NYILATKOZAT

Név: Oláh Sarolta

ELTE Természettudományi Kar, szak: matematika

NEPTUN azonosító: DLJS5M

Szakdolgozat címe:
Algoritmusok az időablakos járműirányítási feladatra

A **szakdolgozat** szerzőjeként fegyelmi felelősségem tudatában kijelentem, hogy a dolgozatom önálló szellemi alkotásom, abban a hivatkozások és idézések standard szabályait következetesen alkalmaztam, mások által írt részeket a megfelelő idézés nélkül nem használtam fel.

Budapest, 2022.05.30.

a hallgató aláírása

Eötvös Loránd Tudományegyetem

Természettudományi Kar

Oláh Sarolta

Algoritmusok az időablakos járműirányítási feladatra

Szakdolgozat

Matematika Bsc, Alkalmazott matematikus szakirány

Témavezető:

Tamási Tímea

Operációkutatás tanszék

Budapest, 2022

Köszönetnyilvánítás

Ezúton szeretném megköszönni témavezetőmnek, Tamási Tímeának az egész éves segítségét. Köszönöm a sok időt, befektetett energiát, és azt, hogy hasznos cikkekkel, észrevételekkel és türelmes magyarázatokkal segítette szakdolgozatom létrejöttét.

Tartalomjegyzék

1. fejezet

Bevezetés

A járműirányítási feladat (VRP - Vehicle Routing Problem) olyan problémákra keres megoldást, melyekben a feladat termékek szállítása egy vagy akár több raktártól a vásárlókig. A feladat NP-teljes, az NP-teljes utazó-ügynök probléma (TSP - Traveling Salesman Problem) általánosítása, ahol a cél megtalálni a legrövidebb kört, mely minden vásárlót pontosan egyszer meglátogat. Először G. B. Dantzig és J. H. Ramser [4] definiálta 1959-ben, akkor még Truck Dispatching Problem néven. Megfelelő modellezéssel megoldást talál rengeteg valós kiszállítási problémára, kiemelkedő a gyakorlati haszna, emiatt rengeteg típusa létezik. Alkalmazása többek között előfordul a szemétszállításban, iskolabuszok irányításában és szállító cégeknél.

Általánosan a kiszállítást néhány *jármű*vel végezzük, melyek akár különbözőek is lehetnek, különböző *kapacitás*okkal. A járművek egy vagy több *depó*ból indulnak, kiszolgálják a *fogyasztókat*, majd visszatérnek a depóba. Előfordul, hogy menetidejüket korlátozzuk. Ezt a mozgást egy *úthálózat*on hajtják végre. Feladatunk, hogy megadjuk a - valamilyen célfüggvény szerint - optimális útvonalak halmazát, melyek a depóban indulnak és végződnek. Ezeken végighaladva a járművek kielégítik a hozzájuk rendelt fogyasztók igényeit és teljesítik a kapacitásra és menetidőre tett feltételeket.

Az úthálózatot egy gráffal reprezentáljuk, ez lehet irányított vagy irányítatlan. Csúcsai a fogyasztóknak és a depónak, élei az úthálózat útjainak és utcáinak felelnek meg. Az élekhez *élköltség*eket rendelünk, ezek a két csúcs közötti út hossza, a jármű gázolaj fogyasztása vagy az utazási idő. Ezek alapján a gráfot ezentúl teljes gráfnak tekintjük, egy élének költsége a két csúcs közti legrövidebb út hossza, menetideje pedig az utcák menetidejének összege.

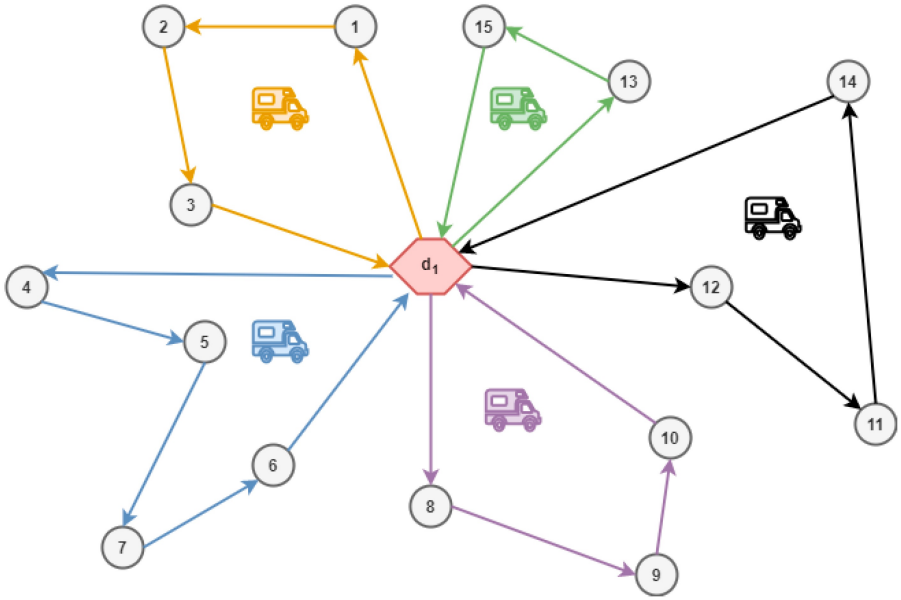
A legegyszerűbb típusfeladat a kapacitásos járműirányítási feladat, vagy röviden CVRP (Capacitated Vehicle Routing Problem). A problémában egyetlen depó van és n darab fogyasztó, a járművek mind egyformák, egyforma kapacitással, és minden csúcs között adott az utazási költség, mint élköltség. Ha a kapacitást végtelennek választjuk és egyetlen jármű áll rendelkezésre, az utazó ügynök feladatot kapjuk vissza.

A CVRP feladatnak több szélesen kutatott speciális esete van. Ilyen például a járműirányítás felvétellel vagy lerakással, röviden VRPPD (Vehicle Routing Problem with Pickup and Delivery), ahol minden fogyasztónak van kiszállítási és elszállítási igénye is, ezeket az igényeket külön tároljuk, a járműveknek mindkettőt figyelembe kell venni a kiszolgálás során.

Egy másik CVRP problémakör az időablakos járműirányítási feladat, röviden VRPTW (Vehicle Routing Problem With Time Windows), amit ebben a dolgozatban alaposabban vizsgálunk.

VRPTW-ben az eddigi megkötéseink (jármű kapacitása, a csúcsok igényei) mellett, minden fogyasztóhoz tartozik egy időablak, és csak azon az időintervallumon belül szolgálhatjuk ki. Ha egy jármű a legkorábbi időpont előtt érkezik a vásárlóhoz, várnia kell. Legtöbbször ez a várakozás költséges, ezért igyekszünk elkerülni. További CVRP és VRP fajták leírása megtalálható Paolo és Vigo járműirányítási feladatokról szóló könyvében [10].

A második fejezetben bemutatásra kerül néhány általános lineáris programozási algoritmus, amelyeket a későbbiekben alkalmazni fogunk, utána a feladat több matematikai modelljét írjuk le. A negyedik fejezetben két algoritmus szerepel, az egyik egy branch and price, a másik egy branch and cut algoritmus. Végül bemutatunk néhány heurisztikát a feladatra.



1.1. ábra. A járműirányítási feladat [8]

.

2. fejezet

Algoritmikai bevezető

Ebben a fejezetben bemutatunk három fontos eljárást, amik segítségével hatékonyan lehet nagyméretű lineáris programozási illetve egészértékű programozási feladatokat megoldani. A későbbiekben ezeknek a módszereknek a VRPTW-re vett alkalmazását is látni fogjuk. A leírásokhoz felhasználtuk Dimitris Bertsimas és John N. Tsitsiklis lineáris programozásról szóló könyvét [2] és Marco E. Lübbecke cikkét az oszlopgenerálásról [9].

2.1. Oszlopgenerálás

Adott a

$$\begin{aligned} \min\ & cx \\ & Ax \geq b, \\ & x \geq 0, \end{aligned}$$

feladat, ahol $A \in \mathbb{R}^{m \times n}$ olyan sok oszlopból áll, hogy egy ekkora mátrix tárolása és a feladat megoldása szimplex algoritmussal lehetetlen lenne [9].

Az ötlet az, hogy nem tároljuk az egész mátrixot, csak a bázist, ami az eredeti mátrixunk néhány oszlopa, és erre alkalmazzuk a primál szimplex algoritmust. Az algoritmusnak csak a tárolt bázisra van szüksége.

Legyen A' az aktuális, néhány A-beli oszlopot tartalmazó mátrix, x^* az erre vett primál optimum, y^* a hozzá tartozó duál megengedett megoldás. Az x^*-ot nullákkal kiegészítve egy primál-megengedett megoldást kapunk az eredeti feladatra nézve. A megoldás optimalitását szeretnénk ellenőrizni. Ha $(x^*, 0)$-ra teljesül, hogy minden redukált költség nemnegatív, akkor kész vagyunk, optimális megoldáshoz jutottunk. Különben frissítjük a bázist, amihez szükségünk van egy oszlopra, ami belép, illetve ami majd kilép belőle. Utóbbi meghatározása egyszerű. A belépő oszlop kereséséhez pedig a következő árazási feladatot (pricing problem) kell megoldani:

$$\bar{c} = \min\{c_j - y^* a_j : 1 \leq j \leq n\}.$$

Ahol a_j A j-edik oszlopa. Abban az esetben, ha $\bar{c} \geq 0$, további iterációk végrehajtására nincs

szükség, megtaláltuk az optimális megoldást. Ha a fenti feladat optimális megoldása olyan j, amelyre $c_j - y^* a_j < 0$, akkor a_j-t A'-hoz vesszük, majd a következő iterációra lépünk.

Fontos megjegyeznünk, hogy $\bar{c}$ kiszámításához nem kell végignéznünk az egész feladatot, ugyanis általában egy jól felépített optimalizálási problémához vezet, ahol a megoldáshoz hatékony algoritmusokkal rendelkezünk.

2.2. Vágósíkos algoritmus

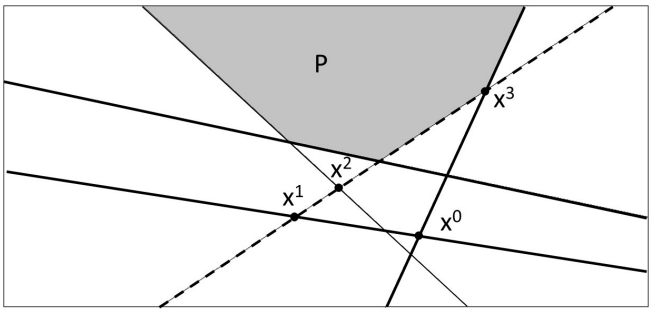
A következő egészértékű programot szeretnénk megoldani:

$$\begin{aligned} &\min\ cx \\ &Ax = b, \\ &x \geq 0, \\ &x \in \mathbb{Z}. \end{aligned} \tag{2.1}$$

Ehhez először tekintsük a lineáris relaxált feladatot:

$$\begin{aligned} &\min\ cx \\ &Ax = b, \\ &x \geq 0. \end{aligned} \tag{2.2}$$

Legyen x^* a (2.2) optimális megoldása. Ha x^* egész, akkor optimumot kaptunk (2.1)-re is. Ha nem, akkor keresünk olyan egyenlőtlenségeket, amiket az eredeti feladat minden egész megoldásának teljesíteni kell, de x^* nem teljesíti, azaz oldjunk meg egy úgynevezett szeparációs feladatot. A sértő egyenlőtlenségeket hozzávesszük a (2.2) feladathoz, és újraoptimalizálunk.



