Eötvös Loránd Tudományegyetem

Természettudományi Kar

Ablonczy Tamás

Karbantartóutak szervezésének optimalizálása az OMSZ-nál

Szakdolgozat

Matematika BSc, elemző matematikus szakirány

Témavezető:
dr. Bérczi-Kovács Erika
Operációkutatási Tanszék

Budapest, 2019

Köszönöm

Sokakat illet köszönet ennek a diplomamunkának az elkészültéért. Elsősorban témavezetőmnek, Bérczi-Kovács Erikának köszönöm a segtíőkészségét. Köszönöm, hogy hitt bennem, lelket öntött belém, amikor én elbizonytalanodtam. Köszönöm a mentori iránymutatásokat, bátorításokat.

Köszönöm Bötkös Tamásnak, az Országos Meteorológiai Szolgálat munkatársának, hogy rendelkezésemre bocsátotta a feladat elvégzéséhez szükséges valós adatokat.

Köszönöm a bátyámnak, Ablonczy Dávidnak a lelkesedését a diplomamunkám iránt, köszönöm a sokféle életszerű észrevételt, és azt is, hogy felvillantotta merre lehetne továbbfejleszteni a projektet.

Köszönöm Madarasi Péternek, hogy rendkívüli segítőkészségével időt nem sajnálva segített a programozási nehézségekben.

Végül, de egyáltalán nem utolsó sorban, sőt inkább ezáltal kiemelve, köszönöm a feleségem és a családom áldozatkész támogatását, hogy ideális hátteret biztosítottak a tanulmányaim elvégzéséhez és ennek a dolgozatnak a megírásához.

Tartalomjegyzék

1. fejezet

Meteorológusok és a műszereik

1.1. Egy őszi nap

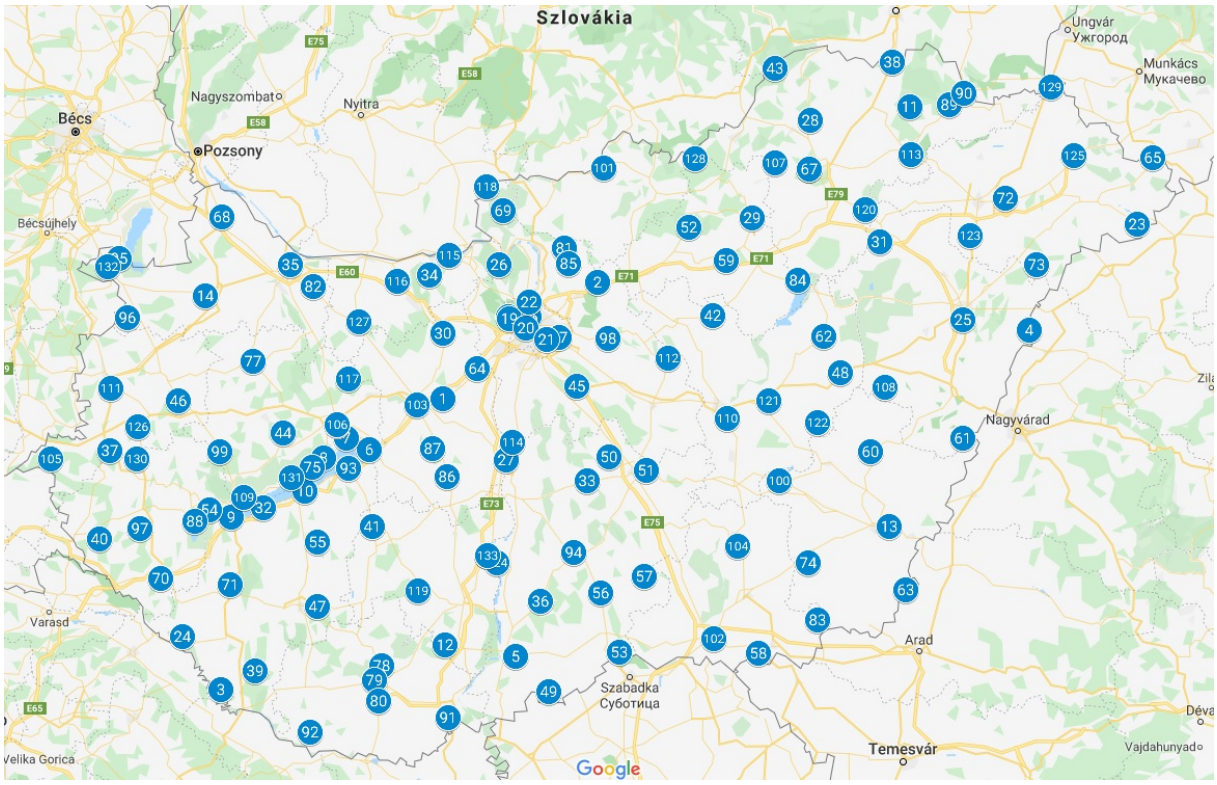
Másfél éve egy családi összejövetelen lelkesen meséltem az aktuális egyetemi projektemről. Azt kaptuk feladatként, hogy háromfős csoportokban keressünk egy életközeli problémát, amit aztán matematikai modellezéssel optimalizálunk. Felvillanyozott, hogy ezek alapján a tanárunk szerint az addig megszerzett tudásunk már elegendő lehet arra, hogy a gyakorlatban is kipróbáljuk mindazt, ami olyan elvontnak és még csak alapozónak tűnt a különböző órákon. Ahogy erről lelkendeztem, az idősebbik bátyámnak is felcsillant a szeme. Ő évekkel korábban az Országos Meteorológiai Szolgálatnál dolgozott, és ahogy elmondta, munkája során többször is adódtak olyan helyzetek, amiket próbáltak szemre optimalizálgatni, de nem jutottak eredményre. Szerinte ezek közül valamelyikkel lehetne foglalkozni, ez tényleg valós életből vett feladat lenne. A kis egyetemi csapatunk végül egy másik cég valós problémája mellett tette le a voksát, így ez a beszélgetés eltemetődött bennem. Aztán tavasszal egy másik kurzuson gyakorlatias optimalizálási feladatokat programoztunk, így a szakdolgozati témán gondolkodva újra előkerült a bátyám felvetése.

1.2. Amikor a nyers erő nem segít

Az Országos Meteorológiai Szolgálatnak (továbbiakban OMSZ) körülbelül 130 mérőállomása van Magyarország területén egyenletesen elosztva, ahogy az 1.1 ábra mutatja.

Ezeken a mérőállomásokon olyan műszerekkel figyelik az időjárást, amiket időről-időre ellenőrizni, újrakalibrálni, cserélni kell. Ezért a mérőállomásokat rendszeresen (általában évente) meglátogatja a budapesti központból az OMSZ egyik mérnöke, a használatban lévő műszert kicseréli egy kalibráltra, a másikat pedig visszaviszi Budapestre kalibrálásra. Ezek a műszerek meglehetősen drágák és térfogatra sem kicsik, így nincs belőlük feleslegesen sok.

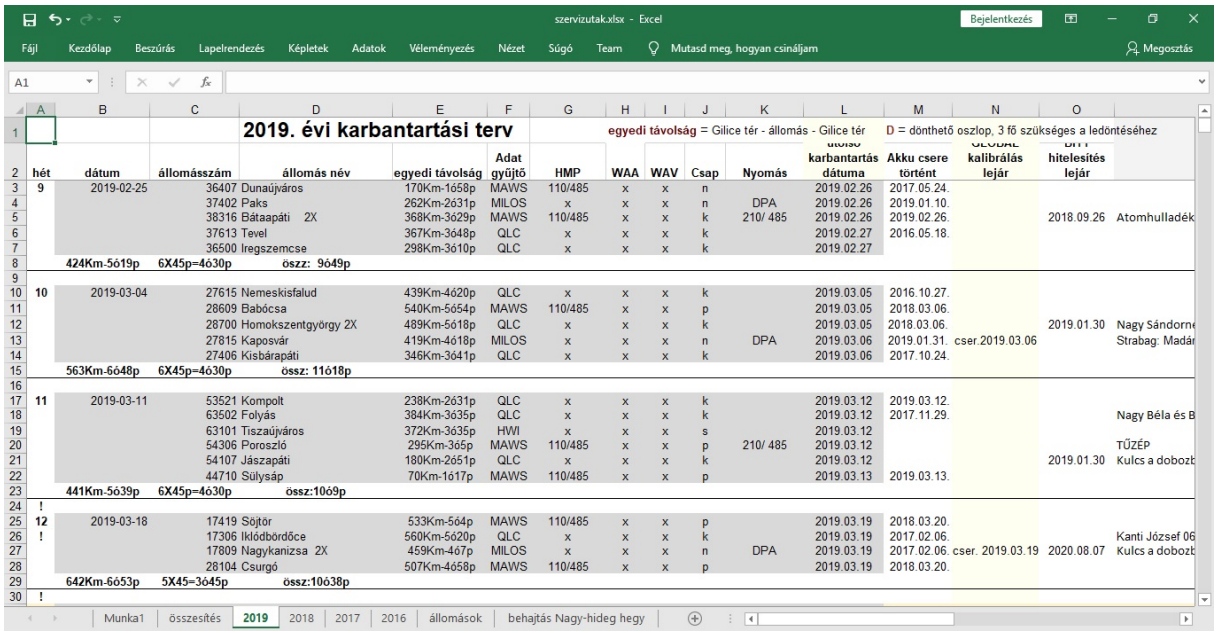
Többnyire egy nap alatt járnak be annyi állomást, amennyi belefér, de az ország távol eső pontjain lévő állomások miatt néha kétnapos utakat is tesznek. Egy nap legfeljebb 12 óra lehet a teljes munkaidő utazással, szereléssel együtt, bár ez túlórázást jelent, így a dolgozók jobban örülnek, ha ennél rövidebb egy munkanap. A szervizutak hosszát, pontosabban az egy szervizúton meglátogatott mérőállomások mennyiségét korlátozza az is hogy szűkös a készlet a használaton kívüli műszerekből.



1.1. ábra. OMSZ mérőállomások az országban

A szervizutaknak van egy évek óta kialakult, megszokáson alapuló jól bevált rendszere. Ez látható az 1.2 ábrán.

