Multi-robot rendszerek megoldása

Diplomamunka

Írta: Fehér Gábor

Alkalmazott matematikus szak

Témavezető:

Illés Tibor, egyetemi docens
Operációkutatási Tanszék
Eötvös Loránd Tudományegyetem, Természettudományi Kar

Eötvös Loránd Tudományegyetem
Természettudományi Kar
2012

Köszönetnyilvánítás

Szeretnék köszönetet mondani mindazoknak, akik a diplomamunkám elkészítésében segítségemre voltak. Legfőképpen témavezetőmnek, Illés Tibornak, akinek köszönhetően megismertem a matematikának ezt a számomra rendkívül érdekes területét, és a diplomamunka írásának hosszúra nyúlt folyamata során végig segítségemre volt. Nagy segítséget jelentettek azok a barátaim, ismerőseim is, akik akár szakmabeliként, akár kívülállóként a készülő dolgozatot elolvasták, és matematikai, nyelvtani, vagy stilisztikai megjegyzéseikkel segítették a munkám elkészülését.

Tartalomjegyzék

1. Bevezetés **1**

1.1. Történeti áttekintés 2

2. A klasszikus CARP **5**

2.1. Matematikai modell 5

2.2. A CARP megoldása 7

2.2.1. Path Scanning algoritmus 8

2.2.2. Augment-Merge algoritmus 10

2.2.3. Construct and Strike algoritmus 13

2.2.4. Metaheurisztikák 15

2.3. Alsó korlát becslések 16

2.4. A CARP optimális megoldása 18

3. A CARP variánsai **19**

3.1. CARP irányított vagy vegyes gráfokon 19

3.2. CARP különböző célfüggvényekkel 19

3.3. CARP határidőkkel 20

3.4. CARP több depóval 20

3.5. CARP közbeeső állomásokkal 20

3.6. CARP mozgó depókkal 20

3.7. CARP behajtási engedélyekkel 21

3.8. Periodikus CARP 21

3.9. CARP sztochasztikus igényekkel 21

4. A több depós CARP feladat **22**

4.1. Matematikai modell 22

4.2. Megoldási módszerek 23

5. Algoritmus a több depós CARP feladat megoldására **25**

5.1. Az eljárás összefoglalása 25

5.2. A megoldások felépítése 26

5.3. Floyd-Warshall algoritmus 29

5.4. Kezdeti megoldás 30

5.5. Costdifference mátrix létrehozása és frissítése 30

5.6. Törlő algoritmusok 32

5.6.1. Véletlen törlő algoritmus 33

5.6.2. Véletlen útvonalat törlő algoritmus 33

5.6.3. Véletlen csúcshoz tartozó éleket törlő algoritmus 33
5.6.4. Leghosszabb szükségtelen útvonalat törlő algoritmus 33
5.6.5. Nem csatlakozó csúcshoz tartozó éleket törlő algoritmus 34
5.6.6. Hasonló igényeket törlő algoritmus 35
5.6.7. Szabad kapacitás alapján törlő algoritmus 35
5.7. Beillesztő algoritmusok 36
5.7.1. Mohó beillesztő algoritmus 36
5.7.2. Előrelátó beillesztő algoritmusok 37
5.7.3. Legnagyobb igényt beillesztő algoritmus 37
5.8. A kiindulási megoldás kiválasztása 37

6. Számítási eredmények **39**
6.1. Klasszikus CARP benchmark tesztek 39
6.2. Több depós tesztek 44
6.2.1. Módosított benchmark adatok tesztelése 44
6.2.2. Grid gráfokon értelmezett feladatok tesztelése 46

7. Összefoglalás **55**

Irodalomjegyzék **56**

Ábrák jegyzéke

1. A *gdb*19 feladat 5
2. Optimális megoldás a *gdb*19 feladatra 7
3. Path-Scanning 1. lépése a *gdb*19 feladatra 9
4. Path-Scanning 2. lépése a *gdb*19 feladatra 9
5. Path-Scanning 3. lépése a *gdb*19 feladatra 10
6. Path-Scanning 4. lépése a *gdb*19 feladatra 10
7. Augment-Merge 1. lépése a *gdb*19 feladatra 11
8. Augment-Merge eredménye a *gdb*19 feladatra 12
9. Construct and Strike, első útvonal elvétele a *gdb*19 feladat esetén . . . 14
10. Construct and Strike, első Matching feladat megoldása a *gdb*19 feladatra 14
11. Construct and Strike, második útvonal a *gdb*19 feladatra 14
12. Construct and Strike, harmadik útvonal a *gdb*19 feladatra 15
13. A *gdb*19 feladathoz tartozó optimális megoldás felépítése 27
14. Útvonal, amelyben az (i, j) él ugyanabban az irányban két alkalommal szerepel 28
15. Útvonal az 1. lépésben végrehajtott csere után 28
16. Útvonal, amelyben az (i, j) él ugyanabban az irányban két alkalommal szerepel, és ezek közül az egyik alkalommal kiszolgálásra kerül . . 28
17. Útvonal a 2. lépésben végrehajtott csere után 28
18. Augment-Merge eredménye a *gdb*19 feladatra, a törlő algoritmusok bemutatásához 33
19. Véletlen csúcshoz tartozó élek törlésének eredménye a *gdb*19 feladatra 34
20. Leghosszabb szükségtelen útvonal törlésének eredménye a *gdb*19 feladatra 34
21. Nem csatlakozó csúcshoz tartozó élek törlésének eredménye a *gdb*19 feladatra 34
22. Hasonló igények törlésének eredménye a *gdb*19 feladatra 35
23. Szabad kapacitás alapú törlés eredménye a *gdb*19 feladatra 36
24. A *gdb*9 feladat optimális megoldása 42
25. A *gdb*23 feladat optimális megoldása 43
26. 2 depó, 1-1 robot 60 kapacitással 48
27. 2 depó, 2 illetve 1 robot 40 kapacitással 48
28. 2 depó, 2-2 robot 30 kapacitással 49
29. 2 depó, 3 illetve 1 robot 30 kapacitással 49
30. 3 depó, 1-1-1 robot 40 kapacitással 50

31. 3 depó, 2-1-1 robot 30 kapacitással 50
32. 3 depó, 2-1-2 robot 24 kapacitással 51
33. 3 depó, 2-2-2 robot 20 kapacitással 51
34. 4 depó, 1-1-1-1 robot 30 kapacitással 52
35. 4 depó, 2-1-1-1 robot 24 kapacitással 52
36. 4 depó, 2-1-1-2 robot 20 kapacitással 53
37. 4 depó a 13, 18, 43, 48 csúcsokban, 1-1-1-1 robot 30 kapacitással . . . 53
38. 24 x 16 méretű grid megoldása 54

1. Bevezetés

A matematikai szakirodalomban olyan problémákat neveznek multi-robot rendszernek, amelyekben gráfokkal reprezentálható útvonalak hálózatának bejárását kell megoldani. Ezen problémák közül alapvető jelentőséggel bír a súlyozott élbejárási feladat (Capacitated Arc Routing Problem, továbbiakban CARP), melynek lényege, hogy ki kell szolgálni egy utcahálózat utcáit, egy központi depónak nevezett helyről induló, kapacitással korlátozott, egyforma járművekből álló flotta segítségével.

A feladat célja a járművek által bejárt útvonalak költségének minimalizálása. A CARP az ismertebb járműirányítási feladat (Vehicle Routing Problem, VRP) élekre vonatkoztatott párjának tekinthető, és bizonyítottan NP-nehéz. [24]

A gyakorlatban a CARP feladat és változatai számos, olyan magán- és közhasznú társaság munkája során jelentkeznek, ahol bizonyos pontok, helyszínek helyett utcaszakaszok kiszolgálása szükséges. Gyakran előfordul, hogy több különböző feltételt is figyelembe kell venni, ilyenek például heterogén járműflotta, kiszolgálási időkorlátok, tiltott U-fordulók, egyirányú utcák. [4]

A legkorábbi gyakorlati problémák, amelyek felmerültek, az utcaseprés és a villanyóraleolvasás feladata volt. A szemétgyűjtés szervezése szintén modellezhető CARP-pal, ebben az esetben a cél az igények egyenletes elosztása az útvonalak között, de ennek a feladatnak számos különböző, speciális változatát tárgyalták a szakirodalomban, ezek a későbbiekben részletes felsorolásra kerülnek.

Az utcák téli síkosságmentesítésének megszervezése is megfogalmazható CARP feladatként, itt néhány esetben határidők, és egyéb feltételek, például utcaszakaszokra vonatkozó prioritások is szerepet játszanak.

A munkám egyik célja, hogy a gráfok éleinek bejárásával kapcsolatban megfogalmazott problémák áttekintése mellett a CARP-ot részletesen bemutassam. A feladat pontos megfogalmazása mellett bemutatom a hagyományos megoldási módszereket, és összefoglalom a feladattal kapcsolatos friss matematikai eredményeket.

A feladattal kapcsolatban számos olyan gyakorlati problémát megemlítek, amelynek a megoldásához a CARP-ra, vagy valamelyik változatára van szükség. Külön kitérek a CARP feladat több depós változatára, amelyet a klasszikus CARP alapján fogalmazok meg.

A dolgozat másik célja, hogy bemutassak egy olyan eljárást, amellyel a több depós CARP feladat hatékonyan, az egy depós változattal megegyező idő alatt megoldható. Az eljárás részeinek és működésének átfogó bemutatása után ismertetem az algoritmus megvalósításával kapott eredményeket.

1.1. Történeti áttekintés

Az első élbejárási feladatnak a Königsbergi hidak problémája tekinthető, amelyet Euler oldott meg 1735-ben. A feladat, amelyet ma Euler-út problémának hívunk, a következőképpen fogalmazható meg:[41]
Adott egy $G = (V, E)$ összefüggő gráf, találjunk benne egy olyan zárt utat, amely E minden élén pontosan egyszer halad végig, vagy döntsük el hogy ez nem lehetséges. Euler bebizonyította hogy ilyen út akkor és csak akkor létezik, ha G minden csúcsának foka páros, évekkel később Fleury mutatott egy algoritmust, ami megadja az Euler utat.[20]

A következő felmerülő feladat a Kínai postás probléma(Chinese Postman Problem, CPP) volt, amelyet Kwan Mei-Ko kínai matematikus 1962-ben fogalmazott meg:[33]
Adott egy $G = (V, E, C)$ összefüggő gráf, ahol C az élek költségének mátrixa. Találjunk egy olyan utat amely minden élen legalább egyszer végighalad, és teszi ezt a lehető legrövidebb úton. Ha G teljesen irányított, vagy teljesen irányítatlan, a CPP polinomiális időben megoldható [16] [13], de ha G vegyes gráf, akkor a probléma NP-nehézzé válik. [35] A CPP-nek és variánsainak önmagában is jelentős szakirodalma van, amelyről [19] -ben olvasható összefoglaló.

1974-ben Orloff vezette be a Vidéki Postás problémát (Rural Postman Problem, RPP): [34]
Adott egy $G = (V, E, C)$ irányítatlan gráf, ahol C az élek költségének mátrixa. Keressünk egy minimális költségű utat, amely egy adott $R \subseteq E$ részhalmaz minden élén végighalad legalább egyszer. Az RPP NP-nehéz [28], és a nehézsége abból következik, hogy nehéz megállapítani hogy kapcsoljuk össze az R részeit tartalmazó komponenseket az útba. Megmutatható, hogy az RPP és az utazóügynök-feladatok osztályai ekvivalensek. Az RPP-hez létezik 3/2 -approximációs algoritmus [22], és az is igaz, hogy a probléma polinomiális időben megoldható amennyiben az R csúcsai által kifeszített részgráf csak adott fix számú komponensből áll. Ennek a megoldására számos ismert eljárás létezik. [25]

Az alábbi Min-Max k-kínai postás probléma (MM k-CPP) Fredericksontól származik 1978-ból: [23]
Adott egy összefüggő, irányított $G = (V, E, C)$ gráf, ahol C az élek költségének mátrixa, és egy kitüntetett csúcs, amit depónak nevezünk. Találjunk k darab, a depóból induló kört úgy, hogy minden élen legalább egy kör végighalad, és a leghosszabb út hossza minimális. A többi postás feladattal ellentétben, amelyekben az összes megtett út minimalizálása szükséges, ennél a problémánál a feladat a végrehajtási idő minimalizálásának tekinthető. Frederickson az említett cikkében egy $\frac{2-1}{k}$-approximációs

algoritmust is bemutatott a feladathoz. [23]

A súlyozott élbejárási feladatot (Capacitated Arc Routing Problem, CARP) Golden és Wong fogalmazta meg 1981-ben: [24]
Adott egy összefüggő, irányított $G = (V, E, C, Q)$ gráf, ahol C az élek költségének mátrixa és Q az élek igényeinek mátrixa. Továbbá adott egy kitüntetett csúcs, amit depónak nevezünk, valamint W kapacitású egyforma robotokból álló flotta. Találjunk a depóból induló köröket, amelyekre:

1. *Minden pozitív igénnyel rendelkező élt pontosan egy kör szolgál ki.*
2. *Az egy robot által kiszolgált igények összege mindegyik robot esetében kisebb, mint W.*
3. *Az körök összköltsége minimális.*

A súlyozott kínai postás probléma(Capacitated Chinese Postman Problem, CCPP) a CARP olyan változata, amelyben a gráf minden éléhez pozitív igény tartozik, Cristofidestől származik 1973-ból [13]. A CARP és a CCPP feladat is NP-nehéz, és szintén NP-nehéz feladat hozzájuk 3/2-approximációs megoldást találni. Ennek bizonyítása [24]-ban található. Egyetlen ismert eljárás van, amely approximációs megoldást ad a CARP-ra, de csak abban az esetben, ha a G gráf éleinek költségei kielégítik a háromszög-egyenlőtlenséget. Ez egy $7/2 - 3/W$-approximációs algoritmus, az A-ALG, és Wohlk publikálta 2008-ban. [43] Az A-ALG Frederickson RPP-re adott 3/2-approximációs algoritmusát [22] használja, amellyel először konstruál egy nagy kört, amelyet utána optimálisan darabol fel megfelelő számú részre.

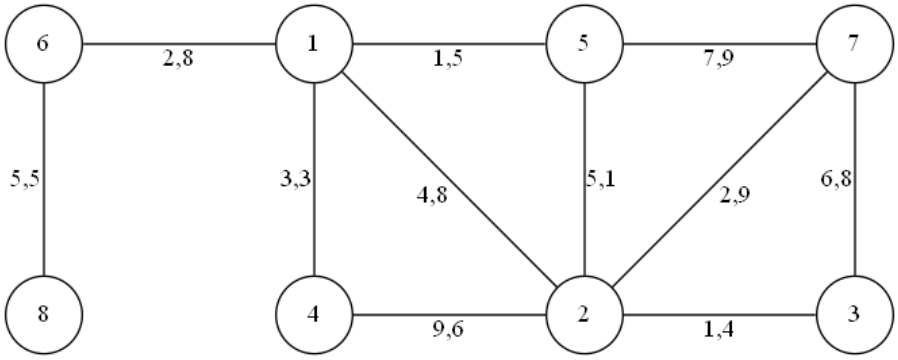
A gráfok bejárási problémáinak létezik egy másik nagy csoportja, ezek a csúcsbejárási feladatok. A fontos különbséget az elnevezésük is megmutatja, utóbbiak esetében a gráf bizonyos csúcsait kell végigjárni az élek helyett. A CARP-pal analóg csúcsbejárási feladat a súlyozott járműirányítási feladat (Capacitated Vehicle Routing Problem, CVRP):
Adott egy összefüggő, irányított $G = (V, E, C, T)$ gráf, ahol C az élek költségének mátrixa, T a csúcsok igényeit tartalmazó vektor, adott egy kitüntetett csúcs, amit depónak nevezünk, továbbá rendelkezésünkre áll egy W kapacitású egyforma robotokból álló flotta. Találjunk olyan köröket, amelyek a depóból indulnak, és:

1. *G minden csúcsát pontosan egy robot szolgálja ki.*
2. *Az egy robot által kiszolgált igények összege mindegyik robot esetében kisebb, mint W.*
3. *Az utak összköltsége minimális.*

Megmutatható, hogy a CVRP CARP feladattá alakítható, és a CARP feladat is felírható CVRP problémaként [24], így a két feladat-osztály ekvivalens. Ezekre az átalakításokra több megoldás is létezik [5] [7], azonban mindegyik esetben bizonyos változók rögzítése, vagy végtelen költségű élek bevezetése szükséges, így a CARP feladatból keletkező CVRP gráf nagyobb és bonyolultabb szerkezetű lesz, mint az eredeti. Léteznek a CARP megoldására olyan eljárások, amelyek a feladatot CVRP-vé alakítják, és a kapott feladat megoldásából nyerik vissza az eredeti CARP megoldását, ezek azonban leginkább a gráf méretének növekedése miatt előnytelenek. Hatékonyabb megoldást szogláltathat a CVRP megoldására használt módszer átalakítása oly módon, hogy a CARP közvetlen megoldására alkalmas legyen.

2. A klasszikus CARP

Ebben a fejezetben az előzőleg megfogalmazott klasszikus CARP feladat tárgyalása következik. A feladat matematikai modelljének pontos bemutatása után megoldások előállítására alkalmas heurisztikus algoritmusokat ismertetek. Ezeknek a közös jellemzője, hogy érvényes megoldást szolgáltatnak a feladatra, azonban nem kapunk arról információt, hogy a kapott megoldások milyen minőségűek. Ebben nyújtanak segítséget a második részben ismertetésre kerülő alsó korlát-becslő algoritmusok. Ezek segítségével bizonyos esetekben még a megoldás optimalitásának ténye is megállapítható. A szakirodalomban négy különböző benchmark-adatbázist használnak a CARP-megoldó algoritmusok hatékonyságának felmérésére, amelyek *gdb* [6], *val* [11], *kshs* [29] és *egl* [18] néven ismertek, és az interneten elérhetőek [8].



1. ábra. A *gdb*19 feladat

Az 1. ábrán az említett adatbázisok egyik feladata látható. A továbbiakban is ezt az egyszerű minta gráfot fogom használni illusztrációként. A depó az 1-es csúcsban található, 3 robotunk áll rendelkezésre 27 kapacitással. Az éleken vesszővel elválasztva a hozzájuk tartozó költség és igény olvasható.

2.1. Matematikai modell

A CARP matematikai modelljétn Golden és Wong [24] cikke, valamint Assad és Golden [4] könyve alapján mutatom be. A CARP egészértékű programozási feladatként definiálható. A használt jelölések a következőek:

- $G = (V, E)$: irányítatlan gráf, amelyben az 1-es csúcs a depó
- c_{ij} : az $(i, j) \in E$ él költsége
- d_{ij} : az $(i, j) \in E$ él igénye
- G csúcsait $1, \ldots, n$-nel jelöljük, és az 1 csúcs lesz a depó

- C : a robotok kapacitása
- Feltesszük, hogy $\forall(i,j) : C > d_{ij}$
- K : a robotok száma
- R : a pozitív igénnyel rendelkező élek halmaza
- $N(i)$: az i csúccsal szomszédos csúcsok halmaza G-ben

Bevezetünk még 4 változó-csoportot:

- x_{ij}^k legyen 1, ha a k robot áthalad $e(i,j)$-n i-ből j felé, különben pedig 0
- l_{ij}^k legyen 1, ha a k robot kiszolgálja $e(i,j)$-t miközben áthalad rajta i-ből j felé, különben 0
- u_S^k legyen 1, ha S-ből nem lép ki olyan él, amin áthalad a k robot, különben 0
- y_S^k legyen 1, ha S-ben van olyan kör, amin a k robot végighalad, különben 0

Ezekkel a jelölésekkel a CARP az alábbi módon írható fel:
(CARP)

$$Min \sum_{k=1}^{K} \sum_{(i,j)} c_{ij} x_{ij}^k$$

Feltételek:

$$\sum_{j \in N(i)} (x_{ji}^k - x_{ij}^k) = 0 \qquad \forall i \in V, k = 1, \ldots, K \tag{1}$$

$$x_{ij}^k \geq l_{ij}^k \qquad \forall (i,j) \in E, k = 1, \ldots, K \tag{2}$$

$$\sum_{k=1}^{K} (l_{ji}^k + l_{ij}^k) = 1 \qquad \forall (i,j) \in E \tag{3}$$

$$\sum_i \sum_j l_{ij}^k d_{ij} \leq C \qquad k = 1, \ldots, K \tag{4}$$

$$l_{ij}^k, x_{ij}^k \in \{0,1\} \qquad \forall (i,j) \in E, k = 1, \ldots, K \tag{5}$$

$$\sum_{(i,j) \in (S,S)} x_{ij}^k - n^2 y_S^k \leq |S| - 1 \qquad \forall S : \emptyset \neq S \subset \{2, \ldots, n\} \tag{6}$$

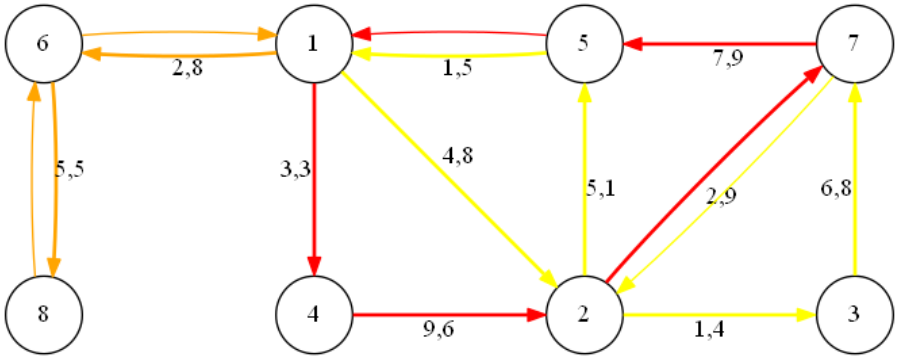
$$\sum_{(i,j) \in (S,\overline{S})} x_{ij}^k + u_S^k \geq 1 \qquad \forall S : \emptyset \neq S \subset \{2, \ldots, n\} \tag{7}$$

$$u_S^k + y_S^k \leq 1 \,,\, u_S^k, y_S^k \in \{0,1\} \qquad \forall S : \emptyset \neq S \subset \{2, \ldots, n\} \tag{8}$$

Az (1) feltétel azt garantálja, hogy az egyes robotok útvonala folyamatos, azaz minden egyes csúcsból pontosan annyi kimenő élen halad át a robot, mint amennyi bejövő élen. A (2) feltétel megszabja hogy egy kiszolgált élen mindenképpen végig is kell a megfelelő irányban haladnia a robotnak, a (3) miatt pedig minden igénnyel rendelkező élt ki kell szolgálni. A (4) minden robot esetén azt jelenti, hogy az általa kiszolgált élek igényének összege nem haladja meg a kapacitást. Az (5) feltétel azt a tényt írja le, hogy minden élen, minden irányban egy robot legfeljebb egy alkalommal haladhat végig. Megfigyelhetjük hogy eddig $\mid V \mid *K + K + \mid E \mid + \mid E \mid *K * 2$ darab feltételt írtunk fel. A modell nehézségét azonban a következő (6)(7)(8) feltételek adják, melyek száma a bennük szereplő lehetséges S halmazok száma miatt exponenciálisan sok. Ezeknek a feltételeknek közösen az a jelentése, hogy egyrészt megtiltják, hogy össze nem kapcsolódó körökből álló megoldást elfogadjunk (önmagában az (1) feltétel ezt megengedi), másrészt viszont lehetővé teszik, hogy zárt köröket tartalmazó útvonalakat elfogadjunk.