2.1. ábra. Vágósíkos algoritmus [2]

A 2.1 ábrán P néhány egyenlőtlenséggel definiált poliéder. Válasszuk ki a három vastag egyenes által definiált egyenlőtlenségeket, ekkor az x^0 megoldáshoz jutunk, ami nem része a poliédernek. Ha vesszük a szaggatott vonal által határolt egyenlőtlenséget, akkor láthatjuk, hogy a többi megoldás ezt teljesíti, de x^0 nem, azaz levágjuk vele x^0-t.

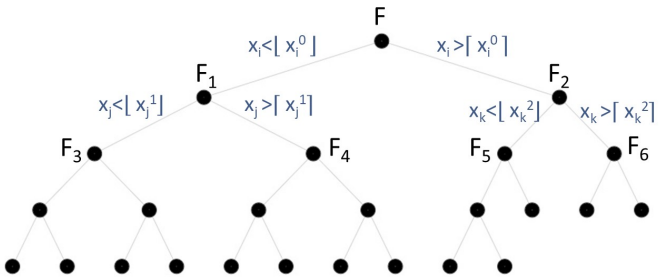
Nagy szabadságot ad a megoldásban, hogy sokféle vágást nézhetünk és hozzáadhatunk, ez befolyásolja az algoritmus hatékonyságát is. Konkrét feladatoknál több ötletet, akár heurisztikákat is alkalmaznak a minél jobb egyenlőtlenségek megtalálásához.

Fontos megjegyezni, hogy a vágósíkos algoritmus nemcsak egészértékű feladatok megoldásánál hasznos, hanem nagyméretű lineáris programozási feladatok megoldásánál is, ahol az oszlopgenerálással ellentétben nem a változók száma, hanem a feltételeké a nagy. Azaz A-nak nagyon sok sora van. A néhány feltételt tartalmazó A' részfeladatot megoldjuk, a megoldásánál kapott optimumot jelöljük x'-vel. Ezután olyan feltételeket keresünk, amiket az eredeti feladat x^* megoldása teljesít, de x' nem, majd ezeket a sorokat A'-höz véve újraoptimalizálunk.

A dualitás tételének következményeként megfigyelhetjük, hogy az oszlopgenerálás és a vágósíkos algoritmus tulajdonképpen egymásnak a duálisai: a primál feladatra az egyiket alkalmazva ugyanazt kapjuk, mintha a duális feladatra a másikat alkalmaznánk.

2.3. Branch and bound

A branch and bound, vagy magyarul korlátozás és szétválasztás, gyakran használt technika az egészértékű feladatok megoldásánál. Az oszd meg és uralkodj elvén működik. Az eredeti problémát részfeladatokra osztjuk fel, a részfeladatokat pedig még további részfeladatokra: ez a branch/szétválasztás része az eljárásnak, a részfeladatok fáját mutatja az ábra.



2.2. ábra. Branch and bound algoritmus, részfeladatok fája.

Mivel a részfeladatok optimális megoldását sokszor ugyanolyan nehéz megtalálni, mint az eredeti feladatét, ezért minden részfeladatra alsó korlátokat számolunk ki, ez a bound/korlátozás rész. Feltételezzük, hogy egy hatékony algoritmussal tudunk alsó becsléseket adni a részfeladatokra, ez általában a relaxált feladatok optimális megoldása.

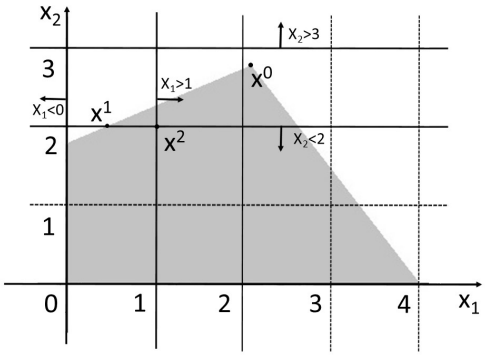
Ha egy adott részfeladatra megtalált alsó korlát nagyobb, mint az addigi megtalált legjobb megoldásunk, akkor azt a részfeladatot a továbbiakban nem kell nézni, hiszen ekkor a részfeladat optimuma sem lehet jobb, mint az eddigi legjobb megoldás.

Legyen U az eddigi legjobb megoldásunk (ez kezdetben végtelen), és legyen $F = \bigcup F_i$ a részfeladatok halmaza. Az algoritmus általános lépése a következő:

1. Vegyünk ki egy F_i részfeladatot F-ből.

2. Ha a részfeladat nem megoldható, akkor töröljük, különben számoljuk ki az alsó korlátot: $b(F_i)$ (a relaxált optimum).
3. Ha $b(F_i) \geq U$, akkor töröljük a részfeladatot, hiszen a jelenlegi legjobb egész megoldásunk is jobb mint a talált törtoptimum.
4. Ha $b(F_i) < U$ és relaxált feladat optimuma egész, akkor jegyezzük meg ezt a megoldást, és legyen $U := b(F_i)$.
5. Ha $b(F_i) < U$ és relaxált feladat optimuma nem egész, bontsuk tovább a részfeladatot további részfeladatokra amiket később megvizsgálunk, ezeket tegyük F-be.

A részfeladatok alsó becslésére, a feldolgozás sorrendjére és a részfeladatokra bontásra is nagyon sok megoldás és ötlet létezik. Azt, hogy melyik a legjobb, az adott feladat típusa és a tapasztalat határozza meg.



2.3. ábra. Példa branch and bound-ra [2]

A 2.3 ábrán egy egyszerű branch and bound algoritmus látható. A kezdeti megoldásunk x^0, ami mindkét változójában tört. Kiválasztjuk a második változóját, az alsó és felső egészrésze szerint választjuk szét két külön részfeladatra, az egyik részfeladatban $x_2 > 3$, ez a poliéder üres, így nem kell vele foglalkoznunk, a másikban $x_2 < 2$. Utóbbit hozzávéve a feladathoz és újraoptimalizálva az x^1 megoldást kapjuk, neki az első változója szerint hajtjuk végre ugyanazon szétválasztásokat.

A branch and bound algoritmusnak van két speciális esete, amit a szakdolgozat további részében is tárgyalni fogunk:

Branch and price

A branch and bound algoritmusban az alsó korlátokat általában a lineáris relaxált feladat optimumával adjuk meg. Ha a feladat változóinak száma nagy, akkor használhatunk oszlopgenerálást a kiszámolásukra. Ez alsó korlátot ad a relaxált részfeladatra, így magára a branch and bound keresési fájában lévő aktuális részfeladatra is.

Branch and cut

A módszer a fenti eljáráshoz hasonlóan a branch and bound algoritmus kis módosítása, de itt nem az oszlopgenerálást, hanem a vágósíkos algoritmust alkalmazzuk. Amikor a részfeladatra

kiszámoltuk a lineáris relaxált alsó korlátját, akkor ezután a szeparációs algoritmus segítségével keresünk sértő feltételeket, ezeket a részfeladathoz tesszük, és ezzel folytatjuk az eljárást.

3. fejezet

A VRPTW feladat leírása

Ebben a fejezetben az időablakos járműirányítási feladat (VRPTW) különböző felírásait mutatjuk be. Adott n fogyasztó, akiknek az igényeit szeretnénk kiszolgálni adott időintervallumokon belül.

A problémát egy irányított gráfon definiáljuk, legyen ez $G = (V_0, A)$, ahol V az $\{1, \ldots, n\}$ fogyasztókat tartalmazó csúcshalmaz, V_0 pedig V uniója a depóval. Legyen Ω a megengedett utak és K a járművek halmaza, a legtöbb feladatban $|K|$ értékét nem korlátozzuk, de előfordul, hogy a járművek száma fix, sőt, az is, hogy pont ezt szeretnénk minimalizálni. A járművek rendelkeznek egy Q kapacitással.

Minden i csúcshoz tartozik egy q_i követelés, ami megmondja, hogy mennyi terméket kell ahhoz a csúcshoz szállítani; egy s_i kiszolgálási idő (ettől az esetek egy részében eltekintünk, azaz nullának vesszük) és egy $[a_i, b_i]$ időablak, ami megadja, hogy az adott csúcsot legkorábban és legkésőbb mikor szolgálhatjuk ki.

Az egyszerűség kedvéért a depóra nincsenek követeléseink, az időablakát úgy állítjuk be, hogy a_0 egy jármű lehető legkorábbi indulását, b_0 a lehető legkésőbbi érkezését mutassa. A depót reprezentálhatjuk két csúccsal: 0 a forrás, és $n + 1$ a nyelő. Így a megengedett útvonalak a forrásból a nyelőbe menő utak. Minden (i, j) élre adott egy c_{ij} költség, ez legtöbbször a két csúcs távolsága, valamint τ_{ij} az utazási idő i csúcsból j-be.

3.1. Matematikai modellek

Ebben a fejezetben bemutatunk néhány sokat használt matematikai modellt a VRPTW feladatra. A modellek közül kettőt a 4. fejezetben leírt algoritmusok is használnak.

3.1.1. Vegyes-egészértékű lineáris programozási feladat (MILP)

Elsőként két vegyes-egészértékű lineáris programozási modellt mutatunk be. Ezekben a felírásokban néhány változó egészértékű, néhány pedig tetszőleges valós értéket is felvehet (MILP - Mixed-Integer Linear Program). Az első modell ([1]) egy két-indexes x_{ij}, a másik modell ([10]) egy három-indexes x_{ijk} bináris változót használ, ezek mellett további segédváltozókat definiálva.

Minden ij élre legyen az x_{ij} bináris változó a következő: x_{ij} pontosan akkor 1, ha a megoldásban használjuk az (i, j) élet, azaz egy jármű áthalad rajta, és különben 0. Ezen kívül jelöljük t_i-vel az i csúcs elhagyásának időpontját, és y_i-vel a jármű i-ig lerakott terhét. Megjegyezzük, hogy y_i definícióján keveset módosítva a feladat egyszerűen átírható pickup and delivery problémává, ahol nem csak kiszállítani, de felvenni is tudunk termékeket.