De felvetődött a kérdés, hogy lehetne-e ezt ennél optimálisabban szervezni, lehetne-e javítani ezen a rendszeren. Az laikus próbálkozásra is hamar kiderült, hogy brute force-szal itt nem lehet célt érni, hiszen az $\frac{(n-1)!}{2}$ lehetőséget jelent, ami már $n = 10$ esetén is több, mint 300.000 lehetőséget jelent, míg $n = 120$ esetén már $6.68 \cdot 10^{198}$ lehetséges útvonal, ami nyilván kezelhetetlen nyers erővel. Így jött a képbe a matematikai modellezés.



szervizutak.xlsx - Excel

2019. évi karbantartási terv

egyedi távolság = Gilice tér - állomás - Gilice tér D = dönthető oszlop, 3 fő szükséges a ledöntéséhez

	hét	dátum	állomásszám	állomás név	egyedi távolság	Adat gyűjtő	HMP	WAA	WAV	Csap	Nyomás	utolsó karbantartás dátuma	Akku csere történt	GLOBAL kalibrálás lejár	DPT hitelesítés lejár	
3	9	2019-02-25	36407	Dunaújváros	170Km-1658p	MAWS	110/485	x	x	n		2019.02.26	2017.05.24.			
4			37402	Paks	262Km-2631p	MILOS	x	x	x	n	DPA	2019.02.26	2019.01.10.			
5			38316	Bátaapáti 2X	368Km-3629p	MAWS	110/485	x	x	k	210/ 485	2019.02.26	2019.02.26.		2018.09.26	Atomhulladék
6			37613	Tevel	367Km-3648p	QLC	x	x	x	k		2019.02.27	2016.05.18.			
7			36500	Iregszemcse	298Km-3610p	QLC	x	x	x	k		2019.02.27				
8		**424Km-5619p**	**6X45p=4630p**	**össz: 9649p**												
9																
10	**10**	2019-03-04	27615	Nemeskisfalud	439Km-4620p	QLC	x	x	x	k		2019.03.05	2016.10.27.			
11			28609	Babócsa	540Km-5654p	MAWS	110/485	x	x	p		2019.03.05	2018.03.06.			
12			28700	Homokszentgyörgy 2X	489Km-5618p	QLC	x	x	x	k		2019.03.05	2018.03.06.		2019.01.30	Nagy Sándorné
13			27815	Kaposvár	419Km-4618p	MILOS	x	x	x	n	DPA	2019.03.06	2019.01.31.	cser.2019.03.06		Strabag: Madár
14			27406	Kisbárapáti	346Km-3641p	QLC	x	x	x	k		2019.03.06	2017.10.24.			
15		**563Km-6648p**	**6X45p=4630p**	**össz: 11618p**												
16																
17	**11**	2019-03-11	53521	Kompolt	238Km-2631p	QLC	x	x	x	k		2019.03.12	2019.03.12.			
18			63502	Folyás	384Km-3635p	QLC	x	x	x	k		2019.03.12	2017.11.29.			Nagy Béla és B
19			63101	Tiszaújváros	372Km-3635p	HWI	x	x	x	s		2019.03.12				
20			54306	Poroszló	295Km-365p	MAWS	110/485	x	x	p	210/ 485	2019.03.12				TŰZÉP
21			54107	Jászapáti	180Km-2651p	QLC	x	x	x	k		2019.03.12			2019.01.30	Kulcs a dobozb
22			44710	Súlysáp	70Km-1617p	MAWS	110/485	x	x	p		2019.03.13	2019.03.13.			
23		**441Km-5639p**	**6X45p=4630p**	**össz:10609p**												
24	**!**															
25	**12**	2019-03-18	17419	Sójtór	533Km-564p	MAWS	110/485	x	x	p		2019.03.19	2018.03.20.			
26	**!**		17306	Iklódbördőce	560Km-5620p	QLC	x	x	x	k		2019.03.19	2017.02.06.			Kanti József 06
27			17809	Nagykanizsa 2X	459Km-467p	MILOS	x	x	x	n	DPA	2019.03.19	2017.02.06.	cser. 2019.03.19	2020.08.07	Kulcs a dobozb
28			28104	Csurgó	507Km-4658p	MAWS	110/485	x	x	p		2019.03.19	2018.03.20.			
29		**642Km-6653p**	**5X45=3645p**	**össz:10638p**												
30	**!**															

Munka1 | összesítés | 2019 | 2018 | 2017 | 2016 | állomások | behajtás Nagy-hideg hegy

1.2. ábra. OMSZ szervizutak 2019-ben

2. fejezet

Ügynökök

A megoldandó feladat egy klasszikus matematikai problémához hasonlít, ami az utazóügynökökről (traveling salesman problem - TSP) kapta a nevét. A feladat arról szól, hogy ha egy ügynök kap egy listát azokról a helyekről, ahova el kell mennie, akkor hogyan tudja ezeket a helyeket a legrövidebb úton végigjárni. Nem lehet tudni pontosan, ki foglalkozott először ezzel a feladattal, de Hamilton és Kirkman öntötték először matematikai formába a 19. század első felében. Ahogy ez már fentebb is szóba került, a feladat nagyon hamar, kevés meglátogatandó helyszín esetén rendkívül bonyolulttá válik, így nem meglepő, hogy a még az '50-es években is kevesebb, mint 50 városra tudták megmondani az optimális megoldást, és csak a számítástechnika fejlődésével párhuzamosan bővült ez a szám a 2000-es években ötszámjegyűvé. [1]

2.1. Az utazóügynök probléma

Mielőtt a problémát matematikai nyelven fogalmaznám meg, először néhány egyszerű fogalmat tisztázok.

2.1.1. Definíció. *A **teljes gráf** olyan egyszerű gráf, amelynek minden csúcsa össze van kötve minden más csúccsal.*

2.1.2. Definíció. *A **kör** élek olyan egymáshoz csatlakozó sorozata, amelyben az élek és pontok egynél többször nem szerepelhetnek, és a kiindulási pont megegyezik a végponttal.*

2.1.3. Definíció. ***Hamilton-kör**nek nevezünk egy kört egy gráfban, ha a gráf összes csúcsán pontosan egyszer halad át.*

Ezek után az utazóügynök probléma: egy n csúcsú teljes gráfon keressük a legrövidebb Hamilton-kört. Jelölje $t(uv)$ az u és v csúcsok távolságát, és legyen V a csúcshalmaz. Első próbálkozásként tekintsük a következő bináris feladatot:

[1] https://hu.wikipedia.org/wiki/Az_utazó_ügynök_problémája

$$min \sum_{(u,v)\in A} t_{uv}x_{uv} \tag{tsp0}$$

$$x_{uv} \in 0,1 \qquad \forall (u,v) \in A \tag{tsp1}$$

$$\sum_{u=1}^{n} x_{uv} = 1 \qquad \forall v \in V \tag{tsp2}$$

$$\sum_{v=1}^{n} x_{uv} = 1 \qquad \forall u \in V \tag{tsp3}$$

A Hamilton-körök ugyan megoldások, de sajnos több diszjunkt kör uniója is lehet megoldás, ha együtt az összes várost tartalmazzák. Ezért még hozzá kell venni az összefüggőséget biztosító egyenlőtlenségeket. Erre többféle megoldás is ismert, ezek közül az egyik az alábbi.

2.2. Miller-Tucker-Zemlin formula

A Miller, Tucker és Zemlin által kidolgozott modell [2] egy folyam alapú algoritmus. Ennek az a lényege, hogy a folyam értéke kezdetben n (csúcsszámnyi), utána pedig minden csúcsnál csökken eggyel, így ha a kör végén 0-ra csökken, akkor minden csúcsot végigjártam, azaz Hamilton kört kaptunk. Ezzel tehát kizárható, hogy a megoldás diszjunkt körök uniója legyen. Ez képletszerűen formalizálva:

$$f_i \in \mathbb{Z} \qquad i = 2, ..., n \tag{mtz1}$$

$$f_i - f_j + nx_{ij} \leq n - 1 \qquad 2 \leq i \neq j \leq n \tag{mtz2}$$

$$0 \leq f_i \leq n - 1 \qquad 2 \leq i \leq n \tag{mtz3}$$

Ezzel a modellezéssel már el lehetett kezdeni dolgozni a konkrét feladatom megoldásán, így a folytatásban ennek a lépéseit ismertetem.

[2] Miller

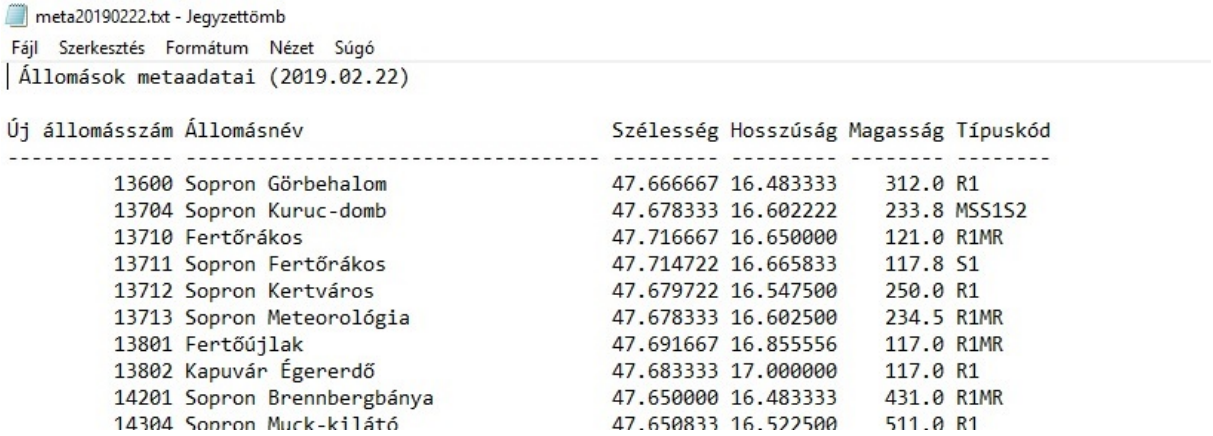
3. fejezet

Adatbányászat: Google Distance Matrix

A fenti elméleti bevezetésből látszódik, hogy a konkrét feladatot azzal kellett kezdeni, hogy létrehoztunk egy teljes gráfot az OMSZ mérőállomások, mint csúcsok között. Ehhez a Google Map Distance Matrix [1] szolgáltatását használtam.