2.2. A CARP megoldása

A CARP feladat NP-teljessége miatt nem létezik olyan eljárás, ami várhatóan polinomiálisan sok lépésben megad egy optimális megoldást, de a feladattal foglalkozó kutatók hatékony heurisztikus módszereket dolgoztak ki a CARP megoldására. Ezek, bár az általuk szolgáltatott megoldások optimalitásáról semmi nem állítható, érvényes megoldásokat szolgáltatnak a feladatokra. Az évek során folyamatosan fejlesztették őket, és a felmerülő speciális feladatok megoldására alkalmas változatok láttak napvilágot.



2. ábra. Optimális megoldás a *gdb*19 feladatra

A megoldásra használt klasszikus heurisztikákat mutatom be a következőkben. Ma már léteznek ezeknél hatékonyabb algoritmusok a hagyományos CARP megoldására is, de ezek a bonyolult eljárások is az egyszerű heurisztikákon, ötleteken alapulnak. A következő jelöléseket vezetem be segítségül az algoritmusok leírásához:

- RE : a ki nem szolgált igénnyel rendelkező élek halmaza az eljárások közben
- RD : a ki nem szolgált igények összege az eljárások közben
- $SP(i, j)$: az i-ből j-be vezető legrövidebb út költsége

2.2.1. Path Scanning algoritmus

Algoritmus 1 Path Scanning algoritmus

```
while (RE ≠ ∅) do
    Legyen X egy üres kör
    while (X üres, vagy az előző lépésben tudtunk élt hozzáadni) do
        function ÉLT VÁLASZT(X)
            X-hez egy hozzáadandó élt kiválaszt
        end function
        if (A kiválasztott élt hozzáadva a robot kapacitását túllépjük) then
            Hazatérés a depóba a legrövidebb úton
        else
            A kiválasztott él hozzáadása X-hez
        end if
    end while
end while
```

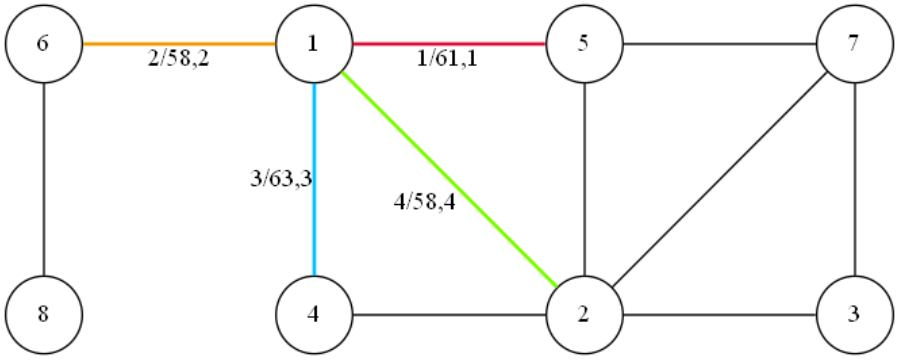
A Path-Scanning algoritmus Golden, DeArmon és Baker 1983-as cikkében jelent meg.[6] Legnagyobb előnye az egyszerű szerkezete és könnyű implementálhatósága. A szerzők öt különböző heurisztikát javasoltak a következőként hozzáadott él kiválasztására.

Ha i az aktuális utolsó csúcs az X útvonalon, vegyük hozzá X-hez az (i, j) élt, ha nem sért kapacitás-feltételt és

1. $c_{i,j}/RD$ minimális, vagy
2. $c_{i,j}/RD$ maximális, vagy
3. $SP(j, 1)$ minimális, vagy
4. $SP(j, 1)$ maximális, vagy
5. $SP(j, 1)$ maximális, ha a robot kapacitásának kevesebb mint a felét használta fel, különben $SP(j, 1)$ minimális.

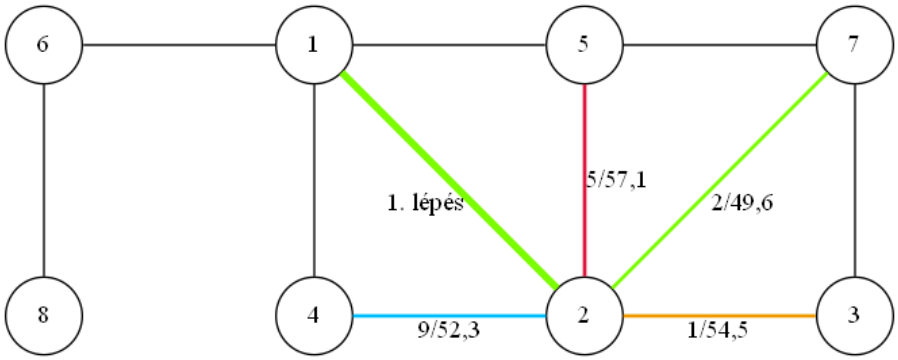
Ezeket a heurisztikákat önállóan használták, és mindegyikre külön Path-Scanning algoritmust lefuttatva kiválasztották a legjobb eredményt. Ennek az eljárásnak a neve Patch-Scanning. Ezt a későbbiekben Pearn fejlesztette tovább úgy, hogy a Path-Scanning algoritmus futás közben minden lépésben véletlenszerűen válasszon ki egyet a heurisztikák közül. [38]

Az öt élválasztó heurisztika működését a *gdb*19 példán mutatom be. Az utolsó heurisztikát fogom használni, de minden lépésben a többi által ajánlott választási lehetőségeket is megadom. Az 3. ábrán az algoritmus első lépése látható, az első robot indul el a depóból. Különböző színekkel jelöltem a választható éleket, rajtuk pedig a hozzájuk tartozó $c_{i,j}/RD$ illetve $SP(j,1)$ értékeket vesszővel elválasztva. Ezek alapján az első és a harmadik heurisztika az $(1,5)$ élt választja, a többi pedig az $(1,2)$ élt.



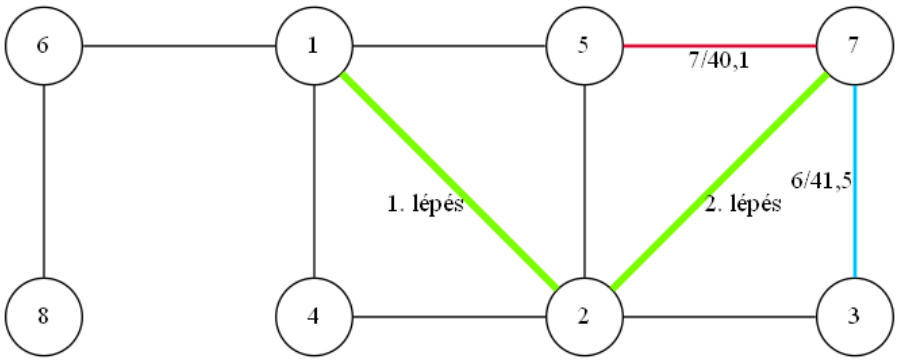
3. ábra. Path-Scanning 1. lépése a *gdb*19 feladatra

Az ötödik heurisztikával az $(1,2)$ él került kiválasztásra. Ez az állapot látható a 4. ábrán. Ebben az esetben a következő lépésben az első heurisztika a $(2,3)$ élt választja, a második a $(2,4)$-et, a harmadik a $(2,5)$-öt, az utolsó kettő pedig a $(2,7)$-et.

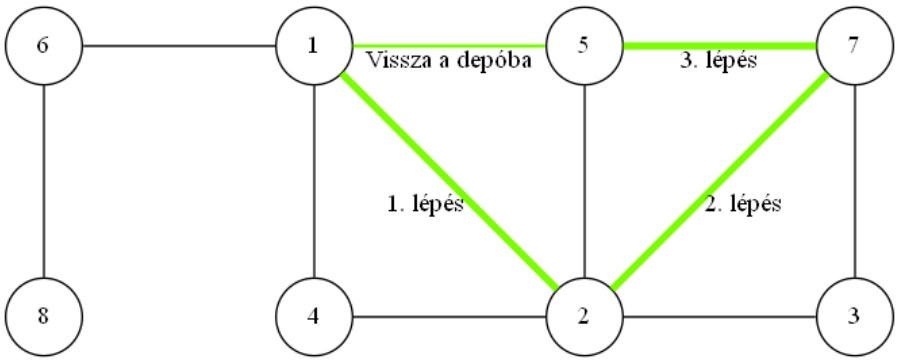


4. ábra. Path-Scanning 2. lépése a *gdb*19 feladatra

A $(2,7)$ élen haladt tovább a robot, ahogy a 5. ábrán látható. Itt csak két él közül választhat. A $(7,3)$ élt az első és a negyedik heurisztika alapján választhatja, a többi szerint a $(7,5)$ élen folytatja az útját. Amikor áthalad a kiválasztott $(7,5)$ élen, nem marad szabad kapacitása, így a legrövidebb úton visszatár a depóba. A robot végleges útvonala a 6. ábrán látható. A vékony zöld vonallal jelölt $(1,5)$ élen a robot a depóba való vissztérésekor a kiszolgálása nélkül halad végig. Az algoritmus ugyanígy folytatódik a következő robot elindulásával, és akkor ér véget, ha minden igény kiszolgálásra került, vagy elfogytak a rendelkezésre álló robotok.



5. ábra. Path-Scanning 3. lépése a *gdb*19 feladatra



6. ábra. Path-Scanning 4. lépése a *gdb*19 feladatra

2.2.2. Augment-Merge algoritmus

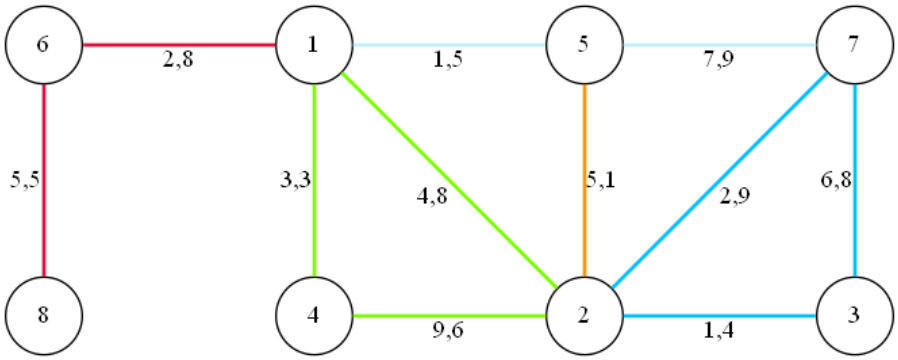
Az Augment-Merge algoritmust Golden és Wong 1981-ben megjelent munkájukban mutatták be. [24] A megtakarítás különböző értelmezésével több variácóban is megvalósítható az algoritmus, és így különböző megoldásokat kaphatunk. Az algoritmust ismét a *gdb*19 feladaton mutatom be. Megtakarításként a megtett útvonalak költségének csökkenését értelmeztem.

Az első lépésben a gráf minden éléhez külön létrehozott körök közül a $(2,3)$ a leghosszabb. Az ezen szereplő élek közül $(7,2)$-t és $(3,7)$-et vehetjük hozzá a ka-

Algoritmus 2 Augment-Merge algoritmus

```
G Minden éléhez egy külön kört hozunk létre, ezek halmaza H
function AUGMENT
    while H ≠ ∅ do
        Legyen a leghosszabb H-beli kör X
        while (X ki nem szolgált élei közül tudunk kiszolgálásra
                választani olyat, ami nem sért kapacitás-feltételt) do
            A kiválasztott él hozzáadása X-hez
            A hozzáadott élhez tartozó kört töröljük H-ból
        end while
    end while
end function
A keletkezett körök halmaza legyen L
function MERGE
    Minden L-beli körpárra számoljuk ki, hogy a két kör uniójának költsége
    mennyi megtakarítással jár a két kör külön vett költségéhez képest
    while (L-ben van két olyan kör, amelyeket összekapcsolva nem sértünk
            kapacitás-feltételt és a megtakarításuk pozitív) do
        Kapcsoljuk össze a legnagyobb megtakarítással egyesíthető párt
        Frissítsük a kör-párok megtakarításait
    end while
end function
```

pacitáskorlát betartása mellett, az ezekhez tartozó külön köröket pedig töröljük. A fennmaradó körök közül a leghosszabb a $(4, 2)$ élhez tartozó, ehhez is hozzá lehet venni éleket. Az így keletkezett körökhöz tartozó élek láthatóak a 7. ábrán.



7. ábra. Augment-Merge 1. lépése a *gdb*19 feladatra

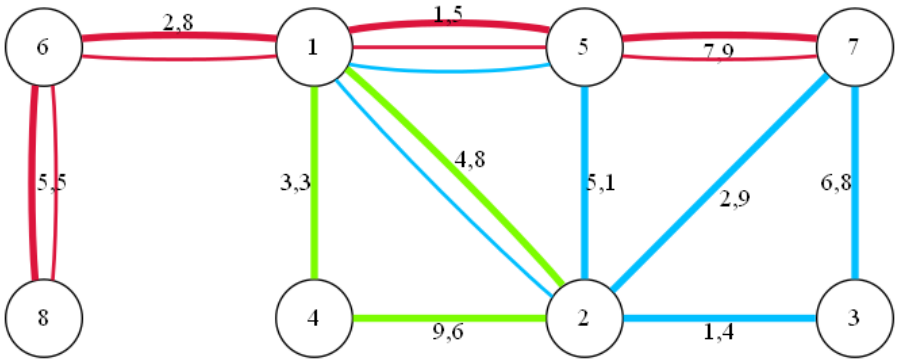
A nagy körök, és a bennük szereplő kiszolgált élek:

1. $\{(2,3)(2,7)(3,7)\}$, költsége 17, használt kapacitása 21
2. $\{(2,4)(1,2)(1,4)\}$, költsége 16, használt kapacitása 17
3. $\{(6,8)(1,6)\}$, költsége 14, használt kapacitása 13
4. $\{(5,7)(1,5)\}$, költsége 14, használt kapacitása 14
5. $\{(2,5)\}$, költsége 10, használt kapacitása 1

A nagy körökből képzett, kapacitást nem sértő párokhoz tartozó megtakarítások:

- $1 \cup 5$: 8
- $2 \cup 5$: 0
- $3 \cup 5$: 0
- $3 \cup 4$: 0
- $4 \cup 5$: 8

Az egyformán 8 megtakarítással járó $1 \cup 5$ és $4 \cup 5$ egyesítések közül a kapacitást jobban kihasználó $1 \cup 5$-re esik a választás. A keletkező kör költsége 19, használt kapacitása 22 lesz. A kapacitás megsértése nélkül a körök közül már csak a 3-as és 4-es egyesíthető 0 megtakarítással. A megoldásként keletkező 3 útvonal különböző színnel látható a 8. ábrán.



8. ábra. Augment-Merge eredménye a *gdb*19 feladatra

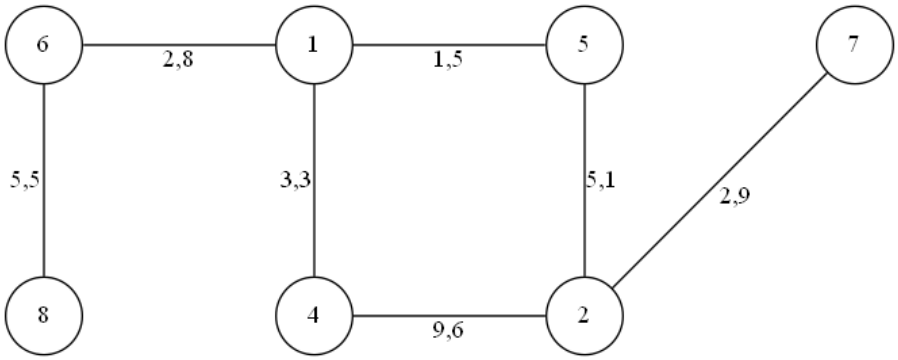
2.2.3. Construct and Strike algoritmus

Algoritmus 3 Contsruct and Strike algoritmus

```
while (RE ≠ ∅) do
    while (RE ≠ ∅ és lehet kört létrehozni) do
        function CONSTRUCT
            Hozzunk létre egy X kört élek hozzáadásával
            úgy, hogy RE \ X összefüggő maradjon
        end function
        function STRIKE
            G-ből vegyük el X-et
            RE = RE \ X
        end function
        A Match által korábban hozzáadott élek eltávolítása G-ből
        function MATCH
            Oldjuk meg a Matching feladatot G-n
            Adjuk hozzá G-hez a szükséges éleket, hogy Euler-gráfot kapjunk
        end function
    end while
end while
```

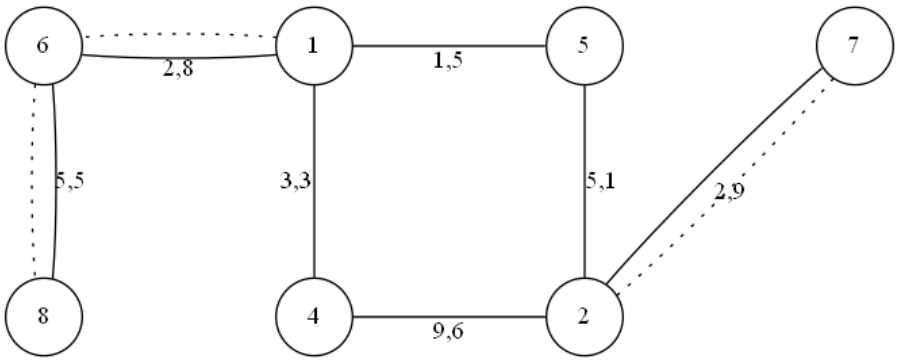
Christofides 1973-as cikkében [13] a súlyozott kínai postás feladat (Capacitated Chinese Postmen Problem, CCPP) bemutatása mellett ismertette a Construct and Strike algoritmust, mint megoldási módszert. A CCPP a CARP azon speciális esete, amikor a gráf minden éléhez pozitív igény tartozik, így ez az algoritmus a CARP megoldására változtatás nélkül alkalmas. A szerző nem adta meg pontosan a Construct lépésben használatos élválasztási kritériumot, erre általában a Path-Scanning algoritmus tárgyalásakor már bemutatott heurisztikák egyikét szokás használni. Az algoritmus működésének bemutatásához, csakúgy mint a Path-Scanning esetén, az 5. heurisztikát fogom használni.

A Construct and Strike első lépéseként az alkalmazott heurisztikával ugyanazt az útvonalat kapjuk mint a Path-Scanning esetében, ez a 6. ábrán már látható volt. A következő lépésben a kiszolgált éleket elvesszük a gráfból, a 0 fokú csúcsokkal együtt. A megmaradó G' gráf látható a 9. ábrán.



