a neki megfelelő megengedett st út költségével. Valójában nem az y , hanem a $-y$ értékeket fogjuk az élekre írni, ezért a továbbiakban a workpiece gráfban legolcsóbb megengedett st utat keresünk.

2.6. Az erőforrás korlátos legrövidebb út

A *resource constrained shortest path* problémának tekintsük a következő változatát. Adott egy irányított $D = (V, A)$ gráf kitüntetett $s, t \in V$ pontokkal. Minden $a \in A$ élre adott a c_a költség, valamint az $R_a = (r_a^1, \dots, r_a^k)$ erőforrás vektor. Adottak tovább a $\lambda_1, \dots, \lambda_k$ értékek is ($c_a, \lambda_i, r_a^i \in \mathbb{R}$). A feladat olyan c szerint legolcsóbb P st utat keresni, melyre $\sum_{a \in P} r_a^i \leq \lambda_i$ fennáll minden i -re, vagyis az út minden erőforrás korlátot teljesít.

A feladat általában NP-teljes ([5]). A megoldási módszerek között szerepel például a dinamikus programozás, vagy a címkéző algoritmus ([6],[7]).

Valójában a mi szubproblémánk is megfogalmazható erőforrás korlátos legrövidebb (legolcsóbb) út kereséseként. A c költségfüggvény értelemszerűen jön az eddigiekből: $c_a = -y_a$, ha a workpiece-él, egyébként pedig 0. Az st út megengedettségiéhez két dolog szükséges. Egyrészt az összhosszának L és U korlátok közé kell esnie, tehát minden a élen lesz egy r_a^1 értékünk, amely az él hosszát jelöli. Másrészt az st útnak pontosan 1 ebéd-élt kell tartalmaznia, ezért minden a élen lesz egy r_a^2 értékünk úgy, hogy $r_a^2 = 1$, ha a ebéd-él, és 0 különben. Ezek után olyan legolcsóbb P st utat kell keresünk, mely teljesíti a következőket: $L \leq \sum_{a \in P} r_a^1 \leq U$, $\sum_{a \in P} r_a^2 = 1$. A következő fejezetben ennek a feladatnak a megoldásával foglalkozunk.

3. Az erőforrás korlátos legrövidebb út probléma

Ebben a fejezetben az erőforrás korlátos legrövidebb út (RCSP) megoldásával foglalkozunk. A probléma inputja az előző szakaszban leírt workpiece gráf, valamint ennek élein a c költségfüggvény, az (r^1, r^2) erőforrás vektor. Inputként kapjuk még az L_1, L_2, U_1, U_2 erőforrás korlátokat is. Az r^1 érték az él hosszát jelöli, az r^2 értéke pedig az ebéd-éleken 1, máshol 0.

A megoldás implementálása során Beasley és Christofides cikkét ([2]) követjük. A cikkben leírt modellt több helyen le tudjuk egyszerűsíteni, és hatékonyabb megoldásokat tudunk alkalmazni, ugyanis az input gráfunk nagyon speciális. Ezt helyenként ki is hangsúlyozzuk. A megoldás során az (RCSP) feladatot egészértékű programozási feladatként fogalmazzuk meg. Bizonyos egyenlőtlenségeket relaxálva korlátok nélküli legolcsóbb út keresésére vezet a probléma, melyet már könnyedén meg tudunk oldani. Az így kapott lagrange alsó korlát maximalizására szubgradiens módszert alkalmazunk, mely után az (RCSP) feladat optimális megoldását Branch and Bound algoritmussal határozzuk meg.

3.1. Egészértékű programozási modell

A gráf minden ij élére bevezetünk egy x_{ij} változót. Azt szeretnénk, hogy x_{ij} értéke 1 legyen, ha az optimális út tartalmazza az ij élt, különben 0. Az (RCSP) feladat a következőképp írható fel egészértékű programként:

$$\begin{aligned} \min \quad & \sum_{ij \in A} c_{ij} x_{ij} \\ L_k \leq \quad & \sum_{ij \in A} r_{ij}^k x_{ij} \leq U_k \quad k = 1, 2 \\ \sum_{i: ij \in A} x_{ij} = \quad & \sum_{i: ji \in A} x_{ji} \quad \forall j \in V - \{s, t\} \\ \sum_{j: sj \in A} x_{sj} = \quad & 1 \\ x_{ij} \in \{0, 1\} \quad & \forall ij \in A \end{aligned} \tag{RCSP}$$

Az utolsó három egyenlőtlenség biztosítja, hogy x 0-1 értékű legyen, és az 1 értékű élek st -utat határozzanak meg. Itt fontos kiemelni, hogy a gráf aciklikus, különben az 1 súlyú élek st sétát alkothattának, sőt ettől diszjunkt köröket is adhatnának. Az első feltétel biztosítja az erőforrás korlátok teljesülését, és ezek után látható, hogy a célfüggvény valóban a legolcsóbb erőforrás korlátos út költségét adja.

