Eötvös Lóránd Tudomány Egyetem

Bsc szakdolgozat

Egészértékű programozási feladatok a közlekedés-tervezésben

Czifra Domonkos

Témvezető:
Jüttner Alpár
Operációkutatás Tanszék

Budapest,2016

Köszönetnyilvánítás

Köszönöm témavezetőm, Jüttner Alpár segítségét, aki hasznos tanácsaival jó irányba terelte e dolgozatot, és akihez mindig fordulhattam kérdéseimmel.

Valamint köszönöm szüleimnek, testvéreimnek, barátaimnak a támogatást, biztatást.

Tartalomjegyzék

Bevezetés

Az emberiség történelme során mindig is törekedett a meglévő eszközök fejlesztésére, és a meglévőkből a legjobbak kihozására, OPTIMALIZÁSÁRA. Azonban a közlekedésben talán csak a 20.században jelent meg akkora forgalom a közlekedési útvonalakon, hogy legyen értelme érdemben optimalizálásról beszélni.

Az egyik legnagyobb igény a közlekedés optimalizálásra a légitársaságoktól érkezett: hiszen az erős verseny, és az erőforrások relatív magas költsége miatt egy ügyes optimalizáláson akár a profitabilitásuk múlhat.

A légitársaságok két legfontosabb feladata az erőforrás elosztás: repülőgépek (Vehicle-), és a személyzet elosztása (Crew-sheduling). Jóllehet ekkor már létezett az a matematikai apparátus, melyekkel modellezni, és megoldani lehetett a problémákat, de látszólag kis problémák megoldásának kiszámítása az ismert eszközökkel "végtelen" sok időbe telt volna, így új módszereket kellett kidolgozni, hogy reális időben megoldják a problémákat.

A 21.században az angolszász területeken a közösségi közlekedés egyes részeit privatizálták, így profitorientált vasúttársaságok, busztársaságok is optimalizálni szerették volna járataikat, már a kezdeti tervezéstől (Network planing) a legkisebb részletekig (Crew-sheduling).

A dolgozat első része a főbb tervezési lépésekkel foglalkozik, mint a hálózattervezés, járattervezés, menetrendtervezés, erőforrás-, illetve személyzetelosztás. A második rész pedig ezek általános megoldását tárgyalja, kezdve a legegyszerűbb algoritmusoktól (Branch and Bound) az egyszerű heurisztkákon át az összetettebb heurisztikákig, amit már a gyakorlatban is alkalmaznak (lásd [1]).

1. Egy közlekedési hálózat tervezése

1.1. A tervezés folyamata

A közlekedési rendszer (pl busz- illletve vasúthálózat) megtervezésénél alapvetően két fázisról beszélhetünk:

- stratégiai (strategic)
- illetve működtetési (operational) tervezésről.

A stratégiai tervezés általában több időt vesz igénybe, a döntések hosszú időre szólnak. Ebből adódóan nehezebb is modellezni, mert akár politikai döntésektől is függhet. Célja meghatározni a szolgáltatás színvonalát, a kivitelezés módját. Itt kap helyet többek között az erőforrások beszerzése, a hálózattervezés (network planning), a járattervezés (line planning), és a menetrendtervezés (timetable planning). A menetrendtervezés operációkutatással azonban már egy széles körben alkalmazott terület, sok vasúti menetrend már nem kézzel készül, úgymint a német, svájci, vagy éppen a holland vasúti menetrend: [2].

A működtetés megszervezése a tényleges folyamat előtt nem sokkal/közben történik, itt már a meglévő erőforrásokat osztjuk be a megfelelő feladatokhoz, például a járműveket (vehicle sheduling), vagy a személyzetet (crew sheduling). Számos légitársaság, és egyéb közlekedési társaság automatizálja ezt a tervezési lépést, mert a rendelkezésre álló algoritmusok olcsóbban, és sokkal hatékonyabban osztják be a személyzetet, mint az ember.

1.2. Folyammodell

Talán ez a legkézenfekvőbb modell egy hálózat tervezéséhez.

Legyen (V, A) irányított gráf, V a csúcsok halmaza, A az (irányított) élek halmaza, $\{s,t\}$ forrás illetve nyelő.

Legyen k_a^{min}, k_a^{max}, $a \in A$ alsó illetve felső kapacitások, hogy $k_a^{max} \geq k_a^{min}$. Valamint legyen c_a, $a \in A$ egy adott folyamegységre jutó költség.

A feladat egy minimális költségű folyam megtalálása s és t között, az adott kapacitás megszorítások mellett. (Természetesen egyéb megszorításokat is lehet tenni.)

Ez formálisan felírva:

$$\min c^T y \tag{1}$$

$$y(\delta^{in}(v)) - y(\delta^{out}(v)) = 0, \qquad \forall v \in V \setminus \{s,t\}, \tag{2}$$

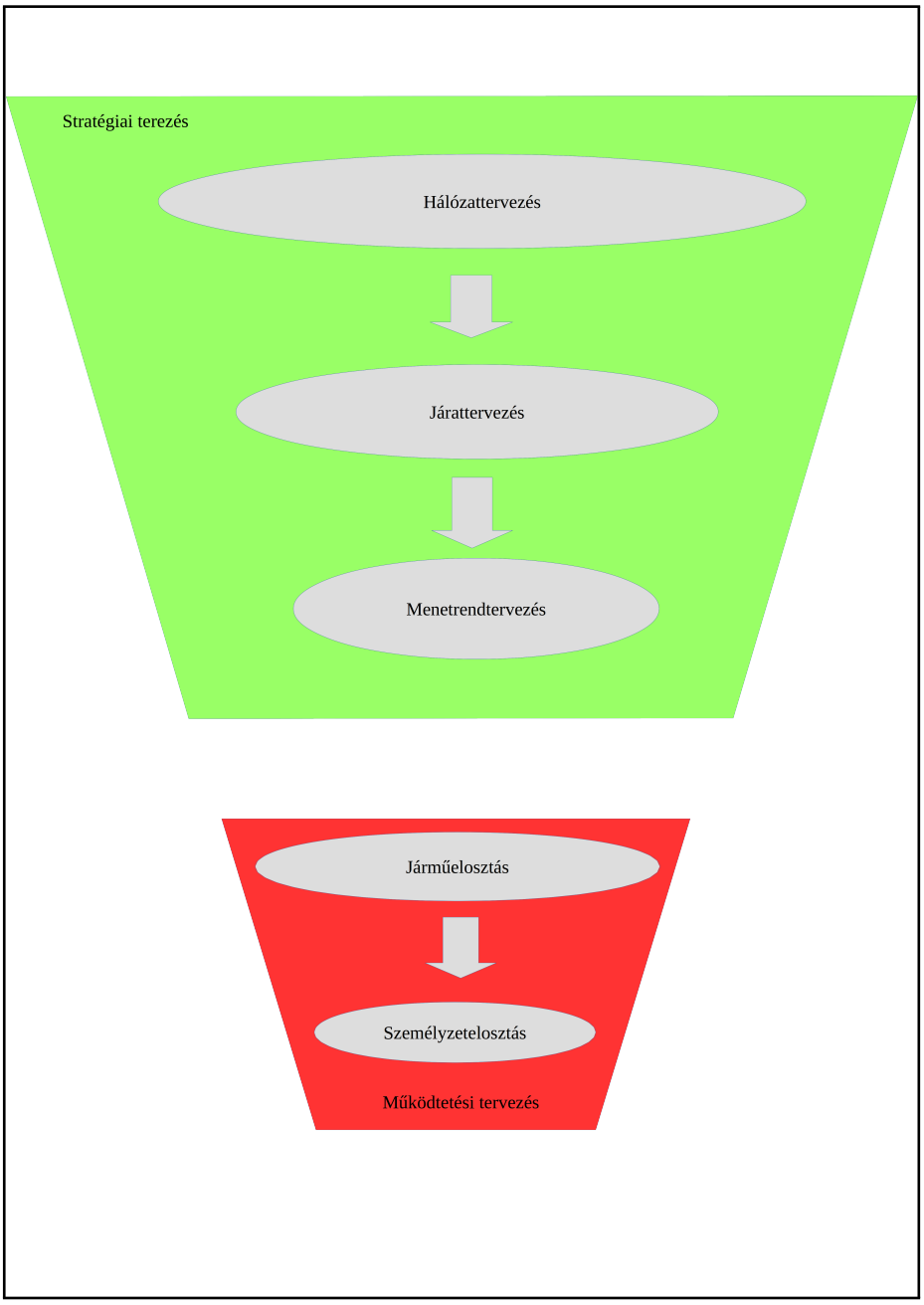
$$k_a^{max} \leq y \leq k_a^{min} \qquad \forall a \in A, \tag{3}$$

$$By = b \tag{4}$$

$$y_a \geq 0, \qquad \forall a \in A \tag{5}$$

$$y \in \mathbb{Z}^A \tag{6}$$

Itt y_a megmondja adott $a \in A$ élre a folyam értékét. (2) biztosítja a folyamfeltételt (amennyi folyam egy csúcsba befolyt, annyi kell kifolynia is; a forrás és a nyelő kivételével). (3) biztosítja az élekre, hogy a folyam egy adott élen



1. ábra. Tervezési lépések

ne lépje túl a korlátait. (4) feltételek megadásával a folyamnak speciális tulajdonságokat írhatunk elő. (5)-(6) pedig a nemnegativitásti, és az egészértékűségi feltételek, hiszen általában egész dolgokat kell szétosztani: pl.: személyzetet vagy járműveket.

Érdemes (4) nevezetes eseteire kitérni

- Ha ugyanis (4) üres, akkor egy minimális költségű folyamot kell megoldanunk, így ez polinomiális időben megy.
- Azonban (4) helyébe többtermékes folyamfeltételeket is tehetünk, így egy többtermékes folyamfeladatot kapunk.
- Jól alkalmazható $\sum_{a \in p} \leq |p| - 1$ feltétel egy $p \in P$ út tiltására.

