E Ö T V Ö S L O R Á N D T U D O M Á N Y E G Y E T E M
I N F O R M A T I K A I K A R
T A N U L M Á N Y I H I V A T A L

1117 Budapest, Pázmány Péter sétány 1/C. Tel: 372-2517

Ikt. sz.: IK/3470/1 (2024)

Határozat

Kebelei Csaba programtervező informatikus (MSc) hallgató *"*Tervezési stratégiák a periodikus hulladékgyűjtés preferált napok szerinti optimalizálásához*"* című TDK dolgozata az Informatikai Kar 2024. május 29-én tartott TDK konferenciáján 2. helyezést ért el. Ezzel jogot szerzett a 2025. évi 37. OTDK-n való részvételre. Az Informatikai Kar Kari Tanácsának 27/2015 (V.12.) számú határozata alapján ("Határozat a TDK dolgozatok kari elismerési rendjéről") a fent említett dolgozatot az Informatika Kar elfogadja diplomamunkának jeles minősítéssel.

Budapest, 2024. május 30.

Dr. Krebsz Anna
oktatási dékánhelyettes

Eötvös Loránd Tudományegyetem

Informatikai Kar

Mesterséges Intelligencia Tanszék

Tervezési stratégiák a periodikus hulladékgyűjtés preferált napok szerinti optimalizálásához

Szerző:
Kebelei Csaba
programtervező informatikus MSc

Külső témavezető:
Dr. Dávid Balázs
egyetemi adjunktus
University of Primorska

Belső témavezető:
Dr. Tóth Tekla
egyetemi adjunktus
Eötvös Loránd
Tudományegyetem

Budapest, 2024

Absztrakt

Manapság a hulladékgyűjtés hatékony szervezése kiemelt fontossággal bír. A hulladékgyűjtő járművek útvonalainak és a begyűjtések időzítésének megfelelő tervezésével jelentős mennyiségű erőforrást és költséget takaríthatunk meg. Az útvonalak tervezése történhet egyetlen napra vagy hosszabb időszakokra, azaz periódusokra is. Periodikus tervezés esetén a begyűjtések időpontja sok gyakorlati esetben ugyanarra az adott időintervallumra esik minden periódusban. Kutatásom célja megvizsgálni, hogy milyen előnyökkel járhat, ha némi rugalmasságot engedélyezünk az ilyen preferált begyűjtési időpontokhoz képest.

A munkám a többperiódusú jármű-útvonaltervezési probléma (multi-period vehicle routing problem) egy változatát vizsgálja. A probléma tervezési időszakja P egymást követő periódust tartalmaz, amelyek mindegyike D napból áll. Minden ügyfél számára adott egy preferált nap, amelyen minden egyes periódusban várják a látogatást. Egy ügyfél meglátogatása a preferált napokon kívül is megengedett, azonban ez az eredeti ütemtervtől való eltérés büntetéssel jár. Továbbá, mivel az ügyfelek az idő elteltével folyamatosan hulladékot halmoznak fel, be kell vezetni egy maximálisan megengedett időtartamot az ugyanazon ügyfélnél tett két egymást követő látogatás között. A feladat megoldásához először az egyes ügyfelek látogatási napjait kell meghatározni, majd a tervezési időszak minden egyes napjára optimális járműútvonalakat kell kialakítani.

A fenti probléma megoldására egy adaptív nagy szomszédsági kereső módszert (adaptive large neighborhood search) terveztem. Ez a módszer figyelembe veszi mind az egyes időszakokban az ügyfelek látogatási napjainak kijelölését, mind pedig az egyes napok járműútvonalainak optimalizálását. A heurisztika hatékonyságának validációja valós adatok alapján generált bemeneteken történik.

Tartalomjegyzék

1. fejezet

Bevezetés

A jármű-útvonaltervezés (Vehicle Routing Problem - VRP) az operációkutatás terén rendkívül széles körben vizsgált téma. A probléma a klasszikus utazó ügynök probléma (Travelling Salesman Problem - TSP) egy általánosítása, ahol a feladat az ügyfelek egy halmazának meglátogatása, és erre több ügynök (általában járművek) is rendelkezésünkre állhat. Míg a TSP esetén egyetlen körutat kell terveznünk az ügyfelek között, a VRP-nél általában több körút is engedélyezett a rendelkezésünkre álló járműszám függvényében. A szakirodalomban a VRP számos változatáról és ezek gyakorlati életben történő használatáról találhatunk kutatásokat. Egy ilyen jelentős valós alkalmazási terület a hulladékgyűjtés során felmerülő útvonaltervezési probléma, ahol gyűjtőpontokról (ezek tekinthetőek az ügyfeleknek) kell a lehető legrövidebb útvonalakon elszállítani a hulladékot a rendelkezésünkre álló járművekkel.

Az útvonaltervezési kérdések alapjában véve is nehéz kombinatorikai optimalizálási problémákat jelentenek, de további korlátozásokkal bővítve még bonyolultabbá válhatnak. A valós életben előforduló változataik általában maguk után vonnak ilyen plusz korlátokat, mint például a járművek maximum tároló kapacitása, vagy a körutak maximális hossza. Ezek fontos korlátozásoknak számítanak a hulladékszállítás során is, kiegészülve további olyan feltételekkel, mint a hulladékgyűjtő jármű maximális kapacitása, vagy hogy az összegyűjtött hulladékot specifikus hulladéklerakó telephelyekre kell elszállítani.

Míg a VRP általánosan egyetlen napra történő útvonaltervezést jelent, valós problémák esetén azonban sok esetben hosszú távú tervezéssel kell számolni, mely több hétre vagy akár hónapokra is kiterjedhetnek. Az ilyen típusú problémák a periodikus, vagy a többperiódusú jármű-útvonaltervezési problémák közé tartoznak.