3.2. Lagrange-relaxáció

Relaxáljuk (RCSP) első feltételét. Legyenek $s_k \geq 0$ ($k = 1, 2$) az alsó korlátokhoz tartozó lagrange multiplikátorok, $t_k \geq 0$ ($k = 1, 2$) pedig a felső korlátokhoz tartozók. A lagrange alsó

korlát programunk (LLBP) ebből az alábbi lesz:

$$\begin{aligned}
\min \sum_{ij \in A} \left(c_{ij} + \sum_{k=1}^2 (t_k - s_k) r_{ij}^k \right) x_{ij} + \sum_{k=1}^2 (s_k L_k - t_k U_k) \\
\sum_{i: ij \in A} x_{ij} = \sum_{i: ji \in A} x_{ji} \quad \forall j \in V - \{s, t\} \\
\sum_{j: sj \in A} x_{sj} = 1 \\
x_{ij} \in \{0, 1\} \quad \forall ij \in A
\end{aligned} \tag{LLBP}$$

Az (LLBP) feladat nem más, mint egy legrövidebb st út keresése a C költségfüggvény szerint, ahol

$$C_{ij} = c_{ij} + \sum_{k=1}^2 (t_k - s_k) r_{ij}^k$$

A legolcsóbb út a gráfunkban könnyen számolható, nincs szükség semmilyen speciális legolcsóbb út kereső algoritmusra. Jelölje ugyanis $d_s(v)$ a legrövidebb sv út költségét. Ekkor a csúcsok egy topologikus sorrendjében haladva ($d_s(s) = 0$) az aktuális v csúcsra: $d_s(v) = \min\{d_s(u) + C_{uv} : uv \in A\}$.

3.3. Szubgradiens algoritmus

A Lagrange-relaxáció tulajdonságai alapján bármilyen $s_k, t_k \geq 0$ multiplikátorok mellett (LLBP) optimumának értéke alsó becslés (RCSP) optimumának értékére. Az alsó becslés maximalizálása érdekében szubgradiens módszert fogunk alkalmazni. Vezessük be a következő jelöléseket. $Z_{\max}$ jelölje az eddig talált legnagyobb Lagrange alsó korlátot, Z_{UB} pedig az eddig talált legjobb felső korlátot (RCSP) optimumára. Z_{UB} kezdőértékét például úgy állíthatjuk be, hogy a gráfban keresünk egy megengedett st utat, és ennek költségét adjuk Z_{UB} -nak (az eredeti költség szerint). $B_{\max}$ jelölje az iterációk maximális számát.

- (i) Inicializáljuk a multiplikátorokat: $s_k = 0 = t_k$ ($k = 1, 2$). $Z_{\max}$ értékét $-\infty$ -nek inicializáljuk, a B iteráció számláló értékét 0-ra állítjuk.
- (ii) Megoldjuk (LLBP)-t az aktuális multiplikátorokkal, a megoldást jelölje X . A Z_{LB} alsó korlátot az alábbi kifejezés adja:

$$Z_{\text{LB}} = \sum_{ij \in A} C_{ij} X_{ij} + \sum_{k=1}^2 (s_k L_k - t_k U_k)$$

- (iii) Ha $Z_{\text{LB}} > Z_{\max}$, akkor $Z_{\max}$ értékét Z_{LB} -re állítjuk.
- (iv) Ha X megengedett megoldás (RCSP) feladatra és c szerinti költsége kisebb mint Z_{UB} , akkor Z_{UB} -nek ezt az értéket adjuk.
- (v) Ha $Z_{\max} = Z_{\text{UB}}$, akkor leállunk, mert Z_{UB} optimális.

(vi) Kiszámoljuk a szubgradienseket:

$$\begin{aligned} G_k &= L_k - \sum_{ij \in A} r_{ij}^k X_{ij} & k = 1, 2 \\ H_k &= -U_k + \sum_{ij \in A} r_{ij}^k X_{ij} & k = 1, 2 \end{aligned}$$

(vii) A T lépéshosszt a következőképp definiáljuk:

$$T = \frac{\tau(Z_{UB} - Z_{LB})}{\sum_{k=1}^2 ((G_k)^2 + (H_k)^2)}$$

ahol $0 < \tau \leq 2$ érték.

(viii) Frissítjük a multiplikátorokat:

$$\begin{aligned} s_k &= \max\{0, s_k + TG_k\} & k = 1, 2 \\ t_k &= \max\{0, t_k + TH_k\} & k = 1, 2 \end{aligned}$$

(ix) B -t eggyel megnöveljük. Ha $B < B_{\max}$, akkor akkor a (ii) lépéssel folytatjuk.

Az implementáció során a szubgradiens algoritmust még ki fogjuk egészíteni, mert $Z_{\max}$ és Z_{UB} értékén kívül mást is fel fogunk használni még. Például szükségünk lesz a $Z_{\max}$ -hoz tartozó megoldásra. Ezt az (iii) lépésben tudjuk frissíteni Z_{UB} -hez hasonlóan.

Ha a szubgradiens módszer nem optimális megoldással tér vissza (tehát nem az (v) lépésben áll le), akkor (RCSP) optimális megoldásának kiszámolására Branch and Bound algoritmust alkalmazunk, ahol a $Z_{\max}$ -hoz tartozó $C_{ij}^{\max}$ költségekkel számolunk tovább.

3.4. Redukciós tesztek

Lehetőség van a workpiece gráf méretének csökkentésére, és ezáltal az algoritmus futási idejének a csökkentésére az alábbiakban bemutatott tesztek segítségével.