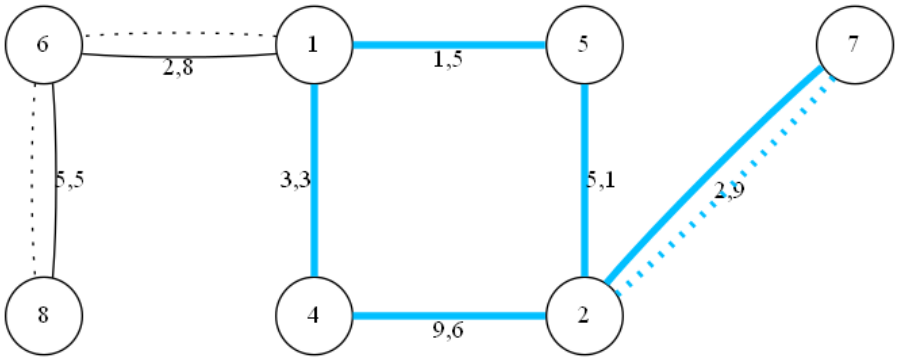
9. ábra. Construct and Strike, első útvonal elvétele a *gdb*19 feladat esetén

Erre megoldva a Matching feladatot, a 10. ábrán látható gráfot kapjuk, szaggatottan jelölve a Matching által hozzávett éleket.

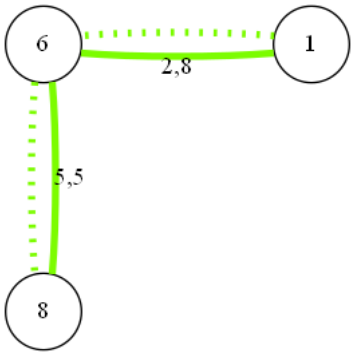


10. ábra. Construct and Strike, első Matching feladat megoldása a *gdb*19 feladatra

Ebben a gráfban a 11. ábrán látható kör kerül kiválasztásra az 5. heurisztika alapján. Ennek a körnek az éleit, és a korábban a Matching által hozzávett éleket a gráfból elvéve, majd a maradék gráfon a Matchinget újra megoldva, és új kört keresve kapjuk a 12. ábrán látható megoldást.



11. ábra. Construct and Strike, második útvonal a *gdb*19 feladatra



12. ábra. Construct and Strike, harmadik útvonal a *gdb*19 feladatra

2.2.4. Metaheurisztikák

Az elmúlt évtizedben a probléma-specifikus eljárások és klasszikus heurisztikák fejlesztésével szemben nagy szerepet kaptak az optimalizálási feladatok megoldásában a metaheurisztikák. A metaheurisztikák olyan eljárások, amelyek az adott feladat egy megoldás-jelöltjét iteratívan próbálják meg tovább javítani. Működésükhöz nem kell sok feltételt szabni a feladattal kapcsolatban, és a megoldások állapotterét nagyon hatékonyan be tudják járni. Hátrányuk, hogy nem adnak információt a megoldás költségéről az optimális megoldáshoz képest. Számos típusuk közül azokat fogom megemlíteni, amelyek segítségével a CARP-ot megoldották.

A Simulated Annealing egy lépésében az addigi megoldás-jelölt segítségével választ egy új megoldást, majd p valószínűséggel elfogadja azt új jelöltnek. A p valószínűséget a két megoldás költségének viszonya határozza meg, ha az új megoldás költsége kisebb, mint az addigi jelölt akkor $p = 1$, és mindenképpen elfogadja az új megoldást új jelöltnek. Az algoritmus előrehaladásával a p értéke egy paraméter által szabályozva folyamatosan csökken. Eglese 1994-ben mutatott be egy ilyen eljárást az utcák síkosságmentesítésének problémájára. [18]

Egy másik metaheurisztika a Tabu keresés, amelyben egy folyamatosan frissített listánk van azokról a megoldásokról, amelyekre nem szeretnénk továbblépni az aktuális megoldás-jelöltről. Ez a lista nem csak a rosszabbnak vélt, hanem azokat a megoldásokat is tartalmazhatja, amelyeket korábban már bejártunk. Minden lépésben keresünk egy olyan új megoldást, amely nem szerepel a listában, és arra továbblépünk. Ezt az eljárást használja a CARPET nevű algoritmus, amelyet Herz és társai publikáltak 2000-ben. [26] Valamint létezik egy teljesen determinisztikus Tabu keresés is a CARP megoldására, amely érvénytelen megoldásokat is megenged, viszont bünteti őket a költségfüggvényben. [12]

A metaheurisztikák újabb csoportját alkotják a genetikus algoritmusok. Ezeknek számos típusa létezik, azonban az alapelv közös bennük: megoldások egy halmazából az elemek egymással való keresztezésével egy új halmaz keletkezik, amelyből a megfelelő elemeket az ún. fitnessük alapján kiválasztjuk és az így kapott halmaz elemeit újra keresztezzük egymással. Lacomme 2004-ben bemutatott genetikus algoritmusa [30] egyike a legjobban teljesítő CARP-megoldó eljárásoknak. Az elemek keresztezése egyetlen nagy körrel történik, és a fitnesst az határozza meg, hogy a kör milyen jól darabolható fel kisebb körökre.

2.3. Alsó korlát becslések

A heurisztikák és metaheurisztikák eredményeinek vizsgálatához szükséges, hogy a kapott megoldások költségét el tudjuk helyezni az optimális megoldás költségéhez viszonyítva. Az optimális megoldást azonban csak a legritkább esetekben ismerjük. Ilyenkor lehet hasznos olyan korlátok ismerete, amelyek alulról becsülik az optimális megoldást, és így segítségükkel felülről becsülhetjük a megoldásaink eltérését az optimálishoz képest.

Ezeknek az alsó korlátoknak az előállítására számos eljárás létezik, amelyeket alapvetően két csoportba sorolhatunk. Az első csoportba tartoznak azok az alsó korlátok, amelyek csak gráfelméleti szempontból közelítik meg a feladatot. A másodikba pedig azok, amelyek lineáris programozási modellt is használnak. Az előbbiek többsége Matching algoritmust használ az alsó korlát számításához.

Fontos megjegyezni, hogy az alsó korlátot számoló algoritmusok általában nem foglalkoznak a pozitív igénnyel rendelkező élek bejárásának költségével, hiszen a feladat megoldása miatt azokat legalább egyszer mindenképpen be kell járni. Azokat az éleket, amelyeket ezeken kívül kell bejárni, vagy amelyeket ezek közül több alkalommal is be kell járni, szükségtelen éleknek nevezzük. Az előbbi megjegyzéssel összevetve látható, hogy egy alsó korlát meghatározása ekvivalens a szükségtelen élek költségének meghatározásával.

Az első algoritmus, amellyel alsó korlátot lehet számítani, a CARP-pal együtt jelent meg Golden és Wong cikkében. [24] Ez az algoritmus, az MLB (Matching Lower Bound) úgy próbálja megbecsülni a szükségtelen élek költségét, hogy az eredeti G gráf alapján bizonyos szabályokkal felépített G' gráfra megkeresi az optimális Matching értékét. Ezt úgy teszi, hogy először a depóból kilépő élek számát hasonlítja össze a minimálisan szükséges robotok számával, tehát az $(1, G \setminus 1)$ vágást vizsgálja. Mivel a depóból minden robotnak ki kell lépnie, és utána vissza is kell érkeznie, így a depóból ki vagy belépő útvonalak számának legalább annyinak kell lennie, mint a szükséges robotok számának kétszerese. Ez alapján építi fel az új G' gráfot,

amelynek egyik tulajdonsága, hogy a depó több példányban található meg benne, valamint bizonyos csúcsok helyett megfelelő tulajdonságú élek szerepelnek.

Egy másik megközelítést alkalmaz az NSLB (Node Scanning Lower Bound), amely az R-beli (azaz pozitív igénnyel rendelkező) élek depótól vett távolsága alapján becsüli meg a hozzájuk való eljutáshoz szükséges utak minimális hosszát. [36] Ez az MLB-nél csak néhány speciális feltétel teljesülése esetén ad jobb korlátot, viszont a kettőt ötvöző MPLB (Matching Path Lower Bound) mindkettőnél jobb eredményt szolgáltat. [37] Ezt úgy teszi, hogy az NSLB-t bizonyos részgráfokra felhasználva épít fel egy G' gráfot, amelyre utána kiszámítja az optimális Matching értékét. Ettől csak a G' gráf felépítésének módjában különbözik az LB1, [11] amely a legjobb eredményt adja azok közül az alsó korlátok közül, amelyek csak a $(1, G \setminus 1)$ vágást használják fel.

A hatékonyabb eljárások más vágásokat is használnak. Legyen U egy depót tartalmazó csúcshalmaz. Az U által generált részgráf, valamint a $V \setminus U$ komplementer csúcshalmaz által generált részgráf költségét LB1 korláttal becsülve, és ezt különböző U halmazokra többször ismételve, a kapott korlátok maximuma lesz az LB2 alsó korlát. Fontos megemlíteni, hogy az egy lépésen belül használt vágások mindig diszjunktak. [11] A legjobb alsó korlát amely Matching-et használ, az MCNDLB (Multiple Cuts Node Duplication Lower Bound), amely az LB2 elvén alapul, azonban minden egyes lépésben egy hatékonyabb algorimust használ a $G \setminus G(U)$ gráf költségének becslésére. [42]

Az alsó korlát becslések második nagy csoportját alkotó eljárások a CARP-nak, vagy valamelyik vele rokon feladatnak a relaxációját oldják meg. Belenguer és Benavent 1998-ban ismertettek egy eljárást, amely vágósík módszerrel oldja meg a CARP relaxációt. [9] Két feltétel-csoportot használnak a megoldásban. A kapacitásfeltételek lényege, hogy minden vágás esetében a depót nem tartalmazó H részgráfra a vágáson keresztül legalább annyi robotnak oda-vissza végig kell haladnia, amennyi a H-beli igények kiszolgálásához legalább szükséges. Páratlan vágásnak nevezzük, ha egy vágás páratlan számú élből áll. Ezek esetében a páratlan vágás feltétel az, hogy a páratlan számú élen oda és vissza is át kell haladni, tehát legalább egy szükségtelen él keletkezik. A feltételek szemléletes jelentése tehát az, hogy egy adott élhalmazhoz legalább hány szükségtelen él tartozik.

A HRLB (Hierarchical Relaxations Lower Bound) 2002-ből az LB2-höz hasonlóan iteratívan választ vágásokat, és általuk érvényes feltételeket, amelyekre megbecsüli a szükségtelen élek költségének minimumát. [3] A használt vágások a korábbi módszerektől eltérően nem diszjunktak, így egyes élek több élhalmazhoz tartozó feltételben is szerepelnek, és a keletkező relaxációs feladat megoldásával pontosabb

becslés kapható az alsó korlátra. Minden egyes lépésben a korábbi vágások által létrejött feltételeket is felhasználja a relaxáció, amelyet minden lépésben újra meg kell oldani.

A HRLB-vel egy időben Belenguer és Benavent bemutatta új vágósíkos algoritmusát. [10] A korábbán általuk, illetve a HRLB-ben is használt kapacitás és páratlan vágás feltételeken túl további feltételeket, és a hozzájuk tartozó azonosítási algoritmusokat ismertettek, amelyek segítségével az összes tesztadaton elérték, vagy megnövelték az addig ismert alsó korlátokat, és ezzel együtt számos ismert megoldásról mutatták meg, hogy optimális.

2.4. A CARP optimális megoldása

Előfordulhat bizonyos esetekben, ha másért nem is, matematikai kíváncsiságból, hogy egy CARP feladatnak meg akarjuk tudni az optimális megoldását, és nem elégszünk meg a heurisztikák eredményeivel. Mivel tudjuk, hogy a probléma NP-nehéz, nem várhatjuk el, hogy minden esetben polinomiális időben megkaphassuk a megoldást, azonban vannak olyan eljárások, amelyek segítségével bizonyos esetekben ennél jóval gyorsabban eljutunk az optimumhoz.

A CARP optimális megoldása Branch and Bound eljáráson alapuló algoritmusokkal lehetséges, melyek alapötlete az, hogy építeni kell egy döntési fát amelyben a megoldások fel vannak sorolva. Alsó korlátok és megfelelő vágások segítségével a fa bizonyos részeit le lehet vágni, ezután el kell dönteni, hogy a fa melyik részére kell további vizsgálat céljából továbblépni. Az első ilyen megoldás a CARP-ra Hirabayashi és szerzőtársai nevéhez fűződik 1992-ből. [27]

A vágósík módszerrel kombinálva kapjuk a Branch and Cut algoritmust, amely a döntési fa minden csúcsában néhány alkalommal megpróbál érvényes egyenlőtlenségeket hozzáadni a feladathoz. Így az alsó korlátok számítása a szomszédos csúcsokra lassabb lesz az új feltételek miatt, a kapott korlátok viszont jobbak lesznek, így összességében kevesebb alkalommal kell az alsó korlátok számítását végrehajtani. [31] További változatai is léteznek a Branch and Bound algoritmusnak, azonban ezeket nem közvetlenül használják a CARP megoldására, hanem a belőle konstruált CVRP feladatot oldják meg. [15]

3. A CARP variánsai

Ebben a fejezetben a klasszikus CARP feladat módosított, illetve általánosított eseteit mutatom be. Mindegyikről elmondható, hogy a való életben felmerülő problémák megoldása során fogalmazódtak meg, mindegyik esethez legalább egy példát is megemlítek. Az egyes esetek megoldási módszerei általában a problémára egyedileg konstruált heurisztikák, ezeknek a bemutatása túlmutat a dolgozat keretein.

3.1. CARP irányított vagy vegyes gráfokon

A klasszikus CARP feladat definíciójában irányítatlan gráf szerepel, azonban számos gyakorlati alkalmazási területen szükség lehet egyirányú utcák, illetve egy utca két oldalának külön, illetve akár párhuzamosan történő kiszolgálása is. Ezekhez a CARP irányított, valamint vegyes gráfon való értelmezése szükséges. Az így kapott feladatot irányított esetben DCARP-nak (Directed Capacitated Arc Routing Problem), vegyes gráf esetében MCARP-nak(Mixed Capacitated Arc Routing Problem) nevezzük. Ezekhez a feladatokhoz is léteznek interneten elérhető benchmark tesztadatok, amelyekkel az eljárások tesztelhetőek.

Az MCARP megoldása speciálisan megváltoztatott Augment-Merge és Path-Scanning algoritmusokkal lehetséges, amelyek figyelembe vesznek egyéb megszorításokat, például tiltott fordulókat is.

3.2. CARP különböző célfüggvényekkel

Minden gráfbejárási feladattal kapcsolatban elmondható, hogy a szokásos célfüggvény általában a teljes megtett út minimalizálására vonatkozik. Ezzel szemben a gyakorlatban előfordul, hogy ettől különböző célok a fontosabbak, például a használt járművek számának minimalizálása, a leghosszabb út hosszának minimalizálása, vagy a terhelések egyenletes elosztása a járművek között.

Egy ilyen probléma, amikor különböző járművekből álló flottánk van, és az egyes járművek használatának van egy fix költsége. A feladat kitűzésétől függően az egyes járműtípusokból végtelen sok, vagy akár megadott számú is rendelkezésünkre állhat. Ilyenkor a célfüggvény a járműhasználatból eredő fix költségek és a megtett útvonal hosszából eredő költség összege, amelyet minimalizálni akarunk.

A korábban említett MM k-CPP értelmezhető egy olyan CARP feladatként, ahol a jármű kapacitása végtelen, és a cél pedig a leghosszabb út minimalizálása.

A DCARP feladatnak létezik olyan verziója, ahol az egyes éleken felmerülő költség attól függ, hogy az élen mikor történik az igény kiszolgálása. A gyakorlatban

felmerülő bizonyos esetekben, például a téli hóeltakarításkor ez a megközelítés sokkal valószerűbb, mint szigorú kiszolgálási határidők megadása.

3.3. CARP határidőkkel

A CARP feladat határidőkkel (CARP with Time Windows, CARPTW) a klasszikus CARP feladatnak felel meg, azzal a hozzáadott megkötéssel, hogy minden él kiszolgálásának egy megadott időintervallumon belül kell megkezdődnie.

A légiirányításban a repülőgép-járatoknak fix indulási ideje van, vagy legfeljebb nagyon kis csúszás engedélyezett a tervezettől, így 0 hosszúságú, vagy nagyon rövid határidőkkel rendelkező CARPTW feladat írható fel rájuk. Az utcaseprés szervezésekor figyelembe kell venni a forgalmas időszakokat a kiemelt útvonalakon. A téli síkosságmentesítésnél pedig bizonyos útszakaszok elsőbbséget élveznek, tehát hamarabb kell őket takarítani, mint másokat. A felsorolt problémák mind megoldhatóak CARPTW feladatként.

3.4. CARP több depóval

A többdepós CARP (Multi Depot CARP, MD-CARP) egy olyan CARP feladat, ahol az egyes járművekhez külön-külön van megadva a csúcs, ahonnan indulniuk, és ahova érkezniük kell. A leggyakoribb esetben az indulási és érkezési depó megegyezik, de előfordul az is, hogy az indulási és érkezési hely különböző. Az MD-CARP feladatokban általában heterogén flotta szerepel, azaz a járművek kapacitásai különbözőek lehetnek.

3.5. CARP közbeeső állomásokkal

A CARP közbeeső állomásokkal(CARP with Intermediate Facilities, CARP-IF) az előző több depós eset egyik általánosítása, amikor az egyes járművek az útvonaluk bejárása közben kapacitásuk újratöltésére szorulnak, amit több különböző helyen egyformán elvégezhetnek. Ez előfordulhat például szemétszállítás esetében, amikor egy városban több szemétlerakó is van, és el kell dönteni, hogy a járművek mikor melyikbe menjenek lerakni az összegyűjtött szemetet.

3.6. CARP mozgó depókkal

A CARP ezen verziójában két különböző típusú jármű van, melyek közül csak az egyik típus jár lerakodni a depóba, a másik típus az előző típusú járműveket látogatja. Ennél a feladatnál az egyes járműtípusok útvonalának meghatározása mellett

el kell dönteni, hogy két jármű mikor és hol találkozzon átrakodás céljából. Ez olyan esetben lehet hasznos, amikor a kiszolgáló jármű, például sószóró autó kisebb kapacitású, és ezért rendszeresen után kell tölteni egy nagyobb kapacitású teherautóból, aminek csak az a dolga, hogy a sószóró autót kiszolgálja.

3.7. CARP behajtási engedélyekkel

Előfordulhat hogy bizonyos megkötések miatt, mint például magassági korlát egy alagútban, nem lehet minden élt minden járművel kiszolgálni. Erre az esetre vonatkozik a CARP behajtási engedélyekkel, amelyben minden egyes élhez meg van adva, hogy a járművek közül melyik szolgálhatja azt ki.

3.8. Periodikus CARP

A periodikus CARP (Periodic CARP, PCARP) egy olyan CARP, ahol egy hosszabb időtartamot veszünk figyelembe, és az egyes éleket több alkalommal is ki kell szolgálni. Ez előfordul például a szemétszállításban, amikor bizonyos háztartásokból hetente 2-3 alkalommal is el kell szállítani a szemetet. A feladat kitűzéséhez általában az is hozzátartozik, hogy meg kell adni, hogy az egyes kiszolgálási időpontok között legalább és legfeljebb mennyi időnek kell eltelnie.

3.9. CARP sztochasztikus igényekkel

A CARP sztochasztikus igényekkel annyiban különbözik a klasszikus feladattól, hogy az egyes élek igényei rögzített értékek helyett valószínűségi változók. Ez felmerülhet a szemétszállítás, a postai kézbesítés, vagy a hóeltakarítás esetén is, amikor a pontos igények nem ismertek.

4. A több depós CARP feladat

4.1. Matematikai modell

A több depós CARP modelljében az eredetihez képest csak néhány változás szerepel. A használt jelölések:

- $G = (V, E)$: irányítatlan gráf
- c_{ij} : az $(i, j) \in E$ él költsége
- d_{ij} : az $(i, j) \in E$ él igénye
- G csúcsait $1, \ldots, n$-nel jelöljük
- D_k : a k robot depója
- C_k : a k robot kapacitása
- Feltesszük hogy $\forall (i, j)\ \forall k : C_k > d_{ij}$
- K : a robotok száma
- R : a pozitív igénnyel rendelkező élek halmaza
- $N(i)$: az i csúccsal szomszédos csúcsok halmaza G-ben

Bevezetjük a klasszikus esetben is használt 4 változó-csoportot:

- $x_{ij}^{k,h}$ legyen 1, ha a k robot melynek a depója h, áthalad $e(i, j)$-n i-ből j felé, különben pedig 0
- $l_{ij}^{k,h}$ legyen 1, ha a k robot melynek a depója h, kiszolgálja $e(i, j)$-t miközben áthalad rajta i-ből j felé
- $u_S^{k,h}$ legyen 1, ha S-ből nem lép ki olyan él, amin áthalad a h depóból induló k robot, különben 0
- $y_S^{k,h}$ legyen 1, ha S-ben van olyan kör, amin a h depóból induló k robot végighalad, különben 0

(MD-CARP)

$$Min \sum_{k=1}^{K} \sum_{(i,j)} c_{ij} x_{ij}^{k,h}$$

Feltételek:

$$\sum_{j \in N(i)} (x_{ji}^{k,h} - x_{ij}^{k,h}) = 0 \qquad \forall i \in V, k = 1, \ldots, K \tag{9}$$

$$x_{ij}^{k,h} \geq l_{ij}^{k,h} \qquad \forall (i,j) \in E, k = 1, \ldots, K \tag{10}$$

$$\sum_{k=1}^{K} (l_{ji}^{k,h} + l_{ij}^{k,h}) = 1 \qquad \forall (i,j) \in E \tag{11}$$

$$\sum_{i} \sum_{j} l_{ij}^{k,h} d_{ij} \leq C_k \qquad \forall k \tag{12}$$

$$l_{ij}^{k,h}, x_{ij}^{k,h} \in \{0,1\} \qquad \forall (i,j), k \tag{13}$$

$$\sum_{(i,j) \in (S,S)} x_{ij}^{k,h} - n^2 y_S^k \leq |S| - 1 \qquad \forall S : \emptyset \neq S \subset \{1, \ldots, n\} \setminus D_k \tag{14}$$

$$\sum_{(i,j) \in (S,\overline{S})} x_{ij}^{k,h} + u_S^k \geq 1 \qquad \forall S : \emptyset \neq S \subset \{1, \ldots, n\} \setminus D_k \tag{15}$$

$$u_S^k + y_S^k \leq 1 \,,\, u_S^k, y_S^k \in \{0,1\} \qquad \forall S : \emptyset \neq S \subset \{1, \ldots, n\} \setminus D_k \tag{16}$$

A (9)-(13) feltételek szinte teljesen megegyeznek az egy depós modellben látott (1)-(5) feltételekkkel, egyedül a robot kapacitását ellenőrző (12)-es feltétel jobb oldalára került az egyedi kapacitás. A (14)-(16) feltételcsoportban jelentősebb változást jelent, hogy a különböző depókhoz tartozó robotokhoz különböző S halmazok tartoznak. Ennek a három feltételnek az értelmezéséhez meg kell figyelnünk, hogy már az egy depós modellben sem volt semmilyen összefüggés az egyes robotok útvonalaira vonatkozó feltételek között, hiszen mindegyik bennük szereplő változó egyedileg az adott robotra vonatkozik. Tehát ha az egy depós modellt úgy tekintjük, hogy először kiválasztunk egy robotot, majd a rá vonatkozó összes S halmazra felírjuk a feltételeket, akkor ezt ugyanúgy megtehetjük itt is, és nem okoz problémát, hogy mások lesznek az S halmazok.

