Eötvös Loránd Tudományegyetem
Természettudományi kar

Horváth Markó

Jármű és vezető ütemezés

Szakdolgozat
Alkalmazott Matematikus MSc
Operációkutatás szakirány

témavezető:
Dr. Kis Tamás
Operációkutatási Tanszék

2013. Május

Köszönetnyilvánítás

Ezúton is szeretnék köszönetet mondani témavezetőmnek, Kis Tamásnak azért a rengeteg segítségért, tanácsért amit a TDK, a szakdolgozat, és tanulmányaim kapcsán kaptam tőle. Köszönöm, hogy mindig tudott időt szakítani rám, kérdéseimmel bármikor felkereshettem, leveleimre szinte azonnal válaszolt.
Szeretném megköszönni barátaimnak, hogy megértették ha néha fontosabbak voltak számomra a tanulmányaim. Köszönöm csoporttársaimnak, hogy tudtuk egymást segíteni az együtt töltött évek alatt. De legfőképpen szüleimnek, családomnak tartozom hálával, amiért szüntelenül támogattak tanulmányaim során.

Tartalomjegyzék

Bevezető **iii**

1. Kombinatorikus optimalizálási feladatok és módszerek **1**

1.1. Kombinatorikus optimalizálási feladatok 2

1.1.1. Minimális költségű folyam, áram 2

1.1.2. Legrövidebb utak 3

1.1.3. Többtermékes folyam 3

1.1.4. Halmaz partíció és halmaz fedés 4

1.2. Kombinatorikus optimalizálási módszerek 5

1.2.1. Lagrange relaxáció 5

1.2.2. Szubgradiens algoritmus 6

1.2.3. Oszlopgenerálás 8

1.2.4. Lagrange relaxációval kombinált oszlopgenerálás 8

1.3. Erőforrás korlátos legrövidebb út 10

1.3.1. Az RCSP formális felírása 10

1.3.2. Megoldási módszerek 10

1.3.3. Beasley és Christofides megoldási módszere 11

2. Jármű és vezető ütemezés **16**

2.1. Jármű ütemezés 16

2.1.1. A jármű ütemezés feladata 16

2.1.2. Modellezési eszközök 17

2.1.3. SDVSP 21

2.1.4. MDVSP 21

2.2. Vezető ütemezés 22

2.2.1. A vezető ütemezés feladata 22

2.2.2. CSP 22

2.3. Egy vezető ütemezési probléma 23

2.3.1. A probléma ismertetése 23

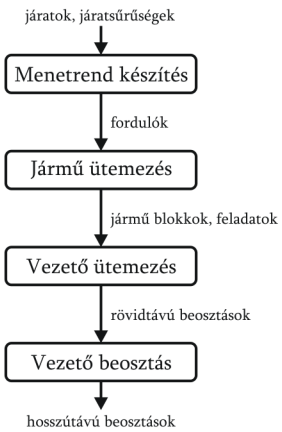
2.3.2. A probléma modellezése 23
2.3.3. A probléma megoldása 24
2.3.4. Implementáció 25
2.3.5. Tesztelés, eredmények 26

3. Integrált jármű és vezető ütemezés 30
3.1. SDVCSP 32
3.1.1. Modellezés 32
3.1.2. Megoldási eljárás 33
3.1.3. A fő feladat 34
3.1.4. Árazási feladat 34
3.1.5. Megengedett megoldás előállítása 35
3.1.6. Futási eredményeink 35
3.2. MDVCSP 40
3.2.1. Modellezés 40
3.2.2. Megoldási eljárás 41

Összefoglalás 43
Hivatkozások I

Bevezető

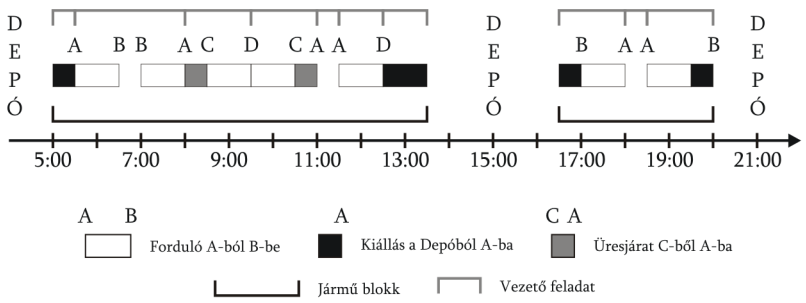
A jármű és a vezető ütemezés a tömegközlekedésbeli optimalizálás legjelentősebb feladatai közé tartozik, hiszen a járművek és a járművezetők alkotják a közlekedési vállalatok fő eszköztárát. Az utóbbi évtizedekben egyre nagyobb érdeklődés mutatkozott a tömegközlekedési optimalizálásra, és főként a számítógéppel történő megoldásokra. Ennek egyik legfőbb oka, hogy a közlekedési vállalatok számára nagy költségekkel jár egy jármű használata/karbantartása illetve a vezetők bérezése. Másrészt, mint látni fogjuk, a vezetők beosztására vonatkozó feltételek kezelése egyre bonyolultabb feladatokat eredményez, melyek manuálisan történő megoldása reménytelenné válik.
Egy közlekedési vállalat tervezési folyamata általában négy fázisból áll. A tervezési folyamatot a 1. ábra szemlélteti.



1. ábra. Tervezési folyamat

Mindenekelőtt meghatározzuk, hogy milyen járatokat szeretnénk létrehozni, és milyen járatsűrűséggel. Például a forgalmasabb csomópontokon át célszerű több járatot létrehozni, nagyobb járatsűrűséggel. Ezekre alapozva készül el a fordulókból álló menetrend úgy, hogy

minden egyes fordulóra adott az indulás/érkezési idő illetve a kezdő-/végállomás.
A tervezés második fázisa a jármű ütemezés, melyben a menetrend készítés során meghatározott fordulókhoz kell járműveket rendelnünk. A járművek különböző depókhoz tartozhatnak. Egy depó alatt ugyanahhoz a garázshoz tartozó, és ugyanolyan típusú (alacsonypadlós, csuklós, stb.) járműveket értünk. Minden egyes fordulóra adott, hogy mely depóból teljesíthető (például a kórházak közelében célszerű alacsonypadlós járműveket használni). A jármű ütemezés során különböző költségeket kell figyelembe venni. Tipikus példa, hogy minden jármű használata maga után von egy fix költséget, és a megtett kilométerek alapján meghatározott működési költséget. Az ütemezés végeztével minden járműre kapunk egy menettervet, amely a depóból indul és ott ér véget. A menettervre egy példát a 2. ábrán láthatunk. Lehetséges, hogy a jármű többször is visszatér a depóba. A jármű menettervének egy depótól-depóig tartó részét jármű blokknak nevezzük. A járművezetők nem csak a depóban, hanem bizonyos állomásokon (átszállási pontokon) is megkezdhetik/befejezhetik a szolgálatukat. Egy jármű blokkon belül több ilyen átszállási lehetőség is adott lehet a vezető számára. Két egymást követő átszállási hely közötti részt vezető feladatnak (röviden: feladatnak) nevezünk. A 2. ábrán a depón kívül csak az A illetve D állomásokon engedjük meg a vezetők átszállását.



2. ábra. Egy jármű menetterve

A harmadik fázisban a vezetőket kell beosztanunk a jármű ütemezés során kapott feladatokra. Egy adott vezetőhöz rendelt feladatok összességét a vezető beosztásának nevezünk. A beosztások megengedettségére különböző feltételek adottak, melyek függhetnek a közlekedési vállalattól, de az adott régió munkatörvényétől is. A legtöbb esetben egy beosztás több (többnyire kettő vagy három) munkarészből kell, hogy álljon. Általában egy munkarész alatt a vezető ugyanazt a járművet vezeti hosszabb szünet nélkül. Tipikus feltételek, hogy egy-egy munkarész, a munkarészek közötti szünet, illetve az egész beosztás időtartamára alsó-felső korlátok adottak.

Az első három fázis során általában egy adott napot veszünk figyelembe. Például a menetrendeknél megkülönböztethetünk munkanapokat és munkaszüneti napokat, iskola előadási napokat, stb. Gyakori, hogy hétfőtől-csütörtökig illeve pénteken eltérő menetrendeket kell alkalmazni. A negyedik fázisban azonban hosszabb időtávra tervezünk. Itt a vezetőknek a heti/havi beosztását kell elkészíteni a vezető ütemezés során megkapott napi beosztásokból. A dolgozatban csak a jármű és vezető ütemezési részekkel fogunk foglalkozni. Az 1. Fejezetben válogatott kombinatorikus optimalizálási feladatokat és módszereket tekintünk át, melyekkel a későbbi fejezetekben találkozni fogunk. Külön foglalkozunk az erőforrás korlátos legrövidebb út feladattal és megoldásával, mert ez a probléma számos jármű és vezető ütemezési feladatnál felmerül.

A 2. Fejezetben a szekvenciális jármű és vezető ütemezéssel foglalkozunk. Ismertetjük a problémákat, és rövid irodalmi áttekintést is nyújtunk a megoldási módszerekről. A fejezet utolsó szakaszában bemutatunk egy speciális vezető ütemezési feladatot, melyhez két implementációt is készítettünk. A szakasz hátralevő részében a tesztelési eredményeinket mutatjuk be.

A 3. Fejezetben az integrált jármű és vezető ütemezésről lesz szó. Itt is ismertetjük a különböző problémákat, megoldási módszereket. Az egy-depós ütemezéshez elkészített implementációnk futási eredményeiről is itt számolunk be.

1. fejezet

Kombinatorikus optimalizálási feladatok és módszerek

Az alábbi fejezetben olyan kombinatorikus optimalizálási feladatokat és módszereket ismertetünk, melyek a későbbi fejezetekben előfordulnak, illetve alkalmazni fogunk.
Az 1.1 szakaszban a teljesség igénye nélkül bemutatjuk a *minimális költségű folyam/áram* és a *többtermékes folyam* feladatokat. Szó lesz még *legrövidebb utakról*, és ismertetjük a *halmaz fedés/partíció* feladatokat is. A feladatokat olyan formában mutatjuk be, amilyenre a későbbiekben szükségünk lesz.
Az 1.2 szakaszban egy átfogó ismertetőt nyújtunk a *Lagrange relaxációról*, a *szubgradiens algoritmusról*, és az *oszlopgenerálásról*. Az oszlopgenerálás Lagrange relaxációval kombinált változatát is bemutatjuk.
Az *erőforrás korlátos legrövidebb út* feladat a 2. Fejezet végén kulcsszerepet játszik, ezért a feladat és megoldási eljárásának ismertetésére egy külön szakaszt áldozunk.

1.1. Kombinatorikus optimalizálási feladatok

Jelölés: a következőkben $D = (V, A)$ irányított gráfot jelöl.

1.1.1. Minimális költségű folyam, áram

Tegyük fel, hogy van egy $D = (V, A)$ hálózatunk, és annak két kijelölt csúcsa: s és t. Folyamon olyan $x : A \to \mathbb{R}_\oplus$ függvényt értünk, hogy minden s és t-től különböző j csúcsra teljesül a *megmaradási szabály*:

$$\sum_{i:ij\in A} x(ij) = \sum_{i:ji\in A} x(ji)$$

azaz tömörebb jelöléssel: $\varrho_x(j) = \delta_x(j)$. Egy x folyam *nagyságán* a $\delta_x(s) - \varrho_x(s)$ értéket értjük.

Ha a hálózatunkon adottak az $l : A \to \mathbb{R}_\oplus$, $u : A \to \mathbb{R}_\oplus$ alsó-felső kapacitás függvények ($l \leq u$), akkor egy x folyamról azt mondjuk, hogy *megengedett*, ha minden $ij \in A$ élre $l(ij) \leq x(ij) \leq u(ij)$ teljesül.

Ha a hálózatunkon adott egy $c : A \to \mathbb{R}$ költségfüggvény is, akkor beszélhetünk egy x folyam $\tilde{c}(x)$ költségéről:

$$\tilde{c}(x) = \sum_{ij\in A} c(ij)x(ij)$$

A folyamokkal kapcsolatos egyik legalapvetőbb feladat egy minimális költségű megengedett folyam megkeresése. A problémát a következő lineáris program segítségével is megfogalmazhatjuk:

$$\begin{aligned} \min \sum_{ij\in A} c_{ij}x_{ij} & \\ \sum_{i:ij\in A} x_{ij} = \sum_{i:ji\in A} x_{ji} & \qquad \forall\, j \in V - \{s,t\} \\ l_{ij} \leq x_{ij} \leq u_{ij} & \qquad \forall\, ij \in A \end{aligned}$$

Rendkívül fontos kiemelnünk a probléma egészértékűségi tulajdonságát [13]: *ha az adott korlátok (l és u) egészértékűek, akkor az optimális x folyam is választható egészértékűnek.*

A minimális költségű folyam megkeresésére több polinomális algoritmus is rendelkezésünkre áll (például Kapacitás-skálázás, Hálózati Szimplex algoritmus).

A folyammal rokon fogalom az *áram*. Áramon olyan $x : A \to \mathbb{R}_\oplus$ függvényt értünk, melynél minden csúcsra teljesül a megmaradási szabály. Hasonlóan értelmezzük a *megengedett áram* fogalmát, és a minimális költségű megengedett áram feladatát, mely szintén rendelkezik az egészértékűségi tulajdonsággal. Az előző algoritmusokkal (Kapacitás-skálázás, Hálózati Szimplex algoritmus) a minimális költségű megengedett áram feladat is polinom időben megoldható.

1.1.2. Legrövidebb utak

A legrövidebb út probléma során egy $D = (V, A)$ hálózat kijelölt s és t pontjai között szeretnénk legrövidebb utat találni egy $c : A \to \mathbb{R}$ távolságfüggvény szerint.
A feladat megfogalmazható egy 1-nagyságú legolcsóbb folyam kereséseként is. A további fejezetekben viszont kizárólag aciklikus gráfban fogunk legolcsóbb utat keresni, és szükségünk lesz minden v csúcs esetén a legrövidebb sv út $d(v)$ hosszára, ezért egy másik eljárást vázolunk (1.1. Algoritmus).

Algoritmus 1.1 Legrövidebb utak aciklikus irányított gráfban egy s forrásból

1: *Topologikus sorrend*
Készítsük el az aciklikus D gráf topologikus sorrendjét.

2: *Inicializálás*
Legyen $d(s) = 0$, és $d(v) = \infty$ minden $v \in V - s$ csúcsra.

3: *Legrövidebb utak*
A topologikus sorrendben haladva egy adott v csúcsra $d(v)$-t a következőképpen számoljuk:
$d(v) = \min_u \{d(u) + c(uv) \mid uv \in A\}$

1.1.3. Többtermékes folyam

A folyam problémára tekinthetünk úgy, mintha egy termékből adott (legtöbb) mennyiséget szeretnénk eljuttatni s-ből t-be, bizonyos kapacitásokat teljesítve, esetleges költségeket figyelembe véve. A többtermékes folyam problémát úgy szemléltethetjük, mint ha K különböző terméket szeretnénk s-ből t-be szállítani, és minden egyes k termékre külön-külön adottak a c^k költség és l^k, u^k kapacitás függvények. Továbbá adottak még az L és U kapacitás függvények is, melyek minden élen az összkapacitást korlátozzák. Az egészértékű többtermékes folyam az alábbi feladatként írható fel:

$$\begin{aligned}
\min \sum_{k=1}^{K} \sum_{ij \in A} c_{ij}^k x_{ij}^k & \\
\sum_{i:ij \in A} x_{ij}^k = \sum_{i:ji \in A} x_{ji}^k & \qquad \forall\, j \in V - \{s,t\},\ \forall\, k = 1, \dots, K \\
l_{ij}^k \le x_{ij}^k \le u_{ij}^k & \qquad \forall\, ij \in A,\ \forall\, k = 1, \dots, K \\
L_{ij} \le \sum_{k=1}^{K} x_{ij}^k \le U_{ij} & \qquad \forall\, ij \in A \\
x_{ij}^k \in \mathbb{N} & \qquad \forall\, ij \in A
\end{aligned}$$

A többtermékes folyamra általában nem teljesül az egészértékűségi feltétel, tehát az $x_{ij}^k \in \mathbb{N}$ feltételt elhagyva nem feltétlenül kapunk egészértékű megoldást (még akkor sem, ha minden

kapacitás egész is). Az egészértékű többtermékes folyam NP-teljes, már két termék esetén is [21].