A következő két tesztet a Lagrange-relaxáció után, a Branch and Bound előtt kell elvégeznünk. Jelölje D_{ij} az optimum értékét annak az (LLBP) feladatnak, ahol i a forrás, és j a nyelő. Mivel a gráfunk aciklikus (valójában annyi is elég lenne, hogy nem tartalmaz negatív kört), ezért D_{ij} egyenlő a $C^{\max}$ költségfüggvény szerinti legrövidebb ij út hosszával, melyhez még hozzá kell adnunk az alábbi kifejezés értékét: $\sum_{k=1}^2 (s_k L_k - t_k U_k)$, ahol s_k és t_k a szubgradiens módszerben a $Z_{\max}$ -hoz tartozó multiplikátorok. Célszerű ezért a szubgradiens algoritmus során a (iii) lépésben eltárolni a $Z_{OFF} = \sum_{k=1}^2 (s_k L_k - t_k U_k)$ eltolt értéket. A D_{ij} értékeket a szubgradiens algoritmus végeztével, a $C_{ij}^{\max}$ költségekkel számoljuk. Ezt ismét tehetjük úgy, hogy topologikus sorrendben haladva az i csúctól kezdve minden j csúcsra meghatározzuk a legrövidebb ij út költségét, de most ügyelnünk kell arra, hogy csak azokat a kj éleket vizsgáljuk, ahol k az i után következik a

topologikus sorrendben.

Ezek után minden olyan i csúcs törölhető a gráfból, melyre teljesül az alábbi:

$$D_{si} + D_{it} - Z_{\text{OFF}} > ZUB \quad (\text{TEST 1})$$

Ha egy i csúcson keresztül nem vezet olyan st út, melynek költsége legfeljebb ZUB , akkor azt törölhetjük a gráfból. A (TEST 1) olyan csúcsokat zár ki, melyekre ez teljesül, ugyanis egy i -n keresztül vezető st út költségére alsó becslés, ha a legolcsóbb si és it utak költségét összeadjuk. Ezek költségét pedig a Lagrange alsó korlátokkal becslve kapjuk (TEST 1)-t.

Teljesen hasonló gondolatmenet alapján minden olyan ij él törölhető a gráfból, melyre a következő fennáll:

$$D_{si} + C_{ij} + D_{jt} - Z_{\text{OFF}} > ZUB \quad (\text{TEST 2})$$

3.5. Branch and Bound

Mint korábban említettük, ha a szubgradiens módszer nem szolgáltat optimális megoldást az (RCSP) feladatra, akkor elvégezzük a leírt teszteket, és Branch and Bound algoritmust alkalmazunk az optimum megtalálására.

A T fát készítsük el a következőképpen. Vegyünk fel egy gyökért pontot, mely s -nek felel meg. Ezután minden $sv \in A$ élre a gráfból felveszünk s -hez egy gyereket, és azt mondjuk, hogy az a gyerek v -nek felel meg. Ugyanezt megcsináljuk az új pontokkal is amíg lehet. Ekkor a fa minden levele t -nek felel meg és a fában minden gyökér-leveél út egyértelműen megfelel a gráf egy st útjának. A Branch and Bound során lényegében ezen a fán végzünk mélységi bejárást, és minden pontban kiszámolunk az odavezető út optimális befejezésére egy alsó korlátot.

Ha a fa egy olyan pontjában vagyunk épp, amely a gráf egy i pontjának felel meg, akkor a fában a gyökértől idáig vezető út megfelel a gráfban egy si útnak. Jelölje ennek $C^{\max}$ szerinti költségét W , az úton eddig felhasznált erőforrást B_k ($k = 1, 2$). Jelölje R_{ij}^k azt, hogy az i csúcsból mennyi a lehető legkevesebb mennyiség a k . erőforrásból amit a j csúcsba elvihetünk. Ezek alapján, minden olyan csúcsot kiküszöbölhetünk az aktuális részfából, mely olyan j -nek felel meg, melyre $ij \in A$ és az alábbi teljesül valamely k esetén:

$$B_k + r_{ij}^k + R_{jt}^k > U_k \quad (\text{TEST 3})$$

Ez ugyanis azt jelenti, hogy az sj utat bárhogy egészítjük ki st úttá, meg fogja sérteni valamely erőforrás-korlátot.

Megemlítjük, hogy az R_{ij}^k értékeket elég egyszer kiszámolni az Branch and Bound kezdete előtt. Ez ugyanolyan technikával történhet, amelyet már párszor felvázoltunk.

Az aktuális részfában minden olyan csúcs kizárható, mely olyan j csúcsnak felel meg, melyre $ij \in A$ és az alábbi teljesül:

$$W + D_{ij} + D_{jt} - Z_{\text{OFF}} > ZUB \quad (\text{TEST 4})$$

Ebben a tesztben a (TEST 2)-höz hasonlóan alulról becsüljük az olyan st utak költségét, melyek az ij élt használják, de most úgy, hogy az utat három részre osztjuk.

A Branch and Bound során akkor léphetünk vissza egy i csúcsból, ha $Z_{LB} > Z_{UB}$ teljesül, vagy ha minden j gyereket vagy bejártuk, vagy kiküszöböltük az előző tesztek valamelyikével. Az eljárás végén megkapjuk (RCSP) optimális értékét, és az optimális utat is visszanyerhetjük.

4. A megoldás implementációja, tesztelése

Az input menetrendeket a BKV menetrendjeiből készítettük el, hogy minél valóságszerűbbek legyenek az adatok. A megoldást két különböző módon implementáltuk. Mindkét esetben egy input fájlt készítettünk számukra, mellyel az adott program dolgozhat. Ezek lényegében a workpiece gráfot tartalmazzák, a program számára megfelelő formátumban. A két implementáció közti legfontosabb különbség az árazási probléma megoldásában van. Az *I. Algoritmus* esetén az árazási problémát egy általános MIP megoldóval oldottuk meg, míg a *II. Algoritmus* esetén az előző fejezetben bemutatott Branch and Bound algoritmussal oldottuk meg ezt a problémát. Mindkét esetben az oszlopgenerálás végeztével, a generált mátrix-szal oldottuk meg az (1) IP-feladatot.