A vegyes-egészértékű lineáris programunk így a következő lesz:

$$\min \sum_{i=1}^{n} c_{ij} x_{ij} \tag{3.1}$$

$$\sum_{j \in V_0} x_{ij} = 1 \qquad \forall\, i \in V, \tag{3.2}$$

$$\sum_{j \in V_0} x_{ij} - \sum_{j \in V_0} x_{ji} = 0 \qquad \forall\ i \in V_0, \tag{3.3}$$

$$t_j \geq t_i + \tau_{ij} x_{ij} - (b_i - a_j)(1 - x_{ij}) \qquad \forall\, i, j \in V, \tag{3.4}$$

$$y_j \geq y_i + q_j - (Q - q_j)(1 - x_{ij}) \qquad \forall\, i, j \in V, \tag{3.5}$$

$$q_i \leq y_i \leq Q \qquad i \in V, \tag{3.6}$$

$$a_i \leq t_i \leq b_i \qquad \forall\, i \in V, \tag{3.7}$$

$$x_{ij} \in \{0, 1\} \qquad \forall\, i, j \in V_0. \tag{3.8}$$

A teljes útvonal költségét szeretnénk minimalizálni (3.1)-ben. (3.2)-(3.3) garantálja, hogy minden csúcsot pontosan egyszer látogassunk meg egy úton, és minden csúcsból annyiszor menjünk ki, ahányszor bementünk. (3.4) megértéséhez vegyük először azt az esetet, ha $x_{ij} = 1$, ekkor az egyenlőtlenség a $t_j \geq t_i + \tau_{ij} x_{ij}$ alakra egyszerűsödik le, ami pont azt jelenti, hogy ha i-ből j-be megyünk, akkor j elhagyásának ideje nem lehet hamarabb mint i elhagyása plusz az utazási idő, ez minden megengedett úthoz szükséges. Abban az esetben, ha $x_{ij} = 0$ az egyenlőtlenség $t_j \geq t_i - (b_i - a_j)$, ami a (3.7) teljesülésével mindig igaz, nem ad új feltételt. Ugyanis ha i elhagyási idejét a lehető legkésőbbi időpontra, b_i-re állítjuk, azzal a jobb oldalt csak növeljük, az eredmény pedig $t_j \geq a_j$. A (3.5) hasonlóan átgondolható, az egymást követő követelésekre ad korlátozást. (3.6) és (3.7) a kapacitáskorlátot és az időablakokat tartatja be minden csúcsra. A bináris egészértékűséget (3.8) követeli meg.

Tekintsük most a másik modellt: itt a járműveket is indexelni fogjuk, egy járművet $k \in K$-val jelölünk. Legyen a bináris változónk x_{ijk}, ami pontosan akkor 1, ha az (i, j) élet használjuk a k. járművel a megoldásban, különben 0. Minden csúcsra és minden járműre számon tartunk egy időváltozót T_{ik}-t, ami megmondja, hogy pontosan mikor kezdtük el az i csúcs kiszolgálását a k. járművel.

A feladat tehát:

$$\min \sum_{k\in K} \sum_{(i,j)\in A} c_{ij} x_{ijk} \tag{3.9}$$

$$\sum_{k\in K} \sum_{j\in\delta^+(i)} x_{ijk} = 1 \qquad \forall\, i \in V, \tag{3.10}$$

$$\sum_{j\in\delta^+(0)} x_{0jk} = 1 \qquad \forall\, k \in K, \tag{3.11}$$

$$\sum_{i\in\delta^-(j)} x_{ijk} - \sum_{i\in\delta^+(j)} x_{ijk} = 0 \qquad \forall\, k \in K, j \in V, \tag{3.12}$$

$$\sum_{i\in\delta^-(n+1)} x_{i,n+1,k} = 1 \qquad \forall\, k \in K, \tag{3.13}$$

$$x_{ijk}(T_{ik} + s_i + \tau_{ij} - T_{jk}) \leq 0 \qquad \forall\, k \in K, (i,j) \in A, \tag{3.14}$$

$$a_i \leq T_{ik} \leq b_i \qquad \forall\, k \in K, i \in V, \tag{3.15}$$

$$\sum_{i\in N} q_i \sum_{j\in\delta^+(i)} x_{ijk} \leq Q \qquad \forall\, k \in K, \tag{3.16}$$

$$x_{ijk} \in \{0,1\} \qquad \forall\, k \in K, (i,j) \in A. \tag{3.17}$$

(3.9) az összköltség minimalizálása. (3.10) biztosítja, hogy minden csúcsot kiszolgáljunk pontosan egyszer. Azt, hogy minden jármű pontosan egyszer hagyja el a kiindulási pontot és pontosan egyszer térjen vissza, (3.11)-al és (3.13)-el érjük el. (3.12)-re azért van szükség, hogy sehol ne akadjon el jármű, vagyis ugyanannyiszor menjen be és jöjjön ki egy csúcsból. (3.14) és (3.15) veszi figyelembe az időt. Előbbi kizárja, hogy időutazók legyünk és hamarabb szolgáljunk ki egy csúcsot mint az odaérkezés ideje, utóbbi az időablakok betartására szolgál. (3.16) az egyes járműkapacitásokat tartatja be és (3.17)- (3.8)-hoz hasonlóan biztosítja az egészértékűséget.

3.1.2. Halmazpartíciós felírás

A halmazpartíciós felíráshoz ([5]) legyen Ω az összes megengedett forrás-nyelő út halmaza. Egy Ω-beli r út költsége c_r. Legyen δ_{ir} értéke 1, ha az r úton meglátogattuk az i-edik pontot, és különben 0. Az utak ismeretében ezeket az értékeket is ismerjük.

Minden úthoz definiálunk egy x_r változót, ami 1, ha az r utat használjuk a megoldásban és 0, ha nem. $T_i = \sum_{k\in\Omega} T_{ik}$, és t_{ij}-t az előző leíráshoz hasonlóan definiáljuk.

Egy utat megengedett útnak tekintünk, ha teljesíti a következő feltételeket:

- Az út kezdőpontja a 0 forrás, végpontja az $n+1$ nyelő.
- Az egy úton lévő csúcsok összigénye nem haladja meg a jármű kapacitását.
- $T_i + t_{ij} \leq T_j$ és $a_i \leq T_i \leq b_i$ teljesül minden, az úton lévő (i,j) élre.

A modell így:

$$\min \sum_{r \in \Omega} c_r x_r \tag{3.18}$$

$$\sum_{r \in \Omega} \delta_{ir} x_r = 1 \qquad \forall\, i \in N, \tag{3.19}$$

$$x_r \in \{0, 1\}, \qquad \forall\, r \in \Omega. \tag{3.20}$$

(3.18) a költségek minimalizálására szolgál. (3.19) éri el, hogy minden pontot pontosan egyszer szolgáljunk ki, míg (3.20) a bináris megkötés. Néhány feladattípusban azt is megköveteljük, hogy a megoldás közben legfeljebb $|K|$ járművet kelljen felhasználni. Olyankor hozzávesszük a következő feltételt: $\sum_{r \in \Omega} x_r \leq |K|$.

3.1.3. Halmazfedési feladat

Legyen γ_{ir} egy konstans, ami megadja, hogy az r út hányszor érinti az i csúcsot. A halmazpartíciós feladathoz definiált δ_{ir}-rel ellentétben γ_{ir} nem bináris, hanem egészértékű.

A korábban definiált jelölésekhez még két további változót vezetünk be: legyen X_d az utak (járművek) száma, X_c pedig az utak összköltsége a megoldásban. X_c egészértékűségét csak az összes c_r egészértékűségével érhetjük el, amit $c_{ij} \in \mathbb{N}\ \forall\ (i, j) \in A$ teljesít. Ebben a felírásban megengedjük, hogy egy csúcsot akár többször is meglátogassunk.

$$\min \sum_{r \in R} c_r x_r \tag{3.21}$$

$$\sum_{r \in R} \gamma_{ir} x_{ir} \geq 1 \qquad \forall\, i\ \in N \setminus \{d\}, \tag{3.22}$$

$$\sum_{r \in R} x_r - X_d = 0, \tag{3.23}$$

$$\sum_{r \in R} c_r x_r - X_c = 0, \tag{3.24}$$

$$x_r \in \{0, 1\} \qquad \forall\, r \in R, \tag{3.25}$$

$$0 \leq X_d, X_c \in \mathbb{N}. \tag{3.26}$$

(3.21) a szokásos minimalizálási feladat. Minden csúcsot meglátogatunk legalább egyszer (3.22) miatt. (3.23) és (3.24) beállítja X_c és X_d értékeit. Az x_r egy bináris változó, ezt (3.25) garantálja. A költségfüggvény, valamint a (3.22) és (3.25) feltételek adják a feladat halmazfedési jellegét.

Fontos megjegyezni, hogy bár a halmazpartíciós modell végiggondolása sokkal egyszerűbb, a halmazfedési feladat a gyakorlatban könnyebben kezelhető, és numerikusan is stabilabb [5].

4. fejezet

Egzakt algoritmusok

4.1. Branch and price

1992-ben Desrochers, Martin, Jacques Desrosiers, és Marius Solomon [5] kiadott egy új optimalizációs algoritmust a VRPTW-re. A feladatot a (2.3) fejezetben már ismertetett branch and price algoritmussal oldják meg, egy olyan árazási feladatot adva hozzá, ami a korábbi megoldásokhoz képest sokkal nagyobb problémákat képes optimálisan megoldani. Ezt mutatjuk be ebben a fejezetben.

Először tekintsük át magát az algoritmust, később az egyes részeket külön-külön is kifejtjük. Oszlopgenerálással megoldjuk a feladat LP relaxáltját. Ehhez a (4.1.2) fejezetben használunk egy algoritmust, aminek segítségével meghatározhatjuk az utak marginális költségét. Ha minden marginális költség pozitív, akkor a megoldás optimális lesz LP-re, különben a sértő oszlopokat hozzávesszük a feladathoz. A kapott relaxált megoldásokat a branch and bound algoritmushoz használjuk, melyben kezeljük a nem megfelelő és tört megoldásokat, a keresőfában új ágakat hozunk létre az élekre kapott x_e értékek szerint és kialakítjuk a végleges megoldást.

4.1.1. A részfeladat definiálása

A (3.1.2) felírásban már definiáltuk, milyen egy megengedett út, most dinamikus programozással próbáljuk kiszűrni a nem megengedetteket. A megoldásban áttérünk a gyakorlatban sokkal alkalmazhatóbb (3.1.3)-ban ismertetett felírásra. Természetesen a kapott eredmény megoldása az eredeti modellnek is.

A következő dinamikus program egy két kör eliminációs procedúra (2-cycle elimination). Ez azt jelenti, hogy elkerüljük a párhuzamos éleket, egy csúcsból nem megyünk közvetlenül vissza az előzőbe. Ötlete, hogy két részutat is számon tartunk a forrásból minden egyes csúcsba, ezek a legjobb és a második legjobb részutak.

