Eötvös Loránd Tudományegyetem

Természettudományi Kar

---

Gál Boglárka

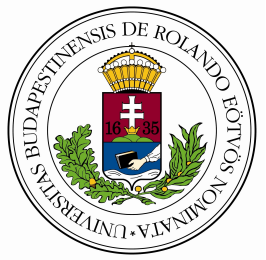
# Optimális szállítás több járművel (VRP)

BSc Szakdolgozat

Témavezető:

Jüttner Alpár

Operációkutatási Tanszék




Budapest, 2018

# Tartalomjegyzék

# 1. Előkészületek

## 1.1. Bevezetés

A gyakorlatban sokszor találkozhatunk szállítási problémákkal. Akár raktárakból az élelmiszerboltokba szeretnénk kivinni az árukat, akár benzinkutakat kell ellátni üzemanyaggal, akár utasokat kell a megfelelő helyekre eljuttatnunk, a cél ugyanaz: kiszolgálni a fogyasztók igényeit a lehető legkevesebb kiadással.

A Vehicle Routing Problems (azaz VRP) azon problémák összefoglaló neve, melyek azzal foglalkoznak, hogy hogyan szállítsuk ki a javakat a raktáraktól a felhasználókig. Ezek a feladatok rendkívül sokfélék lehetnek, ugyanis kitűzésük közvetlenül a valós életből származik. Mivel a probléma NP-teljes, ezért nem ismert a megoldására polinomiális futásidejű algoritmus. Így különböző eljárásokat, heurisztikákat fejlesztettek ki azért, hogy az optimális megoldást viszonylag rövid idő alatt elég jól meg lehessen közelíteni. A dolgozatom fő célja, hogy ismertessem a VRP alapvető matematikai modellezéseit, valamint eljárásokat, heurisztikákat mutassak be, melyek alkalmasak a VRP optimális megoldásának közelítésére. Mindezt természetesen a teljesség igénye nélkül kellett tennem, hiszen nem csak a probléma sokszínűsége, hanem a megoldására adott eljárások változatossága is egy rendkívül tág és sokrétű témakört eredményez. Ezért a dolgozat terjedelmére vonatkozó korlátok miatt a problémakör csak egy szeletére áll módomban kitérni.

A továbbiakban Paolo Toth és Daniele Vigo The Vehicle Routing Problem című könyvének [1] koncepcióját követem, melyet igyekeztem minél több magyarázattal, és további forrásokból származó információkkal kiegészíteni. A dolgozatban többször hivatkozom az utazó ügynök problémára, illetve ennek optimumának közelítésére adott heurisztikákra. Ezen eljárások részleteire a dolgozat írása során nem térek ki, ezeket ismertnek tekintettem. Ezen kívül néhány esetet olyan problémákra fogok visszavezetni, melyek nem kapcsolódnak közvetlenül az egyetemi tananyaghoz. Ilyenek például a Bin Packing Problem, vagy a Generalized Assignment Problem, melyeket szintén a dolgozat korlátozott terjedelme miatt nem részletezek.

## 1.2. Alapfogalmak

Ahhoz, hogy megértsük a problémát, és matematikailag modellezni tudjuk, szükségünk lesz az alapfogalmak tisztázására, a feladat precízebb megfogalmazására, és a felmerülő különböző esetek számbavételére

Először nézzük meg, hogy pontosan mi is a feladatunk. Ki szeretnénk szállítani a javakat egy adott *időintervallumon* belül *fogyasztók* egy halmazának úgy, hogy *járművek* egy halmazát használjuk fel, melyek akár különbözőek is lehetnek. A járművek egy vagy több *depóban* helyezkednek el kezdetben, és *sofőrök* egy halmaza vezeti őket. A mozgásukat egy *úthálózaton* hajtják végre. Tehát a feladatunk lényegében a következő. Meg kell határoznunk az egyes járművek által bejárt utakat, vagyis az utaknak egy halmazát úgy, hogy minden út a megadott saját depójából indul, és valamely megadott depóba érkezik, miközben teljesen kielégíti az általa meglátogatott fogyasztók igényét, és megfelel minden kapacitásra és menetidőre vonatkozó feltételnek. Mindezt úgy szeretnénk tenni, hogy az összesített szállítási költség minimális legyen.

Az általunk vizsgált úthálózatot egy gráffal reprezentáljuk. A gráf élei felelnek meg az utaknak, ezek lehetnek irányítottak, vagy irányítatlanok. A gráf pontjai pedig az útkereszteződéseket, a fogyasztókat és a depókat reprezentálják. Az egyes éleken élköltségeket értelmezünk az út hossza (költsége) és az áthaladás idejének megjelenítéséhez. Egyértelmű, hogy a megoldásban két egymás után következő fogyasztó, vagy egy fogyasztó és egy depó között a lehető legrövidebb úton fogunk haladni. Ezért értelmezhetünk egy teljes gráfot a következők szerint. Legyenek a teljes gráf pontjai a depók és a fogyasztók, és a gráf bármely $uv$ élének a hossza (költsége) legyen az $u$ és $v$ (melyek fogyasztók vagy depók) között menő legrövidebb (legolcsóbb) út hossza az eredeti gráfban, valamint az $uv$ menetideje legyen a legrövidebb út éleinek menetidejének az összköltsége. Ez a gráf lehet szimmetrikus vagy aszimmetrikus aszerint, hogy az eredeti gráfban az utak költsége és menetideje oda-vissza szimmetrikus volt-e, vagy nem. A továbbiakban a problémát ezen a teljes gráfon fogjuk vizsgálni.

A *fogyasztók* jellemzése. A fogyasztók talán legfontosabb tulajdonsága az *elhelyezkedése* és az *igénye*. Utóbbi adja meg, hogy a javakból mekkora mennyiségre van szüksége. A továbbiakban az $i$. fogyasztó igényét $d_i$-vel fogjuk jelölni. Előfordulhat, hogy egy fogyasztó a napnak csak egy bizonyos időszakában szolgálható ki. Ez lehet például az üzlet nyitvatartási ideje. Ezért minden fogyasztóra értelmezhetünk egy időintervallumot (*time windows*), amelyikben ő kiszolgálható. Ezen kívül egyes esetekben fontos lehet a *kirakodás idejének* figyelembe vétele. Ez az az idő, amelyet a járműnek a fogyasztónál el kell

töltenie. Ezt az adott fogyasztóból kiinduló élek menetidejének alkalmas számmal való megnövelésével jól tudjuk kezelni. Előfordulhat, hogy nem tudjuk minden fogyasztó igényét kielégíteni. Ez akkor történhet meg, ha például túl kevés jármű áll a rendelkezésünkre, vagy nem elég a rendelkezésre álló idő. Ilyenkor akár csökkentjük a kiszállított mennyiséget, akár kiszolgálatlanul hagyunk néhány fogyasztót, mindenképpen veszteségünk lesz. Ezért a fogyasztókhoz rendelhetünk különböző *prioritásokat* vagy *büntetéseket*, amelyek a kiszolgálatlanul hagyásból adódó veszteségekből származnak.

A *depó* jellemzése. A depót az elhelyezkedésével jellemezhetjük. Ezen kívül minden depóra megadhatjuk, hogy mely jármű fajtából hányat lehet ott elhelyezni, valamint az ott tárolható javak maximális mennyiségét.

A *járművek* jellemzése. A feladat kitűzésekor mindenképpen meg kell adni, hogy a járműveknek egy rögzített számával rendelkezünk-e, vagy a jobb megoldás érdekében változtathatunk-e ezen a számon. (Például előfordulhat, hogy az összköltség kevesebb lesz, ha nem használunk fel minden járművet, vagy a fogyasztók kiszolgálásához nem rendelkezünk elég járművel.) A felhasználható járművek rögzített számát $K$-val fogjuk jelölni. A járművek egyik legfontosabb tulajdonsága a *kapacitás*. Ez adja meg, hogy legfeljebb mennyi rakományt képes a jármű szállítani. Így az egy jármű által meglátogatott fogyasztók számára egy felső korlátot nyerünk, hiszen egyértelmű, hogy egy jármű legfeljebb azokat a fogyasztókat tudja teljesen kiszolgálni, amelyek összesített igénye nem haladja meg a kapacitását. A következőkben az $i$. jármű kapacitását $C_i$-vel fogjuk jelölni. Ha minden járműnek azonos a kapacitása, akkor ezt a közös értéket $C$-vel jelöljük. A kapacitáson kívül minden járműhöz meg kell adni, hogy melyik depóból indul, és melyik depókba érkezhet. A járművek használatának *költsége* is van, ami az egy távolság- vagy időegységre vonatkozó összeg. A járműre adott feltételek közé sorolhatjuk a sofőrökre vonatkozó feltételeket is. Ezek lehetnek például a maximális munkaórák száma, valamint a pihenők száma és ideje.

A VRP-nek amellett, hogy rendkívül sok feltétele lehet, az egyes esetekben a célok is különbözőek lehetnek. A leggyakrabban a következő célok szoktak előfordulni:

1. Az összköltség minimalizálása
2. A járművek számának a minimalizálása úgy, hogy minden fogyasztó igénye ki legyen elégítve.
3. Jól kiegyensúlyozni az egyes utak menetidejét és a szállított rakomány mennyiségét.
4. Minimalizálni a büntetéseket abban az esetben, amikor nem tudunk minden fogyasztót kiszolgálni.

## 1.3. VRP típusai

A fentiekből láthattuk, hogy a VRP feltételei nagyon sokfélék lehetnek. A könnyebb kezelhetőség érdekében a VRP különböző típusait értelmezhetjük attól függően, hogy milyen feltételeket szabunk meg.

A VRP legalapvetőbb formája a *Capacitated VRP* (CVRP). A többi típust ennek általánosításaként kapjuk. A CVRP fő tulajdonságai a következők:

1. Az igények determinisztikusak (előre megadottak)
2. Azonos járműveket használunk
3. Egyetlen depó van
4. Minden járműre csak kapacitási korlát adott.
5. Célunk az összköltség minimalizálása

Legyen $G = (V, A)$ teljes gráf, melynek a ponthalmaza: $V = \{v_0, v_1, \ldots, v_n\}$, és az élhalmaza $A$. A depónak a $v_0$ felel meg, a többi $n$ pont felel meg a fogyasztóknak. Az út költségét a $c_{ij}$ költségfüggvény reprezentálja minden $(i, j)$ élre. Általában hurokélt nem engedünk meg, így legyen $c_{ii} = +\infty$. Ha $G$ irányított gráf, akkor $c$ értéke aszimmetrikus. Ezt aszimmetrikus CVRP-nek (ACVRP-nek) nevezzük. Ha $G$ irányítatlan, akkor $c_{ij} = c_{ji}$ minden $(i, j) \in A$ esetén, és ezt az esetet szimmetrikus CVRP-nek (SCVRP)-nek nevezzük. Könnyen látható, hogy ha $c_{ij}$ költségek mindig az eredeti gráfban az $i$ és $j$ között menő legrövidebb utakat jelölik, akkor a háromszög-egyenlőtlenség teljesül. Egyes tesztesetekben $V$ pontjai a koordináta-rendszer pontjai, és $c$ a köztük lévő euklideszi távolság. Ez szimmetrikus, és teljesíti a háromszög-egyenlőtlenséget. (Hogy ne is sértsük meg a háromszög-egyenlőtlenséget, a valós költségeket felfele kerekítjük.) Ezeket az eseteket Euclidean SCVRP-nek hívjuk.

Minden $v_i$ fogyasztóra $(i = 1, \ldots n)$ adott a $d_i \geq 0$ igénye, és a depóra értelmezünk egy fiktív $d_0 = 0$ igényt. Egy $S \subseteq V$ ponthalmaz esetén $d(S) = \sum_{i \in S} d_i$ jelöli a halmaz összesített igényét. A megoldhatóság érdekében tegyük fel, hogy $d_i \leq C$ minden $(i = 1, \ldots, n)$ esetén.

Minden jármű legfeljebb egy utat teljesít, és feltesszük, hogy $K \geq K_{min}$, ahol $K_{min}$ jelöli, hogy legalább hány járműre van szükségünk ahhoz, hogy minden fogyasztó igényét kielégítsük. Legyen $S \subseteq V \backslash \{v_0\}$. Ekkor jelölje $r(S)$ azt a minimális számot, ahány járműre szükség van az $S$-beli fogyasztók kiszolgálásához. Így $K_{min} = r(V \backslash \{v_0\})$. $r(S)$

értékének pontos kiszámításához a CVRP-re értelmezett *Bin packing problémát* (BPP) kellene megoldani. Mivel ez egy NP-teljes feladat, gyakran használják a BPP egy triviális alsó korlátját, amely ebben az esetben: $\lceil \frac{d(S)}{C} \rceil$.

Lényegében, amikor megoldjuk a CVRP-t, a következő a feladatunk. Meg kell találnunk pontosan $K$ darab kört minimális összköltséggel úgy, hogy

1. minden kör érinti a depó pontot,
2. minden fogyasztó pontosan egy körben szerepel és
3. a kör által bejárt fogyasztók összesített igénye ne haladja meg $C$-t.

Előfordulhatnak olyan esetek, hogy nem kell minden járművet felhasználni, azaz $K > K_{min}$. Ekkor az a cél, hogy minimalizáljuk a járművek számát, beleértendő az összköltségek minimalizálásába. Ekkor a megfelelő feltételek úgy módosulnak, hogy nem *pontosan* $K$ utat keresünk, hanem *legfeljebb* $K$ utat.

CVRP esetén fontos előre tisztázni, hogy megengedjük-e azt, hogy egy jármű csak egy fogyasztót látogasson meg. Ebben az esetben ugyanis bizonyos élekre meg kell engednünk, hogy egynél többször legyenek felhasználva.

Észrevehetjük, hogy a TSP a CVRP egy speciális esete. A TSP ugyanis egy olyan CVRP, amelyben $C \geq d(V)$ és $K = 1$. Innen rögtön látszik, hogy a CVRP NP-teljes probléma. Ezen kívül azt is elmondhatjuk, hogy minden relaxációja a TSP-nek érvényes a CVRP-re is.

A *Distance-Constrained VRP* (DVRP) a CVRP egy variációja. Akkor kapunk DVRP-t, ha az eredeti kapacitási feltételek helyett a menetidőt korlátozzuk. Ez azt jelenti, hogy minden élre értelmezünk egy $t$ menetidőt, és kikötjük, hogy egy jármű által bejárt útvonal menetideje nem haladhat meg egy adott $T$ értéket. Továbbá minden fogyasztóra megadhatunk egy $s_i$ (*service time*) értéket is, amely azt adja meg, hogy a járműnek mennyi időt kell eltöltenie az adott fogyasztónál. Ezt úgy tudjuk könnyen kezelni, ha az $s$ értékeket megfelelően hozzáadjuk a menetidőhöz a következő formában:

$$t'_{ij} = t_{ij} + \frac{s_i}{2} + \frac{s_j}{2}$$

A célunk az, hogy minimalizáljuk a teljes út időtartamát, ha a kiszolgálás idejét is hozzáadtuk a menetidőhöz.

Ha a kapacitási és menetidőre vonatkozó feltételeket is megköveteljük, akkor *Distance-Constrained CVRP*-ről beszélünk.

A *VRP width Time Windows* (VRPTW) a CVRP egy kiterjesztése. Ekkor minden $i$-re az $i$. fogyasztóhoz hozzárendelünk egy $[a_i, b_i]$ időintervallumot. Adott az az időpillanat, amikor a jármű elhagyja a depót, és adott a menetidő minden $(i, j)$ élre, valamint minden fogyasztónak az $s_i$ kiszolgálás ideje. Minden fogyasztónál a kiszolgálás a megfelelő időintervallumba kell, hogy essen.

A *VRP with Backhauls* (VRPB) szintén a CVRP egy kiterjesztése. Ekkor a fogyasztókat két részhalmazba osztjuk. Jelölje $L$ azon $n$ darab fogyasztó halmazát, melyek mindegyike azt igényli, hogy bizonyos mennyiségű árut kiszállítsanak neki. $B$ legyen az az $m$ darab fogyasztó halmaza, melyeknél a járműnek egy adott mennyiségű árut fel kell venni. A feltétel az, hogy minden $L$-beli fogyasztót ki kell szolgálni a $B$-beliek előtt. Itt $d_i$ jelöli a kiszállításra vagy begyűjtésre adott igényeket.

A *VRP with Pickup and Delivery* (VRPPD) esetén minden fogyasztóra értelmezünk egy $d_i$ és egy $p_i$ értéket, melyek azt mutatják, hogy az adott fogyasztónál mennyi árut kell letenni, és mennyit felvenni. Néha erre csak a $d'_i = d_i - p_i$ értéket használjuk. Ezen kívül minden fogyasztóra adott, hogy honnan igényli a szállítást ($O_i$), és hogy neki hova kell küldeni az árut ($D_i$). Tehát az új feltételünk az, hogy minden jármű aktuális rakománya nemnegatív legyen, illetve hogy minden $i$ fogyasztóra teljesüljön, hogy ugyanaz a jármű látogassa meg, mint $O_i$-t és $D_i$-t, valamint hogy a jármű előbb látogatja meg $O_i$-t, mint $i$-t, és $D_i$-t később.

Mivel a dolgozat terjedelme korlátozva van, ezért a továbbiakban csak a VRP legspeciálisabb típusával, a CVRP-vel fogok foglalkozni, esetenként pedig kitérek arra, hogy hogyan lehet az adott módszert DCVRP-re alkalmazni.

## 1.4. Alapvető modellek

A VRP matematikai modellezésének három alapvető megközelítése van. Ezek használata függ attól, hogy a VRP melyik esetével foglalkozunk. Előfordulhat ugyanis, hogy bizonyos feltételeket könnyebb az egyik modellbe beépíteni, mint a másikba. Azonban nem lehet figyelmen kívül hagyni azt a tényt sem, hogy a különböző modellek az adott eredmény minőségében is nagyban különbözhetnek egymástól.

### 1.4.1. Vehicle flow models

**Aszimmetrikus eset** Először az ACVRP-t vizsgáljuk. Legyen $x$ egy élszám hosszú bináris vektor, melynek minden koordinátája azt adja meg, hogy az adott élt felhasználtuk-e a megoldás során. Az $x_{ij}$ jelöli az $(i, j)$ élhez tartozó koordinátát az $x$ vektorban. Ekkor a

problémát a következők szerint írhatjuk le.

$$\sum_{i \in V} x_{ij} = 1 \qquad \forall j \in V \backslash \{v_0\} \tag{1}$$

$$\sum_{j \in V} x_{ij} = 1 \qquad \forall i \in V \backslash \{v_0\} \tag{2}$$

$$\sum_{i \in V} x_{i0} = K \tag{3}$$

$$\sum_{j \in V} x_{0j} = K \tag{4}$$

$$\sum_{i \notin S} \sum_{j \in S} x_{ij} \geq r(S) \qquad \forall S \subseteq V \backslash \{v_0\}, \quad S \neq \emptyset \tag{5}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i, j \in V \tag{6}$$

$$\min cx \tag{7}$$

Itt az (1)-(4) feltételek vonatkoznak az egyes pontok ki- és befok feltételeire, azaz hogy minden fogyasztót pontosan egyszer látogassunk meg, és a depót pontosan $K$ darab út tartalmazza. Az (5) feltételt *capacity-cut constraints*-nek (CCCs) hívjuk. Ez gondoskodik arról, hogy az utak megfeleljenek a kapacitási feltételeknek, és hogy a megoldás összefüggő legyen. A feltételek szükségessége könnyen látható. Lássuk be, hogy elégséges is, azaz hogy minden $x$ karakterisztikus vektor, amely a fentieket kielégíti, valóban egy jó CVRP megoldást ad. Vegyünk egy ilyen $x$ megoldást. Ez (1) és (2) miatt biztosan meglátogat minden fogyasztót pontosan egyszer, és (3), (4) miatt a depót pontosan $K$-szor. Mivel (5) miatt a megoldásnak összefüggőnek kell lennie, ezért minden körnek tartalmaznia kell a depót, ami (3) és (4) miatt csak úgy történhet, ha minden kör pontosan egyszer tartalmazza a depót. Már csak azt kellene látni, hogy nem fordulhat elő olyan út, amely teljesíti a fenti feltételeket, azonban az igények összege meghaladja a jármű kapacitását. Ennek belátásához indirekt tegyük fel, hogy létezik $R$ út, amely egy, a fentieket teljesítő megoldásban szereplő kör, azonban az $R$-ben szereplő igények összege meghaladja a jármű kapacitását. Ekkor az $R$ út pontjai által alkotott ponthalmaz sérti az (5) feltételt, hiszen ha a $d(R)$ meghaladja $C$-t, akkor kell legalább két jármű, amely kiszolgálja $R$ pontjait, tehát $x$ nem lehetett megengedett megoldás.

(1)-(4) feltételek miatt $\sum_{i \notin S} \sum_{j \in S} x_{ij} = \sum_{i \in S} \sum_{j \notin S} x_{ij}$ minden $S \subseteq V \backslash \{v_0\}$ esetén, ahol $S \neq \emptyset$. Így az $S$ halmaz, és $V \backslash S$ komplementere közül egyiknek sincs kitüntetett szerepe, ezért az (5) feltételre ekvivalens átfogalmazást nyerünk:

$$\sum_{i \notin S} \sum_{j \in S} x_{ij} \geq r(V \backslash S) \qquad \forall S \subset V, \quad v_0 \in S.$$

Az (5) feltételnek egy másik felírása (1)-(4) feltételek segítségével:

$$\sum_{i\in S}\sum_{j\in S} x_{ij} \leq |S| - r(S) \qquad \forall S \subseteq V\backslash\{v_0\}, \quad S \neq \emptyset. \tag{8}$$

Ezt *generalized subtour elimination constraints*-nek (röviden GSECs-nek) hívjuk. Ahhoz, hogy belássuk, hogy ez az egyenlőtlenség (1)-(5) feltételekből következik, vizsgáljuk meg a bal oldalon szereplő összeget. Ez azon éleket számlálja meg, amelyek beletartoznak az $x$ által jelölt megoldásba, és mindkét végpontjuk $S$-ben van. Mivel $S$ nem tartalmazza a depó pontot, így (1) és (2) feltételek miatt minden $S$-beli pontba pontosan egy $x$-beli él érkezik be. Így a bal oldalon szereplő szummát felírhatjuk úgy, hogy $S$ elemszámából kivonjuk azon élek számát, melyek belépnek $S$-be. Azaz

$$\sum_{i\in S}\sum_{j\in S} x_{ij} = |S| - \sum_{i\notin S}\sum_{j\in S} x_{ij} \leq |S| - r(S).$$

Visszafelé, az (1)-(4) és (8) feltételekből következik (5), ami azonnal látszik, ha (8)-ban az előzőhöz hasonlóan átírjuk a bal oldalon szereplő szummát, az egyenlőtlenség mindkét oldalából levonjuk $|S|$-et, és szorozzuk mindkét oldalt -1-gyel. Ezzel beláttuk, hogy (8) az (5) feltétellel ekvivalens.