A többtermékes folyam jól modellezi például több járműparkból való járműütemezést, vagy a telekommunikációs hálózatok működését is, hiszen minden kapcsolat egy külön termék folyamának tekinthető.

A harmadik speciális feltétel kényelmes lehet bizonyos utak kizárására, azonban ez a feltétel elronthatja a feladatunk szerkezetét, ami az így módosult feladat megoldásánál komoly hátrányokat okozhat.

Mint látszik a célfüggvényen is, ez egy **él-központú** modell. Ha bizonyos utakat gyakran kell kitiltani az algoritmus során, akkor jobban használható a következő modell, mely **út-központú**, mert a korlátozó feltételeket bizonyos utakra szabjuk meg.

1.3. Útpakolási modell

Ha az előző modellben (3)-ban többtermékes korlátozó feltétel van, sok termékkel a többtermékes folyam megoldása időigényes lehet. Ennek elkerülése végett a folyamfeladatot utak és körök uniójára bontjuk föl (lásd Datzig-Wolfe [3]).
Az általános útpakolási modell:
(General Path Based Model)

$$\min d^T x \tag{7}$$

$$k_a^{max} \leq \sum_{a \in p} x_p \leq k_a^{min}, \forall a \in A, \qquad \text{az alsó felső korlát } (p \in S) \tag{8}$$

$$x_p \geq 0, \forall p \in S, \qquad \text{a változók nemnegatívak} \tag{9}$$

$$x \in \mathbb{Z}^S, \qquad \text{és a változók egészek.} \tag{10}$$

Ahol S bizonyos a hálózatban haladó utak halmaza. Itt d_p a költségfüggvény, például: $\sum_{a \in p} c_a$.

GPBM-nek három nevezetes speciális esete van:

- Path Packing Problem: (8a) $1 \leq \sum_{a \in p} x_p$
- Path Partitioning Problem: (8b) $1 = \sum_{a \in p} x_p$
- Path Packing Problem: (8c) $1 \geq \sum_{a \in p} x_p$

S mérete így nagyon nagy lehet, ezért ezt a problémát érdemes valamilyen oszlopgenerálós algoritmussal megoldani.

1.4. Hálózattervezés

A hálózat és járattervezés még nem olyan jól kidolgozott, és ritkán alkalmazott tervezési lépések, és ha használják is, inkább csak a városi tömegközlekedésben. Ugyanis még a városi tömegközlekedésben is ritka az az alapszituáció, hogy alapjaitól kell, illetve lehet egy város közlekedését megszervezni, hiszen az infrastruktúra (buszmegállók, táblák, járatinformációk) már adottak, ezek lecserélése és az utasok tájékoztatása nagy marketingköltséggel járhat. Sokkal inkább jellemző, hogy egy meglévő hálózatot bővítenek, optimalizálnak az új igényeknek megfelelően.

A hálózattervezési feladatban adottak végállomások, és transzfer pontok, ahol a járatok indulnak, pályát módosíthatnak. A célunk adott forgalmú közlekedés megszervezése minimális költséggel.

Az alábbi modellt [4] szerzői írták fel először. Legyen $N = (V, A)$ egy hálózat, ahol V a végállomások, és transzfer pontok halmaza, A azon útszakaszok/utcák, ahol a járművek közlekedhetnek. Legyen egy p útra $O(p)$ illetve $D(p)$ az út két végpontja, b^p közlekedési forgalom $O(p), D(p)$ között. F legyen a járműfajták halmaza (mint például: csuklósbusz, alacsonypadlós busz, trollibusz, villamos stb). Ennek segítségével minden járműtípusra különböző kapacitásokat, illetve költségeket tudunk felírni. Tehát u_a^f mondja meg egy f típusú járműfajtából a élen hány ember fér fel. Legyen c_a^p és d_a^f $(f \in F; a \in A)$ az a költség, amely a közlekedési társaságnak a élen keletkezik p úton, illetve f járművön.

Jelölje y_a^f $(f \in F; a \in A)$ azon közlekedő egységek/emberek számát, akik a élen f típusú járművel utaznak, és x_a^p azon folyamérték, mely a élen folyik át.

(Network Design Problem)

$$\min \sum_{p \in P} b^p ({c^p}^T x^p) + \sum_{f \in F} {d^f}^T y^f$$

$$\sum_{p \in P} b^p x_a^p - \sum_{f \in F} u^f y_a^f \leq 0, \qquad \forall a \in A \quad (11)$$

$$x(\delta_p^{in}(u)) - x(\delta_p^{out}(u)) = 0, \qquad \forall p \in P, u \in V, \setminus (O(p) \cup D(p)) \quad (12)$$

$$x(\delta_p^{in}(u)) = x(\delta_p^{out}(u)) = 1, \qquad \forall p \in P \quad (13)$$

$$x_a^p \geq 0 \qquad \forall p \in P, a \in A \quad (14)$$

$$y_a^f \in \mathbb{N}_0, \qquad \forall a \in A, f \in F \quad (15)$$

Ahol a szumma első tagja minden egyes útra a közlekedőkre jutó költség, a második tag a járműveken eső költség. (11) biztosítja, hogy minden élen teljesüljön a kapacitáskorlát, (12) a folyamfeltétel, (13) biztosítja, hogy az előírt közlekedési forgalom teljesüljön. (14) és (15) a nemnegativitás, és egészértékűségi feltételek.

Azonban ez a modell nem mindig szolgáltat megfelelő eredményt. Ugyanis a kapott hálózatot a következő fázisban (járattervezés) nem biztosan fogjuk tudni gazdaságosan felbontani, mert egyes útvonalak akár túl hosszú/rövidre sikerülhetnek. Valamint még az is megoldandó feladat,hogy az utasok hány átszállással fognak eljutni a célállomásukra.

1.5. Járattervezés

Az előző feladatban kaptunk egy hálózatot lehetséges útvonalakkal, ezeket kell most járatokra felbontani, és meghatározni azok a gyakoriságát. Tehát adott egy hálózat végállomásokkal, transzfer pontokkal, és lehetséges összekötő útvonalakkal (lehetséges azonban, hogy egyes utakat csak bizonyos járműtípusok használhatnak), valamint az egyes utakra az előírt forgalommal.

A cél a költségek és az utazók élégedetlenségének minimalizálása (elégedettségi feltétel például: a belvárosban bárhonnan bárhova adott idő alatt avagy átszállással el lehessen jutni).

Mint már az előző részben is megemlítettem, nem jellemző egy hálózatot alapjaitól tervezni, sokkal inkább itt a járattervezésbe kalkulálják bele egy esetleges útvonal-módosítás költségét.

1.5.1. 1.modell

Szétválasztjuk a feladatot: első körben meghatározzuk az utasok útvonalát, és mértékét közlekedési szokásaik szerint. Majd meghatározzuk a járatokat, és azok gyakoriságát az ismert forgalom mellett, egy útalapú algoritmussal.

Tehát legyen L lehetséges járat-útvonalak halmaza. Legyen továbbá c_l, $l \in L$, az l járatra jutó költség, F_l az l járat gyakorisága ($0 \in F_l$). A legyen az él-járat incidenciamátrix: $A \in \{0,1\}^{AxL}$. És $b \in \mathbb{R}^A$ a forgalom az éleken.

(Line Planning Problem)

$$\min c^T x \tag{16}$$

$$Ax \geq b \tag{17}$$

$$x_l \in F_l \qquad \forall l \in L \tag{18}$$

És a feladat megoldása: x az előre megadott járatok gyakoriságát adja meg.

1.5.2. 2.modell

A második megoldás (lásd [1]) az előző két fázist próbálja meg együtt megoldani egy útalapú algoritmussal, ahol az utasokat és járatokat próbáljuk összeegyeztetni.

Jelölje $y_p \in \mathbb{R}_+$ a forgalmat, ahol $p \in P$ potenciális járatútvonal. Egy új x_l bináris változót vezetünk be minden $l \in L$ járatra, mely megmondja, fogjuk-e használni az adott járatot. Az f_l egész változó mondja meg az l járat gyakoriságát (a maximum kapacitást F_l jelöli). Jelölje b^p azt a változót, amely megadja p útra $O(p), D(p)$ közötti forgalmat, P^p pedig tartalmazza $O(p), D(p)$ közötti összes utat. $\mathbb{P}$ legyen ezek uniója, $\mathbb{P}(a)$ pedig azon $q \in \mathbb{P}$ melyek $a \in A$ élen átmennek. $L(a)$ azon járatok, amelyek átmennek a élen.$\kappa \in \mathbb{R}_+^L$ adja meg a járatkapacitást. Végül d_a az utasonkénti költség $a \in A$ élen , c_l pedig a járatköltség $l \in L$ járatra.

A hosszas jelölések után végre felírhatjuk a feladatot:

$$\min c^T x + d^T y + c^T f \tag{19}$$

$$y(P^p) = b^p, \qquad \forall \text{OD párra}, \tag{20}$$

$$\sum_{l \in L(a)} \kappa_l f_l - y(\mathbb{P}(a)) \geq 0, \qquad \forall a \in A \tag{21}$$

$$f \leq Fx \tag{22}$$

$$x \in \{0,1\}^L, \qquad 0 \leq f \leq F, y \geq 0. \tag{23}$$

$$\tag{24}$$

(20) biztosítja, hogy az adott forgalmat elvezettük. (21) miatt nem lépjük át a kapacitáskorlátokat. (22) feltétel kényszeríti ki x változótól, hogy 1 legyen, amikor az adott járatot betettük a megoldáshalmazba.