4.2. Megoldási módszerek

A több depós CARP feladatra az egy depóshoz képest sokkal kevesebb specifikus megoldás létezik. A Path-Scanning, Contstruct and Strike, Augment-Merge algoritmusok egyszerű módosításokkal alkalmassá tehetőek a megoldásra. Az egy depós CARP-nál megismert alsó korlátokat meghatározó eljárások több depó esetén nem ismertek, így az algoritmusok hatékonyságáról nem rendelkezünk információval.

Az ismert metaheurisztikák segítségével ebben az esetben is lehet megoldásokat találni. Eglese a téli síkosságmentesítési feladatra adott egy megoldást, amely Simulated Annealing-et használ [17], később ennek egy továbbfejlesztett változatát is

bemutatta. [32] Érdekes megoldási ötleten alapul Amberg és társainak algoritmusa, amely nem közvetlenül a CARP feladatot, hanem a belőle származtatott súlyozott minimális feszítőfa-problémát oldja meg Tabu kereséssel, majd az így nyert megoldásból kapja vissza a CARP megoldását. [2] A több depós CARP általánosításának tekinthető a korábban már említett CARP közbülső állomásokkal, erre ad megoldást szomszédsági kereséssel Polacek és társainak algoritmusa [39], valamint [1].

5. Algoritmus a több depós CARP feladat megoldására

5.1. Az eljárás összefoglalása

A megoldó eljárás ötletéhez a CVRP egyik megoldási módszerét vettem alapul. A korábban említett CVRP a CARP-hoz nagyon hasonló feladat, azonban élek helyett a gráf bizonyos csúcsainak bejárási sorrendjét kell megadni. Ennek a problémának még a CARP-nál is nagyobb ismertsége és szakirodalma van, a gyakorlatban felmerülő szinte összes logisztikai probléma ennek az általánosításának tekithető.

A CVRP egyik ilyen változata a Rich Pickup and Delivery Problem (RPDP), ennek megoldása kapcsán találkoztam az ALNS (Adaptive Large Neighbourhood Search) algoritmussal, amely egy Simulated Annealing-et használó szomszédsági keresés, és részletes leírása [40]-ben olvasható.

Algoritmus 4 ALNS metaheurisztika

```
function ALNS(S_0 ∈ megoldások, q ∈ N)
    S_best = S = S_0
    repeat
        s = S
        q kiszolgált igényt törlünk s-ből
        a ki nem szolgált igényeket beillesztjük s-be
        if  költség(s) < költség(S_best) then
            S_best = s
        end if
        if  elfogad(s, S) then
            S = s
        end if
        törlő és beillesztő heurisztikák súlyozásának frissítése
    until megállási feltétel teljesül
end function
```

Az algoritmus egy S_0 kezdeti megoldásból indul ki, amelyet valamilyen egyszerű heurisztikával lehet előállítani, majd a fő részében minden egyes iterációban három lényeges lépés történik. Először kiválasztunk egy törlő algoritmust, amely bizonyos számú kiszolgált élt eltávolít az aktuális S megoldásból, az így kapott megoldás s. Ezután egy beillesztő algoritmus kiválasztása történik meg, amely az RE-beli éleket egyesével beilleszti s-be. A harmadik lépésben kiszámítjuk, hogy mennyi a keletkezett s megoldás költsége, és ez alapján eldöntjük, hogyan folytatjuk az algoritmust.

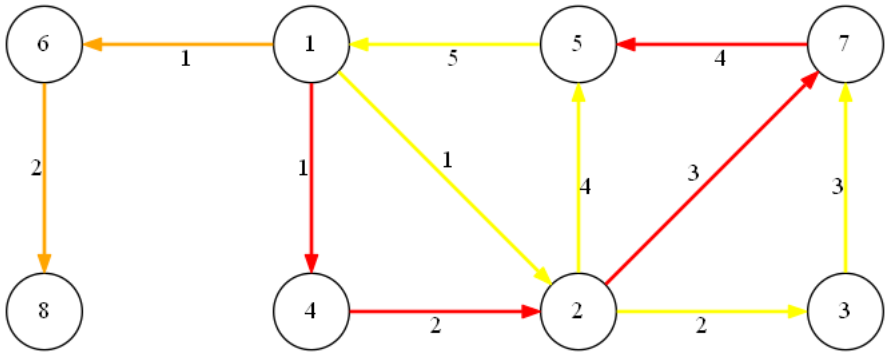
Ha az így kapott s megoldás költsége az eddigi legolcsóbb, akkor elmentjük új legolcsóbb megoldásként S_{best}-nek, és beállítjuk, hogy a következő iterációs lépésben már ebből a megoldásból induljunk ki, vagyis ez legyen az S. Ha nem olcsóbb, akkor az alapján, hogy mennyivel drágább, egy külön eljárással döntjük el, hogy a következő iterációtól kezdve ez legyen-e S. Az ALNS egy tanuló algoritmus, erre utal a nevében található "Adaptive" jelző is. Az egyes törlő és beillesztő heurisztikák teljesítményét értékelve folyamatosan változtatja azok kiválasztásának valószínűségét. Az egyszerűbb megvalósíthatóság és a paraméterek pontosabb megválasztásának lehetősége miatt ezt a lépést kihagyom az algoritmusomból, a heurisztikákhoz rögzített súlyozás fog tartozni. Az egyes lépéseket, a törlő és beillesztő algoritmusok típusait részletesen is bemutatom.

5.2. A megoldások felépítése

A CARP egy megoldásának azt tekintjük, ha a rendelkezésre álló robotok mindegyikéről meg tudjuk mondani, hogy milyen sorrendben járja be G éleit, és ezek közül melyiket szolgálja ki, valamint az összes R-beli élt kiszolgálja valamelyik robot. Megfigyelhető, hogy a ki nem szolgált élek két szempontból fontosak a megoldásokban: az egyik, hogy az útvonalak folytonosságát biztosítják, a második pedig az, hogy rajtuk való áthaladás költséggel jár. Bármely (a, b) és (c, d) kiszolgált élek között ki nem szolgált élek sorozata található (ami esetleg 0 hosszúságú). Ha ezt az élsorozatot a b és c közötti legrövidebb úthoz tartozó élek sorozatával helyettesítjük, akkor szintén megoldást kapunk, amelynek költsége legfeljebb annyi, mint az eredeti megoldás költsége. Ha tehát minden robothoz megadjuk az általa kiszolgált élek sorozatát úgy, hogy minden kiszolgált élhez megadjuk azt is, hogy milyen irányban halad át rajta a robot a kiszolgálásakor, akkor egyértelműen definiált bejárást kapunk. A 13. ábrán a $gdb19$ feladat optimális megoldása látható ilyen formában, az egyes éleken olvasható szám azt jelöli, hogy a robot által kiszolgált igények sorában hányadik helyen szerepel az adott él.

Gondot jelenthet viszont az, hogy a CARP matematikai modellje nem engedi meg, hogy egy élen egy irányban több alkalommal is áthaladjon egy robot, és nem tudhatjuk előre, hogy a legrövidebb utak használnak-e azonos éleket több alkalommal. Erre a problémára jelent megoldást az alábbi állítás:

5.1. Állítás. *A CARP bármely, kiszolgált élek sorrendje és kiszolgálásuk iránya által definiált megoldásából előállítható olyan, legfeljebb ugyanakkora költségű megoldás, amelyben minden él minden robot útvonalában egy irányban legfeljebb egyszer szerepel.*



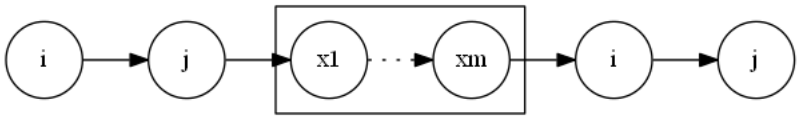
13. ábra. A *gdb*19 feladathoz tartozó optimális megoldás felépítése

Bizonyítás: Megadunk egy eljárást, amely minden útvonalat külön-külön átalakít úgy, hogy megfeleljen a szabályoknak.

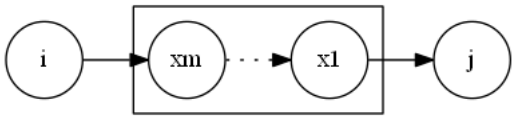
- 0. lépés: Többször előforduló élek megkeresése. Ha egy él többször fordul elő, akkor a kiszolgálása "áthelyezhető" az élek között, vagyis mindegy, hogy a kiszolgálása melyik áthaladáskor történik meg.

- 1. lépés: Válasszuk ki azt az élt, amelyik a legtöbbször fordul elő. Ha ennek az előfordulása kettőnél kevesebb, vagy kettő és az él R-beli, vagy kettő, és ellentétes irányúak az áthaladások, lépjünk tovább a következő lépésre. Ha kettőnél többször fordul elő, akkor lennie kell két olyan előfordulásának, amikor ugyanolyan irányban halad rajta a robot. Keressük meg ezt a két előfordulását, és ha ezek közül valamelyiken történik meg a kiszolgálása, akkor azt helyezzük át egy másik előfordulásra. A másik lehetséges eset az, hogy kettőször fordul elő, azonos irányban és nem R-beli az él. Mindkét esetben van egy olyan útvonalunk, amely D_k-ban kezdődik és végződik, és van rajta egy (i, j) él amelyen ebben a sorrendben két alkalommal is végighalad a robot, de egyik esetben sem szolgálja ki az élt. Legyenek a két végighaladás közötti csúcsok: $i, j, x_1, x_2, \ldots, x_{m-1}, x_m, i, j$. Ez az útvonalrészlet látható a 14 ábrán. Helyettesítsük ezt a csúcs-sorozatot $i, x_m, x_{m-1}, \ldots, x_2, x_1, j$-vel, ami a 15 ábrán látható. Ekkor a robot által kiszolgált élek halmaza nem változik, tehát megoldást kaptunk, viszont a keletkezett útvonalból kikerült az (i, j) él mindkét áthaladása, tehát a kapott megoldás költsége $2 * c_{i,j}$-vel kevesebb mint előtte. Ezt a lépést egészen addig ismételjük, amíg találunk olyan élt, ami megfelel a feltételeknek, és mivel közben az útvonal hossza csökken, így az eljárás előbb-utóbb véget ér.

- 2. lépés: Válasszuk ki az első olyan élt, amelyen kétszer haladunk át azonos irányban, ezek közül az egyik alkalommal ki is szolgáljuk. Ha ilyen nincs, ak-

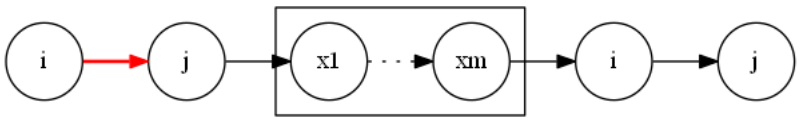


14. ábra. Útvonal, amelyben az (i, j) él ugyanabban az irányban két alkalommal szerepel

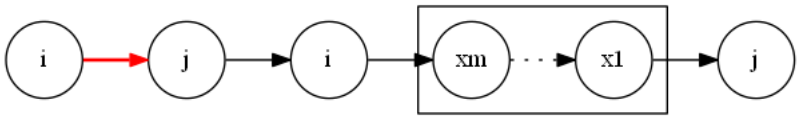


15. ábra. Útvonal az 1. lépésben végrehajtott csere után

kor készen vagyunk. Tegyük fel hogy találtunk egy ilyen (i, j) élt. Ekkor a két áthaladás közötti útvonal $i, j, x_1, x_2, \ldots, x_{m-1}, x_m, i, j$ alakú. Ez látható a 16 ábrán, ahol az (i, j) él piros színe jelöli az él kiszolgálását. Helyettesítsük ezt $i, j, i, x_m, x_{m-1}, \ldots, x_2, x_1, j$ útvonallal. Ekkor egy olyan útvonalat kapunk, amelynek hossza megegyezik az eredetivel, viszont benne az (i, j) élt már a 17 ábrán látható módon, szabályosan járjuk be, mindkét irányban egy alkalommal. Ezután lépjünk vissza az előző lépésre. Mivel az (i, j) él két ellentétes irányú bejárása egymás mellé került, így bármely későbbi lépésben történő bejárási sorrend csere után szintén ellentétes irányú marad a bejárásuk, tehát ezzel az élen többet nem kell javítanunk,vagyis a javításra szoruló élek száma csökkent, így nem kerülhetünk végtelen ciklusba. □



16. ábra. Útvonal, amelyben az (i, j) él ugyanabban az irányban két alkalommal szerepel, és ezek közül az egyik alkalommal kiszolgálásra kerül



17. ábra. Útvonal a 2. lépésben végrehajtott csere után

Megjegyzendő, hogy mivel az élek többszörös bejárása többletköltséget okoz, ezért a kapott megoldás elegendő iterációs lépés után nem fog többszörös élbejárásokat tartalmazni, tehát ez a javító algoritmus csak elméleti szempontból fontos. Az állítás alapján megtehetjük, hogy a megoldásokat olyan formában keressük, hogy minden egyes robothoz megadjuk az élek kiszolgálásának sorrendjét és irányát.

Egy kapott S megoldás $C(S)$-sel jelölt költségét a alábbi képlet határozza meg:

$$C(S) = \sum_{(i,j)} c_{ij} x_{ij}^{k,h} + RE_S * C_{RE}$$

A képletben szereplő RE_S az S megoldásban ki nem szolgált igények számát jelöli, a C_{RE} paraméter pedig a ki nem szolgált igényekhez tartozó büntetést.

5.3. Floyd-Warshall algoritmus

A CARP-ot megoldó eljárás első lépése, hogy minden u, v csúcspárhoz meghatározzuk a köztük levő legrövidebb út költségét. Erre a célra Floyd-Warshall algoritmust használok, amely $O(n^3)$ lépésben a gráfban levő összes pontpárhoz meghatározza a legrövidebb utak költségét.

Algoritmus 5 Floyd-Warshall algoritmus

1: $path[i][j] = c_{i,j}$, ha $(i, j) \in E$; 0 ha $i = j$; különben ∞
2: **for** $i = 1 \ldots n$ **do**
3: **for** $j = 1 \ldots n$ **do**
4: **for** $k = 1 \ldots n$ **do**
5: $path[i][j] = \min(path[i][j], path[i][k] + path[k][j])$
6: **end for**
7: **end for**
8: **end for**

Az eljárást Robert W. Floyd 1962-ben publikálta [21], tőle függetlenül alkalmazta Stephen Warshall gráfok tranzitív lezártjának meghatározására. Eredményeként csak az egyes pontpárokhoz tartozó legrövidebb út költségét kapjuk meg, magát az útvonalat nem. Amennyiben ez szükséges, az útvonalak megkapására az algoritmus egy módosított változatát lehet használni. A bemutatásra kerülő módszer során ezt nem használjuk, hiszen az algoritmus futása közben keletkező útvonalaknak csak a költségét használjuk fel, maga a konkrét bejárási útvonal egy köztes lépésben nem hordoz információt. Az eredményül kapott megoldásban szereplő útvonalak például Dijkstra-algoritmussal megkaphatóak, így csak a feltétlenül szükséges utak kerülnek kiszámításra.

5.4. Kezdeti megoldás

A szomszédsági keresés kezdeti megoldásának előállítására Path-Scanning algoritmust alkalmazok. Az élválasztási kritériumokat az alábbiak szerint határoztam meg, mindegyik esetben feltételezve azt, hogy a jármű az adott lépésben az i csúcsban van:

1. Válasszuk ki a legkisebb ki nem szolgált igénnyel rendelkező (i, j) élt, ha ilyen nincs, akkor válasszuk ki azt a ki nem szolgált élt, amelyik i-ből a legrövidebb úton elérhető.

2. Válasszuk ki a legnagyobb ki nem szolgált igénnyel rendelkező (i, j) élt, ha ilyen nincs, akkor válasszuk ki azt a ki nem szolgált élt, amelyik i-ből a legrövidebb úton elérhető.

A kapott S_0 megoldás K_{S0} költsége, és egy p paraméter alapján számítjuk ki a Simulated Annealing használatához szükséges kezdeti hőmérsékletet:

$$T_{start} = \frac{-p * C_{S0}}{\ln \frac{1}{2}}$$

Ezt a T értéket minden lépésben egy T_c paraméterrel szorozva csökkentjük. A hőmérséklet arra szolgál, hogy segítségével meghatározzuk egy kapott s megoldás elfogadásának valószínűségét. Ennek pontos leírása a vonatkozó 5.8 szakaszban olvasható.

5.5. Costdifference mátrix létrehozása és frissítése

A kezdeti megoldás felépítése után létrehozunk, és a későbbiekben folyamatosan frissítünk egy tömböt, amelyben minden k robothoz és d ki nem szolgált igényhez elmentjük, hogy a robot útvonalába melyik helyen, milyen áthaladási sorrendben és milyen költséggel lehet az igény kiszolgálását beilleszteni. A tömb mérete $|R| * K * 3$, azonban minden iterációs lépésben ebből csak az RE-beli igényekhez tartozó értékeket használjuk. A tömb egy $[d, k]$ eleme, amely a d igényhez és k robothoz tartozik, az alábbi információt tartalmazza:

- $CD[d, k]$: a d igény k robot útvonalába való legolcsóbb beillesztésének költsége.

- $IND[d, k]$: a legolcsóbb lehetséges beillesztés sorszáma a robot útvonalában, azaz a k robot által hányadik kiszolgált igény lenne a d.

- $DIR[d, k]$: a legolcsóbb lehetséges beillesztés esetén a robot milyen irányban haladna át a d igényhez tartozó (i, j) élen, ahol i és j úgy van meghatározva, hogy $i < j$.

- ha a robot $i \to j$ irányban halad végig az élen, miközben a d igényt kiszolgálja, akkor $DIR[d,k] = 1$
- ha a robot $j \to i$ irányban halad végig az élen, miközben a d igényt kiszolgálja, akkor $DIR[d,k] = 0$

Egy $[d,k]$ elem értékeinek kiszámítása az alábbi módon történik, erre csak olyan lépésben kerül sor, amikor $d \in RE$. Tegyük fel, hogy a k robot az aktuális s megoldásban z darab igényt szolgál ki, útvonalának költsége $C(s) = c$, fennmaradó kapacitása pedig w.

Algoritmus 6 CostDifference frissítése

```
if d_{i,j} > w then
    CD[d,k] = ∞
else
    CD[d,k] = ∞
    for ind = 1...z+1 do
        for dir = 0...1 do
            Illesszük be a d igényt a k robot útvonalába úgy,
            hogy az ind-edik kiszolgált igény legyen,
            és a robot a dir iránynak megfelelően haladjon át rajta.
            Legyen a keletkezett útvonal költsége u
            if u − c < CD[d,k] then
                CD[d,k] = u − c
                IND[d,k] = ind
                DIR[d,k] = dir
            end if
            Töröljük ki a d igényt a k robot útvonalából
        end for
    end for
end if
```

Az algoritmussal kapott értékek a legkisebb költségű beillesztési lehetőséghez tartoznak. Megjegyzendő, hogy az $IND[d,k]$ és $DIR[d,k]$ értékeket csak abban az esetben mentjük el, ha $CD[d,k]$-ra véges értéket kapunk. Ez azért lehetséges, mert a későbbiekben csak azok közül a CD-beli értékek közül fogunk válogatni, amelyek végesek, a végtelen értékeket nem vesszük figyelembe. A frissítésre szoruló értékek mindig azok, amelyeknek a kiszolgálása megszűnt, vagy amelyeknél az értékhez tartozó k robot útvonala megváltozott. A szükséges frissítések számontartására két halmazt használunk:

- UR: elemei igények, $d \in UR$ azt jelenti, hogy a d igényhez tartozó összes értéket frissíteni kell.
- UK: elemei robotok, $k \in UK$ azt jelenti, hogy a k robothoz tartozó összes értéket frissíteni kell.