**Szimmetrikus eset** Az aszimmetrikus esethez hasonlóan értelmezhetjük, mivel a megoldás szempontjából lényegtelen, hogy a jármű melyik irányban halad végig a megfelelő úton. Viszont az előző esettől eltérően most meg kell különböztetni azokat az eseteket, amikor megengedjük, hogy egy jármű csak egy fogyasztót szolgáljon ki, és amikor nem. Ez azért fontos, mert előbbi esetben előfordulhat, hogy egy jármű egy élt többször is felhasznál. Ilyenkor az egyik megoldás, hogy a depóból induló éleket megduplázzuk, és a megfelelő koordinátákat is kétszer szerepeltetjük $x$-ben. Másik lehetőség, hogy $x$ megoldásban megengedjük, hogy a kettes is szerepeljen a koordináták között. Így a szimmetrikus esetre az

alábbi egyenlőtlenség-rendszert írhatjuk fel:

$$\sum_{e\in\delta(i)} x_e = 2 \qquad \forall i \in V\backslash\{v_0\} \tag{9}$$

$$\sum_{e\in\delta(v_0)} x_e = 2K \tag{10}$$

$$\sum_{e\in\delta(S)} x_e \geq 2r(S) \qquad \forall S \subseteq V\backslash\{v_0\}, \quad S \neq \emptyset \tag{11}$$

$$x_e \in \{0,1\} \qquad \forall e \notin \delta(v_0) \tag{12}$$

$$x_e \in \{0,1,2\} \qquad \forall e \in \delta(v_0) \tag{13}$$

$$\min cx \tag{14}$$

Az előző esethez hasonlóan, most is felírhatjuk a (11) feltételt másik alakban:

$$\sum_{e\in E(S)} x_e \leq |S| - r(S),$$

ahol $E(S)$ jelöli azokat az éleket, melyek mindkét végpontja $S$-ben van. Annak a belátása, hogy ezek a feltételek pontosan a megengedett megoldásokat írják le, az aszimmetrikus esethez hasonlóan történik.

**Kiterjesztések** A fenti modelleket *két indexű* modelleknek nevezzük. [1] szerint ezek jól alkalmazhatók az ACVRP és SCVRP alapvető eseteire, viszont a komplexebb verziókra általában alkalmatlanok. Ezen kívül csak akkor tudjuk őket használni, ha a megoldás költsége csak a bejárt élek összköltségétől függ. Azaz nem alkalmas, ha a költség például függ a jármű típusától, vagy a pontok sorrendjétől. Megjegyezzük, hogy az előbbi eset kiküszöbölhető az ún. *három indexű modellek* használatával, melyek esetén explicit jelöljük, hogy melyik jármű használja az adott élt, amiről bővebben [1] 15. oldalán olvashatunk.

Ha a céljaink közé tartozik az is, hogy a lehető legkevesebb járművet használjuk, akkor több lehetőségünk is van a fenti feltételek megfelelő módosítására. Ez akkor is fontos lehet ha a járművek használatának fix költsége van, és a teljes költség a járművek használatától is függ. Néhány példa az ilyen esetek kezelésére:

1. A depóra vonatkozó (3), (4) feltételeket lecseréljük a $\sum_{i\in V} x_{i0} \leq K$, $\sum_{j\in V} x_{0j} = \sum_{i\in V} x_{i0}$ feltételekre.

2. A BPP segítségével kiszámoljuk a $K_{min}$ értéket, és a fenti rendszerekben ezt írjuk $K$ helyére.

3. Egy nagy konstans értéket adunk a depóból induló élek költségéhez, így az algoritmus majd a lehető legkevesebb ilyen élt fogja felhasználni, így minimalizálja a körök számát. (Nyilvánvaló, hogy ez arra az esetre vonatkozik, amikor $K$-nál kevesebb járművet is megengedünk)

Azokban az ACVRP-re vonatkozó esetekben, amikor nem akarjuk megengedni, hogy egy jármű csak egy fogyasztót szolgáljon ki, bevehetjük a következő feltételt:

$$x_{0j} + x_{j0} \leq 1$$

Ugyanezt az SCVRP esetében úgy érhetjük el, hogy nem engedjük meg a depóból induló élekre a kettes értéket. A valóságban azonban ritkán számítanak ezek a feltételek. Az egy fogyasztót ($v_j$) tartalmazó utak ugyanis csak akkor fordulhatnak elő, ha a maradék $K-1$ jármű ki tudja szolgálni a maradék $n-1$ fogyasztót, azaz ha $r(V\backslash\{v_j\}) \leq K-1$. Ha $r$ a triviális alsó becslés, akkor ez azt jelenti, hogy

$$d_j \geq C_{min} = d(V) - (K-1)C.$$

A valóságban általában ilyen nagy igény nem szokott előfordulni.

### 1.4.2. Commodity flow models

A feladat most következő leírása a két indexű modellektől abban különbözik, hogy további változókat használ fel az egyes éleken átfolyó áruk reprezentálásának céljából. Mivel a következő, folyamokra vonatkozó formulák explicit tartalmazzák az élek irányítását, ezért csak az ACVRP esetét nézzük meg.

Vegyünk egy $G' = (V', A')$ teljes gráfot, amely $G$-ből jön létre úgy, hogy hozzáveszünk egy $n+1$. pontot, ami a depó másolata. Így a körök most olyan utaknak felelnek meg, melyek a $v_0$ pontból indulnak, és $v_{n+1}$-be érkeznek. Értelmezzünk minden $(i,j) \in A'$ élre két nemnegatív folyamváltozót, $y_{ij}$-t és $y_{ji}$-t. Ha a jármű $i$-ből $j$-be megy, akkor $y_{ij}$ adja meg az aktuális rakományt, $y_{ji}$ pedig a fennmaradó kapacitást, azaz $y_{ji} = C - y_{ij}$. A szerepek felcserélődnek, ha a jármű $j$-ből megy $i$-be. Vegyük észre, hogy az $y_{ij}$ és $y_{ji}$ szerepe valóban szimmetrikus, hiszen $y_{ij} + y_{ji} = C$. Tehát most egy jármű által bejárt utat úgy képzelünk el, hogy minden irányított $(i,j)$ él mellé behúzunk egy $(j,i)$ élt is. Az $(i,j)$ élen áthaladó áru mennyisége $y_{ij}$, a $(j,i)$ élen „áthaladó" mennyiség $y_{ji}$. Tehát egy lehetséges megoldásban minden útra a folyamváltozók két irányított utat határoznak meg: az egyik megy $v_0$-ből $v_{n+1}$-be, a másik $v_{n+1}$-ből $v_0$-ba. Az előbbi a rakományt reprezentálja, utóbbi

pedig a fennmaradó kapacitást. $x$ továbbra is a megoldások karakterisztikus vektora. Így a következő egyenlőtlenség-rendszert írhatjuk fel.

$$\sum_{j \in V'} (y_{ji} - y_{ij}) = 2d_i \qquad \forall i \in V' \backslash \{v_0, v_{n+1}\} \tag{15}$$

$$\sum_{j \in V' \backslash \{v_0, v_{n+1}\}} y_{0j} = d(V \backslash \{v_0, v_{n+1}\}) \tag{16}$$

$$\sum_{j \in V' \backslash \{v_0, v_{n+1}\}} y_{j0} = KC - d(V \backslash \{v_0, v_{n+1}\}) \tag{17}$$

$$\sum_{j \in V' \backslash \{v_0, v_{n+1}\}} y_{n+1,j} = KC \tag{18}$$

$$y_{ij} + y_{ji} = Cx_{ij} \qquad \forall (i,j) \in A' \tag{19}$$

$$\sum_{j \in V'} (x_{ij} + x_{ji}) = 2 \qquad \forall i \in V' \backslash \{v_0, v_{n+0}\} \tag{20}$$

$$y_{ij} \geq 0 \qquad \forall (i,j) \in A' \tag{21}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in A' \tag{22}$$

$$\min cx \tag{23}$$

Itt (19) $y$ definíciója miatt kell, hogy teljesüljön. A (15) feltétel azt mutatja, hogy az $i$. fogyasztónál el kell helyeznünk $d_i$ mennyiségű árut. Hiszen az $i$. fogyasztó meglátogatása előtti rakomány, és a látogatás utáni rakomány különbsége megadja, hogy mennyit kapott az $i$. fogyasztó. Hasonlóan, a látogatás előtti és utáni üres helyek számának különbsége szintén megegyezik az elhelyezett rakománnyal. A (16) feltétel azt jelenti, hogy a depóból éppen annyi áru megy ki, amennyit el fogunk helyezni az út során, (17) pedig azt mutatja, hogy az eredeti depóba éppen annyi üres hely „folyik be", amennyi szabad hellyel indul el a jármű a depóból ($y$ definíciója miatt). (18) feltétel szerint az $n+1$-gyel jelölt depóba az üres jármű megy be. Végül pedig (20) gondoskodik a megfelelő fokszám feltételek teljesüléséről.

### 1.4.3. Set-Partitioning models

Legyen $\mathcal{H} = \{H_1, \ldots, H_q\}$ a $G$ összes körének a halmaza. Az $x \in \mathbb{R}^q$ vektor $i$. koordinátája pontosan akkor egy, ha az $i$. kör szerepel a megoldásban. Minden $H_i$ körnek van egy $c_i$ költsége. Ezen kívül legyen $a_{ji}$ olyan változó, amelynek értéke egy, ha $v_j$ a $H_i$ kör által van

meglátogatva, különben nulla. Ekkor a feladatunk a következőképp fogalmazható meg.

$$\sum_{j=1}^{q} a_{ij} x_j = 1 \qquad \forall i \in V \backslash \{v_0\} \tag{24}$$

$$\sum_{j=1}^{q} x_j = K \tag{25}$$

$$x_j \in \{0,1\} \qquad \forall j = 1, \ldots, q \tag{26}$$

$$\min cx \tag{27}$$

Az (24) azt jelenti, hogy minden fogyasztó pontosan egy körben szerepel. (25) pedig azt követeli meg, hogy pontosan $K$ darab kört válasszunk a megoldásba. Vegyük észre, hogy ennek a feladatnak egy lehetséges $x$ megoldása nem feltétlenül megoldása a CVRP-nek is. Ez a felírás ugyanis nem követeli meg sem azt, hogy minden kör tartalmazza a depót, sem pedig azt, hogy egy körben szereplő fogyasztók összesített igénye ne haladja meg $C$-t. A későbbiekben látni fogjuk a modell egy olyan alkalmazását, amelyben a nem megengedett utaknak megfelelő koordinátákat eltöröljük a változók közül. Ezzel nagymértékben csökkentjük a feladatban szereplő utak számát, amely eredetileg exponenciális méretű is lehet.

Észre lehet venni, hogy ha $c$ kielégíti a háromszög-egyenlőtlenséget, akkor a set partitioning modell (SP) átírható egy vele ekvivalens *set covering* (SC) modellre úgy, hogy a (24) feltételt lecseréljük a

$$\sum_{j=1}^{q} a_{ij} x_j \geq 1$$

feltételre. Az azonnal látszik, hogy az SP egy megoldása SC-t is megoldja. Fordítva, ha egy megoldás megengedett megoldása SC-nek, de SP-nek nem, akkor ezt könnyen áttranszformálhatjuk egy megengedett SP megoldássá anélkül, hogy az összköltség növekedjen. Ekkor ugyanis létezik olyan fogyasztó, melyet egynél több kör fed le. Mivel a háromszög-egyenlőtlenség teljesül, ha az egyik ilyen kör kihagyja ezt a fogyasztót, a költségeink biztosan nem növekedtek.

# 2. Alulról közelítő heurisztikák

Ebben a fejezetben azon eljárásokat fogjuk megnézni, melyek olyan módszereket használnak, hogy bizonyos feltételeket elhagynak az eredeti feladatból, ezzel nem megengedett megoldásokon keresztül közelítik a CVRP optimumát.

## 2.1. Branch-and-bound

Mivel a CVRP a TSP egy általánosítása, így sok megoldási módszer származik a TSP-re adott heurisztikákból. Egyik leghatékonyabb közülük a *Branch-and-Bound*, amiről ez a paragrafus szólni fog. Először megnézzük, milyen módszerekkel lehet jó alsó korlátot találni egy megoldásra, majd a szétválasztás (*branching*) eljárásait ismertetjük.

### 2.1.1. Alsó korlátot kereső módszerek

A továbbiakban megkülönböztetjük a szimmetrikus, és aszimmetrikus eseteket. (Ha mégsem akarunk különbséget tenni, csak CVRP-nek hívjuk a problémát)

**Visszavezetés párosításokra** Először az aszimmetrikus esettel foglalkozunk. A most következő ötlet a [2] cikkből származik. A vehicle flow modellre adunk egy relaxációt: hagyjuk el az (5) feltételt, így a feladat a következőképpen módosul. Keressük körök olyan összeállítását, hogy minden fogyasztót pontosan egyszer látogatunk meg, és a depót pontosan $K$-szor. Viszont nem követeljük meg, hogy minden kör áthaladjon a depón, és azt sem, hogy egy körön belül az igények összege ne haladja meg $C$-t. A kapott feladatot *Transportation Problem*-nek (TP-nek) nevezzük.

Legyen $G'$ az a teljes gráf, melyet $G$-ből kapunk úgy, hogy a depóról készítünk $K-1$ darab másolatot. Az új, $c'$ élköltséget a következőképp értelmezzük.

$$c'_{ij} = \begin{cases} c_{ij} & \forall i,j \in V\backslash\{v_0\} \\ c_{i0} & \forall i \in V\backslash\{v_0\}, \quad j \in W \\ c_{0j} & \forall i \in W, \quad j \in V\backslash\{v_0\} \\ \lambda & \forall i,j \in W \end{cases}$$

ahol $W$ a depó pontok halmaza, és $\lambda$ valamilyen alkalmas érték (alkalmas megválasztásával meghatározhatjuk, hogy legfeljebb vagy pontosan $K$ darab járművet szeretnénk megválasztani). Így a depókra a fogyasztókhoz hasonló fokszámfeltételt szabhatunk, vagyis a

következő rendszert írhatjuk fel.

$$\text{(AP)} \tag{28}$$

$$\sum_{i \in V} x_{ij} = 1 \qquad \forall j \in V' \tag{29}$$

$$\sum_{j \in V} x_{ij} = 1 \qquad \forall i \in V' \tag{30}$$

$$x_{ij} \geq 0 \qquad \forall i, j \in V' \tag{31}$$

$$L_{AP} = \min c'x \tag{32}$$

Ezt AP-vel fogjuk jelölni (*assignment problem*), ugyanis a feladatunk úgy módosult, hogy legolcsóbb teljes párosítást kell találnunk. Ez a következő transzformációból látszik. A gráf minden $u$ pontját „húzzuk szét" $u_1$ és $u_2$-re úgy, hogy az $u$-ba bemenő élek $u_1$-be menjenek, az $u$-ből kimenő élek pedig $u_2$-ből menjenek ki. Így az eredeti gráfban minden irányított kör megfelel az új gráfban egy teljes párosításnak.

Most vizsgáljuk meg az irányítatlan esetet, melyről részletesebben [3]-ban olvashatunk. Hagyjuk el megint a kapacitási feltételeket, vagyis (11)-t. Az előzőekhez hasonlóan a feladatunk megint az, hogy megtaláljuk az olyan körök legolcsóbb összeállítását, melyek minden fogyasztót pontosan egyszer fednek le, a depón pedig összesen $2K$-szor haladnak át. Látható, hogy ez nem ad feltétlenül megengedett SCVRP megoldást. A problémát a következőképp írhatjuk fel. Legyen $b_i = 2$, ha $i$ egy fogyasztót jelöl, és $b_0 = 2K$ a depó esetén. Így egy $b$-párosítási feladatot kapunk.

$$\sum_{e \in \delta(v_i)} x(e) = b_i \qquad \forall i \in V \tag{33}$$

$$x(e) \in \{0, 1\} \qquad \forall e \notin \delta(v_0) \tag{34}$$

$$x(e) \in \{0, 1, 2\} \qquad \forall e \in \delta(v_0) \tag{35}$$

$$L_{bM} = \min cx \tag{36}$$

Az előzőekhez hasonlóan, ha a depóról készítünk $K - 1$ másolatot, akkor a $b$-párosításból 2-párosítás lesz.

**Visszavezetés fenyőkre és fákra** A következő relaxációt megint először az aszimmetrikus esetre nézzük meg. Hagyjuk el most a fogyasztókra vonatozó kifok feltételeket (2), és gyengítsük a kapacitásra vonatkozó (5) feltételeket úgy, hogy a kiszolgálhatóságot nem követeljük meg, csak az összefüggőséget. Ez azt jelenti, hogy (5)-ben $r(S)$-t 1-re cseréljük.

Így kapjuk a *K-legolcsóbb feszítő fenyő* problémát (*K-shortest spanning arborescence*, amit KSSA-nak rövidítünk).

$$\text{(KSSA)} \tag{37}$$

$$\sum_{i \in V} x_{ij} = 1 \qquad \forall j \in V \backslash \{v_0\} \tag{38}$$

$$\sum_{i \in V} x_{i0} = K \tag{39}$$

$$\sum_{j \in V} x_{0j} = K \tag{40}$$

$$\sum_{i \notin S} \sum_{j \in S} x_{ij} \geq 1 \qquad \forall S \subseteq V \backslash \{v_0\}, \quad S \neq \emptyset \tag{41}$$

$$x_{ij} = \{0, 1\} \qquad \forall i, j \in V \tag{42}$$

$$L_{KSSA} = \min cx \tag{43}$$

A feladat két elkülöníthető problémára bomlik. Egyrészt meg kell találnunk a legolcsóbb feszítőfenyőt, ahol a depó (gyökér) pont kifoka $K$. Másrészt meg kell találnunk azt a legolcsóbb $K$ darab élt, amely belép a depóba. [4], [5] szerint az első feladatot megoldó algoritmus $O(n^2)$, a másodikat megoldó pedig $O(n)$ futásidejű, így a megoldást $O(n^2)$ időben találhatjuk meg. Egy másik alsó becslést kaphatunk, ha az fenyőben az elágazások a fogyasztóktól a depóba menő utak uniója, vagyis amikor KSSA-ban az utak az ellenkező irányba mennek. Ezt jelöljük KSSAA-val. Tehát érdemes a kapott két alsó becslés közül a nagyobbikat kiválasztani, hiszen ez lesz közelebb az optimumhoz. Az így kapott alsó korlát tehát $L'_{KSSA} = \max\{L_{KSSA}, L_{KSSAA}\}$.

Szimmetrikus esetben a TSP-re adott *1-fa* alsó becslést próbáljuk meg kiterjeszteni. Erről legkorábban a [6]-ban olvashatunk. A most következő módszer [7]-ből származik. Most $K$-fákat fogunk értelmezni, melyeket úgy kapunk, hogy vesszük a depó pontot, és belőle kiinduló $2K$ darab élt, majd a gráf többi pontján veszünk egy feszítőfát. Tehát most úgy szeretnénk alsó becslést adni, hogy megkeressük a legolcsóbb K-fát, amit a következő IP feladat megoldásával tudunk megtalálni:

$$\sum_{e \in \delta(v_0)} x_e = 2K \tag{44}$$

$$\sum_{e \in \delta(S)} x_e \geq 1 \qquad S \subseteq V \backslash \{v_0\}, \quad S \neq \emptyset \tag{45}$$

$$x_e \in \{0, 1\} \qquad \forall e \in E \tag{46}$$

$$L_{Kt} = \min cx \tag{47}$$

Természetesen ez nem lehet SCVRP-nek megoldása, hiszen lehetnek olyan fogyasztók, melyek fokszáma nem kettő, valamint lehetnek olyan ágak, melyek összesített igénye meghaladja a járműi kapacitását.

Az eddig látott alsó korlátot kereső módszerek gyenge eredményt adnak. Az [1] 34. oldalán található táblázatok alapján látszik, hogy a most ismertetett alsó becslések az optimumtól távol eshetnek. Míg $L_{AP}$ átlagosan 8,7%-kal tér el az optimális értéktől, addig ez az arány $L'_{KSSA}$-nál 22,9% is lehet. Egyértelmű, hogy további módszerekre lesz szükség ahhoz, hogy elég jó alsó korlátot találjunk.

**Additív korlátozás** Az alsó korlátok keresésének additív megközelítése [8] cikkből származik. Most a stratégiánk az lesz, hogy megpróbáljuk a fent említett módszereket kombinálni. Ha alkalmazunk egy minimalizáló eljárást egy bizonyos $\min\{cx : x \in F\}$ feladatra, akkor kapunk eredményül egy $\rho_1$ alsó korlátot, és egy $\tilde{c_1} \geq 0$ *maradék* költségmátrixot, amelyre teljesül, hogy $\rho_1 + \tilde{c_1}x \leq cx$, minden $x \in F$ esetén. Ha most ismét alkalmazunk egy minimalizáló eljárást $c$ helyett $\tilde{c_1}$-re, akkor kapunk eredményül egy $\rho_2$ alsó korlátot, és egy $\tilde{c_2}$ maradék költséget, amire $\rho_2 + \tilde{c_2}x \leq \tilde{c_1}x$. Utóbbi egyenlőtlenséget az elsőbe helyettesítve kapjuk, hogy $\rho_1 + \rho_2 + \tilde{c_2}x \leq cx$. Világos, hogy az eljárást tovább ismételve az alsó korlátok összege alulról fogja közelíteni a minimumot. Ezt additív korlátozásnak nevezzük.

**Diszjunktív korlátozás**

**2.1. Definíció.** Egy $B \subset A$ élhalmazt *nem megengedettnek* nevezünk, ha az ACVRP egyik lehetséges megoldása sem használja $B$ összes élét, azaz

$$\sum_{(a,b)\in B} x_{ab} \leq |B| - 1.$$

Egy adott minimális nem megengedett $B \subset A$ halmazra teljesül, hogy

$$\bigvee_{(a,b)\in B} (x \in \{x \in \mathbb{R}^a \colon x_{ab} = 0\})$$

minden $x \in F$-re, ahol $F$ a lehetséges megoldások halmaza. Ekkor definiálhatunk $|B|$ darab megszorított problémát; egy ilyen problémát jelöljünk $RP^{ab}$-vel, ha az $x_{ab} = 0$ feltételt tartalmazza. Minden ilyen $RP^{ab}$-re adhatunk egy $\vartheta^{ab}$ alsó korlátot mondjuk AP segítségével úgy, hogy feltesszük, hogy $c_{ab} = \infty$, vagyis egy nagyon nagy számnak definiáljuk. A *diszjunktív alsó korlátot* $L_D = \min \vartheta^{ab} \colon (a,b) \in B$ érték kiszámításával kapjuk.