1.1.4. Halmaz partíció és halmaz fedés

Tegyük fel, hogy adott egy véges M halmazunk, és részhalmazain egy $c : 2^M \to \mathbb{R}$ költségfüggvény. A célunk az M halmaz olyan $\mathcal{P}$ partícióját meghatározni, melynek $\tilde{c}(\mathcal{P})$ összköltsége minimális.

$$\tilde{c}(\mathcal{P}) = \sum_{P \in \mathcal{P}} c(P)$$

A halmaz partíció feladatot a következő egészértékű feladattal írhatjuk fel:

$$\min \sum_{j \in N} c_j x_j \tag{1.1}$$

$$\sum_{j \in N} a_{ij} x_j = 1 \qquad \forall\, i \in M \tag{1.2}$$

$$x_j \in \{0, 1\} \qquad \forall\, j \in N \tag{1.3}$$

ahol N jelöli M részhalmazainak halmazát, x_j pedig a $j \in N$ részhalmazhoz tartozó döntési változó.

$$a_{ij} = \begin{cases} 1 & \text{ha a } j \text{ részhalmaz tartalmazza az } i \text{ elemet} \\ 0 & \text{különben} \end{cases}$$

$$x_j = \begin{cases} 1 & \text{ha a } j \text{ halmazt kiválasztjuk} \\ 0 & \text{különben} \end{cases}$$

Az (1.2) feltétel biztosítja azt, hogy M minden eleme pontosan egy választott részhalmazhoz tartozzon.

A halmaz fedési feladatnál is fednünk kell az M minden elemét, de nem követeljük meg azt, hogy a fedő halmazok partíciót adjanak. A halmaz fedési feladatot úgy írhatjuk fel egészértékű feladatként, hogy az (1.2) feltételben '$\geq$' jelet írunk. Megjegyezzük, hogy a halmaz partíció feladatban, illetve a halmaz fedési feladatban pozitív költségfüggvény mellett elegendő $x_j \in \mathbb{N}$ megkötéseket tennünk.

Mindkét feladat NP-nehéz [21].

1.2. Kombinatorikus optimalizálási módszerek

1.2.1. Lagrange relaxáció

Tekintsük a következő egészértékű feladatot:

$$Z_{\mathrm{IP}} = \min \sum_{j \in N} c_j x_j \tag{1.4}$$

$$\sum_{j \in N} a_{ij} x_j = d_i \qquad \forall\, i \in M_1 \tag{1.5}$$

$$\sum_{j \in N} b_{ij} x_j = e_i \qquad \forall\, i \in M_2 \tag{1.6}$$

$$x_j \in \mathbb{Z}_+ \qquad \forall\, j \in N \tag{1.7}$$

Tegyük fel, hogy az (1.5) feltételek komplikált feltételek abban az értelemben, hogy őket kihagyva a feladat könnyen megoldható lenne.

A *Lagrange relaxáció* alapötlete az, hogy a nehéz feltételeket kihagyjuk a feltételek közül, de nem-teljesülésüket a célfüggvényben büntetjük. A komplikált feltételeket beépítve a célfüggvénybe (*relaxálva, dualizálva*) kapjuk a *Lagrange feladatot:*

$$Z_{\mathrm{LR}}(\lambda) = \min \sum_{j \in N} c_j x_j + \sum_{i \in M_1} \lambda_i \left(d_i - \sum_{j \in N} a_{ij} x_j \right)$$

$$\sum_{j \in N} b_{ij} x_j = e_i \qquad \forall\, i \in M_2$$

$$x_j \in \mathbb{Z}_+ \qquad \forall\, j \in N$$

ahol $\lambda = (\lambda_i)_{i \in M_1}$ a komplikált feltételekhez tartozó *Lagrange szorzók* (*multiplikátorok*). Egy $j \in N$ oszlop *redukált költsége* (*Lagrange költsége*) $\bar{c}_j$:

$$\bar{c}_j = c_j - \sum_{i \in M_1} a_{ij} \lambda_j$$

A megengedett megoldások halmazát jelölje X, és legyen $\tilde{X}$ a relaxált poliéder:

$$X = \left\{ x \in \mathbb{Z}_+^{|N|} \;\middle|\; \sum_{j \in N} b_{ij} x_j = e_i \;\forall\, i \in M_2 \right\}$$

$$\tilde{X} = \left\{ x \in \mathbb{R}_+^{|N|} \;\middle|\; \sum_{j \in N} b_{ij} x_j = e_i \;\forall\, i \in M_2 \right\}$$

A Lagrange feladat az alábbi formába írható:

$$Z_{\mathrm{LR}}(\lambda) = \min \left\{ \sum_{j \in N} \bar{c}_{ij} x_j + \sum_{i \in M_1} \lambda_i d_i \;\middle|\; x \in X \right\} \tag{1.8}$$

Ha egyenlőtlenséges feltételeket dualizálunk, akkor a megfelelő multiplikátorokra előjel megkötést teszünk. Például a $\sum_{j \in N} a_{ij} x_j \geq d_i$ feltétel dualizálása esetén feltesszük, hogy $\lambda_i \geq 0$. Könnyen látható, hogy tetszőlegesen rögzített λ mellett $Z_{\mathrm{LR}}(\lambda)$ alsó korlátja az eredeti feladat optimumának, azaz: $Z_{\mathrm{LR}}(\lambda) \leq Z_{\mathrm{IP}}$ teljesül. A legjobb alsó korlát megkeresését *Lagrange duális feladatnak* nevezzük:

$$Z_{\mathrm{LD}} = \max_{\lambda} Z_{\mathrm{LR}}(\lambda)$$

Megmutatható [22], hogy a Lagrange duális optimumának értéke legalább olyan jó alsó korlátot ad az eredeti feladatra, mint az LP-relaxált feladat optimuma:

$$Z_{\mathrm{LP}} \leq Z_{\mathrm{LD}} \leq Z_{\mathrm{IP}}$$

illetve, ha $\tilde{X}$ egész poliéder, akkor $Z_{\mathrm{LP}} = Z_{\mathrm{LD}}$ teljesül [39]. Továbbá, a Lagrange duális feladat Z_{LD} optimuma megegyezik az alábbi feladat optimumával [39]:

$$\min \left\{ \sum_{j \in N} c_{ij} x_j \Big| \sum_{j \in N} a_{ij} x_j = d_i \ \forall\ i \in M_1, x \in \mathrm{conv}(X) \right\}$$

A Lagrange duális feladat egy szakaszonként lineáris, nem differenciálható konkáv függvény maximalizálása. Megoldására a legtöbb esetben szubgradiens módszert alkalmaznak, melyet a következő szakaszban ismertetünk.

1.2.2. Szubgradiens algoritmus

A gradiens módszer egy iteratív eljárás differenciálható függvények szélsőértékének meghatározására. Egy kezdeti megoldásból kiindulva minden lépésben az adott pontbeli gradiens segítségével határozzuk meg a következő pontot. Egy nem differenciálható függvény esetén azonban a gradiens módszer nem működik, hiszen nem minden pontban létezik gradiens. A szubgradiens módszer egy megoldást nyújt a problémára.

Egy konkáv $\Phi : \mathbb{R}^{|M_1|} \to \mathbb{R}$ függvény λ-beli szubgradiensén egy olyan $s \in \mathbb{R}^{|M_1|}$ vektort értünk, melyre

$$\Phi(\bar{\lambda}) \leq \Phi(\lambda) + s^T(\bar{\lambda} - \lambda)$$

teljesül minden $\bar{\lambda} \in \mathbb{R}^{|M_1|}$ esetén. Ha Φ nem differenciálható λ-ban, akkor ott több szubgradiens is létezik. Jelölje $\partial\Phi(\lambda)$ a λ-beli szubgradiensek halmazát. Az algoritmus ezek közül választ egy tetszőlegeset. (Ha Φ differenciálható λ-ban, akkor $\partial\Phi(\lambda)$ a λ-beli gradiensből áll.)

A szubgradiens algoritmus alkalmazható a Lagrange duál feladat megoldására, melyre most egy keret algoritmust mutatunk (1.2. Algoritmus). A konvergencia biztosítására egy τ lépéshossz paramétert is bevezetünk, melynek kezdőértékét 2-re állítjuk, és ha egy adott számú iteráció alatt nem sikerül javítani az alsó becslésünkön, akkor τ értékét megfelezzük.

Algoritmus 1.2 Szubgradiens algoritmus $Z_{\mathrm{LR}}(\lambda)$ maximalizálására

1: *Inicializálás*

Megadjuk a kezdeti λ^0 multiplikátorokat, beállítjuk az ε pontossági paramétert, a t lépésszámlálót 0-ra állítjuk, és kiszámolunk egy heurisztikus Z_{UB} felső korlátot az eredeti feladatunkra.

2: *Lagrange feladat megoldása*

Megoldjuk $Z_{\mathrm{LR}}(\lambda^t)$-t, és elmentjük az x^t optimális megoldást.

3: *Felső korlát frissítése*

Ha x^t megengedett megoldás, és költsége kisebb, mint Z_{UB}, akkor frissítjük Z_{UB} értékét.

4: *A δ^t lépés irány meghatározása*

Kiszámoljuk a szubgradienseket:

$$s_i^t = d_i - \sum_{j \in N} a_{ij} x_j^t \ (i \in M_1)$$

Legyen $\delta^t = s^t$.

5: *A w^t lépéshossz meghatározása*

$$w^t = \tau \cdot \frac{Z_{\mathrm{UB}} - Z_{\mathrm{LR}}(\lambda^t)}{\|\delta^t\|^2}$$

6: *Multiplikátorok frissítése*

$$\lambda_i^{t+1} = \lambda_i^t + w^t \delta_i^t \ (i \in M_1)$$

7: *Leállási kritériumok ellenőrzése*

Leállunk ha valamelyik kritérium teljesül:

$s^t = 0$

$\|\delta^t\|^2 \leq \varepsilon$

$\tau \leq \varepsilon$

$t \geq t_{\max}$

$Z_{\mathrm{UB}} - Z_{\mathrm{LR}}(\lambda^t) \leq \varepsilon$

egyébként eggyel növeljük t értékét, és a 2. lépésre ugrunk.

1.2.3. Oszlopgenerálás

Oszlopgenerálást abban az esetben alkalmazunk, ha a feladatunknak jelentősen sok oszlopa (változója) van, vagy az oszlopok mindegyikét nem tudjuk egyszerűen előállítani. A módszert először *Dantzig* és *Wolfe* definiálta [7]. Az alapötlet szerint egymás után megoldunk olyan redukált problémákat, melyek az oszlopoknak csak egy kis részét tartalmazzák. Miután egy ilyen problémát megoldunk, a duális információk alapján veszünk új oszlopo(ka)t a jelenlegi oszlopkészlethez. Egy új oszlop generálásának feladatát *árazási feladatnak* nevezzük.
Az oszlopgenerálás folyamatát az alábbi feladaton részletezzük:

$$\begin{aligned} \min \sum_{j \in N} c_j x_j & \\ \sum_{j \in N} a_{ij} x_j = d_i & \quad \forall\, i \in M \\ x_j \geq 0 & \quad \forall\, j \in N \end{aligned} \tag{1.9}$$

A feladatot először egy $N^0 \subset N$ oszlophalmazon oldjuk meg, így kapunk egy $\bar{x}$ primál megoldást, és egy $\bar{u}$ duál megoldást. Az $\bar{x}$ primál megendegett a jelenlegi redukált feladatra, és 0 koordinátákkal kiegészítve primál megengedett az eredeti (1.9) feladatra is. Egy $j \in N$ oszlop redukált költségét jelöle $\bar{c}_j$:

$$\bar{c}_j = c_j - \sum_{i \in M} a_{ij} \bar{u}$$

Ha nincs negatív redukált költségű oszlop N-ben, akkor $\bar{u}$ duál megengedett megoldása (1.9)-nek is, tehát $\bar{x}$ optimális megoldása (1.9)-nek. (Megjegyezzük, hogy N^0-ban minden oszlop redukált költsége nemnegatív.) Jelöle k a legkisebb redukált költségű oszlopot és legyen ξ a k redukált költsége:

$$k = \arg\min_{j \in N} \bar{c}_j \qquad \xi = \bar{c}_k$$

Ha $\xi \geq 0$, akkor minden oszlop redukált költsége nemnegatív.
A duál megengedettség eldöntéséhez tehát nem kell minden oszlopot ismernünk, elég megkeresnünk a legkisebb redukált költségűt. Ha ez a legkisebb költség nemnegatív, akkor az előbb leírtak alapján optimális megoldásunk van az eredeti feladatra, ha pedig a költség negatív, akkor az oszlopot hozzávesszük a jelenlegi oszlopkészlethez, és megoldjuk az új redukált feladatot. Az eljárás gyorsítása céljából egyszerre több oszlopot is hozzávehetünk a jelenlegiekhez, illetve különböző árazási eljárásokat alkalmazhatunk [1].

1.2.4. Lagrange relaxációval kombinált oszlopgenerálás

A gyakorlatban az oszlopgenerálást legtöbbször az adott feladat LP-relaxáltjának megoldására alkalmazzák, azonban az oszlopgenerálás Lagrange relaxált feladat megoldására

is alkalmas [28]. A dolgozatban mindkét alkalmazásra fogunk példát látni: előbbire a 2. fejezetben egy vezető ütemezési problémánál, utóbbira pedig a 3. fejezetben egy integrált jármű és vezető ütemezési feladat megoldásánál.
Térjünk vissza az (1.4)-(1.7) feladathoz és az (1.8) Lagrange relaxálthoz. Legyen (1.4)-(1.7) LP-relaxáltjának optimális megoldása $\bar{x}$ és az M_1 feltételekhez tartozó optimális duál megoldás $\bar{u}$. Megmutatható, hogy az $\bar{u}$-hoz tartozó Lagrange feladat optimuma megegyezik az előző LP-optimummal. Ennek ismeretében ahelyett, hogy optimálisan megoldanánk a redukált feladatokat (szimplex módszerrel) a Lagrange duális feladatot oldjuk meg közelítően (szubgradiens algoritmussal). A szubgradiens algoritmus befejeztével a Lagrange multiplikátorok az aktuális redukált feladat optimális duál változóinak egy közelítését adják. Ezek után ezen multiplikátorokat használjuk új oszlop kereséséhez.
Több dolog is indokolja az előző módszer használatát. Például a szubgradiens algoritmus gyors, könnyű implementálni, és az eljárás alatt lehetséges, hogy megengedett megoldást állítunk elő. A módszernél azonban felmerül egy probléma. Mivel a duál változókra csak egy közelítést adunk, elképzelhető, hogy a jelenlegi redukált feladat oszlopkészlete negatív redukált költségű oszlopot is tartalmaz. Ezért fennáll a veszélye, hogy az oszlopgenerálás során olyan oszlopot állítunk elő, mely már szerepel a jelenlegi oszlopkészletben. Ennek elkerülése érdekében úgy módosítjuk a multiplikátorokat, hogy a jelenlegi oszlopaink nemnegatív redukált költségűek legyenek, de az alsó korlátunk ne csökkenjen. Az 1.3. Algoritmus olyan példán mutatja be az eljárást, ahol a (1.9) feladatban $a_{ij} \in \{0, 1\}$, és az N^0 oszlopkészlet mellett az M-hez tartozó feltételeket λ multiplikátorokkal dualizáltuk.