A halmazok különválasztására azért van szükség, mert bizonyos lépésekben nem szükséges mindkét halmaz elemeinek frissítése. Ahol frissítés történik, ott mindig meg fogom jegyezni, hogy az melyik halmazra vonatkozik.

5.6. Törlő algoritmusok

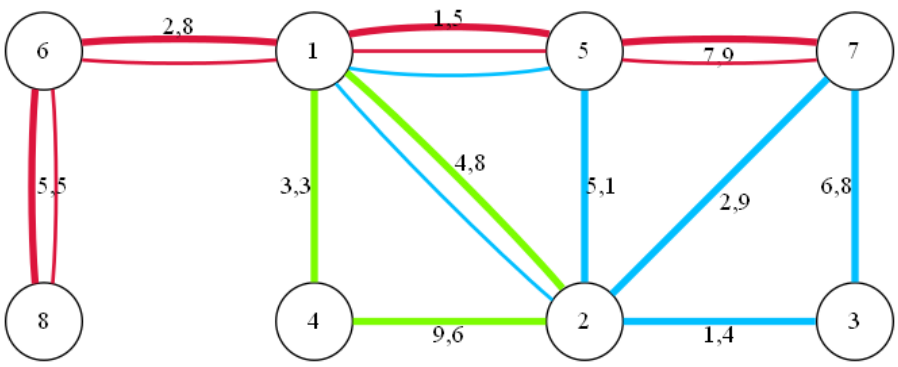
A törlő algoritmusok az aktuális S megoldásból egyedi stratégiájuk szerint választanak kiszolgált igényeket, amelyek kiszolgálását törlik a megoldásból, és az éleket áthelyezik RE-be, a kiszolgálatlan élek halmazába. Egy törlés lépései:

1. a törlendő igények kiválasztása
2. a kiválasztott igények kikerülnek a kiválasztott robot útvonalából
3. az igények bekerülnek RE-be
4. a robot útvonalának költsége újraszámolódik
5. a robot bekerül UK-ba
6. a törölt igények bekerülnek UR-be

Az egyes törlő algoritmusok csak az 1. lépésben különböznek egymástól, a többi mindegyik esetében ugyanúgy történik. Az 1. lépésben többes szám használatára azért van szükség, mert vannak olyan törlő algoritmusok, amelyek egy lépésben több élt, vagy akár egy egész robot útvonalát is kitörlik. Az egy iterációs lépésben törlésre kerülő igények TI száma minden egyes lépésben a megadott korlátok közül véletlenszerűen kerül kiválasztásra. A használt alsó és felső korlátokat a TI_q paraméter segítségével kapjuk meg: $TI \in \{\min(\max(4, \frac{TI_q * R}{2}), 25) \ldots \min(\max(6, TI_v * R), 50)\}$ A törlési eljárás addig ismétlődik, amíg a törölt igények száma el nem éri TI-t.

Mindegyik törlő algoritmushoz tartozik egy paraméterként megadott súly, amelyek alapján minden iterációs lépésben véletlenszerűen választjuk ki az aktuális lépésben alkalmazásra kerülő algoritmust.

A törlő algoritmusokat az Augment-Merge algoritmus $gdb19$ feladatra adott megoldásán mutatom be, amely a 18. ábrán látható. Az egyszerűség kedvéért a minimális $TI = 4$ esetet mutatom be, azaz a törlések addig folytatódnak, amíg 4 igény törlésre nem kerül. Az első kettő véletlen algoritmushoz nem mellékelek példákat.



18. ábra. Augment-Merge eredménye a *gdb*19 feladatra, a törlő algoritmusok bemutatásához

5.6.1. Véletlen törlő algoritmus

Ez a legegyszerűbb törlő algoritmus, véletlenszerűen kiválaszt szükséges számú kiszolgált igényt, és törli őket az őket kiszolgáló robotok útvonalaiból.

5.6.2. Véletlen útvonalat törlő algoritmus

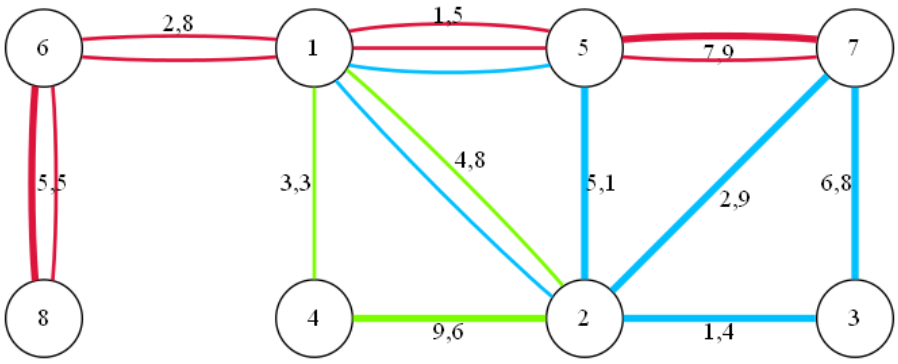
Kiválaszt egy robotot véletlenszerűen, és törli az összes általa kiszolgált igény kiszolgálását.

5.6.3. Véletlen csúcshoz tartozó éleket törlő algoritmus

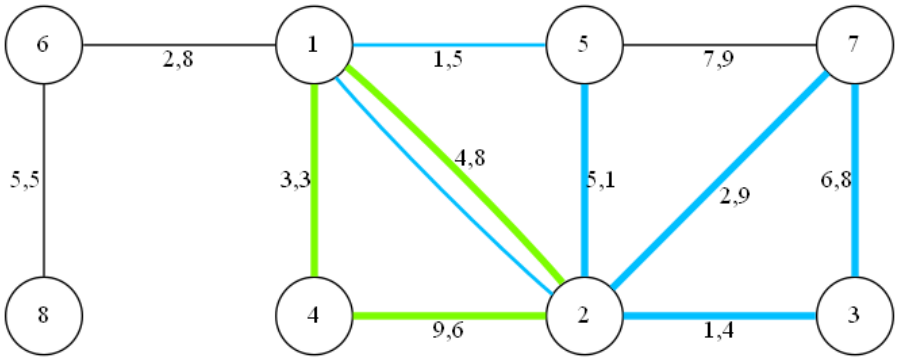
Véletlenszerűen kiválaszt egy csúcsot, és az összes rá illeszkedő élhez tartozó kiszolgált igényt törli. Például az 1-es csúcsot választva az $(1,2),(1,4),(1,5),(1,6)$ élek kiszolgálása kerül törlésre, amint az látható a 19. ábrán. Az, hogy egy igény kiszolgálása törlésre kerül, nem feltélenül jelenti azt, hogy utána a robot nem halad át az élen. Előfordulhat, hogy két másik, általa kiszolgált él között a törölt igényhez tartozó él továbbra is szerepel a legrövidebb útvonalban. Ebben a konkrét példában, mivel a depóra illeszkedő élek igényeinek kiszolgálásai törlődtek, ez mindegyik törölt élre igaz.

5.6.4. Leghosszabb szükségtelen útvonalat törlő algoritmus

Megkeresi a leghosszabb olyan utat, amit egy robot két kiszolgált él között tesz meg, és az összes kiszolgált igényt törli az adott útvonalból. Ebben az esetben ez az út a piros színel jelölt robothoz tartozik, az $(1,5)$ és $(5,7)$ élek kiszolgálása után 8 költséggel tér vissza az 1-es csúcsig, hogy az $(1,6)$ élt kiszolgálja. A robot útvonalának törlése utáni állapot látható a 20. ábrán.



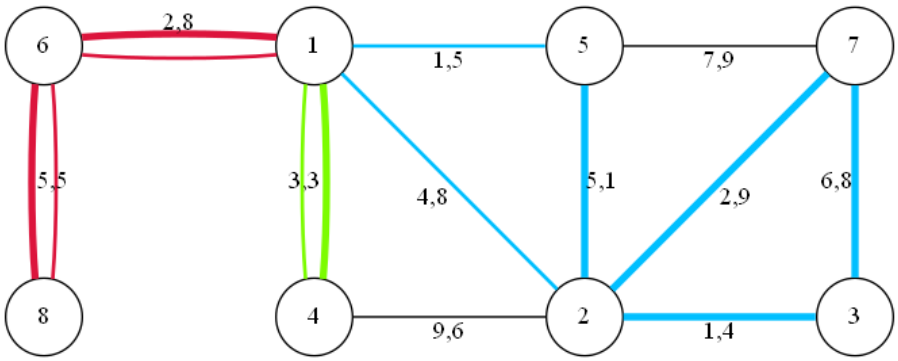
19. ábra. Véletlen csúcshoz tartozó élek törlésének eredménye a *gdb*19 feladatra



20. ábra. Leghosszabb szükségtelen útvonal törlésének eredménye a *gdb*19 feladatra

5.6.5. Nem csatlakozó csúcshoz tartozó éleket törlő algoritmus

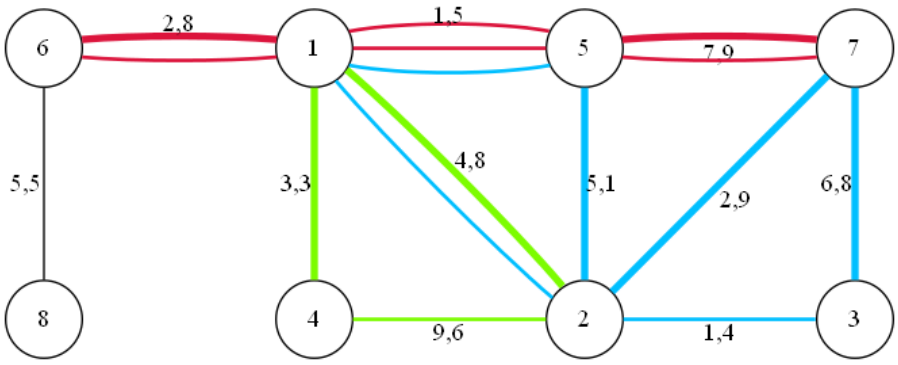
Az algoritmus először véletlenszerűen kiválaszt egy robotot, majd keres az útvonalában olyan csúcsokat, amelyek között szükségtelen élek vannak. A második lépésben a többi robot útvonalából törli az összes olyan kiszolgált igényt, amelyek a korábban kiválasztott csúcsok valamelyikére illeszkednek. A példában a kék színű robotot kiválasztva ezek a csúcsok a 2 és az 5, a hozzájuk tartozó $(1,2)(1,5)(2,4)(5,7)$ élek kerülnek törlésre. Ez látható a 21. ábrán.



21. ábra. Nem csatlakozó csúcshoz tartozó élek törlésének eredménye a *gdb*19 feladatra

5.6.6. Hasonló igényeket törlő algoritmus

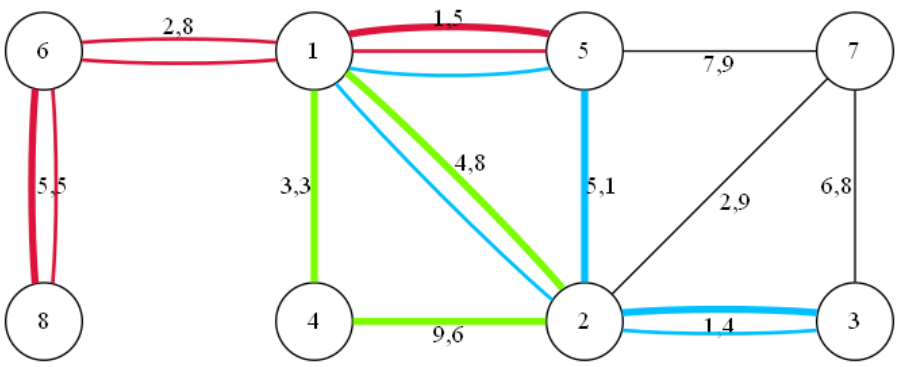
Az algoritmus véletlenszerűen kiválaszt egy igényt, amelynek veszi az m mennyiségét. Először törli az összes olyan igényt, amelynek a mennyisége m, majd törli az összes olyan d igényt amelynek az m_d mennyiségére $m - i <= m_d <= m + i$. Ez az eljárás egészen addig folytatódik amíg minden lépésben 1-el növelve az i értékét a törölt igények száma eléri TI-t. Az példában az $(1, 5)$ élt kiválasztva a hozzá tartozó igény 5, így először az 5 igénnyel rendelkező élek kiszolgálása kerül törlésre, ezek az $(1, 5)$ és a $(6, 8)$ élek. Mivel az így törölt igények száma csak 2, az $[5 - 1, 5 + 1]$ intervallumba eső igények is törlődnek. Ezek a $(2, 3)$ és a $(2, 4)$ élekhez tartoznak, velük együtt már elegendő számú igény törlődött. A törlés eredménye látható a 22. ábrán.



22. ábra. Hasonló igények törlésének eredménye a $gdb19$ feladatra

5.6.7. Szabad kapacitás alapján törlő algoritmus

Működése annyiban tér el az előző algoritmustól, hogy egy igény helyett egy robotot választ véletlenszerűen, és a robot szabad kapacitásával megyegyező, valamint annál kisebb mennyiségű igényeket törli. A minta megoldásban a zöld robotot kiválasztva a szabad kapacitása 10, ilyen igény nincs, ezért először törli a 9 igénnyel rendelkező élek kiszolgálását. Ezek a $(2, 7)$ és az $(5, 7)$ élek. A törölt igények száma még nem elegendő, így a 8 igénnyel rendelkező éleket is törli, vagyis az $(1, 6)$-ot és a $(3, 7)$-et. Az eredmény a 23. ábrán látható.



23. ábra. Szabad kapacitás alapú törlés eredménye a *gdb*19 feladatra

5.7. Beillesztő algoritmusok

A beillesztő algoritmusok szerepe az, hogy az aktuálisan RE-beli igényeket egyesével beillesztik az aktuális S megoldásba. Minden iterációs lépésben egészen addig illesztenek be igényeket, amíg $RE \neq \emptyset$, vagy az RE-beli igényeket már nem lehet szabályosan beilleszteni. Egy igény beillesztésének lépései:

1. a beillesztendő igény kiválasztása, legyen ez d
2. a d igény bekerül a kiválasztott k robot útvonalában az $IND[d, k]$-adik helyre
3. az útvonalban elmentődik hozzá az élen való áthaladás $DIR[d, k]$ iránya
4. a robot útvonalának költsége újraszámolódik
5. a robot bekerül UK-ba
6. az igény kitörlődik RE-ből
7. UK elemeinek frissítése

A beillesztéseket végző algoritmusoknak két csoportja van, az egyik csoportba sorolandó a mohó algoritmus, a másikba pedig az előrelátó algoritmusok. A törlő algoritmusokkal megegyezően, mindegyik beillesztő algoritmushoz tartozik egy paraméterként megadott súly, amelyek alapján minden iterációs lépésben véletlenszerűen választjuk ki a alkalmazandó beillesztő algoritmust.

5.7.1. Mohó beillesztő algoritmus

A mohó algoritmus mindig a lokálisan legjobb megoldást választja, ebben az esetben ez azt jelenti, hogy megkeresi azt a d igényt, amelyet a legolcsóbban tud beilleszteni az aktuális s megoldásba. Ez pontosan az az igény, amely a CD mátrix RE-beli igényekhez tartozó oszlopaiban található legkisebb értékhez tartozik.

5.7.2. Előrelátó beillesztő algoritmusok

Az előrelátó algoritmusok azonos elv alapján működnek, csak egy t paraméter megválasztásában különböznek egymástól. Az algoritmusok minden d igényhez számítanak egy $RCD(t, d)$ értéket, amely azt próbálja számszerűsíteni, hogy mennyivel járhatunk rosszabbul t lépéssel később, ha az aktuális lépésben nem a d igényt illesztjük be a számára lehetséges legolcsóbb módon. Az algoritmusok minden lépésben azt a d igényt illesztik be, amelyre a kapott $RCD(t, d)$ érték maximális. A beillesztés a lehetséges legolcsóbb módon történik, tehát annak a k robotnak az útvonalába, amelyre $CD[d, k]$ minimális.

Algoritmus 7 RCD(t,d) kiszámítása

```
function RCD(t ∈ {2...5}, d ∈ RE)
    for k = 1...K do
        min[k] = CD[d, k]
    end for
    min értékek közül a legkisebb t darab rendezése
    RCD = −min[0]
    for i = 2...t do
        RCD = RCD + min[i]
    end for
end function
```

5.7.3. Legnagyobb igényt beillesztő algoritmus

A legnagyobb igényt beillesztő algoritmust a tesztelés első tapasztalatai alapján vezettem be. Azt vettem észre, hogy azokban a feladatokban, ahol a robotok összes kapacitása nagyon közel van a kiszolgálandó igények összes mennyiségéhez, gyakran előfordult hogy az igények kiszolgálását nem sikerült megfelelően szétosztani, és így kiszolgálatlan igény maradt a megoldásban. Ennek az elkerülésére fogalmaztam meg ezt az algoritmust. Mindig megkeresi a legnagyobb $d \in RE$ igényt, és a hozzá tartozó minimális $CD[d, k]$ értékkel megfelelően beilleszti a k robot útvonalába.

5.8. A kiindulási megoldás kiválasztása

Minden iterációs lépés végén el kell döntenünk, hogy a következő lépésben melyik megoldásból induljunk ki. Két lehetőségünk van; az első hogy nem változtatunk, és ugyanabból az S-ből indulunk, mint amelyikből az aktuális lépésben. A második pedig, hogy az aktuális lépésben keletkezett s megoldást elfogadjuk új kiindulási

megoldásnak. Ha s az addigi legjobb, akkor mindenképpen elfogadjuk új S-nek, ha nem, akkor az alábbi valószínűséggel fogadjuk el:

$$P_{elfogad} = e^{\frac{(C(S)-C(s))}{T}}$$

A kapott valószínűség szemléletes jelentése, hogy a kezdeti T_{start}-tal számolva egy S-nél p százalékkal rosszabb s megoldást $\frac{1}{2}$ valószínűséggel fogadunk el. Miközben T értékét folyamatosan csökkentjük, ez a valószínűség nő, így biztosítva azt, hogy az algoritmus a futása során ne ragadjon lokális minimumban.

6. Számítási eredmények

A programot C# nyelven készítettem Microsoft Visual Studio 2010 segítségével, a gráfok megjelenítéséhez Graphviz programot használtam. A számítási teszteket egy 2.4Ghz-es processzormagon végeztem. A mesterkéltnek tűnő megfogalmazás oka, hogy a processzor ugyan két magos, azonban az egy szálon futó kód csak az egyik magot veszi igénybe. Az elvégzett teszteket két csoportba soroltam. Az első csoportba tartoznak a klasszikus CARP feladathoz létező benchmark adatok, ezeket az algoritmus eredményességének felmérése miatt oldottam meg, hiszen megoldásaikról és a hozzájuk tartozó alsó korlátokról bőséges dokumentáció található. A második csoportban az egy depós benchmark adatok több depósra módosított változatai, illetve egyszerű szerkezetű négyzethálós ún. grid gráfok tartoznak.

6.1. Klasszikus CARP benchmark tesztek

A benchmark adatbázisokat J.M. Belenguer honlapjáról töltöttem le. [8] A *kshs* tesztadatokat Kiuchi és társai használták egy párhuzamos Branch & Bound eljárás tesztelésére [29], a *gdb* adatok Golden-től származnak [6]. A *val* tesztek Benavent és társai cikkében jelentek meg [11], 10 különböző gráfhoz generáltak a járművek kapacitásának megváltoztatásával különböző eseteket. Ezeknek az adatoknak közös jellemzői hogy kézzel, véletlenszerűen hozták létre őket, és a gráfokban az összes élhez pozitív igények tartoznak.

A negyedik csoportban találhatóak az *egl* feladatok, amelyek Eglese-től származnak, és valós adatok felhasználásával készültek. [18] A tesztadatok az Egyesült Királyságban található Lancashire megye keleti és déli területének úthálózatán alapulnak, az ezeket reprezentáló gráfokhoz hozzárendelt különböző kapacitású járművek összesen 24 *egl* feladatot határoznak meg.

Az algoritmust mindegyik esetben 10 alkalommal futtattam, és ezek közül a legjobb eredményt vettem figyelembe. A használt paraméterek az alábbiak voltak (az ettől eltérő eseteket jelezni fogom):

- Iterációszám: 1000
- $p = 0.01$ Azaz a kezdeti hőmérséklettel számolva egy annál 1%-kal rosszabb megoldás elfogadásának valószínűsége $\frac{1}{2}$.
- $TI_q = 0.3$ Azaz egy lépésben az összes igény legfeljebb 30%-a kerül törlésre.
- Az összes törlő és beillesztő heurisztika súlyozása egyenlő, tehát mindegyiket ugyanakkora valószínűséggel választja az algoritmus.

A táblázatok oszlopaiban látható értékek:

- Név: a feladat neve
- V: a gráf csúcsainak száma
- R: a pozitív igénnyel rendelkező élek száma
- D: az összes igény mennyisége
- K: a rendelkezésre álló robotok száma
- C: a robotok kapacitása
- LB: a legjobb ismert alsó korlát (Lower Bound) [15]
- UB: a legjobb ismert megoldás (Upper Bound) [15]
- S_{best}: Az algoritmussal kapott legjobb eredmény
- %: S_{best} eltérése a legjobb ismert megoldástól
- I: S_{best} megtalálásához szükséges iterációk száma
- T: az algoritmus futási ideje ezredmásodpercben

Az első táblázatban láthatóak a *kshs* adatokhoz kapott megoldások. A feladatok kis mérete miatt mindegyik esetben nagyon gyorsan megkaptam az ismert optimális megoldásokat. Mindegyik esetben elegendő lett volna 1000 helyett 100 iteráció futtatása is, így azonban a futási idők jobban összehasonlíthatóak a többi eredménnyel.