Hogyan lehet ilyen $B$ halmazt találni? Először is oldjuk meg az $AP$ relaxációt, és legyen $x^*$ az aktuális optimum. Ha $x^*$ az ACVRP-nek is megoldása, akkor $L_{AP}$ nem javítható. Ha nem, akkor választunk egy alkalmas $B$-t a következőek szerint, ami remélhetőleg javít. Ha $x_{ab} = 0$-t olyan $ab$-re követeljük meg, melyre $x^*_{ab} = 0$, akkor $\vartheta^{ab} = L_{AP}$-t kapunk, ahonnan $L_D = L_{AP}$ következne. (Hiszen erre az esetre $x^*$ volt az optimum, melynek értéke $L_{AP}$.) Ezért válasszuk $B$-t olyannak, hogy a $B$-beli $(i, j)$ élekre $x^*_{ij} = 1$ teljesüljön. Tehát $B$ legyen olyan, hogy $B \subset A^* = \{(i, j) \in A' \colon x^*_{ij} = 1\}$ és $B$ az alábbiak közül valamelyik:

1. egy kör, amely nem tartalmazza a depót,
2. egy olyan sorozata a fogyasztóknak, melyek összesített igénye meghaladja $C$-t, vagy
3. egy olyan lehetséges kör, amely fedetlenül hagyja a fogyasztók egy olyan $S$ részhalmazát, melyet $K - 1$ járművel nem lehet kiszolgálni.

Ilyen biztosan létezik, hiszen $x^*$ nem volt megoldása ACVRP-nek. Megjegyezzük, hogy $B$ különböző megválasztásai különböző alsó korlátokat adhatnak.

**ADD_DISJ** A most következő eljárás a [9] cikkből származik. Ez a módszer minden nem megengedett halmazt vizsgál, és azt választja ki, amelyik a legjobb alsó korlátot adja. Először az $AP$ relaxációt oldja meg, és a kezdeti alsó korlát $L_{AP}$ lesz, melyhez $A^*$ élhalmaz tartozik. Ezután iteratívan választunk $B$ nem megengedett halmazt $A^*$-ból, és megkeressük a diszjunktív $L_D$ alsó korlátot. Ehhez tartozik egy maradék költségfüggvény, amit a következő iterációban használunk. Így az additív korlátozásnál megismert módszer szerint a kapott $L_D$ értéket mindig az előző iterációban kapott alsó korláthoz adva kapjuk az új alsó korlátot. $A^*$ élhalmazát frissítjük úgy, hogy eltöröljük azokat az éleket, melyek nem szerepelnek az aktuális diszjunktív korlátozás optimális megoldásában. Ezt addig folytatjuk, amíg $A^*$ tartalmaz nem megengedett élhalmazt. Ezzel egy $O(n^4)$ futásidejű algoritmust kaptunk.

**Visszavezetés minimális költségű folyam feladatra** A következő módszer szintén a [9] cikkből származik. Legyen $\{S_0, S_1, \ldots, S_m\}$ a $V$ egy partíciója úgy, hogy $0 \in S_0$. Legyen $A_1 = \bigcup_{h=0}^{m} E(S_h)$ és $A_2 = A \backslash A_1$. Ekkor az $L_p$ alsó korlátot úgy szeretnénk megkapni, hogy $L_p = \vartheta_1 + \vartheta_2$, ahol $\vartheta_t$ egy alsó korlát $\sum_{(i,j) \in A_t \cap A^*} c_{ij} \quad (t = 1, 2)$ minden optimális $A^* \subset A$ ACVRP megoldásra.

Kezdetben legyen $\vartheta_1 = 0$. A $\vartheta_2$ értéket a következőképp számolhatjuk ki. Gyengítsük az ACVRP-re adott Vehicle flow modellből (1)-(4) feltételeket egyenlőtlenséggé azért, hogy az

eltörölt $A_1$-beli éleket is figyelembe tudjuk majd venni, valamint a (5)-t csak $S_0, S_1, \ldots, S_m$ halmazokra követeljük meg. Így kapjuk az alábbi IP feladatot.

$$\sum_{i \in V\colon (i,j) \in A_2} x_{ij} \leq 1 \qquad \forall j \in V \backslash \{v_0\} \tag{48}$$

$$\sum_{j \in V\colon (i,j) \in A_2} x_{ij} \leq 1 \qquad \forall i \in V \backslash \{v_0\} \tag{49}$$

$$\sum_{i \in V\colon (i,j) \in A_2} x_{ij} \leq K \qquad j = 0 \tag{50}$$

$$\sum_{j \in V\colon (i,j) \in A_2} x_{ij} \leq K \qquad i = 0 \tag{51}$$

$$\sum_{i \notin S_h} \sum_{j \in S_h} x_{ij} \geq r(V \backslash S_h) \qquad h = 0 \tag{52}$$

$$\sum_{i \notin S_h} \sum_{j \in S_h} x_{ij} \geq r(S_h) \qquad \forall h = 1, \ldots, m \tag{53}$$

$$x_{ij} \in \{0, 1\} \qquad \forall (i,j) \in A_2 \tag{54}$$

$$\vartheta = \min \sum_{(i,j) \in A_2} c_{ij} x_{ij} \tag{55}$$

Ezt hatékonyan meg tudjuk oldani, hiszen visszavezethető egy minimális költségű folyam feladatra a következők szerint.

Hozzunk létre $2(n+m)+2$ pontot úgy, hogy minden $v_i \in V$ pontot megduplázunk, így kapjuk $v_i^+$ és $v_i^-$ pontokat; minden $h = 1, \ldots, m$-re létrehozunk egy $a_h$ és egy $b_h$ pontot; és vegyünk $s$ és $t$ forrás és nyelőpontokat. Most értelmezzük az ezek közt menő éleken a költségeket, és a kapacitásokat.

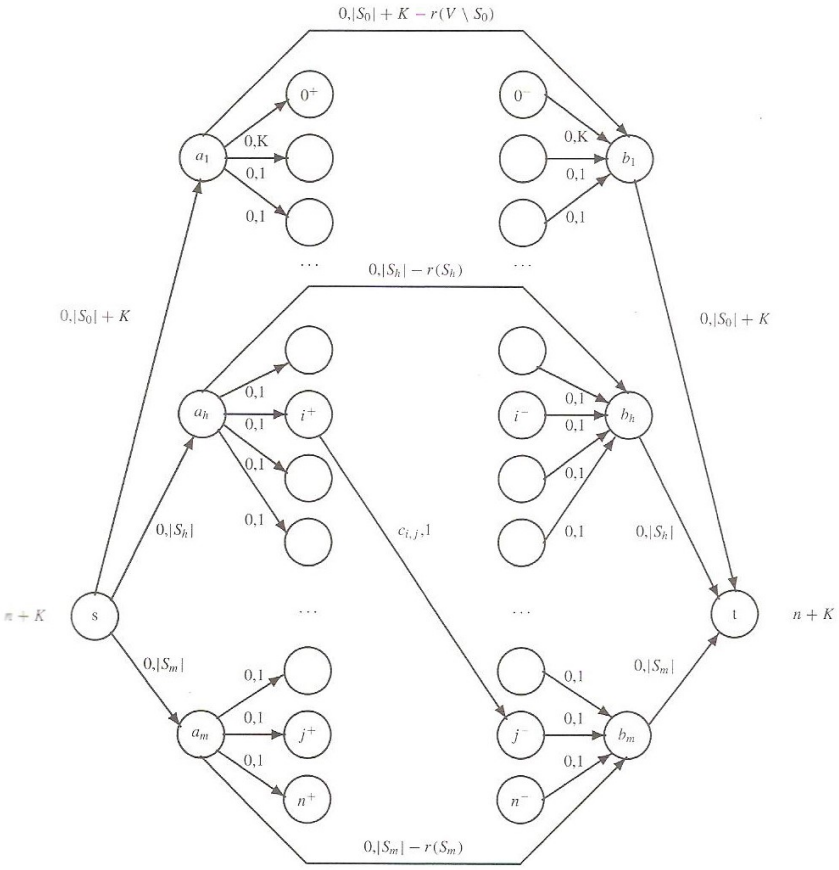
1. Minden $(v_i, v_j) \in A_2$-re legyen az $(v_i^+, v_j^-)$ él költsége $c_{ij}$ és a kapacitás $\infty$.

2. Minden $h = 0, \ldots, m$-re és minden $i \in S_h$-ra legyen az $(a_h, v_i^+)$ és $(v_i^-, b_h)$ él költsége 0. Legyen kapacitása 1, ha $i \neq 0$; és $K$, ha $i = 0$.

3. Minden $h = 0, \ldots, m$ esetén az $(a_h, b_h)$ él költsége 0. Ha $h \neq 1$, akkor legyen a kapacitása $|S_h| - r(S_h)$; ha $h = 1$, akkor legyen a kapacitás $|S_0| + K - r(V \backslash S_0)$, .

4. Minden $h = 0, \ldots, m$ esetén az $(s, a_h)$ és a $(b_h, t)$ költsége legyen 0. Ha $h \neq 1$, akkor legyen a kapacitása $|S_h|$; ha $h = 1$, akkor legyen a kapacitás $|S_0| + K$.

A gráfba csak az itt említett irányított éleket húzzuk be a következő ábra alapján. Könnyen látható, hogy így a fent említett probléma valójában egy minimális folyamkeresés ebben a gráfban.




**ADD_FLOW** [9] cikkben a fenti eljárásnak egy továbbfejlesztett változatáról is olvashatunk. A minimális folyamra visszavezető procedúrát alkalmazzuk most többször egymás után additív korlátozást használva úgy, hogy minden lépésben más $S_h$ halmazrendszert választunk. Kiindulásképp legyen $S_h = \{h\}$ minden $h \in V$-re, amit meg tudunk oldani, hiszen ez éppen az AP feladat. A következő halmazrendszert mindig úgy hozzuk létre az előzőből, hogy bizonyos halmazoknak az $S^*$ unióját vesszük úgy, hogy az unió egy nem megengedett halmaz legyen. Vagyis olyan halmaz, melyben a fogyasztók összesített igénye meghaladja a kapacitást.

De hogyan találunk ilyen sértő $S_{h_1}, S_{h_2}, \ldots, S_{h_r}$ halmazrendszert? Legyen $A_2^* = \{(i,j) \in A_2 \colon x_{ij}^* = 1\}$. A (52) és (53) feltételek miatt az $S_h$ halmazba ugyanannyi $A^*$-beli él lép be, mint amennyi kilép. Így ha vesszük a $G^* = (V, A^*)$ gráfot, akkor az $S_h$ ponthalmazok egyetlen ponttá történő összehúzásával kapott $\tilde{G}$ gráf minden összefüggő komponense egy Euler gráf lesz. Így minden összefüggő komponensre értelmezhetünk egy pontsorozatot aszerint, hogy milyen sorrendben követik egymást az Euler körben. Ekkor az $S_{h_1}, \ldots, S_{h_r}$ halmazrendszert egy $\tilde{G}$-beli $h_1, \ldots, h_r$ pontsorozatból kapjuk meg, melyek

egy Euler-körhöz tartoznak. Azaz legyen $e_1, \ldots, e_q$ egy, a $\tilde{G}$-beli Euler-körhöz tartozó pontok sorozata. Ekkor minden $i = 1, \ldots, q$-re meghatározzuk $e_t$-t, ami az a pont, melyre az $e_i, e_{i+1}, \ldots, e_q, e_1, \ldots, e_{i-1}$ sorban először teljesül, hogy $S_{e_i} \cup S_{e_{i+1}} \cup \ldots \cup S_{e_t}$-re sérül a kapacitási kritérium. Ezt $O(|A_2^*|)$ lépésben megtalálhatjuk. Az így talált halmazrendszerek közül azt választjuk ki, melyre $S^*$ elemszáma minimális. Ettől csak akkor térjünk el, ha tudunk olyan halmazrendszert választani, ami nem tartalmazza a depót.

**Lagrange relaxáció** Lagrange relaxációs eljárásokról [3] és [7] cikkekben olvashatunk. Az eredeti probléma nehézsége első sorban a CCCs vagy GSECs feltételek exponenciálisan nagy számából adódik, ezért nem tudjuk explicit belefoglalni mindet a célfüggvénybe. Ennek egy kezelési módja az, hogy kiválasztunk a CCCs vagy GSECs feltételeknek egy $\mathcal{F}$ halmazát kiindulásképp, majd az iteráció során mindig az aktuális megoldásban szereplő sértő halmazra vonatkozó feltételt adjuk hozzá az eddigiekhez. Az ilyen sértő halmazokat kereső *szeparációs* módszer a következő. Töröljük el a depóból induló éleket, majd vizsgáljuk csak az összefüggő komponensekre a megfelelő feltételeket. Ez a szeparációs módszer pontos, hiszen ha $S$ sértő halmaz, akkor létezik olyan komponense a kapott gráfnak, amely teljesen tartalmazza $S$-et. A sértő halmazra vonatkozó feltételeket hozzávesszük a problémához egy megfelelő szorzóval, és iteráljuk az eljárást, amíg elő nem fordul, hogy már nem találunk több sértő feltételt, vagy amíg el nem értük az iterációk egy előre rögzített számát.

[3] szerint egyik lehetőség a Lagrange-relaxáció végrehajtásához a következő. Legyen kezdetben $\mathcal{F}$ az üres halmaz. A szubgradiens módszer minden lépésében hozzáveszünk az eddigi feltételekhez egy sértő halmazra vonatkozó GSECs, és egy továbbit az alábbiak közül.

1. További GSECs feltétel, ha az aktuális megoldás legalább $k$ $(k \geq 2)$ túlterhelt utat tartalmaz. Az újonnan bevett feltétel vonatkozzon azon $S_1, \ldots, S_k$ halmazok uniójára, melyek a $k$ darab túlterhelt út ponthalmazai és az aktuális megoldásban sértik az eddigi kapacitási feltételeket.

2. Ha vannak a megoldásban alulterhelt utak (azaz olyan utak, melyek összesített kapacitása kisebb, mint a kapacitás), akkor belevesszük a feltételek közé az $\sum_{e \in E(S)} x_e \leq |S| - 1$ feltételt minden olyan $S$ halmaz esetén, melyre $v_0 \in S$. Ekkor megtörhetjük az aktuális megoldásban szereplő alulterhelt utat.

**Alsó korlát a Set-Partitioning modellből** [10] szerint most az első fejezetben bevezetett Set-Partitioning formula segítségével szeretnénk alsó korlátot nyerni. Vegyük a fenti ismertetett LP feladat duálisát.

$$y_0 + \sum_{i \in H_j} y_i \leq c_j \qquad \forall j = 1, \ldots, M \tag{56}$$

$$\max K y_0 + \sum_{i=1}^{n} y_i \tag{57}$$

A dualitás tételből tudjuk, hogy ennek a feladatnak az optimális megoldásának költsége megegyezik a primál feladat optimális megoldásának költségével. Így a duál feladat minden megoldása egy alsó korlátot jelent a primál feladat megoldásaira. A duál feladat optimumát [10] szerint két relaxáció segítségével additiv módon kereshetjük

Az utóbb említett módszerek eredménye látványosan jobb, mint az alap relaxációké. [1] 43. oldalán található összefoglalás szerint nehéz valóban jó összehasonlításokat végezni, mert a legtöbb szerző különböző tesztesetekre futtatta az algoritmusát. Az azonban elmondható, hogy a vizsgált tesztesetek alapján a fent említett Lagrange procedúra [3] átlagosan 2%-kal tér el az optimumtól, hasonlóan a Set-Partitioning formulából kapott alsó korláthoz. Meg kell jegyezni, hogy ezek az algoritmusok a feladatoknak gyakran az optimumát is megtalálta.

### 2.1.2. Branching módszerek

Miután ismertettünk néhány alsó korlátot kereső heurisztikát, nézzük meg, hogy milyen lehetőségeink vannak a „branching", azaz szétválasztás végrehajtására. Először vegyük az irányított esetet, majd az irányítatlant.

**Irányított eset** A következő két eljárás [2] és [9] cikkekből származnak, melyek ugyanazon az alapgondolaton alapszanak. Legyenek a döntési fa minden $v$ pontjában értelmezve az $I_v$ és $F_v$ élhalmazok. $I_v$ jelöli a kötelezően beválasztandó élek halmazát (azaz azokat az éleket, melyekről már eldöntöttük, hogy beválogatjuk), és $F_v$ legyen a tiltott élek halmaza. Kezdetben legyen $I_v$ és $F_v$ is az üres halmaz. A branch-et egy olyan $B$ élhalmazon akarjuk végrehajtani, amely nem megengedett halmaz, és nincsenek kötelezően beválasztandó ($I_v$-beli) elemei. Legyen $B = \{(a_1, b_1), (a_2, b_2), \ldots, (a_h, b_h)\} \subset A^*$, ahol $A^*$ jelöli az aktuális megoldás élhalmazát. $B$-t úgy szeretnénk megválasztani, hogy elemszáma minimális legyen azon $A^*$-beli részhalmazok között, melyek egy nem megengedett utat vagy kört határoznak meg (mint ahogy diszjunktív korlátozás esetében tárgyaltuk). Ez azért kell,

hogy a döntési fában minél kevesebb leszármazott pont legyen. Fontos megjegyezni, hogy ha additív korlátozást használunk, akkor a célfüggvény változik. Ezért, ha az egyik relaxált problémának optimális megoldása megengedett ACVRP-nek is, akkor egyáltalán nem biztos, hogy optimális is. Ezért, ha $A^*$ olyan lehetséges ACVRP megoldás, melynek költsége nagyobb, mint az aktuális alsó korlát, akkor válasszuk $B$-t egy olyan körnek, amely keresztülmegy a depón, és a lehető legkevesebb nem kötelező élt tartalmazza. A döntési fa $v$ pontjának $|B| = h$ darab leszármazott pontja lesz. Az $i$. leszármazott pont $(v_i)$ által jelölt probléma olyan, hogy kizárjuk a $B$ halmaz $i$. elemét, és belevesszük az első $i-1$ élt. Azaz $I_{v_i} = I_v \cup \{(a_1, b_1), \ldots, (a_{i-1}, b_{i-1})\}$ és $F_{v_i} = F_v \cup \{(a_i, b_i)\}$, és legyen $I_{v_1} = I_v$.

[2] szerint minden leszármazott pont legfeljebb $r$-et hagy el $B$-ből, ahol $r = \lceil \frac{d(S)}{C} \rceil$, és $S$ a $B$-beli élek által kifeszített ponthalmaz. Így legfeljebb $\binom{|B|}{r}$ leszármazott pont lesz. Értelemszerűen válasszuk meg $B$-t úgy, hogy ez az érték minél kisebb legyen.

Ezek a stratégiák a *best-bound-first* módszert alkalmazzák, ami azt jelenti, hogy először mindig azt a pontot választjuk a döntési fa még megvizsgálatlan pontjai közül, amelyhez tartozó alsó korlát a legkisebb. Így kevesebb részproblémával kell foglalkoznunk, vagyis időt és memóriát takaríthatunk meg.

**Irányítatlan eset** A *branching on arcs* (lásd [6]) módszer már meglévő részutakat próbál kiterjeszteni. Azaz olyan utakat, melyek a depóból indulnak, és valamely $i$. fogyasztónál végződnek. A döntési fa minden pontjának két leszármazott pontja van aszerint, hogy az $(i, j)$ élt használjuk-e az aktuális részút növeléséhez vagy nem, vagyis hogy $x_{ij} = 0$ vagy $x_{ij} = 1$. De hogyan válasszuk ki azt az $(i, j)$ élt, amelyen a branching történni fog? Egy alkalmas módszer [3] szerint a következő. Először keressünk egy megoldást Lagrange relaxációval. Majd ha egy részút az $i$ pontban végződik, és létezik $h$, hogy a $(i, h)$ él a Lagrange relaxációval kapott megoldás része, akkor ezt az élt választjuk a branching-hez. Ha nincsen ilyen részben rögzített út (például a döntési fa gyökér pontjában, vagy ha éppen most zártunk be egy részutat egy depóból induló éllel), akkor egy új részút megkezdéséhez válasszunk olyan fedetlen pontot, melynek az igénye a legnagyobb. Abban az esetben, amikor az egy fogyasztót tartalmazó utakat is szeretnénk vizsgálni, megengedhetjük, hogy három leszármazott pont legyen a döntési fában, hiszen ekkor $x_{0j} = 2$ is egy lehetséges választás.

[7] egy másik lehetőséget kínál. Ha nincs részben rögzített út, akkor válasszuk azt a fedetlen pontot, melynek igénye a legnagyobb, majd azt az $(i, j)$ élet válasszuk a branchingnek, amelyre $j$ az $i$-hez legközelebbi kiszolgálatlan pont. A döntési fa azon pontjában, amely esetében $(i, j)$ élt kizárjuk, alkalmazzuk a következő, *branching on customers* eljá-

rást. Ha a részút az $i$ pontban végződik, akkor vizsgáljuk a kiszolgálatlan fogyasztók egy bizonyos $T$ részhalmazát, és a döntési fában hozzunk létre $|T| + 1$ leszármazott pontot. Ezek jelentsék azt, hogy az $(i, j)$ élt beválasztjuk a megoldásba, ahol $j \in T$, valamint legyen egy olyan leszármazott pont is, ami annak az esetnek felel meg, hogy minden $(i, j)$ élt kizárunk a megoldásból.

## 2.2. Branch-and-Cut

Mivel a CVRP-nek a TSP egy speciális esete, ezért nem meglepő, hogy a TSP-re adott módszereket a CVRP-re is alkalmazni szeretnénk. Innen ered az az ötlet, hogy a *Branch-and-Cut* algoritmussal próbáljuk megtalálni a CVRP oprimumát. A következőkben a két indexű vehicle flow modelt használjuk alapul ((9)-(14)), hiszen a szimmetrikus TSP esete miatt ettől várjuk a jobb eredményeket.

A továbbiakban *Útnak* fogjuk nevezni az élek egy olyan $R$ részhalmazát, ahol $G(R)$ egy kör, amely a depót tartalmazza, és az összesített igénye nem haladja meg a kapacitást. Nevezzük *K-útnak* $K$ darab út unióját, melyekre teljesül, hogy minden fogyasztó pontosan egy darab úthoz tartozik. Ekkor minden $K$-út egy lehetséges CVRP megoldást határoz meg, valamint egy $P = \{S_1, \ldots, S_K\}$ partíciót ad a $V \backslash \{v_0\}$ pontjai között, ahol $d(S_i) \leq C$ minden $i = 1, \ldots, K$-ra teljesül. Ezt hívjuk egy lehetséges $K$-partíciónak.