3.1. Adatkonvertálás

A Google Map Distance Matrix szolgáltatásához az OMSZ szervizelő mérnökeitől kapott állomáslistát használtam. Ez a lista azonban egy meglehetősen nyers .txt fájlban volt, ahogy a mellékelt ábra mutatja.



meta20190222.txt - Jegyzettömb

Fájl Szerkesztés Formátum Nézet Súgó

Állomások metaadatai (2019.02.22)

Új állomásszám	Állomásnév	Szélesség	Hosszúság	Magasság	Típuskód
13600	Sopron Görbehalom	47.666667	16.483333	312.0	R1
13704	Sopron Kuruc-domb	47.678333	16.602222	233.8	MSS1S2
13710	Fertőrákos	47.716667	16.650000	121.0	R1MR
13711	Sopron Fertőrákos	47.714722	16.665833	117.8	S1
13712	Sopron Kertváros	47.679722	16.547500	250.0	R1
13713	Sopron Meteorológia	47.678333	16.602500	234.5	R1MR
13801	Fertőújlak	47.691667	16.855556	117.0	R1MR
13802	Kapuvár Égererdő	47.683333	17.000000	117.0	R1
14201	Sopron Brennbergbánya	47.650000	16.483333	431.0	R1MR
14304	Sopron Muck-kilátó	47.650833	16.522500	511.0	R1

3.1. ábra. OMSZ mérőállomások nyers adatai

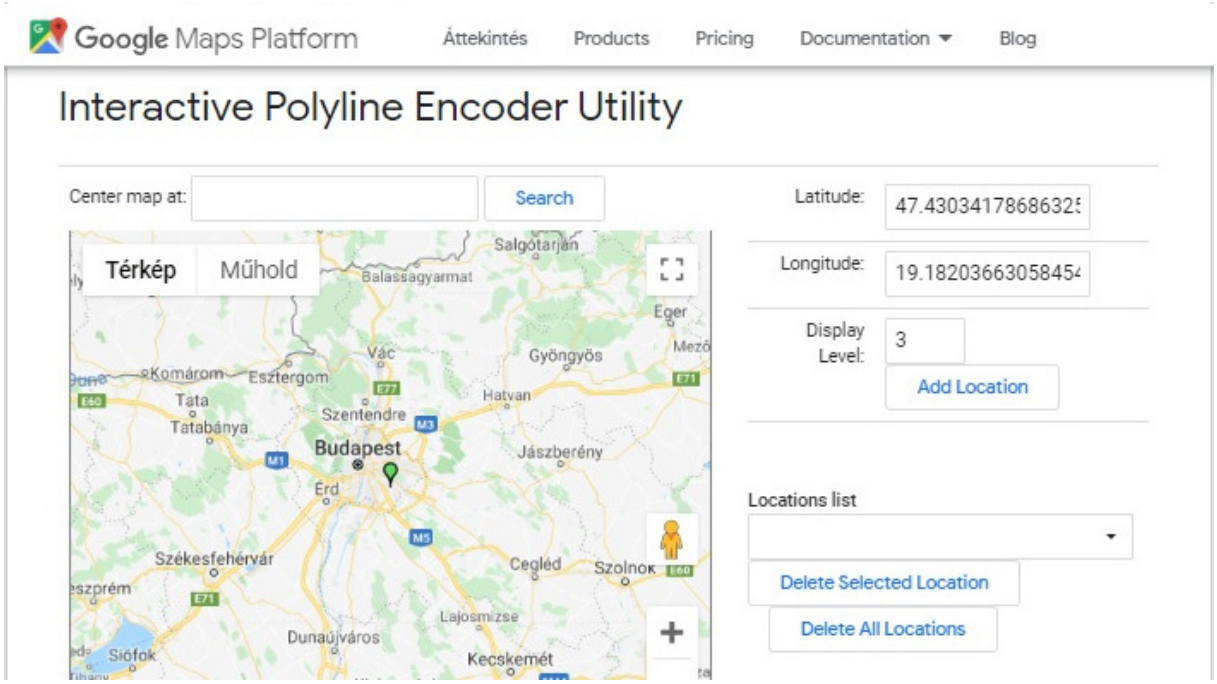
Ezt a szöveges fájlt először átkonvertáltam Excelbe. Utána ebből a több, mint 750 adatsorból kellett kiszűrni azokat az állomásokat, amelyeket a szervizutakon végiglátogatnak. Ezt elméletileg egyszerű lett

[1] https://cloud.google.com/maps-platform/

volna megtenni, mert elvileg az S1 típuskód jelöli a szükséges állomásokat. Azonban kaptam egy másik táblázatot is, ami pedig a szervizutakat listázza.

Ezzel a listával összevetve az S1 típuskódú állomások listáját, kiderült, hogy bőven vannak eltérések: vannak hiányzók és vannak többletek is. Ráadásul a szervizutas listában nem minden esetben ugyanaz az állomások megnevezése, mint a nyers adattáblázatban, így névre sem lehetett magától értetődően szűrni. Az állomásszám bizonyult a legbiztosabb összekötő attributumnak, de ez meg néhány helyen hiányzik a szervizutas listából. Így végeredményben egyesével végig kellett bogarászni a táblázatokat, hogy ki lehessen szűrni a szükséges adatsorokat.

Ezután kezdhettem el használni a Google Distance Matrix szolgáltatását. Itt azonban többféle korlátozás van, az egyik az adatlekérő kód hosszára vonatkozik. Ahhoz, hogy hatékonyan tudjam használni a szolgáltatást, az állomások koordinátáit át kellett konvertálni a Google interaktív felületén.

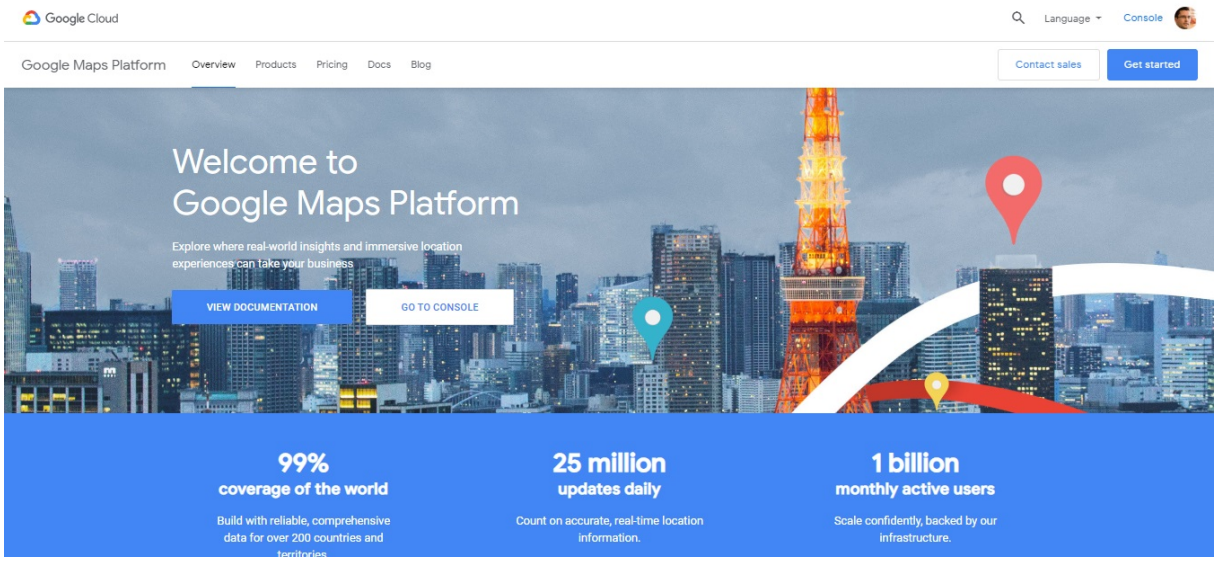


3.2. ábra. koordináták konvertálása

3.2. Adatok rendezése

Csak ezt követően tudtam nekikezdeni a Google Distance Matrix használatának. Ez szerencsére havi 40.000 lekérdezésig ingyenes, nekem pedig 130 állomást kellett egymással összekötni, azaz szűk 20.000 adatra volt szükségem, így ezen a ponton nem ütköztem korlátozásba. Azonban egy lekérdezéssel legfejebb 10 kezdőpontot és 10 végpontot lehet összekötni, azaz ez is meglehetősen időigényes munka volt. Főképp, hogy az életszerűség érdekében meghagytam a lehetőségét annak, hogy asszimetrikus legyen a gráf, azaz két állomás között nem feltétlenül ugyanakkora a távolság egyik, illetve a másik irányba.

A kapott .json fájlokból szintén ki kellett nyerni a szükséges információt, így ezeket is átkonvertáltam



3.3. ábra. Google Distance Matrix API

két excel táblázatba: az egyikben a km-ben megkapott távolságokat rögzítettem, a másikban pedig az utazási időt. (Még ehhez is kellett egy kis mértékegységváltás, mert a Google Distance Matrix alapértelmezetten SI mértékegységekben, azaz méterben és másodpercben adja meg az adatokat.)

4. fejezet

Első nekifutás: klónozott ügynökök

Mikor az adatok megfelelően kezelhető formában rendelkezésre álltak, azaz felépült a teljes gráf, akkor tudtam nekikezdeni az érdemi modellezésnek, majd a programozásnak.

Ahogyan a fejezet címében is utaltam erre, azt terveztem, hogy először egy TSP-t implementálok, majd klónozom az utazóügynököt, azaz az általam ismert TSP-t terjesztem ki úgy, mintha több kis részgráfon egy-egy utazóügynök-problémát kellene megoldani.