A megoldások tesztelését és a két megoldás összehasonlítását 4 input fájl alapján végeztük el. A 1. táblázat tartalmazza, hogy az egyes esetekben mekkora volt a workpiece gráf mérete. A *II. Algoritmus*nál megvizsgáltuk azt is, hogy a 3.5 szakaszban leírt tesztek mennyire tették hatékonyabbá a Branch and Bound algoritmust.

A tesztelést 1.86 GHz-es Intel Pentium Dual CPU T2390 processzorral; Windows 7 operációs rendszeren végeztük.

4.1. I. Algoritmus

Az *I. Algoritmus*t az Xpress Mosel programozási nyelven implementáltuk. Felépítettük a (2)-es LP-feladatot, majd elkészítettük az oszlopgenerálás kódját. Az árazási feladat megoldása itt MIP solverrel történt.

4.2. II. Algoritmus

A *II. Algoritmus*t C++ nyelven implementáltuk, a lemon könyvtár¹, és az XPRESS C++ programkönyvtár felhasználásával. Utóbbi segítségével felépítettük a (2)-es LP-feladatot, és implementáltuk az oszlopgenerálás algoritmusát. Az árazási feladat megoldása a 3. szakaszban bemutatott módszerekkel történt.

4.3. Tesztelés kisebb inputon

Kisebb inputon 15-30 fordulóból álló menetrendet értünk. Az A Input esetében ez az érték 19. A menetrend 3 villamos délelőtti fordulót tartalmazza, ezért ebben az esetben $L = 3$ (óra), $U = 4$ (óra), $a = 15$ (perc), $e_l = 15$ (perc), $e_u = 50$ (perc) korlátokkal készítettük el a workpiece gráfot. A futtatás során legfeljebb 200 új oszlop generálást engedélyeztük.

Mindkét algoritmust ötször-ötször lefuttatva számoltuk ki az átlagos futási időket, melyek az oszlopgenerálás időtartamát mérték. A 2. táblázat tartalmazza az átlagos futási időket, a megoldás

¹<http://lemon.cs.elte.hu/trac/lemon>

során generált új oszlopok számát, a megtalált LP-optimumot illetve az egészértékű egészértékű optimum értéket.

A B Input 4 busz 30 fordulóját tartalmazza. Ebben az esetben is az előző a korlátokkal készítettük el workpiece gráfot. A két fenti, és további tesztek alapján azt a következtetést vonhatjuk le, hogy a II. Algoritmus esetén rövidebb ideig tart az oszlopgenerálás folyamata, akár 7-8-szor kevesebb idő alatt eléri az LP-optimumot, mint az I. Algoritmus. Az új oszlopok számában jelentős különbség nincsen. A B Input esetén viszont figyeljük meg, hogy a II. Algoritmus eléri az LP-optimumot, de a leggenerált oszlopokból nem tud ennyi vezetővel megengedett ütemezést készíteni. Ez természetes jelenség, hiszen az LP-optimum és az IP-optimum nem feltétlenül esik egybe. A két példán ugyan az I. Algoritmusnál megegyezik az LP-optimum és az egészértékű optimum, de a nagyobb példákon majd látni fogjuk, hogy itt sem fog egybe esni mindig a két érték. A 2. a) ábrán láthatóak a C Input fordulói. A b) illetve c) ábrákon a II. illetve az I. Algoritmus által adott megoldások szerepelnek.

4.4. Tesztelés nagyobb inputon

A C Input 6 villamos napi menetét tartalmazza, mely 86 fordulót jelent. Az $L = 7$ (óra), $U = 9$ (óra), $a = 30$ (perc), $e_l = 30$ (perc), $e_u = 60$ (perc) korlátokkal elkészített workpiece gráffal futtattuk az algoritmusokat. A futtatás során legfeljebb 400 új oszlop generálást engedélyeztük. A futási eredményeket a 4. táblázat tartalmazza.

A D Input 6 busz napi fordulóit tartalmazza. Az előző korlátokkal elkészített workpiece gráfnak most már több mint 200 csúcsa és közel 1000 éle van. A futási eredmények a 5. táblázatban találhatók.

Mindkét összehasonlításban azt kaptuk, hogy az I. Algoritmus kevesebb oszlop generálásával találta meg az LP-optimumot, de a II. Algoritmus jelentősen rövidebb idő alatt. Ez további, hasonló méretű teszteken is megfigyelhető volt.

A 3. ábrán láthatóak a C Input fordulói. A 4. illetve 5. ábrán pedig az I. illetve a II. Algoritmus által adott megoldások láthatók.

4.5. A Branch and Bound algoritmusban szereplő tesztek

A 3.5 szakaszban két tesztet is készítettünk a Branch and Bound algoritmus hatékonyabbá tételének érdekében. A (TEST 3) és (TEST 4) tesztek kizártak bizonyos csúcsokat, és ezáltal a Branch and Bound során egyes részfákat kihagyhattunk a vizsgálat alól. Most megnézzük mennyire szükségesek ezek a tesztek.