### 2.2.1. A Branch-and-Cut algoritmus lényege

Szeretnénk megoldani a fenti (9)-(14) feltételek által meghatározott IP feladatot, amelyben most a kapacitási feltételeket $r(S)$ helyett $\lceil \frac{d(S)}{C} \rceil$ értékkel írjuk fel. Vegyük ennek a feladatnak az LP relaxáltját. Ha az LP feladatnak nincs túl sok feltétele, akkor egy LP-megoldó algoritmust alkalmazhatunk rá, majd az IP megoldást Branch-and-bound algoritmussal kapjuk meg. Ha a feltételek száma túl nagy ahhoz, hogy az LP feladatot meg tudjuk oldani, akkor vágósíkos algoritmust alkalmazunk. Ez a következőt jelenti. Legyen $LP(\infty)$ az IP feladat LP relaxáltja, és legyen $LP(h)$ az a feladat, amelyet úgy kaptunk, hogy $LP(\infty)$-ből elhagytunk bizonyos számú feltételt úgy, hogy $LP(h)$ már kezelhető legyen. Ezután egy bizonyos *szeparációs algoritmust* használunk. Ez eldönti, hogy $LP(h)$-nak az $x^h$ megoldása megoldja-e $LP(\infty)$-t, és ha nem oldja meg, akkor megad egy feltételt, amit $x^h$ megsértett. Ezt a feltételt $LP(h)$-hoz adjuk, és így kapjuk az $LP(h+1)$ feladatot. Ha $z_h$ jelöli az $LP(h)$ optimumát, akkor a következő teljesül:

$$z_h \leq z_{h+1} \leq z_\infty \leq z_{IP}$$

Az eljárás akkor áll le, ha IP optimumot találtunk, vagy az $LP(\infty)$ optimumát találtuk meg. Meg kell jegyeznünk, hogy a gyakorlatban a szeparációs algoritmusok nem mindig működnek tökéletesen. Ez azt jelenti, hogy előfordulhat, hogy van sértett feltétel, viszont az algoritmus ezt nem adja meg. De mivel ebben az esetben is kapunk egy érvényes alsó korlátot az IP feladatra, ezért ezeket az algoritmusokat is használni szoktuk.

Ha végül nem az IP feladat optimumát kaptuk, akkor a tört koordinátákra Branch-and-bound algoritmust használunk. Az így létrejött eljárást nevezzük összességében *Branch-and-Cut* algoritmusnak. Az eljárás előnye egyrészt, hogy a vágósíkos algoritmus általában jobb alsó korlátot ad, mint amiket az általános Branch-and-bound algoritmusoknál használtunk. Másrészt a szeparációs algoritmus „hasznot húz" a branching-ből, hiszen ekkor a döntési fa egy adott pontjában egy újabb feltételt adunk az eddigiekhez, amit a későbbiekben sem törlünk el. Ezzel gyorsítjuk a feltételek hozzáadásának a folyamatát.

Mivel a feladat nehézségét (11) feltételek száma jelenti, ezért a legtöbbször ezekből próbálunk elhagyni néhányat. Ezért definiáljuk $LP(0)$-t a következőképpen.

$$\sum_{e\in\delta(v_i)} x_e = 2 \qquad \forall v_i \in V\backslash\{v_0\} \tag{58}$$

$$\sum_{e\in\delta(v_0)} x_e = 2K \tag{59}$$

$$0 \le x_e \le 1 \qquad \forall e \in E\backslash\delta(v_0) \tag{60}$$

$$0 \le x_e \le 2 \qquad \forall e \in \delta(v_0) \tag{61}$$

$$\min cx \tag{62}$$

A braching-re két módszert szoktunk használni. Az egyik, hogy azon az $e$ élen hajtjuk végre a branch-et, amelyik koordinátája tört értékű. Ekkor az egyik alprobléma az $x_e = 0$, a másik az $x_e = 1$ feltételt fogja tartalmazni. Egy másik lehetőség, hogy választunk egy olyan $S^*$ halmazt, melyre $\lceil \frac{d(S^*)}{C} \rceil = t$, és $\sum_{e\in\delta(S^*)} x_e^*$ közel van $2t+1$-hez. Ekkor a két alprobléma közül az egyik tartalmazza, hogy $\sum_{e\in\delta(S^*)} x_e = 2t$, a másik pedig, a $\sum_{e\in\delta(S^*)} x_e \ge 2t+2$ feltételt.

Előfordulhat, hogy a Branch-and-Cut algoritmus teljesítménye nagyon gyenge. A bajt a következő esetek okozhatják.

1. Nem elég jó a szeparációs algoritmus
2. A vágósíkos fázisban túl nagy az iterációk száma
3. Az LP feladat akkorára nő, hogy kezelhetetlenné válik.

4. A branching fázisban a döntési fa túl nagy lesz, így az algoritmus nem áll le ésszerű időn belül.

Az első két probléma nem orvosolható. Az első esetben azonban nem is feltétlenül szükséges a pontos szeparációs algoritmus, és a branching fázisba akkor is átkerülhetünk, ha az LP feladatnak nem találtuk meg az optimumát. Tehát egy jó heurisztika is elégséges lehet. A harmadik probléma elkerülhető, ha az eljárás során mindig kitöröljük az inaktívvá váló feltételeket.

A Branch-and-Cut algoritmus valódi problémája a negyedik. Az eddig ismertetett procedúra esetén ugyanis ez a hiba még kisebb esetekre is nagyon könnyen előfordulhat. A probléma az, hogy az LP feladat megoldása nincs elég közel ez IP feladat megoldásához. Ezen úgy tudunk segíteni, ha az LP relaxációt szűkítjük, azaz további feltételeket veszünk az eddigiek közé. Ez azt jelenti, hogy olyan egyenlőtlenségeket kell keresnünk, melyeket minden megengedett egész megoldás kielégít. Ezek nem szükségesek az IP feladat megoldásakor, viszont az LP relaxált optimumát közelebb viheti az egészértékű feladat optimumához. Természetesen nem fogja minden bevett egyenlőtlenség, amelyet minden megoldás teljesít, közelebb vinni az LP optimumot az IP optimumhoz. A hasznos egyenlőtlenségek megtalálásához a CVRP poliéderét kell tanulmányozni. A következőben néhány egyenlőtlenséget fogunk megnézni, melyeket érdemes az eddigi feltételek közé venni.

### 2.2.2. Bevehető egyenlőtlenségek

A kapacitási feltételek esetén több lehetőségünk van az egyenlőtlenség jobb oldalának megválasztására. Minél nagyobb a jobb oldal értéke, annál erősebb az egyenlőtlenség, azonban annál nehezebb megoldani a megfelelő szeparációs problémát.

**Tört kapacitási egyenlőtlenség** A kapacitási feltételek jobb oldalának nem vesszük felső egészrészét. Így kapjuk, hogy

$$x(\delta(S)) \geq 2 \cdot \frac{d(S)}{C}.$$

Ekkor a szeparációs probléma könnyű, és polinomiális időben megoldható (ahogy azt később látni fogjuk). Viszont ez az egyenlőtlenség szinte soha nem visz közelebb minket a CVRP megoldáshoz, hiszen az egyenlőtlenség bal oldalán egész számot szeretnénk kapni a megoldás során. Azonban könnyen látható, hogy a felső egészrész elhagyásával szintén egy jó megfogalmazását kaptuk az IP feladatnak, így ennek az LP relaxáltja olyan alsó korlátot ad, amelyet polinomiális időben tudunk kiszámítani.

**Kerekített kapacitási egyenlőtlenség** A kapacitási feltétel jobb oldalát kerekítsük a közelebbi egész számhoz, és ennek a kerekített számnak vegyük a kétszeresét. Ekkor a szeparációs probléma jóval bonyolultabb lesz, mint az előző esetben, viszont még megoldható (később ezt is látni fogjuk). Ha $r(S)$ jelöli a megfelelő BPP megoldását, és $r(S) \neq \lceil \frac{d(S)}{C} \rceil$, akkor a kerekített kapacitási egyenlőtlenség erre az $S$-re nézve nem visz közelebb minket az optimális CVRP megoldáshoz.

**Gyenge kapacitási egyenlőtlenség** Ebben az esetben a kapacitási feltételek jobb oldala $2r(S)$. Mivel a BPP egy NP-nehéz probléma, ezért az ehhez kapcsolódó szeparációs probléma bonyolult. Ez az egyenlőtlenség meglepő módon nem visz minket közelebb az egészértékű megoldáshoz, hiszen nem vettük figyelembe az $S$-en kívüli fogyasztók igényeit.

**Kapacitási egyenlőtlenség** Legyen $\mathcal{P}$ a lehetséges $K$-partíciók halmaza $V\backslash\{v_0\}$-ban. Legyen minden $S \subseteq V\backslash\{v_0\}$ halmazra és minden $P$ $K$-partícióra

$$\beta(P, S) = |\{i \colon S_i \cap S \neq \emptyset\}|.$$

Ekkor $\beta$ éppen az $S$ kielégítéséhez felhasznált járművek számát adja meg. Legyen $R(S) = \min_{P \in \mathcal{P}} \beta(P, S)$, ami az $S$ kielégítéséhez szükséges járművek minimális száma. Azon $P \in \mathcal{P}$ partíciót, amelyen a minimum felvétetik, *megengedett szoros* $K$-partíciónak hívjuk $S$-re nézve. Egy $K$-út *szoros* az $S$-re nézve, ha $\delta(S)$-ből pontosan $2R(S)$ élt tartalmaz. A gyenge kapacitási egyenlőtlenség azért nem segített a megoldásban, mert előfordulhat, hogy nem létezik megengedett szoros $K$-partíció, melyre $\beta(P, S) = r(S)$, habár az $S$ igényei bepakolhatók $r(S)$ darab járműbe.

Tehát legyen az egyenlőtlenségünk jobb oldala $2R(S)$. Ez az egyenlőtlenség a definíciója miatt közelebb visz minket az optimális CVRP megoldáshoz, azonban a szeparációs probléma nehéz marad, ugyanis $r(S)$ kiszámítása egy speciális esete $R(S)$ kiszámításának.

**Általánosított kapacitási feltételek** Legyen $\mathcal{S} = \{S_1 \cup \ldots \cup S_t\}$ a $V\backslash\{v_0\}$ halmaz néhány diszjunkt részhalmazának uniója, ahol $t > 1$. Vegyük ezekre a részhalmazokra a fent kapott kapacitási egyenlőtlenségeket, és adjuk össze őket, mellyel egy érvényes egyenlőtlenséget kapunk $\mathcal{S}$-re. Ez az egyenlőtlenség dominálva van a másik $t$ darab egyenlőtlenség által, vagyis a másik $t$ egyenlőtlenség teljesüléséből ez is következik. Azonban ha meg tudjuk állapítani, hogy nem létezik olyan $K$-partíció, amely minden $S_i$-re nézve szoros ($i = 1 \ldots t$), akkor az $\mathcal{S}$-re felírt egyenlőtlenség jobb oldalát növelhetjük legalább kettővel.

Ebben az esetben ugyanis biztosan nem lehet $\mathcal{S}$ halmazt az eredeti $R(\mathcal{S})$ számú járművel kiszolgálni, ennél többet kell használnunk. Az egyenlőtlenség jobb oldala legfeljebb

$$2R(\mathcal{S}) = 2\min\left\{\sum_{i=1}^{t}\beta(P,S_i)\colon P\in\mathcal{P}\right\}$$

lehet. Ezzel egy érvényes egyenlőtlenséget kaptunk, amit nem dominálnak a kapacitási egyenlőtlenségek:

$$\sum_{i=1}^{t} x(\delta(S_i)) \geq 2R(\mathcal{S}).$$

Ezt *általánosított kapacitási egyenlőtlenségnek* hívjuk.

**Gyenge általánosított kapacitási egyenlőtlenség** Mivel $R(\cdot)$ értékét nehéz kiszámolni, ezért alkalmazhatjuk az általánosított kapacitási feltétel gyengített verzióját. Legyen $H \subseteq V\backslash\{v_0\}$ olyan, hogy tartalmazza mindegyik $S_i$-t, és tegyük fel, hogy $d(S_i) \leq C$ ahol $i = 1,\ldots,t$. Legyen $r(H \mid S_1,\ldots,S_t)$ az a BPP megoldás, melyet úgy kaptunk, hogy minden jármű kapacitása $C$, minden $u \in H\backslash\bigcup_{i=1}^{t} S_i$ pontra az igény $d(u)$ marad, viszont az $S_i$ halmazok igényét ezentúl egy darab $d(S_i)$ méretű igénynek tekintjük. Ez azt jelenti, hogy megköveteljük, hogy az $S_i$ halmaz egy jármű által legyen kiszolgálva. Ha $H = V\backslash\{v_0\}$-t választjuk, akkor a következő, *gyenge általánosított kapacitási egyenlőtlenséget* kapjuk (az egyszerűség kedvéért használjuk a $V\backslash\{v_0\} = V_0$ jelölést).

$$x(\delta(V_0)) + \sum_{i=1}^{t} x(\delta(S_i)) \geq 2t + 2r(V_0 \mid S_1, S_2, \ldots, S_t)$$

Ennek egy ekvivalens formája, ha $x(\delta(V_0)) = 2K$,

$$\sum_{i=1}^{t} x(\delta(S_i)) \geq 2t + 2(r(V_0 \mid S_1, S_2, \ldots, S_t) - K).$$

Könnyen látható, hogy ezek az egyenlőtlenségek minden lehetséges megoldásra teljesülnek, hiszen az $S_i$ halmazokba belépő élek közül legalább annyit kell beválasztani a megoldásba, amennyi a halmazok kiszolgálásához szükséges járművek számának kétszerese, és a fenti egyenlőtlenség $r(H \mid S_1,\ldots,S_t)$ definíciója miatt éppen ezt jelenti.

Ekkor a szeparációs algoritmusnak a BPP-t kell megoldania, hogy $r(H \mid S_1,\ldots,S_t)$ értéket megkapjuk. Ekkor könnyebb helyzetben vagyunk, mint $R(\cdot)$ kiszámításakor, hiszen a BPP optimális megoldása sok esetben kiszámítható.

**Keretes kapacitási egyenlőtlenség** A fentiekhez hasonlóan definiált $\mathcal{S}$ és $H$ halmazokra (ahol $H$ minden $S_i$ halmazt tartalmaz) mondjuk ki az imént felírt egyenlőtlenséget:

$$x(\delta(H)) + \sum_{i=1}^{t} x(\delta(S_i)) \geq 2t + 2r(H \mid S_1, S_2, \ldots, S_t). \tag{63}$$

Ezt *Keretes kapacitási egyenlőtlenségnek* hívjuk (bővebben lásd: [11] és [12]). Most nem bizonyítjuk, hogy ezt minden megengedett egész megoldás kielégíti, mert amint azt a későbbiekben látni fogjuk, ez a *path bin inequality* egy speciális esete, tehát ennek a bizonyítását akkor tárgyaljuk.

**Egyenlőtlenségek a TSP-ből** Az eddig tárgyalt egyenlőtlenségek a CVRP megoldások kapacitására vonatkoztak, és nem foglalkoztunk az utak struktúrájával. A következő egyenlőtlenségek viszont az utak struktúrájára fognak vonatkozni. Egy Hamilton körnek és egy $K$-útnak hasonló struktúrája van, ugyanis mindkettő lefedi a gráf összes pontját, és a depó kivételével minden pont fokszáma pontosan kettő. Így nem meglepő, hogy a szimmetrikus TSP-re (STSP) adott egyenlőtlenségeket meg szeretnénk próbálni CVRP-re is valamilyen formában alkalmazni. A következőkben be fogjuk látni, hogy minden STSP-re adott egyenlőtlenség egy bizonyos formában felírva minden $K$-útra is érvényes marad. (Lásd: [13])

**2.2. Definíció.** Egy $ax \geq a_0$ egyenlőtlenség (ahol $a$ és $x$ élszám hosszú vektorok) *szoros háromszög formájú*, ha minden különböző $i, j, k \in V$ esetén $a_{jk} \leq a_{ij} + a_{ik}$ teljesül, valamint minden $i \in V$ esetén léteznek $j(i) \in V$ és $k(i) \in V$ pontok, hogy $a_{j(i)k(i)} = a_{ij(i)} + a_{ik(i)}$. Ezt az angol *tight triangular form* elnevezés miatt röviden TT-formájúnak fogjuk hívni.

**1. Tétel.** *Minden egyenlőtlenség, amely érvényes az STSP politópra, és TT-formában van írva, érvényes lesz a CVRP politópra is.*

**Bizonyítás.** Indirekt tegyük fel, hogy létezik az STSP-re érvényes $ax \geq a_0$ egyenlőtlenség, amely TT-formájú, viszont nem érvényes a CVRP politópra. Ekkor létezik olyan $R$ $K$-út, melynek $x^R$ karakterisztikus vektorára $ax^R < a_0$ teljesül. Legyenek $(0, i)$ és $(0, j)$ olyan élek, melyekre $i$ és $j$ pontok az $R$ két különböző útjához tartoznak. Mivel teljesül a háromszög egyenlőtlenség, ezért ezt a két élt eltörölve, és $(i, j)$ élt bevéve egy, az előzőnél nem drágább megoldást kapunk, amely egy $K - 1$-út (jelöljük $R'$-vel). Ekkor talán sérül néhány kapacitási feltétel, viszont $ax^{R'} < a_0$ egyenlőtlenség teljesül. Ezt a folyamatot ismételjük addig, amíg egy $\Gamma$ Hamilton-kört nem kapunk, amire $ax^{\Gamma} < a_0$ szintén teljesülni fog. Ez azonban egy megengedett STSP megoldás, melyre $ax \geq a_0$-t feltettük, így ellentmondásra jutottunk. $\square$

Vegyük először a TSP-re adott *fésű egyenlőtlenséget.*

**2.3. Definíció.** *Fésűnek* nevezzük a gráf pontjai alkotta halmazok azon rendszerét, amely áll egy $H$ *nyélből*, és $t$ darab $T_1, \ldots, T_t$ darab *fogból*, ahol $t$ páratlan szám. Ezen kívül az alábbiak teljesülnek.

$$\begin{aligned} T_j \backslash H &\neq \emptyset && \forall 1 \leq j \leq t \\ T_j \cap H &\neq \emptyset && \forall 1 \leq j \leq t \\ T_i \cap T_j &= \emptyset && \forall 1 \leq i \leq j \leq t \\ t &\geq 3 \end{aligned}$$

Nézzük meg, hogy milyen fésű egyenlőtlenséget adhatunk meg STSP-re, amely TT-formájú.

**2. Tétel.** *A következő fésű egyenlőtlenség TT-formájú, és érvényes STSP-re:*

$$x(\delta(H)) + \sum_{i=1}^{t} x(\delta(T_i)) \geq 3t + 1.$$

**Bizonyítás.** Először belátjuk, hogy az egyenlőtlenség TT formájú. Világos, hogy a definícióban szereplő $a$ vektor szerepét most az a vektor fogja betölteni, amely minden $e$ élre megadja, hogy $H$ és $T_i$ halmazok közül ($i = 1, \ldots, t$) az $e$ hány halmaz határát lépi át, jelöljük ezt itt is $a$-val. Be szeretnénk látni, hogy $a_{jk} \leq a_{ij} + a_{ik}$ minden $i, j, k \in V$ ponthármasra. Ez nyilván teljesül, hiszen ha a $j$ pontból „át akarunk jutni" a $k$-ba, éppen $a_{jk}$ halmaz határát léptük át. Amikor $k$-ból az $i$-n keresztül „megyünk vissza" $j$-be, akkor $a_{ki} + a_{ij}$ darab halmaz határát lépjük át, aminek nyilván legalább annyinak kell lennie, mint $a_{jk}$. Az $a_{j(i)k(i)} = a_{ij(i)} + a_{ik(i)}$ egyenlőség bizonyításához válasszunk ki egy tetszőleges $i$ pontot. Mivel van legalább három fog, ezért biztosan létezik kettő, ami nem tartalmazza az $i$-t. Mivel minden fognak van $H$-n kívüli és $H$-n belüli pontja is, ezért tudunk úgy választani $j$ és $k$ pontokat, hogy $j$ az egyik, $k$ a másik $i$-t nem tartalmazó fogban van, és $(i, j)$ és $(i, k)$ élek éppen egyszer lépik át a fog határát.

Most belátjuk, hogy az egyenlőtlenség teljesül minden STSP megoldásra. Most csak azt az esetet nézzük, amikor a $\Gamma$ Hamilton-kör olyan, hogy $|\Gamma \cap \delta(T_i)| = 2$ minden $i = 1, \ldots, t$ esetén. Ekkor a Hamilton-kör minden fogba pontosan egyszer lép be, ott bejár minden pontot, majd pontosan egyszer kilép. Ekkor legalább egyszer át kellett, hogy lépje $H$ határát, és mivel minden fog esetén egyszer át kell lépni $H$ határát, ezért $|\Gamma \cap \delta(H)| \geq t$. Mivel $t$ páratlan, és egy kör minden halmaz határát páros sokszor lépi át, ezért $|\Gamma \cap \delta(H)| \geq$

$t+1$. Ha ezt az egyenlőtlenséget összeadjuk azzal, hogy $\sum_{i=i}^{t} |\Gamma \cap \delta(T_i)| = 2t$, ami a feltevésünkből következik, akkor a keresett egyenlőtlenséget kapjuk. $\square$

Nézzük meg, hogy a kapott egyenlőtlenség közelebb visz-e minket az egészértékű megoldáshoz. Ehhez a következőket figyelhetjük meg.

1. Ha a nyél nem tartalmazza a depót, és $R(H) \geq \frac{t+1}{2}$, akkor a kapacitási feltételek magukba foglalják a fésű egyenlőtlenséget. Ez könnyen látszik, ha a jobb oldalt felbontjuk: $3t+1 = 2(t+\frac{t+1}{2})$. Mivel a $t$ darab fog kiszolgálásához mindenképp kell legalább egy jármű, és $H$ kiszolgálásához legalább $\frac{t+1}{2}$ darab jármű kell, ezért ez a feltétel nem szűkíti az eredeti feladatot. Ugyanez teljesül, ha a nyél tartalmazza a depót, és $R(V\backslash H) \geq \frac{t+1}{2}$.

2. Ha minden fog kiszolgálásához elég egy jármű, de nincs kettő, amelyet ugyanaz a jármű szolgál ki, akkor nem létezik $K$-út, amire a fésű egyenlőtlenség szoros. Hiszen ekkor minden fogat pontosan egy jármű szolgál ki, tehát minden fog esetén az adott út kétszer átlépi $H$ határát, mivel minden $T_i$-nek van $H$-n belüli, és $H$-n kívüli pontja. Ekkor $H$ határát legalább $2t$-szer léptük át, így az egyenlőtlenség nem lehet szoros.

3. Ha egy $T_i$ kiszolgálásához több, mint egy jármű szükséges (vagy ha tartalmazza a depót, és a komplementerének kiszolgálásához kell több, mint egy jármű), akkor Minden $K$-útnak lesz legalább négy éle $\delta(T_i)$-ben, így a fésű egyenlőtlenség dominálva van más egyenlőtlenségek által.