Megjegyzés: az első modellt inkább vasúti modellekre alkalmazható, hiszen itt az utasoknak nincs sok választása ha A városból B városba akar menni, így itt az utasok útjainak előre meghatározása nem nagy korlátozás.
A második modell mivel nem határozza meg ezt előre, a városi közlekedéstervezésben jobban alkalmazható.
Azonban egyik modell sem veszi figyelembe, hogy az utasoknak hányszor káll átszállniuk céljuk eléréséhez.
Sőt, mivel csak végállomásokat és transzfer pontokat vettünk be a modellbe, így a köztes megállókat nem is határoztuk meg, amik jelentősen befolyásolják egy járat menetidejét.

1.6. Menetrendtervezés

1.6.1. Periodikus Menetrendtervezés(Periodic Event Sheduling Problem)

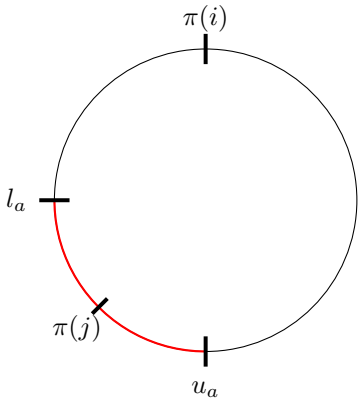
A periodikus menetrendek egy gyakran használt menetrend forma/tulajdonság, amikor egy menetrend során periódusonként ismétlődnek az események. Ez a periódus tipikusan egy óra. Ennek az utazók számára legkézzelfoghatóbb előnye, hogy csak egyetlen óra menetrendjét kell az adott állomásra megjegyezni: például a Budapest-Bécs railjet mindig 'óra negyvenkor' indul Budapestről (és minden köztes állomáson miden óra azonos percén halad át).

Az alábbi modellt Serafini és Ukovitch [5] vezette be 1989-ben. Tehát egy periódikus modellben adott egy T periódusidő (ami gyakran egy óra), egy V események halmaza, ami általában vonatok adott állomáson való érkezési/indulási idejeinek halmaza. Meg van adva természetesen egy A feltételhalmaz: ami egy eseménypárból áll, valamint minden élhez adott egy alsó-felső korlát (u_a, l_a: $\forall a \in A$).

A célunk egy menetrend; egy olyan függvény, ami megmondja egy eseményre, mikor következzen be. Formálisan keressük $\pi : V \to [0, T)$ függvényt, hogy

$$\pi(j) - \pi(i) - l_a (\text{mod} T) \leq u_a - l_a \qquad \forall a = (i,j) \in A; i,j \in V. \tag{25}$$

Másképpen röviden: $\pi(j) - \pi(i) \in [l_a, u_a]_{modT}$.



2. ábra. Illusztráció korlátokra

Def : Egy ilyen π függvényt menetrendnek nevezünk.

- Speciálisan: $l_a = u_a : \pi(j) - \pi(i) = l_a \bmod T$.
- Például ha $a = (i, j); l_a = 10, u_a = 15$ és $\pi(j) = 5$. Ekkor $\pi(i)$ 'óra 50 és 'óra 55 között következhet be.

1. Megjegyzés. *Alkalmazhatunk egy másik felírást is, ha* $0 \leq l_a < T$ *és* $\delta_a = u_a - l_a$ *változókat használjuk, ekkor*

$$\pi(j) - \pi(i) - l_a(modT) \leq \delta_a \qquad \forall a \in A. \tag{26}$$

Ezen a felíráson látszik, hogy két eseményre megmondhatjuk, hogy mennyi idő teljen el közöttük egy menetrendben (l_a)*, valamint hogy mennyire engedünk a pontosságból* (δ)*.*

2. Megjegyzés. *Ha egy adott eseményt rögzíteni szeretnénk, akkor ezt bátran megtehetjük, ugyanis* $(\forall i_0 \in V)(\forall t_0 \in [0, T))(\forall \pi$ *menetrendhez* $)(\exists \pi' : \pi(i_0) = t_0)$.
Biz: *Ugyanis el tudjuk forgatni a menetrendünket: legyen* π *egy menetrend,* i_0*,* t_0 *adott, ahol* $\pi(i_0) = t_0^1$*. Ekkor* $\pi' := \pi + (t_0 - t_0^1)$*. Ez ugyanúgy menetrend lesz, és* $\pi'(i_0) := \pi(i_0) + (t_0 - t_0^1) = t_0^1 + t_0 - t_0^1 = t_0$*.*

3. Megjegyzés. *Ha nem egy, hanem több eseményt szeretnénk rögzíteni, azt is megtehetjük egy egyszerű trükkel. Válasszunk ki egy tetszőleges* i_0 *rögzíteni kívánt eseményt,* t_0 *rögzíteni kívánt időponttal. Ekkor* $i_1 \dots i_n$ *legyen a maradék rögzíteni kívánt esemény,* $t_1 \dots t_n$ *rögzíteni kívánt időpontokkal. Ekkor* A*- hoz adjunk hozzá* n *élet:* $a_i := (i_0, i_i)$ $\forall i = 1 \dots n$*,* $l_a = u_a = t_i - t_0$ *alsó-felső korláttal. Ha van megoldás, akkor már csak az előző megjegyzés szerint* i_0 *eseményt be kell forgatni* t_0 *időpontba, és akkor minden rögzíteni kívánt esemény* t_i *időpontban fog megtörténni.*

Egy menetrend hasznosságát/pontosságát egy c költségfüggvény méri. Azaz egy menetrend annál jobb, minél kisebb

$$\sum_{a=(i,j)\in A} c_a(\pi(j) - \pi(i) - l_a). \tag{27}$$

Itt l_a helyébe u_a is írható lenne, sőt bármely köztes szám (mod T), attól függően, hogy az esemény pontosságát mihez viszonyítjuk

Az IP feladat felírásához egy fogalom:

1. Definíció (tenzió). *Adott egy (G, A) gráf. Egy $A \to \mathbb{R}$ függvényt* ***tenziónak*** *nevezünk, ha minden C körre az éleket irányítás szerinti előjellel összeadva 0-t kapunk.*

Mivel az eddigi képletekben π csak mint potenciálkülönbség jelent meg, így érdemes bevezetni:

$$\hat{x}_0 = \pi(j) - \pi(i) \tag{28}$$

függvényt.

Ez tenzió triviálisan, hiszen egy potenciálból származtattuk. Tehát az MIP (Mixed Integer Program: kevert egészértékű programozási feladat) probléma:

$$\begin{aligned} \min & c(\hat{x} + pT) \\ & \Gamma\hat{x} = 0 \\ l_a \leq & \hat{x} + pT \leq u_a \\ & p \in \mathbb{Z}^A \end{aligned}$$

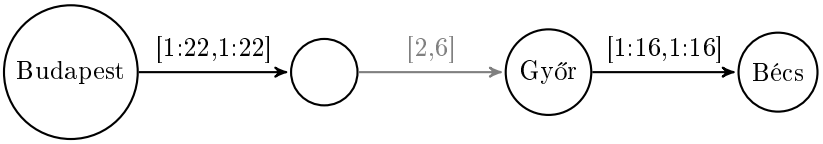
1. Lemma (Serafini és Ukovich). *Ha nem $\pi : V \to [0, T)$, hanem $\pi : V \to \mathbb{Q}$ menetrendet keresünk, akkor bármeny H feszítőfára bármeny π menetrendhez létezik olyan is, amire $p = 0$ minden élre.*

1.6.2. Pesp által fedett menetrendtervezési lépések

Mivel az előző szekciókban meghatároztuk az utasok forgalmát, a vonatok útvonalát, és gyakoriságát, ezért ezek adottak.

A három legelemibb dolog, amit modelleznünk kell, az az

- **utazás/mozgás:** az adott vonalon mennyi ideig megy a vonat
- **megállás:** azaz mikor áll meg a vonat, és mennyi ideig várakozik
- **átszállás:** mennyi ideig kell várakozni ha egyik vonatról egy másikra akarunk átszállni. Adott egy minimális átszállási idő, hiszen ha a vonatok nem ugyanazon a vágányon állnak meg, akkor az utasoknak át kell sétálniuk egyik peronról a másikra.



3. ábra. Példa járatok modellezésére: a halvány vonal a megállási él, a vastagabb nyilak a vonat közlekedését írják le.

A megállási éleknek általában pici a fesztávja: azaz a minimum és maximum várakozási idő közötti különbség, hiszen egy sok helyen megálló vonatnál ezen

fesztávok összeadódnak, tehát a modell felírása után az indulási idő ismeretében még csak körülbelül sem tudnánk, mikor érkezik meg a vonat. Ezért érdemes ezt a fesztávot nagy állomásoknál picire venni, kis állomásoknál nullára venni (azaz megmondjuk előre, mennyi ideig várakozik a vonat).

További előnye a 0 fesztávnak, hogy ilyenkor a modellünket tovább tudjuk egyszerűsíteni. Hiszen legyen $a = (i, j)$ 0 fesztávú él. Ekkor összehúzhatjuk az élt j csúccsal, éspedig úgy, hogy j minden élét $u_a = l_a$ értékkel eltoljuk (minden intervallumot eltolunk). Továbbá párhuzamos él elhagyható, ha intervalluma tartalmazza a másik intervallumot (azaz $a_1 = (i, j), a_2 = (i, j)$ és $[l_{a_1}, u_{a_1}] \subseteq [l_{a_2}, u_{a_2}]$ ekkor a_2 elhagyható).

Ha k vonat közlekedik egy adott vonalon, és egy egyenletes/kiegyensúlyozott menetrendet szeretnénk, könnyen megtehetjük ezt: vegyünk egy új élet az indulási csúcstól $[l_a, u_a] = [T/k, T - T/k]$ korlátokkal. Ez az él "kényszeríti" a vonatokat egyenlő időközönként indulni.

Az egyszerűség kedvéért tegyük fel, hogy minden vonat mozgási tulajdonságra ugyan olyan típusú: azaz ugyanolyan a gyorsulása, végsebessége stb.