Algoritmus 1.3 Multiplikátor módosítási eljárás

1: *Negatív oszlopok keresése*

Jelölje a negatív redukált költségű N^0-beli oszlopok halmazát N'.

2: *Multiplikátor módosítás*

Válasszunk egy N'-beli j oszlopot, melyre $\bar{c}_j < 0$.

- Legyen $\delta = \dfrac{\bar{c}_j}{\sum\limits_{i \in M} a_{ij}}$
- $\forall\ i \in M$, $a_{ij} = 1$ esetén legyen $\lambda_j := \lambda_j + \delta$
- Frissítsük az eddig nem vizsgált N'-beli oszlopok redukált költségét.

Ha létezik még negatív redukált költségű oszlop N'-ben, akkor a 2. lépéshez ugrunk, egyébként leállunk.

1.3. Erőforrás korlátos legrövidebb út

Mint azt már említettük, az erőforrás korlátos legrövidebb út (*Resource Constrained Shortest Path*, RCSP) probléma kulcsszerepet játszik a későbbi fejezetekben, hiszen számos jármű és vezető ütemezéssel kapcsolatos probléma során felmerül.
Az RCSP során adott egy irányított gráf, és benne két kijelölt csúcs. Az éleken adott egy költségfüggvény, és véges sok erőforrás függvény, melyek az erőforrás felhasználásokat mérik az adott élen. A célunk olyan legolcsóbb utat találni a két kijelölt csúcs között, amelyen az erőforrás felhasználások előre adott korlátok közé esnek. A feladat NP-nehéz még abban az esetben is, ha a gráf aciklikus, egyetlen erőforrásunk van, és minden élen az erőforrás felhasználás és a költség nemnegatív [24], [30].

1.3.1. Az RCSP formális felírása

Legyen $D = (V, A)$ egy irányított gráf, és $s, t \in V$ két kijelölt pont. Adott az éleken egy $c : A \to \mathbb{R}$ költségfüggvény és $r^k : A \to \mathbb{R}_{\oplus}$ erőforrás függvények ($k = 1, \ldots, K$). Az erőforrásokhoz adottak L_k, U_k alsó-felső korlátok ($k = 1, \ldots, K$). Egy P *st* utat *megengedettnek* nevezünk, ha az úton felhasznált erőforrások teljesítik a korlátokat:

$$L_k \leq \tilde{r}^k(P) = \sum_{ij \in P} r^k_{ij} \leq U_k, \ \forall \ k = 1, \ldots, K$$

A célunk olyan megengedett P *st* utat találni, melynek költsége: $\tilde{c}(P) = \sum_{ij \in P} c_{ij}$ minimális.

1.3.2. Megoldási módszerek

Címkéző módszerek

A címkéző módszerek hátterében a dinamikus programozás elve áll. A gráf minden egyes i csúcsához címkéket készítünk, melyek bizonyos si utakról tartalmaznak információkat, mint például az addigi költséget és az addigi erőforrás felhasználásokat. Különböző dominancia szabályok segítségével redukáljuk a csúcsoknál a címkék számát, és az si utat továbbépítve csak a nem-dominált címkéket vesszük figyelembe, míg végül elérünk t-ig.
A címkézős módszereket főként az RCSP két speciális esetére szokták alkalmazni. A WCSPP (*Weight Constrained Shortest Path Problem*, [12], [31]) feladatra tekinthetünk úgy, hogy egyetlen (nemnegatív) erőforrásunk van, és az erőforrás felhasználásra csak felső korlát adott. A SPPTW (*Shortest Path Problem with Time Windows*, [8], [10]) esetében pedig az erőforrás függvényünk azt mondja meg, hogy egy adott élen mennyi idő végighaladni. Továbbá minden i csúcson adott egy $[a_i, b_i]$ időintervallum úgy, hogy az i csúcsot nem látogathatjuk meg a_i-nél előbb, sem pedig b_i-nél később, de a csúcsoknál szabad várakozni.

A címkézős módszer egyik előnye, hogy a t csúcshoz elérve több megengedett utat is találhatunk, így ha egy oszlopgenerálás árazási feladataként oldjuk meg, akkor egy iteráció alatt több oszlopot is hozzávehetünk az addigi oszlopkészlethez.

Lagrange relaxáción alapuló módszerek

Az erőforrás korlátokat relaxálva a probléma egy korlátok nélküli legrövidebb út keresésére vezet vissza. A Lagrange relaxáción alapuló megoldások általában három fázisból állnak: 1) Az RCSP optimumára alsó és felső korlátokat számolunk (előbbit a relaxált feladat megoldásával), 2) ezek segítségével redukáljuk a gráfot, 3) bezárjuk a dualitás rést.
Handler és *Zang* [24] vágósíkos eljárással oldották meg a Lagrange duális feladatot, és K-legrövidebb út keresésével határozták meg az optimumot. *Beasley* és *Christofides* [2] szubgradiens algoritmussal oldották meg a duál feladatot, majd korlátozás és szétválasztás eljárással keresték meg az optimumot. *Mehlhorn* és *Ziegelmann* [36] szubgradiens algoritmus helyett hull módszert alkalmaztak, és végül az optimumot K-legrövidebb út keresésével határozták meg.

Korlátozás és vágás módszerek

Garcia [20] a WCSPP feladatra LP-alapú megközelítést használt és az egész megoldásokra új, érvényes egyenlőtlenségekkel erősítette az LP-feladatot. Ezeket felhasználva készített korlátozás és vágás eljárást az RCSP feladatra.

1.3.3. Beasley és Christofides megoldási módszere

A következőkben *Beasley* és *Christofides* [2] megoldási módszerét ismertetjük. Az eljárást olyan változatban írjuk le, melyben feltesszük, hogy a gráfunk aciklikus. Azért élünk ezzel a feltevéssel, mert a későbbi fejezetekben kizárólag aciklikus gráfokon kell az RCSP feladatot megoldanunk.
Az eljárás során az RCSP feladatot egészértékű programozási feladatként írjuk fel. A megoldáshoz relaxáljuk az erőforrás korlátokat, így egy feltétel nélküli legolcsóbb út feladatot kapunk, melyet már polinomiálsan meg tudunk oldani. A Lagrange alsó korlát javítására szubgradiens algoritmust alkalmazunk, de nem törekedünk az alsó korlát maximalizására, csupán néhány iteráción keresztül futtatjuk az eljárást. Ha az adott iterációk során nem kapunk optimális megoldást, akkor az eljárás során kapott legjobb megoldással és a hozzá tartozó multiplikátorokkal számolunk tovább, és korlátozás és szétválasztás algoritmussal számoljuk ki az RCSP feladat optimális megoldását.

Felírás egészértékű feladatként

Minden ij élhez rendeljünk egy x_{ij} bináris változót, mely azt mondja meg, hogy az él benne van-e az optimális útban vagy nincsen:

$$x_{ij} = \begin{cases} 1 & \text{ha az } ij \text{ él benne van az optimális útban} \\ 0 & \text{különben} \end{cases}$$

Ezek után az RCSP feladatot a következőképpen írhatjuk fel:

$$\min \sum_{ij \in A} c_{ij} x_{ij} \tag{1.10}$$

$$L_k \leq \sum_{ij \in A} r_{ij}^k x_{ij} \leq U_k \qquad k = 1, \ldots, K \tag{1.11}$$

$$\sum_{i:\ ij \in A} x_{ij} = \sum_{i:\ ji \in A} x_{ji} \qquad \forall\ j \in V - \{s, t\} \tag{1.12}$$

$$\sum_{j:\ sj \in A} x_{sj} = 1 \tag{1.13}$$

$$x_{ij} \in \{0, 1\} \qquad \forall\ ij \in A \tag{1.14}$$

Az (1.12)-(1.14) feltételek biztosítják azt, hogy azon élek, melyeken az x változó 1-t vesz fel értékül, valóban egy st utat határozzanak meg. Az (1.11) feltétellel érjük el, hogy az st úton az erőforrás korlátok teljesüljenek. A célfüggvény pedig az út költségét minimalizálja. Megjegyezzük, hogy ebben a felírásban kihasználtuk, hogy a gráfunk aciklikus, ugyanis egyéb esetben akár egy st séta is kielégíthetné a feltételeket. Ekkor a fenti feladathoz még bizonyos vágás-feltételeket is hozzá kellene vennünk.

Lagrange feladat

Dualizáljuk az erőforrás korlátokat úgy, hogy a felső korlátokhoz $s_k \geq 0$, az alsó korlátokhoz $t_k \geq 0$ Lagrange-szorzókat rendelünk. Ekkor a Lagrange feladat (LLBP) az alábbi alakba írható:

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

A célfüggvényben szereplő utolsó tag konstans, tehát az (LLBP) probléma egy legolcsóbb st út meghatározása a C perturbált költségfüggvény szerint:

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

Szubgradiens algoritmus

Vezessük be a következő jelöléseket:

$Z_{\max}$:	az eddig talált legjobb alsó korlátunk az LLBP feladatra
Z_{UB}:	az eddig talált legjobb felső korlátunk az RCSP feladatra
B:	iteráció számláló
$B_{\max}$:	maximális iterációk száma

A szubgradiens módszer leírását az 1.4. Algoritmus tartalmazza.

Korlátozás és Szétválasztás eljárás

A korlátozás és szétválasztás eljárás előtt célszerű lehet különböző redukciós teszteket elvégezni, melyek kizárhatnak bizonyos éleket, amelyeket optimális st út nem tartalmazhat. Ezzel a korlátozás és szétválasztás eljárás során egyes részfákat meg sem kell vizsgálnunk.
A korlátozás és szétválasztás eljárás során az RCSP feladatra egy optimális st utat keresünk. Ehhez az s csúcsból indulva építünk utat. Egy aktuális si útnál különböző tesztekkel ellenőrizzük, hogy befejezhető-e optimális st úttá. A tesztek során felhasználjuk a szubgradiens algoritmus során eltárolt legjobb multiplikátorokat. A tesztek jelentősen gyorsíthatják az eljárás futását, és ezt a 2.3.5. alszakaszban meg is vizsgáljuk.

Erőforrás teszt. Jelölje R_{ij}^k azt a legkevesebb mennyiséget a k. erőforrásból, amelyet az i csúcsból a j-be el tudunk vinni (azaz R_{ij}^k a legrövidebb ij út hossza, ahol a távolságokat a k erőforrás felhasználások adják). Megjegyezzük, hogy az R_{ij}^k értékeket elég egyszer, a korlátozás és szétválasztás eljárás előtt kiszámolni, és az értékeket eltárolni.
A gráfból minden olyan ij él elhagyható, melyre a következő teljesül valamely k erőforrás esetén:

$$R_{si}^k + r_{ij}^k + R_{jt}^k > U_k \qquad \text{(TEST 1)}$$

A teszt azt vizsgálja meg, hogy a k. erőforrás esetén mennyi a lehető legkevesebb erőforrás felhasználás azon st-utakon, melyek az ij élt tartalmazzák. Ha ez az érték nagyobb, mint a k. erőforrás felső korlátja, akkor az ij kiküszöbölhető, hiszen egyetlen megengedett st út sem tartalmazhatja.

Költség teszt. Jelölje D_{ij} a legolcsóbb ij út hosszát a $C^{\max}$ költségfüggvény szerint. Megjegyezzük, hogy a D_{ij} értékeket is elegendő egyszer kiszámolni a korlátozás és szétválasztás eljárás előtt.
Minden olyan i csúcs elhagyható a gráfból, melyre a következő teljesül:

$$D_{si} + D_{it} + Z_{\mathrm{OFF}} > Z_{\mathrm{UB}} \qquad \text{(TEST 2)}$$

Algoritmus 1.4 Szubgradiens algoritmus az LLBP feladatra

1: *Inicializálás*

Legyenek $s_k = t_k = 0$ $(k = 1, \ldots, K)$. Legyen $Z_{\max} = -\infty$. Legyen $B = 0$.

2: *Oldjuk meg az LLBP feladatot az aktuális multiplikátorokkal*

A megoldást jelölje X, az aktuális alsó korlátot Z_{LB}:

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

3: *Alsó korlát frissítése*

Ha $Z_{\mathrm{LB}} > Z_{\max}$, akkor legyen $Z_{\max} = Z_{\mathrm{LB}}$.

4: *Felső korlát frissítése*

Ha X megengedett megoldás az RCSP feladatra, akkor frissítjük Z_{UB}-t:

$$Z_{\mathrm{UB}} = Z_{\mathrm{UB}}, \sum_{ij \in A} c_{ij} X_{ij}$$

5: *Megállási kritérium ellenőrzése*

Ha $Z_{\max} = Z_{\mathrm{UB}}$, akkor megállunk, hiszen optimális megoldásunk van az RCSP feladatra.

6: *Szubgradiensek kiszámítása*

$$G_k = L_k - \sum_{ij \in A} r_{ij}^k X_{ij} \qquad k = 1, \ldots, K$$
$$H_k = -U_k + \sum_{ij \in A} r_{ij}^k X_{ij} \qquad k = 1, \ldots, K$$

7: *Multiplikátorok frissítése*

Definiáljuk a T lépéshosszt:

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

ahol $0 < \tau \leq 2$ paraméter. A szubgradiensek a következők lesznek:

$$s_k = \max\{0, s_k + TG_k\} \qquad k = 1, \ldots, K$$
$$t_k = \max\{0, t_k + TH_k\} \qquad k = 1, \ldots, K$$

8: *Iterációk számának ellenőrzése*

Növeljük B értékét eggyel. Ha $B < B_{\max}$, akkor a 2. lépéstől folytatjuk az algoritmust. Egyébként leállunk és elmentjük a legjobb megoldáshoz ($Z_{\max}$-hoz) tartozó multiplikátorokat $(s_k^{\max}, t_k^{\max})$ és a megoldást. Jelölje $C^{\max}$ az $s_k^{\max}, t_k^{\max}$ multiplikátorokkal perturbált költségfüggvényt, illetve legyen $Z_{\mathrm{OFF}} = \sum_{k=1}^{K} s_k^{\max} L_k - t_k^{\max} U_k$.

Továbbá minden olyan ij él elhagyható a gráfból, melyre teljesül a következő:

$$D_{si} + C_{ij}^{\max} + D_{jt} + Z_{\mathrm{OFF}} > Z_{\mathrm{UB}} \qquad \text{(TEST 3)}$$

A tesztek során az i csúcsot (az ij élt) tartalmazó st-utak költségére adunk alsó becslést a perturbált költségfüggvény segítségével. Ha a kapott érték nagyobb, mint az optimális st útra talált legjobb felsőbecslésünk, akkor az i csúcs (az ij él) kiküszöbölhető, hiszen nem tartalmazhatja optimális út.