Lefuttattuk a II. Algoritmust az egyes inputokon úgy, hogy valamely tesztet kikapcsoltuk. Megnéztük, hogy az oszlopgenerálás mennyi idő alatt futott le, továbbá azt, hogy az oszlopgenerálás során mennyi volt a legtöbb csúcs, amit egy Branch and Bound során bejártunk, illetve az összes Branch and Bound során hány csúcsot jártunk be. Megjegyezzük, hogy a tesztek be illetve

kikapcsolása ugyanazt a futást eredményezik, hiszen pontosan azért hagytunk ki egy pontot a Branch and Bound során, mivel az a részfa nem szolgáltatott új optimális megoldást. A futási eredményeket a 6. táblázat tartalmazza. A négy nagyobb sor a négy inputnak felel meg. Ezeken belül 3-3 sor szerepel. Az elsőben a futási időket tüntettük fel (másodpercben), a másodikban azt, hogy mennyi volt a legtöbb csúcs amit egy Branch and Bound során bejártunk. Végül a harmadik sorba az összes Branch and Bound által bejárt csúcsok összegét írtuk. Az oszlopok azoknak a futásoknak felelnek meg, amikor az egyes tesztek be voltak kapcsolva. Például a $\{3, 4\}$ fejlécű oszlop annak a futtatásnak felel meg, ahol mind a (TEST 3), mind a (TEST 4) be volt kapcsolva (ez az alapértelmezett).

A futási időben a kis példák (A,B) nem jelentkezett nagy eltérés, azonban a bejárt csúcsok számában már itt is komoly különbségek mutatkoztak. A B Inputnál például volt olyan árazási probléma, amikor a Branch and Bound legalább hétszer annyi csúcsot járt be, ha (TEST 3) ki volt kapcsolva. Az input méretének növekedésével az előző jelenség még drasztikusabb lett. A C és D Inputnál (TEST 4) hiányában még lefutott az algoritmus, de a D Inputnál ekkor 1.077.458 helyett 164.734.731 csúcsot vizsgáltunk át, és ez körülbelül hét és félszer több időt vett igénybe, mintha minden teszt be lett volna kapcsolva. Ha (TEST 3)-t kikapcsoltuk, akkor pedig körülbelül a 174. oszlop generálása után kifogytunk a memóriából.

Itt egyértelműen levonhatjuk a következtetést, miszerint a hasonló tesztek elengedhetetlenek egy hatékony Branch and Bound algoritmus készítéséhez.

4.6. Konklúzió

A II. Algoritmust tehát abban a reményben készítettük, hogy az árazási feladatra hatékonyabb, gyorsabb megoldást adjunk. A hatékonyságot a Branch and Bound során szerettük volna elérni. Ugyanis a 3.5. szakaszban bemutatott módszer esetén egy-egy Branch and Bound csúcsnál már nem végeztük összetett számításokat, hiszen itt a szubgradiens algoritmus által kiszámított értékekkel végeztünk el teszteket. Ezzel szemben egy IP megoldása során alkalmazott Branch and Bound módszernél, a csúcsokban egy-egy LP feladatot kell megoldani.

A fenti eredmények azt mutatják, hogy megérte az összetettebb algoritmust elkészíteni. Természetesen most csak 4 input tesztelését írtuk le, de megjegyezzük, hogy az algoritmusok futása nagyban függ a gráf struktúrájától. A 3. ábrán látható struktúra (a fordulók közel azonos menet-idejük, és periódikusság is felfedezhető az indulási időkben) pedig jellemző azokra a workpiece gráfokra, melyeket az ugyanazon útvonalat teljesítő járatokra készítettünk el.

5. Összefoglalás, kitekintés

A dolgozatban ismertettünk egy vezető ütemezési problémát. Fő célként az tűztük ki, hogy a probléma megoldására programot készítsünk. Ehhez a problémát egészértékű programozási feladatként fogalmaztuk meg. A feladat mérete miatt a megoldást oszlopgenerálás segítségével végeztük el. Az árazási feladat egy erőforrás korlátos legrövidebb út keresésére vezetett vissza, melyet első esetben Branch and Bound algoritmussal kerestünk meg, másodszor pedig egy általános MIP megoldó segítségével. Különböző adatokon összehasonlítottuk a két implementált algoritmus futásidejét, és megoldásait. Végül a második algoritmusnál megvizsgáltuk, hogy a Branch and Bound során használt bizonyos tesztek mennyire növelik az algoritmus hatékonyságát.

Az általunk vizsgán feladat egy nagyon széleskörű problémakör egy kis szelete. A feladat számos továbbfejlesztést, kutatást kínál. Például láthattuk, hogy a modell megoldásában előfordult az, hogy egy fordulóra két vezetőt is beosztottunk. Felvetődhet az ötlet, hogy esetleg túlórákat megengedve a vezetőknek ez kiküszöbölhető. Egy másik természetesen felvetés lehet az, hogy olyan beosztásokat készítsünk, ahol a vezetőknek lehetőleg minél kevesebbszer kelljen járművet váltaniuk. Érdekes kérdés, hogy vajon bele lehet-e építeni ezt a feltételt valahogy a célfüggvénybe. Láthatjuk tehát, hogy számos új irányban haladhatunk tovább, ezen az igazán sokrétű területen.