1. táblázat. *kshs* adatok tesztje

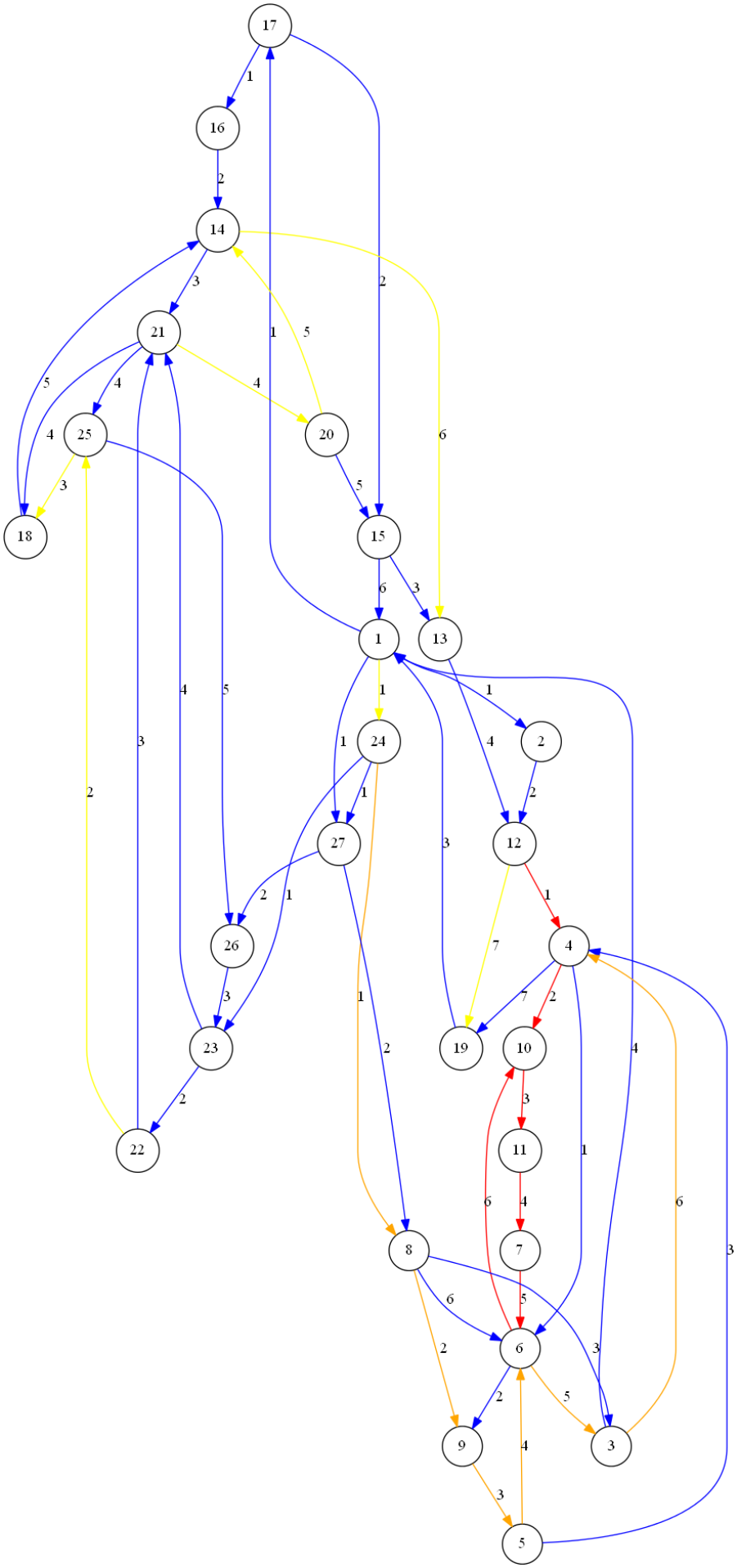
Név	V	R	D	K	C	LB	UB	S_{best}	%	I	T
kshs1	8	15	535	4	150	14661	14661	14661	0	16	47
kshs2	10	15	497	4	150	9863	9863	9863	0	3	48
kshs3	6	15	565	4	150	9320	9320	9320	0	3	46
kshs4	8	15	594	4	150	11498	11498	11498	0	31	44
kshs5	8	15	443	3	150	10957	10957	10957	0	43	47
kshs6	9	15	389	3	150	10197	10197	10197	0	3	46

Az *gdb* adatok tesztelését az előzőekkel megegyező paraméterekkel kezdtem, azonban néhány esetben nem kaptam meg a várt eredményt, ezeket *-gal jelöltem, majd az iterációszámot 10000-re növelve újra futtattam a programot. A változtatás

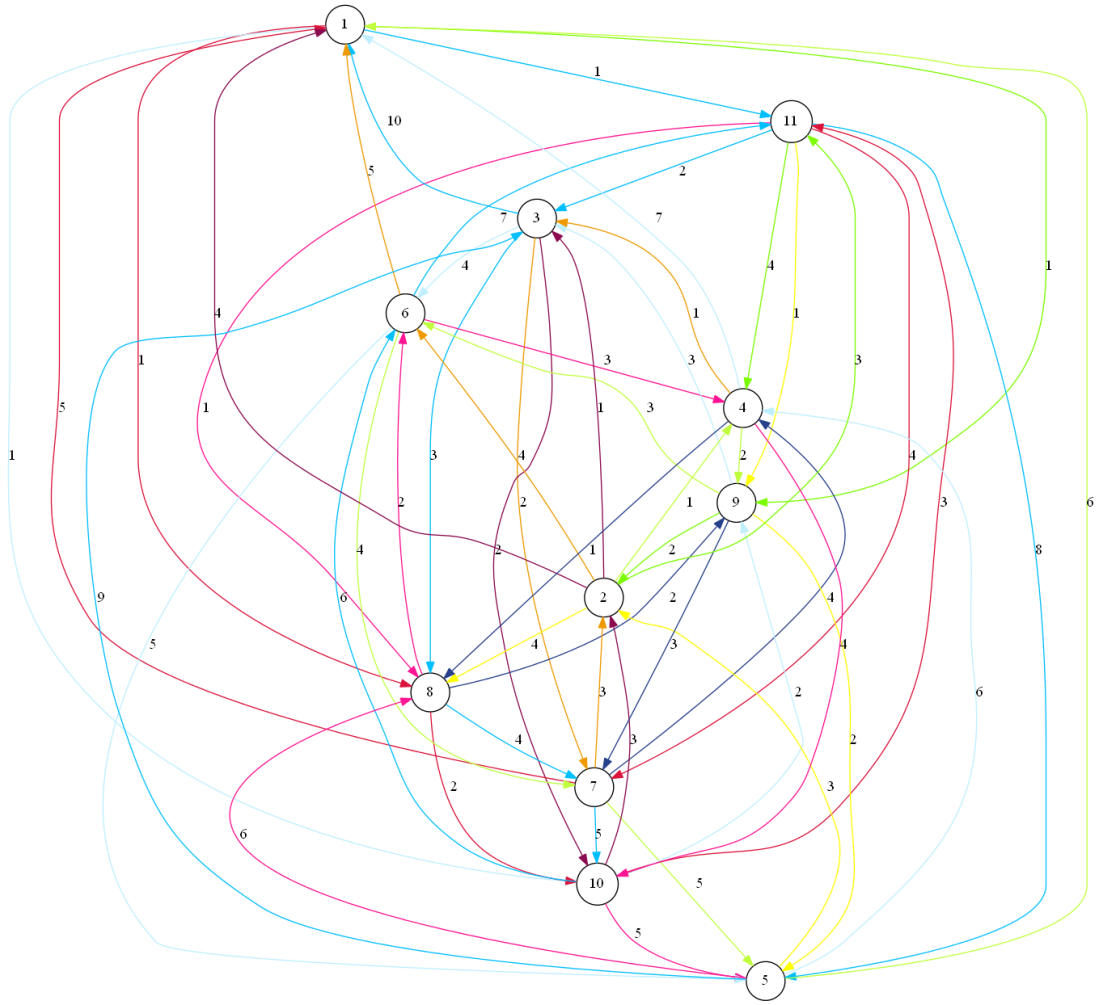
a szükséges futási időt hozzávetőlegesen 10-szeresésére növelte ezekben az esetekben. Így az összes feladatra megkaptam az ismert legjobb megoldást. Közülük a legnagyobbaknak, a *gdb*9-nek és a *gdb*23-nak az optimális megoldásai a 24. illetve a 25. ábrán láthatóak.

2. táblázat. *gdb* adatok tesztje

Név	V	R	D	K	C	LB	UB	S_{best}	%	I	T
gdb1	12	22	22	5	5	316	316	316	0	17	121
gdb2	12	26	26	6	5	339	339	339	0	120	168
gdb3	12	22	22	5	5	275	275	275	0	75	157
gdb4	11	19	19	4	5	287	287	287	0	7	109
gdb5	13	26	26	6	5	377	377	377	0	68	155
gdb6	12	22	22	5	5	298	298	298	0	72	135
gdb7	12	22	22	5	5	325	325	325	0	9	135
gdb8	27	46	249	10	27	348	348	348	0	796	288
gdb9	27	51	258	10	27	303	303	303	0	*7857	4004
gdb10	12	25	37	4	10	275	275	275	0	78	150
gdb11	22	45	224	5	50	395	395	395	0	95	388
gdb12	13	23	212	7	35	457	458	458	0	29	71
gdb13	10	28	245	6	41	534	536	536	0	807	94
gdb14	7	21	89	5	21	100	100	100	0	13	75
gdb15	7	21	112	4	37	58	58	58	0	3	73
gdb16	8	28	116	5	24	127	127	127	0	18	115
gdb17	8	28	168	5	41	91	91	91	0	2	118
gdb18	9	36	153	5	37	164	164	164	0	35	257
gdb19	8	11	66	3	27	55	55	55	0	2	37
gdb20	11	22	107	4	27	121	121	121	0	371	64
gdb21	11	33	154	6	27	156	156	156	0	63	174
gdb22	11	44	205	8	27	200	200	200	0	314	332
gdb23	11	55	266	10	27	233	233	233	0	*2919	4798



24. ábra. A *gdb*9 feladat optimális megoldása



25. ábra. A *gdb*23 feladat optimális megoldása

Az *egl* tesztadatok az előzőeknél jelentősen nagyobb méretűek, ezeknek a tesztelését 100000 iterációval végeztem, és az s jelű adatok esetében a TI_q paramétert 0.2-re csökkentettem. Különösen a nagyobb feladatokban, ahol szűkebb a robotok rendelkezésre álló kapacitása, az algoritmus nem teljesített olyan jól, mint a korábbi példákban. Ezzel együtt is, az ismert legjobb megoldásokat legalább 2-3%-ra sikerült megközelítenie. Az $egl - s4 - C$ adat esetében a legjobb eredményt 250000 iteráció futtatásával kaptam.

A val [11] tesztadatok vizsgálatának megkezdésekor sajnos észre kellett vennem, hogy az adatokban hiba van, a feltüntetett élek költségeinek összege nem egyezik meg azzal az értékkel, amely az igénnyel rendelkező élek súlyainak összegeként meg van adva. Az első futtatáskor kapott eredmények is látványosan jobbak lettek az alsó korlátoknál is, így ezek tesztelésétől eltekintettem.

Az egy depós tesztek összefoglalásaként megállapítható, hogy a program a legnagyobb *egl* adatok kivételével mindegyik esetben optimális, vagy közel optimális megoldást talált a feladatra.

3. táblázat. *egl* adatok tesztje

Név	V	R	D	K	C	LB	UB	S_{best}	%	I	T
egl-e1-A	77	51	1468	5	305	3548	3548	3548	0	2679	33331
egl-e1-B	77	51	1468	7	220	4487	4498	4498	0	5720	33467
egl-e1-C	77	51	1468	10	160	5580	5595	5609	0,25	57550	30531
egl-e2-A	77	72	1879	7	280	5012	5018	5018	0	6334	81170
egl-e2-B	77	72	1879	10	200	6284	6317	6348	0,49	54772	76923
egl-e2-C	77	72	1879	14	140	8319	8335	8343	0,09	49795	73858
egl-e3-A	77	87	2188	8	280	5898	5898	5898	0	62248	133351
egl-e3-B	77	87	2188	12	190	7711	7775	7801	0,33	91841	134434
egl-e3-C	77	87	2188	17	135	10244	10292	10307	0,14	94325	135238
egl-e4-A	77	98	2453	9	280	6395	6444	6470	0,40	87912	175938
egl-e4-B	77	98	2453	14	180	8935	8961	9026	0,72	75101	167209
egl-e4-C	77	98	2453	19	130	11493	11562	11634	0,62	44610	161766
egl-s1-A	140	75	1394	7	210	5018	5018	5018	0	13576	
egl-s1-B	140	75	1394	10	150	6388	6388	6421	0,51	83116	
egl-s1-C	140	75	1394	14	103	8517	8518	8526	0,09	42166	
egl-s2-A	140	147	3174	14	235	9825	9884	10140	2,59	67205	246779
egl-s2-B	140	147	3174	20	160	13017	13100	13402	2,30	87492	229691
egl-s2-C	140	147	3174	27	120	16407	16425	16746	1,95	90934	272197
egl-s3-A	140	159	3379	15	240	10145	10220	10509	2,82	75003	337190
egl-s3-B	140	159	3379	22	160	13648	13682	13878	1,43	58516	129694
egl-s3-C	140	159	3379	29	120	17163	17188	17549	2,10	99409	322439
egl-s4-A	140	190	4186	19	230	12141	12268	12577	2,51	27000	670728
egl-s4-B	140	190	4186	27	160	16098	16321	16645	1,98	92836	547373
egl-s4-C	140	190	4186	35	120	20430	20481	*21218	3,59	241403	1374939

6.2. Több depós tesztek

6.2.1. Módosított benchmark adatok tesztelése

Az algoritmus több depós tesztelését két lépésben végeztem. Először az egy depós tesztadatokat módosítottam úgy, hogy a K darab robotot egy központi depó helyett az első K csúcsra helyeztem el. Mindegyik esetben ugyanazokat a paramétereket használtam mint az egy depós esetben, hogy a futási idő összehasonlítható legyen. Látható, hogy több depó használata nem okozott változást a futási időkben. Az is megfigyelhető, hogy a kapott eredményeket az eredeti egy depós feladatok eredményeivel összehasonlítva több depó ilyen jellegű használata nem minden esetben okoz jelentős csökkenést a költségben.

4. táblázat. Módosított *kshs* adatok tesztje

Név	V	R	D	K	C	S_{best}	I	T
kshs1-MD	8	15	535	4	150	9617	4	50
kshs2-MD	10	15	497	4	150	8099	10	51
kshs3-MD	6	15	565	4	150	9061	3	48
kshs4-MD	8	15	594	4	150	9851	646	49
kshs5-MD	8	15	443	3	150	10353	13	44
kshs6-MD	9	15	389	3	150	10197	1	60

5. táblázat. Módosított *gdb* adatok tesztje

Név	V	R	D	K	C	S_{best}	I	T
gdb1-MD	12	22	22	5	5	294	18	191
gdb2-MD	12	26	26	6	5	315	157	300
gdb3-MD	12	22	22	5	5	259	77	199
gdb4-MD	11	19	19	4	5	266	63	147
gdb5-MD	13	26	26	6	5	350	90	293
gdb6-MD	12	22	22	5	5	279	20	255
gdb7-MD	12	22	22	5	5	304	72	209
gdb8-MD	27	46	249	10	27	368	473	320
gdb9-MD	27	51	258	10	27	271	*2991	4300
gdb10-MD	12	25	37	4	10	275	20	189
gdb11-MD	22	45	224	5	50	387	185	485
gdb12-MD	13	23	212	7	35	410	133	80
gdb13-MD	10	28	245	6	41	530	987	101
gdb14-MD	7	21	89	5	21	96	22	57
gdb15-MD	7	21	112	4	37	56	7	62
gdb16-MD	8	28	116	5	24	125	88	124
gdb17-MD	8	28	168	5	41	91	2	124
gdb18-MD	9	36	153	5	37	158	37	162
gdb19-MD	8	11	66	3	27	55	6	44
gdb20-MD	11	22	107	4	27	121	97	71
gdb21-MD	11	33	154	6	27	154	227	154
gdb22-MD	11	44	205	8	27	196	492	393
gdb23-MD	11	55	266	10	27	225	*5273	5084

6. táblázat. Módosított *egl* adatok tesztje

Név	V	R	D	K	C	S_{best}	I	T
egl-e1-A-MD	77	51	1468	5	305	3176	16971	33035
egl-e1-B-MD	77	51	1468	7	220	3915	7929	33861
egl-e1-C-MD	77	51	1468	10	160	4590	60195	30612
egl-e2-A-MD	77	72	1879	7	280	476	30584	82605
egl-e2-B-MD	77	72	1879	10	200	5127	89798	81351
egl-e2-C-MD	77	72	1879	14	140	6157	50982	76428
egl-e3-A-MD	77	87	2188	8	280	5004	19323	143980
egl-e3-B-MD	77	87	2188	12	190	6092	64671	133983
egl-e3-C-MD	77	87	2188	17	135	7254	96656	143914
egl-e4-A-MD	77	98	2453	9	280	5427	26601	190541
egl-e4-B-MD	77	98	2453	14	180	6756	32364	183397
egl-e4-C-MD	77	98	2453	19	130	8309	95162	166053
egl-s1-A-MD	140	75	1394	7	210	3992	24015	66476
egl-s1-B-MD	140	75	1394	10	150	4626	55347	53635
egl-s1-C-MD	140	75	1394	14	103	6098	59038	47767
egl-s2-A-MD	140	147	3174	14	235	7225	38579	305478
egl-s2-B-MD	140	147	3174	20	160	10127	68178	246138
egl-s2-C-MD	140	147	3174	27	120	13631	43237	290314
egl-s3-A-MD	140	159	3379	15	240	7934	32218	409217
egl-s3-B-MD	140	159	3379	22	160	10628	98713	365357
egl-s3-C-MD	140	159	3379	29	120	13909	84165	362423
egl-s4-A-MD	140	190	4186	19	230	10096	27180	602484
egl-s4-B-MD	140	190	4186	27	160	14127	88895	566427
egl-s4-C-MD	140	190	4186	35	120	19600	35001	625008

6.2.2. Grid gráfokon értelmezett feladatok tesztelése

A több depós tesztek második csoportját grid gráfokon végeztem. A paramétereket a korábbiakkal megegyezően választottam, csak az iterációszámot állítottam 10000-re. A gráfokban minden csúcs költsége és igénye 1, a járművek számát, kapacitását és elhelyezését változtattam a tesztelés során. A részletes tesztelésre 6x10 méretű gráfot választottam, de ennél nagyobb 24x16-os gráfra kapott megoldás is látható a fejezet végén a 38. ábrán.

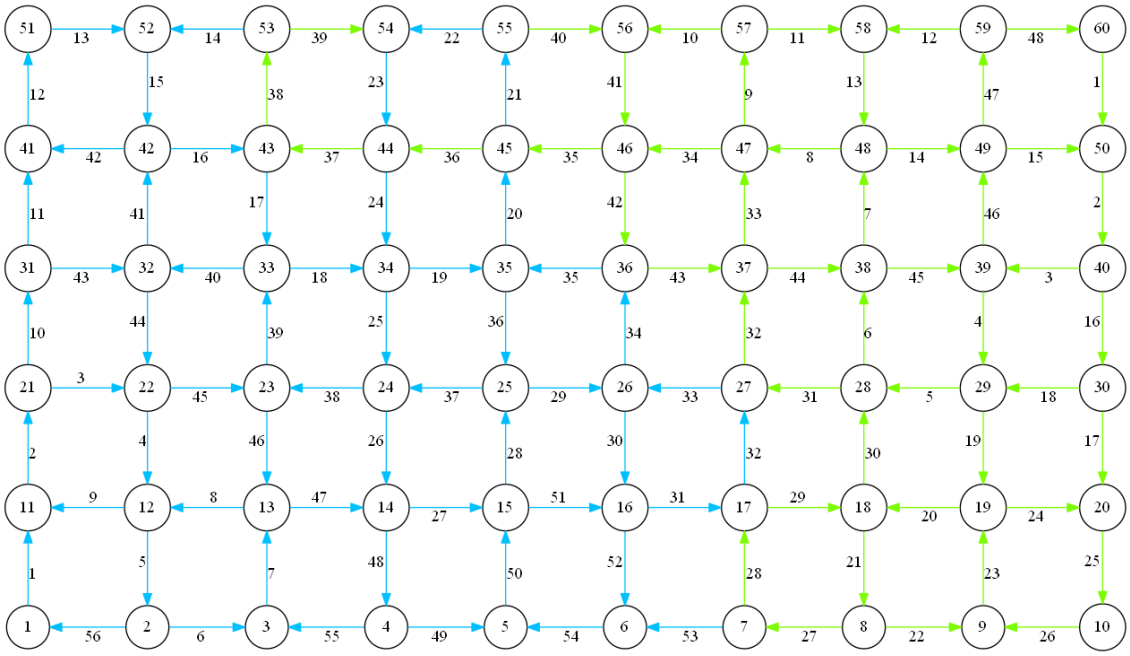
A 6x10-es gridben 104 él található, ezekhez ugyanennyi igény tartozik. Az eredményeket táblázatban foglaltam össze. Az első 4 esetben a depókat az 1-es és 60-as

csúcsban jelöltem ki. (26,27,28,29 ábrák) A következő 4 esetben 3 depót helyeztem el a gráf 3 sarkában az 1-es, 10-es, 60-as csúcsokban.(30,31,32,33 ábrák) Ezek után 3 esetben 4 depót helyeztem el a gráf 4 sarkában.(34,35,36 ábrák) Végül a gráf belső részeire helyeztem 4 depót, a 13, 18, 43, 48 csúcsokba, és mindegyikbe 1-1 30 kapacitású robotot, ez látható a 37. ábrán.