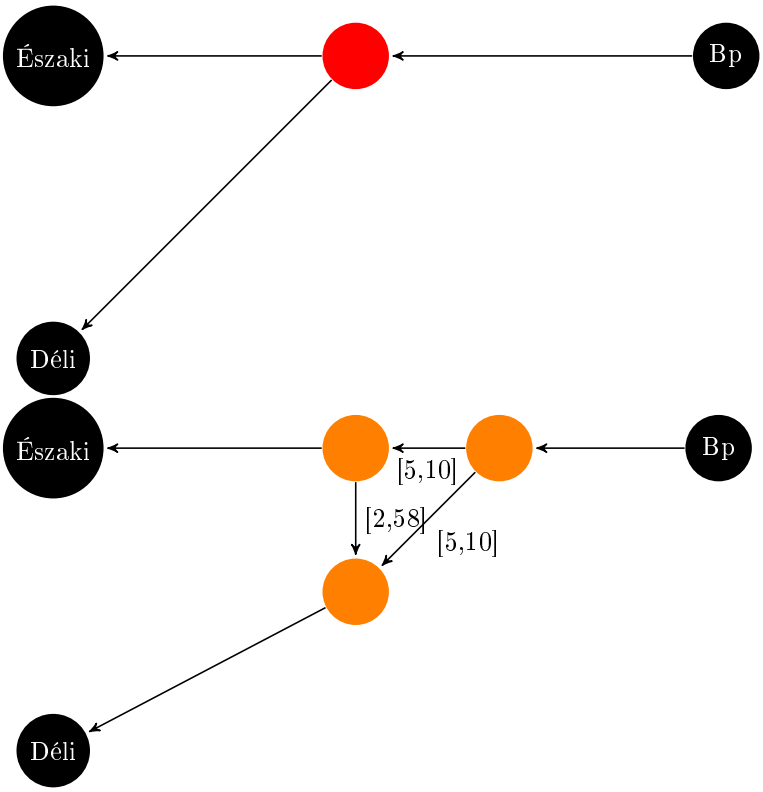
Követési távolság: A biztonságos közlekedés céljából bizonyos esetekben szükség van követési távolság beállítására. Ezt természetes módon egy plusz él felvételével tudjuk megtenni $[l, T - l]$ intervallummal, azon vágány elejéhez, vagy végéhez toldva, amit több vonat használ. Így biztosan nem lesz ütközés, hiszen a vonatok ugyan olyan sebességgel közlekednek, azaz nem tudják megelőzni egymást, és a követési távolságot is betartják a biztonsági él bevezetése miatt.

Szembemenő forgalom esetén a legegyszerűbb példa: Tegyük fel, hogy a hármas metró felújítása során az egyik alagutat teljes egészében lezárják a *Deák Ferenc tér*től *Újpest-Központ*ig. Ezért ezen a szakaszon csak az egyik alagútban közlekedik a metró (oda-vissza). Itt a menetidő 16 perc a megfordulási idő 4 perc, és egy 2 perces biztonsági távolságot szeretnénk minden szerelvény között. Ekkor egy új élet veszünk fel Újpest-Központ-nál az alábbi $[l_a, u_a] = [2, T - (16 + 16)] = [2, T - 32]$ korlátokkal.

Amennyiben egy vonat,f egységgel gyorsabb egy másik vonatnál, úgy $[d + f, T - d]$ korlátokat kell közéjük tenni, hogy betartsuk a biztonsági rést.

Biztosan tapasztaltuk már, hogy ha egy kétsávos útból lezárnak egy sávot, és egy sávon engedik a forgalmat, akkor nem a felére, hanem annál nagyobb mértékben visszaesik az út kapacitása. Sőt és nem is mindegy, a váltakozó forgalmat hogyan szabályozzuk. Ha ugyanis a forgalmat "csomókban" engedjük (azaz egy irányból egyszerre többet is), adott idő alatt nagyobb forgalmat tudunk átengedni, mintha egyesével engednénk az autókat/vonatokat. De nem csak egyirányú útszakaszokra, hanem többsebességes útszakaszokon (ahol ICE/IC és személyvonat is egy vágányon közlekedik) is célszerű "csomósítani" a vonatokat. Tehát nem mindegy, milyen sorrendben indítjuk a vonatainkat, mert ettől függően a menetrend optimalitása is változhat.

Szerencsére a $PESP$ modellbe ezt is könnyen belevehetjük. Vegyünk ugyanis fel egy plusz csúcsot, és $i_1, ..., i_n \in V$ a "csomósítani kívánt vonatok" azon idejét jelzi, amikor a közös vágányra lépnek. A plusz csúcsból vegyünk minden



4. ábra. Példa járatok szétkapcsolására

$i_1, ..., i_n$ csúcsba egy élet $[0, T/2 - t]$ korlátokkal, ahol t a vágányon az adott vonatok menetideje (csak azonos tulajdonságú vonatokat szoktunk "csomósítani"). És így nem is kell meghatároznunk a "csomósított" vonatok sorrendjét, amit esetleg már egy felsőbb tervezési szinten hoztunk.

Egy vasúthálózatban előfordulhatnak olyan megoldások is, hogy egy vonatszerelvényt egy állomáson **szétvág**nak két szerelvénnyé, és más útvonalon folytatják útjukat, vagy fordítva, két külön útról **összecsatlakoztat**ják őket Gondoljunk csak Budapest-Balaton útvonalakra: Északi-Déli part szerint szét is bonthatnák a szerelvényeket, és így az utasoknak nem kellene átszállniuk. Ez üzemeltetési költségek szempontjából is meggondolandó, de sokkal inkább túlterhelt útvonalakon lehet hasznát venni, hiszen két külön vonat között akár több perc biztonsági rés is lehet, míg ha összekapcsoljuk őket; 0.

Az ábra mutatja, egy ilyen összekapcsoló csúcsot csak szét kell szedni 3 csúcsba, két átszállási él időkorlátja természetesen adódik, míg a harmadik él behúzásával biztosítjuk, hogy például 2 perc biztonsági réssel induljanak el az állomásról. Sőt ezzel a megoldással még csak a szétkapcsolt vonatok indulási sorrendjét sem kellett meghatároznunk.

1.7. A járművek elosztása

Ez a negyedik tervezési lépés, amely már az operációs tervezési fázisba esik. Általában egy fix menetrenddel dolgozunk, azaz az egyes járatokra meg van határozva a kezdeti- és végállomás, valamint az induló, és érkező buszok időpontja is. A járműveink kezdetben, és a munka befejeztével (ugyan abba) a garázsba térnek vissza. A célunk az, hogy költséghatékonyan határozzuk meg az egyes buszok feladatait/útvonalait, hogy minden menetrend szerinti járaton egy (és csak egy) jármű haladjon végig, miközben ez a közlekedési társaságnak a legkevesebb költségbe kerüljön (például: minimális jármű üzembe helyezése, minimális üzemanyag fogyasztása stb.).

Tehát a feladataink, amelyeket el kell végezni, két helykoordináta (kiindulási és végcél) valamint két időkoordináta (kiindulási és végcél időpont) jellemzi. Ezeket **menetrend szerinti járat**oknak (timetable trip) nevezzük. Hogy ezeket el lehet-e végezni egymás után, az **üres járat**ok adják meg a kapcsolatot. Tipikusan ilyen járat a garázs, és az első/utolsó járat közötti út, ezt pull-in/pull-out trip -nek nevezi a szakirodalom. De természetesen az is *üres járat*nak minősül, ha a végállomáson megfordul a jármű.

Nem minden jármű közlekedhet minden útvonalon (a csuklós busz nem közlekedhet kis utcákon, hanem a nagyforgalmú járatokat kell szolgálnia). Sőt minden garázsra meg van határozva, milyen típusú járművek parkolhatnak itt, és típusonként vagy összesen adva van egy kapacitáskorlát.

Egy jármű végállomások közötti *üres járat*ok és *menetrend szerinti járat*ok egymásutáni fűzését blokkoknak hívjuk.

Egy közlekedési társaságnak az alábbi költségei keletkezhetnek: fix költség a járművenként (fenntartási, tárolási, lízing költség), valamint a garázson kívül töltött idő és távolság függvényében további költségek (emberi erőforrás, üzemanyag stb.).

Összefoglalva célunk tehát egy blokkokból álló olyan halmaz meghatározása, mely elvégzi az összes feladatot, miközben az említett költségeket minimalizálja.

Modell: Legyen $G = (V, A)$, ahol a csúcsok *menetrend-szerinti útvonal*aknak, az élek pedig az *üres útvonal*aknak felelnek meg, d_a költséggel (minden $d_a \in A$ élre). Természetesen a *menetrend-szerinti útvonal*aknak is meghatározhatnánk költséget, de mivel úgyis mindegyiket teljesíteni kell, ezért nincs értelme ebbe a modellbe bevenni. Legyen A_g azon élek halmaza, melyeken a g depóból induló járművek haladhatnak. Azon éleket, amelyek v csúcsból be- illetve kiindulnak, és g depóból induló járművek haladhatnak rajta, $\delta_g^{in/out}$-nak jelöljük.

Tehát formálisan a feladatunk:

(Járműelosztás)

$$\begin{aligned} \min\ & d^T y & \\ y(\delta_g^{in}(v)) - y(\delta_g^{out}(v)) &= 0 & \forall v \in V \setminus \{s,t\}, g \in G \\ y(\delta_g^{in}(v)) &= 1 & \forall v \in V \\ y(\delta_g^{out}(v)) &\leq k_g & \forall v \in V, \forall g \in G \\ y &\in \{0,1\}^A & \end{aligned}$$

Lényegében egy többtermékes folyamot írtunk le, itt az első három feltétel ezt fejezi ki. A negyedik feltétel pedig a depó tárolási kapacitása.