Függelék

Táblázatok

	Fordulók száma	Csúcsok száma	Élek száma
A Input	19	39	100
B Input	30	61	185
C Input	86	173	728
D Input	105	211	963

1. táblázat. Az egyes inputokhoz tartozó workpiece gráfok adatai

	Átlagos futási idő	Új oszlopok száma	OPT _{LP}	OPT _{IP}
I. Algoritmus	0,8 másodperc	19	7	7
II. Algoritmus	0,1 másodperc	17	7	7

2. táblázat. Futási eredmények (A Input)

	Átlagos futási idő	Új oszlopok száma	OPT _{LP}	OPT _{IP}
I. Algoritmus	2,1 másodperc	36	10	10
II. Algoritmus	0,3 másodperc	38	10	11

3. táblázat. Futási eredmények (B Input)

	Átlagos futási idő	Új oszlopok száma	OPT _{LP}	OPT _{IP}
I. Algoritmus	131,2 másodperc	124	11	13
II. Algoritmus	36,6 másodperc	170	11	13

4. táblázat. Futási eredmények (C Input)

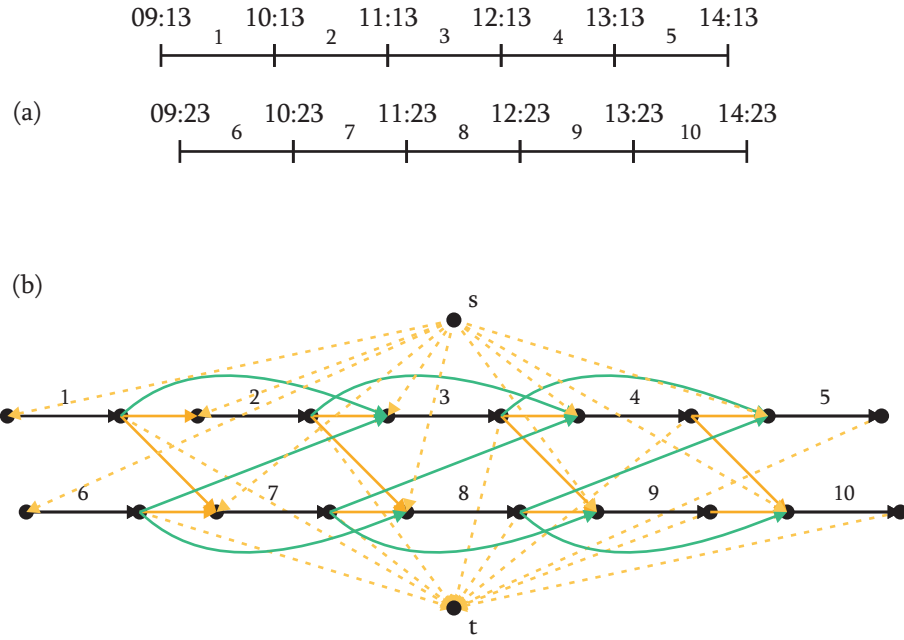
	Átlagos futási idő	Új oszlopok száma	OPT _{LP}	OPT _{IP}
I. Algoritmus	794,5 másodperc	310	13,125	14
II. Algoritmus	165,2 másodperc	374	13,125	14

5. táblázat. Futási eredmények (D Input)

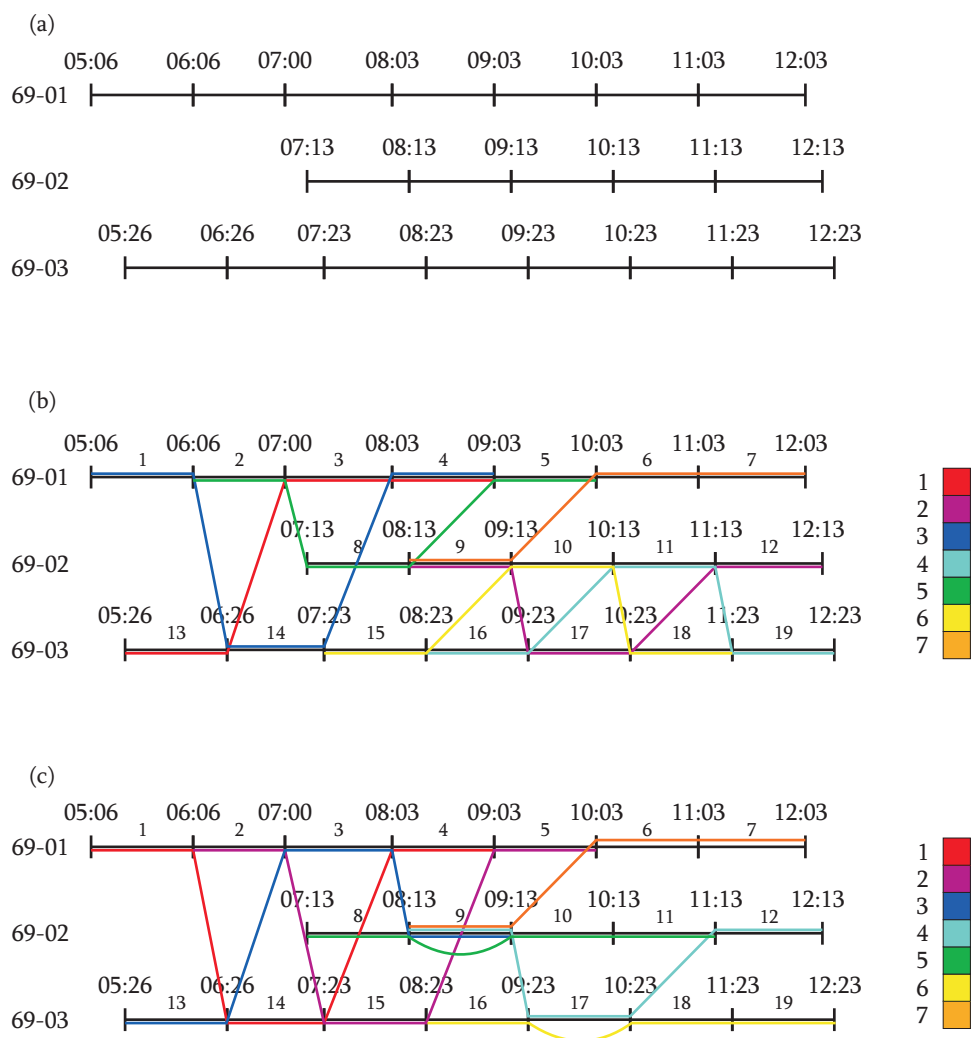
		$\{3, 4\}$	$\{3\}$	$\{4\}$	$\emptyset$
A Input	Futási idő (másodperc)	0,11	0,12	0,19	0,12
	Max csúcs (darab)	462	539	1 677	2 607
	Összes csúcs (darab)	3 102	5 688	9 273	24 638
B Input	Futási idő	0,26	0,27	0,39	0,25
	Max csúcs	880	1 039	5 571	13 515
	Összes csúcs	6 440	22 551	22 780	97 632
C Input	Futási idő	36,6	85,8	NA	NA
	Max csúcs	398 509	1 052 606	NA	NA
	Összes csúcs	1 444 458	55 941 925	NA	NA
D Input	Futási idő	39,6	295,2	NA	NA
	Max csúcs	214 651	2 049 527	NA	NA
	Összes csúcs	1 077 458	164 734 731	NA	NA