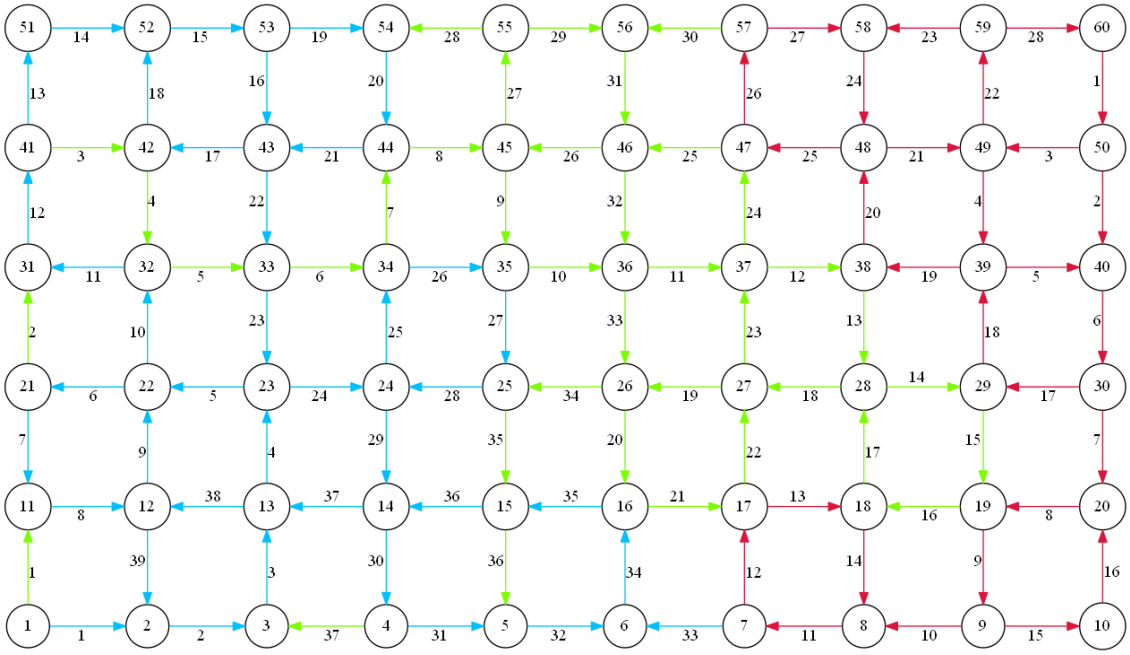
7. táblázat. 6x10-es méretű grid tesztje

Ábra sorszáma	Depók	E	K	C	S_{best}	I	T
26	1,60	1,1	2	60	116	1186	149854
27	1,60	2,1	3	40	118	1321	96555
28	1,60	2,2	4	30	120	2362	49119
29	1,60	3,1	4	30	126	3171	81265
30	1,10,60	1,1,1	3	40	116	2772	103980
31	1,10,60	2,1,1	4	30	118	4029	99550
32	1,10,60	2,1,2	5	24	120	5879	71925
33	1,10,60	2,2,2	6	20	130	3915	42264
33	1,10,51,60	1,1,1,1	4	30	118	3200	93392
35	1,10,51,60	2,1,1,1	5	24	120	7225	100470
36	1,10,51,60	2,1,1,2	6	20	122	8230	73624
37	13,18,43,48	1,1,1,1	4	30	120	8508	100923

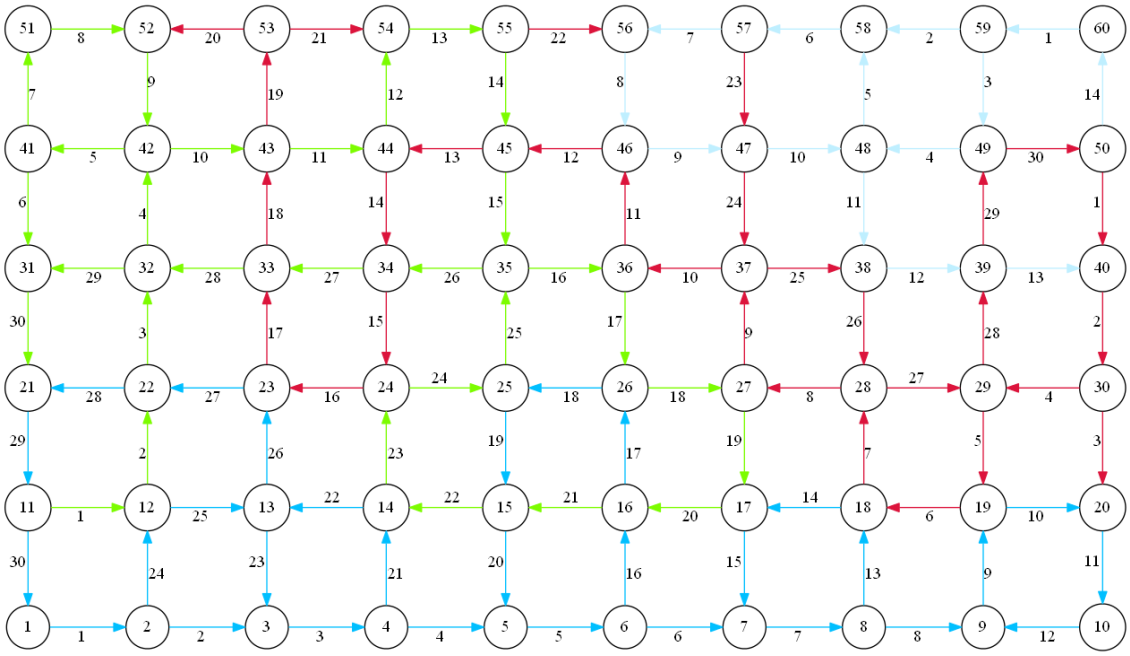
Az eredményekkel kapcsolatban megállapítható, hogy nem okoz túl nagy eltérést a járművek számának, kapacitásainak és elhelyezésének változtatása. Egy kis tendencia megfigyelhető azzal kapcsolatban, hogy a járművek kapacitásának csökkenése esetén nem minden esetben tudnak a távoli élek kiszolgálásához szükségtelen élek közbeiktatása nélkül eljutni. Nagyobb gráfok részletes vizsgálata esetén előfordulhat, hogy ezek a tendenciák felerősödnek, erre azonban a rendelkezésemre álló számítási kapacitással nem volt lehetőségem.



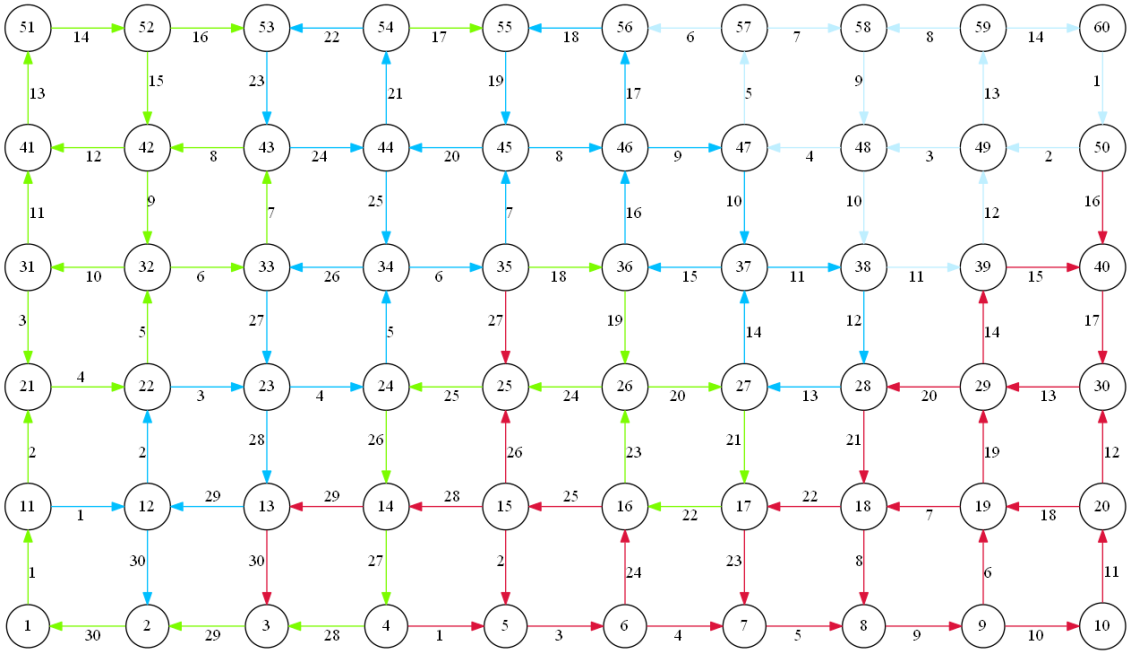
26. ábra. 2 depó, 1-1 robot 60 kapacitással



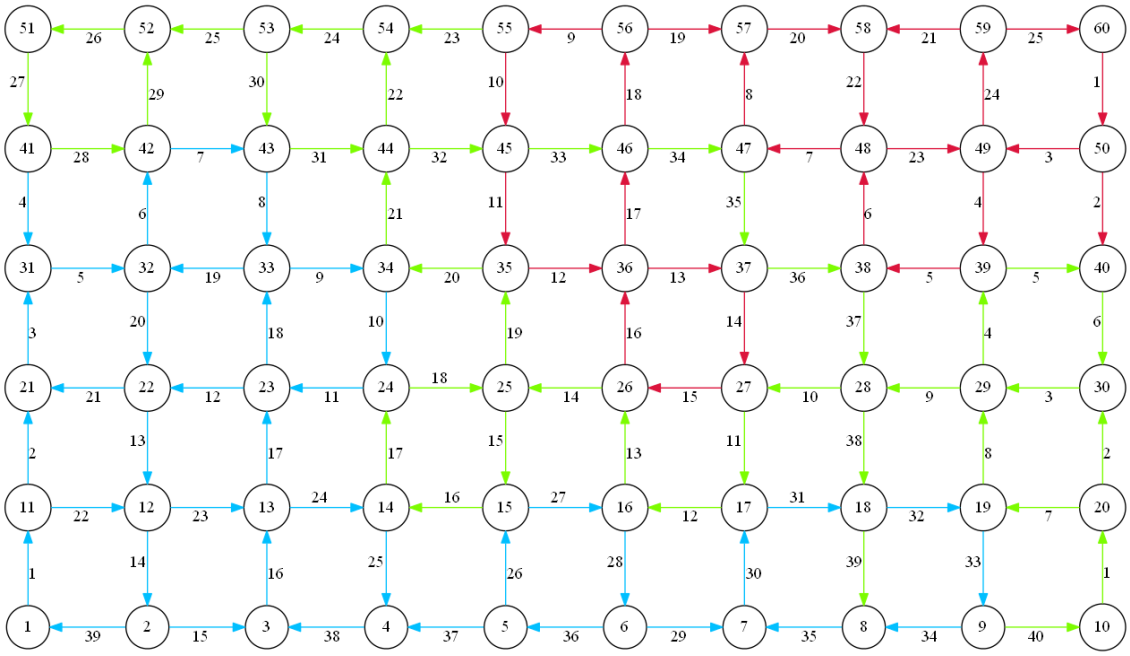
27. ábra. 2 depó, 2 illetve 1 robot 40 kapacitással



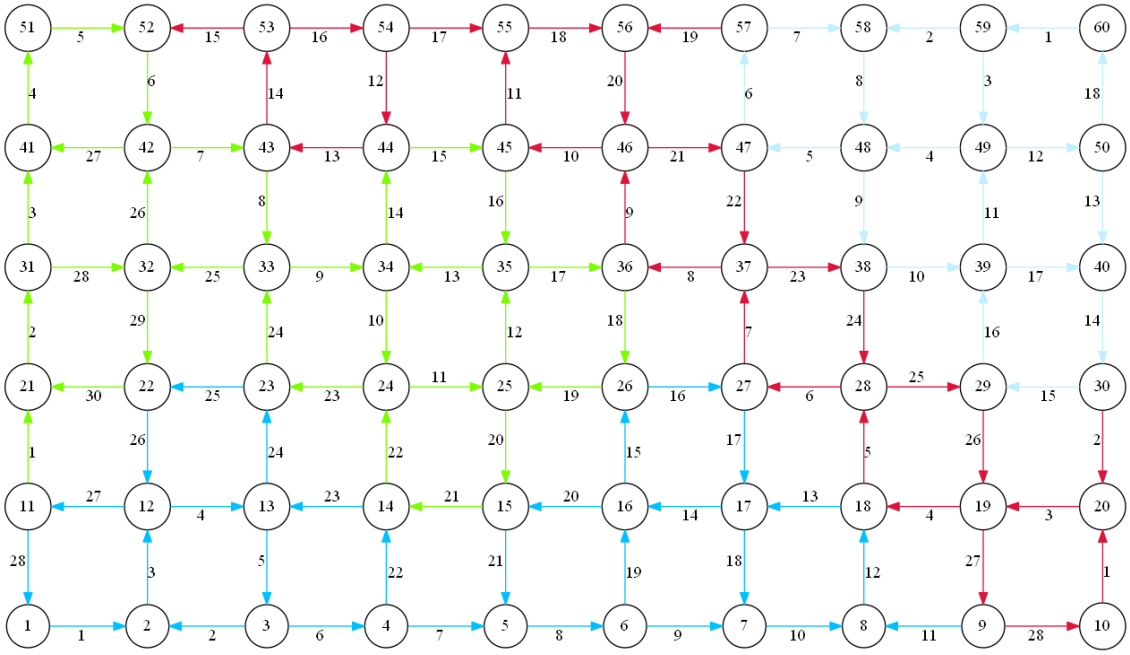
28. ábra. 2 depó, 2-2 robot 30 kapacitással



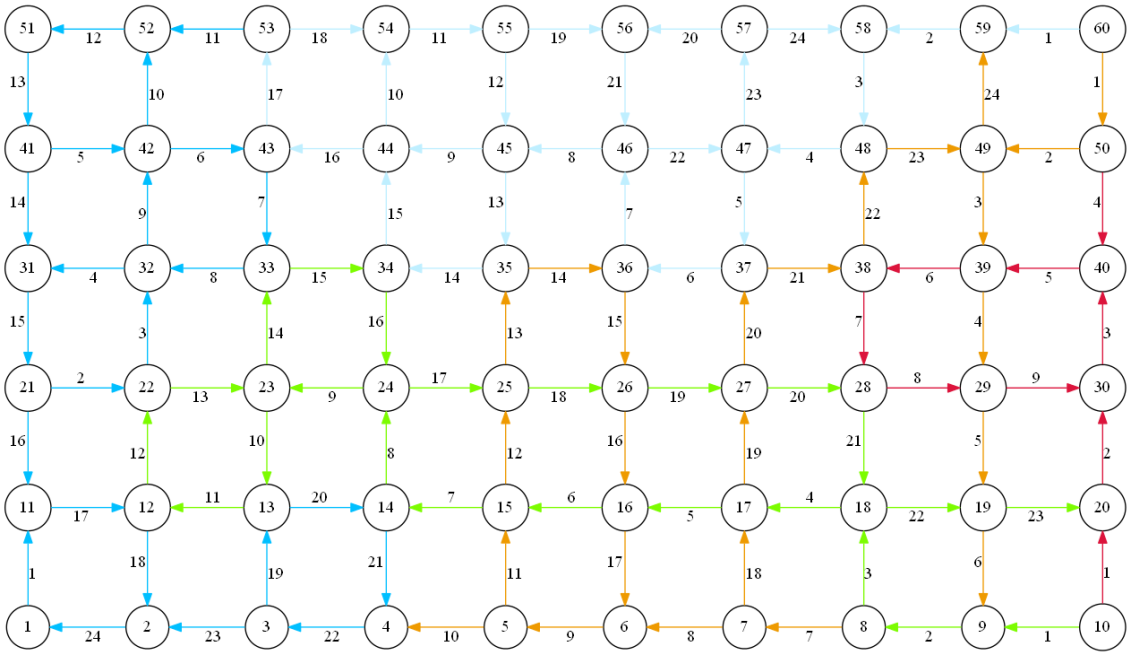
29. ábra. 2 depó, 3 illetve 1 robot 30 kapacitással



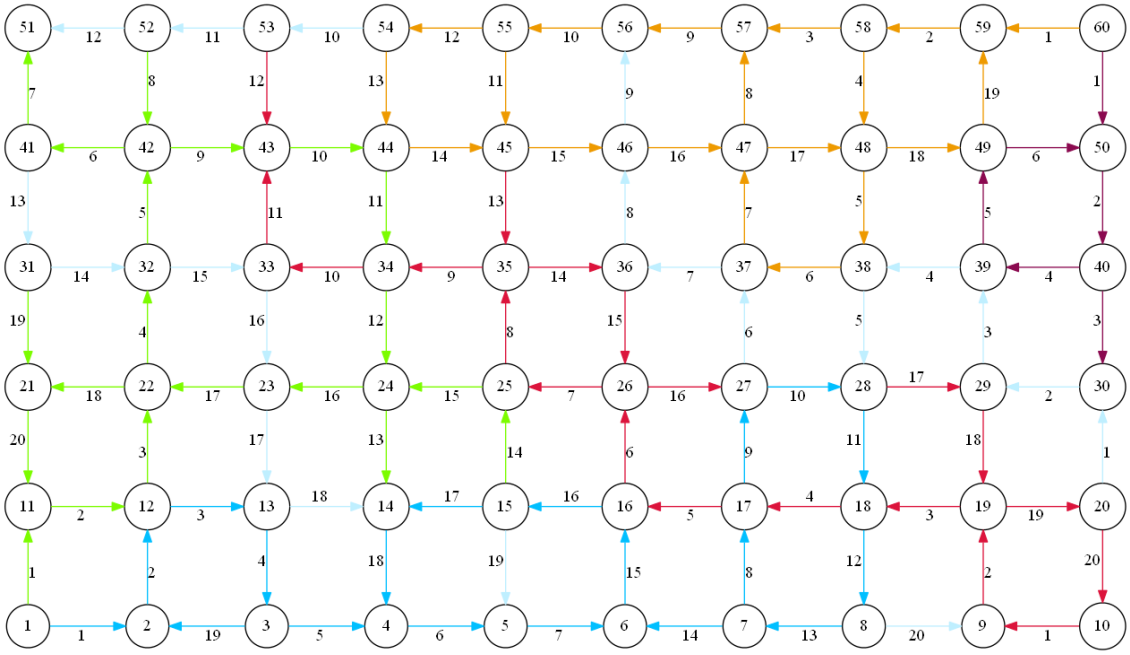
30. ábra. 3 depó, 1-1-1 robot 40 kapacitással



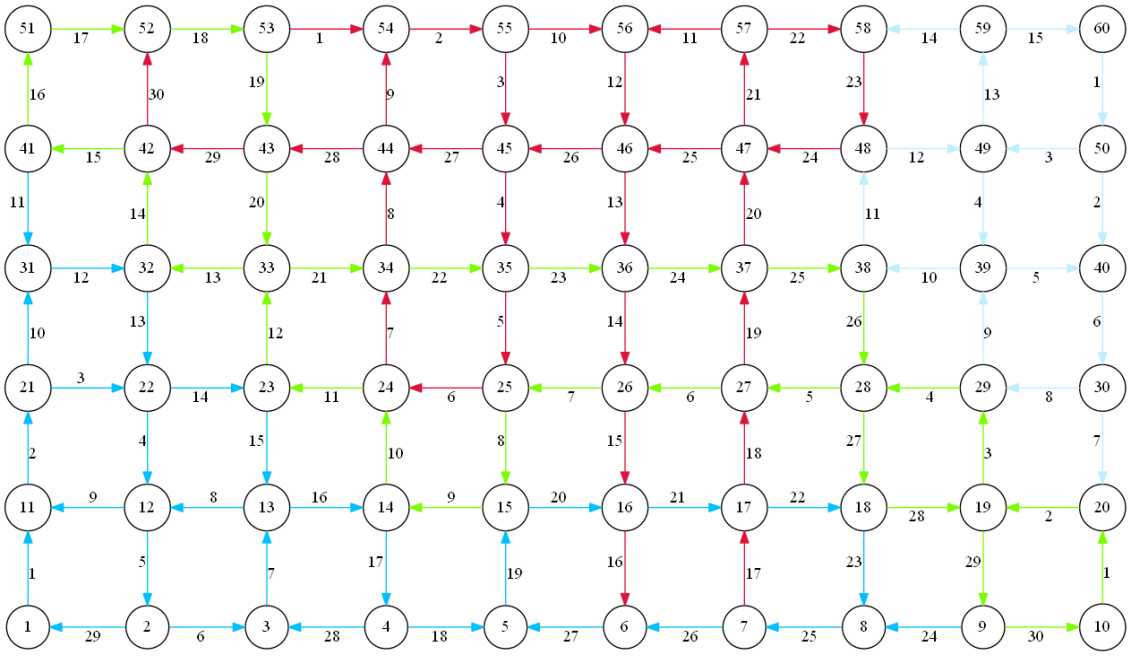
31. ábra. 3 depó, 2-1-1 robot 30 kapacitással