Korlátozás és Szétválasztás tesztek. A korlátozás és szétválasztás eljárás során minden korlátozás és szétválasztás csúcshoz tartozik a gráf egy i csúcsa, és az addig felépített si út. Jelölje ennek az si útnak a c költségfüggvény szerinti hosszát w, a $C^{\max}$ költségfüggvény szerinti hosszát W. Jelölje az si úton felhasznált erőforrás mennyiségét a k. erőforrás szerint B_k.
Minden olyan j csúcs kiküszöbölhető az adott részfában, melyre létezik ij él, és valamely k erőforrásra a következő teljesül:

$$B_k + r_{ij}^k + R_{jt}^k > U_k \qquad \text{(TEST 4)}$$

A teszt során azt vizsgáljuk, hogy mennyi a lehető legkevesebb felhasználás a k. erőforrásból az aktuális si út egy st úttá történő kiegészítésénél. Ha ez az érték nagyobb, mint a k. erőforrás felső korlátja, akkor az adott részfában a j csúcs kiküszöbölhető, hiszen nem tartalmazhatja optimális út.
Továbbá minden olyan j csúcs kiküszöbölhető az adott részfában, melyre a következő teljesül:

$$W + D_{ij} + D_{jt} + Z_{\mathrm{OFF}} > Z_{\mathrm{UB}} \qquad \text{(TEST 5)}$$

A teszt során az si útból kiegészített st-utak hosszára adunk alsó becslést a perturbált költségfüggvény segítségével. Ha a kapott érték nagyobb, mint az optimális st útra talált legjobb felsőbecslésünk, akkor a j csúcs kiküszöbölhető az adott részfában, hiszen nem tartalmazhatja optimális út.

Visszalépés. Amennyiben a korlátozás és szétválasztás eljárás során egy si utat nem tudunk folytatni (mert a tesztek minden lehetséges befejezést kiküszöböltek), vagy $Z_{\mathrm{LB}} > Z_{\mathrm{UB}}$ teljesül, akkor visszalépünk a korlátozás és szétválasztás fában és folytatjuk az eljárást.

2. fejezet

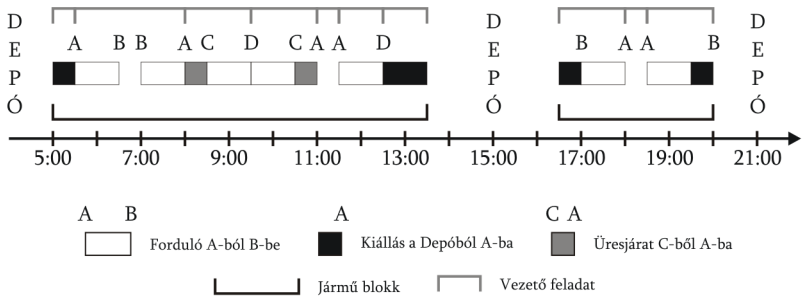
Jármű és vezető ütemezés

2.1. Jármű ütemezés

A bevezetőben már felvázoltuk a jármű és a vezető ütemezés feladatait és az ezzel kapcsolatos fogalmakat, de a következő szakaszokban részletesebben is áttekinjük őket.

2.1.1. A jármű ütemezés feladata

A jármű ütemezés (*Vehicle Scheduling Problem*, VSP) során a menetrendben szereplő fordulókhoz kell járműveket rendelnünk úgy, hogy a járművek összköltsége minimális legyen. Ehhez akár több depó is rendelkezésünkre állhat.
Tekintsük az i és j fordulókat. Az i forduló érkezési időpontja e_i, célállomása E_i. A j forduló kezdőállomása S_j, indulási időpontja s_j. Az E_i és S_j állomások közötti utazási idő τ_{ij}. Az (i, j) fordulópárt *kompatibiliseknek* nevezzük, ha ugyanazon járművel teljesíthetők ebben a sorrenben, azaz $e_i + \tau_{ij} \leq s_j$ teljesül. Egy jármű *menetterve* a hozzárendelt (páronként kompatibilis) fordulókból, és bizonyos kiállásokból/beállásokból illetve üresjáratokból áll. A 2.1.1. ábrán egy jármű menettervére láthatunk egy példát. A jármű 5:00-kor áll ki a depóból és 5:30-tól 12:30-ig fordulókat teljesít, majd ezután beáll a depóba. 16:30-kor ismét kiáll, hogy két fordulót teljesítsen, végül 20:00-kor befejezi a napi menetét. Két egymást követő forduló között lehetséges, hogy nincs szünet (például 8:30-tól 10:30-ig szünet nélkül teljesít két fordulót), egyébként vagy egy adott állomáson várakozik a jármű (például 6:30-tól 7:00-ig a B állomáson), vagy két állomás között utasok nélkül közlekedik (például 8:00-tól 8:30-ig A-ból C-be). A menetterv két *jármű blokkból* áll: 5:00-tól 13:30-ig illetve 16:30-tól 20:00-ig. Minden egyes jármű használata egy fix költséget von maga után, illetve egy olyan működési költséget, amely a depóból kint töltött időtől függ.



2.1.1. ábra. Egy jármű menetterve

Egy-depós jármű ütemezés. Az egy-depós jármű ütemezés (*Single-Depot Vehicle Scheduling Problem*, SDVSP) során egyetlen depóból kell járműveket rendelnünk a fordulókhoz.

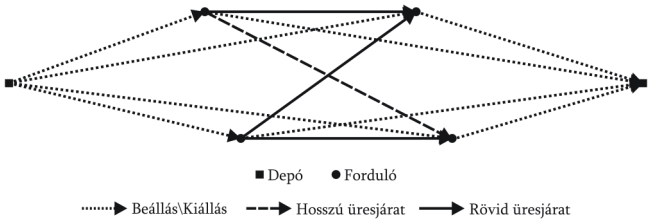
Több-depós jármű ütemezés. A több-depós jármű ütemezés (*Multi-Depot Vehicle Scheduling Problem*, MDVSP) során több depó áll rendelkezésünkre, és minden fordulóra adott, hogy az mely depókból teljesíthető. Minden fordulóhoz pontosan egy depóból kell pontosan egy járművet rendelni.

2.1.2. Modellezési eszközök

A jármű ütemezés illetve az integrált jármű és vezető ütemezés során két, lényeges eltérő megközelítést szoktak alkalmazni, melyeket most be is mutatunk. Az elsőben egy kapcsolat alapú hálózatot (*Connection Based Network*, CBN) használunk, a másodikban pedig egy tér-idő hálózatot (*Time-Space Network*, TSN). Mindkét esetben az adott hálózaton, egészértékű feladatként fogalmazzuk meg az adott problémát. A tér-idő hálózat előnye, hogy jelentősen tudjuk csökkenteni az üresjáratokat reprezentáló élek számát, és ezzel a feladatban szereplő feltételek számát.
A kapcsolat alapú megközelítés tulajdonképpen a legkézenfekvőbb modellezési formája a problémának, és előszeretettel használták a VCSP [15], és a MDVCSP feladatra is [26] [27]. A tér-idő hálózatos megközelítést a MDVSP feladatra elsőként *Kliewer et al.* [32] [33] vezette be, habár légiközlekedésbeli ütemezési problémáknál már régebben ismert módszer volt [25]. A MDVCSP feladatra először *Gintner* [23] alkamazta. A tér-idő hálózat megjelenése egészen új megközelítést adott mind a MDVSP [4] [32] [33], mind a MDVCSP problémákra [42] [43].

Kapcsolat alapú hálózat

Legyen $\mathcal{T} = \{1, 2, \ldots, n\}$ a fordulók halmaza, $\mathcal{D}$ pedig a depóké. A kompatibilis párok (üresjáratok) halmazát jelölje E: $E = \{(i, j) : i \in \mathcal{T}, j \in \mathcal{T}, (i, j) \text{ kompatibilis}\}$. Egy



2.1.2. ábra. Kapcsolat alapú hálózat 4 fordulóhoz

adott $d \in \mathcal{D}$ depóhoz a következőképpen készítünk el egy $G^d = (V^d, A^d)$ kapcsolat alapú hálózatot. Minden egyes fordulóhoz felveszünk egy forduló-csúcsot, továbbá az s^d és t^d és csúcsokat, melyek a depót reprezentálják. Az s^d csúcsból minden forduló-csúcsba, és minden forduló-csúcsból a t^d csúcsba húzunk egy irányított élt. Két forduló-csúcsot pedig akkor kötünk össze irányított éllel, ha a nekik megfelelő csúcsok kompatibilisek. Formálisan ez a következőt jelenti:

$$V^d = \mathcal{T}^d \cup \{s^d, t^d\},\ A^d = E^d \cup (s^d \times \mathcal{T}^d) \cup (\mathcal{T}^d \times t^d)$$

ahol $\mathcal{T}^d \subseteq \mathcal{T}$ azon fordulók, melyek a d depóból teljesíthetőek, és $E^d \subseteq E$ a $\mathcal{T}^d$ fordulókhoz tartozó üresjáratok. Egy jármű menetterve ekkor kölcsönösen egyértelműen megfeleltethető egy $s^d t^d$ útnak a hálózatban, és a járművek menettervei együtt belsőleg diszjunkt $s^d t^d$ utakat alkotnak.

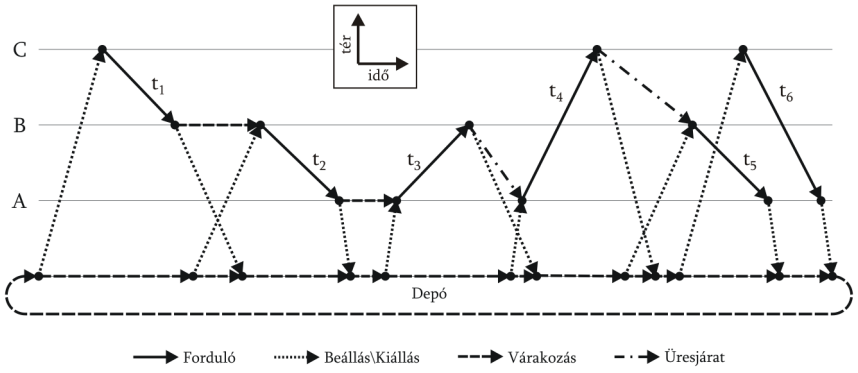
A feladatban szereplő feltételeinek számának (nagymértékű) csökkentése érdekében *Huisman* [26] bevezette a *rövid* és *hosszú élek* fogalmát. Feltesszük, hogy ha két egymást követő forduló között az üresjárati idő akkora, hogy a járművel vissza tudunk térni a depóba és újra kiállni, akkor ezt meg is tesszük. Ekkor az üresjárati élt hosszú élnek nevezzük, egyébként rövidnek.

A kapcsolat alapú hálózatra a 2.1.2. ábrán láthatunk példát.

Tér-idő hálózat

Legyen a fordulók halmaza $\mathcal{T} = \{1, 2, \ldots, n\}$ a depók halmaza $\mathcal{D} = \{1, 2, \ldots, m\}$. A d depóból teljesíthető fordulók halmazát $\mathcal{T}^d$-vel jelöljük. A tér-idő hálózatra érdemes úgy tekinteni, hogy minden csúcs a tér-idő skálán helyezkedik el: hozzárendelünk egy időpontot és egy állomást (vagy a depót). A 2.1.3. ábrán hat fordulóhoz készítettünk el egy tér-idő hálózatot. A következőkben részletesen leírjuk, hogy egy $d \in \mathcal{D}$ depóhoz és az adott fordulókhoz hogyan készítjük el a tér-idő hálózatot.

Fordulók, be-, és kiállások. Első lépésként minden fordulónak felveszünk egy forduló-élt. Az él tövéhez/fejéhez a forduló indulásának/érkezésének időpontját és helyét rendeljük.

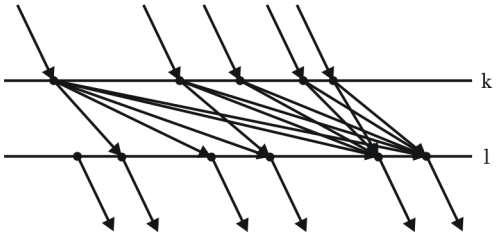


2.1.3. ábra. Tér-idő hálózat 6 fordulóhoz

Felvesszük még hozzá a ki-, és beállás éleket.

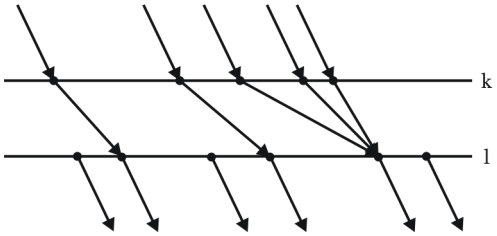
Idővonalak. A hálózat csúcsai már adottak. Minden egyes állomáshoz és a depóhoz készítünk egy idővonalat. Ez azt jelenti, hogy az adott állomáshoz/depóhoz tartozó csúcsokat az időpontjuk szerinti növekvő sorrendben tekintjük, és minden csúcsból húzunk egy élt a következő csúcsba. Megjegyezzük, hogy (a depó kivételével) nem muszáj minden élt behúzni, ugyanis nincs mindre szükség ahhoz, hogy biztosítsuk két kompatibilis forduló közötti kapcsolatot. Például az 2.1.3. ábrán a t_3 és t_5 fordulók között nincsen idővonalbeli él behúzva, de a depón keresztül a t_3 él fejéből a t_5 él töve elérhető.
A depó esetén az idővonal utolsó csúcsából húzunk egy *cirkulációs-élt* az idővonal első csúcsába. Ez csak technikai megoldás, hogy a későbbiekben folyam helyett áram-feladatként fogalmazzunk meg bizonyos feladatokat.

Üresjáratok. Most következik a tér-idő hálózat igazi erőssége. Tekintsünk egy k állomást, rajta azon csúcsokkal, melyek valamely fordulók érkezési pontjai, illetve egy ℓ állomást, rajta azon csúcsokkal, melyek valamely fordulók indulási pontjai. Ha minden üresjárati élt közvetlenül bevennénk a hálózatba, akkor a 2.1.4. ábrán látható állapothoz jutnánk, de az üresjárati éleket most két lépésben redukáljuk.



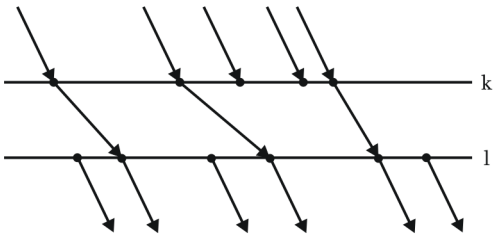
2.1.4. ábra. Tér-idő hálózat. A k és ℓ állomások közti üresjárat-élek.

Először a k állomás minden (érkezési) csúcsára csak a legelőbb végződő üresjárat-élt tartjuk meg (2.1.5. ábra). Azaz minden olyan t^k fordulóhoz, mely a k állomásra érkezik, megkeressük azt a legelőbb kezdődő t^ℓ fordulót, mely az ℓ állomásról indul és kompatibilis t^k-val.



2.1.5. ábra. Tér-idő hálózat. A k állomás csúcsaihoz tartozó legelső üresjárat-élek.

Másodszor az ℓ állomás (indulási) csúcsainál csak az időben legkésőbb kezdődő üresjáratéleket tartjuk meg az előzőek közül (2.1.6. ábra).



2.1.6. ábra. Tér-idő hálózat. Az ℓ állomás csúcsaihoz tartozó legkésőbbi üresjárat-élek.