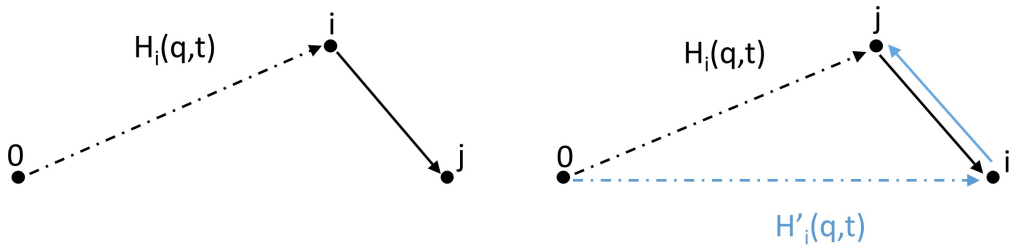
Legyen $H_i(q,t)$ a minimális marginális költsége (minimal marginal cost) a forrásból i-be menő legjobb részútnak, ami eddig q mennyiségű terméket szolgált ki és i-t legkorábban t időben képes elhagyni. $p_i(q,t)$ az i pont szülője azon az úton, amit a $H_i(q,t)$ költséggel azonosítunk. Legyen továbbá $H'_i(q,t)$ a marginális költsége a legjobb forrásból i-be menő útnak, amin az eddig felhalmozott termék mennyisége q, leghamarabb t időben képes elhagyni i-t és az i csúcs

szülője nem $p_i(q,t)$. A definíciókból adódik, hogy $H_i(q,t) \leq H'_i(q,t)$. $H_j(q,t)$-t és $H'_j(q,t)$-t a következő dinamikus programmal tudjuk kiszámolni:

$H_0(0,0) = 0$, és minden $j \in N, q_j \leq q \leq Q, a_j \leq t \leq b_j$-re:

$$H_j(q,t) = \min_{(i,j)\in A} \{[H_i(q',t') + \overline{c}_{ij} | j \neq p_i(q',t'), t' + t_{ij} \leq t, a_i \leq t' \leq b_i \text{ és } q' + q_j \leq q],$$
$$[H'_i(q',t') + \overline{c}_{ij} | j = p_i(q',t'), t' + t_{ij} \leq t, a_i \leq t' \leq b_i \text{ és } q' + q_j \leq q]\},$$
$$H'_j(q,t) = \min_{(i,j)\in A} \{[H_i(q',t') + \overline{c}_{ij} | j \neq p_i(q',t'), i \neq p_j(q,t), t' + t_{ij} \leq t, a_i \leq t' \leq b_i \text{ és } q' + q_j \leq q],$$
$$[H'_i(q',t') + \overline{c}_{ij} | j = p_i(q',t'), i \neq p_j(q,t), t' + t_{ij} \leq t, a_i \leq t' \leq b_i \text{ és } q' + q_j \leq q]\}.$$

A 4.1 ábra szemlélteti a $H_j(q,t)$ és $H'_j(q,t)$ kiszámítását. A szaggatott nyíl jelöli a 0-ból i-be menő utat, útközben több csúcsot is bejárhatunk. A kettő hosszú köröket minden lépésben elkerüljük, hiszen olyan (i,j) élek vizsgálatakor, mikor j szülője i-nek a legrövidebb megfelelő úton, vesszük a második legrövidebbet. Ez jól definiált, ugyanis $H'_j(q,t)$ számításában sem i, sem j irányából nem engedjük meg a szülőséget, és csak olyan utakat választunk, amik eleget tesznek a feltételeknek. Iterációnként csak egy csúcs címkéit kell frissítenünk.



4.1. ábra. $H_j(q,t)$ és $H'_j(q,t)$ számítása

Egy fontos jelölést eddig még nem definiáltunk, a marginális költséget, $\overline{c}_{ij}$-t. Ehhez írjuk fel (3.1.3) duálisát, ami a következő:

$$\begin{aligned} \max & \sum_{i\in V_0} \pi_i \\ & \pi_i \gamma_{ir} + \pi_d + \pi_c c_r \leq c_r, \\ & -\pi_d \geq 0, \\ & -\pi_c \geq 0. \end{aligned}$$

Innen a második sor átrendezésével leolvasható, hogy

$$\overline{c}_r = c_r - \sum_{i\in V} \pi_i \gamma_{ir} - \pi_d - \pi_c c_r \geq 0.$$

Az út költségét az élköltségek összegével definiáltuk, és a kezdő és végpont is a depó, így a költség:

$$\overline{c}_r = \sum_{(i,j)\in r} [(1-\pi_c)c_{ij} - \pi_i]. \tag{4.1}$$

Most már (4.1) segítségével tudjuk definiálni a marginális költséget egy (i,j) élre:

$$\overline{c}_{ij} = (1-\pi_c)c_{ij} - \pi_i, \forall\ (i,j) \in A.$$

4.1.2. A részfeladat megoldása Pulling algoritmussal

Az előzőekben definiált dinamikus programot felhasználjuk a következő algoritmusban. Az algoritmus lényege, hogy alsó és felső becsléseket adunk $H_j(q,t)$-re és $H'_j(q,t)$-re, ezekkel a becslésekkel címkézzük fel az összes csúcsot. Ezeket addig módosítjuk, míg elérjük az optimális megoldást, az alsó és felső becslés egyezését.

Jelölje $\underline{H}_j(q,t)$ és $\overline{H}_j(q,t)$ az alsó és a felső becslést (H'-re hasonlóan), P_j pedig azon (q,t) állapotok halmaza a j csúcsra, ahol az alsó és felső becslés megegyezik. Jelöljük $\overline{P}_j$-vel P_j komplementerét (azaz azon állapotok halmazát, ahol szigorú egyenlőtlenség teljesül).

Definíció. *Két állapot,* (q_1,t_1) *és* $(q_2,t_2) \in \mathbb{R}^2$ *közül* (q_1,t_1) *lexikografikusan kisebb* (q_2,t_2)*-nél pontosan akkor, ha* $q_1 \leq q_2$ *és* $t_1 < t_2$.

1. Inicializálás. Beállítjuk a következőket:

$\underline{H}_j(q,t) = -\infty$, $\overline{H}_j(q,t) = \infty$, $\overline{H}'_j(q,t) = \infty$.
$p_j(q,t) =$ nil, minden $j \in V \setminus \{0\}$, $0 \leq q \leq Q$, és $a_j \leq t \leq b_j$-re.
$\overline{H}_0(0,0) = \underline{H}_0(0,0) = 0$.
$P_j = \emptyset$, $j \in N \setminus \{0\}$-ra és $P_0 = \{(0,0)\}$.

2. Megkeressük a lexikografikusan legkisebb (q,t) állapotot a $W = \bigcup_{j \in N}(\overline{P}_j)$ halmazból. Ha $W = \emptyset$, STOP.

3. Használjuk a korábban ismertetett dinamikus programot:

$$\overline{H}_j(q,t) = \min_{(i,j)\in A}\{[\overline{H}_i(q',t') + \overline{c}_{ij} | j \neq p_i(q',t'), t' + t_{ij} \leq t, a_i \leq t' \leq b_i \text{ és } q' + q_j \leq q],$$
$$[\overline{H}'_i(q',t') + \overline{c}_{ij} | j = p_i(q',t'), t' + t_{ij} \leq t, a_i \leq t' \leq b_i \text{ és } q' + q_j \leq q]\}$$

Beállítjuk szülőnek $p_j(q,t)$-t.

$$\overline{H}'_j(q,t) = \min_{(i,j)\in A}\{[\overline{H}_i(q',t') + \overline{c}_{ij} | j \neq p_i(q',t'), i \neq p_j(q,t), t' + t_{ij} \leq t, a_i \leq t' \leq b_i \text{ és } q' + q_j \leq q],$$
$$[\overline{H}'_i(q',t') + \overline{c}_{ij} | j = p_i(q',t'), i \neq p_j(q,t), t' + t_{ij} \leq t, a_i \leq t' \leq b_i \text{ és } q' + q_j \leq q]\}.$$
$$\underline{H}_j(q,t) = \min_{(i,j)\in A}\{\underline{H}_i(q',t') + \overline{c}_{ij} | t' + t_{ij} \leq t, a_i \leq t' \leq b_i \text{ és } q' + q_j \leq q\}$$

Frissítsük P_j-t és térjünk vissza a 2. lépésre.

4.1.3. A branch and bound algoritmus

Ha az algoritmus első lépésében már nem generálunk több negatív marginális költségű utat, akkor megkaptuk az LP relaxált optimumát. Ha ez nem egész, vagy a megoldásban nem szolgálunk ki minden csúcsot pontosan egyszer (többször szolgáljuk ki, vagy egyszer sem), akkor szükséges bejárni a branch and bound fát.

Általános esetben már láttuk, hogyan működik az algoritmus a (2.3) fejezetben. Most a szétválasztás módszereit két kategóriába soroljuk. Az első, ha a használt járművek száma a szimplex algoritmussal kapott megoldásban tört, $X_d = \vartheta$, két ágra bontjuk szét, az egyik $X_d \leq \lfloor \vartheta \rfloor$, a

másik $X_d \geq \lceil \vartheta \rceil$. Ha még mindig tört megoldást kapunk, akkor a következő ágon egy korlátozást hajtunk végre (csökkentjük az eseteket). Ezt az össztávon tesszük, ha $X_c = c$, akkor mostantól csak az olyan utakat nézzük, melyekre $X_c < \lceil c \rceil$. Végrehajtjuk a branch and bound fa (lásd kép fentebb) minden szükséges részén.

A második kategóriában az éleket nézzük. Először a kört tartalmazó utakat választjuk ki (kettő hosszú köreink nincsenek, de mások lehetnek), és pontozzuk azokat az éleket, amik olyan csúcshoz kapcsolódnak, amit többször látogatunk meg. A pontszám $x_e = \sum_{r:e \in r} x_r$, ahol az r utat használjuk a (4.1.2)-ban kapott megoldásban. Kiválasztjuk a legjobb pontszámmal rendelkező élet és ennek mentén hozunk létre két új ágat, egyet ahol ezen az élen az indikátorváltozó értéke 0, azaz nem használjuk a megoldásban és egy másikat, ahol 1, azaz használjuk az élet.

Amikor már nincs több kört tartalmazó oszlop, akkor a tört megoldások következnek. Ha egy oszlop minden élének értéke 1, nem kell vizsgálnunk. A többi oszlopot a változók függvényében rangsoroljuk, és kiválasztjuk közülük a vizsgálni kívántakat. A kiválasztott oszlopok éleinek ismét kiszámítjuk a pontszámát és a legjobb pontszámmal rendelkező élen két új ágat hozunk létre 0 és 1 értékkel az előzőhöz hasonlóan. Ha egy él 0 értékű, eltávolítjuk a részfeladat gráfjából, és minden utat, ami használta ezt az élet, büntetünk. Ezen oszlopokon való optimalizálás után (megoldjuk újra az LP-t) új oszlopokat veszünk be a feladatba. Ha egy (i, j) él értéke 1, minden $(i, k) \in A, k \neq j$ és $(l, j) \in A, l \neq i$ élet törlünk a gráfból és büntetünk minden utat, ami ezeket az éleket használja. Ha szükséges, a kapott oszlopokkal újra megoldjuk a feladatot.

4.2. Branch and cut

A branch and cut algoritmusok szintén gyakran alkalmazottak lineáris programok megoldására, így az általunk vizsgált feladatra is. Az első branch and cut algoritmust a VRPTW feladatra Jonathan F. Bard, George Kontoravdis és Gang Yu [1] mutatták be a (3.1.1)-ben ismertetett modellre, egy apró változtatással. Elsődleges célfüggvénynek ugyanis nem az utak összköltségének minimalizálását, hanem azok számának minimalizálását vették, (3.1) képletből így min $\sum_{i=1}^{n} x_{0i}$ lesz. Ugyanakkor ez a feladat könnyen átalakítható az általános célfüggvényre is, amikor az összköltséget szeretnénk minimalizálni, ezt itt nem részletezzük.