4.1. Modellezés Gusekben

A tanulmányaim során korábban használtam a Gusek nevű ingyenes IP solvert, így ezt már ismertem valamennyire, ezért ebben kezdtem el programozni, a fenti modellt implementálni. Ebből született az alábbi program:

```
param n, integer, >= 3      /* csúcsok száma */
set V := 1..n;              /* csúcsok halmaza */
set E, within V cross V;    /* élek halmaza */
param c{(i,j) in E};        /* i csúcs távolsága j-től */

var x{(i,j) in E}, binary;  /* x[i,j] = 1 azt jelenti, hogy az i-ből j-be
                               menő élt használjuk */
var y{(i,j) in E}, >= 0;    /* y[i,j] a folyam értéke az ij élen */

minimize total: sum{(i,j) in E} c[i,j] * x[i,j];

s.t. leave{i in V}: sum{(i,j) in E} x[i,j] = 1; /* az ügynök mindenhova */
s.t. enter{j in V}: sum{(i,j) in E} x[i,j] = 1; /* egyszer megy*/

s.t. cap{(i,j) in E}: y[i,j] <= (n-1) * x[i,j];
/* a folyam értéke minden élen legfeljebb n-1 lehet */
```

```
s.t. node{i in V}:  /* ez a konzervativitási feltétel */
      sum{(j,i) in E} y[j,i]    /* i csúcsba érkező összes folyamérték */
      + (if i = 1 then n)       /* kezdőcsúcsnál n */
      =
      sum{(i,j) in E} y[i,j]    /* i csúcsból induló összes folyamérték */
      + 1;                      /* az i-edik csúcsnál eggyel csökken
                                   a folyamérték */
solve;
```

Ez a program tizenegynéhány csúcsra még ki tudja számolni az optimumot, de már elkezd belassulni, ami előrevetítette, hogy tízszer ennyi csúccsal a bonyolultabb VRP már nem fog tudni értelmezhető időn belül lefutni.

4.2. Váltás CPLEX-re

A Gusek korlátai miatt keresnem kellett egy hatékonyabb solvert. Némi utánajárással kiderült, hogy a legjobbként számon tartott CPLEX-hez egyetemi hallgatóként ingyenes hozzáférést kaphatok, amit sikerült is elintézni.

Ezt viszont még nem ismertem, így első lépésként meg kellett ismerkednem a programozói környezettel és a nyelvvel annyira, hogy aztán tudjam is használni. Bár a CPLEX C alapú nyelv, azért végig kellett csinálni néhány tutorialt, amíg belejöttem.

Ezután először átimplementáltam a TSP-s folyamalgoritmust CPLEX-re. A gyorsulás látványos volt, 50 csúcsra is pár másodperc alatt megadta az optimumot. Így magabiztosan vágtam neki, hogy a fenti módon klónozzam az ügynökömet.

Ennek a lényege, hogy részgráfonként oldok meg egy-egy TSP-t, ezeket összegzem, és az összeget minimalizálom. Ehhez az élek és az éleken értelmezett folyam dimenzióját eggyel (a szervizutak dimenziójával) növeltem, valamint bevezettem még egy bináris változót, ami azt jelzi, hogy egy adott csúcsot melyik szervizúton érintik. Ezek a bővítések egyébként is szükségszerűnek tűntek, hiszen egyszerre több részgráfon terveztem kezelni a TSP-t, és valahogyan mindenképpen meg kellett különböztetni ezeket.

Viszont ha csak ennyit tennék, akkor a megoldás egyetlen Hamilton kör lenne, hiszen a háromszög-egyenlőtlenség miatt biztosan hosszabb két csúcs között előbb még visszamenni a depót reprezentáló csúcsba. Ezért szükséges még korlátozni, hogy egy-egy szervizút legfeljebb milyen hosszú lehet.

Ennek eredménye az alábbi program:

```
using CPLEX;

int n = ...; // állomások
range A = 1..n;

int km[A][A] = ...;
int ora[A][A] = ...;
int ido[A] = ...;
```

```
int     s = ...; // szervizutak
range   S = 1..s;
int  T = ...; // időkorlát egy útra

dvar boolean u[A][A][S]; // u[i][j]=1 jelentése: az i,j állomások közötti
                         // közvetlen utat használjuk a k-adik szervizúton
dvar int f[A][A][S]; // folyamérték

dvar boolean w[A][S]; // w[i][k]=1 jelentése: i-edik állomás a k-adik
                       // szervizútra került

minimize sum(i in A, j in A, k in S) (31*km[i][j]+5000*ora[i][j])*u[i][j][k];
/*
nettó fizetés: 250.000 Ft
szuperbruttó: 250.000 * (1+0,15+0,1+0,085)*(1+0,17+0,015) = kb. 400.000 Ft
munkaidő: 160 óra
két ember dolgozik
órabér: 400.000 / 160 *2 = 5.000 Ft

üzemanyagár: 405 Ft (dízel)
fogyasztás: 7,6 l / 100 km
km-ár: 7,6 * 405 / 100 = kb. 31 Ft
*/


subject to {

forall (k in S) w[1][k] == 1; // a bázis minden szervizútnak legyen
                               // a része
forall (i in 2..n) sum(j in A, k in S) u[j][i][k] == 1;
// a bázis kivételével minden állomásra egyszer megyünk be és ki

forall (i in 2..n) sum(j in A, k in S) u[i][j][k] == 1;
forall (k in S, i in A) sum(j in A) u[i][j][k] == sum(j in A) u[j][i][k];

forall (i in 2..n) sum(k in S) w[i][k] == 1;
// a bázis kivételével minden állomást pontosan egy szervizúton látogatunk meg

sum(i in A, k in S) u[1][i][k] == s;
```

```
forall (k in S) sum(i in A, j in A) w[i][k]*u[i][j][k]*(ora[i][j]+ido[i]) <= T;
// minden szervizút az időkorlátnál rövidebb

forall (i in A, j in A, k in S) f[i][j][k] >= 0;
// minden állomáson nemnegatív a folyamérték

forall (i in A, j in A, k in S) f[i][j][k] <= (n-1) * u[i][j][k];

forall (k in S, i in A) if (i != 1)
    sum(j in A) f[j][i][k]-sum(j in A) f[i][j][k] == w[i][k];
};
```

Minden CPLEX iránti kezdeti lelkesedésem ellenére azonban hamar kiderült, hogy a TSP-ről VRP-re való módosítás annyival elbonyolította a programot, hogy már tizenegynéhány csúcsnál újra abba ütköztem, hogy a futási idő kezelhetetlenül hosszúra nyúlt.

5. fejezet

Egy gyorsabbnak tűnő lehetőség

A folyamalapú algoritmus kudarca miatt megint valami újat kellett keresni. Az egyik lehetőség az lett volna, hogy bevett heurisztikákat kezdek el alkalmazni, azonban túl hamar, túl kevés csúcsnál felmondta a szolgálatot a solver, így egyértelművé vált, hogy valamilyen hatékonyabb algoritmusra lesz szükség.

5.1. Ügynökök másképpen

Fentebb már utaltam rá fél szóval, hogy a TSP-nél a diszjunkt körök unióját többféle modellezéssel is ki lehet zárni. Ennek egyik változata a 2.2. pontban ismertetett Miller-Tucker-Zemlin formula. Egy másik megközelítést dolgozott ki Dantzig, Fulkerson és Johnson [1] . Ez a modell azt zárja ki, hogy a kezdőcsúcsot kivéve a többi csúcs semmilyen kombinációja nem alkot kört. Ennek a képlete:

$$\sum_{u \in Q} \sum_{v \in Q} x_{uv} \leq |Q| - 1 \qquad Q \subsetneq \{1, ..., n\}, |Q| \geq 2 \tag{dfj}$$

Azonnal feltűnő, hogy ez exponenciálisan sok feltétel, amit nem érdemes egy modellbe bevenni, mert túl lassú lesz a megoldó algoritmus, tehát ezzel kezdeni kell még valamit. Ennek a kezeléséhez még néhány fogalmat bevezetünk.

5.1.1. Definíció. *Legyen $A \in \mathbb{R}^{nxm}$ mátrix, $b \in \mathbb{R}^m$ és $c \in \mathbb{R}^n$ vektorok adottak. Ekkor keressük azt az $x \in \mathbb{R}^n$ vektort, melyre cx maximális, feltéve, hogy $Ax \leq b$. Ezt a feladatot lineáris programnak (**LP**) nevezzük.*

5.1.2. Definíció. *Ha $x \in \mathbb{Z}^n$ Akkor a fenti feladatot egészértékű lineáris programnak (integer linear program - **IP**) nevezzük.*

5.1.3. Definíció. *Ha egy IP-ből elhagyjuk az egészértékűségi feltételt, akkor azt az ő **LP relaxált**jának nevezzük.*

5.1.4. Definíció. *A **szimplex módszer** egy olyan algoritmus, ami egy LP-ről eldönti, hogy megoldható-e, és ha igen, akkor megadja az optimális megoldást is.*

[1] https://en.wikipedia.org/wiki/Vehicle_routing_problem

Mivel a TSP IP feladattal modellezhető, viszont a TSP Dantzig-Fulkerson-Johnson féle formalizációja még túl komplex, így az alábbiakban szükséges bemutatni még egy IP-ket megoldó módszert is.

Vágósíkos eljárás

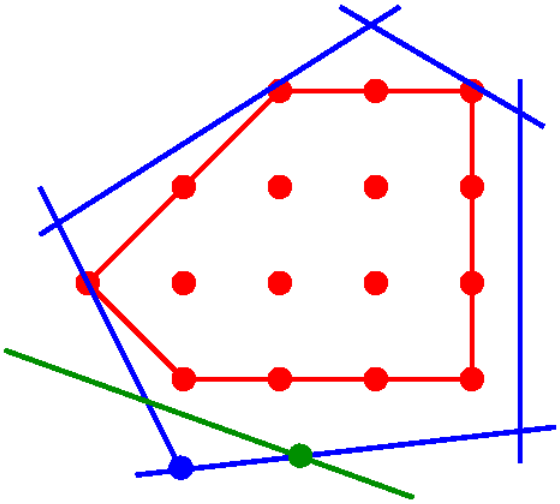
Ennek az eljárásnak a célja a $max\{cx : Ax \leq b, x \in \mathbb{Z}^n\}$ optimalizálási feladat megoldása. Legyen $P = \{x \in \mathbb{R}^n : Ax \leq b\}$. Az eljárás lépései a következők:

1. Oldjuk meg a $max\{cx : x \in P\}$ feladatot (például szimplex módszerrel)!

2. Ha az x^* optimális megoldás egész, akkor készen vagyunk.

3. Ha x^* nem egész, akkor keressünk olyan $\alpha x \leq \beta$ egyenlőtlenséget, amire $\alpha x^* > \beta$ de $\alpha x \leq \beta$ $\forall x \in P \cap \mathbb{Z}^n$.