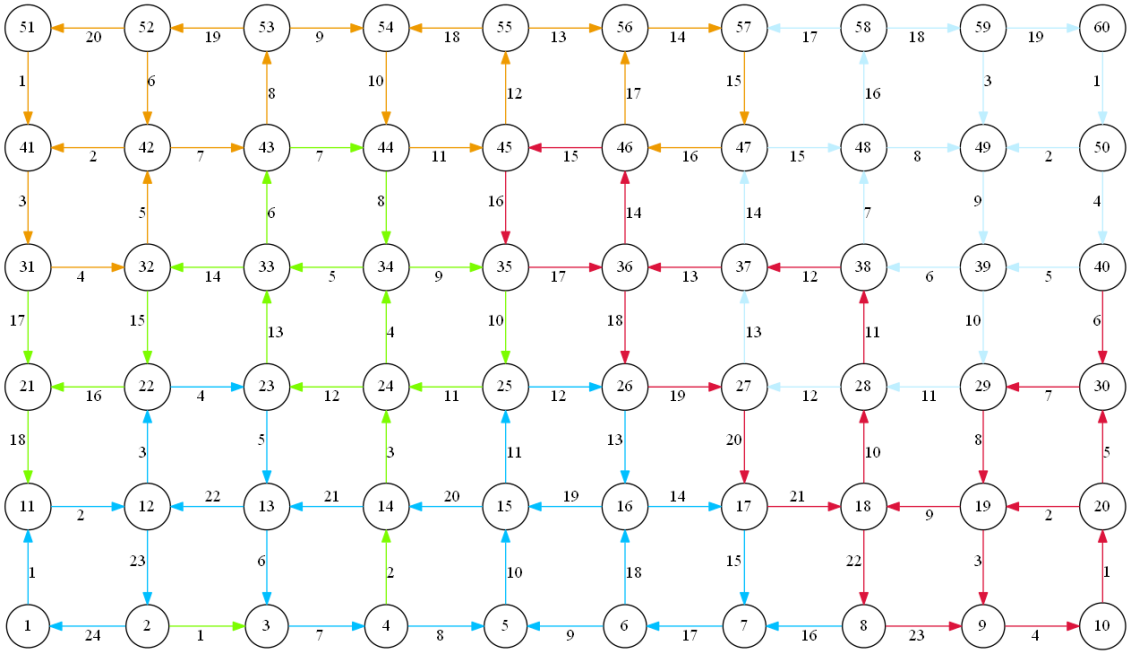
32. ábra. 3 depó, 2-1-2 robot 24 kapacitással



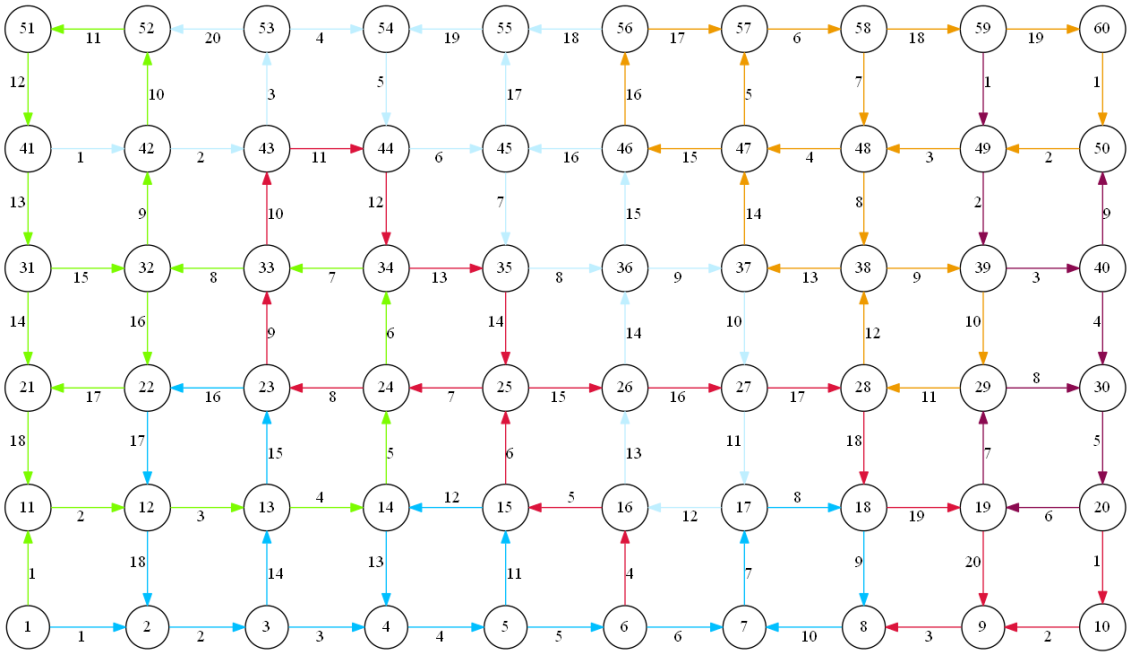
33. ábra. 3 depó, 2-2-2 robot 20 kapacitással



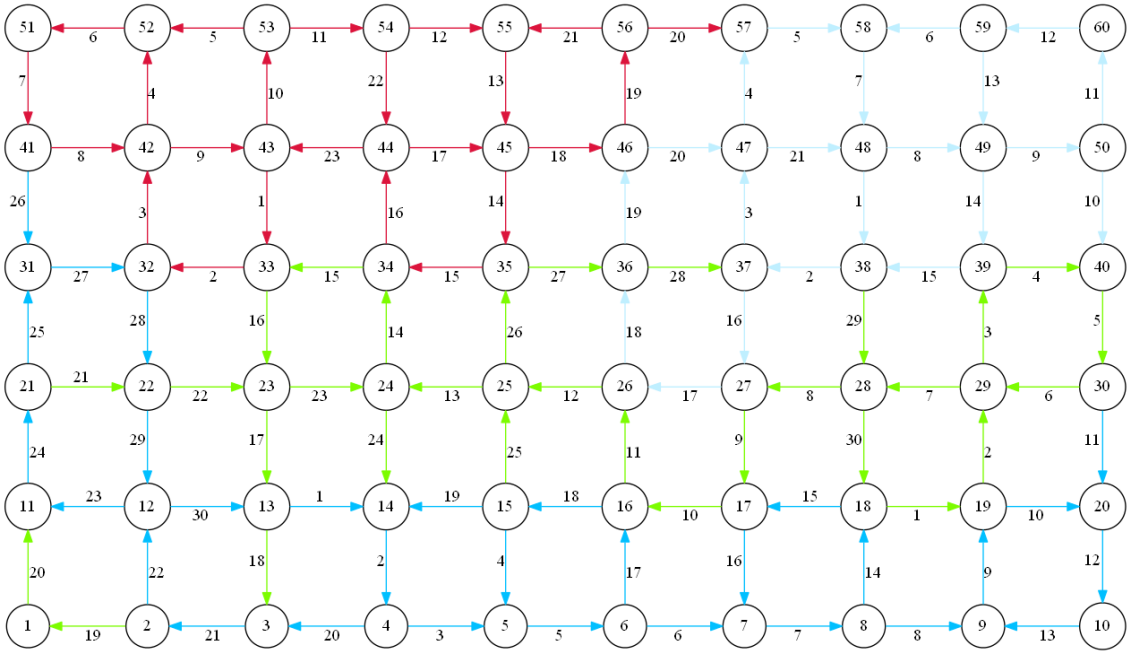
34. ábra. 4 depó, 1-1-1-1 robot 30 kapacitással



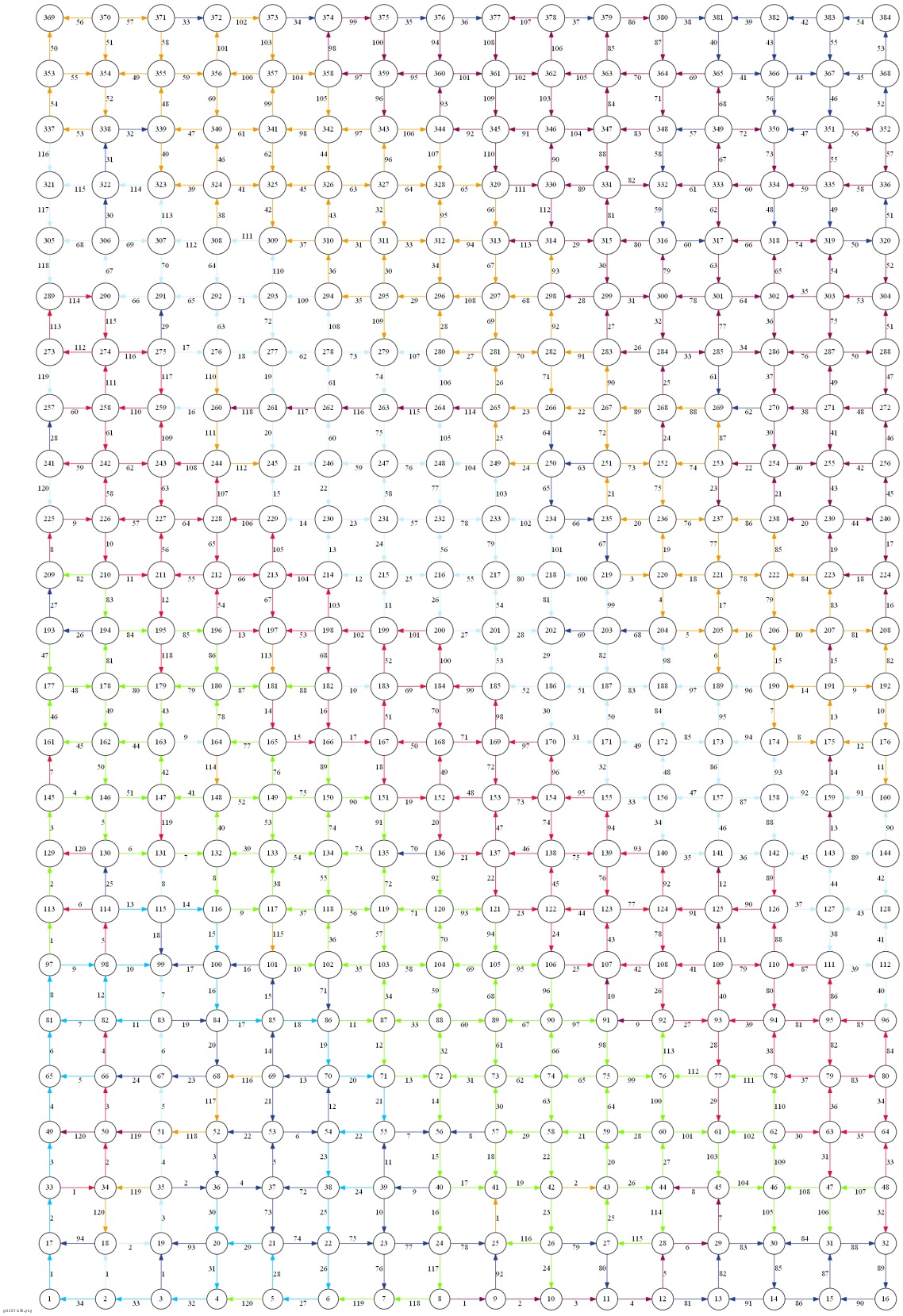
35. ábra. 4 depó, 2-1-1-1 robot 24 kapacitással



36. ábra. 4 depó, 2-1-1-2 robot 20 kapacitással



37. ábra. 4 depó a 13, 18, 43, 48 csúcsokban, 1-1-1-1 robot 30 kapacitással



38. ábra. 24 x 16 méretű grid megoldása

7. Összefoglalás

A munkám egyik célja volt a gráfok éleinek bejárásával kapcsolatos klasszikus problémák áttekintése. A königsbergi hidak problémájától jutottam el a súlyozott élbejárási feladatig, a CARP-ig, amely szinte mindegyik egyéb élbejárási feladat általánosításának tekinthető. A CARP-pal kapcsolatban megfogalmaztam a feladat pontos matematikai modelljét, és bemutattam a XX. század második feléből származó klasszikus megoldási módszereket. A Construct and Strike, Path-Scanning, és Augment-Merge algoritmusok működését példákkal is illusztráltam. Ezek mellett megemlítettem az utóbbi 10 évből származó friss kutatások eredményeit a feladatot megoldó metaheurisztikákkal, és a megoldásra alsó korlátot szolgáltató eljárásokkal kapcsolatban is. A teljesség igénye nélkül felsoroltam számos olyan gyakorlati, logisztikai problémát, amelyek a CARP valamilyen változatának tekinthető feladatot definiálnak. Ezek egyikének, a több depós CARP-MD feladatnak megfogalmaztam a matematikai modelljét, és megemlítettem a vele kapcsolatos kutatási eredményeket.

A másik célom az volt, hogy bemutassak egy eljárást a több depós CARP feladat megoldására. A megoldási módszer ötletét egy másik logisztikai feladat, a CVRP egyik megoldása szolgáltatta. Megmutattam, hogy az eljárás eredményeként kapott megoldások miért megoldásai a CARP-nak. Részletesen ismertettem az eljárás felépítését, és az egyes részeinek működését. Az algoritmust megvalósító, általam írt program eredményeit összehasonlítottam a klasszikus CARP feladathoz létező ismert benchmark-adatok segítségével. A program segítségével a legtöbb feladathoz megtaláltam, és a többi esetben is megközelítettem az ismert legjobb megoldást. Az algoritmus több depós teszteléséhez az egy depós mintaadatok módosított változatait, valamint grid gráfokat használtam. Megállapítottam, hogy az algoritmus futási ideje az egy- és a több depós esetben lényegében megegyezik.

Összességében a szakdolgozatom célját sikerült teljesítenem, működő, használható eljárást mutattam be a több depós CARP megoldására.

Hivatkozások

[1] A. Amaya, A. Langevin, and M. Trépanier. The capacitated arc routing problem with refill points. *Operations Research Letters*, 35(1):45–53, 2007.

[2] A. Amberg, W. Domschke, and S. Voß. Multiple center capacitated arc routing problems: A tabu search algorithm using capacitated trees. *European Journal of Operational Research*, 124:360–376, 2000.

[3] A. Amberg and S. Voß. A hierarchical relaxations lower bound for the capacitated arc routing problem. In *Proceedings of the 35th Hawaii International Conference on System Sciences*, pages 1415–1424, 2002.

[4] A. A. Assad and B. L. Golden. Arc routing methods and applications. In *Handbooks in Operations Research and Management Science*, volume 8, pages 375–483. Elsevier, 1995.

[5] A.A. Assad, B.L. Golden, and W.L. Pearn. Transforming arc routing into node routing problems. *Computers and Operations Research*, 14(4):285–288, 1987.

[6] E.K. Baker, J.S. DeArmon, and B.L. Golden. Computational experiments with algorithms for a class of routing problems. *Computers and Operations Research*, 10(1):47–59, 1983.

[7] R. Baldacci and V. Maniezzo. Exact methods based on node routing formulations for undirected arc routing problems. *Networks*, 47(1):52–60, 2006.

[8] J.M. Belenguer. http://www.uv.es/ belengue/carp.html.

[9] J.M. Belenguer and E. Benavent. The capacitated arc routing problem: Valid inequalities and facets. *Computational Optimization and Applications*, 10(2):165–187, 1998.

[10] J.M. Belenguer and E. Benavent. A cutting plane algorithm for the capacitated arc routing problem. *Computers and Operations Research*, 30:705–728, 2003.

[11] E. Benavent, V. Campos, A. Corberan, and E. Mota. The capacitated arc routing problem: Lower bounds. *Networks*, 22:669–690, 1992.

[12] J. Brandao and R.W. Eglese. A deterministic tabu search algorithm for the capacitated arc routing problem (carp). *Computers and Operations Research*, 35(4):1112–1126, 2008.

[13] N. Christofides. The optimum traversal of a graph. *OMEGA, The International Journal of Management Science*, 1(6):719–732, 1973.

[14] B. Crevier, J-F. Cordeau, and G. Laporte. The multi-depot vehicle routing problem with inter-depot routes. *European Journal of Operational Research*, 176:756–773, 2007.

[15] E., J-F. Cordeau, and G. Laporte. Improved lower bounds and exact algorithm for the capacitated arc routing problem. *Mathematical Programming*, 2011.

[16] J. Edmonds and E.L. Johnson. Matching, euler tours and the chinese postman. *Mathematical Programming*, 5(1):88–124, 1973.

[17] R.W. Eglese. Routeing winter gritting vehicles. *Discrete Applied Mathematics*, 48(3):231–244, 1994.

[18] R.W. Eglese and L.Y.O. Li. An interactive algorithm for vehicle routeing for winter-gritting. *Journal of the Operational Research Society*, 47:217–228, 1996.

[19] H.A. Eiselt, M. Gendreau, and G. Laporte. Arc routing problems, part 1: The chinese postman problem. *Operations Research*, 43(2):231–242, 1995.

[20] M. Fleury. Deux problemes de geometrie de situation. *Journal de Mathematiques Elementaires*, pages 257–261, 1883.

[21] R.W. Floyd. Algorithm 97: shortest path. *Communications of the ACM*, 1962.

[22] G.N. Frederickson. Approximation algorithms for some postman problems. *Journal of Association fot Computing Machinery*, 26(3):538–554, 1979.

[23] G.N. Frederickson, M.S. Hecht, and C.E. Kim. Approximation algorithms for some routing problems. *SIAM Journal of Computing*, 7(2):178–193, 1978.

[24] B.L. Golden and R.T. Wong. Capacitated arc routing problems. *Networks*, 11:305–315, 1981.

[25] A. Hertz. Recent trends in arc routing. In *Graph theory, Combinatorics and algorithms: Operations research/computer science interfaces series.* M.C. Golumbic and I.B.A Hartman, 2005.

[26] A. Hertz, G. Laporte, and M. Mittaz. A tabu search heuristic for the capacitated arc routing problem. *Operations Research*, 48(1):129–135, 2000.

[27] R. Hirabayashi, N. Nishida, and Y. Saruwatari. Tour construction algorithm for the capacitated arc routing problem. *Asia-Pacific Journal of Operational Research*, 9:155–175, 1992.

[28] A.H.G. Rinnooy Kan and J.K. Lenstra. On general routing problems. *Networks*, 6:273–280, 1976.

[29] M. Kiuchi, R. Hirabayashi, Y. Saruwatari, and Y. Shinano. An exact algorithm for the capacitated arc routing problem using parallel branch and bound method. *Abstracts of the 1995 Spring National Conference of the Oper. Res. Soc. of Japan*, pages 28–29, 1995.

[30] P. Lacomme, C. Prins, and W. Ramdane-Cherif. Competitive memetic algorithms for arc routing problems. *Annals of Operations Research*, 131:159–185, 2004.

[31] A. N. Letchford, R. W. Eglese, and J. Lysgaard. A new branch-and-cut algorithm for the capacitated vehicle routing problem. *Mathematical Programming*, 100:423–445, 2004.

[32] L.Y.O. Li and R.W. Eglese. An interactive algorithm for vehicle routeing for winter - gritting. *The Journal of the Operational Research Society*, 47(2):217–228, 1996.

[33] Kwan Mei-Ko. Graphic programming using odd or even points. *Chinese Mathematics*, 1:237–277, 1962.

[34] C.S. Orloff. A fundamental problem in vehicle routing. *Networks*, 4:35–64, 1974.

[35] C.H. Papadimitriou. On the complexity of edge traversing. *Journal of the Association for Computing Machinery*, 23(3), 1976.

[36] W.L. Pearn. *The capacitated Chinese postman problem.* PhD thesis, University of Maryland, 1984.

[37] W.L. Pearn. New lower bounds for the capacitated arc routing problem. *Networks*, 18(3):181–191, 1988.

[38] W.L. Pearn. Approximate solutions for the capacitated arc routing problem. *Computers and Operations Research*, 16(6):589–600, 1989.

[39] M. Polacek, K.F. Doerner, R.F. Hartl, and V. Maniezzo. A variable neighborhood search for the capacitated arc routing problem with intermediate facilities. *Journal of Heuristics*, 14(5):405–423, 2008.

[40] S. Ropke and D. Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 2005.

[41] H. Sachs, M. Stiebitz, and R.J. Wilson. An historical note: Eulers königsberg letters. *Journal of Graph Theory*, 12(1):133–139, 1988.

[42] S. Wohlk. New lower bound for the capacitated arc routing problem. *Computers and Operations Research*, 33(12):3458–3472, 2006.

[43] S. Wohlk. An approximation algorithm for the capacitated arc routing problem. *The Open Operational Research Journal*, 2:8–12, 2008.

[44] S. Wohlk. A decade of capacitated arc routing problem. In *The Vehicle Routing Problem : Latest Advances and New Challenges*. Springer, 2008.