Ezekből látszik, hogy a fésű egyenlőtlenséget erősítenünk kell. Ehhez a kapacitási feltételeket fogjuk felhasználni. A következőkben az egyszerűség kedvéért használjuk $R(S)$ helyett $r(S)$ vagy $\lceil \frac{d(S)}{C} \rceil$ értéket. Ezen kívül, ha $S \subset V$, $T \subset V$ és $S \cap T = \emptyset$, akkor jelöljük $(S\colon T)$-vel az éleknek azt a halmazát, melyek egyik végpontja $S$-ben, a másik pedig $T$-ben van.

Ha egy $T_i$ fog pontjait a szükséges járművek minimális számával szolgáljuk ki, akkor legalább egy élt felhasználunk $(T_i\backslash H\colon T_i \cap H)$-ból. Legyen most $H, T_1, \ldots, T_t$ olyan fésű, hogy egyik fog sem tartalmazza a depót, és

$$r(T_i\backslash H) + r(T_i \cap H) > r(T_i).$$

Ekkor [14] szerint belátható, hogy a következő teljesül.

$$x(\delta(H)) + \sum_{i=1}^{t} x(\delta(T_i)) \geq t+1+\sum_{i=1}^{t} r(T_i). \tag{64}$$

Ezzel a fésű egyenlőtlenség egy általánosítását kapjuk.

Most tegyük fel, hogy az egyik fog tartalmazza a depót, legyen ez most $T_1$. Mivel a fésű egyenlőtlenség akkor is igaz, ha $H$ helyett $H$ komplementerét vesszük, így feltehetjük, hogy $v_0 \in T_1 \backslash H$. Ezen kívül tegyük fel, hogy a többi fogra $r(T_i) = 1$. Így a fésű egyenlőtlenség jobb oldala felírható $t + 1 + 2(t - 1 + r(V \backslash T_1)) = 3t - 1 + 2r(V \backslash T_1)$ alakban.

Eddig a jobb oldal változtatásával akartuk a fésű egyenlőtlenséget erősíteni. A következő egyenlőtlenséghez redukáljuk a VRP-t a TSP-re. Ezt a következőképp tehetjük meg. A depó pontot lecseréljük $K$ darab depó pontra, és jelöljük a depók halmazát $D$-vel. Azon élek költsége, melyeknek mindkét végpontja $D$-ben van, legyen nulla. A többi élköltséget értelemszerűen választjuk, hogy az új élek költsége megfeleljen az eredeti gráf élköltségeinek. Ekkor egy $K$-út az eredeti gráfban most egy H-kört jelent az új gráfban. Értelmezzük erre a gráfra a fésű egyenlőtlenséget.

**2.4. Definíció.** *Fésűnek* nevezzük a gráf pontjai alkotta halmazok azon rendszerét, amely áll egy $H$ *nyélből*, és $t$ darab $T_1, \ldots, T_t$ darab *fogból*, ahol $t$ páratlan szám. Ezen kívül az alábbiak teljesülnek.

$$\begin{aligned}
T_j \backslash H &\neq \emptyset && \forall 1 \leq j \leq t \\
T_j \cap H &\neq \emptyset && \forall 1 \leq j \leq t \\
T_i \cap T_j &= \emptyset && \forall 1 \leq i < j \leq r \\
T_i \cap T_j &= \{v_0\} && \forall r+1 \leq i < j \leq t \\
t &\geq 3 \\
0 \leq r &\leq t
\end{aligned}$$

Tegyük fel, hogy a depót tartalmazó fogakon kívül lévő fogyasztók kiszolgálásához mind a $K$ darab járműre szükségünk van, azaz $R(V \backslash T_j) = K$ minden $j = r + 1, \ldots, t$ esetén. Ekkor a következő egyenlőtlenség teljesül minden megoldásra

$$x(\delta(H)) + \sum_{i=1}^{t} x(\delta(T_i)) \geq t + 1 + 2r + 2K(t - r).$$

Ezt most nem bizonyítjuk precízen, de adunk rá egy intuíción alapuló indoklást a következőképp. Ha $T_j$ olyan, hogy tartalmazza a depót, és határát a megoldás legfeljebb $2K$-szor lépi át, akkor használni kellett legalább egy $(H \cap T_j : T_j \backslash H)$-beli élet. Hiszen ekkor minden út legfeljebb egyszer lépi át $T_j$ határát, ezért attól függően, hogy hol nincs a depó, $T_j \cap H$ vagy $T_j \backslash H$-beli pontok meglátogatásához mindenképpen használni kell egy ilyen élt. Tehát $x(\delta(H))$ alsó becslésére $t + 1$ továbbra is érvényes marad. Ezen kívül, mivel

$\sum_{i=1}^{t} r(T_i) = \sum_{i=1}^{r} r(T_i) + \sum_{i=r+1}^{t} r(T_i) \geq r + K(t-r)$, ezért ezt (64)-re használva az egyenlőtlenség teljesül.

**Bin Packing és STSP kombinálása** Most kombinálni fogjuk az eddig kapott eredményeket azért, hogy olyan egyenlőtlenséget kapjunk, ami az út szerkezetére vonatkozó feltételeket is, és a kapacitási feltételeket is szigorítja (Lásd: [12] és [11]). Ehhez vezessük be a következő definíciót.

**2.5. Definíció.** *Path-bin-support*-nak nevezzük a gráf pontjai alkotta halmazok azon rendszerét, amely áll egy $H$ *nyélből*, $t$ darab $T_1, \ldots, T_t$ darab *fogból*, és $s$ darab $S_1, \ldots, S_s$ *támasztékból*, melyekre a következők teljesülnek. (Az egyszerűség kedvéért jelöljük $T_{t+1}$-gyel $S_i$-t.)

$$\begin{aligned} d(T_j) &\leq C \qquad \forall 1 \leq j \leq t+s \\ S_j &\subset H \qquad \forall 1 \leq j \leq s \\ T_j \cap H &\neq \emptyset \qquad \forall 1 \leq j \leq t \\ T_j \backslash H &\neq \emptyset \qquad \forall 1 \leq j \leq t \\ T_i \cap T_j &= \emptyset \qquad \forall 1 \leq i < j \leq t+s \\ t+s &\geq 1 \end{aligned}$$

Legyen $I$ az áruk halmaza, és legyen minden $T_j$-re $(1 \leq j \leq t)$ és minden $S_j$-re $(1 \leq j \leq s)$ egy darab áru értelmezve, melynek mérete $d(T_j)$ és $d(S_j)$. A többi $v \in H \backslash \bigcup_{j=1}^{t+s} T_j$ pontra legyen az áru mérete $d_v$. A tároló (jármű) kapacitása továbbra is $C$. Legyen $r'(H \mid T_1, \ldots, T_{t+s})$ az a szám, ahány jármű kell legalább, hogy $I$ elemeit elhelyezzük feltéve, hogy egy járműben csak annyi árut tudunk legfeljebb elhelyezni, ami legfeljebb kettő fognak felel meg. Definiáljuk még a $H$*-út* fogalmát, ami egy összefüggő út $K \cap E(H)$-ban. Könnyen látszik, hogy $\delta(H)$-nak a $K$-úthoz tartozó éleinek száma megegyezik a $K$-út beli $H$-utak számának kétszeresével.

A célunk az, hogy meghatározzuk a $H$-utak minimális számát minden $K$-útra úgy, hogy egy fog vagy egy támaszték egy jármű által legyen kiszolgálva. Ekkor minden $K$-út pontosan kétszer lépi át bármely támaszték vagy fog határát. Minden $H$-út legfeljebb kettő fogat tartalmazhat, és minden $H$-út egy lehetséges tartálynak (járműnek) felel meg. A kapcsolatot a $H$-utak és a bin packing probléma között a következő tétel szemlélteti.

**3. Tétel.** *Ha $x$ olyan megoldása CVRP-nek, melyre*

$$x(\delta(T_j)) = 2 \qquad \forall 1 \leq j \leq t+s,$$

*akkor a* $x(\delta(H)) \geq 2r'(H \mid T_1, \ldots, T_{t+s})$ *teljesül, ahol* $r'(H \mid T_1, \ldots, T_{t+s})$ *a megfelelő BPP megoldás.*

**Bizonyítás.** Legyen $\Theta$ egy $K$-út, amely a $\delta(H)$ minimális számát tartalmazza azon $K$-utak között, melyek a feltételnek megfelelnek. Legyen $F$ a $\Theta$ azon éleinek a halmaza, melyek mindkét végpontja $H$-ban van. $G' = (H, F)$ összefüggő komponensei éppen a fele $|\Theta \cap \delta(H)|$-nak. Be szeretnénk látni, hogy az ilyen összefüggő komponensek száma éppen a BPP megoldása, ahonnan az állítás adódik. Ehhez tegyük fel indirekt, hogy egy ilyen összefüggő komponens három fogba metsz bele, legyenek ezek rendre $T_i, T_j, T_p$. Mivel a $H$-út $H$-n belül halad, ezért $T_j$-t (amelyikbe másodszor metsz bele) nem tudja teljesen kiszolgálni, tehát $\delta(T_j)$-nek kell még lennie olyan éle, ami beletartozik a megoldásba, azaz $|\Theta \cap \delta(T_j)| \geq 4$, ami ellentmond a feltételeknek. □

A BPP feladatról tudjuk, hogy ha egy árut kettévágunk, akkor az optimális megoldás legfeljebb eggyel csökkenhet csak. Ezt fogjuk most kihasználni. Ha $T_j$ árut egy tartályba teszünk bele, akkor egy járművel szolgáltuk ki, ekkor $x(\delta(T_j)) = 2$. Ha az árut két részre vágjuk, akkor megengedjük, hogy $T_j$-t két jármű szolgálja ki, tehát $x(\delta(T_j)) = 4$. Így ha $\sum_{j=1}^{t+s} x(\delta(T_j))$ páros szám, akkor mindig, amikor $\sum_{j=1}^{t+s} (x(\delta(T_j)) - 2)$-t kettővel növeljük, akkor a $H$-utak száma nem csökkenthető egynél többel. Így kapjuk a következő tételt.

**4. Tétel.** *Minden path-bin support* $(H, T_1, \ldots, T_t, S_1, \ldots, Ss)$*-re a megfelelő path-bin egyenlőtlenség*

$$x(\delta(H)) \geq 2r'(H \mid T_1, \ldots, T_{t+s}) - \sum_{j=1}^{t+s} (x(\delta(T_j)) - 2)$$

*teljesül a CVRP politópra.*

Ezzel a keretes kapacitási egyenlőtlenség (63) teljesülését is beláttuk, hiszen az ennek egy speciális esete, ha nincs egy fog sem, csak az $S_i$ támasztékok.

$r'(H \mid T_1, \ldots, T_t, S_1, \ldots, Ss)$ meghatározása nehezebb mint $r(H)$ esetében. Kis $H$-ra pontosan meghatározható, viszont nagy $H$-ra csak alsó korlátot tudunk adni. Két triviális alsó korlát a:

$$\left\lceil \frac{t}{2} \right\rceil \quad \text{és} \quad \left\lceil \frac{d(H \cup (\bigcup_{j=1}^{t+s} T_j))}{C} \right\rceil.$$

Ha $x(\delta(T_j)) \geq 2$ minden lehetséges CVRP megoldásra, akkor egy $T_j$ vagy $S_j$ halmazt akkor veszünk bele a path-bin egyenlőtlenségbe, ha az ő eltávolítása csökkenti a BPP megoldásának értékét. Különben a $T_j$ nélküli egyenlőtlenség dominálja a $T_j$-t tartalmazó egyenlőtlenséget.

### 2.2.3. Szeparációs módszerek

A továbbiakban tegyük fel, hogy $\bar{x}$ olyan tört megoldás, amely teljesíti a (9),(10) fokszám feltételeket, valamint $0 \leq \bar{x}_e \leq 1$ teljesül minden élre, ami nem a depóból indul, és $0 \leq \bar{x}_e \leq 2$ teljesül minden depóból induló élre. (Vagyis az eredeti IP feladatból elhagytuk a kapacitási feltételeket és az egészértékűségi feltételt.) A célunk most az, hogy olyan eljárásokat keressünk, amelyek találnak a fent leírt egyenlőtlenségek közül olyat, ami érvényes CVRP-re, de megsérül $\bar{x}$ által.

**Tört kapacitási feltételek** Most belátjuk, hogy létezik a tört kapacitási egyenlőtlenségre szeparációs algoritmus, amely polinomiális idejű. (Forrás: [15]) Legyen $G_{\bar{x}}$ súlyozott gráf, amely úgy jön létre az eredetiből, hogy csak $\bar{x}$ pozitív koordinátáihoz tartozó éleket vesszük bele a megfelelő költségekkel, valamint a depóból induló éleket. Legyen $G'_{\bar{x}}$ olyan gráf, melyet $G_{\bar{x}}$-ból kaptunk úgy, hogy a depóból induló élekhez tartozó koordinátákat lecseréltük $\bar{x_e} - \frac{d_i}{C}$ értékre minden $e = (0, i)$ él esetén. Könnyen látszik, hogy ha a minimális vágás $G'_{\bar{x}}$-n negatív értékű, akkor ennek a vágásnak a depót nem tartalmazó része egy kapacitási feltételt sértő halmaz lesz. Ez abból látszik, hogy egy ilyen $S$ halmaz esetén $x(\delta(S)) - 2\frac{d(S)}{C} \geq 0$ nem teljesül. Sajnos a standard minimális vágást kereső algoritmusaink nem működnek, mert a gráf negatív éleket is tartalmaz. Legyen $G''_{\bar{x}}$ az a gráf, melyet $G_{\bar{x}}$-ből kapunk egy $n+1$. pont hozzávételével, melyet a $v_1, v_2, \ldots, v_n$ pontokkal kötünk össze. Azon élek súlya, melyek a $G''_{\bar{x}}$ gráf $\{v_1, \ldots, v_n\}$ pontjait kötik össze, maradjon ugyanaz, mint $G_{\bar{x}}$-ben és minden $i \in \{v_1, \ldots, v_n\}$ esetén legyen $(0, i)$ illetve $(i, n+1)$ él súlya $\max\{0; \bar{x}_{0i} - \frac{d_i}{C}\}$ illetve $\max\{0; \frac{d_i}{C} - \bar{x}_{0i}\}$. Ekkor polinomiális időben találhatunk $v_0$ és $v_{n+0}$-t szétválasztó minimális vágást. Legyen $F''$ egy ilyen minimális vágás $G''_{\bar{x}}$-ben. Belátható, hogy minden ilyen $F''$-höz található $F'$ minimális vágás $G'_{\bar{x}}$-ben, melyet úgy kapunk $F''$-ből, hogy kitöröljük a $v_{n+1}$-ből induló éleket. $F''$ súlya ekkor $Z = \sum_{i=1}^{n} \max\{0, \frac{d_i}{C} - \bar{x}_{(0,1)}\}$-vel több, mint $F'$ súlya. Így ha $F''$ súlya szigorúan kisebb $Z$-nél, akkor az $n+1$-et nem tartalmazó ponthalmaz sértő halmazt eredményez. Ha $F''$ súlya nagyobb $Z$-nél, de kisebb $Z+2$-nél, akkor a tört kapacitási egyenlőtlenség teljesül, de a kerekített kapacitási egyenlőtlenség sérülhet. Ha a $F''$ súlya nagyobb, mint $Z$, akkor egyik kapacitási egyenlőtlenség sem sérül. Így a következő eljárást alkalmazhatjuk. Rendezzük a $v_0 - v_{n+1}$ vágásokat növekvő sorrendbe, és addig vizsgáljuk őket, amíg nem találtunk egy legalább $Z+2$ súlyú vágást. Ekkor ha rögzített $\epsilon$ számú sértést keresünk, akkor $n$-ben polinomiális algoritmust kapunk. Tehát minden új vágást polinomiális időben találhatunk meg, így a szeparációs probléma polinomiális.

**Mohó kerekített kapacitási heurisztika** A következőkben heurisztikus szeparációs algoritmusokat fogunk megnézni az általánosított kapacitási egyenlőtlenségre. Először figyeljük meg a következőt. Ha adott egy $\bar{x}$ megoldás, és egy $S \subset V$ halmaz, és szeretnénk azt a $v$ pontot belevenni $S$-be, amely esetén $\bar{x}(\delta(S \cup \{v\}))$ minimális, akkor értelemszerűen azt a $v$-t kell választani, melyre $\bar{x}(\{v\} : S)$ maximális. Azt a módszert, amikor kiindulunk egy $S_0$ halmazból, és mindig az itt leírt $v$-t hozzávéve növeljük a halmazt, *max-back order* eljárásnak nevezzük.

Legyen $S_0$ olyan halmaz hogy $\bar{x}(\delta(S_0)) = 2$. Ez lehet például egy pont, vagy egy maximális hosszú út pontjainak halmaza, amely olyan élekben végződik, melyekre $\bar{x}$ értéke 1. Alkalmazzuk a max-back order eljárást addig, amíg el nem érjük $V_0$-t. Minden lépésben leellenőrizzük, hogy a megfelelő kerekített kapacitási feltétel sérült-e. Az $\bar{x}(\delta(S))$ értéke a fokszám feltételek miatt könnyen nyomon követhető, hiszen mindig csak az $S$-be beveendő $v$-re az $\bar{x}(\{v\} : S)$ értéket kell ismerni, hiszen tudjuk, hogy a $v$-ből induló éleken az $\bar{x}$ értéke összesen kettő.

**Heurisztika a gyenge általánosított kapacitási feltételekre** A heurisztika azzal kezdődik, hogy keresünk $V_0$ egy partícióját, amely remélhetőleg maximálisan sok olyan halmazból áll, melyekből pontosan kettő darab $\bar{x}$-beli él lép ki. Belátható, hogy ilyen partíciót polinomiális időben tudunk találni (lásd: [16]). Az algoritmus minden lépésében leellenőrizi, hogy az aktuális partíció sérti-e az általánosított kapacitási feltételeket. Ehhez a bin packing problémát kell megoldani, aminek az optimuma az eset méretétől függően elfogadható időn belül megtalálható. Ha a partíció nem sérti a feltételt, akkor $S_i$ és $S_j$ halmazokat összeolvasztjuk, ahol $S_i$ és $S_j$ azok a halmazok, melyre $\bar{x}(S_r : S_s)$ értéke maximális.

### 2.2.4. Branching módszerek

A Branch-and-cut algoritmus utolsó fázisa az, amikor a fenti módszerekkel kapott tört megoldásból valamilyen branching módszerrel IP megoldást próbálunk találni. Ilyenkor a legfontosabb kérdés az, hogy mely éleken történjen a branch.

Az STSP-re egy olyan általános szabályt szoktak használni, hogy azt az $x_e$ koordinátát választjuk, amelyik az $\frac{1}{2}$ egy kicsi környezetébe esik. Ha több ilyen van, akkor válasszuk azt az élt, amelyik költsége a legnagyobb. Ekkor azonban felmerül a kérdés, hogy biztosan jó döntés-e éppen az $\frac{1}{2}$-et választani erre a célra. A CVRP esetében ugyanis figyeljük meg, hogy attól függően, hogy egy koordinátát nullára vagy egyre állítunk, különböző értékű

információt kaphatunk a megoldásról. Amikor kizárunk egy élt a megoldásból ($x_e = 0$-t követeljük meg), akkor az csak azt jelenti, hogy a továbbiakban eggyel kevesebb élt használhatunk az $O(n^2)$ él közül. Ha viszont egy élre $x_e = 1$-et követeljük meg, akkor egy élt sikerült már a megoldásban rögzíteni az $n + K$ él közül, amelyik a megoldásban szerepelni fog. Innen jön az ötlet, hogy válasszunk olyan koordinátát a branching-hez, amely $0,75$ egy kis környezetében van. Ez a kísérletek szerint STSP esetében nem volt sikeres, viszont CVRP-ben még reménykedhetünk a jó eredményekben.

Egy másik ötlet a megfelelő él kiválasztásához az, hogy a részben rögzített utakat vizsgáljuk, és olyan élt választunk a branching-hez, hogy egy ilyen részben kész utat folytatni tudjuk. Ez azért lehet eredményes, mert ilyenkor kizárhatjuk a megoldásból az összes élt, amely az út egy belső pontjából indult, valamint azokat az éleket, melyek másik végpontjához egy olyan fogyasztó tartozik, amely igénye túl nagy ahhoz, hogy belevegyük az útba.

Ahhoz, hogy jól válasszunk élt, használhatunk LP-tesztelést is (lásd [17]). Ez azt jelenti, hogy egy $e$ él esetén megoldjuk mindkét LP-t, amit úgy kaptuk, hogy $x_e$ értékét nullára vagy egyre állítottuk. Ekkor kaptunk két célértéket, legyen ez $z_1$ és $z_2$. Ezután megjegyezzük a $z_e = \min\{z_1, z_2\}$ értéket. Ezt megcsináljuk minden $e$ élre, és végül azt az élt választjuk, melyre $z_e$ a legnagyobb. Azonnal látszik, hogy ez a módszer rendkívül időigényes.

A *branching az egyenlőtlenségeken* (lásd: [19]) módszer a halmazhatárokon lévő élekkel foglalkozik. Ez STSP esetében úgy történik, hogy ha egy $S$ ponthalmazra $\bar{x}(\delta(S)) \approx 2t+1$, akkor a problémát úgy bontjuk két részre, hogy az egyik esetben $x(\delta(S)) \leq 2t$, és a másikban $x(\delta(S)) \geq 2t + 2$. Mivel ez jó eredményeket adott STSP-re, ezért remélhetjük, hogy CVRP esetén is hasznos lesz (lásd: [11], [18]). Ekkor a legérdekesebb szituáció, ha egy $S$ halmazra $\bar{x}(\delta(S)) \approx 3$ teljesül. Ekkor $x(\delta(S)) = 2$ és $x(\delta(S)) \geq 4$ eseteink lesznek, melyek a fentiekhez hasonlóan aszimmetrikus jelentéssel bírnak. $x(\delta(S)) = 2$ választás esetén rögzítettük azt, hogy $S$-et egy járművel szolgáljuk ki. Ha $S$-et összehúzzuk egy ponttá, melynek igénye a benne lévő igények összege, és ez az igény elég nagy, akkor egy utat szinte már meg is tudtunk határozni a $K$-ból. Ebben az esetben a megoldás folyamata sokkal rövidebb lesz. Így nem meglepő, hogy a $2,75 \leq x(\delta(S)) \leq 3$ feltétel gyorsabb algoritmust eredményez, mint amikor ez az érték három felett van. Az [1] 77. oldalán lévő összehasonlítások alapján a következő három stratégia a legeredményesebb.