4. Legyen $P := P \cap \{x : \alpha x \leq \beta\}$, és lépjünk az 1. pontra.



5.1. ábra. A zöld vágósík elválasztja a kék LP optimumot a lehetséges megoldások piros színnel jelölt politópjától. A következő iterációban a zöld pont is LP optimum lesz.[3]

A fenti algoritmus ebben a formában nem feltétlenül véges, még két dimenzióban sem. A másik probléma, hogy hogyan találunk az algoritmus 3. pontjában egy megfelelő $\alpha x \leq \beta$ egyenlőtlenséget. Erre ad egy lehetséges megoldást az itt és most nem részletezett Gomory-vágás.

Ha a Dantzig-Fulkerson-Johnson modellt szeretnénk alkalmazni, már az elején egy problémába ütközünk: ahogy már említettük, a (dfj) típusú egyenlőtlenségekből exponenciálisan sok van, ezért a szimplex módszerben exponenciális méretű mátrixokkal kellene dolgozni, ami túl lassú. Ezért már az LP relaxáció megoldásához is egy vágósíkos eljárást használunk:

1. Kezdetben a (tsp0) - (tsp3) feladatot oldjuk meg (például szimplexszel). Ha a megoldásra teljesül (dfj), akkor készen vagyunk. Ez utóbbit úgy ellenőrizhetjük, ha keresünk egy minimális vágást: az élekre kapacitásként x-et írunk, rögzítünk egy s pontot, majd minden t-re folyamalgoritmussal keresünk minimális vágást.

[3] https://commons.wikimedia.org/wiki/File:Cutting_plane_algorithm2.png

2. Ha (dfj)-nek van olyan részfeltétele, ami nem teljesül, akkor vegyük hozzá ezt a sértő feltételt az eddigiekhez, és kezdjük elölről.

Megmutatható, hogy $n-1$ maximális folyam kereséssel eldönthető, hogy x^* teljesíti-e az összes (dfj) típusú egyenlőtleséget, és ha nem, található egy olyan, amit nem teljesít. Ezt hozzávéve az eddigiekhez, újra megoldjuk az LP-t, és ezt addig folytatjuk, amíg a (dfj) feltételek nem teljesülnek. Ha a végén kapott megoldás nem egész, akkor alkalmazhatunk Gomory-vágást, vagy korlátozást és szétválasztást. [4]

5.2. Második nekifutás

A Dantzig-Fulkerson-Johnson féle Subtour-modell implementálásánál nem próbáltam meg eleve kizárni a kis köröket, hanem ha futás közben, azt látja a program, hogy a kapott optimum nem Hamilton kör, hanem diszjunkt körök uniója, akkor ezeket a konkrét köröket kizárva folytatja a keresést. (Pontosabban a talált kiskörök (subtour-ok) közül csak a legrövidebbet zárja ki a további keresésből.) Egy ügynök esetére ezt mutatja be az alábbi program:

```
/*************************************************************************
* ADATOK
*************************************************************************/
int     n       = ...; // allomasok
range   Allomasok  = 1..n;

tuple     ut        {int i; int j;} // úthálózat
setof(ut) Utak       = {<i,j> | ordered i,j in Allomasok};
int       km[Utak] = ...;

dvar boolean x[Utak];

tuple Subtour { int size; int subtour[Allomasok]; }
{Subtour} subtours = ...;

/*************************************************************************
* MODEL
*************************************************************************/
minimize sum (<i,j> in Utak) km[<i,j>]*x[<i,j>];
subject to {
   forall (j in Allomasok) // minden állomás két másikkal van összekötve
  sum(<i,j> in Utak) x[<i,j>]+sum(<j,k> in Utak) x[<j,k>] == 2;

   forall (s in subtours) // subtour eliminálás
      sum(i in Allomasok : s.subtour[i] != 0)
```

[4] Frank, *Operációkutatás*, 191-194.

```
      x[<minl(i, s.subtour[i]), maxl(i, s.subtour[i])>] <= s.size-1;
};

/*************************************************************************
* POST-PROCESSING
*************************************************************************/
int thisSubtour[Allomasok]; // tájékoztató adatok a megoldásról
int newSubtourSize;
int newSubtour[Allomasok];

int visited[i in Allomasok] = 0;
setof(int) adj[j in Allomasok] = {i | <i,j> in Utak : x[<i,j>] == 1}
   union {k | <j,k> in Utak : x[<j,k>] == 1};
execute {
  newSubtourSize = n;
  for (var i in Allomasok) { // Find an unexplored node
    if (visited[i]==1) continue;
    var start = i;
    var node = i;
    var thisSubtourSize = 0;
    for (var j in Allomasok)
      thisSubtour[j] = 0;
    while (node!=start || thisSubtourSize==0) {
      visited[node] = 1;
      var succ = start;
      for (i in adj[node])
        if (visited[i] == 0) {
          succ = i;
          break;
        }

      thisSubtour[node] = succ;
      node = succ;
      ++thisSubtourSize;
    }

    writeln("Found subtour of size : ", thisSubtourSize);
    if (thisSubtourSize < newSubtourSize) {
      for (i in Allomasok)
        newSubtour[i] = thisSubtour[i];
        newSubtourSize = thisSubtourSize;
```

```
    }
  }
  if (newSubtourSize != n)
    writeln("Best subtour of size ", newSubtourSize);
}

/*************************************************************************
* SCRIPT
*************************************************************************/
main {
    var opl = thisOplModel
    var mod = opl.modelDefinition;
    var dat = opl.dataElements;

    var status = 0;
    var it =0;
    while (1) {
        var cplex1 = new IloCplex();
        opl = new IloOplModel(mod,cplex1);
        opl.addDataSource(dat);
        opl.generate();
        it++;
        writeln("Iteration ",it, " with ", opl.subtours.size, " subtours.");
        if (!cplex1.solve()) {
            writeln("ERROR: could not solve");
            status = 1;
            opl.end();
            break;
        }
        opl.postProcess();
        writeln("Current solution : ", cplex1.getObjValue());

        if (opl.newSubtourSize == opl.n) {
          opl.end();
          cplex1.end();
          break; // not found
        }

        dat.subtours.add(opl.newSubtourSize, opl.newSubtour);
opl.end();
cplex1.end();
```

```
    }

    status;
}
```

Ez a gyorsabbnak tűnő lehetőség végül zsákutcának bizonyult két okból is. Egyrészről kis méretű TSP-n ugyan gyorsabb, mint a folyamos, de nagyobb gráfokon már megfordulnak az erőviszonyok. Másrészről első nekifutásra magamtól nem is sikerült klónozni az ügynököt. Utánajárva kiderült, hogy ez másnak sem ment egyszerűen, és csak több módosítás eredményeként sikerült polinomiális méretű IP-t készíteni.

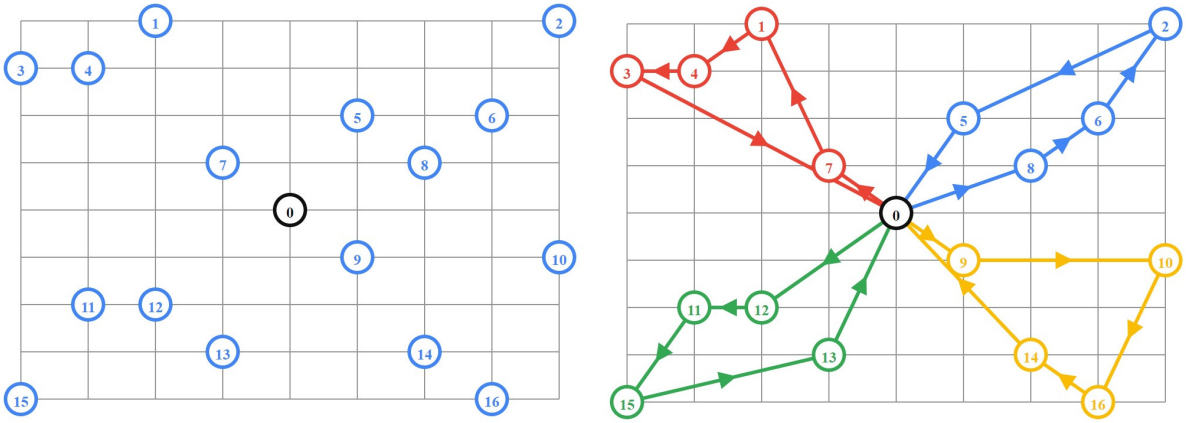
6. fejezet

Klónozás helyett járművek

Az ügynökklónozás kudarca vezetett el arra a felfedezésre, hogy ennek az általam megoldani próbált feladatot nem feltétlenül TSP-ként kell kezelni, hanem ennek külön neve van.

6.1. A járműútvonal-tervezési probléma

A TSP egy fokkal bonyolultabb változata, általánosítása a járműútvonal-tervezési probléma (vehicle route problem - VRP). Ennél már az a kérdés, hogy ha egy cégnek több ügynöke - vagy még inkább: járműve, jellemzően teherautója - is van, akkor melyik merre menjen, hogy minél optimálisabb legyen a végeredmény. (Ha csak egyetlen jármű van, abban a speciális esetben TSP-ről van szó.)



6.1. ábra. VRP[1]

A VRP először George Dantzig és John Ramser munkájában tűnik fel 1959-ben [2] , melyben megírták és alkalmazták az első algoritmusos megközelítést üzemanyag szállítóknál. Gyakori kontextusa a VRP-nek, mikor a szállítmányok a vevőkör középponti lerakatában helyezkednek el.

[1] https://developers.google.com/optimization/routing/vrp
[2] Dantzig

Az alapfeladat itt is hasonló, mint a TSP esetében, csak az élekhez rendelt bináris változó egy dimenzióval (a szervizutak dimenziójával) bonyolódik, valamint a kezdőcsúcs befoka és kifoka 1 helyett m (szervizutak száma):

$$min \sum_{(u,v)\in A, k\in K} t_{uv} x_{uv}^k \tag{vrp0}$$

$$x_{uv}^k \in 0,1 \qquad \forall (u,v) \in A, k \in K \tag{vrp1}$$

$$\sum_{u=1}^{n}\sum_{k=1}^{m} x_{uv}^k = 1 \qquad \forall v \in 2..n \tag{vrp2}$$

$$\sum_{v=1}^{n}\sum_{k=1}^{m} x_{uv}^k = 1 \qquad \forall u \in 2..n \tag{vrp3}$$

$$\sum_{u=1}^{n}\sum_{k=1}^{m} x_{u1}^k = m \tag{vrp4}$$

$$\sum_{v=1}^{n}\sum_{k=1}^{m} x_{1v}^k = m \tag{vrp5}$$

$$\sum_{u=1}^{n} x_{uv}^k - \sum_{w=1}^{n} x_{vw}^k = 0 \qquad \forall v \in V \forall k \in K \tag{vrp6}$$

Jó kérdés viszont, hogy mit értünk optimális végeredmény alatt? Ha csak a járművek által megtett össztávolságot szeretnék a lehető legkisebbre leszorítani, és nincs más egyéb megkötés, akkor az a megoldás, hogy elég egyetlen járműre bízni, hogy járja végig az állomásokat a lehető legrövidebb úton. Azaz egy TSP-t kapunk - de valószínűleg nem ez a cél.