Elsőként bemutatunk néhány egyenlőtlenséget a TSP és VRP feladatokra, majd néhány feladatspecifikus megfelelőjét a VRPTW feladatra, végül a definiált egyenlőtlenségek segítségével felírjuk az algoritmust.

4.2.1. Egyenlőtlenségek a TSP és a VRP feladatra

Legyen most $G_0 = (V, E)$ egy irányítatlan, súlyozott gráf, ami csak olyan (i, j) éleket tartalmaz ahol $x_e = x_{ij} + x_{ji} > 0$. Egy $S \subset V$ halmazra legyen $E(S) = \{(i, j) : i, j \in S\}$ az S-beli élek, $\delta(S) = \{(i, j) : i \in S \text{ és } j \notin S\}$ az S-t elhagyó élek halmaza. Az S_1 és S_2 csúcshalmaz között futó élek halmazát (S_1, S_2)-vel jelölüljük. G_0 gráfon egy $e = (i, j)$ élre az él súlya legyen $\omega_e = x_{ij} + x_{ji}$.

Legyen $x(E(S)) = \sum_{e \in E(S)} x_e$ és legyen λ_S alsó becslés az S-hez szükséges járművek minimális számára. Természetesen TSP feladat esetén, ez az érték 1, VRP esetén egy lehetséges választás $\lambda_S = \lceil \sum_{i \in S} q_i / Q \rceil$. Jelöljük $\vec{G_0} = (V, A)$-val a súlyozott irányított gráfot, szintén minden éléhez

tartozik egy $\omega_{ij} = x_{ij}$ súly. G illetve $\overrightarrow{G}$ jelöli a gráfokat a forrás csúcs nélkül.

Részút eliminációs egyenlőtlenségek

A következő egyenlőtlenségek arra szolgálnak, hogy egy S halmazát a csúcsoknak legalább λ_S jármű szolgálja ki. Irányítatlan esetben:

$$\sum_{e \in (S, V \setminus S)} x_e \geq 2\lambda_S. \tag{4.2}$$

Irányított esetben tekintsünk egy $k \geq 3$ csúcsból álló részutat, ezekre súlyozást elvégezve élesebb, de bonyolultabb egyenlőtlenséget kapunk.

$$\sum_{j=1}^{k-1} x_{i_j i_{j+1}} + x_{i_k i_1} + 2\sum_{j=2}^{k-1} x_{i_j i_1} + \sum_{j=3}^{k-1}\sum_{h=2}^{j-1} x_{i_j i_h} \leq k - 1, \tag{4.3}$$

$$\sum_{j=1}^{k-1} x_{i_j i_{j+1}} + x_{i_k i_1} + 2\sum_{j=3}^{k} x_{i_j i_1} + \sum_{j=4}^{k}\sum_{h=3}^{j-1} x_{i_j i_h} \leq k - 1. \tag{4.4}$$

A két egyenlőtlenséget $\overleftarrow{D}_k$ és $\overrightarrow{D}_k$ egyenlőtlenségeknek nevezik. Bővebb magyarázat Fischetti és Toth 1997-es aszimmetrikus TSP-ről szóló cikkében szerepel [6].

Fésű egyenlőtlenségek

Definíció. *Legyen $H, W_1, \ldots, W_k \subset V$. A csúcsok egy halmazát $(H, W_1, \ldots, W_k)$ fésűnek nevezzük, ha teljesülnek rá a következő feltételek:*

$$\begin{aligned} |H \cap W_i| &\geq 1,\ i = 1, \ldots, k, \\ |W_i \setminus H| &\geq 1,\ i = 1, \ldots, k, \\ 2 \leq |W_i| &\leq |V| - 2,\ i = 1, \ldots, k, \\ W_i \cap W_j &= \emptyset,\ i \neq j, \\ k &\geq 3 \text{ és páratlan}. \end{aligned}$$

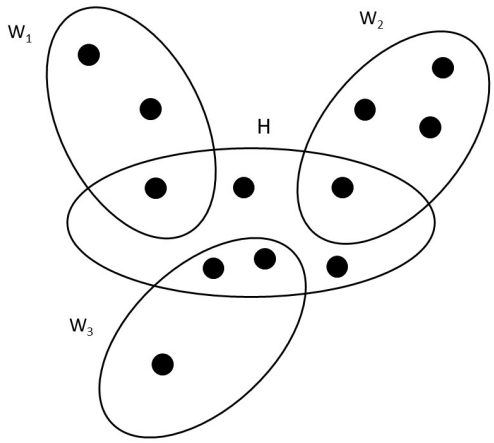
A H halmazt nevezzük nyélnek, a W_i halmazokat a fésű fogainak, az elnevezést és definíciót a 4.2 ábra szemlélteti.

Először a TSP feladatra alkalmazták őket, ahol a következő egyenlőtlenség teljesül:

$$x(E(H)) + \sum_{i=1}^{k} x(E(W_i)) \leq |H| + \sum_{i=1}^{k} |W_i| - \frac{3k+1}{2}. \tag{4.5}$$

Ahhoz, hogy a VRP problémára is felírjuk őket, figyelembe kell vegyük, hogy a forrás melyik halmazban található. Ha a forrás a fésű valamelyik fogában van, (a teljesség megszorítása nélkül feltehető, hogy az elsőben) de nem tartozik a nyélhez, akkor így néz ki az egyenlőtlenség:

$$x(\delta(H)) \geq (k+1) - (x(\delta(W_1)) - 2\lambda_{V \setminus W_1}) - \sum_{i=2}^{k} (x(\delta(W_i)) - 2). \tag{4.6}$$



4.2. ábra. Példa fésűre

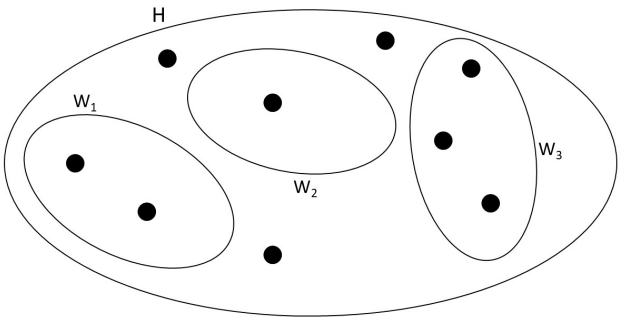
Ha a forrás egyáltalán nem tartozik a fésűhöz, valamint a fogakra teljesül, hogy $\lambda_{W_i \setminus H} + \lambda_{W_i \cap H} > \lambda_{W_i}$, akkor az egyenlőtlenség a következő:

$$x(\delta(H)) \geq (k+1) - \sum_{i=1}^{k} (x(\delta(W_i)) - 2\lambda_{W_i}). \tag{4.7}$$

Doboz egyenlőtlenségek

Definíció. *Legyen $H, W_1, \ldots, W_k \subset V$. A csúcsok egy halmazát $(H, W_1, \ldots, W_k)$ doboznak nevezzük, ha teljesülnek rá a következő feltételek:*

$$\begin{aligned} \sum_{\eta \in W_i} q_\eta \leq Q,\ i = 1, \ldots, k, \\ W_i \subset H,\ i = 1, \ldots, k, \\ W_i \cap W_j = \emptyset,\ i \neq j, \\ k \geq 1. \end{aligned}$$



4.3. ábra. Példa dobozra

A definícióban megköveteljük, hogy a W_i-ben lévő csúcsok beleférjenek egy járműbe, illetve a fésűkkel ellentétben most minden W_i részhalmaza a nyélnek.

Egy dobozra definiáljunk egy ládapakolás (bin packing) feladatot. A tárgyak száma $k + |H \setminus \cup_{i=1}^{k} W_i|$ és a ládák mérete Q. Így minden W_i halmaz egy-egy tárgynak számít, méretük: $\sum_{\eta \in W_i} q_\eta$, a többi tárgy ($\eta$ csúcs) mérete q_η. Vegyük a pakolási feladat optimális megoldását, legyen ez $B_{H|W_1,\ldots,W_k}$, felírhatjuk a következő egyenlőtlenséget:

$$x(\delta(H)) \geq 2B_{H|W_1,\ldots,W_k} - \sum_{i=1}^{k}(x(\delta(W_i)) - 2). \tag{4.8}$$

4.2.2. Egyenlőtlenségek a VRPTW feladatra

Az előző fejezetben bemutatott összes egyenlőtlenség alkalmazható a VRPTW feladatra, hiszen a VRP és a TSP feladatok ennek a speciális esetei. Azonban használva a feladat tulajdonságait, tovább erősíthetünk az egyenlőtlenségeken.

Részút eliminációs egyenlőtlenségek

Ha a depót figyelmen kívül hagyjuk, VRPTW felfogható mint diszjunkt utak uniója. A megengedett utakban körök nem szerepelhetnek, ezeket szeretnénk eliminálni, csökkentve a lehetséges megoldások halmazán. A következő néhány egyenlőtlenség segít ebben.

$$\sum_{e \in E(S)} x_e \leq |S| - \lambda_S. \tag{4.9}$$

Könnyű végiggondolni az egyenlőtlenség igazságát, felhasználva, hogy legalább λ_S út szükséges az S halmazhoz, és hogy p csúcsból álló úthoz $p - 1$ él kell. Figyelembe véve a gráf irányítását, a következő egyenlőtlenséget írhatjuk fel:

$$\sum_{(i,j) \in (S, V \setminus S)} x_{ij} \geq \lambda_S. \tag{4.10}$$

Jelentése, hogy a csúcsok bármely S részhalmazából legalább annyi jármű ki kell jöjjön, mint az S kiszolgálásához szükséges járművek alsó becslése. A két egyenlőtlenségre teljesül a következő lemma, amelyet most nem bizonyítunk.