1. Olyan $S$ halmazt választunk, melyre $2,75 \leq x(\delta(S)) \leq 3$ és $d(S)$ maximális.

2. Olyan $S$ halmazt választunk, melyre $2,75 \leq x(\delta(S)) \leq 3$ és $S$ távolsága a depótól

maximális. (Ezalatt azt értjük, hogy vesszük $S$ két olyan pontját, melyek legközelebb vannak a depóhoz, és ezeket a távolságokat összeadjuk.)

3. Olyan $S$ halmazt választunk, melyre $2,75 \leq x(\delta(S)) \leq 3$, és a lehető legtöbb *szuperpontot* tartalmazza. Szuperpont alatt egy olyan halmazt értünk, ami az aktuális megoldásban olyan, hogy a határán éppen kettő megoldásban szereplő él megy át.

# 3. Felülről közelítő heurisztikák

Ebben a fejezetben olyan heurisztikákat fogunk megnézni, melyek megengedett CVRP megoldást eredményeznek, de ezek nem feltétlenül optimálisak. Az ilyen heurisztikákat két osztályba sorolhatjuk. A *klasszikus heurisztikák* az 1960-1990 években születtek. Ezek viszonylag jó eredményeket adnak, és a futási idejük sem túl nagy. Ezen kívül gyakran ki lehet terjeszteni őket úgy, hogy a valóságban előforduló esetekre is alkalmazni lehessen őket. A másik osztály a *metaheurisztikák*, melyek 1990 után születtek, és mélyebb kutatási munkát igényeltek. Ezekben a módszerekben szofisztikáltabb algoritmusokat használnak, és több heurisztikát is kombinálnak. Az így kapott megoldások sokkal jobb minőségűek, mint a klasszikus esetben, viszont az algoritmusok futási ideje is több. Emellett ezek a procedúrák jobban függenek a felépített kontextustól, így nehezebb őket valóságban előforduló esetekre alkalmazni.

## 3.1. Klasszikus heurisztikák

Először a klasszikus heurisztikákat nézzük meg. A konstruktív és a kétfázisú módszerek egy megoldás megkonstruálásával foglalkoznak, míg a javító módszerek már meglévő megoldásokból próbálnak mindig jobbat létrehozni.

### 3.1.1. Konstruktív módszerek

**Clarke és Wright Savings Algoritmusa** [20] A heurisztika azon az eljáráson alapszik, hogy ha egy $(0, \ldots, i, 0)$ és egy $(0, \ldots, j, 0)$ út összekapcsolása lehetséges, akkor összekapcsoljuk őket, így kapjuk a $(0, \ldots, i, j, \ldots, 0)$ utat. Nézzük ezt meg precízebben. Legyen $s_{ij} = c_{i0} + c_{0j} - c_{ij}$. Ez az az érték, amellyel kiadásaink változnak az $(i, j)$ él bevételével és az $(i, 0)$ és $(0, j)$ élek törlésével, azaz ez lesz a *megtakarításunk* (*saving*). Természetesen ennél az algoritmusnál a felhasznált járművek száma folyamatosan változik. Nézzük meg az algoritmus lépéseit.

1. lépés (Megtakarítások kiszámítása) Kiszámítjuk $s_{ij}$-ket minden $i, j = 1, \ldots, n$-re, ahol $i \neq j$. Létrehozunk $n$ darab $(0, i, 0)$ utat minden $i = 1, \ldots, n$-re. Az $s_{ij}$-ket csökkenő sorrendbe rendezzük.

2. lépés (Párhuzamos verzió, legjobb lehetséges összefűzés) Az $s_{ij}$-k listájának első elemével kezdünk, és sorban haladunk. Ha $s_{ij}$-re létezik két út, hogy az egyik tartalmazza $(0, j)$-t, és a másik tartalmazza $(i, 0)$-t, és összekapcsolható, akkor összekapcsoljuk.

2. lépés (Szekvenciális verzió, utak kiterjesztése) Sorban veszünk minden $(0, i, \ldots, j, 0)$ utat, és meghatározzuk az első $s_{ki}$-t vagy $s_{jl}$-et, ahol $k$ illetve $l$ olyan, hogy az aktuális út összekapcsolható egy másikkal, amely tartalmazza $(k, 0)$ vagy $(0, l)$-et. Ekkor a két megfelelő utat összekapcsoljuk, majd erre az aktuális útra addig ismételjük az eljárást, amíg több út már nem kapcsolható hozzá. Ekkor a következő útra térünk át. Ezt addig folytatjuk, amíg van lehetséges összekapcsolás.

Itt a párhuzamos eljárás és szekvenciális eljárás közül kell választani. A gyakorlat azt mutatja, hogy a párhuzamos verzió egyértelműen jobb a megoldás minőségét, és a futási időt tekintve.

**A Clark és Wright algoritmus erősítése** (Lásd: [21] és [22]) Az eredeti módszer csak az algoritmus elején ad jó utakat, majd később egyre kevésbé érdekes utakkal foglalkozik, ezért szeretnénk erősíteni a módszert. Legyen $s_{ij} = c_{i0} + c_{0j} - \lambda c_{ij}$ ahol $\lambda$-t *route shape paraméternek* nevezzük. Az eljárás legidőigényesebb része a megtakarítások kiszámolása és rendezése, ezen kívül még a tárolásuk is memóriaigényes. Ezen különböző adatstruktúrával és rendezési módszerekkel segíthetünk, mint például a quicksort, az iteratív kupacrendezés, vagy az iteratív maximumkeresés. Egy másik módszer lehet az, hogy hash-függvényekkel minden útra számon tartjuk a belső pontjait.

**Párosításon alapuló saving algoritmus** (Lásd: [23] és [24]) A következőkben a megtakarítás $s_{pq}$ értékét a $p$ és $q$ utak összekapcsolása esetén úgy számítjuk ki, hogy bizonyos ponthalmazokon a TSP optimális megoldását használjuk fel a következőképp.

$$s_{pq} = t(S_p) + t(S_q) - t(S_p \cup S_q),$$

ahol $S_k$ jelöli a $k$ út ponthalmazát, és $t(S_k)$ az optimális TSP megoldást $S_k$-n. Ekkor $S_k$ ponthalmazokra egy maximális párosítást kereső feladatot kapunk, ahol $s_{pq}$ az élsúlyok. A kapott párosítás azt adja meg, hogy mely utakat kapcsoljuk össze. Ezt erősíthetjük

még úgy, hogy előnyben részesítjük azoknak az utaknak a bekapcsolását valahová, melyek igénye messze $C$ alatt van, vagy az út menetideje messze kisebb a megengedettnél.

**Szekvenciális beszúró heurisztikák** Először nézzük meg Mole és Jameson [25] heurisztikáját, amely egyszerre egy út kiterjesztésével foglalkozik. Használjunk két paramétert, $\lambda$-t és $\mu$-t, illetve vezessük be a következő jelöléseket:

$$\alpha(i,j,k) = c_{ik} + c_{kj} - \lambda c_{ij}$$
$$\beta(i,j,k) = \mu c_{0k} - \alpha(i,j,k)$$

Az algoritmus a következőképp néz ki.

1. lépés (Utak inicializálása) Ha minden pont már tartozik egy úthoz, STOP. Különben konstruálunk egy $(0,k,0)$ utat, ahol $k$ még fedetlen pont.

2.lépés (Következő pont) Minden fedetlen $k$ pontra kiszámítjuk a lehetséges beszúrás költségét, ami $\alpha^*(i_k,k,j_k) = \min_{r,s}\{\alpha(r,k,s)\}$ minden szomszédos $r$ és $s$ pontokra az aktuális útban. Ha egy beszúrás sem lehetséges: GO TO STEP 1. Különben beszúrjuk az útba $k^*$-ot, ahol $k^*$ az a pont, amelyre $\beta(i_{k^*},k^*,j_{k^*}) = \max_k\{\beta(i_k,k,j_k)\}$ minden fedetlen $k$-ra, amely beszúrható.

3. lépés (Út optimalizálása) Az aktuális utat optimalizáljuk 3-opt eljárással, majd GO TO STEP 2.

A paraméterek különböző megválasztásával variálhatjuk az algoritmust. Például $\lambda = 1$ és $\mu = 0$ esetén az algoritmus azt a pontot szúrja be, amely esetén legkevésbé nő az extra távolság. Ha $\lambda = \mu = 0$, akkor azt a pontot szúrja be az algoritmus, amely esetén legkisebb a két szomszédja közötti távolságok összege. Ha $\mu = \infty$, és $\lambda = 0$, akkor a depótól legmesszebbi pont kerül beszúrásra.

Most nézzük meg Christophides, Mingozzi és Toth [26] heurisztikáját.

**1. Fázis** Szekvenciális útkonstrukció

1. lépés (Első út) Minden út rendelkezzen egy $k$ indexszel, és vegyük kezdetben a $k = 1$ indexűt.

2. lépés (A beszúrás költsége) Választunk egy fedetlen $i_k$ csúcsot, amivel inicializáljuk a $k$ utat. Minden fedetlen $i$-re kiszámoljuk $\delta_i = c_{0i} + \lambda c_{ii_k}$, ami a beszúrás költsége.

3. lépés (Pont beszúrása) Legyen $\delta_{i^*} = \min_{i \in S_k}\{\delta_i\}$, ahol $S_k$ azon fedetlen pontok halmaza, amelyek beszúrhatók a $k$ útba. Beszúrjuk az $i^*$-ot a $k$ útba. Optimalizáljuk a $k$ utat 3-opt eljárással, majd ismételjük a harmadik lépést addig, amíg több pontot már nem tudunk beszúrni.

4. lépés (Következő út) Ha minden pont be van szúrva: STOP. Különben legyen $k = k + 1$, és GO TO STEP 2.

**2. Fázis** Párhuzamos útkonstrukció

5. lépés (Út inicializálás) Inicializáljuk a $k$ utat $R_t = (0, i_t, 0)$-vel, ahol $(t = 1, \ldots, k)$, és $k$ az első fázis végén kapott utak száma. Legyen $J = \{R_1, \ldots, R_k\}$.

6. lépés (Költségek megfeleltetése) Minden $i$ pontra, ami még nincs megfeleltetve egyik útnak sem, és minden lehetséges $R_t \in J$ útra kiszámoljuk az $\epsilon_{ti} = c_{0i} + \mu c_{ii_t}$ értéket. Legyen $t^*$ annak az útnak az indexe, melyre $\epsilon_{t^*i} = \min_t\{\epsilon_{ti}\}$. Ekkor az $i$ pontot feleltessük meg a $R_{t^*}$ útnak, és ismételjük a hatodik lépést addig, amíg meg nem feleltettünk minden pontot egy útnak.

7. lépés (Költségek meghatározása) Vegyünk egy $R_t \in J$-t, és legyen $J = J \backslash \{R_t\}$. Minden $R_t$-nek megfeleltetett $i$ pontra számítsuk ki $\epsilon_{t'i} = \min_{R_t \in J}\{\epsilon_{ti}\}$-t, és legyen $\tau_i = \epsilon_{t'i} - \epsilon_{ti}$.

8.lépés (Pont beszúrása) Beszúrjuk $R_t$-be $i^*$-ot, ahol $i^*$ olyan, hogy $\tau_{i^*} = \max_{i \in S_t}\{\tau_i\}$. Itt $S_t$ az $R_t$-nek megfeleltetett fedetlen pontok halmaza. Optimalizáljuk $R_t$-t 3-opt eljárással, majd ismételjük a nyolcas lépést addig, amíg több pontot már nem lehet $R_t$-be szúrni.

9. lépés (Terminálás feltétele) Ha $|J| \neq \emptyset$, akkor GO TO 6. lépés. Különben, ha minden pontot lefedtünk STOP. Ha pedig maradtak fedetlen pontok, akkor új utat készítünk, és GO TO 1. lépés.

A harmadik lépésben értelemszerűen olyan pontokat szúrhatunk be, melyeknek az adott úthoz való hozzávétele még nem sértené meg a kapacitásra vonatkozó feltételt. Az első

fázis végén $k$ darab út keletkezik, és mindegyikre adott az $i_t$ pont ($t = 1, \ldots, k$), amellyel inicializáltuk őket. Ezek segítségével hozunk létre új utakat az ötös lépésben. Fontos, hogy a hatodik lépésben a megfeleltetés még nem jelenti a pont beszúrását. Ekkor is kiszámoljuk a beszúrás költségét, csak most a második, $\mu$ paramétert használjuk. A hetedik és nyolcadik lépésben lényege, hogy az az $R_t$-nek megfeleltetett pont kerül előbb beszúrásra az $R_t$-be, amelyiket a legdrágább lenne máshova beszúrni.

### 3.1.2. Két fázisú módszerek

A két fázisú módszerek alapelve az, hogy az algoritmus egyik fázisában megfelelő partícióját keressük meg a pontoknak úgy, hogy egy ponthalmaz összköltsége se haladja meg a kapacitást. Egy ilyen ponthalmaz fog megfelelni azoknak a fogyasztóknak, amelyeket egy járművel fogunk meglátogatni. A másik fázisban egy ponthalmazon keressük meg a TSP optimumot. Aszerint, hogy az algoritmus milyen sorrendben foglalkozik ezzel a két fázissal, megkülönböztethetünk *Cluster-first, route-second* és *Route-first, cluster-second* módszereket. Mi most csak a cluster-first, rout-second módszerekkel fogunk foglalkozni.

**Sweep algoritmus** (Lásd: [27]) A pontokat a depó körüli sugárirány szerint fogjuk osztályokba rendezni. Pontosabban legyen minden $i$ pont polárkoordinátája $(\varphi_i, \rho_i)$ úgy, hogy egy tetszőleges $i$ pontra válasszuk meg a $\varphi_i = 0$-t. Ezután rendezzük $\varphi$ alapján növekvő sorrendbe a pontokat. Az algoritmus lépései a következők.

1. lépés Választunk egy, még használatlan $k$ járművet.

2. lépés A még fedetlen pontok közül a legkisebb forgásszögűt hozzárendeljük $k$-hoz addig, amíg át nem léptük a kapacitásra vagy út hosszára vonatkozó korlátokat. Minden beszúrás után alkalmazhatunk 3-opt eljárást.

3. lépés Minden járműre a hozzárendelt pontok sorrendjét a TSP megoldásával optimalizáljuk.

Ha adott a menetidőre vonatkozó korlát is, akkor a második lépésben értelmezni kell azt, hogy addig vehetjük be a pontokat, amíg meg nem haladtuk a megengedett menetidőt. Hiszen az algoritmus ezen lépésében még nincsenek utak, melyek menetidejét ellenőrizni lehet. Erre egy (nem feltétlenül leggyorsabb) megoldás lehet a következő. Amikor eldöntjük, hogy az adott pontot még be tudjuk-e venni, vagy sem, akkor először a kapacitási feltételt ellenőrizzük le (ez a könnyebb). Ha ez teljesül, akkor a már útban lévő pontokat rendezzük a TSP optimum megkeresésével (beleértve az aktuális pontot), és ennek az

útnak a menetidejét ellenőrizzük le, hogy meghaladja-e a menetidőre vonatkozó feltételt. Ha a járművek kapacitása nem túl nagy, akkor egy járműhöz nem tartozhat túl sok pont, tehát ez az eljárás remélhetőleg nem lesz túl időigényes.

**Fischer és Jaikumar algoritmusa** (Lásd: [28]) Ebben az algoritmusban az általánosított párosítási problémára (GAP-re) fogjuk visszavezetni a feladatot. Ennek egy általános megfogalmazása úgy néz ki, hogy adott valamennyi tárgy, melyeket tárolókba szeretnénk elhelyezni. Minden tároló felhasználásának van valamennyi költsége. Az $i$-edik tárgy $j$-edik tárolóba történő elhelyezésének van egy $p_{ij}$ haszna, és egy $w_{ij}$ súlya. Minden tárolónak van egy kapacitása, vagyis a maximális súly, amit bele lehet helyezni. A cél, hogy maximalizáljuk a hasznot. A következő heurisztika ennek a problémának a megoldását használja fel a következő módon.

1. lépés Választunk egy $j_k$ *mag fogyasztót*, mellyel a $k$ osztályt inicializálni fogjuk. Tehát álljon most a $k$-adik osztály a $j_k$ pontból.

2. lépés Minden $i$ fogyasztóra és $k$ osztályra kiszámoljuk a

$$\delta_{ik} = \min\{c_{0i} + c_{ij_k} + c_{j_k0}, c_{0j_k} + c_{j_ki} + c_{i0}\} - (c_{0j_k} + c_{j_k0})$$

értéket.

3. lépés Mivel a pontokat szeretnénk hozzárendelni a meglévő osztályokhoz (járművekhez), megoldjuk a GAP-et úgy, hogy a költségek $\delta_{ij}$ minden $i$ pontra, és $j$ osztályra, az $i$. fogyasztók súlya $d_i$, és minden jármű kapacitása $C$.

4. lépés A GAP megoldásaként kapott osztályokra megoldjuk a TSP-t.

**Bramel és Simchi-Levi algoritmusa** A következő ötletek [29] cikkből származnak. A fenti algoritmusokhoz hasonlóan most is mag fogyasztókat keresünk, de ezeket ezúttal a *Capacitated location problem* (CCLP) segítségével határozzuk meg. A CCLP általánosan a következő feladatot jelenti. Adott $m$ lehetséges hely, ahová a koncentrátorokat (magokat) helyezhetjük. Minden ilyen $j$ helyre adott egy $Q_j$ kapacitás. Minden koncentrátornak van egy elhelyezési költsége, amely függ a helytől is, ahová helyezzük. Szeretnénk elhelyezni a koncentrátorokat az $m$ darab hely egy részhalmazán, és hozzákötni valamennyi terminált. Az $i$. terminál $w_i$ egységet használ fel a koncentrátor kapacitásából. Az $i$. terminál $j$. koncentrátorhoz való bekötésének költsége $c_{ij}$. A célünk az, hogy minden terminál pontosan egy darab koncentrátorhoz legyen kötve úgy, hogy a bekötött terminálok összköltsége ne

haladja meg a koncentrátor kapacitását, valamint a bekötés összköltsége legyen a lehető legolcsóbb.

Alkalmazzuk most a CCLP-t a CVRP-re a mag fogyasztók kiválasztásához. Minden fogyasztó legyen egy lehetséges hely. Az elhelyezés költsége a $j$-edik helyen megfelel annak a költségnek, hogy a $j$-edik fogyasztót magnak választjuk. Ez annak a költsége, hogy a depóból elmegyünk a $v_j$-be, meg vissza, azaz $c_{0j} + c_{j0}$. A többi fogyasztó a CVRP-ben a CCLP-ben egy terminálnak felel meg. Az $i$-edik terminál bekötése a $j$-edik helyen lévő koncentrátorhoz éppen annak a költsége, hogy az $i$-edik fogyasztót beszúrjuk a depó és a $j$-edik fogyasztó közötti útba. Értelemszerűen egy koncentrátor kapacitása legyen a $C - d_j$, ha a $j$-edik helyen helyezzük el a koncentrátort. Egy $i$ fogyasztó (terminál) által felhasznált kapacitás pedig legyen $d_i$.

Tehát az algoritmus úgy kezdődik, hogy miután kiszámoltuk a fent definiált költségeket, megoldjuk a CCLP-t. Ezzel kiválasztottuk a mag fogyasztókat. Ezután a többi fogyasztót kötjük a mag fogyasztókhoz a következőképp: legyen egy részben kész út $(v_0 = i_0, i_1, \ldots, i_{l+1} = v_0)$, $T_k = \{v_0, i_1, \ldots, i_l\}$, és legyen $t(T_k)$ az optimális TSP megoldás hossza. A beszúrás költsége minden $i$ fedetlen pontra a $k$ útba:

$$\delta_{ik} = t(T_k \cup \{i\}) - t(T_k).$$

Azt az $i$ pontot fogjuk beszúrni a $k$ útba, melyre $\delta_{ik}$ minimális. Mivel $\delta_{ik}$ kiszámítása költséges, ezért helyette használhatjuk a $\overline{d}_{ik} = \min_{h=1\ldots l}\{2c_{ii_h}\}$ *direkt költséget*, vagy a $\overline{d}_{ik} = \min_{h=0\ldots l}\{c_{i_h i} + c_{ii_{h+1}} - c_{i_h i_{h+1}}\}$ legközelebbi beszúrásának a költségét.

**Csonkított branch-and-bound** (Lásd.: [26]) A következőkben $K$ változóra alkalmazzuk a Branch-and-bound módszert. Vagyis minden branch esetén azt vizsgáljuk meg, hogy az aktuális változónak megfelelő jármű melyik útvonalat járja be a lehetségesek közül. A következő algoritmus az ún. *csonkított branch-and-bound* eljárást használja, ami azt jelenti, hogy minden branch esetén csak egy lehetséges leszármazott pontra vonatkozó problémát vizsgálunk tovább. Az algoritmus a következő.

1. lépés (Inicializálás) Legyen $h = 1$ és $F_h = V \backslash \{v_0\}$. ($F_h$ jelöli a $h$-adik szinten a fedetlen pontok halmazát.)

2. lépés (Utak generálása) Ha $F_h = \emptyset$, STOP. Különben kiválasztunk egy még fedetlen $v_i$ pontot, és létrehozzuk azon utak halmazát, melyek a még fedetlen pontokat és $v_i$-t tartalmazzák. Legyen ez a halmaz $R_i$.

3. lépés (Kiértékelés) Minden $r \in R_i$-re kiszámoljuk az $f(r) = t(S_r \cup \{v_0\}) + u(F_h \backslash S_r)$ értéket, ahol $S_r$ jelöli az $r$ út pontjainak a halmazát, $t$ a megfelelő halmazon vett optimális TSP megoldás hossza, valamint $u$ a megfelelő halamazon vett legolcsóbb feszítőfa költsége.

4. lépés (Út kiválasztás) Legyen $r^*$ az az út, ahol $\min_{r \in R_i}\{f(r)\}$ felvétetik. Legyen $h = h+1$, és $F_h = F_{h-1} \backslash S_{r^*}$, majd GO TO 2. lépés.

**Szirom algoritmus** A következő módszerek a sweep algoritmust terjesztik ki úgy, hogy különböző utakat *(szirmokat)* generálnak, majd ezekre oldjuk meg a set partitioning problémát. A szirmok generálásával lényegében egy megszorítást teszünk az utakra, azonban a változók száma még így is rengeteg.

Balinski és Quant [30] ötlete, hogy töröljük ki a változók közül azokat, melyek *dominált*ak. A $j$ utat akkor nevezzük dominánltnak, ha létezik az utaknak egy olyan részhalmaza, melyek pontosan azokat a fogyasztókat szolgálják ki, amelyeket $j$ is, viszont az összköltségük kisebb, mint $j$ költsége. Azaz a $H_j$ dominálva van, ha létezik olyan $S$ indexhalmaz, hogy

$$\sum_{i \in S} A_i = A_j \quad \text{és} \quad \sum_{i \in S} c_i \leq c_j,$$

ahol $H_j$ jelöli a set partitioning modellben a $j$-edik utat, és $A_i$ jelöli ugyanebben a modellben a mátrix $i$-edik oszlopvektorát. Mivel a dominált utak biztosan nem fognak az optimális megoldáshoz tartozni, ezeket törölhetjük a változók közül.