6.2. Variációk egy témára

A fenti kérdésre adott válasz alapján sokféle fajtáját különböztethetjük meg a VRP-nek. A teljesség igénye nélkül ezt mutatja az alábbi 6.2 ábra.

Egy gyakori eset, hogy a járművekbe nem fér akármennyi áru, korlátos a rakterük (capacity constrained VRP - CVRP). Ekkor a feltételek sora az alábbival bővül [3] :

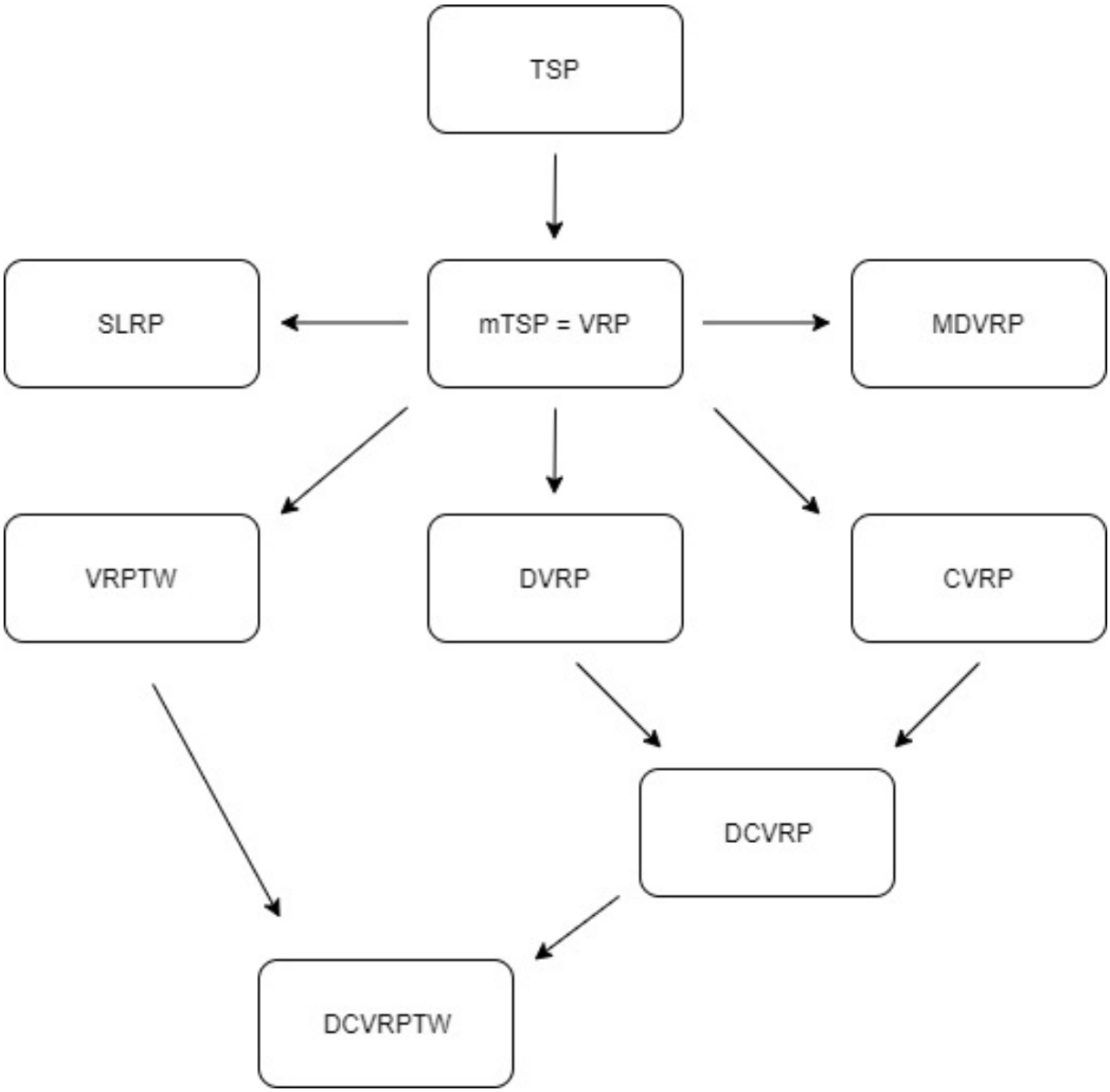
$$\sum_{u=1}^{n} d_u \leq Q \tag{cvrp1}$$

Ahol d_u az u állomás igényét jelöli, míg Q a jármű kapacitását. Ehhez hozzávehetünk két segédfeltételt, amik biztosítják, hogy a járművek elég nagyok az egyes igények (cvrp2), illetve az összigény teljesítéséhez (cvrp3):

$$d_u \leq Q \qquad \forall u \in V \tag{cvrp2}$$

$$\sum_{u=1}^{n} d_u \leq mQ \tag{cvrp3}$$

[3] Takes 14



6.2. ábra. a VRP különböző verziói

Egy másik eset, amikor a járművek úton töltött ideje korlátos (distance constrained VRP - DVRP): elsősorban a járművet vezető személy munkaidejének korlátozottsága miatt. Ekkor a pluszfeltétel:

$$\sum_{u=0}^{n}\sum_{v=0}^{n} t_{uv}x_{uv}^{k} \leq D \tag{dvrp}$$

Gyakran nem csak egy bázisállomás van, hanem több olyan hely is rendelkezésre áll, ahonnan indítani lehet a járműveket (multi depot VRP - MDVRP). Ekkor annyival bonyolódik az eredeti VRP, hogy nem csak az 1-es állomás a kitüntetett, hanem D ilyen van. Ekkor a modell [4] :

[4] Mirabi 1377-1378

$$min \sum_{u,v \in V \cup D, k \in K} t_{uv} x_{uv}^k \tag{mdvrp0}$$

$$x_{uv}^k \in 0,1 \qquad \forall u,v \in V \cup D, k \in K \tag{mdvrp1}$$

$$\sum_{u \in V \cup D} \sum_{k=1}^{m} x_{uv}^k = 1 \qquad \forall v \in V \tag{mdvrp2}$$

$$\sum_{v \in V \cup D} \sum_{k=1}^{m} x_{uv}^k = 1 \qquad \forall u \in V \tag{mdvrp3}$$

$$\sum_{u \in V \cup D} x_{uv}^k - \sum_{w \in V \cup D} x_{vw}^k = 0 \qquad \forall v \in V \cup D \forall k \in K \tag{mdvrp4}$$

Az is jellemző, hogy az állomásoknak van egy nyitvatartásuk, egy olyan idősávjuk, amin belül kell megérkeznie az árut szállító járműnek (VRP with time window - VRPTW). Ekkor a pluszfeltételek [5] :

$$a_0 = w_0 = s_0 \tag{vrptw1}$$

$$\sum_{k=1}^{m} \sum_{u=1}^{n} x_{uv}^k (a_u + w_u + s_u + t_{uv}) \leq a_v \qquad \forall v \in V \tag{vrptw2}$$

$$e_u \leq (a_u + w_u) \leq l_u \qquad \forall u \in V \tag{vrptw3}$$

Itt az e_u jelöli az állomásra érkezés legkorábbi lehetséges időpontját, míg az l_u a legkésőbbit, a_u a jármű tényleges érkezési időpontját, w_u az esetleges várakozási időt ($w_u = max0, e_u - a_u$), míg s_u az állomásról indulás időpontját. Ezekkel a jelölésekkel a (vrptw1) azt fejezi ki, hogy a bázisállomásról feltöltve indul a jármű, a (vrptw2) az érkezési időt számolja ki, a lényeget pedig a (vrptw3) jelenti: a jármű a megadott időintervallumon belül van az állomáson.

Szintén tipikus, ha a járművek által bejárt utak közül a leghosszabbikat akarjuk minimalizálni (shortest longest route problem - SLRP). Ez akkor jó választás, ha az a cél, hogy minél hamarabb kézbesítsünk mindent. Ekkor a célfüggvény [6] :

$$min\{w \in \mathbb{R}\} \tag{slrp0}$$

és egy pluszfeltétel a (vrp1) - (vrp5) mellett:

$$\sum_{u,v \in A} t_{uv} x_{uv}^k \leq w \qquad \forall k \in K \tag{slrp1}$$

Természetesen a való életben ezeknek sokféle bonyolítása is előfordul, hiszen lehet, hogy a járműveink nem egyforma kapacitásúak, bizonyos árukat meghatározott helyen kell felvenni és/vagy leadni, az időablak valamilyen költségért cserébe tágítható vagy esetleg a járműveknek nem kell visszatérniük a bázisra. Persze ezeknek a kombinációi is tágíthatják a lehetőségeket, mert előfordulhat például, hogy a járműveinknek egyszerre van véges kapacitása és korlátozott munkaideje. Mivel mindegyik probléma önmagában

[5] Joubert 17
[6] Valle 135

is meglehetősen bonyolult, így egyáltalán nem magától értetődő, hogy a kombinációikra a modellek kombinálása használható megoldást jelent, ezért jellemzően mindegyikre újabb és újabb modelleket dolgoznak ki.