Lemma. *A $\vec{G}$ irányított gráf felett definiált (4.10) egyenlőtlenség sérül, pontosan akkor, ha a G irányítatlan gráfra definiált (4.9) sérül.*

Fésű egyenlőtlenségek

A (4.2.1) fejezetben definiált fésűre egy VRPTW-specifikus egyenlőtlenséget is felírhatunk:

Tétel. *Minden G-beli fésűre érvényes a következő egyenlőtlenség (VRPTW):*

$$\sum_{e \in E(H)} x_e + \sum_{i=1}^{k} \sum_{e \in E(W_i)} x_e \leq |H| + \sum_{i=1}^{k} |W_i| - \frac{1}{2} \sum_{i=1}^{k} (\lambda_{W_i} + \lambda_{H \cap W_i} + \lambda_{W_i \setminus (H \cap W_i)}). \tag{4.11}$$

Bizonyítás. A (3.2) és (3.3) egyenlőtlenségek megadhatók (4.12) formában:

$$\sum_{e\in\delta(\{v\})} x_e = 2 \;\forall\, v \in V \setminus \{0\}. \tag{4.12}$$

$\frac{1}{2}$-del szorozva és összegezve minden $v \in H$-ra, egy a (4.9)-hez hasonló egyenlőtlenséget kapunk:

$$\sum_{e\in E(H)} x_e \le |H| - \frac{1}{2}\sum_{e\in\delta(H)} x_e. \tag{4.13}$$

Ezt az összeget két komponens összegeként is felírhatjuk. Az első komponens azon H-t elhagyó élek értékeit összegzi, amik a fogakba mennek, a másik komponens azokat, amik a fogakon kívüli más csúcsokba.

$$\sum_{e\in\delta(H)} x_e = \sum_{i=1}^{k}\sum_{e\in\delta(H)\cap E(W_i)} x_e + \sum_{i=1}^{k}\sum_{e\in\delta(H)\setminus\bigcup_{i=1}^{k} E(W_i)} x_e. \tag{4.14}$$

Írjuk át a (4.13) egyenlőtlenséget, felhasználva csak a fogakba menő H-t elhagyó éleket, így átrendezve szintén érvényes gyengébb becslést kapunk:

$$\sum_{e\in E(H)} x_e + \sum_{i=1}^{k}\sum_{e\in\delta(H)\cap E(W_i)} x_e \le |H|. \tag{4.15}$$

Most tekintsük a W_i, $H \cap W_i$ és $W_i \setminus H$-ra, a (4.9)-ben leírt részút eliminációs egyenlőtlenséget:

$$\sum_{e\in E(W_i)} x_e \le |W_i| - \lambda_{W_i} i = 1, \ldots, k, \tag{4.16}$$

$$\sum_{e\in E(H\cap W_i)} x_e \le |H \cap W_i| - \lambda_{H\cap W_i} i = 1, \ldots, k, \tag{4.17}$$

$$\sum_{e\in E(W_i\setminus H)} x_e \le |W_i \setminus H| - \lambda_{W_i\setminus H} i = 1, \ldots, k. \tag{4.18}$$

Szummázzuk minden i fogra, és szorozzuk $\frac{1}{2}$-del, a kapott egyenlőtlenségeket adjuk össze (4.15)-tel. Vegyük észre, hogy

$$E(H \cap W_i) \cup E(W_i \setminus H) \cup (\delta(H) \cap E(W_i)) = E(W_i) \text{ és } |W_i| = |H \cap W_i| + |W_i \setminus H|.$$

Így már látszik, hogy az összeadott egyenlőtlenségek a tétel állítását adják. □

Ha mindegyik alsó becslés 1, akkor az egyenlőtlenség leegyszerűsödik:

$$\sum_{e\in E(H)} x_e + \sum_{i=1}^{k}\sum_{e\in E(W_i)} x_e \le |H| + \sum_{i=1}^{k} |W_i| - \frac{3k}{2}. \tag{4.19}$$

Inkompatibilis pár egyenlőtlenségek

Definíció. *Legyen i és j a G két csúcsa, definiáljuk a következőket:*

- *Ha egy megengedett úton j közvetlen utódja lehet i-nek, akkor ezt $i \rightarrow j$-vel jelöljük.*
- *Ha nincs olyan megengedett út, amin j utódja i-nek, akkor ezt $i \nrightarrow j$-vel jelöljük.*
- *$i \leftrightarrow j$-vel jelöljük, ha mindkét $i \rightarrow j$, $j \rightarrow i$ megengedett útvonal létezik*
- *$i \nleftrightarrow j$-vel jelöljük, ha egyik irányban sem létezik megengedett útvonal, azaz $i \nrightarrow j$ és $j \nrightarrow i$.*

Ha $i \nleftrightarrow j$, akkor i-t és j-t inkompatibilis párnak nevezzük.

Feltesszük, hogy minden G-beli i, j csúcspár utazási idejére teljesül a háromszög egyenlőtlenség, így, ha $i \nleftrightarrow j$, akkor mindenképpen két külön komponensbe kell essenek. Legyen $i \nleftrightarrow j$ egy inkompatibilis pár és P egy út G-ben, $P = \{i, k_1, \ldots, k_{|P|-2}, j\}$. Erre az útra felírhatunk egy egyenlőtlenséget, jelentése, hogy ha van egy nem megengedett utunk (hiszen csak ilyen i, j utak vannak) akkor a megoldásban ennek az útnak nem használhatjuk az összes élét. Mivel párhuzamos éleket sem szeretnénk, és ezeket a megoldásban más egyenlőtlenségekkel kizárjuk, így azzal, hogy az egyenlőtlenségben mindkét irányú éleket nézzük, rögtön kizárunk két nem megengedett utat is.

$$x_{i,k_1} + x_{k_1,i} + \cdots + x_{k_{|P|-2},j} + x_{j,k_{|P|-2}} \leq |P| - 2. \tag{4.20}$$

(4.20)-at tovább erősíthetjük, ha a jobb oldalát lecseréljük P kiszolgálásához szükséges járművek alsó becslésére, ha az legalább kettő.

$$x_{i,k_1} + x_{k_1,i} + \cdots + x_{k_{|P|-2},j} + x_{j,k_{|P|-2}} \leq |P| - \max\{2, \lambda_P\}. \tag{4.21}$$

Az egyenlőtlenség (4.9)-hez hasonlóan meggondolható.

Inkompatibilis út egyenlőtlenségek

Az inkompatibilis pár egyenlőtlenségekhez hasonlóan figyelembe vesszük az irányított gráfon i-ből j-be és j-ből i-be menő éleket. Különbség, hogy kifejezetten azokat az eseteket nézzük, amikor $i \nrightarrow j$, $j \rightarrow i$. Mint korábban, itt is fennáll, hogy a háromszög-egyenlőtlenség miatt, ami érvényes az utazási időre, nem lehet olyan út a megoldásban, ami i-ből j-be megy. Ezeket az utakat zárja ki a következő két egyenlőtlenség, ami az előzőekhez hasonlóan gondolható át.

$$x_{i,k_1} + x_{k_1,k_2} + \cdots + x_{k_{|P|-2},j} \leq |P| - 2, \tag{4.22}$$

$$x_{i,k_1} + x_{k_1,k_2} + \cdots + x_{k_{|P|-2},j} \leq |P| - \max\{2, \lambda_P\}. \tag{4.23}$$

4.2.3. A branch and cut algoritmus

A feladat leírásában és a (4.2.2) fejezetben definiált egyenlőtlenségeket használjuk fel. Korábban a (2.3) fejezetben leírtuk, hogy hogyan működik a branch and cut algoritmus általánosan, most speciálisan bemutatjuk erre a feladatra. Legyenek x, t és y vektorok, amik a folyam, idő és termék változókat reprezentálják.

Legyen P a megengedett megoldások poliédere, és legyen F az összes, a poliéder lapjait meghatározó egyenlőtlenség halmaza (ezek az egyenlőtlenségek a hasznosak az optimalizáció szempontjából):

$$F = \{(a_1, a_2, a_3, a_0) \in \mathbb{R}^{(n+1)\times(n+1)} : a_1x + a_2t + a_3y \leq a_0, (x, t, y) \in P\}. \tag{4.24}$$

A feladatot így felírhatjuk a következőképpen is:

$$\min\{fx : (x, t, y) \in \text{conv}(P)\}, \tag{4.25}$$

$\text{conv}(P)$ a P poliéder konvex burka. Kombinálva (4.24) és (4.25) az LP(F)-hez tartozó lineáris program felírható:

$$z_{LP}(F) = minfx$$

alakban, feltéve, hogy

$$a_1x + a_2t + a_3y \leq a_0, \forall(a_1, a_2, a_3, a_0) \in F$$
$$(x, t, y) \in \overline{P} \coloneqq P \cup \{0 \leq x \leq 1\}.$$

Az algoritmus a következő:

1. Legyen $z^* = z$, z input. Legyen Θ azon csúcsok halmaza, amiket még nem vizsgáltunk meg, kezdetben tartalmazza a keresőfa gyökerét.
2. Ha $\Theta = \emptyset$ STOP. Különben kiválasztunk egy csúcsot $\omega \in \Theta$, amit vizsgálunk és folytatjuk a 3. lépéssel.
3. Megoldjuk a megfelelő LP(F) relaxáltját: $z^k_{LP}(F) = \min\{fx : (x, t, y) \in \overline{P}^k\}$. Legyen (x^k, t^k, y^k) az optimális megoldás a k-adik iterációban.
4. Ha LP(F) nem megengedett, akkor legyen $\Theta = \Theta \setminus \{\omega\}$ és ugorjunk a 2. lépésre.
5. Ha $z^k_{LP}(F) \geq z^*$, akkor legyen $\Theta = \Theta \setminus \{\omega\}$ és ugorjunk a 2. lépésre.
6. Ha LP(F) optimális megoldása egész, akkor legyen $z^* = \min\{z^*, z^k_{LP(F)}\}$, $\Theta = \Theta \setminus \{\omega\}$ és ugorjunk a 2. lépésre.
7. Oldjuk meg a szeparációs problémát. Legyen $F^k \in F$ a sértett egyenlőtlenségek halmaza. $\overline{P}^k = \overline{P}^k \cap \{(x, t, y) \in \mathbb{R}^{(n+1)\times(n+1)} : a_1x + a_2t + a_3y \leq a_0,$ ahol $(a_1, a_2, a_3, a_0) \in F^k\}$. Legyen a sértő egyenlőtlenségek száma κ.
8. Ha $\kappa > 0$, akkor ugorjunk a 3. lépésre.
9. Új $\hat{\omega}$-t hozunk létre szétválasztással (branching), ezt hozzáadjuk Θ-hoz, $k = k + 1$ és ugorjunk a 3. lépésre.

Az 1. lépésben inicializálunk. A 2. lépésben leírjuk a megállási feltételeket, és kiválasztunk egy csúcsot amit feldolgozunk. A 3. lépésben megoldjuk a lineáris programot. A 4-6. lépésekben szükséges eseteket nézünk végig, ha az LP megoldás nem megengedett, nem optimális, vagy egészértékű. A 7. lépésben megoldjuk a szeparációs problémát, ami egy $(x, t, y) \in \mathbb{R}^{(n+1)\times(n+1)}$ megoldásra megmutatja, hogy az összes egyenlőtlenséget teljesíti F-ben, vagy talál legalább egyet amit nem. A sértő egyenlőtlenségeket hozzáadja a feladathoz és az új LP relaxáltat optimalizáljuk. Ha az összes egyenlőtlenség teljesül, új csúcsot generálunk a kilencedik lépésben.

Szeparációs heurisztikák

A 7. lépésben a szeparációs feladat megoldása nagyon időigényes lehet, sokszor heurisztikákat alkalmaznak a gyorsabb keresésre. Fontos az egyenlőtlenségfajták keresésének sorrendje is, néhányat most bemutatok ezek közül.