Foster és Ryan [31] úgy próbálták csökkenteni a változók számát, hogy az utakra implicit feltételeket szabtak meg. Ez azon a megfigyelésen alapszik, hogy az optimális megoldásban szereplő utak hasonló tulajdonságokkal rendelkeznek.

1. Ha egy út kiszolgálja a fogyasztóknak egy szektorát (vagyis valamely, a depóból induló sugarak által meghatározott fogyasztók egy halmazát), akkor egy, a szektoron belül lévő pontot ritkán hagy ki. Ez csak olyan kivételes esetben történik meg, ha sérülnek a kapacitási, vagy menetidőre vonatkozó feltételek.

2. Szomszédos utak ritkán keresztezik egymást.

3. Fogyasztók egy rögzített halmazára a meglátogatás optimális sorrendjét a TSP megoldásával nyerhetjük.

Tehát a lehetséges úthalmazt, amelyre a feladatot meg szeretnénk oldani, úgy kapjuk, hogy meghatározzuk az olyan utakat, melyekre a fentiek közül az első kettő teljesül, majd min-

den útra megkeressük a TSP optimumot. Ezt a következőképp fogjuk csinálni. Rendezzük a fogyasztókat depó körüli forgásszög alapján növekvő sorrendbe, majd ebben a sorrendben az egymást követő fogyasztók részhalmazai lesznek a lehetséges utak. Azon utakat, melyek összesített igénye meghaladja a kapacitást, töröljük a változók közül. A megmaradt lehetséges ponthalmazokra megkeressük a TSP optimumot, majd töröljük azon utakat a változók közül, melyek menetideje meghaladja a menetidőre vonatkozó korlátot. Ezzel megkapjuk a *lehetséges szirmok* halmazát. Belátható, hogy ha az IP feladatot csak a szirmokhoz tartozó oszlopok alkotta mátrixszal írjuk fel, akkor a kapott mátrix TU, így elég az LP feladatot megoldani az IP helyett. Egy további lehetőség a lehetséges szirmok számának csökkentésére, ha a kapacitásra alsó korlátot is bevezetünk. Ez azt eredményezi, hogy kizárjuk az olyan lehetséges utakat, melyek olyan kevés fogyasztót szolgálnak ki, hogy a maradék fogyasztó már nem szolgálható ki a még fel nem használt járművel. Ehhez adjunk alsó korlátot arra, hogy mennyi járműre lesz szükségünk; a triviális

$$v = \left\lceil \frac{\sum_{i=1}^{n} d_i}{C} \right\rceil$$

alsó korlát jó lesz. Ha feltesszük, hogy pontosan $v$ jármű elég a fogyasztók kiszolgálásához, akkor az alsó korlát a kapacitásra minden lehetséges útnál

$$c_l = \sum_{i=1}^{n} d_i - (v-1)C,$$

amit a fenti képletből kapunk. Természetesen, ha a megoldás során több járművet is fel szeretnénk használni, akkor ezt a $c_l$ alsó korlátot megfelelően csökkenthetjük.

Ryan, Hjorring és Glover [32] az előzőhöz hasonló módon kezdték el vizsgálni a problémát.

**3.1. Definíció.** *Feszítőszirom halmaznak* nevezzük a szirmoknak egy olyan halmazát, amelyek együtt minden fogyasztót pontosan egyszer szolgálnak ki.

A célunk az, hogy megkeressük a minimális költségű feszítőszirom halmazt. Erre a következőkben egy legrövidebb út keresésen alapuló algoritmust fogunk látni. De előbb meg kell jegyezni, hogy a fogyasztók depó körüli forgásszög alapján történő rendezése nem feltétlenül ad optimális megoldást, viszont létezhet több olyan sorrend, amelyekből megkapható az optimum. Ehhez értelmezzük az *általánosított szirom* fogalmát, ami az előző szirom fogalomtól csupán abban különbözik, hogy nem követeljük meg a pontok sugárirányú rendezését. Tehát most az előző módszertől abban térünk el, hogy a kapott megengedett szirmokra nem az IP feladatot írjuk fel, hanem a problémát visszavezetjük

egy legrövidebb út kereső problémára. Ezt a következőképpen tesszük. A szirmokat reprezentáljuk egy súlyozott, körkörös gráffal úgy, hogy a gráf legyen $G = (N, A)$, ahol az $N$-beli pontok az eredeti fogyasztók, és $i$-ből $j$-be pontosan akkor megy irányított él, ha létezik megengedett szirom, amely első pontja $i$, és $j$ az első pont a pontok sorrendjében, ami már nincs benne a sziromban. Ezt a reprezentációt ugyanúgy alkalmazhatjuk a szirmokra és általánosított szirmokra is.

**3.2. Definíció.** *Kompakt körnek* nevezzük azt a kört, melynek minden $(i, j)$ élére teljesül, hogy nincs olyan pontja a körnek, amely $i$ és $j$ között van a fent leírt körkörös rendezés szerint.

**5. Tétel.** *Létezik egyértelmű megfeleltetés a feszítőszirom halmazok és a kompakt körök között.*

**Bizonyítás.** Egy feszítőszirom halmazban minden pont pontosan egyszer fordul elő. Így a neki megfelelő élek a fent leírt digráfban éppen egy kompakt kört fognak adni. Fordítva, egy adott kompakt kör esetén az élek a szirmoknak felelnek meg, és egy szirom azokat a fogyasztókat látogatja meg, melyek a hozzá tartozó él kezdő- és végpontja között vannak, ezért a kapott sziromrendszerben minden fogyasztó pontosan egyszer fog szerepelni, ami azt jelenti, hogy feszítőszirom halmazt kaptunk. □

A feladatunk tehát, hogy megtaláljuk a legolcsóbb kompakt kört a gráfban. Ehhez transzformáljuk a fenti körkörös gráfot aciklikussá a következőképp: válasszunk ki egy $k$ pontot, és vágjuk ketté. A $k_0$-ból induljon az összes $k$-ból kilépő él, és $k_d$-be irányítjuk a $k$-ba belépő éleket. Legyen $k_0$ indexe nulla, $k_d$ indexe pedig $n + 1$, a többi pontot pedig értelemszerűen indexeljük $k_0$-tól $k_d$-ig. Ezután töröljük az eredeti gráf azon éleit, melyek „$k$ mellett elhaladnak”, vagyis melyeknek az új indexelés szerint a végpontja kisebb pozícióban van, mint a kezdőpontja. Ezzel a $k$-n átmenő köröket vizsgáljuk, vagyis az olyan szirommegoldásokat, melyeknél a $k$-adik fogyasztó valamely sziromnak az első pontja. Az ilyen sziromrendszerek optimumának megtalálásához keressük meg a $k_0$-ból $k_d$-be menő utak közül a legolcsóbbat. Nyilván, ha ezt minden $k$-ra megcsináljuk, akkor a kapott legrövidebb utak közül a legolcsóbb adja majd meg az optimális szirom megoldást. De mivel ez egy idő-, és helyigényes művelet, ezért szeretnénk minél kevesebb $k$ pontra végrehajtani ezt. Ezért további meggondolásokra lesz szükségünk.

Ha a fenti eljárást csak a $k$ pontra hajtjuk végre, akkor nem vettük figyelembe azokat az eseteket, amikor $k$, és az őt megelőző pont egy sziromba tartozik. Legyen $P_k$ az az állítás, hogy az optimális megoldásban $k$ és az őt megelőző pont ugyanabban a sziromban van,

és legyen $\overline{P_k}$ az állítás komplementere. Ha $\overline{P_k}$ igaz, akkor az optimális szirom megoldást megtalálhatjuk, ha csak $k$-ra hajtjuk végre a legrövidebb út keresést. Ezt általánosabban úgy is megfogalmazhatjuk, hogy ha $s(k)$ jelöli a $k$ pont rákövetkezőjét a megfelelő sorba rendezésben, akkor ha $k, s(k), \ldots, s^i(k)$ ugyanabban a sziromban vannak, akkor $P_{s(k)} \wedge P_{s^2(k)} \wedge \ldots \wedge P_{s^j(k)}$ igazak.

**6. Tétel.** *Ha az $a_{i,s^{j+1}(i)}$ él nem létezik, akkor az optimális szirom megoldás megtalálható $j$ darab legrövidebb útkeresésből.*

**Bizonyítás.** Ekkor nem létezik olyan szirom, amely tartalmazza az $i, s(i), \ldots, s^j(i)$-ket, így a $\overline{(P_{s(i)} \wedge P_{s^2(i)} \wedge \ldots P_{s^j(i)})}$ állítás igaz lesz, azaz $\overline{P_{s(i)}} \vee \overline{P_{s^2(j)}} \vee \ldots \vee \overline{P_{s^j(i)}}$ igaz. Így elég erre a $j$ darab pontra megkeresni a legrövidebb utat, és ezek minimuma lesz az optimum. □

Tehát ha a lehető legkevesebb legrövidebb út keresést szeretnénk végrehajtani, akkor meg kell keresnünk azt az $i$-t, melyre $j$ minimális, és $a_{i,s^{j+1}(i)} \notin A$.

A most ismertetett Ryan-Hjorring-Glover-féle algoritmus szemléltetésére készítettem egy programot, melynek C++ nyelven megírt kódja a függelékben található. A program a szirmok létrehozásához a fogyasztók hagyományos, depó körüli forgásszög szerinti sorrendjét használja. A fogyasztók egyes halmazain a TSP optimumot a legtávolabbi pont beszúrásának heurisztikájával közelíti meg. A program euklideszi pontok koordinátáira működik, melyek között az élköltség és a menetidő a pontok közötti euklideszi távolság. Az élköltségek felfelé, egészekre vannak kerekítve. Ezt a programot a Christofides, Mingozzi és Toth 14 ismert tesztesetére futtattam, és a következő oldalon található táblázatban összefoglalt eredményeket kaptam. Látható, hogy a megírt program átlagosan 26%-kal ad rosszabb megoldásokat, mint amennyi az eddig ismert legjobb. Ennek a nem túl jó eredménynek több oka is lehet. Egyik például, hogy az alkalmazott TSP heurisztika nem biztos, hogy elég jó utakat talál. A másik probléma, hogy a fogyasztókat kezdetben nem biztos, hogy a legjobb sorrendbe rendeztük, és úgy kerestünk megengedett szirmokat. Ezek javíthatók, ha például összetettebb TSP közelítő módszert alkalmazunk, és a fogyasztók más sorrendjét is kipróbáljuk.

Renaud, Boctor és Laporte [33] szintén a szirom fogalmát általánosítja a következőképp: legyen az *r-szirom* olyan $r$ darab útnak a halmaza, melyek együtt kiszolgálják egy szektor összes fogyasztóját. A következőkben 1-szirmokat és 2-szirmokat fogunk sweep folyamattal generálni. 1-szirom heurisztika esetén a fogyasztók egy $S$ halmazán és a depón konstruálunk egy utat TSP heurisztikával. A 2-szirom heurisztika során szintén egy $S$ halmazon szeretnénk szirmokat létrehozni. Ehhez válasszunk ki kettő mag fogyasztót úgy,

| Teszteset | Paraméterek (fogyasztók száma, kapacitás, megengedett menetidő, depó koordinátái) | Futási idő (sec) | Megoldás költsége | Eddig ismert legjobb | Százalékos eltérés |
|---|---|---|---|---|---|
| 1. | 50; 160; - ; 30; 40 | 0,0312 | 660 | 524,61 | 26% |
| 2. | 75; 140; - ; 40; 40 | 0,0156 | 1103 | 835,26 | 32% |
| 3. | 100; 200; - ; 35; 35 | 0,0624 | 1006 | 826,14 | 22% |
| 4. | 150; 200; - ; 35; 35 | 0,078 | 1269 | 1028,42 | 23% |
| 5. | 199; 200; - ; 35; 35 | 0,0936 | 1757 | 1291,29 | 36% |
| 6. | 50; 160; 200; 30; 40 | 0,0156 | 660 | 555,43 | 19% |
| 7. | 75; 140; 160; 40; 40 | 0,0156 | 1103 | 909,68 | 21% |
| 8. | 100; 200; 230; 35; 35 | 0,0624 | 1006 | 865,94 | 16% |
| 9. | 150; 200; 200; 35; 35 | 0,0936 | 1269 | 1162,55 | 9% |
| 10. | 199; 200; 200; 35; 35 | 0,078 | 1757 | 1395,85 | 26% |
| 11. | 120; 200; - ; 10; 45 | 0,156 | 1709 | 1042,11 | 64% |
| 12. | 100; 200; - ; 40; 50 | 0,0312 | 1079 | 819,56 | 32% |
| 13. | 100; 200; 720; 10; 45 | 0,156 | 1709 | 1541,14 | 11% |
| 14. | 100; 200; 1040; 40; 50 | 0,0468 | 1079 | 866,37 | 25% |

Eddig ismert legjobb eredmények forrása: http://neo.lcc.uma.es/vrp/known-best-results/

hogy egymástól a lehető legtávolabb legyenek. Ezzel értelmezünk kettő oda-vissza utat a depó és a két mag fogyasztó között. A többi fogyasztót ezután ebbe a két útba szúrjuk be úgy, hogy a költség a legkevésbé növekedjen, és a megengedettség ne sérüljön. Ezután 4-opt eljárással újra optimalizálunk. Ahhoz, hogy ne kapjunk kiegyensúlyozatlan utakat, bevezethetünk egy *szorossági együtthatót*, ami legyen

$$\alpha = 2C - \sum_{v_i \in S} d_i,$$

és amikor beszúrjuk a következő fogyasztót, megkövetelhetjük, hogy a két út teljes igényének a különbsége ne haladja meg $\alpha$-t. Mivel az utak költségén később reoptimalizálással még javíthatunk, ezért a beszúrásnál megengedjük, hogy az út hossza meghaladja a megengedett maximális menetidőt.

### 3.1.3. Javító módszerek

A már meglévő utakat különböző módszerekkel megpróbálhatjuk javítani. Ilyen módszerekből két fajtát különböztetünk meg. Az egyik, amikor egyszerre csak egy jármű által bejárt utat vizsgálunk, és azt próbáljuk meg optimalizálni. A másik, amikor két utat egymáshoz képest vizsgálunk, és javító élcserékkel próbálkozunk.

Az utak külön-külön javítására adott eljárások gyakorlatilag megegyeznek a TSP-ben alkalmazott javítómódszerekkel. Ilyenek például a $\lambda$-optimalizáló eljárások, melyeket a fent

leírt heurisztikákban is alkalmaztunk már. Ezek lényege, hogy az útból eltörlünk $\lambda$ darab nem egymást követő élt, és a keletkező $\lambda$ szakadásra megpróbálunk minden lehetséges módon egy újra csatlakozást keresni, és ezekből a legolcsóbbat kiválasztani. A procedúra a lokális minimumnál áll le, és $O(n^{\lambda})$ futásidejű.

A TSP-től eltérően a VRP esetében alkalmazhatunk olyan módszereket, melyek egyszerre több utat vizsgálnak, és egymáshoz képest próbálják javítani őket. Van Breedam [34] leírásában négy műveletet ismerhetünk meg az utak javítására.

1. *String cross* (SC). Két különböző út fogyasztóinak egy-egy sorozatát választjuk ki, és ezeket kicseréljük a két út között úgy, hogy az eredeti utak egy-egy élét keresztezzük.

2. *String exchange* (SE). Két legfeljebb $k$ hosszú sorozatát a pontoknak kicseréljük az utak között.

3. *String relocation* (SR). Fogyasztók egy $k$ hosszú sorozatát egy másik útba tesszük.

4. *String mix* (SM). Kiválasztjuk SE és SR közül a legjobb mozgást.

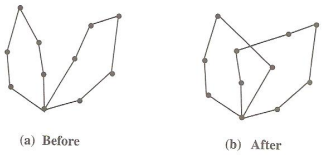



**Figure 5.1.** *String cross.*

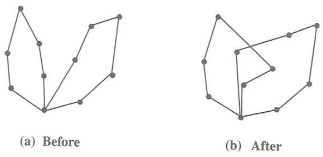



**Figure 5.2.** *String exchange.*

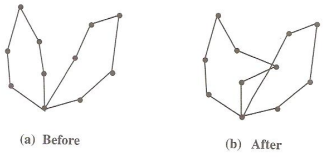



**Figure 5.3.** *String relocation.*

A javítás minőségét befolyásoló tényezők az alábbiak lehetnek.

- A kezdeti megoldás minősége.
- A kiválasztott fogyasztók száma ($k$).
- A mozgatások típusa (SC, SE, SR, SM).

- A kiválasztás stratégiája, azaz hogy az első javítás esetén cseréljük-e le az éleket, vagy az összes lehetőséget megvizsgálva a legjobbat választjuk ki.

- Annak a procedúrának a hossza, amellyel kiválasztjuk azt a $k$-t, és azt a fogyasztóláncot, melyre a javítást érdemes véghezvinni.

## 3.2. Metaheurisztikák

A most következő heurisztikákat hat lényegesebb osztályba sorolhatjuk. Ezek a *Simulated Annealing* (SA), *Deterministic Annealing* (DA), *Tabu Keresés* (TS), *Genetikus Algoritmus* (GA), *Ant Systems* (AS), és a *Neurális Hálók* (NN). Ezek közül nem fogom mindet kifejteni, most csak az SA, TS, és AS heurisztikákkal foglalkozom.

Az SA és TS heurisztikák alapgondolata, hogy egy kezdeti $x$ megoldásból indulunk ki, és minden $t$ iterációban az $x_t$ megoldást valamilyen módon áttranszformáljuk $x_{t+1}$ megoldássá. Ennek a transzformációnak a lehetséges lépései előre rögzítettek, és így $x_t$-nek egy *szomszédságát* (jelölés: $N(x_t)$) határozzák meg. Ebből a szomszédságból határozzuk meg valamilyen módon, hogy melyik legyen az iteráció következő $x_{t+1}$ megoldása. Természetesen különböző módszereket kell arra alkalmazni, hogy a ciklizálást elkerüljük. Ezeket a következőkben látni fogjuk.

### 3.2.1. Simulated Annealing

Ez a módszer a $t$-edik iterációnál random választ egy $x$ megoldást $N(x_t)$-ből. Ha $f(x) \leq f(x_t)$ (ahol $f(x)$ jelöli az $x$ megoldás költségét), akkor legyen $x_{t+1} = x$. Ha ez nem teljesül, akkor legyen

$$x_{t+1} = \begin{cases} x, & p_t \quad \text{valószínűséggel} \\ x_t, & 1 - p_t \quad \text{valószínűséggel,} \end{cases}$$

ahol $p_t$ egy csökkenő függvénye $t$-nek, és $f(x) - f(x_t)$-nek. Pontosabban

$$p_t = \exp\left(-\frac{f(x) - f(x_t)}{\theta_t}\right),$$

ahol $\theta_t$ a $t$-edik iteráció *hőfoka*. Ez egy csökkenő függvénye $t$-nek, amely kezdetben kap egy $\theta_1 > 0$ értéket, és minden $T$-edik iterációban egy $0 < \alpha < 1$ számmal szorozzuk. Ezzel azt érjük el, hogy az idő múlásával $p_t$ folyamatosan csökkenjen, így egyre kisebb valószínűséggel fogadjuk el a nem javító megoldásokat.

A megállás feltételei a következők lehetnek.

- Az aktuális $x$-re $f$ értéke már nem csökkent legalább egy előre adott $\pi_1$ értéket legalább bizonyos $k_1$ darab $T$ hosszú iteráció óta.
- Az elfogadott lépések száma $T$ iterációban kevesebb, mint egy rögzített $\pi_2$ szám $k_2$ darab $T$ hosszú iteráció óta.
- A $T$ iterációk egy $k_3$ számát már végrehajtottuk.

**Osman SA algoritmusa** Osman [35] az SA algoritmusában $\lambda$ *interchange generation mechanism* módszert használt a szomszédság kialakításához. Ez úgy történik, hogy választunk $p$ és $q$ utakat, melyekhez tartozó fogyasztók egy részhalmaza $S_p$ és $S_q$, melyek olyanok, hogy $|S_p| \leq \lambda$ és $|S_q| \leq \lambda$. Az eljárás kicseréli $S_p$ és $S_q$ között a fogyasztókat, amíg az lehetséges. $S_p$ vagy $S_q$ egyike üres is lehet, ebben az esetben csak áttoljuk a fogyasztókat az egyik útból a másikba. Mivel nagy $\lambda$ esetén nagyon sok lehetőség van a pontok cseréjére, ezért általában $\lambda = 1$ vagy $\lambda = 2$ esetet szokták használni. A keresés akkor áll meg, ha javítást találtunk. Az algoritmus a következő.

**1. Fázis** (Descent Algoritmus)

1. lépés (Inicializálás) Clark-and-Wright algoritmussal keresünk egy kezdeti megoldást.
2. lépés Keresünk egy megoldási teret $\lambda$-interchange módszerrel. A javítást rögtön implementáljuk, amint azt megtaláltuk. Megállunk, ha a teljes szomszédság átkutatása nem ad jobb eredményt.

**2. Fázis** (SA keresés)

1. lépés (Inicializálás) Kezdeti megoldás legyen az első fázis eredménye, ami akár a Clark-and-Wright algoritmus eredménye is lehet, ha a második lépésben megálltunk. A teljes szomszédságon alkalmazzuk a $\lambda$-interchange módszert anélkül, hogy bármely mozgást implementálnánk. Legyen $\Delta_{max}$ és $\Delta_{min}$ a célfüggvény legnagyobb és legkisebb változása, a nem végrehajtott mozgások között. Legyen $\beta$ a legkisebb cserék száma, azaz azon irányok száma, amelybe léphettünk volna. Legyen $\theta_1 = \Delta_{max}$, $\delta = 0$, $k = 1$, $k_3 = 3$, $t = 1$ és $t^* = 1$. Az eddig talált legjobb megoldás az aktuális iterációból jön, ez legyen $x^* = x_1$.
2. lépés (Következő megoldás) $\lambda$-interchange módszerrel keresünk egy szomszédságát $x_t$-nek. Ha $x$ megoldásra $f(x) < f(x_t)$, akkor legyen $x_{t+1} = x$. Ha $f(x) < f(x^*)$, akkor $x^* = x$, és $\theta^* = \theta_k$. Ha a teljes keresés során nem tudunk javítani,

akkor legyen $x$ a legjobb megoldás $x_t$ szomszédságában, és legyen $x_{t+1} = x$, $p_t$ valószínűséggel, vagy $x_{t+1} = x_t$ legyen $1 - p_t$ valószínűséggel. Ha $x_{t+1} = x_t$, akkor legyen $\delta = 1$.