4. Megjegyzés. *Lehetséges a járművek elosztását összevonni/beleépíteni a menetrendtervezésbe, hogy még optimálisabb/költséghatékonyabb közlekedési rendszert kapjunk. Például megengedhetünk a menetrendben kisebb módosításokat (shiftelés).*

1.8. Személyzetelosztás

Az eddigiekben sikeresen megalkottunk egy közlekedési hálózatot, meghatároztuk a járatok útvonalát, menetrendjét, és végül pedig a rendelkezésre álló járműveket beosztottuk menetrend szerint. Már csak az emberi erőforrás elosztása maradt. Ez nagyon hasonló az előbb tárgyalt problémához, azonban mivel nem gépekről beszélünk, más korlátozásokat kell figyelembe vennünk. Mint például megfelelő pihenőidő közbeiktatása, az esetleges fix munkaidő betartása, vagy esetleg túlórák minimalizálása, és egyéb munkaügyi előírások betartása. Továbbá a járművezetőknek egyéb feladatokat is el kell látniuk, mint a jármű átnézése (sign in/off time), üzembe helyezése, különböző naplók vezetése, az utazásra való felkészülés (lásd például légitársaságok pilótái).

1.8.1. Rotáció

Az előbbi rész után azt gondolhatnánk, hogy most már nincs mit szervezni a személyzeten, azonban ez nem így van. Ugyanis beosztottuk a személyzetünket feladatokra, az igényeiknek, és a jogszabályoknak megfelelően. Azonban a feladatokat névtelen sofőröknek osztottuk be. Ez azért probléma, mert előfordulhat, hogy egy sofőr egyik nap sokáig dolgozik,akkor másnap nem kezdhet túl korán, hiszen pihenőidőre van szüksége. Ezért egyes közlekedési társaságok úgynevezett rotációs rendszerben osztják be járművezetőiket (lásd 1.8.1 ábra).

	Hétfő	Kedd	Szerda	Csütürtök	Péntek	Szombat	Vasárnap
1	éjszakás	éjszakás	nappali	nappali	délelőtti	délelőtti	
2		éjszakás	éjszakás	nappali	nappali	délelőtti	délelőtti
3			éjszakás	éjszakás	nappali	nappali	délelőtti
4	délelőtti			éjszakás	éjszakás	nappali	nappali
5	délelőtti	délelőtti			éjszakás	éjszakás	nappali
6	nappali	délelőtti	délelőtti			éjszakás	éjszakás
7	nappali	nappali	délelőtti	délelőtti			éjszakás
8	éjszakás	nappali	nappali	délelőtti	délelőtti		

1. táblázat. Példa rotációra (A műszakokat rotációs rendszerben cserélik)

2. Nagy IP feladatok megoldása

Az előző részben sok egészértékű programozási feladatot írtunk fel, ebben a részben ezek megoldásával foglalkozunk. A következőkben, ha nem mondjuk a következő általános egészértékű programozási feladattal foglalkozunk:

$$\begin{aligned} \min\ & cx \\ & Ax \leq b \\ l \leq\ & x \leq u \\ & x_j \in \mathbb{Z}; \forall j \in S \subseteq \mathbb{N}; \end{aligned} \tag{29}$$

2. Definíció (LP fízibilis megoldás). *Adott egy egészértékű feladat. Ennek csak az optimalitást nem teljesítő x megoldását fízibilis megoldásnak nevezzük.*

3. Definíció (IP fízibilis megoldás). *Adva van egy egészértékű programozási feladat, amely esetleg $x \in \mathbb{Z}^S \times \mathbb{R}^{n-S}$, azaz csak S indexhalmaz egészértékűségét követeli meg. Ennek egy IP fízibilis (és nem feltétlen LP fízibilis) megoldása egy olyan x vektor, melynek S koordinátái egészek.*

Az LP és IP fízibilis megoldást egyszerűen fízibilis megoldásnak nevezzük.

4. Definíció (lock). *Egy (29)-cel megadott IP feladat x megoldás j-edik koordinátához tartozó 'lock' száma azon feltételek száma, mely x_j kerekítése során megsérülhetnek.*

2.1. Branch and Bound

Talán a legáltalánosabb ismert, egészértékű programozási feladatokra használt algoritmus, vagy inkább megoldási séma.

A *branch and bound* lényege, hogy a diszkrét megoldási halmazt alproblémákra osztja/szétdarabolja ("branch"), majd elvet néhányat ("bound"). Ezt rekurzív ismételve addig folyatjuk, amíg meg nem találjuk az optimális megoldást.

Egy lehetséges séma pszeudókódja:

Data: Adatszerkezet, Az IP feladat LP relaxáltja
Result: Az IP megoldása
Adatszerkezet.push({LP}) **while** *Adatszerkezet* $\neq \emptyset$ **do**
  Problem=Adatszerkezet.pop() ;
  if *Problem.egész() AND Problem.vizsgáld()* **then**
    optimális megoldás frissítése;
  else
    x_i=Problem.törtVáltozó();
    $Problem_i$=$\{Problem\} \cup \{\ branch_1\}$ i=1 ... k;
    Adatszerkezet.push($Problem_i$) i=1 ... k;
  end
end

Algorithm 1: Branch and Bound

Itt persze sok múlik az adatszerkezetünkön, hogy melyik problémát fogja kiadni, amikor kérünk egyet tőle (.pop()). Példa ilyen stratégiára (verem/sor): mindig az utolsó/első betett problémát adja vissza (ilyen típusú kereséssel mélységi/szélességi bejárást kapunk).
A *Problem.vizsgáld()* függvény logikai értékű függvény, ezzel olyan ágakat vethetünk el az algoritmus során, amelyről már tudjuk, hogy nem fog optimális megoldást adni.

5. Definíció (branch and bound fa). *Egy olyan gyökeres fa, melynek csúcsai olyan Lp problémák, mely az eredeti Lp problémánkból kaphatunk néhány korlátozó feltétel hozzáadásával, és egy probléma leszármazottja egy másik problémának, ha származtatható néhány feltétel hozzáadásával.*

Ha van jó becslésünk az adott problémára, bizonyos alproblémákat "levághatunk". Hiszen például egy ágat levághatjuk, ha Lp relaxáltja nagyobb az eddigi legjobb megoldás költségénél (minimalizációs problémáknál). Ugyanis ezen az ágon hiába keresnénk egész megoldásokat, az LP relaxáltnál csak nagyobb-egyenlő költségűt találhatunk, ami nagyob-egyenlő az eddigi optimális megoldásnál, így biztosan nem lesz optimális. Esetleg ha tudjuk, hogy a gyökér csúcsban az Lp relaxált jól közelíti az Ip megoldást, az olyan alproblémákat is levághatjuk, amelyek előre megadott (mondjuk 5) százalékkal rosszabb Lp relaxálttal rendelkeznek a gyökér LP optimumnál.

Ha tehát van egy közel optimális Ip megoldásunk, általában sok alproblémát "levághatunk", így gyorsítjuk algoritmusunkat. Egy ilyen megoldás megtalálására érdekében többféle stratégiánk lehet. Például a *branch and bound fá*ban szélességi bejárást végzünk. Azonban ez nagyon sok memóriát igényelhet. Szigorúbb memóriakorlátok mellett a mélységi bejárás alkalmazása gyakoribb, valamilyen alámerülési heurisztikával. Vagy ezeket kombinálhatjuk is: amíg van elég memóriánk szélességi bejárást végzünk, mikor már nem sok maradt, a kigenerált csúcsokból mélységi bejárást indítunk egymás után (a sorrendet okosan/heurisztikusan megválasztva).

2.1.1. Branch and Cut

Ezt olyan feladatok megoldásánál használhatjuk, ahol nagyon sok(például exponenciálisan sok) korlátozó feltétel van. Ennek az algoritmusnak az alapja a *branch and bound*, azonban a sok korlátozó feltétel miatt az Lp megoldása hosszadalmas lehet. Ezért kezdetben sorok csak egy alkalmasan választott részhalmazával foglalkozunk. Ha a *branch and bound fá*ban a részprobléma egy megoldáshoz jutunk, akkor megnézzük, teljesíti-e az részprobléma megoldása az összes feltétel. Ha nem, akkor a részproblémához hozzáadunk néhány sértő sort,és folytatjuk az eljárást.

2.1.2. Branch and Price

Tipikusan olyan feladatokra alkalmazható, melyekre nagyon sok változónk van. Ez a *branch and bound* egy oszlopgenerálós változata. A *Branch and Cut* algoritmushoz hasonlóan működik, csak a kiindulási részprobléma itt oszlopok egy

alkalmasan megválasztott részhalmazából áll. Az algoritmus során a rész LP-hez az olyan oszlopokat vesszük hozzá, amelyeknek a redukált költsége negatív.

A *Branch and Price* és *Branch and Cut* nyilván akkor működnek jól, ha az alkalmas oszlopok/sorok generálását gyorsan (polinomiális időben) meg tudjuk tenni, valamint a kezdőproblémát jól meg tudjuk választani, amely nem túl sok oszlopot/sort tartalmaz, és viszonylag nem is kell túl sok oszlop/sorgenerálást végezni.

2.1.3. Diving heurisztikák

Az előzőekben leírtam, miért is fontos egy fízibilis megoldást találni, minél gyorsabban, hogy a *Branch and Bound* minél hatékonyabb legyen. Tehát ebben a részben olyan heurisztikákat mutatok be, amely egy ilyen fízibilis megoldást talál, lehetőleg minél optimálisabbat.

A heurisztikák mélységi bejárással egy fízibilis, és minél optimálisabb megoldás felé törekednek, úgy, hogy a *branch and bound* fában ígéretes változókat kerekítenek a megfelelő irányba.