6. táblázat. Futási eredmények (Tesztek)

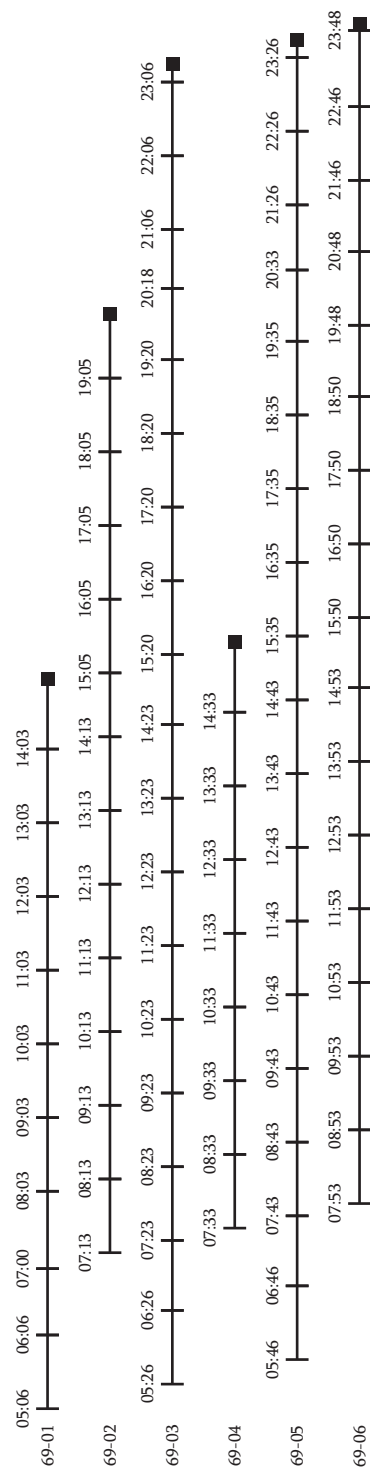
Ábrák



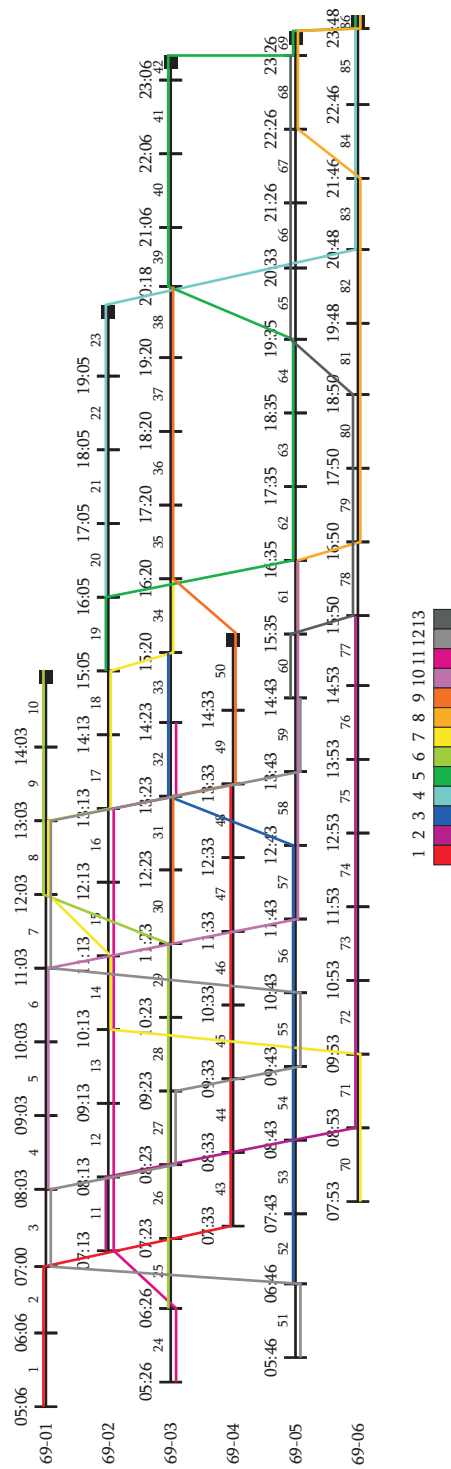
1. ábra. A workpiece gráf készítése 10 fordulóból, $a = 30$, $e_l = 30$, $e_u = 60$ korlátokkal. A kékes-zöld élek az ebéd-éleket jelölik, a vastag narancssárgák pedig az átszállásokat.