Először a (4.2.2)-ben definiált *részút eliminációs* egyenlőtlenségeket keressük, pontosabban egy olyan S részhalmazát a csúcsoknak, mely sérti (4.9)-et. Az egyik ötlet ilyen sértő halmazok keresésére, ha megkeressük a gráf minimális vágását (S olyan részhalmaza a csúcsoknak, hogy a gráfban S és $V \setminus S$ között megy a legkevesebb él). Ha TSP feladatot nézünk, akkor ez egy egzakt megoldás, ha a minimális vágás kisebb, mint 2, akkor megtaláltuk a sértő halmazt. A TSP-vel ellentétben a VRP feladatra ez egy heurisztika, előfordulhat ugyanis, hogy a kapott minimális vágás nem sért, de a gráf tartalmaz sértő halmazt.

Egy másik heurisztika ugyanerre a feladatra, a gráf-csökkentés (shrinking). A csökkentést úgy érjük el, hogy összevonunk csúcsokat, a köztük lévő éleket törölve, ezek a csúcsok lesznek a vizsgálandó részhalmazaink. A kapott új csúcsokat súlyozzuk, súlyuk a bennük lévő élek összsúlya, ezért ha megnézzük az érték (4.9) bal oldala. Az ötlet az, hogy minél nagyobb egy súly, annál nagyobb az esélye, hogy a csúcs (részhalmaz) sért. Többször ismételjük az eljárást, leghatékonyabban a randomizált csökkentést használjuk, hogy több fajta sértő halmazt is felfedezhessünk. A feladat komplexitása $O(|V||E|log|E|)$, mert a csökkentést legfeljebb $|V| - 1$-szer kell végrehajtanunk, két csúcs összevonása legfeljebb $O(|E|)$ élet érint, és a gráf éleinek karbantartása $O(\log|E|)$ lépés.

4.4. ábra. Gráf-csökkentés

A következő vizsgált egyenlőtlenségek a fésű egyenlőtlenségek. Fésűk keresésénél először nyeleket keresünk, majd hozzájuk meghatározzuk a lehetséges fogakat. Hozzunk létre G-ből egy G' részgráfot, ami nem tartalmazza az ε-nál kisebb és az $1 - \varepsilon$-nál nagyobb ω_e súlyú éleket. A kapott gráf 2-összefüggő komponenseit fogjuk nézni. Egy 2-összefüggő komponens legalább

3 ponttal nyél lehet. Kettő, közös elvágó csúccsal rendelkező 2-összefüggő komponens uniója szintén egy lehetséges nyél. Legyen H^* ezen nyelek halmaza.

A fogakat kétféle módon is meghatározhatjuk. Az első, ha keresünk egy G-beli legalább $1-\varepsilon$ súlyú $(H, V \setminus H)$ vágást, $H \in H^*$, a hozzá tartozó csúcsok halmaza egy lehetséges fog. A másik, ha veszünk egy $v \in H$ csúcsot és készítünk egy fogat úgy, hogy az tartalmazza v-t és még egy nem H-beli csúcsot, vagy keresünk egy 2-összefüggő komponenst G'-ben ami H-t a v csúcsban metszi.

Töröljük a metsző fogakat, és ha a kapott fogak H-val olyan fésűt alkotnak, ami sérti a (4.7), vagy a (4.11) egyenlőtlenségeket, akkor az egyenlőtlenséget hozzáadjuk a modellhez. A feladat $O(|E|)$ komplexitású, függ a G'-ben lévő 2-összefüggő komponensek számától, ami egy nagy gráfra nézve általában konstans.

Inkompatibilis pár egyenlőtlenségek keresésénél is alkalmazunk egy segédgráfot $\overline{G'}$-t, amit úgy kapunk, hogy kicseréljük az összes $e \in G$ és ω_e súlyát $\omega_{\max} - \omega$-ra, ahol $\omega_{\max}$ a maximális súly G-ben. Nézzük az összes $i \nleftrightarrow j$ párt, ha G-ben ők egy komponensbe tartoznak, akkor az őket összekötő út sérthet, így vegyük a legrövidebb, i és j közti utat $\overline{G'}$-ben. Ha az út sérti a (4.21)-et, akkor hozzávesszük a feladathoz az egyenlőtlenséget. Ha a feladat megoldását úgy kezdjük, hogy minden csúcs között megkeressük a legrövidebb utakat, akkor a feladat komplexitása $O(n^3)$, megoldható a Floyd-Warshall algoritmussal.

Végül tekintsük az inkompatibilis út egyenlőtlenségeket. Legyen $\vec{G'}$ súlyozott, irányított gráf, mely nem tartalmazza a depót és minden $\vec{G}$-beli ω_e súlyú él, $1-\omega_e$ súlyú benne. Az inkompatibilis párokhoz hasonlóan megkeressük $\vec{G'}$-n a legrövidebb i és j közti utat, melyre $i \nrightarrow j$ és $j \rightarrow i$. Komplexitása megegyezik az előzőével.

5. fejezet

Heurisztikák

Legyen Ω a megengedett utak halmaza. Célunk most is az, hogy megkeressük azon utak $s^* \subset \Omega$ halmazát, melyek megengedettek a VRPTW feladatra, és melynek költsége minimális: $c(s^*) \leq c(s), \forall s \subset \Omega$. Heurisztikák esetében azonban megelégszünk olyan s halmazzal is, aminek költsége talán egy picivel több, mint $c(s^*)$, cserébe, hogy ezen utakat gyorsan megtaláljuk.

5.1. Szomszédsági keresés

Legyen $N(s)$, s szomszédainak halmaza, amit felhasználunk a keresés során. Meghatározása algoritmusonként változik, általában s-hez néhány szempont szerint "közeli" megoldásokat keresünk, amiket kisebb javításokkal érünk el. Ennek a halmaznak mérete nagyban befolyásolja az alkalmazandó heurisztikát, ugyanis míg kisméretű szomszédsági halmaz esetén az algoritmusok gyorsan lefutnak, addig a nagyméretűek pontosabb eredményt szolgáltatnak, de a futtatásukhoz sokkal több időre van szükség. Ebben a fejezetben Toth és Vigo könyvét vesszük alapul [10].

Definíció. *Egy s megoldást lokálisan optimálisnak nevezünk, ha $c(s) \leq c(s'),\ \forall s' \in N(s)$-re.*

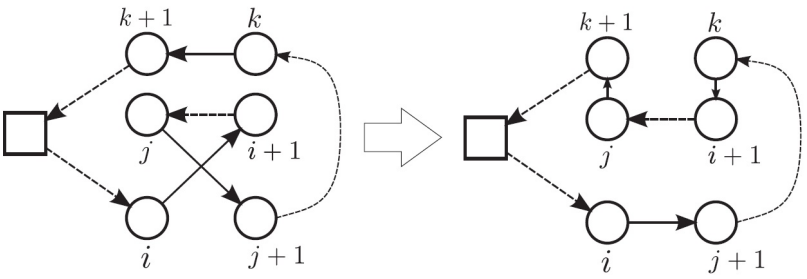
A következő algoritmus bemutatja, hogy a legtöbb szomszédsági keresés hogyan működik. Mindig megkeresi az aktuális $N(s)$ halmaz legjobb megoldását (s'), ha ez a megoldás jobb, mint s, akkor s' felülírja s-t. Ismételjük addig amíg megfelelően jó megoldást kapunk.

1. Legyen $s \in \Omega$ kezdeti megoldás.
2. Amíg nem jutottunk lokális optimumba, legyen $s' \in argmin_{s" \in N(s)}\{c(s")\}$. Ha $c(s') < c(s)$, legyen $s := s'$, különben STOP.
3. Eredmény: a talált lokális optimum s.

A továbbiakban két szomszédsági definíciót mutatunk be, és egy algoritmust a szomszédok bejárására. Mint korábban említettük szomszédokat sokféleképpen létre lehet hozni, a megoldásokon mindig kicsit javítva, változtatva.

5.1.1. Or-opt

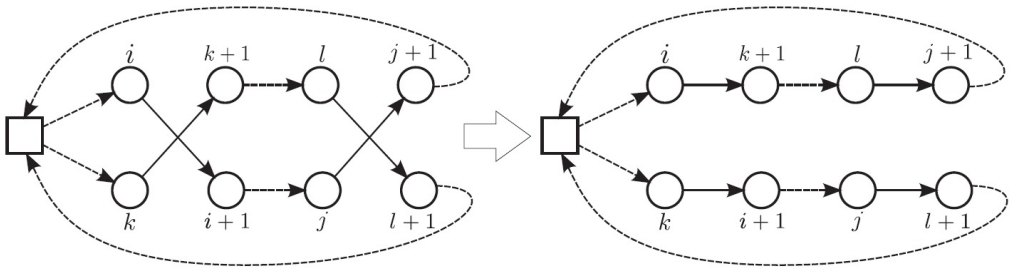
Az or-opt egy olyan szomszédsági algoritmus, ami egyszerre mindig egy utat néz. Az adott út egy részútját áthelyezi úgy, hogy ezzel optimálisabb megoldást kapjunk. Ezt szemlélteti az ábra. Nézzük az $(i+1, j)$ részutat, ha áthelyezzük $(k, k+1)$ közé, akkor javítunk az úton, legtöbbször ezeket a keresztező útvonalakat így lehet javítani. Futási ideje lényegesen csökken, ha csak olyan részutakat helyezünk át, amikben egy konstanssal korlátozott számú csúcs van.



5.1. ábra. Or-opt szomszédság [10]

5.1.2. Cross-exchange

Az előzővel ellentétben, most két részutat választunk ki két különböző útból, és kicseréljük őket. A példában a $(k+1, l)$ és $(i+1, j)$ részutakat cseréljük ki. Az algoritmust lehet alkalmazni egyetlen útra is.



5.2. ábra. Cross-exchange szomszédság [10]

5.1.3. Tabu keresés

A tabu keresés (tabu search) egy sokat alkalmazott heurisztika a megoldások bejárására. Azért, hogy megakadályozzuk az algoritmust korábbi megoldások ismételt megkeresésében, ciklizálást tiltó szabályokat vezetünk be. Korábban vizsgált csúcsokat tabu címkével látunk el, ezt a címkét idővel, vagy bizonyos feltételek teljesülésével levesszük róla. Emellett alkalmazunk olyan eljárásokat, amik lehetővé teszik a változatosabb útvonalak megtalálását.

Most Cordeau, Jean-François, Gilbert Laporte, és Anne Mercier [3] egy 2001-es cikkében adott tabu kereső algoritmusát nézzük át ebben a bekezdésben. Megoldásukban a szokásos feltételek

mellett korlátozzák az egyes járművek utazási idejét is, a k járműre ez a korlát D_k, emellett olyan úthalmazokat keresünk, amikre $|K| = m$ fix.

Legyen $s \subset \Omega$ egy m útból álló halmaz úgy, hogy minden csúcs pontosan egy úthoz tartozzon, és minden út a forrás csúcsban, v_0-ban kezdődjön és végződjön. s a modell néhány feltételét teljesíti, de sértheti az időablakokat, a járműkapacitás és időtartam korlátokat. Legyenek $q(s)$, $d(s)$, $w(s)$ az s megoldás által sértett kapacitások, időkorlátok és várakozási idők összessége. A megoldások értékelésére bevezetünk egy függvényt, $f(s) \coloneqq c(s) + \alpha q(s) + \beta d(s) + \gamma w(s)$, ahol α, β és γ pozitív súlyok. Ezeket az algoritmus futása közben állítjuk be, vagy változtatjuk meg.