De előtte vezessünk be néhány fogalmat az egyszerűbb tárgyalás kedvéért:

6. Definíció (legközelebbi egész). *Legyen x_j törtváltozó. Az $\lfloor x_j + 0.5 \rfloor$ egész számot az x_j-hez tartozó legközelebbi egésznek hívjuk.*

7. Definíció (töredékrész). *Egy x_j tört koordináta töredékrészén (nem összetévesztendő a törtrésszel) $|x_j - \lfloor x_j + 0.5 \rfloor|$ számot értjük. Azaz a töredékrész a legközelebbi egésztől való távolság. A továbbiakban $f(x_j)$-ként jelöljük.*

8. Definíció (triviálisan le/felkerekítés). *Egy (29)-el adott IP egy x_j változója triviálisan lefelé kerekíthető, ha $A_{.j} \geq 0$.*
Hasonlóan egy x_j változó triviálisan felfelé kerekíthető, ha $A_{.j} \leq 0$.

- **Töredékrészes módszer**: Addig branchel a minimális töredékrész változó mentén a legközelebbi változó felé, amíg van egész változó, vagy amíg infízibilissé nem válik a megoldás.
- **Együttható módszer**: Azon koordinátát kerekítjük a megfelelő irányba, melynek a 'lock' száma a legkisebb. Ha több ilyen is van, akkor a legkisebb töredékrészűt választjuk.
- **Lineáris keresés**: Legyen $\hat{x}$ az *branch and bound* fa gyökerében levő LP relaxált megoldása, és $\tilde{x}$ az aktuális csúcsban levő megoldás $x_j = k$ szerint kerekítünk, melyre a s= $\hat{x}\tilde{x}$ szakasz először metszi az $x_j = k$ hipersíkot.
- **Vectorlength módszer**: Ez a módszer Set Packing problémákra alkalmazható, ami a valóságban gyakran előforduló problématípus. A Set packing probléma:

$$
\begin{aligned}
&(SPP)\\
&\min cx\\
&Ax = 1\\
&x \in \{0,1\}^n\\
&\text{ahol } A \in \{0,1\}^{m \times n}
\end{aligned}
$$

Ennél a feladatnál ha egy tört változót 1-re fixálunk, akkor az új Lp megoldásban legalább egy másik változó egészre (0-ra) állítódik, hiszen A $\{0,1\}$ mátrix, és a sorösszeg 1.
Azt a j. koordinátát állítjuk 1-re, amire

$$\min \frac{([\bar{x}_j]_* - \bar{x}_j) * c_j}{|\{a_{ij} \neq 0\}|} \tag{30}$$

felveszi minimumát.Ahol

$$[\bar{x}_j]_* = \begin{cases} \lfloor \bar{x}_j \rfloor & \text{if } c_j \geq 0 \\ \lceil \bar{x}_j \rceil & \text{if } c_j \leq 0 \end{cases} \tag{31}$$

Vagyis afelé próbálunk kerekíteni, amerre a célfüggvény kevésbé romlik, és a nullára állítható változók száma egyre nagyobb (Hiszen $x_j = 1$ esetén $\forall i : a_{ij} = 1 \implies x_j = 0, \text{mert } \sum_{i=1}^{m} a_{ij} = 1$).

Továbbá bármely eddigi heurisztikára igaz, hogy érdemes (ha lehet) bináris változók mentén kerekíteni, hiszen ez fixálást jelent, így elkerülhetjük egy nem bináris változó többszöri korlátozását. Valamint bináris változók a modellekben, és az eredmény szempontjából is általában fontos szerepet játszanak. Például a járattervezés 2. modelljében (1.5.2) először x bináris változóval eldöntöttük, az adott járatot bevesszük-e a megoldásba, és egy másik változóval döntöttük csak el, milyen gyakorisággal közlekedik. Így ha először a bináris változók mentén *branch*elünk, akkor megmondjuk, melyik járatot választjuk is be a megoldásba, így a megmaradt jóval kevesebb járat gyakoriságait kell csak úgy megválasztani, hogy az adott forgalmat elvezessék.

5. Megjegyzés. *Az előbbi heurisztikák akár bejárási technikák is lehetnek, hiszen csak azt mondták meg, egy adott csúcsában a branch and bound fának melyik ágán haladjunk egy fízibilis, vagy közel optimális megoldás megtalálása érdekében.*

2.1.4. Simple Feasibility Pump

A *Feasibility Pump* [6] fejlesztették ki először bináris problémákra, majd [7] fejlesztették tovább általános egészértékű problémákra. Ennek a primál heurisztikának a lényege, hogy két sorozatot definiál;

- egy Lp fízibilis pontokból állót, amely általában nem IP fízibilis
- és egy IP fízibilis pontokból állót, amely nem feltétlen LP fízibilis.

A heurisztika szerint ezek a sorozatok konvergálnak, és a határérték ezek szerint már IP, és LP fízibilis is lesz.

9. Definíció (S halmazra vett egészrész). *Legyen* $S \subseteq \mathbb{N}$.

$$[x]^S := \begin{cases} [x_j + 0.5] & ha\ j \in S \\ x_j & ha\ j \notin S \end{cases} \tag{32}$$

10. Definíció (S halmazra vett L^1 távolság). *Legyen* $S \subseteq \mathbb{N}$.

$$\delta^S(x,y) := \sum_{j \in S} |x_j - y_j| \tag{33}$$

Valamint hasonlóan $f^S(x) := \sum_{j \in S} f(x_j)$ ahol már $f(x_j)$ töredékrészt már definiáltuk.

Az eljárás menete a következő: megoldjuk a feladat LP relaxáltját, jelöljük ezt $\bar{x}$-al. Ehhez megtaláljuk a legközelebbi egész megoldást: $\tilde{x} = [x]^S$ ahol $S = \{$ bináris változók$\}$ vagy $S = \{$ egész változók$\}$. Ha $\tilde{x}$ fízibilis, megállunk. Különben egy újabb LP feladat megoldásával keressük az $\tilde{x}$-hoz $\delta^S(x,\tilde{x})$ távolság szerinti legközelebbi $\bar{x}_1$ pontot. Majd elölről addig ismételjük ezt az eljárást, amíg fízibilis megoldást nem kapunk, vagy egy bizonyos időkorlátot el nem érünk.

Az algoritmus pszeudókódja:

Data:
$\bar{x}$: LP fízibilis megoldás
maxIter: maximális iterációszám
Result: Az IP egy fízibilis megoldása, vagy hogy sikertelen volt az algoritmus
while $i<maxIter$ **do**
  $\tilde{x} := [\bar{x}]^S$;
  if $\tilde{x}$ *LP fízibilis* **then**
    break;
  end
  $\bar{x} = \{x|\delta^S(x,\tilde{x})$minimális és x LP fízibilis$\}$;
end

Algorithm 2: Feasibility Pump

Ahhoz hogy meghatározzuk

$$\bar{x} = \{x|\delta^S(x,\tilde{x}) \text{ minimális, és } x \textit{ LP fízibilis}\} \tag{34}$$

pontot, meg kell oldanunk a következő LP feladatot:

$$\min \sum_{j \in S: \tilde{x}_j = l_j} (x_j - l_j) + \sum_{j \in S: \tilde{x}_j = u_j} (u_j - x_j) + \sum_{j \in S: l_j < \tilde{x}_j < u_j} d_j \tag{35}$$

$$Ax \leq b \tag{36}$$

$$d \geq x - \tilde{x} \tag{37}$$

$$d \geq \tilde{x} - x \tag{38}$$

$$l \leq x \leq u. \tag{39}$$

Ez látszik hogy jó, hiszen a egy optimális megoldás esetén d_j (37),(38) közül az egyiket egyenlőséggel teljesíti, így x valóban $\tilde{x}$-hoz legközelebbi LP fízibilis megoldás lesz. Természetesen ha az egész változóink binárisak, akkor d-re nincs szükségünk, ugyanis a célfüggvényben a harmadik szumma $j \in S : l_j < \tilde{x}_j < u_j$-re megy, ami üres halmaz, hiszen $l_j, \tilde{x}_j, u_j \in \{0,1\}$, így mindkét egyenlőtlenség

nem teljesülhet. [6] szerzői azt javasolják, osszuk három fázisra az eljárást.

Az első fázisban csak a nem bináris egész változókat relaxáljuk így $S = B :=$ { bináris változók }. Az eljárás

- vagy egy fízibilis megoldás megtalálása után áll meg,
- vagy egy bizonyos abszolút számú iterációszám elérése után,
- vagy egy bizonyos számú olyan iteráció után, amikor a törekékrészek összege ($f^s(x)$) nem csökken mondjuk p % -nál nagyobb mértékben. Ezzel elkerülhetjük olyan megoldások keresését, amelyek csak kevéssé mozdítanak minket a kitűzött cél irányába.

A második fázisban az egész változókat akarjuk kerekíteni, azaz $S = I :=$ { egész változók }. Azért választottuk szét az első két fázist, mert abban reménykedünk, hogy az első fázisban sok nem bináris egész változót sikerült egészre állítani, így csak kevés új d_j változót kell bevezetnünk. Addig futtatjuk az eljárásunkat, amíg egy IP, és LP fízibilis megoldást nem találunk, vagy egy bizonyos iterációs határt el nem értünk a fentiekhez hasonlóan.

Az utolsó fázisban azt reméljük, hogy a második fázisban kapott $\tilde{x}$ már majdnem fízibilis, így a környezetében találunk egy jó fízibilis megoldást. A környezetben levő megoldások átvizsgálásához, az alábbi MIP problémát oldjuk meg:

$$\begin{aligned} \min\ & \delta^I(x, \tilde{x}) \\ & Ax \leq b \\ & d \geq x - \tilde{x} \\ & d \geq \tilde{x} - x \\ & l \leq x \leq u \\ & x_j \in \mathbb{Z}; \qquad\qquad \forall j \in I. \end{aligned}$$

2.1.5. Objective Feasibility Pump

A *Simple Feasibility Pump* a gyakorlatban elég hatékonynak bizonyult egy fízibilis megoldás megtalálásában, azonban gyenge primál megoldást nyújt. Jóllehet mert a fenti algoritmus csak kiindulása során veszi figyelembe a célfüggvényt: csak a kezdeti $\bar{x}$ LP optimum megtalálásakor. Ezen úgy lehet segíteni, hogy a távolságfüggvénybe beépítjük c-t, ezáltal remélve hogy a heurisztika jobban az optimalitás felé mozdul.