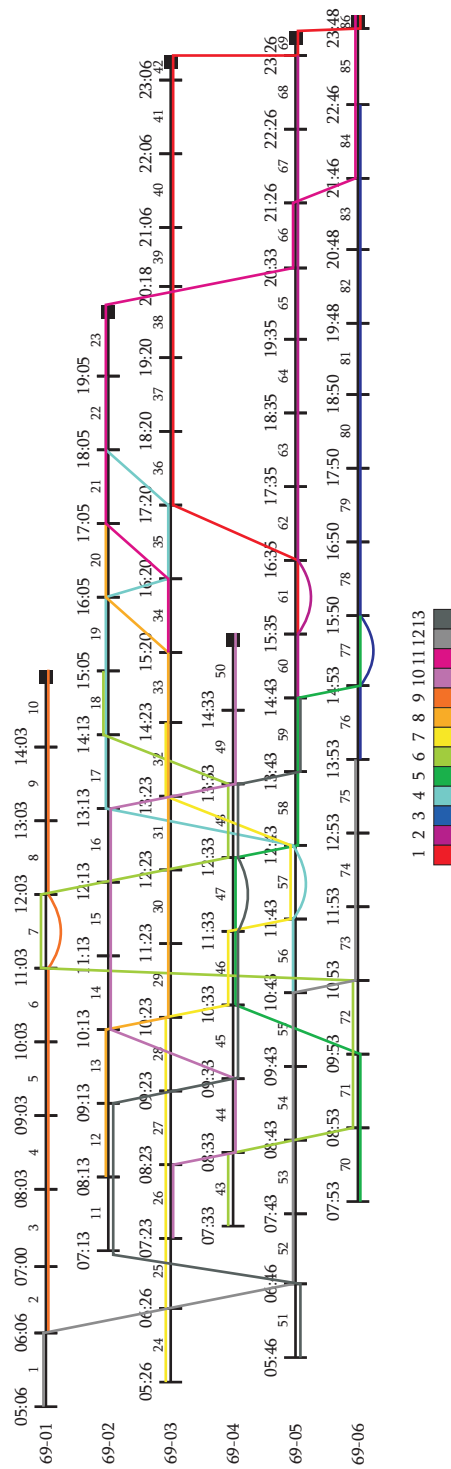
2. ábra. a) Az A Inputban szereplő fordulók. A b) illetve c) ábrán a II. Algoritmus illetve az I. Algoritmus által adott beosztások láthatók 7 vezetővel.



3. ábra. A C Inputban szereplő fordulók.



4. ábra. Az I. Algoritmus által adott beosztások 13 vezetővel a C Inputra.



5. ábra. A II. Algoritmus által adott beosztások 13 vezetővel a C Inputra.

Hivatkozások

- [1] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, P. H. Vance: *Branch and price: Column Generation for Solving Huge Integer Programs*, Operations Research, pp 316-329 (1998)
- [2] J. E. Beasley and N. Christofides: *An Algorithm for the Resource Constrained Shortest Path Problem*, Networks, vol.19, no.3, pp 379-394 (1988)
- [3] S. Fores: *A column generation approaches to bus driver scheduling*, Ph.D, Thesis, University of Leeds (1996)
- [4] S. Fores, L. Proll, A. Wren: *An Improved ILP System for Driver Scheduling*, Report 97.21, University of Leeds (2007)
- [5] G. Handler, I. Zang: *A dual algorithm for the constrained shortest path problem*, Networks, 10, pp 293-310 (1980)
- [6] S. Irnich, G. Desaulniers: *Shortest Path Problems with Resource Constraints*, Les Cahiers du GERAD, HEC Montréal, G-2004-11 (2004)
- [7] J. Kjerrström: *The resource constrained shortest path problem*, Resources (2001)
- [8] R.S.K. Kwan, A.S.K. Kwan, A.S.K. Wren: *Evolutionary Driver Scheduling with Relief Chains*, Evolutionary Computation, 9, pp 445-460 (2001)
- [9] Y. Shen, R.S.K. Kwan: *Tabu Search for Driver Scheduling*, Computer-Aided Scheduling of Public Transport, pp 121-135 (2001)
- [10] A. Wren, J. Rousseau: *Bus Driver Scheduling - An Overview*, Report 93.31, University of Leeds (1993)
- [11] A. Wren, S. Fores, A.S.K. Kwan, R.S.K. Kwan, M.E. Parker, L. Proll: *A flexible system for scheduling drivers*, Journal of Scheduling, 6, pp 437, (2003)