Az ábrákon könnyen ellenőrizhető, hogy ezzel a redukcióval bármely két kompatibilis forduló közötti kapcsolat továbbra is biztosítva van, esetleg közvetetten (azaz kell üresjárat-, és várakozás-élt is használnunk az eléréshez). A tér-idő hálózat esetében is minden jármű

menettervének megfelel egy út a gráfban, de nem feltétlenül egyértelműen. Sőt, a cirkulációs-élt (és a szükséges depó-éleket) hozzávéve, minden menetterv egy kört alkot. Ezzel a menettervek együttesen egy áramot határoznak meg a gráfban.

2.1.3. SDVSP

Készítsük el a fordulókhoz a kapcsolat alapú hálózatot. Jelölje c_{ij} a jármű működési költségét az ij élen. A járművekre vonatkozó fix költséget a forrásból induló élekre írjuk. Legyen y_{ij} döntési változó az ij élen, hogy azt használja jármű vagy sem:

$$y_{ij} = \begin{cases} 1 & \text{ha egy jármű az } i \text{ forduló után közvetlenül a } j \text{ fordulót teljesíti} \\ 0 & \text{különben} \end{cases}$$

Ekkor a SDVSP feladat a következő, kvázi-hozzárendelési feladatként írható fel [40]:

$$\min \sum_{ij \in A} c_{ij} y_{ij} \tag{2.1}$$

$$\sum_{j:ij \in A} y_{ij} = 1 \qquad \forall\, i \in \mathcal{T} \tag{2.2}$$

$$\sum_{i:ij \in A} y_{ij} = 1 \qquad \forall\, j \in \mathcal{T} \tag{2.3}$$

$$y_{ij} \in \{0, 1\} \qquad \forall\, ij \in A \tag{2.4}$$

A (2.1)-(2.4) feladat felfogható úgy, mint egy minimális költségű folyam keresése (ahol a csúcsokon adottak alsó-felső korlátok), de párosítási feladatként is lehet rá tekinteni. A SDVSP feladat tehát polinom időben megoldható. Leggyakrabban *Freling et al.* [16] Aukciós algoritmusát alkalmazzák a feladat megoldására.
A SDVSP feladat modellezéséhez és megoldásához a tér-idő hálózatos megközelítés is alkalmazható. Ekkor minden forduló-élre alsó és felső korlátként 1-t írva egy minimális költségű megengedett áram keresésével oldható meg a feladat.

2.1.4. MDVSP

Az egy-depós esettel szemben a MDVSP feladat NP-nehéz, már két depó esetén is [3]. A MDVSP feladat nem játszik fontos szerepet a dolgozatban, ezért nem adunk róla részletes modellezési és megoldási módszereket, csupán egy rövid áttekintést nyújtunk.
A MDVSP feladat modellezése a legtöbb esetben többtermékes folyamként történik [33] [35] [38], ahol minden depóra felépítünk egy hálózatot, melyeket egy multigráffá kombinálunk. Az egészértékű feladat során általában két típusú változót használunk: egyiket arra, hogy a fordulókat a depókhoz rendeljük, másikat pedig a folyamfeltételek teljesítésére.
Löbel [35] kapcsolat alapú hálózatot használt a modellezéshez. Az élek nagy száma miatt egy speciális Lagrange árazási eljárással oldotta meg a feladatot. *Kliewer et al.* [33] tér-idő

hálózatot használt. Mivel ebben a megközelítésben a üresjárat-éleket drasztikusan sikerült redukálni (97-99%-kal), ezért egy MIP-solver segítségével oldották meg a feladatot.

2.2. Vezető ütemezés

2.2.1. A vezető ütemezés feladata

A vezető ütemezés (*Crew Scheduling Promlem*, CSP) során a járművekhez kell vezetőket rendelnünk a vezetőkre vonatkozó költségek minimalizálásával. A vezetők nem csak a depóban kezdhetik/fejezhetik be a szolgálatukat/szünetüket, hanem bizonyos kijelölt állomásokon, úgynevezett *átszállási pontokon*. A jármű ütemezés során meghatározott menetterveknél egy-egy jármű blokkon belül több átszállási lehetőség is adott lehet a vezetők számára. Két átszállási pont közötti részt (fordulók, üresjáratok, beállások/kiállások, várakozások sorozatát) *vezető feladatnak* (röviden: feladatnak) nevezünk. Tulajdonképpen a vezető ütemezés során ezekhez a feladatokhoz kell vezetőt rendelnünk, ezzel összeállítva egy-egy vezető *beosztását*. A 2.1.1. ábrán olyan példa látható, ahol a depón kívül az átszállási pontok csak az A illetve D állomások lehetnek. Ekkor az első jármű blokk hét, a második pedig három feladatból áll össze.
A beosztások megengedettségére különböző feltételek adhatók. Megemlítünk most néhány tipikus megkötést, mint például a megosztás időtartamára vonatkozó alsó-felső korlát. A vezetők számára biztosítani kellhet egy hosszabb ebédszünetet vagy több, kisebb szünetet. Ezek a beosztást *munkarészekre* osztják. Egyrészt a szünetek/munkarészek számára, időtartamára is vonatkozhatnak alsó-felső korlátok, másrészt a vezető munkával töltött idejére (beosztás hossza mínusz a szünetek időtartama) is adhatók korlátok.
A vezetők bérezése általában egy fix, illetve a munkaidőből számított összegből áll össze.

2.2.2. CSP

A vezető ütemezés feladat NP-nehéz még azokban az esetekben is, mikor kizárólag a beosztás hosszára, vagy a munkával töltött időre vannak korlátaink [17] [18].
A CSP feladatot általában halmaz partícionálási/fedési feladatként fogalmazzák meg, és oszlopgenerálással oldják meg. Ennél a megközelítésnél a beosztásra vonatkozó feltételekkel csupán az árazási feladat során kell foglalkoznunk. A fő probléma LP-relaxációval és Lagrange relaxációval is megoldható, mindkét esettel találkozhatunk a szakirodalomban ([9] [11] illetve [6] [15]).

2.3. Egy vezető ütemezési probléma

Az alábbi szakaszban egy vezető ütemezési problémát ismertetünk részletesen, melynek implementációját is elkészítettük. A szakasz végén a futási eredményeket mutatjuk be.

2.3.1. A probléma ismertetése

Eddig csak általánosságban beszéltünk a vezető ütemezési feladatról, de most egy speciális változatát fogjuk vizsgálni. Tegyük fel, hogy van egy központi állomásunk, és a menetrendben szereplő fordulók olyanok, hogy a központi állomáson kezdődnek és fejeződnek be. Tehát amikor egy jármű egy fordulót teljesít, azalatt most azt értjük, hogy a központi állomásról elindul (egy másik végállomásig), majd a központi állomásra tér vissza.
Tegyük fel, hogy a jármű ütemezési rész befejeződött. Az előző megkötések miatt a vezető feladatok is a központi állomáson kezdődnek/fejeződnek be, így csak a központi állomás lehet átszállási lehetőség a vezetők számára.
A célunk minden feladathoz vezetőt rendelni úgy, hogy a lehető legkevesebb vezetőt használjuk. A vezetők beosztására a következő feltételeket vesszük figyelembe:

- Egy vezető munkaideje (a beosztás időtartama) legalább L, legfeljebb U perc lehet.
- Minden vezető számára biztosítanunk kell pontosan egy, legalább b_l, legfeljebb b_u perces ebédszünetet.
- Az ebédszüneten kívül nem lehet b percnél hosszabb szünete a vezetőnek. (A b percnél rövidebb szüneteket *átszállásnak* nevezzük.)

Ha egy beosztásra az előző feltételek teljesülnek, akkor a beosztást *megengedettnek* nevezzük.

2.3.2. A probléma modellezése

Fedési feladat

A problémát a szokásos módon, fedési feladatként fogalmazzuk meg. Jelölje K a megengedett beosztások halmazát, $K(i)$ pedig az i feladatot tartalmazó megengedett beosztások halmazát. Egy $k \in K$ beosztás költségét jelölje f_k (esetünkben $f_k \equiv 1$). Legyen x_k a megengedett beosztásokhoz tartozó döntési változó:

$$x_k = \begin{cases} 1 & \text{ha a } k \text{ beosztáshoz rendelünk vezetőt} \\ 0 & \text{különben} \end{cases}$$

A fedési feladat:

$$\min \sum_{k \in K} x_k \tag{2.5}$$

$$\sum_{k \in K(i)} x_k \geq 1 \tag{2.6}$$

$$x_k \in \{0, 1\} \qquad \forall\, k \in K \tag{2.7}$$

Hálózat

Az árazási problémához a következő hálózatot készítjük el, mely lényegében egy kapcsolat alapú hálózat a feladatokhoz. Minden w feladatnak felveszünk egy $w'w''$ irányított élt. Felvesszük az s és t csúcsokat. Ha egy vezető a w_1 és w_2 feladatokat teljesíteni tudja egy átszállás vagy egy ebédszünet közbeiktatásával, akkor behúzzuk a $w_1''w_2'$ élt. Az eddigi élek l hossza a megfelelő esemény időtartama lesz. Végezetül minden w feladatra behúzzuk az sw' és az $w''t$ éleket, melyek 0 hosszúak lesznek.
Ebben az aciklikus irányított gráfban egy st utat nevezzünk *megengedettnek*, ha hossza L és U közé esik, illetve pontosan egy (ebédszünetnek megfelelő) ebéd-élt tartalmaz. Ezzel kölcsönösen egyértelmű megfeleltetést adtunk a megengedett beosztások és a megengedett st utak között.

2.3.3. A probléma megoldása

Első lépésként tekintsük a (2.5)-(2.7) feladat LP-relaxáltját, melyben a (2.7) feltételek helyett annyit teszünk fel, hogy $x_k \geq 0$ ($\forall\, k \in K$). Ekkor persze minden optimális megoldásra $0 \leq x_k \leq 1$ is teljesül. A megoldás során oszlopgenerálást alkalmazunk. A kezdeti oszlopkészletünk olyan (nem megengedett) beosztásokból áll, melyek 1-1 feladatot tartalmaznak. Mivel ezek nem megengedett beosztások, ezért a célfüggvénybe nagy költséggel vesszük be őket, hogy optimális megoldás ne tartalmazza.
Az aktuális K' oszlopkészlettel megoldjuk az LP-relaxáltat. Jelölje $\bar{y}$ az optimális duál megoldást. Egy k megengedett beosztás redukált költsége ekkor:

$$\bar{f}_k = 1 - \sum_{w \in W(k)} \bar{y}_w$$

ahol $W(k)$ a k beosztásban szereplő feladatok halmaza. Definiáljunk a hálózat élein egy költségfüggvényt:

$$c_{ij} = \begin{cases} -\bar{y}_w & \text{ha } ij \text{ a } w \text{ feladatnak megfelelő él} \\ 0 & \text{különben} \end{cases}$$

Ezek alapján egy megengedett beosztás redukált költsége negatív, ha a neki megfelelő megengedett st út c szerinti költsége kisebb, mint -1. Ezért, hogy eldöntsük létezik-e negatív

		Erőforrás felhasználás	
Az ij él típusa	Költség	Ebéd	Hossz
Feladat-él	$-\bar{y}_w$	0	l_{ij}
Átszállás-él	0	0	l_{ij}
Ebéd-él	0	1	l_{ij}

2.3.1. táblázat. A hálózatra vonatkozó adatok

redukált költségű oszlop, megkeressük a legolcsóbb megengedett st utat. Ha ennek költsége kisebb, mint -1, akkor a neki megfelelő oszlopot hozzávesszük a jelenlegi oszlopkészlethez. A legolcsóbb megengedett st út keresése egy erőforrás korlátos legrövidebb út keresése két erőforrással. Érdemes kiemelni, hogy az oszlopgenerálás során a hálózat nem változik, csupán a kölségfüggvény módosul. A hálózatra vonatkozó adatokat a 2.3.1. táblázatban foglaljuk össze. Az LP-optimum elérése után az addigi oszlopkészlettel megoldjuk az fedési feladatot. Az eddig leírt módszert a 2.1. Algoritmus foglalja össze.

Algoritmus 2.1 A vezető ütemezési probléma megoldása

1: *Inicializálás*
Kiindulunk a K' oszlopkészletből.

2: *LP-relaxált megoldása*
Megoldjuk az aktuális oszlopkészlettel az LP-relaxált feladatot

3: *Árazási feladat*
Erőforrás korlátos legrövidebb út keresésével negatív redukált költségű oszlopot keresünk. Ha találunk ilyen oszlopot, akkor hozzávesszük a jelenlegi oszlopkészlethez és a 2. lépéshez ugrunk. Egyébként a 4. lépéssel folytatjuk.

4: *Egészértékű megoldás keresése*
A jelenlegi oszlopkészlettel megoldjuk a halmaz fedési problémát.

2.3.4. Implementáció

A probléma megoldására két implementációt készítettünk. A két implementáció közti legfontosabb különbséget az árazási feladat megoldása jelentette.

I. Algoritmus

Az I. Algoritmus implementációja Mosel programozási nyelven történt. Felépítettük a fedési problémát, majd az árazási feladathoz tartozó problémát. Az árazási feladatot, mint egészértékű feladatot oldottuk meg.

II. Algoritmus

A II. Algoritmus implementációja C++ nyelven történt, a LEMON könyvtár [34] felhasználásával. Felépítettük a fedési problémát, majd az árazási feladathoz tartozó hálózatot. Az árazási feladat megoldása az 1.3.3. alszakaszban bemutatott eljárással történt.

2.3.5. Tesztelés, eredmények

A teszteléshez a BKV menetrendjeiből készítettünk inputokat. A tesztelést 1.86 GHz-es Intel Pentium Dual CPU T2390 processzorral; Windows 7 operációs rendszeren végeztük. Az egészértékű feladatok megoldására a FICO Xpress Optimization Suite 7.3.0 verzióját használtuk.
A következőkben 4 inputon történő tesztelés eredményeit mutatjuk be. Az A és B (kisebb) inputok buszokhoz, villamosokhoz tartozó délelőtti feladatokat tartalmaznak, a C és D (nagyobb) inputok pedig buszokhoz, villamosokhoz tartozó egész napos feladatokat tartalmaznak. A 2.3.2. Táblázat tartalmazza az egyes inputokban szereplő feladatok számát, illetve a hozzájuk tartozó hálózatok adatait.

	Feladatok száma	Csúcsok száma	Élek száma
A Input	19	40	100
B Input	30	62	185
C Input	86	174	728
D Input	105	212	963

2.3.2. táblázat. Az egyes inputokhoz tartozó hálózatok adatai

A két algoritmus összehasonlítása

Az I. és a II. Algoritmus az árazási feladat megoldásában tér el lényegesen. Az I. Algoritmus esetén az RCSP feladatot egészértékű programként oldottuk meg minden egyes iterációban. Ez azt jelenti, hogy minden egyes oszlop generálása egy olyan korlátozás és szétválasztás eljárással történt, amelynél minden csúcsban egy (kisebb) egészértékű feladatot oldottunk meg. A II. Algoritmus esetén pedig olyan korlátozás és szétválasztás algoritmust alkalmaztunk, melynél a csúcsokban csupán könnyen számolható teszteket hajtottunk végre, egy néhány iteráción át futtatott szubgradiens algoritmusból kapott értékek segítségével.
Azt feltételeztük tehát, hogy a II. Algoritmus hatékonyabb lesz az I. Algoritmusnál. Ennek vizsgálatához mindkét algoritmus futása esetén megmértük az oszlopgenerálás idejét.