3. lépés (Hőfok frissítés) Ha $\delta = 1$, akkor legyen $\theta_{t+1} = \max\{\frac{\theta_t}{2}, \theta^*\}$, és legyen $\delta = 0$ és $k = k + 1$. Ezt *véletlen növekedési szabálynak* nevezzük. Ha $\delta = 0$, akkor a *normális növekedési szabályt* alkalmazzuk, ami azt jelenti, hogy legyen $\theta = \frac{\theta_t}{(n\beta + n\sqrt{t})\Delta_{max}\Delta_{min}}$. Legyen továbbá $t = t + 1$. Ha $k = k_3$, STOP. Különben GO TO 2. lépés.

Itt $p_t$ a fent leírt képlettel kapható meg. A $\delta$ egy logikai változó, ami akkor lesz 1, ha az aktuális iterációban nem tudtunk javítani, és egy eddiginél rosszabb megoldást választottunk az iteráció következő megoldásának. Értelemszerűen $k$ jelöli az iterációk számát, amikor ezt a nem javító lépést alkalmaztuk, $t$ pedig az összes eddig megtett iterációk száma. $\theta^*$ az eddig talált legjobb megoldás esetén a hőfok.

### 3.2.2. Tabu Keresés

Az SA eljárás során a ciklizálást úgy kerültük el, hogy néhány lépésben megengedtük, hogy a következő iterációban szereplő megoldás rosszabb legyen, mint az aktuális. A tabu keresés esetében azonban a mozgást minden esetben úgy hajtjuk végre, hogy $x_{t+1}$ az $x_t$ legjobb szomszédja legyen. A ciklizálást most úgy kerüljük el, hogy a már megvizsgált megoldásokat egy bizonyos időre tiltjuk, azaz tabunak nyilvánítjuk. Azért, hogy időt és memóriát takarítsunk meg, gyakran csak néhány tulajdonságát jegyezzük meg a tiltott utaknak a teljes megoldás helyett. Ezeket *tabu feltételeknek* nevezzük. Például ha egy másik megoldásba mozgás során az $R_p$ útnak a $v_i$ fogyasztóját kicseréljük az $R_q$ útnak a $v_j$ fogyasztójával, akkor a tabu feltétel legyen az, hogy $v_i$ visszakerül $R_p$-be, és $v_j$ visszakerül $R_q$-ba.

**Osman algoritmusa** Osman [35] a szomszédság létrehozásához a fent látottakhoz hasonlóan $\lambda$-interchenge módszert használ $\lambda = 2$-re.

Ahhoz, hogy memóriát és időt takarítsunk meg, adatstruktúrának alkalmazzunk egy mátrixot (legyen ez $TABL$), melynek sorai a fogyasztókat és a depót jelölik, oszlopai pedig az utakat. Így a fent leírt két fogyasztó cseréjét meg tudjuk határozni a mátrix két elemével, $(i, R_p)$-vel, és $(j, R_q)$-val. $TABL(i, p)$ tartalmazza annak az iterációnak a számát, amelyben $i$-t eltávolítottuk $R_p$-ből. Kezdetben ezek legyenek negatív számok. A továbbiakban használni fogjuk az alábbi fogalmakat. A *Fogyási stratégia* az a módszer, amely

szerint eldöntjük, hogy melyik megoldások kerülhetnek ki a tabuból $|T_s|$ iteráció után, ahol $T_s$ jelöli a tabu lista méretét. $|T_s|$ értéke a probléma karakterisztikájától (fogyasztók száma, járművek száma, kapacitás és igények aránya) és a mozgáskiválasztó stratégiától függ. Ennek értékét gyakorlati kísérletek alapján szokták kiszámolni. Feladatunk, hogy könnyen és gyorsan ellenőrizzük, hogy valami tabu státuszban van-e. Ezt úgy tehetjük meg, ha a $k$-adik iterációban az alábbi egyenlőtlenség teljesülését ellenőrizzük.

$$k - TABL(i, p) < |T_s|.$$

A *Short-term* stratégia az a módszer, amely a tabu feltételekkel rendelkező megoldásokat ignorálja, és kiválasztja a következő megoldást $N(x)$-ből az alábbi eljárások valamelyikét használva.

- A *BA* (best admissible) eljárás a teljes szomszédságot vizsgálja, és az össze lehetőség közül a legjobbat választja ki.
- Az *FBA* (first best admissible) módszer során az első elfogadható javítást választjuk ki, ha létezik. Ha nem létezik, BA-t alkalmazunk. (Megjegyezzük, hogy ennek futásideje speciális adatszerkezettel javítható.)

Egy mozgást elfogadottnak nevezünk, ha nem tabu, vagy olyan tabu, amely megfelel a *törekvési feltételeknek*. Utóbbi fogalom bevezetésére azért van szükség, hogy kiküszöböljük azt a problémát, hogy nem tububan lévő utakat is tabuként kezelünk, hiszen a tabu megoldásoknak csak egy bizonyos tulajdonságát tároljuk. A törekvési feltétel legyen a következő. Ha $x_b$ az eddig talált legjobb megoldás, és $x'$ egy tabu megoldás, akkor $x'$-t elfogadjuk, ha $c(x') < c(x_b)$.

A megállási kritérium az eddigi legjobb megoldás megtalálása utáni iterációk számán alapszik, vagyis hogy milyen sokáig próbáljuk javítani az eredményt. Nevezzük ezen iterációk maximális számát $MAXI$-nek. Ennek kiszámítása $|T_s|$-hez hasonlóan gyakorlati kísérleteken alapszik.

Az algoritmus lépései a következők.

1. lépés Legyen $x$ kezdeti megoldás a Clark-and-Wright heurisztikából. Ha BA-t használunk, akkor az adatstruktúra inicializáláshoz hajtsunk végre egy keresést. Állítsuk be $|T_s|$-et, $TABL$-nek magas értékeket, $MAXI$ értékét, és legyen $x_b = x$ az eddig talált legjobb megoldás, legyen $k = 1$, és $k_b = 0$.

2. lépés Válasszunk egy lehetséges elfogadott mozgást, legyen az új megoldás: $x' \in N(x)$ BA vagy FBA alapján. Rögzítsük az új mozgás tulajdonságát $TABL$-ben. Frissítsük az

aktuális megoldást: $x' = x$ és $k = k + 1$. Ha $c(x') < c(x_b)$, akkor legyen $x_b = x'$ és $k_b = k$. Ha BA-t használunk, akkor frissítsük az ott használt adatstruktúrában az eredményeket.

3. lépés Ha $k - k_b > MAXI$, akkor STOP. Különben GO TO 2. lépés.

**Tabu utak módszere** A tabu utak módszere [37] nagyon jól működik, és gyakran eredményezheti az eddig ismert legjobb megoldást. Ebben az eljárásban az aktuális megoldásnak egy másik megoldás akkor szomszédja, ha az aktuálisból megkapható úgy, hogy elhagyunk belőle egy pontot, és beszúrjuk egy olyan útba, amely tartalmaz egyet a pont első $p$ darab legközelebbi szomszédja közül. A beszúrást egy, a TSP-re adott beszúró módszerrel, a *Generalized Insertion Procerure* (GENI) [36] eljárással tesszük. A procedúra olyan utakat is vizsgál, melyek nem lehetnek megengedett megoldás részei, például mert sérülnek a kapacitásra vagy menetidőre vonatkozó feltételek. Ezért felveszünk a célfüggvénybe két további büntető tagot. Egyik a kapacitásra, másik a menetidőre vonatkozik. Ezeket egy-egy paraméterrel súlyozzuk, melyeket kettővel osztunk, ha az utolsó tíz megoldás megengedett volt, és kettővel szorozzuk, ha az az utolsó tíz nem megengedett volt. Ez azt jelenti, hogy a megoldásokon történő mozgást úgy befolyásoljuk, hogy a megengedett megoldások költsége jóval kisebb lesz, mint a nem megengedetteké, ezért ezeket előnyben részesítjük. Ha már kellően régóta nem mozogtunk nem megengedett megoldás felé, akkor a büntetést csökkentve ismét nagyobb eséllyel választunk nem megengedett utakat, ezzel csökkentve a valószínűségét annak, hogy egy lokális optimumnál megragadjunk. Ha pedig már régóta nem mozogtunk megengedett megoldásokon, akkor a megoldást a megengedettség felé akarjuk terelni, ezért növeljük a büntetést. Minden beszúrás után újraoptimalizáljuk az aktuális utat, amelynél szintén a TSP-re adott *Unstringing and stringing* (US) [36] módszert használjuk.

A tabu lista helyett most random tabu elemeket használunk. Ez azt jelenti, hogy ha egy pontot egy másik útba szúrunk be a $t$-edik iterációban, akkor $t + \theta$ iteráción keresztül nem szúrhatjuk vissza az eredeti helyére, ahol $\theta \in [5; 10]$ random generált érték. Ezen kívül az eljárásnak még egy sajátossága van, amit *diverzifikációs stratégiának* nevezünk. Ez azt jelenti, hogy megbüntetjük azokat a pontokat, melyeket gyakran mozgatunk azért, hogy előnyben részesítsük a ritkán mozgó pontokat. Ezzel mindig újabb és újabb megoldásokat kapunk az iteráció során. Ezt úgy tesszük, hogy a célfüggvényhez mesterségesen adunk hozzá újabb tagokat az aktuális pont mozgásának gyakoriságával arányos módon. Az algoritmus ismertetése előtt még azt is meg kell jegyezni, hogy az eljárás *false start*

módszert használ az első megoldás inicializálásához. A false start esetén kezdetben több megoldást generálunk, és a legjobbat választjuk kezdő megoldásnak.

A továbbiakban minden iterációban jelölje $W$ azon pontok halmazát, melyeket a következő beszúrás alanyai lehetnek, és $q \leq |W|$ legyen azon pontok száma, melyeket próbaként beszúrtunk. Ezen kívül jelöljük $k$-val azon iterációk számát, melyekkel nem javítottunk. Az algoritmus egy rövid leírása a következő.

1. lépés (Inicializálás.) Hozzunk létre $\lceil \frac{\sqrt{n}}{2} \rceil$ darab kezdeti megoldást, és hajtsuk végre a tabu keresést úgy, hogy $W = V \backslash \{v_0\}$, $q = 5m$, és $k = 50$. Ekkor $q$ értéke gondoskodni fog arról, hogy minden pontot legalább 90% valószínűséggel válasszunk.

2. lépés (Megoldás javítása.) Az első lépés legjobb megoldásával kezdünk, és hajtsuk végre a tabu keresést $W = V \backslash \{v_0\}$, $q = 5m$, és $k = 50n$ értékekre.

3. lépés (Megerősítés.) Induljunk ki a második lépésben kapott legjobb megoldásból, és hajtsuk végre a tabu keresést $k = 50$-re. $W$ azon $\lfloor \frac{|V|}{2} \rfloor$ darab pont halmaza, melyek az első és második lépésben a legtöbbet voltak alkalmazva, és legyen $q = |W|$.

### 3.2.3. Ant Systems

Végül nézzünk meg főleg az érdekesség kedvéért még egy eljárást, melyet most csak a TSP-re fogok bemutatni.

Az Ant Systems módszereken alapuló algoritmusok jó példát adnak arra, hogy hogyan lehet a természetből ihletet meríteni egy probléma megoldása során. Ezek az algoritmusok ugyanis a hangyák ételkeresési módszereit használják fel alapul. Ha a hangyák valahol ételt találnak, akkor az oda vezető utat illatanyaggal (feromonnal) látják el, melynek minősége információt ad a többi hangyának az út hosszáról, és az ott található étel minőségéről. A többet ígérő útvonalakat persze több hangya használja, így ezek feromon szintje felerősödik. A kevésbé használt utakon pedig az ott elhelyezett illatanyag lassan elpárolog.

Hogyan tudjuk ezeket a módszereket hasznosítani? [38] szerint az alapötlet, hogy a célfüggvényre gondoljunk úgy, mint az étel minőségére, és a memóriában tárolt információk viselkedjenek a feromonokhoz hasonló módon. Az elgondolás a TSP esetén a következőképpen néz ki. Minden $(i, j)$ élhez rendelünk két értéket. A láthatóságot jelölje $n_{ij}$, ami az élhossz inverze, a feromon értéket pedig jelölje $\Gamma_{ij}$, ami az algoritmus során dinamikusan frissülni fog. Minden iterációban minden pontból elindul egy hangya, egy szomszédos pontba, melyet $n_{ij}$ és $\Gamma_{ij}$ alapján választanak ki. Így azon pontok lesznek előnyben, melyek

közelebb vannak, és magasabb a feromon szintjük. Minden iterációban frissítjük a feromonok értékét. Először egy tört értékkel szorozzuk be a régi nyomokat (legyen ez $i - \rho$, ahol $0 \leq \rho \leq 1$), hogy reprezentáljuk az idő múlásával a párolgást, majd a legutóbbi iterációban használt élekre új feromont fektetünk. Ha az $(i, j)$ élt a $k$-adik hangya használta, és az ő útjának a hossza $L_k$ volt, akkor a feromon értékét növeljük $\Delta_{ij}^k = \frac{1}{L_k}$-val. Így az élen lévő nyom frissítése:

$$\Gamma_{ij} = \rho\Gamma_{ij} + \sum_{k=1}^{N} \Delta_{ij}^k,$$

ahol $N$ jelenti a hangyák számát. A párolgás azt akadályozza meg, hogy a korábban kapott gyenge megoldások befolyásolják az eredményt.

Ez a módszer jó eredményeket ad, viszont nem lehet összehasonlítani a korábban megismert módszerekkel. Akkor hatásos csak igazán, ha más eljárásokkal vegyítjük össze. Így tettek azon cikkek írói is, akik a VRP-re próbálták az eljárást alkalmazni ([39], [40], [41]), de jelen dolgozatban nem térek rá ki.

# 4. Befejezés

A dolgozat írása során az volt a célom, hogy az algoritmusok elvét ismertessem. Az eljárások összehasonlítása azért nehéz, mert a fent leírt módszereket a legtöbbször különböző eseteken futtatták. Másik kihívás például a futásidők összemérése, hiszen a számítógépek sebessége rohamtempóban növekszik, ezért a néhány évvel ezelőtt mért futásidők mára már jelentőségüket vesztik. Összefoglalásképp azonban megállapíthatjuk, hogy az alap módszerek általában kevésbé adnak jó eredményeket, a legjobb megoldásokat úgy tudjuk elérni, ha több eljárást ötvözünk. Ezt láthattuk például az alsó korlát kereső algoritmusok esetében, amikor az alapvető ötleteket additív és diszjunktív módon összekapcsoltuk. Ezen kívül a felülről közelítő heurisztikák is egymásba ágyazhatóak, hiszen ahogy azt láthattuk, gyakran egy klasszikus heurisztikát használunk például egy metaheurisztikus módszer inicializálásához. Ezért a végső eredmény a főprogramon kívül attól is függ, hogy mennyire működnek jól a benne szereplő alprogramok. Így például hiába várunk jó eredményt egy újonnan kitalált módszertől, ha az nem dolgozik elég jó részeredményekkel.

Annak ellenére, hogy igyekeztem minél mélyebb betekintést nyújtani a vehicle routing problémába, meg kell jegyezni, hogy a dolgozat még így is csak a töredékét dolgozta fel a teljes problémakörnek. Ez egyrészt azért van így, mert a valóságban előforduló problémák jóval összetettebbek, mint amilyeneket most tárgyaltunk. Másrészt a leírt eljárásoknak még rengeteg változata és variációja létezik, melyek alkalmasak az eredmények javítására.

Összefoglalva a vehicle routing problémakör elég széleskörű ahhoz, hogy újabb és újabb kutatási területeket adjon a matematikusoknak.

# Hivatkozások


[1] Paolo Toth, Daniele Vigo. The Vehicle Routing Problem. Università degli Studi di Bologna, Bologna, Italy, 2002.

[2] G. Laporte, H. Mercure, Y. Norbert. An exact algorithm for the asymmetrical capacitated vehicle routing problem. Networks, 16:33-46, 1986.

[3] D. L. Miller. A matching based exact algorithm for capacitated vehicle routing problem. ORSA Journal on Computing, 7:1-9, 1995.

[4] H. N. Gabow and R. E. Tarjan. Efficient algorithms for a family of matroid intersection problems. Journal of Algorithms, 5:80-131, 1984.

[5] An exact algorithm for the capacitated shortest spanning arborescence. Annals of Operations Research, 61:121-142, 1995.

[6] N. Christophides, A. Mingozzi, P. Toth. Exact algorithms for the vehicle routing problem based on the spanning tree and shortest path relaxations. Mathematical Programming, 20:255-282, 1981.

[7] M. L. Fischer. Optimal solution of vehicle routing problems using minimum $k$-trees. Operations Research, 42:626-642, 1994.

[8] M. Fischetti, P. Toth. An additive bounding procedure for combinatorial optimization problems. Operations Research, 37:319-328, 1989.

[9] M. Fischetti, P. Toth, D. Vigo. A branch-and-bound algorithm for the capacitated vehicle routing problem on directed graph. Operations Research, 42:846-859, 1994.

[10] E. Hadjiconstantinou, N. Christophides, A. Mingozzi. A new exact algorithm for the vehicle routing problem based on $q$-paths and $k$-shortest paths relaxations. Annals of Operations Research, 61:21-43, 1995.

[11] P. Augerat. Approche Polyèdrale du Problème de Tournées de Véhicules. Ph.D. thesis, Institut National Polytechnique de Grenoble, France 1995.

[12] Y. Pochet. New valid inequalities for the vehicle routing problem. In preparation.

[13] D. Naddef and G. Rinaldi. The graphical relaxation: A new framework for the symmetric traveling salesman polytope. Mathematical Programming, 58:53-88, 1993.

[14] G. Laporte, Y. Norbert. Comb inequalities for the vehicle routing problem. Methods of Operations Research, 51:271-276, 1984.

[15] T. McCormick, M. R. Rao, G. Rinaldi. When is min cut with negative edges easy to solve? Easy and difficult objective functions for max cut. IASI-CNR Research Report. In preparation.

[16] A. V. Karzanov, E. A. Timofeev. Efficient algorithm for finding all minimal edge cuts of a nonoriented graph. Kibernetika, 2:8-12, 1986 (in Russian). Translated in Cybernetics, 22:156-162, 1986.

[17] D. Applegate, R. Bixby, V. Chvátal, and W. Cook. Solving travelling salesman problems. 15th International Symposium on Mathematical Programming, University of Michigan, Ann Arbor, MI, 1994.

[18] P. Augerat, J. M. Belenguer, E. Benavent, A. Corberán, D. Naddef, G. Rinaldi. Computational results with a branch and cut code for the capacitated vehicle routing problem. Technical Report RR 949-M, Université Joseph Fourier, Grenoble, France, 1995.

[19] J.-M. Clochard and D. Naddef. Use of path inequalities for TSP. In G. Rinaldi and L. Wolsey, editors, Proceedings of the Third Workshop on Integer Programming and Combinatorial Optimization, CORE, University of Louvain La Neuve, Belgium, 1993, pp. 291-312.

[20] G. Clarke, J. V. Wright. Scheduling of vehicles from a central depot to a number of delivery points. Operations Research, 12:568-581, 1964.

[21] T. J. Gaskell. Bases for vehicle fleet scheduling. Operational Research Quarterly, 18:281-295, 1967.

[22] P. Yellow. A computational modification to the savings method of vehicle scheduling. Operational Research Quarterly, 21:281-283, 1970.

[23] A matching based savings algorithm for the vehicle routing problem. Technical Report Cahiers du GERAD G-89-04, École des Hautes Études Commericales de Montréal, Canada, 1989.

[24] K. Altinkemer and B. Gavish. Parallel savings based heuristic for the delivery problem. Operations Research, 39:456-469, 1991.

[25] R. H. Mole and S. R. Jameson. A sequential route-building algorithm employing a generalized savings criterion. Operational Research Quarterly, 27:503-511, 1976.

[26] N. Christofides, A. Mingozzi, P. Toth. The vehicle routing problem. In N. Christofides, A. Mingozzi, P. Toth, and C. Sandi, editors, Combinatorial Optimization, Wiley, Chichester, UK, 1979, pp 315-338.

[27] B. E. Gillett and L. R. Miller. A heuristic algorithm for the vehicle dispatch problem. Operations Research, 22:340-349, 1974.

[28] M. L. Fischer, R. Jaikumar. A generalized assignment heuristic for the vehicle routing problem. Networks, 11:109-124, 1981.

[29] J. Bramel and D. Simchi-Levi. A location based heuristic for general routing problems. Operations Research, 43:649-660, 1995.

[30] M. Balinski and R. Quandt. On an integer program for a delivery problem. Operations Research, 12:300-304, 1964.

[31] B. A. Foster and D. M. Ryan. An integer programming approach to the vehicle scheduling problem. Operations Research, 27:367-384, 1976.

[32] D. M. Ryan, C. Hjorring, and F. Glover. Extensions of the petal method for the vehicle routing problem. Journal of Operational Research Society, 44:289-296, 1993.

[33] J. Renaud, F. F. Boctor, and G. Laporte. An improved petal heuristic for the vehicle routing problem. Journal of Operational Research Society, 47:329-336, 1996.

[34] A. Van Breedam. An analysis of the behavior of heuristics for the vehicle routing problem for a selection of problems with vehicle-related, customer-related, and time-related constraints. Ph. D. dissertation, University of Antwerp, 1994.

[35] I. H. Osman. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. Annals of Operations Research, 41:421-451, 1993.

[36] M. Gendreau, A. Hertz, and G. Laporte. New insertion and postoptimization procedures for the traveling salesman problem. Operations Research, 40:1086-1094, 1992.

[37] M. Gendreau, A. Hertz, and G. Laporte. A tabu search heuristic for the vehicle routing problem. Management Science, 40:1276-1290, 1994.

[38] A. Colorni, M. Dorigo, and V. Maniezzo. Distributed optimization by ant colonies. In F. Varela and P. Bourgine, editors, Proceedings of the European Conference on Artificial Life, Elsevier, Amsterdam, 1991.

[39] H. Kawamura, M. Yamamoto, T. Mitamura, K. Suzuki, and A. Ohuchi. Cooperative search on pheromone communication for vehicle routing problems. IEEE Transactions on Fundamentals, E81-A:1089-1096, 1998.

[40] B. Bullnheimer, R. F. Hartl, and C. Strauss. Applying the ant system to the vehicle routing problem. In S. Voss, S. Martello, I. H. Osman, and C. Roucairol, editors, Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization, Kluwer, Boston, MA, 1998, pp. 109-120.

[41] B. Bullnheimer, R. F. Hartl, and C. Strauss. An improved ant system for the vehicle routing problem. Annals of Operations Research, 89:319-328, 1999.

# Függelék