6.3. Harmadik nekifutás: Kara formulája

A fenti lehetőségeket megismerve kiderült, hogy az OMSZ-os feladatra célszerűbb nem m-TSP-ként tekinteni, hanem érdemesebb DVRP-ként kezelni. Ennek a modellezésével már fentebb foglalkoztam, de az ottani formalizálása n^3-ös változószámot eredményez. Ezt Kara többszöri nekifutásra [7] jócskán egyszerűsíteni tudta, aminek eredményeképpen a változók is n^2-esek. A modell így kicsit bonyolultabbá, nehezebben átláthatóvá vált (a futási idő azonban jelentősen javult) [8] :

$$min \sum_{(u,v)\in A} t_{uv}x_{uv} \tag{K0}$$

$$\sum_{u=1}^{n} x_{uv} = 1 \qquad \forall v \in 2..n \tag{K1}$$

$$\sum_{v=1}^{n} x_{uv} = 1 \qquad \forall u \in 2..n \tag{K2}$$

$$\sum_{u=1}^{n} x_{u1} = m \tag{K3}$$

$$\sum_{v=1}^{n} x_{1v} = m \tag{K4}$$

$$\sum_{(u,v)\in A} z_{uv} - \sum_{(u,v)\in A} z_{vu} - \sum_{v=1}^{n} t_{uv}x_{uv} = 0 \qquad \forall u \in V \tag{K5}$$

$$z_{uv} \leq (T_{max} - t_{v1})x_{uv} \qquad v \neq 1, \forall (u,v) \in A \tag{K6}$$

$$z_{u1} \leq T_{max}x_{u1} \qquad v \in V \tag{K7}$$

$$z_{uv} \geq (t_{uv} + t_{1u})v_{uv} \qquad u \neq 1, \forall (u,v) \in A \tag{K8}$$

$$z_{1v} = t_{1v}x_{1v} \qquad \forall v \in V \tag{K9}$$

$$x_{uv} \in 0, 1 \qquad \forall (u,v) \in A \tag{K10}$$

A (K0) a szokásos célfüggvény, a (K1)-(K4) pedig a szokásos fokszámfeltételek. A (K5) a modell kulcsa: ebben van benne, hogy minden csúcsot egy út során érintünk. A z_{uv} azt fejezi ki, hogy a bázisállomásról a v csúcsig mekkora távolságot tettünk meg, ha közvetlenül előtte u csúcsban voltunk. A (K6)-(K10) feltételek a szokásos távolságkorlátokat érvényesítik.

Ezt a modellt is implementáltam CPLEX-ben, a következőképpen:

```
using CPLEX;
```

[7] Kara 2010, Kara 2011
[8] Kara-Derya 1716

```
/*************************************************************************
* ADATOK
*************************************************************************/
int n = ...; // állomások
range A = 1..n;
int ido[A] = ...;
string nev[A] = ...;

int km[A][A] = ...; // úthálózat
int ora[A][A] = ...;

int s = ...; // szervizutak

float idolimit_float = ...; // időlimit
int idolimit = ftoi(round(idolimit_float * 100));

int ember = ...; // szervizelő emberek száma

int fizetes = ...; // órabér (szuperbruttó)
float oraber = fizetes * (1+0.15+0.1+0.085) * (1+0.17+0.015) / 160;
int oradij = ftoi(round(oraber * ember / 100));

float fogyasztas = ...; // üzemanyagár
int uzemanyagar = ...;
int km_ar = ftoi(round(fogyasztas * uzemanyagar / 100));

int eredeti_km = ...; // eredeti adatok
float eredeti_ora = ...;

/*************************************************************************
* VÁLTOZÓK
*************************************************************************/
dvar int u[A][A]; // u[i][j]=1 jelentése: az i,j állomások közötti
                        közvetlen utat használjuk a k-adik szervizúton
/* Ez tulajdonképpen egy bináris változó, ahogy a későbbi feltételekből
látszani fog. Azért nem valódi bináris változó, hogy a bázisállomás hurokéle
felvehessen bármilyen egész értéket, ami azt jelzi, hogy hány szervizút-
lehetőség marad kihasználatlan. */

dvar int f[A][A]; // gráféleken értelmezett folyamféle függvény
```

```
/*************************************************************************
* CÉLFÜGGVÉNY
*************************************************************************/
minimize sum(i in A, j in A) (km_ar*km[i][j]+oradij*ora[i][j])*u[i][j];

/*************************************************************************
* FELTÉTELEK
*************************************************************************/
subject to {

forall (i in A, j in A) // binárissá alakítjuk az éleket
if (i != 1 || j != 1)
0 <= u[i][j] <= 1;
u[1][1] >= 0; // kivéve a bázisállomás hurokélét, mert az fejezi ki,
                // hogy hány szervizút-lehetőséget nem használunk

forall (i in 2..n) sum(j in A) u[j][i] == 1; // a bázis kivételével
forall (i in 2..n) sum(j in A) u[i][j] == 1; // minden állomásra egyszer
                                              // megyünk be és ki
sum(j in 1..n) u[1][j] == s; // minden szervizút a
sum(i in 1..n) u[i][1] == s; // bázisállomásról indul
                                              // és ott ér véget

forall (i in 2..n) // Kara formula
     sum(j in A: j != i) f[i][j]
   - sum(j in A: j != i) f[j][i]
   - sum(j in A) ((ora[i][j] + ido[i]) * u[i][j])
   == 0;

forall (i in A, j in A) f[i][j] <= (idolimit-(ora[j][1]+ido[j]))*u[i][j];
// a szervizutak az időlimitnél nem tarthatnak tovább

forall (i in 2..n, j in A) f[i][j] >= (ora[1][i]+ora[i][j]+ido[i])*u[i][j];

forall (i in 2..n) f[1][i] == ora[1][i] * u[1][i];

forall (i in 1..n) u[i][i] != 1; // kizárjuk a hurokélek jelentette
                                  // subtourokat
};
```

A programot igyekeztem minél inkább paraméterezhetővé tenni, így ezeket a paramétereket kívülről

olvasom be:

```
n = 120; // állomások száma
s = 28; //szervizutak száma
idolimit_float = 12.5; // időlimit
ember = 2; // szervizelő emberek száma
fizetes = 250000; // nettó fizetés
fogyasztas = 7.6; // https://www.nav.gov.hu/nav/szolgaltatasok/uzemanyag/fogyaszt_normak/gjnorma.html
uzemanyagar = 405; // https://www.nav.gov.hu/nav/szolgaltatasok/uzemanyag/uzemanyagarak
eredeti_km = 12046; // eredeti adatok
eredeti_ora = 153.48;

// beolvasandó adatok
SheetConnection sheet("adatok_csak_egynapos.xlsx");
km from SheetRead(sheet, "km");
ora from SheetRead(sheet, "ora");
ido from SheetRead(sheet, "ido");
nev from SheetRead(sheet, "nev");
```

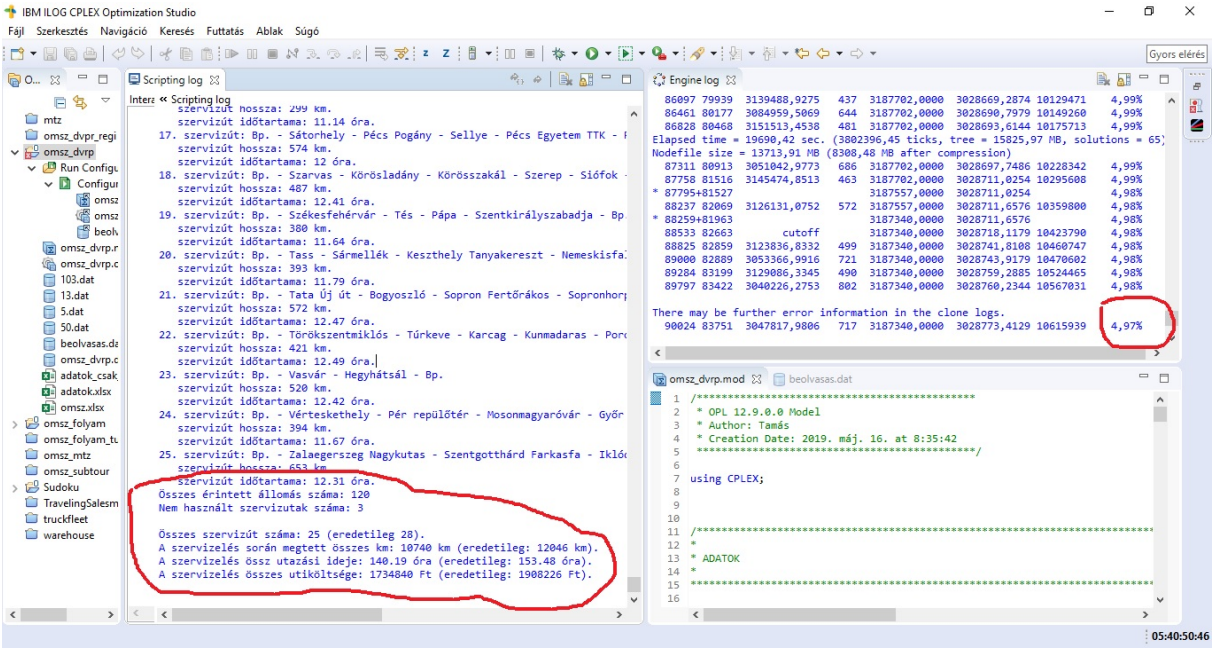
Az eredményt szintén külön íratom ki, a következőképpen:

```
execute {
var k = 1;
var sz = s - u[1][1];
var i = 1;
var j = 1;
var tmp = 1;
var c = 1; //számlálja, hány állomást járunk be
while (k <= sz){
var reszhossz = 0;
var reszido = 0;
write(k , ". szervizút: Bp.");
while (u[i][1] != 1){
while (u[i][j] != 1)
++j;
write(" - " , nev[j]);
if (i == 1)
tmp = j;
reszhossz = reszhossz + km[i][j];
reszido = reszido + ora[i][j] + ido[i];
i = j;
j = 1;
```

```
++c;
}
reszhossz = reszhossz + km[i][1];
reszido = reszido + ora[i][1] + ido[i];
++k;
j = tmp + 1;
i = 1;
writeln(" - Bp.");
writeln("   szervizút hossza: " , reszhossz , " km.");
writeln("   szervizút időtartama: " , reszido / 100 , " óra.");
}

writeln("Összes érintett állomás száma: " , c);
writeln("Nem használt szervizutak száma: " , u[1][1]);

writeln("");
writeln("Összes szervizút száma: " , sz , " (eredetileg " , s , ").")

var osszkm = 0;
for (var i in A) {
for (var j in A) {
osszkm = osszkm + km[i][j] * u[i][j];
}
}
writeln("A szervizelés során megtett összes km: " ,
osszkm , " km (eredetileg: " , eredeti_km , " km).");

var osszora = 0;
for (var i in A) {
for (var j in A) {
osszora = osszora + ora[i][j] * u[i][j];
}
}
writeln("A szervizelés össz utazási ideje: " ,
osszora / 100 , " óra (eredetileg: " , eredeti_ora , " óra).");

var osszkoltseg = 0;
for (var i in A) {
for (var j in A) {
osszkoltseg=osszkoltseg+(km_ar*km[i][j]+oradij*ora[i][j])*u[i][j];
}
```

```
}
var eredeti_koltseg=km_ar*eredeti_km+ember*oraber*eredeti_ora;
writeln("A szervizelés összes utiköltsége: " ,
osszkoltseg , " Ft (eredetileg: " , eredeti_koltseg , " Ft).");
writeln;
writeln;
}
```

Ennek a programnak már használható eredménye lett, ahogyan az 6.3 ábra szerinti képernyőképen látható is ennek egy részlete.