	Átlagos futási idő	Új oszlopok száma	$\mathrm{OPT_{LP}}$	$\mathrm{OPT_{IP}}$
I. Algoritmus	0,8 másodperc	19	7	7
II. Algoritmus	0,1 másodperc	17	7	7

2.3.3. táblázat. Futási eredmények (A Input)

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

2.3.4. táblázat. Futási eredmények (B Input)

Tesztelés kisebb inputokon. A futási eredményeket az A és B inputokra a 2.3.3. illetve a 2.3.4. Táblázat tartalmazza. A várakozásoknak megfelelően a II. Algoritmus 7-8-szor rövidebb idő alatt végzett az oszlopgenerálással, habár közel ugyanannyi új oszlopot generált mindkét algoritmus. Azonban ilyen rövid (pár másodperc) futási időkből hosszabbtávú következtetést még nem szabad levonni.
Érdemes megfigyelni, hogy az I. Algoritmus mindkét esetben előállította az egészértékű optimumot, a II. Algoritmus pedig a második esetben csak eggyel több vezetőt használt.

Tesztelés nagyobb inputokon. A futási eredményeket a C és D inputokra a 2.3.5. illetve a 2.3.6. Táblázat tartalmazza. Az eredmények alapján most már hatékonyabbnak nyilváníthattuk a II. Algoritmust az I. Algoritmussal szemben. A II. Algoritmus 3-4-szer gyorsabban érte el az LP-optimumot annak ellenére, hogy 20-40%-kal több új oszlopot állított elő. A II. Algoritmus esetén egyetlen oszlop generálása átlagosan 5-ször rövidebb idő alatt történt, mint az I. Algoritmus esetén.
Megfigyelhetjük, hogy mindkét algoritmus ugyanolyan jó egészértékű megoldást állított elő a C, D inputokon, sőt a D input esetén elérték az optimumot.

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

2.3.5. táblázat. Futási eredmények (C Input)

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

2.3.6. táblázat. Futási eredmények (D Input)

A II. Algoritmus vizsgálata

A 1.3.3. alszakaszban azt ígértük megvizsgáljuk, hogy a korlátozás és szétválasztás során használt tesztek mennyire gyorsítják az algoritmus futását. Ennek most jött el az ideje.
A (TEST4) és (TEST5) tesztek arra szolgáltak, hogy a szubgradiens algoritmus során kapott legjobb megoldáshoz tartozó perturbált költségfüggvény segítségével kiküszöbölhessünk bizonyos részfákat a korlátozás és szétválasztás eljárásban. Az előző 4 inputon futtattuk a II. Algoritmust úgy, hogy a csúcsokban mindkét tesztet elvégeztük ($\{4,5\}$), csak az egyik tesztet végeztük el ($\{4\}$ és $\{5\}$) illetve egyik tesztet sem végeztük el ($\emptyset$). A futási eredményeket a 2.3.7. Táblázat tartalmazza.

		$\{4,5\}$	$\{4\}$	$\{5\}$	$\emptyset$
A Input	Futási idő (másodperc)	0,11	0,12	0,19	0,12
	Összes csúcs (darab)	3 102	5 688	9 273	24 638
B Input	Futási idő (másodperc)	0,26	0,27	0,39	0,25
	Összes csúcs (darab)	6 440	22 551	22 780	97 632
C Input	Futási idő (másodperc)	36,6	85,8	NaN	NaN
	Összes csúcs (darab)	1 444 458	55 941 925	NaN	NaN
D Input	Futási idő (másodperc)	39,6	295,2	NaN	NaN
	Összes csúcs (darab)	1 077 458	164 734 731	NaN	NaN

2.3.7. táblázat. Futási eredmények (Tesztek)

Mindenekelőtt megvizsgáltuk, hogy a futási időt hogyan befolyásolják az egyes tesztek kikapcsolásai. Az A és B (kisebb) inputok esetén jelentős különbség még nem volt tapasztalható. A C és D (nagyobb) inputokon esetében már a (TEST5) kikapcsolása is jelentősen megnövelte a futási időt. A C input esetében több mint 2-szer tovább tartott az oszlopgenerálás folyamata, míg a D input esetén közel 7,5-szer több időt vett igénybe az LP-optimum megtalálása. A (TEST4) kikapcsolása esetén pedig nem ért véget az algoritmus futása 10

perc alatt sem. A (TEST4) azért is olyan jelentős, mert ez igazából két tesztet foglal magába: mindkét erőforrásra vonatkozóan egyet-egyet.
Azt már az eddigi eredmények alapján is kijelenthettük, hogy a (TEST4) és a (TEST5) tesztek elengedhetetlen eszközei az eljárás hatékony, gyors működésének. A tesztek kikapcsolása esetén a hosszabb futásidőt természetesen az jelentette, hogy egy-egy oszlop generálásakor a korlátozás és szétválasztás algoritmusban nem tudtunk bizonyos részfákat kiküszöbölni. Megnéztük ezért azt is, hogy az egyes tesztek kikapcsolása mennyivel növelte meg a korlátozás és szétválasztás fában megvizsgált csúcsok számát. Az erre vonatkozó adatokat is a 2.3.7. Táblázat tartalmazza.
Már a legkisebb A input esetén is megfigyelhető az egyes tesztek hatékonysága. Mindkét teszt kikapcsolása majdnem 8-szor több csúcs ávizsgálását eredményezte az összes korlátozás és szétválasztás eljárást figyelembe véve. A B input esetén már egyetlen teszt kikapcsolása során is 3,5-szer több csúcsot vizsgáltunk át, míg mindkét teszt kikapcsolása esetén 15-ször többet. A nagyobb inputok esetén a (TEST5) kikapcsolása 38-szor illetve 163,5-szer több csúcs átvizsgálásával járt együtt.

3. fejezet

Integrált jármű és vezető ütemezés

Az előző fejezetben a jármű és a vezető ütemezési feladatokat szekvenciálisan oldottuk meg: először a járműveket ütemeztük, majd a vezetőket. A járművek ütemezése során tehát egyáltalán nem vettük figyelembe a vezetőket. Ezt a megközelítést *Bodin et al.* [5] azonban erősen kritizálta, ugyanis a vezetők összköltségei általában meghaladják a járművek összköltségeit. Ezáltal előfordulhat, hogy egy optimális jármű ütemezéshez csak nagyobb költséggel tudunk vezetőket rendelni, mint egy másik optimális (esetleg nem is optimális) jármű ütemezéshez. Nézzünk erre egy egyszerű példát.

Példa. Tegyük fel, hogy adott 3 forduló, melyeket le kell fednünk, és két depó áll rendelkezésünkre. Az első forduló 8:00-kor indul az A állomásról, és 13:00-kor érkezik a C állomásra. A második forduló a C állomásról megy a B állomásra, és 14:00-tól 15:00-ig tart. A harmadik forduló pedig 15:30-kor kezdődik a B állomáson, és 21:30-kor fejeződik be az A állomáson. Az első depóból az utazási idők az A, B illetve C állomásokra rendre 20, 50 illetve 30 perc, míg a második depóból ezen idők 50, 10 illetve 30 perc.
A beosztásokra a következő feltételeket tesszük. 1) Egy beosztás hossza legfeljebb 8 óra. 2) Egy beosztás ugyanazon depóban fejeződik be, mint ahol elkezdődött. Vagy ha mégsem a depóban kezdődik/fejeződik be, akkor (az utazási időnek megfelelő) sétálási időt kell a beosztáshoz adnunk. Minden jármű és vezető 1000 fix költséget von maga után, továbbá 1 költséget fizetünk minden olyan percért, amikor egy jármű utasok nélkül a depón kívül tartózkodik.
Először oldjuk meg a feladatot szekvenciálisan. Az optimális jármű ütemezés során az első depóból teljesítjük mindhárom fordulót. Ennek költsége $1000+20+30+30+20 = 1100$. Az ehhez tartozó optimális vezető ütemezés során 3 vezetőt használunk, tehát a szekvenciális

ütemezés összköltsége $1100 + 3000 = 4100$. Két vezetővel azért nem tudjuk teljesíteni a feladatokat, mert az első beosztás 7:40-kor kezdődik az első depóban, tehát az 1) szabály miatt legkésőbb 15:40-kor be kell fejeződnie. Mivel a B állomásról az első depóba 50 perc az út, így ez a beosztás nem tartalmazhatja a második fordulót. Hasonló a helyzet az utolsó fordulóval is.
De tekintsük azt a megoldást, amikor mindhárom fordulót a második depóból teljesítjük. Ekkor már az ehhez tartozó feladatok 2 vezetővel megoldhatók, így az összköltség $2 \cdot 1000 + 1000 + 50 + 30 + 30 + 50 = 3160$. Ezzel a költségek 23%-át megtakarítottuk! Már ezen a kis példán is látható a szekvenciális ütemezés egy hátránya, mely a fő motivációt adta az integrált jármű és vezető ütemezés alkalmazására.
Az integrált jármű és vezető ütemezés (*Integrated Vehicle and Crew Scheduling Problem*, VCSP) során adottak a teljesítendő fordulók. Célunk a fordulókhoz járműveket rendelni, a járművekhez pedig vezetőket oly módon, hogy az összköltség minimális legyen. Az egy-depós integrált jármű és vezető ütemezés (*Single-Depot Vehicle and Crew Scheduling Problem*, SDVCSP) során egyetlen depóból kell a fordulókat teljesítenünk, míg a több-depós integrált jármű és vezető ütemezésnél (*Multiple-Depot Vehicle and Crew Scheduling Problem*, MDVCSP) több depó áll rendelkezésünkre. A következő két szakaszban a SDVCSP és a MDVCSP feladatok modellezési és megoldási eljárásait mutatjuk be. Előtte azonban lerögzítjük a következő feltételeket:

- (MDVCSP) Minden jármű pontosan egy depóhoz tartozik. Innen indul és itt fejezi be a napi menetét.
- (MDVCSP) Minden vezető pontosan egy depóhoz tartozik. Kizárólag olyan járműveket vezethet, melyek az adott depóhoz tartoznak.
- Egy vezető munkarésze pontosan akkor megengedett, ha időtartama teljesíti az adott alsó-felső korlátokat (azaz nincsen rá semmilyen más megkötésünk). Egy beosztás megengedettsége kizárólag a következőktől függ: a beosztás hossza, a munkával töltött idő, a munkarészek száma.
- Ha egy jármű menettervében két egymást követő forduló közötti szünet (várakozás/üresjárat) olyan hosszú, hogy a jármű vissza tud állni a depó és újra kiállni, akkor ezt meg is teszi.
- Minden forduló pontosan két átszállási lehetőséget tartalmaz: egyet a kiinduló állomáson illetve egyet az érkezési állomáson. (Tehát egy forduló teljesítése közben nem válthatják egymást a vezetők.)
- Ha egy jármű a depón kívül tartózkodik, akkor hozzá vezetőt kell rendelnünk. (Tehát ha a jármű egy állomáson várakozik, akkor is jelen kell lennie egy vezetőnek.)

- Egy vezető csak akkor válthat járművet, ha a szünetét megkezdi/befejezi.

3.1. SDVCSP

A jármű és vezető ütemezésre *Patrikalakis* és *Xerocostas* [41] egy heurisztikus módszereken alapuló integrációt adtak. Az első teljesen integrált modellt és megoldási módszert *Freling* [14] készítette el. Freling egészértékű modellje három részből állt: egy jármű ütemezési részből, egy vezető ütemezési részből, illetve az ezeket összekapcsoló feltételekből. A megoldás során Lagrange-relaxáción alapuló oszlopgenerálást alkalmazott. Freling megközelítése számos további cikk alapját adta [15] [26] [27]. Freling modelljének hátterében kapcsolat alapú megközelítés áll. A továbbiakban azonban egy tér-idő hálózaton alapuló modellel foglalkozunk. A megoldási módszer lényegében mindkét esetben ugyanaz, de utóbbi esetben lényegesen kevesebb feltételünk lesz.

3.1.1. Modellezés

Legyen $\mathcal{T} = \{1, \ldots, n\}$ a fordulók halmaza, és $D = (V, A)$ a hozzájuk tartozó tér-idő hálózat. Jelölje $\tilde{A} \subset A$ azon élek halmazát, amelyeken egyszerre kell járművet és vezetőt is használnunk (ha használunk). Tehát $\tilde{A}$ tulajdonképpen a depón kívüli élekből áll. Legyen $c : A \to \mathbb{R}_+$ a járművekhez tartozó költségfüggvény, mely az $\tilde{A}$-beli éleken tipikusan az utazási/várakozási idők függvénye, a cirkulációs-élen pedig a járművekre vonatkozó fix költséget jelenti. Adott még a vezetőkre vonatkozó $g : A \to \mathbb{R}_+$ költségfüggvény is, de a modellben ezt közvetlenül nem használjuk. Az $u : A \to \mathbb{Z}_\oplus$ kapacitásfüggvény értéke a fordulókon 1, egy kiállás élen pedig annyi, ahány forduló-él tartozik hozzá. Minden más élen a kapacitás a depóból rendelkezésre álló járművek száma. Jelölje K a megengedett beosztások halmazát, illetve $K(ij) \subset K$ az $ij \in \tilde{A}$ élt tartalmazó beosztások halmazát. Legyen $\tilde{A}(k) \subset \tilde{A}$ a $k \in K$ beosztásban szereplő élek halmaza. Egy $k \in K$ beosztás költségét f_k-val jelöljük.

Kétféle döntési változót vezetünk be. Jelentse y_{ij} az ij élen áthaladó járművek számát, illetve x_k azt, hogy a $k \in K$ beosztáshoz rendelünk-e vezetőt vagy sem. A SDVCSP feladat

a következő egészértékű feladatként írható fel:

$$\min \sum_{ij \in A} y_{ij} c_{ij} + \sum_{k \in K} x_k f_k \tag{3.1}$$

$$\sum_{j:ji \in A} y_{ji} - \sum_{j:ij \in A} y_{ij} = 0 \qquad \forall\, i \in V \tag{3.2}$$

$$\sum_{k \in K(ij)} x_k - y_{ij} = 0 \qquad \forall\, ij \in \tilde{A} \tag{3.3}$$

$$0 \leq y_{ij} \leq u_{ij},\ y_{ij} \in \mathbb{N} \qquad \forall\, ij \in A \tag{3.4}$$

$$x_k \in \{0, 1\} \qquad \forall\, k \in K \tag{3.5}$$

A (3.1) célfüggvény a járművek és a vezetők összköltségét minimalizálja. A (3.2) és (3.4) feltételek a jármű ütemezéshez tartozó áram-feltételek. A (3.3) feltételek kötik össze a jármű és vezető ütemezési részeket: minden egyes élt pontosan annyi beosztással kell fednünk, mint ahány jármű áthalad rajta.

3.1.2. Megoldási eljárás

A SDVCSP feladatot Lagrange relaxáción alapuló oszlopgenerálással oldjuk meg. Az eljárást a 3.1. Algoritmus tartalmazza.

Algoritmus 3.1 Megoldási eljárás a SDVCSP problémára

1: *Inicializálás*
Meghatározunk egy K' kezdeti beosztás készletet. A t lépésszámlálót 0-ra állítjuk.

2: *Redukált feladat megoldása*
Megoldjuk a Lagrange duális feladatot az adott oszlopkészlet mellett. A kapott alsó korlátot és a multiplikátorokat eltároljuk.

3: *Árazási feladat megoldása*
Negatív redukált költségű oszlop(ok)at keresünk. Ezeket hozzávesszük a jelenlegi oszlopkészlethez.