Tehát $\delta^S(., \tilde{x})$-t lecseréljük; $\delta^S(., \tilde{x})$ és c konvex kombinációjára:

$$\delta^S_\alpha(x, \tilde{x}) := (1 - \alpha)\delta^S(x, \tilde{x}) + \alpha \frac{\sqrt{|S|}}{\|c\|} c^T x \tag{40}$$

Tehát az algoritmus ugyan az, csak más távolságfüggvényt alkalmazunk. Sajnos az optimalitás növelése a fízibilitás terhére tehető meg. Így az algoritmus előrehaladtával érdemes c hatását csökkenteni, ezt pedig α csökkentésével tehetjük meg, ennek egyik módja, ha minden i iterációban $\alpha_i := \beta^i$ valamilyen

$\beta \in [0,1)$ valós számra. Természetesen $c \equiv 0$ vagy $\beta \equiv 0$ a *Simple Feasibility Pump* algoritmust adja vissza ($c \equiv 0$-t azért is érdemes külön venni, mert $\delta^S_\alpha(x,\tilde{x})$ definíciójában 0-val osztanánk ...) .

Bár a leírt algoritmus csak egy heurisztika, ezért nem is várunk tőle **mindig** megoldást, még inkább nem optimális megoldást. De van egy olyan eshetőség, amit érdemes orvosolni, éspedig a ciklizálás. Azaz az algoritmus visszatérése ugyan abba a pontba. Különösen fontos ez a *Simple Feasibility Pump* esetében, ugyanis ha ugyanabba a $\tilde{x}$ egész változóba értünk vissza, akkor ugyanabba a $\bar{x}$ törtváltozó lesz ehhez a legközelebb, mint az első alkalommal. Így az algoritmus ciklizál, és nem lesz esélye egy fízibilis megoldás megtalálására. A hibát kétféleképpen próbáljuk orvosolni:

- $\bar{x}$ néhány változóját random fel-lefelé toljuk el
- ha a kerekítés ismét $\tilde{x}$ lesz, akkor pedig néhány nagy töredékrészű változót a távolabbi egész felé kerekítünk.

Majd ezután folytatjuk az algoritmust.

Az *Objective Feasibility Pump* is visszatérhet ugyan abba az $\tilde{x}$ csúcsba (az első és a második látogatás it_1, it_2 iterációban történjen). Azonban a távolságfüggvény nem ugyan az a két lépés során ($\delta^S_{\alpha_{it_1}}(x,\tilde{x})$, és $\delta^S_{\alpha_{it_2}}(x,\tilde{x})$), így a legközelebbi fízibilis csúcs $\bar{x}$ sem lesz feltétlen ugyanaz. Így tehát csak $\delta^S_{\alpha_{it_1}}(x,\tilde{x})$, és $\delta^S_{\alpha_{it_2}}(x,\tilde{x})$ nagymértékű egyezése (azaz $\alpha_{it_1} - \alpha_{it_2} \leq \delta_0$; $\delta_0 \in (0,1)$) során kell a fenti javítási módszert alkalmazni.

2.2. Rapid Branching

A *Rapid Branching* egy hatékony *branch and bound* heurisztika, amit főleg nagyméretű tömegközlekedési problémák megoldására használnak. Eredetileg Weider [8] használta ezt az algoritmust összehangolt személyzet és járműelosztásra. De sikeresen alkalmazta Borndörfer [9] *track allocation* és egyéb [10] problémák megoldására is.

A továbbiakban az alapproblémánk egy hálózatban éleknek valamilyen részhalmazát kell megtalálnunk (általában utak és körök unióját), valamilyen c költségfüggvény mellett. A probléma méretét a hálózat csúcsszámában mérjük.

A *Rapid Branching* egyik ötlete, hogy a célfüggvény ügyes módosításával a feladatott az egészértékű megoldások felé tereli (*Perturbation Branching*). Másrészt megpróbál minél több koordinátát 1-re rögzíteni. Ezt persze nem mindig jó ötlet, ezt felügyeljük egy bináris féle kereséssel: *binary search barnching*. És természetesen LP approximációs technikákat is felhasználhatunk, hogy gyorsítsuk az algoritmust.

A

$$\min\{c^T x | Ax = b, x \in \{0,1\}^n\} \tag{41}$$

típusú problémákkal foglalkozunk ($A \in \mathbb{R}^{m \times m}$, $b \in \mathbb{R}^m$), ahol n nagyságrendekkel nagyobb, mint m (például n a hálózat csúcsainak lineáris, m exponenciális függvénye). Emiatt, hogy kezelni tudjuk a problémát, oszlopgenerálást is alkalmaznunk kell az algoritmus során (a generált sorok indexeinek halmaza: $N \subseteq \mathbb{N}$). Mivel nagy problémákkal foglalkozunk, nem nagy megszorítás, ha feltesszük, hogy az LP approximáció jól közelíti az IP optimumot.

- (IP)(l,u) $\min_{l \le x \le u} \{c^T x : Ax = b, x \in \{0,1\}^n\}$
- (RIP)(l,u) legyen (IP)(l,u) megszorítása N koordinátákra.
- (MLP)(c,l,u) az (IP)(l,u) probléma LP relaxáltja.
- (RMLP)(c,l,u) (MLP)(c,l,u) megszorítása N koordinátákra.

Mint már ezelőtt megemlítettem, a célfüggvényt perturbálni fogjuk az algoritmus során (azt remélve, a perturbált célfüggvénnyel $f^I(x)$ csökken), ezért kell MLP, RMLP-hez paraméterként hozzáadni egy célfüggvényt.

Célunk egy minél jobb IP megoldás megtalálása, ezt egy IP$(0,1)$ gyökerű fából fogjuk indítani, melynek csúcsai IP(l,u). A fa bejárása mélységi jellegű, a megfelelő irányt a *perturbation branch* fogja megadni az adott csúcsban, azaz néhány változót fixálunk l és u segítségével. Ahogyan a *branch and bound* algoritmus pszeudókódjában is jeleztük, minden csúcsot analizálunk, érdemes-e ebbe az irányba tovább keresnünk egy jó IP megoldás érdekében.

A *perturbation branching* tipikusan sok változót ad vissza (k darabot), amiket fixálásra javasol. Ha később kiderül, valamiért mégsem érdemes az összes (k) javasolt változót fixálni, akkor megpróbáljuk a $k/2$ legígéretesebb változót fixálni. Ha ez is soknak bizonyul, $k/4$ változót fixálunk, és így tovább... Azaz

"bináris módon" próbáljuk megkeresni melyik az az l egész szám, amennyi változót érdemes lerögzíteni.

A *Rapid Branching* pszeudókódja:

Data: IP feladat, és egy δ tolerancia küszöb
Result: Sikeres futás esetén x^* megoldás, és az optimalitást bizonyító lb alsó korlát, amire fenn kell hogy álljon $lb + \delta \geq c^T x$

S_0 :=IP feladat;
$ub^* := \infty$; $lb^* := -\infty$;
$i := 0$;
while $S \neq \emptyset$ **do**
  set $N_i \in S_i$ /*binary search alapján*/ ;
  compute $lb_i \leq \min_{x \in N_i} c^T x_i$;
  if $x_i \in N_i$ *megoldást találtunk* **then**
    $ub_i := c^T x_i$;
    $ub^* = \min_{1 \leq k \leq i} ub_k$;
    $x^* := \text{argmin}_{1 \leq k \leq i} c^T x_k$;
  else
    $ub_i := \infty$;
  end
  if $lb_i \geq ub^*$ **then**
    Ezt az ágat nem érdemes vizsgálni: $S_{i+1} := S_i \setminus N_i$;
    $i := i + 1$;
    continue;
  end
  $lb^* := \min\{lb_k : N_k \in S_i\}$;
  if $lb^* + \delta \geq ub^*$ **then**
    Egy elég jó megoldást találtunk, így megállhatunk.
    break;
  end
  A perturbáció branching:
  compute $\cup_{j=1}^{k^i} Q_i^j$, $1 \leq j \leq k_i$;
  $S_{i+1} = (S_i \setminus \{N_i\}) \cup_{j=1}^{k^i} Q_i^j$;
  $i := i + 1$;
end

Algorithm 3: Rapid Branching, [10] nyomán

Mivel mi csak egy közel optimális megoldást keresünk, N_i szétbontásából nem fogjuk az összeset végigvizsgálni, arra fogjuk használni a *perturbation branching* eljárást, hogy pár igéreteset kiválasszunk ezek közül.

2.2.1. Perturbation Branching

A *perturbation branching* egy sor MLP-t old meg, miközben a célfüggvényt úgy változtatja (c_i, i=0,1,2 ...), hogy a megoldás egészértékűség felé mozduljon (például mérjük ezt a koordináták töredékrész összegével: $f^I(x)$, ami ezek szerint csökken). Az algoritmus során azoknak a koordinátáknak a költségét csökkent-

jük, amelyek közel vannak egyhez. A célfüggvény perturbációja:

$$c_j^0 := c_j; j \in N$$
$$c_j^i + 1 := c_j^i + c_j \alpha x_j^2; j \in N \qquad\qquad j = 0, 1, 2, \ldots$$

Azaz így elérjük, hogy az 1-hez közel levő változók 1 felé mozduljanak el. A folyamat sikerességét a

$$v(x_i) := c^T x - \delta |B(x_i)| \tag{42}$$

méri, ahol $B(x^i) := \{\text{majdnem 1 változók}\} = \{j \in N : x_j^i > 1 - \epsilon\}$. Ahol δ, ϵ segítségével az egészértékűséget hangolhatjuk (például $\delta = 1$, $\epsilon = 0.1$).