6.3. ábra. Az eredmény

Ezen néhány fontosabb részletet külön be is karikáztam. Egyrészről a kapott eredmény nem optimum, hanem csak közelít ahhoz: 5%-os távolságon belüli. Ez már elég jónak mondható, tekintve, hogy egy átlagos számítógépen (Intel Core i5-ös, 1,7Ghz-es processzor, 4 GB memória) belátható idő alatt, kevesebb, mint 6 óra futásidő után érte ezt el. Azért állt meg itt, mert a gépen elfogyott a memória.

Ráadásul a kapott eredmény is bíztató, hiszen az OMSZ saját gyakorlatához képest több, mint 10%-os javulást sikerült elérni, ami anyagiakban, útiköltségben, munkaidőben számottevőnek mondható.

7. fejezet

Továbbfejlesztési lehetőségek

A kapott eredmény biztató, de továbbfejleszthető. Egyrészről a megoldhatóság érdekében az eredeti feladatot néhány ponton egyszerűsítettem, illetve az eredményről az OMSZ munkatársaival konzultálva kiderült, hogy van néhány olyan szempont, amit korábban nem jeleztek. Másrészről olyan értelemben is továbbfejleszthető az eredmény, hogy hasonló modellezéssel további OMSZ-os feladatokat is érdemes volna kezelni.

7.1. Ami kimaradt

A modellezés folyamán egyszerűbb volt számomra sima DVRP-t megoldani, amikor minden szervizútnak ugyanaz az időlimitje. A valóságban azonban minden évben van olyan út, amit kétnaposan oldanak meg. Ez már az egy fokkal összetettebb DCVRP témaköre. Ráadásul az is érdekes lenne, hogy mekkora nyereséget jelentene, ha az eddigi gyakorlathoz képest több ilyen kétnapos út lenne betervezve.

A valóságban előfordulnak olyan állomásokis, amik "összetartoznak", amiket fixen egy útban kell tervezni.

Emellett klasszikus kapacitásbeli korlátozások is felvetődhetnek, mert az állomásokon beszerelt mérőműszerekből véges mennyiségű tehető az autóba. Ennek elsősorban nem térfogatbeli okai vannak, hanem az áll a hátterében, hogy a használt mérőműszerek meglehetősen drágák, így csak annyi van belőlük pluszban, amennyit épp egy szokásos út során be kell szerelni. A következő út előtt pedig a legutóbb kiszerelteket kalibrálják. Ezért egy-egy út során csak korlátozott mennyiségű állomást lehet meglátogatni.

A karbantartómunkák jellemzően a szabadban történnek, ott kell cserélni a mérőműszereket, ezért a valóságban sokszor nagyon fontos a természetes fény, azaz érdemes lenne a sötétedés előtt befejezni a szereléseket. Ráadásul néhány állomásnál (például repterek esetében) szigorú nyitvatartási időket is figyelembe kell venni. Ezek már DCVRPTW felé mutatnak. Ebbe a témakörbe tartozik, hogy az OMSZ-os gyakorlat és az általam írt program is abból indul ki, hogy a mérőműszereket évente kell újrakalibrálni, azaz minden állomást évente meg kell látogatni. A valóságban azonban 13 hónap a szavatossági idő, így ezzel is lehetne tovább optimalizálni.

További izgalmas, bár kevésbé matematikai fejlesztési lehetőség lenne a kapott eredmények vizualizálása. A felhasználók számára segtíséget jelenthetne, ha a kapott útvonalak egy térképen berajzolva

megjelennének. Emellett a munka szervezését segítené, ha az egyes utak eszközigényét is kiírná a program, értelemszerűen egy előzetes input fájlból lekérdezve a szükséges adatokat.

Mindezeket beépítve a modellbe még inkább indokolttá válik, hogy valamilyen komolyabb számítógépen futtassuk a programot. Egyrészt úgy kevésbé korlátozná a gép memóriája a gap méretét, másrészről a futási idő is kezelhetőbbre rövidülne. Próbálkoztam egy kicsit erősebb géppel, de érdemes lenne valamilyen szuperszámítógépet bérelve futtatni a programot.

7.2. További OMSZ-os feladatok

A fentiek mellet további feladatok is adódnak az OMSZ-nál, amikkel érdemes lenne hasonló módszerekkel foglalkozni.

Az OMSZ-nak van egy vízügyi hálózata is, ahol szintén automata mérések folynak, jelenleg körülbelül 130 állomáson, és szintén évente kell, de csak hőmérő szenzort cserélni. Itt viszont a csapadékmérőt helyben kell kalibrálni, ami időbe kerül. Jelenleg a két automata hálózat karbantartása teljesen független egymástól. Érdemes lenne megvizsgálni, hogy esetleg költséghatékonyabban lehetne-e a kettő hálózatot egyben kezelni, illetve hány emberrel lehetne ezt megoldani.

A hálózat karbantartásának másik oldala a rendszeres felügyelet, ellenőrzés. Ezt vidéki telephelyű hálózati ellenőrök (VTH) végzik. Az ő feladatuk, hogy egy munkautasítás alapján bizonyos időközönként látogassák meg a megadott állomásokat, és végezzenek el bizonyos feladatokat. Itt a megoldandó feladat az lenne, hogy hogyan tudná az OMSZ optimálisan felosztani a VTH-k területeit úgy, hogy nagyjából egyenlően legyenek leterhelve. Ez egyfajta kiterjesztése az első feladatnak, mert itt már több központból kellene a bejárást megszervezni (MDVRP). Tehát nam csak a bejárások optimalizálásáról lenne szó, hanem arról is, hogy hogyan hat a bejárásokra a területi felosztás.

A feladat további kiterjesztése az lehetne, hogy a központból induló karbantartóhoz csatlakozik egy VTH és együtt mennek karbantartani. Így az OMSZ budapesti központja részéről kevesebb emberi erőforrást igényelne a karbantartás, illetve egyszerre többfelé lehetne menni, időben gyorsabban elvégezve a munkát.

Irodalomjegyzék

[1] ALMOUSTAFA, SAMIRA, *Distance-constrained vehicle routing problem: exact and approximate solution (mathematical programming)*, Brunel University, 2013

[2] BORCINOVA, ZUZANA, *Two models of the capacitated vehicle routing problem*, Croatian Operational Research Review, 8(2017), 463-469

[3] DANTZIG, GEORGE BERNARD - RAMSER, JOHN, *The truck dispatching problem*, Management Science, 1959(6) 80-91

[4] FRANK ANDRÁS - KIRÁLY TAMÁS, *Operációkutatás*, Typotex, 2013.

[5] JOUBERT, JOHANNES WILHELM, *An integrated and intelligent metaheuristic for constrained vehicle routing*, University of Pretoria, 2006

[6] KARA, IMDAT, *On the Miller-Tucker-Zemlin Based Formulations for the Distance Constrained Vehicle Routing Problems*, International Conference on Mathematical Science, 2010 (1309) 551-561

[7] KARA, IMDAT - DERYA TUSAN, *Polynomial Size Formulations for the Distance and Capacity Constrained Vehicle Routing Problem*, AIP Conference Proceedings, 2011 (1389) 1713-1718

[8] KARA, IMDAT, *Arc based integer programming formulations for the distance constrained vehicle routing problem*, 3rd IEEE International Symposium on Logistics and Industrial Informatics, 2011, 33-38

[9] KULKARNI, RAJA VITHALRAO - BHAVE, P.R., *Integer Programming Formulations of Vehicle Routing Problems*, European Journal of Operational Research, 1985(20), 58-67

[10] LAPORTE, GILBERT - DESROCHERS, MARTIN - NOBERT, YVES, Two Exact Algorithms for Solving the Distance Constrained Vehicle Routing Problem, Networks, 1984(14), 161-172

[11] LIU, YIHAN, *Optimization of Vehicle Routing Problem for Field Service*, Beihang University, 2017

[12] MILLER, C. E. - TUCKER, E. W. - ZEMLIN, RICHARD ANTHONY, *Integer Programming Formulations and Travelling Salesman Problems*, Journal of the ACM, 1960(7), 326–329

[13] MIRABI, MOHAMMAD - SHOKRIB, NASIBEH - SADEGHIEB, AHMAD, *Modeling and Solving the Multi-depot Vehicle Routing Problem with Time Window by Considering the Flexible End Depot in Each Route*, International Journal of Supply and Operations Management, 2016(3), 1373-1390

[14] SHEN, LING - TAO, FENGMING - WANG, SONGYI, *Multi-Depot Open Vehicle Routing Problem with Time Windows Based on Carbon Trading*, International Journal of Enviromental Research and Public Health, 2018(15), 1-20

[15] TAKES, FRANK, *Applying Monte Carlo Techniques to the Capacitated Vehicle Routing Problem*, Leiden University, 2010

[16] VALLE, CRISTIANO ARBEX - CUNHA, ALEXANDRE SALLES DA - MATEUS, GERALDO ROBSON - MARTINEZ, LEONARDO, *Exact algorithms for a selective Vehicle Routing Problem where the longest route is minimized*, Electronic Notes in Discrete Mathematics, 2009 (35), 133–138

[17] https://cloud.google.com/maps-platform/

[18] https://commons.wikimedia.org/wiki/File:Cutting_plane_algorithm2.png

[19] https://developers.google.com/optimization/routing/vrp

[20] https://en.wikipedia.org/wiki/Vehicle_routing_problem

[21] https://hu.wikipedia.org/wiki/Az_utazó_ügynök_problémája