4: *Leállási kritériumok ellenőrzése*
Az 5. lépéssel folytatjuk, ha az alábbiak valamelyike teljesül:
Nem találtunk negatív redukált költségű oszlopot
$t > t_{\max}$
A célfüggvény értéke nem javult jelentősen az utolsó iterációk alatt
Egyébként t értékét eggyel növeljük és a 2. lépéstől folytatjuk.

5: *Megengedett megoldás előállítása*
A legjobb megoldást alapul véve, állítjuk elő a SDVCSP feladat egy megengedett megoldását.

3.1.3. A fő feladat

Először a (3.3) egyenlőséges feltételeket nagyobb vagy egyenlő ($\geq$) feltételekre cseréljük. Másodszor, az immár egyenlőtlenséges (3.3) feltételeket $\mu \geq 0$ multiplikátorokkal dualizáljuk. A Lagrange feladat ekkor a következő:

$$\Phi(\mu) = \Phi_x(\mu) + \Phi_y(\mu)$$

ahol

$$\begin{aligned} \Phi_x(\mu) = \min \sum_{k \in K} x_k \bar{f}_k & \\ x_k \in \{0,1\} & \qquad \forall\, k \in K \end{aligned}$$

illetve

$$\begin{aligned} \Phi_y(\mu) = \min \sum_{ij \in A} y_{ij} \bar{c}_{ij} & \\ \sum_{j:ji \in A} y_{ji} - \sum_{j:ij \in A} y_{ij} = 0 & \qquad \forall\, i \in V \\ 0 \leq y_{ij} \leq u_{ij},\ y_{ij} \in \mathbb{N} & \qquad \forall\, ij \in A \end{aligned}$$

A redukált költségek:

$$\bar{f}_k = f_k - \sum_{ij \in \tilde{A}(k)} \mu_{ij}$$

illetve

$$\bar{c}_{ij} = \begin{cases} c_{ij} + \mu_{ij} & \text{ha } ij \in \tilde{A} \\ c_{ij} & \text{különben} \end{cases}$$

Induljunk ki egy K' kezdeti beosztás készletből, mellyel létezik a (3.1)-(3.5) feladatnak megengedett megoldása (az egyenlőtlenséges feltételek mellett). A $\max_{\mu} \Phi(\mu)$ Lagrange duális feladatot szubgradiens algoritmussal oldjuk meg. Adott μ multiplikátorok és adott K' oszlopkészlet mellett a $\Phi_y(\mu)$ feladat egy minimális $\bar{c}$-költségű megengedett áram keresése, a $\Phi_x(\mu)$ feladat esetén pedig az x_k értékét 1-re állítjuk, ha a $k \in K'$ oszlop $\bar{f}_k$ költsége nempozitív, egyébként pedig x_k értéke 0 lesz.

3.1.4. Árazási feladat

Miután az adott oszlopkészlettel megoldottuk a Lagrange duális feladatot, a kapott multiplikátorokat használjuk fel, hogy negatív redukált költségű oszlopo(ka)t keressünk. (A 1.2.4. alszakaszban leírtak miatt először egy multiplikátor javítási eljárást alkalmazunk). Mivel a végleges jármű blokkok még nem ismertek, ezért a munkarészek száma nagyon nagy lehet. *Freling* [16] és *Huisman* [26] két-fázisú oszlopgeneráló eljárást javasoltak. Az első lépésben

a munkarészek generálása történt, és a második fázisban készítették el a megengedett beosztásokat.
Huisman a második fázisban egyszerűen összepárosította a munkarészeket és kiválasztott bizonyos negatív költségű megengedett beosztásokat. A későbbiekben mi is ezt tesszük. Megjegyezzük azonban, hogy ez az eljárás akkor hatékony, ha egy beosztás legfeljebb 2 vagy 3 munkarészből áll. Freling a második fázisban egy beosztás generáló hálózatot épített az első fázis során generált munkarészekből. A legolcsóbb megengedett beosztás megtalálása egy erőforrás korlátos legrövidebb út keresésére vezetett, melyet dinamikus programozással oldott meg. Mindkét eljárás előnye, hogy egy iteráció alatt több új oszlop is hozzávehető a jelenlegi oszlopkészlethez.
Freling és Huisman is kapcsolat alapú megközelítést használtak a VCSP feladatra, és Freling a beosztás generáló hálózatot is kapcsolat alapú hálózatként építette fel. Ezzel szemben *Steinzen* [42] már a generáló hálózatot is tér-idő hálózatos megközelítéssel építette fel (az MDVCSP feladtra).

3.1.5. Megengedett megoldás előállítása

Mivel bizonyos feltételeket relaxáltunk, ezért az a megoldás amit az eddigi fázisok végeztével kaptunk, nem feltétlenül megengedett. (A szubgradiens algoritmus során esetleg előállítottunk néha megengedett megoldásokat, azok viszont 'messze' lehetnek az optimumtól.) Ezért olyan megengedett megoldást szeretnénk előállítani, amely nem növeli túlzottan a célfüggvény értékét.
Freling et al. [15] és *Huisman* [26] a legjobb alsó korláthoz tartozó jármű ütemezést lerögzítették, és erre megoldották a vezető ütemezés feladatot.

3.1.6. Futási eredményeink

Az alábbi alszakaszban a SDVCSP feladat megoldására elkészített implementációnk futási eredményeit mutatjuk be. A tesztelést 1.86 GHz-es Intel Pentium Dual CPU T2390 processzorral, 2GB memóriával, Windows 7 operációs rendszeren végeztük. Az implementáció C++ nyelven történt a LEMON [34] könyvtár felhasználásával. Az egészértékű feladatok megoldására a FICO Xpress Optimization Suite 7.3.0 verzióját használtuk.

Huisman adatok

A teszteléshez *Huisman* véletlen generált példáit [29] [26] használtuk, melyeket mások is széleskörben alkalmaznak [27] [42] [43]. A példák eredetileg a több-depós feladathoz készültek, de az egy-depós esetben is felhasználhatók. A példák két osztályba (A és B) vannak sorolva. Az A osztályban szereplő fordulók rövidebbek, ezért a jármű blokkok és ezzel együtt

a beosztások is több fordulót fednek le, mint a B osztályban szereplő példák esetén. Bizonyos értelemben tehát a A osztályban szereplő példák bonyolultabbak, mint a B osztályban szereplők.

Paraméterek

A probléma paramatéreinek megválasztásánál törekedtünk az eddigi irodalomban szereplő, de főként *Steinzen et al.* [43] által használt paramétereket alkalmazni.
A célfüggvény a járművek és vezetők összköltségének minimalizálása, melyen belül a fő cél a járművek és vezetők számának minimalizálása. Minden jármű és minden vezető (beosztás) használata 1000 fix költséget von maga után. Ezen felül, ha egy jármű a depón kívül tartózkodik, akkor percenként 1 költséget fizetünk utána, továbbá egy vezető minden munkával töltött perce után 0.1 költséget fizetünk.
4 különböző beosztás-típust engedünk meg. Az ezekre vonatkozó adatokat a 3.1.1. Táblázat tartalmazza.

Típus	Early		Day		Late		Split	
	min	max	min	max	min	max	min	max
Kezdési idő			8:00		13:15			
Befejezési idő		16:30		18:14				19:30
Munkrész hossza	0:30	5:00	0:30	5:00	0:30	5:00	0:30	5:00
Szünet hossza	0:45		0:45		0:45		1:30	
Beosztás hossza		9:45		9:45		9:45		12:00
Munkával töltött idő		9:00		9:00		9:00		9:00

3.1.1. táblázat. A beosztás-típusok adatai

Inicializálás. A kezdeti oszlopkészletünk nem megengedett beosztásokat tartalmazott, így ezekre nagy költséget adtunk meg, és az oszlopgenerálás folyamata alatt, amint lehetett töröltük őket.

Redukált feladat. A szubgradiens algoritmus során minden iterációban legfeljebb $100|\mathcal{T}|$ iterációt hajtottunk végre.

Árazási feladat. Az árazási feladat megoldása során az első fáziban legeneráltuk a munkarészeket. A második fáziban a legenerált munkarészeket párosítottuk össze, és ellenőriztük, hogy megfelelnek-e valamely beosztás-típus követelményeinek. Egy oszlopgenerálási fázisban minden munkarészhez legfeljebb 1 új beosztást vettünk hozzá az eddigi készletünkhöz.

Leállási kritériumok. Az oszlopgenerálás során legfeljebb $|\mathcal{T}|$ iterációt hajtottunk végre. Leálltunk, ha nem találtunk több negatív redukált költségű oszlopot, illetve ha az utolsó 10 iteráció alatt az alsó korlát kevesebb, mint $0,2\%$-kal javult.

Megengedett megoldás. Az oszlopgenerálási fázis leállását követően az adott oszlopkészlet mellett megoldottuk az egészértékű programot.

Futás eredmények

A 3.1.2. és 3.1.3. Táblázatok a *Huisman* [29] 80 fordulós B osztálybeli példáin történő futási eredményeit tartalmazzák az első üresjárat-távolságot tartalmazó fájllal kombinálva. Minden példánál mind a 4 depóhoz 1-1 inputot készítettünk. Az első három oszlop az elkészített tér-idő hálózat fontosabb paramétereit tartalmazza: csúcsok száma (#Csúcsok), élek szám (#Élek) illetve a fontos élek száma (#Fontos élek), melyek alatt az $\tilde{A}$-beli éleket értjük. Ezen paraméterek szerint nincs nagy különbség a hálózatok között.
A következő három oszlop az oszlopgenerálási fázisra vonatkozik: megmértük az oszlopgenerálás idejét (CPU (o.g.)) (másodperc), megnéztük az iterációk számát (#Iterációk), illetve azt, hogy az iterációk alatt összesen hány beosztást generáltunk le (#Beosztások). Az oszlopgenerálási folyamat átlagosan 260,7 másodpercig tartott, és átlagosan 2543,7 beosztást generáltunk le.
Az utolsó három oszlop tartalmazza a megengedett megoldás keresésére vonatkozó adatokat: megmértük az egészértékű program megoldásának idejét (CPU (mip)) (másodperc), és megnéztük, hogy a megtalált megoldás során hány járművet (#Járművek) illetve vezetőt (#Vezetőt) használtunk. A végső oszlopkészlettel megoldott feladat átlagosan 143 másodpercig tartott, de nagy szórás figyelhető meg az értékek között.

#Csúcsok	#Élek	#Fontos élek	CPU (o.g.)	#Iterációk	#Beosztások	CPU (mip)	#Járművek	#Vezetők
315	672	517	372,4	53	2947	23,1	11	22
313	663	510	308,4	41	2618	13,9	11	21
293	643	510	222,1	36	2517	11,4	11	22
316	669	513	312,5	47	2647	24,0	11	22
306	659	513	278,4	48	2932	476,2	10	19
306	652	506	225,4	43	2630	729,7	10	19
313	660	507	203,4	38	2503	16,6	10	18
295	647	512	260,6	46	2856	482,7	10	19
313	670	518	269,1	50	2531	45,2	10	21
315	660	506	225,2	50	2404	12,6	10	21
310	657	508	286,3	47	2396	0,8	10	20
300	657	518	205,2	47	2402	239,8	10	21
305	657	513	275,3	46	2632	60,9	12	24
320	665	506	249,7	46	2491	31,8	12	23
311	658	508	257,9	38	2311	10,6	12	23
314	667	515	342,8	51	2500	115,6	12	23
316	681	526	270,5	43	2764	1118,4	9	19
310	667	517	249,6	47	2929	91,3	9	19
311	672	522	302,0	49	2700	83,6	9	19
311	675	526	289,7	44	2788	597,9	9	19

3.1.2. táblázat. SDVCSP: 80 forduló. Futási eredmények.

#Csúcsok	#Élek	#Fontos élek	CPU (o.g.)	#Iterációk	#Beosztások	CPU (mip)	#Járművek	#Vezetők
308	664	516	228,3	48	2607	11,5	11	21
316	667	511	309,7	46	2497	0,943	11	20
311	664	513	253,1	41	2457	2,5	11	20
308	666	518	259,6	45	2579	214,6	11	21
304	662	519	320,5	49	2651	371,6	11	21
299	655	517	249,2	49	2739	171,3	11	21
311	664	514	284,4	37	2402	14,4	11	21
316	676	521	321,5	53	2695	432,1	11	22
317	675	519	203,4	38	2292	133,0	14	28
307	661	515	273,4	45	2383	17,9	14	28
295	647	514	197,1	32	2135	8,7	14	28
313	671	519	291,2	49	2554	75,4	14	28
300	649	502	206,0	43	2361	13,9	15	28
304	646	496	182,3	43	2283	9,4	15	27
291	630	492	240,0	39	2361	11,7	15	28
303	650	500	189,6	42	2255	12,6	15	28

3.1.3. táblázat. SDVCSP: 80 forduló. Futási eredmények.

3.2. MDVCSP

Az első modellt *Huisman* [26] majd *Huisman et al.* [27] készítették el a MDVCSP feladatra. (Előtte *Gaffi* és *Nonato* [19] is készített egy több-depós integrált jármű és vezető ütemezési modellt egy távolsági közlekedési feladatra, de a szorosabb megkötések miatt a probléma eltért az általános esettől). A szerzők *Freling* [14] egy-depós modellje alapján készítették el két saját modelljüket. A megoldási eljárás szintén *Freling* [14] módszerén alapult.
Mesquita és *Paias* [37] is két modellt készített *Huisman* [26] modellje alapján, bár ők megengedték, hogy egy vezető bármely depóból vezethessen járművet, és bármikor válthasson járművet. A jármű ütemezés részt több-termékes folyam feladatként írták fel, a vezető ütemezés részt pedig halmaz fedési/partícionálási feladatként. A megoldás során oszlopgenerálást alkalmaztak, ahol az árazási feladat erőforrás korlátos legrövidebb út keresésére vezett vissza. Ezt dinamikus programozási eljárással oldották meg, végül egy MIP-megoldó segítségével állítottak elő megengedett megoldást.
A MDVCSP feladatra az első tér-idő hálózatos megközelítést *Gintner* [23], majd *Steinzen* [42] illetve *Steinzen et al.* [43] alkalmazták. A következőkben *Steinzen et al.* [43] modelljét ismertetjük.