Ezt addig folytatjuk, amíg ez a tesztfüggvény csökken, ha már egy bizonyos iteráció után sem csökken, a legnagyobb olyan indexet fixáljuk 1-re, és a célfüggvényét 0-ra állítjuk. Ha ez sem segít leállítjuk az algoritmust.

2.2.2. Binary Search Branching

A *preturbation branching* egy B^* koordináták egy halmazát jelöli ki, amit arra javasol, hogy minden $i \in B^*$-ra x_j alsó korlátját 1-re fixáljuk. De mint ezt már írtam, nem mindig célszerű az összes változót fixálni, mert így gyorsan infízibilis megoldásokhoz juthatunk, és messze kerülünk az optimális megoldástól.

Ezért is építünk be egy olyan mechanizmust, ami az ilyen eltévedések után a 'megfelelő' csúcshoz vezet minket vissza. Tehát az algoritmus során egy RMLP(c, l, u) csúcsban adva van egy B^* indexhalmaz, $K := |B^*|$. Feltehetjük hogy B^* a redukált költség szerint növekvően van rendezve. Azaz $i, j \in B^*$ esetén $i < j \implies (c - A^T y)_i \leq (c - A^T y)_j$ ha RMLP(c, l, u) (29) alakban van megadva (y a duális vektor). Ekkor

$$B_k^* := i_1 ... i_k \qquad\qquad k = 1...K; \tag{43}$$

Vizsgáljuk az alábbi alproblémákat:

$$P_k := \min_{l \leq x \leq u} \{c^T x : Ax \leq b, x \in [0,1]^n | x_j = 1, j \in B_k^*\} \tag{44}$$

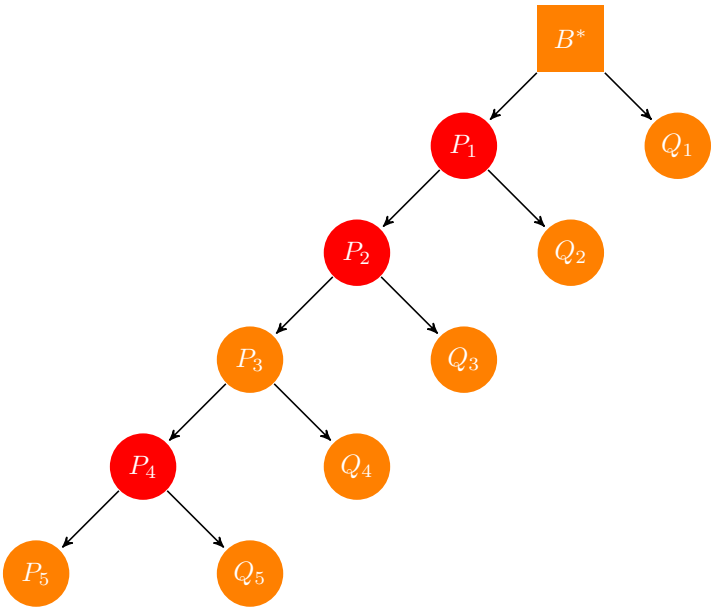
Így P_K komplementere, ha az utolsó feltételben egy változóra is $j \in B_k^* : x_j = 0$ áll fenn. Ezt egy egyszerű feltétellel is megfogalmazhatjuk:

$$\sum_{i \in B_k^*} x_i \leq |B_K^*| - 1$$

Ez azonban elronthatja a feladat szerkezetét: sokszor az oszlopgenerálás ezt a speciális szerkezetet tuja kihasználni, hogy gyorsan tudjon új oszlopot negatív redukált költséggel generálni.

Ezért is inkább $|K|$ részhalmazra fogjuk RMLP(c, l, u)-t szétbontani:

$$Q_k := RMLP(c, l, u) \cap \{x \in [0,1]^n | x_j = 1, j \in B_{k-1}^* \text{ és } x_i = 0, i \in B_k^* \setminus B_{k-1}^*\}$$



5. ábra. (Binary Search) csak a pirossal jelölt csúcsokat vizsgálja csak meg a Rapid Branching

Így tehát $\cup_{k=1}^{K}(Q_k \cup P_k) =$RMLP$(c, l, u)$.

Az algoritmus során az alábbi csúcsokat fogjuk végigvizsgálni (lásd az ábra):

$$x_j = l_j = 1, j \in B_k^*; \qquad k = K, \lceil K/2 \rceil, \lceil K/2 \rceil, ..., 2, 1.$$

Tehát a *Rapid Branching* algoritmusában:

$$S_{i+1} = (S_i \setminus \{N_i\}) \cup \{B^*_{\lceil K/2 \rceil}\} \cup \{B^*_{\lceil K/4 \rceil}\} \cup ... \cup \{B_2^*\} \cup \{B_1^*\}$$

szerint bontjuk szét $S_i + 1$-et. Így tehát ahelyett hogy sorra végigvizsgálnánk $2K$ csúcsot, csak ennek a logaritmusának megfelelő csúcsot vizsgálunk át.

3. Kitekintés

Ezen szakdolgozat első része csak közösségi közlekedési problémákat írt fel. Azonban a nem közösségi közlekedés is érdekes témákat rejt: mint például az autós közlekedési lámpákat hogyan állítsuk be. Hogyan állítsuk be a közlekedési lámpákat, hogy az adott menetrend szerinti busz illetve villamos járatok mindig zöld lámpát kapjanak? Ha nem lehetséges, lehet-e a menetrendet úgy módosítani, hogy ez lehetséges legyen?
Meg lehet-e oldani egy adott csomópontban egy adott forgalommal, hogy semelyik autónak se kelljen lassítania amelyik ott áthalad? Ha nem, átlagosan mennyit kell várakoznia?

A GPS-ek térnyerése kapcsán egyre fontosabb kérdés, ha mindegyik autós a legkevesebb idő alatt akar elérni kezdő és végcélja között, egy idő után mindig ugyan azon az útvonalon érdemes autóznia (szigorú Nash-egyensúly) ? Lehetséges-e bizonyos utcák kiszélesítésével, esetleg lezárásával Nash-egyensúlyt létrehozni? (lásd [11]).

Az önvezető autók szintén érdekes kérdéseket vethetnek fel. Tegyük fel, hogy az egyes résztvevők nem törekednek saját veszteségüket/menetidejüket minimalizálni (nem 'racionálisak'), vagy esetleg hajlandóak $p\%$ pluszveszteséget vállalni, hogy az összforgalom gyorsuljon. Ekkor mennyire lehet felgyorsítani a forgalmat?

Tehát van még mit tenni egy optimális közlekedés eléréséig...

Hivatkozások

[1] R. Borndörfer, M. Grötschel, and M. E. Pfetsch, *Computer-aided Systems in Public Transport*, ch. Models for Line Planning in Public Transport, pp. 363–378. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008.

[2] D. Huisman, „The new dutch timetable: The or revolution," *Interfaces*, vol. 39, pp. 6–17, 2009.

[3] P. W. George B. Dantzig, „Decomposition principle for linear programs," in *ATMOS*, 1960.

[4] D. Kim and C. Barnhart, *Computer-Aided Transit Scheduling: Proceedings, Cambridge, MA, USA, August 1997*, ch. Transportation Service Network Design: Models and Algorithms, pp. 259–283. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999.

[5] P. Serafini and Ukovich, „A mathematical for periodic scheduling problems," *SIAM J. Discret. Math.*, vol. 2, pp. 550–581, Nov. 1989.

[6] F. G. M. Fischetti and A. Lodi, „The feasibility pump," *Mathematical Programming*, p. 91–104, 2005.

[7] L. Bertacco, M. Fischetti, and A. Lodi, „A feasibility pump heuristic for general mixed-integer problems," *Discrete Optimization*, vol. 4, no. 1, pp. 63 – 76, 2007. Mixed Integer ProgrammingIMA Special Workshop on Mixed-Integer Programming.

[8] S. Weider, *Integration of Vehicle and Duty Scheduling in Public Transport*. PhD thesis, 2007.

[9] R. Borndörfer, T. Schlechte, and S. Weider, „Railway Track Allocation by Rapid Branching," in *10th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'10)* (T. Erlebach and M. Lübbecke, eds.), vol. 14 of *OpenAccess Series in Informatics (OASIcs)*, pp. 13–23, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2010.

[10] R. Borndörfer, A. Löbel, M. Reuther, T. Schlechte, and S. Weider, „Rapid branching," *Public Transport*, vol. 5, no. 1, pp. 3–23, 2013.

[11] E. Tardos, „Network games," in *Proceedings of the Thirty-sixth Annual ACM Symposium on Theory of Computing*, STOC '04, (New York, NY, USA), pp. 341–342, ACM, 2004.

[12] C. Liebchen and R. H. Möhring, *Algorithmic Methods for Railway Optimization: International Dagstuhl Workshop, Dagstuhl Castle, Germany, June 20-25, 2004, 4th International Workshop, ATMOS 2004, Bergen, Norway, September 16-17, 2004, Revised Selected Papers*, ch. The Modeling Power of the Periodic Event Scheduling Problem: Railway Timetables — and Beyond, pp. 3–40. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007.

[13] T. Berthold, „Primal heuristics for mixed integer programs," Master's thesis, Technischen Universität Berlin, 2006.

[14] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, and P. H. Vance, „Branch-and-price: Column generation for solving huge integer programs," *Oper. Res.*, vol. 46, pp. 316–329, Mar. 1998.