Szomszédsági keresésnél fontos, hogyan határozzuk meg $N(s)$ halmazt, s szomszédait. Ehhez legyen $B(s) = \{(i,k) : \text{ az } i \text{ csúcsot a } k \text{ jármű szolgálja ki}\}$. $N(s)$ az olyan s' úthalmazokat tartalmazza, ahol i-t nem a k jármű, hanem egy másik k' látogatja meg, $B(s') = \{(i,k') : \text{ az } i \text{ csúcsot a } k' \text{ jármű szolgálja ki}\}$. Ilyen utakat hozunk létre, ha i-t kitöröljük a k útról és beszúrjuk k'-be. Törlés után k az i volt szülője és utódja közti élet használja, a beillesztés k'-be pedig úgy történik, hogy közben figyelembe vesszük $f(s)$ minimalizálását. Miután az i csúcsot töröltük k-ról, a következő θ iterációban nem tehetjük vissza, az (i,k) párra teszünk egy tabu címkét.

További eltérő megoldások kereséséhez egy $\overline{s} \in N(s)$ úthalmazt büntetünk, azaz $f(\overline{s})$-hez adunk egy $p(\overline{s})$ büntetést, ha $f(\overline{s}) \geq f(s)$, egyéb esetben $p(\overline{s}) = 0$. Ezzel az algoritmus eddig kevésbé felfedezett területekre vezet a megoldások terén.

Az algoritmus

Egy kezdeti megoldást generálunk a következőképpen. Először a csúcsokat sorrendbe rendezzük egy tetszőleges rendezés szerint. (A gyakorlatban a csúcsoknak adott az euklideszi koordinátája, olyankor aszerint rendezzük sorba őket, hogy mekkora szöget zárnak be a forrással.) Három lépéssel generáljuk az utakat:

1. Válasszunk véletlenszerűen egy $j \in \{1 \dots n\}$ csúcsot.
2. Legyen $k \coloneqq 1$.
3. Nézzük a csúcsokat a következő sorrendben: $j, j+1, \dots, n, 1, \dots, j-1$, minden, i csúcsra végrehajtjuk a következő két lépést:
   - Ha az i csúcs beszúrása a k útba sértené az kapacitás-, vagy az időkorlát feltételeket, akkor $k \coloneqq \min\{k+1, m\}$.
   - Szúrjuk be i-t a k útba figyelve, hogy a jármű utazási idejét minimalizáljuk.

A beszúrást csak olyan két j_1, j_2 csúcs között végezhetjük el, ahol $a_{j_1} \leq a_i \leq a_{j_2}$, ezzel segítjük az időablakok betartását. Az eljárás végén elértük, hogy az első $m-1$ út biztosan teljesíti a kapacitás- és időkorlátokat.

A kapott megoldásból elindítjuk a tabu kereső algoritmust, ami minden iterációban a legjobb nem tabu csúcsot találja meg $N(s)$-ben. Minden iteráció után módosítjuk az $f(s)$ függvény súlyait, $1+\delta$-val, ahol $\delta > 0$ úgy, hogy ha a megoldás betartja a kapacitás korlátokat, osztjuk

α-t $1 + \delta$-val, ha sérti őket, akkor szorozzuk $1 + \delta$-val. Ugyanígy járunk el a β, γ súlyokkal is. Ismételjük az eljárást η alkalommal, és a legjobb s^* megengedett megoldás javításával térünk vissza. A javítást Gendreau, Michel, Alain Hertz, és Gilbert Laporte[7] TSP feladatra írt utóoptimalizáló heurisztikájával végezzük. A heurisztikát egyenként minden útra alkalmazzuk. Lényege, hogy egy úton minden csúcsot egyesével kiszedünk és megpróbáljuk azon az úton másik helyre beszúrni. A műveletet *Unstringing and Stringing* műveletnek nevezik.

1. Legyen $\alpha := 1$, $\beta := 1$ és $\gamma := 1$. Ha s megengedett, akkor $s^* := s$.
2. $\kappa = 1, \ldots, \eta$-re:
   - Kiválasztunk egy $\overline{s} \in N(s)$ szomszédot ami minimalizálja az $f(\overline{s}) + p(\overline{s})$ függvényt és nincs tabu címkéje.
   - Ha $\overline{s}$ megengedett megoldás és $c(\overline{s}) < c(s^*)$, legyen $s^* := \overline{s}$ és $c(s^*) := c(\overline{s})$.
   - Számítsuk ki $q(\overline{s})$, $d(\overline{s})$ és $w(\overline{s})$ értékeket és frissítsük α, β és γ súlyokat aszerint.
   - $s := \overline{s}$.
3. Használjunk egy utóoptimalizáló heurisztikát s^*-ra.

5.2. Evolúciós algoritmusok

Az evolúciós algoritmusok [10] a populáció alapú algoritmusok egy alfaja. Populáció alapú algoritmusoknál a megoldások halmaza adott, ezt hívjuk a populációnak, ami minden iterációban fejlődik. Az iterációk számában és az algoritmusok beállításaiban nagy szerepe van a tapasztalatnak.

Az evolúciós algoritmusok a jelenlegi megoldások kombinációival generálnak utódokat. Egy fitnesz függvény segítségével ezekből kiválogatja a jobbakat, és velük írja felül a populációt. Két fajtája az ilyen algoritmusoknak a genetikus és a memetikus algoritmusok.

Ezek az algoritmusok generációnként 3 tevékenységet hajtanak végre a populáción: *kereszteződés* (crossover), *mutáció* (mutation) és *kiválasztódás*, ez utóbbi a megoldások halmazának frissítésével (updatepool) történik. A kapott megoldásokat *lokális keresés*sel (local search) javítjuk. Változatossá az teszi őket, hogy ezen tevékenységeket hogyan végzik, erre rengeteg ötlet van, az egészen egyszerűtől a bonyolultabbig.

A mutáció lépését nem mindegyik algoritmus tartalmazza, sőt egy klasszikus genetikus algoritmus a lokális keresés javítást sem használja. A kereszteződés az egyik legfontosabb része ezeknek az algoritmusoknak. Rengeteg megvalósítása létezik, ilyen például az OX crossover, az EAX crossover és a multiparent recombination, lásd [10].

1. Létrehozzuk P-t, a kezdeti megoldások halmazát.

2. Amíg egy kilépő feltétel nem teljesül:

$S :=$crossover(P),

$S' :=$ mutation(S),

$S'' :=$localsearch(S'),

P :=updatepool(P, S'').

3. Eredmény: a keresés alatt talált legjobb megoldás.

OX crossover

Az OX crossover egy permutáció alapú reprezentáció. Véletlenszerűen választunk két számot i-t és j-t, segítségükkel létrehozzuk az utódot úgy, hogy az első szülő i-től j-ig terjedő részét átmásoljuk az utód ugyanezen helyére. A fennmaradó helyeket elkezdjük feltölteni a második szülő $j+1$ helyétől kezdve, eddig még nem használt értékekkel ciklikusan. A példa jól szemlélteti a leírtakat $i = 2$ és $j = 6$ értékekkel:

	1	2	3	4	5	6	7	8	9
Szülő 1	1	4	2	3	8	6	7	5	9
Szülő 2	2	3	6	1	5	8	4	9	7
Utód	5	4	2	3	8	6	9	7	1

Alkalmazásához szükség van egy módszerre, ami képes áttérni egy megoldásról (útról) egy permutációra, majd a permutációról vissza az útra. Előbbire egy megoldás, ha vesszük egy rendezését az s úthalmaznak, és sorrendben leírjuk az utak csúcsait. A visszaváltáshoz alkalmazzuk a következő daraboló (split) eljárást, ami feldarabolja a permutációt különböző részekre, melyek mind egy-egy utat alkotnak. A legoptimálisabb darabolásokat vesszük, ezeket egy segédgráfon végzett legrövidebb útkereső algoritmussal polinom időben meg tudjuk határozni.

Olyan daraboló eljárás is létezik, ami használ nem megengedett utakat is, a segédgráfban büntetve azon éleket amiket használnak. Ezen utakat lokális keresésekkel nagy eséllyel javíthatjuk. A megengedett és a nem megengedett utakat külön tároljuk a megoldások halmazában.

6. fejezet

Összefoglalás

A dolgozatom célja az volt, hogy megismertessem a járműirányítási feladatot, annak speciális esetét az időablakos járműirányítási feladatot, és érdekes ötleteket és algoritmusokat mutassak a megoldására. Rengeteg más algoritmus, ötlet és modellezési technika van rá, ezeket most a dolgozat kereteiben nem tudtam érinteni.

Míg a TSP feladatra több egyszerű approximációs algoritmus is ismert (például Christofides algoritmusa), addig a feladat komplexsége miatt erre a problémára nagyon keveset találni. Ami mégis létezik, az is sokkal összetettebb, bonyolultabb eszközöket és modelleket kíván.

Az approximációs algoritmusokkal ellentétben, a heurisztikák teret hódítottak a témában. A két csoportja (populáció alapú és szomszédsági), a gyakorlatban is a legtöbbet használt technikákhoz tartozik. Az ötletek javításával, a különböző súlyok, vagy iterációszámok beállításával az optimumhoz kellően közeli megoldásokat tudunk kapni, viszonylag rövid idő alatt. Utóbbi a feladatban nagyon fontos szempont.

A probléma kezelése az elmúlt évtizedekben rengeteget fejlődött. A való életbeli jelentősége és széleskörű problémái miatt a kutatókat továbbra is foglalkoztatja.

Irodalomjegyzék

[1] Bard, J. F., Kontoravdis, G., & Yu, G. (2002). A branch-and-cut procedure for the vehicle routing problem with time windows. Transportation Science, 36(2), 250-269.

[2] Bertsimas, Dimitris, and John N. Tsitsiklis. Introduction to linear optimization. Vol. 6. Belmont, MA: Athena Scientific, 1997.

[3] Cordeau, J. F., Laporte, G., & Mercier, A. (2001). A unified tabu search heuristic for vehicle routing problems with time windows. Journal of the Operational research society, 52(8), 928-936.

[4] Dantzig, G. B., & Ramser, J. H. (1959). The truck dispatching problem. Management science, 6(1), 80-91.

[5] 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.

[6] Fischetti, M., & Toth, P. (1997). A polyhedral approach to the asymmetric traveling salesman problem. Management Science, 43(11), 1520-1536.

[7] Gendreau, M., Hertz, A., & Laporte, G. (1992). New insertion and postoptimization procedures for the traveling salesman problem. Operations Research, 40(6), 1086-1094.

[8] Kovács, L., Agárdi, A., & Bányai, T. (2020). Fitness Landscape Analysis and Edge Weighting-Based Optimization of Vehicle Routing Problems. Processes, 8(11), 1363.

[9] Lübbecke, M. E. (2010). Column generation. Wiley encyclopedia of operations research and management science. Wiley, New York, 1-14.

[10] Toth, Paolo, and Daniele Vigo, eds. Vehicle routing: problems, methods, and applications. Society for Industrial and Applied Mathematics, 2014.