3.2.1. Modellezés

Legyen $\mathcal{T} = \{1, \ldots, n\}$ a fordulók halmaza, $\mathcal{D} = \{1, \ldots, m\}$ pedig a depóké. Legyen $D^d = (V^d, A^d)$ a $d \in \mathcal{D}$ depóhoz tartozó tér-idő hálózat. Jelölje $\tilde{A}^d \subset A^d$ azon élek halmazát, amelyeken egyszerre kell járművet és vezetőt is használnunk (ha használunk). Legyenek $c^d : A^d \to \mathbb{R}_+$ a járművekhez tartozó, $g^d : A^d \to \mathbb{R}_+$ a vezetőkre vonatkozó költségfüggvények. Az $u^d : A^d \to \mathbb{Z}_\oplus$ kapacitásfüggvény értéke a fordulókon 1, a kiállás éleken pedig annyi, ahány forduló-él tartozik hozzájuk. Minden más élen a kapacitás a d depóból rendelkezésre álló járművek száma. A $d \in \mathcal{D}$ depóra jelölje K^d a megengedett beosztások halmazát, $K^d(ij) \subset K^d$ az $ij \in \tilde{A}^d$ élt tartalmazó beosztások halmazát, illetve $\tilde{A}^d(k) \subset \tilde{A}^d$ a $k \in K^d$ beosztásban szereplő élek halmazát. Egy $k \in K^d$ beosztás költségét f_k^d-val jelöljük. $A^d(t)$ jelölje a $t \in \mathcal{T}$ fordulót reprezentáló élt a d depóban ($A^d(t)$ üres halmaz, ha a t forduló nem teljesíthető a d depóból).
Kétféle döntési változót vezetünk be. Jelentse y_{ij}^d az $ij \in A^d$ élen áthaladó járművek számát, illetve x_k^d azt, hogy a $k \in K^d$ beosztáshoz rendelünk-e vezetőt vagy sem. A MDVCSP feladat

a következő egészértékű feladatként írható fel:

$$\min \sum_{d\in\mathcal{D}} \sum_{ij\in A^d} y_{ij}^d c_{ij}^d + \sum_{d\in\mathcal{D}} \sum_{k\in K^d} x_k^d f_k^d \tag{3.6}$$

$$\sum_{d\in\mathcal{D}} \sum_{ij\in A^d(t)} y_{ij}^d = 1 \qquad \forall\, t\in\mathcal{T} \tag{3.7}$$

$$\sum_{j:ji\in A^d} y_{ji}^d - \sum_{j:ij\in A^d} y_{ij}^d = 0 \qquad \forall\, d\in\mathcal{D},\ \forall\, i\in V^d \tag{3.8}$$

$$\sum_{k\in K^d(ij)} x_k^d - y_{ij}^d = 0 \qquad \forall\, d\in\mathcal{D},\ \forall\, ij\in \tilde{A}^d \tag{3.9}$$

$$0\le y_{ij}^d \le u_{ij}^d,\ y_{ij}^d\in\mathbb{N} \qquad \forall\, d\in\mathcal{D},\ \forall\, ij\in A^d \tag{3.10}$$

$$x_k^d\in\{0,1\} \qquad \forall\, d\in\mathcal{D},\ \forall\, k\in K^d \tag{3.11}$$

A (3.6) célfüggvény a járművek és a vezetők összköltségét minimalizája. A (3.8) és (3.10) feltételek a jármű ütemezéshez tartozó áram-feltételek. A (3.9) feltételek kapcsolják össze a jármű és a vezető ütemezési részeket. A (3.7) feltételek teremtik meg a depók közötti kapcsolatokat: minden egyes fordulót pontosan egy depóból kell fednünk.

3.2.2. Megoldási eljárás

A következőkben *Steinzen et al.* [43] megoldási módszerét vázoljuk. Az eljárás alapja hasonló az előző szakaszban bemutatotthoz (3.1. Algoritmus).
Először a (3.9) egyenlőséges feltételeket nagyobb vagy egyenlő ($\geq$) feltételekre cseréljük. Másodszor, az immár egyenlőtlenséges (3.9) feltételeket $\mu^d \geq 0$ multiplikátorokkal dualizáljuk. Ezzel a Lagrange feladat ismét két különálló feladatra bomlik, azonban most az egyik egy MDVSP feladat, mely NP-nehéz. Ezért a (3.7) feltételeket is dualizáljuk a π multiplikátorokkal. A Lagrange feladatunk így a következő:

$$\Phi(\mu,\pi) = \Phi_x(\mu) + \Phi_y(\mu,\pi) + \sum_{i\in\mathcal{T}} \pi_t$$

ahol

$$\Phi_x(\mu) = \min \sum_{d\in\mathcal{D}} \sum_{k\in K^d} x_k^d \bar{f}_k^d$$
$$x_k^d\in\{0,1\} \qquad \forall\, d\in\mathcal{D},\ \forall\, k\in K^d$$

illetve

$$\Phi_y(\mu,\pi) = \min \sum_{d\in\mathcal{D}} \sum_{ij\in A^d} y_{ij}^d \bar{c}_{ij}^d$$
$$\sum_{j:ji\in A^d} y_{ji}^d - \sum_{j:ij\in A^d} y_{ij}^d = 0 \qquad \forall\, d\in\mathcal{D},\ \forall\, i\in V^d$$
$$0\le y_{ij}^d \le u_{ij}^d,\ y_{ij}^d\in\mathbb{N} \qquad \forall\, d\in\mathcal{D},\ \forall\, ij\in A^d$$

A redukált költségek:

$$\bar{f}_k^d = f_k^d - \sum_{ij \in \tilde{A}^d(k)} \mu_{ij}^d$$

illetve

$$\bar{c}_{ij}^d = \begin{cases} c_{ij}^d + \mu_{ij}^d - \pi_t & \text{ha } ij \in \tilde{A}^d, \text{ és létezik } t \in \mathcal{T}, \text{ hogy } ij \in A^d(t) \\ c_{ij}^d + \mu_{ij}^d & \text{ha } ij \in \tilde{A}^d, \text{ és nem létezik } t \in \mathcal{T}, \text{ hogy } ij \in A^d(t) \\ c_{ij}^d & \text{különben} \end{cases}$$

Adott K' beosztás-készletre a $\max_{\mu,\pi} \Phi(\mu, \pi)$ Lagrange duális feladatot szubgradiens módszerrel oldjuk meg. Adott K' oszlopkészlet és adott μ, π multiplikátorok mellett a $\Phi_x(\mu)$ feladat a nempozitív redukált költségű oszlopok kiválasztása. $\Phi_y(\mu, \pi)$ pedig $|\mathcal{D}|$ darab minimális költségű áram-feladatot jelent.

Steinzen et al. [43] megoldották a MDVSP feladatot *Kliewer et al.* [33] módszerével, majd a kapott jármű ütemezéshez depónként megoldotta a CSP feladatot. A vezető ütemezésből kapott beosztások adták a K' kiinduló oszlopkészletet a MDVCSP feladathoz. A Lagrange feladatot egy kissé módosított, legfeljebb 1000 iteráción keresztül futtatott szubgradiens algoritmussal oldották meg. A legjobb megoldáshoz tartozó multiplikátorokat használták fel, hogy új oszlopokat állítsanak elő. Az oszlopgeneráláshoz *Freling* [16] és *Huisman* [26] módszeréhez hasonlóan két-fázisú eljárást alkalmaztak. Első lépésben legenerálták a munkarészeket, majd ezekből (minden egyes beosztás-típushoz) egy tér-idő hálózatot készítettek el. Az árazási feladat erőforrás korlátos legrövidebb út keresésére vezetett vissza az adott hálózatokban, melyet 4-fázisú dinamikus programozási eljárással oldottak meg. Az eddig kapott megoldásból (oszlopkészletből) kiindulva állítottak elő megengedett megoldást a következőképpen. Már csak a (3.9) feltételeket, mint egyenlőséges feltételeket relaxálták. A Lagrange feladatban szereplő MDVSP feladatot ismét *Kliewer et al.* [33] módszerével oldották meg. A megoldás megengedettségét egy heurisztikus árazás és szétválasztás eljárással biztosították, ahol az oszlopgenerálás során bizony y_{ij}^d változók értékét lerögzítették.

Összefoglalás, kitekintés

Az 1. Fejezetben bemutattuk a dolgozatban felmerülő, alapvető kombinatorikus feladatokat és módszereket. A fejezet végén részletesebben foglalkoztunk az erőforrás korlátos legrövidebb út feladattal.
A 2. Fejezetben a szekvenciális jármű és vezető ütemezés feladatait és megoldási módszereit ismertettük. A fejezet végén egy speciális vezető ütemezési feladattal foglalkoztunk, és bemutattuk a megoldására elkészített implementációnk futási eredményeit.
A 3. Fejezetben az integrált jármű és vezető ütemezés feladataival foglalkoztunk. Bemutattuk a különböző feladatokat és a megoldási eljárásokat. Az egy-depós ütemezési feladatnál ismertettük az elkészített implementációnk futási eredményeit.
A témakör még mindig számos kihívást tartalmaz. Továbbra is fő kutatási téma a minél hatékonyabb eljárások kifejlesztése. Ebben az erőforrás korlátos út problémának is nagy szerep jut, hiszen a legtöbb ütemezési feladat során valamilyen formában felmerült.

Irodalomjegyzék

[1] Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W., Vance, P. H. (1998), *Branch-and-price: Column generation for solving huge integer programs.* Operations research, 46(3), 316-329

[2] Beasley, J. E. and Christofides, N. (1989), *An algorithm for the resource constrained shortest path problem.* Networks, 19: 379-394. doi: 10.1002/net.3230190402

[3] Bertossi, A. A., Carraresi, P. and Gallo, G. (1987), *On some matching problems arising in vehicle scheduling models.* Networks, 17: 271-281. doi: 10.1002/net.3230170303

[4] Békési, J., Brodnik, A., Krész, M., Pash, D. (2009) *An Integrated Framework for Bus Logistics Management: Case Studies* (pp. 389-411). Physica-Verlag HD.

[5] Bodin, L., Golden, B., Assad, A., Ball, M. (1981), *The state of the art in the routing and scheduling of vehicles and crews* (No. UMTA-MD-11-0004-81-2Final Rpt.).

[6] Carraresi, P., Nonato, M., Girardi, L. (1995), *Network models, Lagrangean relaxation and subgradients bundle approach in crew scheduling problems.* In Computer-Aided Transit Scheduling (pp. 188-212). Springer Berlin Heidelberg.

[7] Dantzig, G. B., Wolfe, P. (1960), *Decomposition principle for linear programs.* Operations research, 8(1), 101-111.

[8] Desrochers, M., Soumis, F., Université de Montréal. Centre de recherche sur les transports. (1986), *A generalized permanent labelling algorithm for the shortest path problem with time windows.* Montréal: Université de Montréal, Centre de recherche sur les transports.

[9] Desrochers, M., Soumis, F. (1989), *A column generation approach to the urban transit crew scheduling problem.* Transportation Science, 23(1), 1-13.

[10] Desrochers, M., Desrosiers, J., Solomon, M. (1992), *A new optimization algorithm for the vehicle routing problem with time windows.* Operations research, 40(2), 342-354.

[11] Desrochers, M., Gilbert, J., Sauvé, M., Soumis, F. (1992), *CREW-OPT: Subproblem modeling in a column generation approach to urban crew scheduling* (pp. 395-406). Springer Berlin Heidelberg.

[12] Dumitrescu, I., Boland, N. (2003), *Improved preprocessing, labeling and scaling algorithms for the Weight-Constrained Shortest Path Problem.* Networks, 42(3), 135-153.

[13] Frank, A. *Operációkutatás.* Egyetemi jegyzet, 2011. http://www.cs.elte.hu/ frank/jegyzet/opkut/ulin.2011.pdf

[14] Freling, R. (1997), *Models and Techniques for Integrating Vehicle and Crew Scheduling.* Tinbergen Institute research series.

[15] Freling, R., Huisman, D., Wagelmans, A. P. (2003), *Models and algorithms for integration of vehicle and crew scheduling.* Journal of Scheduling, 6(1), 63-85.

[16] Freling, R., Wagelmans, A. P., Paixao, J. M. P. (2001), *Models and algorithms for single-depot vehicle scheduling.* Transportation Science, 35(2), 165-180.

[17] Fischetti, M., Martello, S., Toth, P. (1987), *The fixed job schedule problem with spread-time constraints.* Operations Research, 35(6), 849-858.

[18] Fischetti, M., Martello, S., Toth, P. (1989), *The fixed job schedule problem with working-time constraints.* Operations Research, 37(3), 395-403.

[19] Gaffi, A., Nonato, M. (1999), *An integrated approach to ex-urban crew and vehicle scheduling.* In Computer-Aided Transit Scheduling (pp. 103-128). Springer Berlin Heidelberg.

[20] Garcia, R, (2009). *Resource constrained shortest paths and extensions.* ProQuest.

[21] Garey, M.R., Johnson, D.S. 1979, *Computers and Intractability: a Guide to the Theory of NP-Completeness.* Freeman, San Francisco.

[22] Geoffrion, A. M. (1974), *Lagrangean relaxation for integer programming. In Approaches to Integer Programming (pp. 82-114).* Springer Berlin Heidelberg.

[23] Gintner, V. (2007), *Integrierte Umlauf- und Dienstplanung im ÖPNV.* PhD thesis, University of Paderborn, Germany.

[24] Handler, G. Y., Zang, I. (1980) *A dual algorithm for the constrained shortest path problem.* Networks, 10(4), 293-309.

[25] Hane, C. A., Barnhart, C., Johnson, E. L., Marsten, R. E., Nemhauser, G. L., Sigismondi, G. (1995), *The fleet assignment problem: solving a large-scale integer program.* Mathematical Programming, 70(1-3), 211-232.

[26] Huisman, D. (2004), *Integrated and dynamic vehicle and crew scheduling.* Erasmus University Rotterdam.

[27] Huisman, D., Freling, R., Wagelmans, A. P. (2005), *Multiple-depot integrated vehicle and crew scheduling.* Transportation Science, 39(4), 491-502.

[28] Huisman, D., Jans, R., Peeters, M., Wagelmans, A. P. (2005), *Combining column generation and Lagrangian relaxation* (pp. 247-270). Springer US.

[29] *Huisman instances*, http://people.few.eur.nl/huisman/instances.htm

[30] Jaffe, J. M. (1984), *Algorithms for finding paths with multiple constraints.* Networks, 14: 95-116. doi: 10.1002/net.3230140109

[31] Joksch, H. C. (1966), *The shortest route problem with constraints (Shortest route problem with constraint, using set of nodes).* Journal of Mathematical analysis and applications, 14, 191-197.

[32] Kliewer, N., Mellouli, T., Suhl, L. (2002), *A new solution model for multi-depot multi-vehicle-type vehicle scheduling in (sub) urban public transport.* In Proceedings of the 13th Mini-EURO Conference and the 9th meeting of the EURO working group on transportation, Politechnic of Bari.

[33] Kliewer, N., Mellouli, T., Suhl, L. (2006), *A time-space network based exact optimization model for multi-depot bus scheduling.* European journal of operational research, 175(3), 1616-1627.

[34] LEMON: *A C++ Library for Efficient Modelling and Optimization in Networks*, http://lemon.cs.elte.hu

[35] Löbel, A. (1998), *Vehicle scheduling in public transit and Lagrangean pricing.* Management Science, 44(12-Part-1), 1637-1649.

[36] Mehlhorn, K., Ziegelmann, M. (2000), *Resource constrained shortest paths.* In Algorithms-ESA 2000 (pp. 326-337). Springer Berlin Heidelberg.

[37] Mesquita, M., Paias, A. (2008), *Set partitioning/covering-based approaches for the integrated vehicle and crew scheduling problem.* Computers & Operations Research, 35(5), 1562-1575.

[38] Mesquita, M., Paixao, J. (1999), *Exact algorithms for the multi-depot vehicle scheduling problem based on multicommodity network flow type formulations.* In Computer-aided transit scheduling (pp. 221-243). Springer Berlin Heidelberg.

[39] Nemhauser, G. L., Wolsey, L. A. (1988), *Integer and combinatorial optimization.* Vol. 18. New York: Wiley.

[40] Paixao, J. P., Branco, I. M. (1987), *A quasi-assignment algorithm for bus scheduling.* Networks, 17(3), 249-269.

[41] Patrikalakis, I., Xerocostas, D. (1992), *A new decomposition scheme of the urban public transport scheduling problem.* In Computer-Aided Transit Scheduling (pp. 407-425). Springer Berlin Heidelberg.

[42] Steinzen, I. (2007), *Topics in integrated vehicle and crew scheduling in public transport.* Unpublished doctoral dissertation, University of Paderborn, Paderborn, Germany.

[43] Steinzen, I., Gintner, V., Suhl, L., Kliewer, N. (2010), *A time-space network approach for the integrated vehicle-and crew-scheduling problem with multiple depots.* Transportation Science, 44(3), 367-382.