Periodikus esetben egy adott ügyfél igényének teljes kielégítéséhez több látogatás is szükséges egy időintervallumon (pl. egy hét) belül, míg többperiódusú esetben ezt egymás után többször, több perióduson (pl. több hét) keresztül meg kell tenni, és a periódusok között további korlátozások merülhetnek fel. Perióduson belüli döntés például, hogy egy adott ügyfélre vonatkozó látogatás mely napokra időzítjük a héten. Periódusok közötti korlátozás lehet, hogy maximálisan mennyi idő telhet el két látogatás között egy ügyfélnél. Jelen dolgozatban a fenti tényezőkön túlmenően azt vizsgálom, hogy milyen előnyökkel járhat, ha az ügyfelek által preferált begyűjtési időpontokat nem fixnek tekintjük, hanem engedélyezett egy némi rugalmasság is az ügyfelek meglátogatását illetően. Ez a rugalmasság azt jelenti, hogy a preferált látogatási naphoz képest egy kis időbeli eltérés még megengedett, az ilyen eltérésekhez azonban büntetési költség társul.

A VRP különböző variánsainak kutatása egy kiterjedt tudományos terület, számos létező megoldási módszertannal. Ezeknek a módszereknek az alkalmazhatósága nagyban függ a probléma méretétől, az esetleges speciális korlátaitól és azok bonyolultságától. Léteznek egzakt módszerek, melyek garantáltan megtalálják a legjobb megoldást, azonban ezek csak kisebb méretű problémákra alkalmazhatóak. Bonyolult feladatosztályok esetén a heurisztikus módszerek az elterjedtebbek, melyek gyorsabban találnak jó megoldást, azonban nem garantálják a globális optimum megtalálását. A kutatásomban adaptív nagy szomszédsági keresés (Adaptive Large Neighborhood Search - LNS) módszert használtam a vizsgált problémaosztályra. Az ALNS egy olyan heurisztika, mely egy kezdeti megoldásból indulva iteratívan jelentős méretű változtatásokat hajt végre a megoldás javítása érdekében, adaptívan módosítva az egyes típusú változtatások valószínűségét a hatékonyságuktól függően.

A kutatásom magába foglalja a javasolt megoldás implementációját, valamint a módszer alkalmazhatóságának tesztelését. A tesztelés során különböző szempontok alapján vizsgáltam a módszer hatékonyságát, különböző méretű adathalmazokon, melyhez egy valós hulladékszállító társaság adatait használtam.

A dolgozatom a 2. fejezetében egy szakirodalmi áttekintés keretében bemutatom az útvonaltervezés problémák alapjait, a VRP problémát és annak kiterjesztéseit, illetve a periodikus VRP irodalmát. A 3. fejezetben az általam vizsgált feladatosztály kerül részletezésre, mind informálisan és mind formálisan definiálva. Ezt követően a 4. fejezetben bemutatásra kerül az ALNS módszertan, melyet a definiált probléma megoldására dolgoztam ki. A módszertan tesztelése során kapott eredmények

az 5. fejezetben kerülnek kiértékelésre. Végezetül a 6. fejezetben összefoglalom a kutatásom eredményeit és a további kutatási irányokat.

2. fejezet

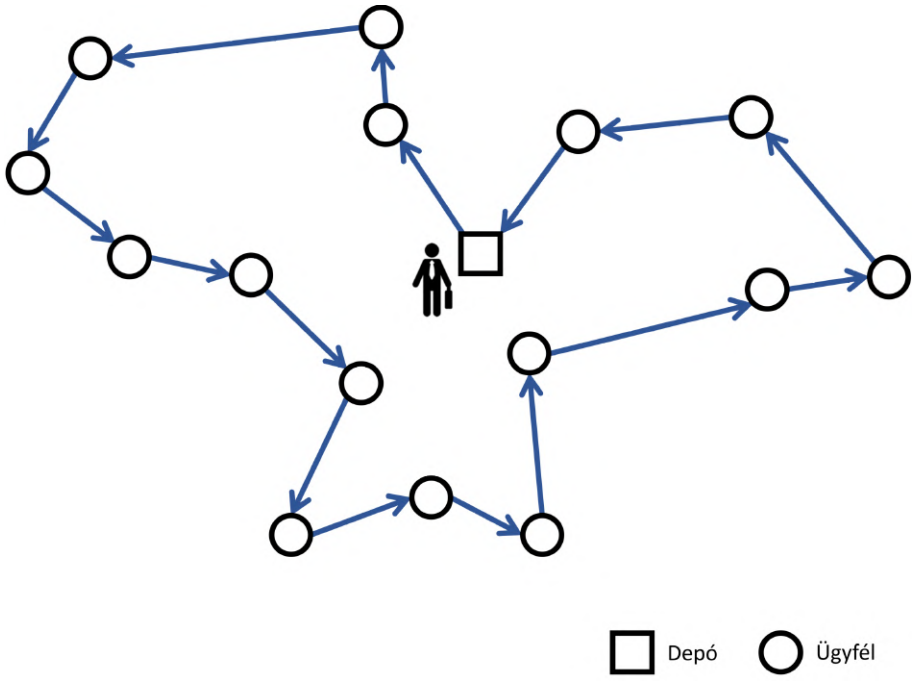
Szakirodalmi áttekintés

2.1. Útvonaltervezési problémák

Az útvonaltervezési problémák az operációkutatás egyik kulcsfontosságú témakörét alkotják. Számos szakterületen találkozhatunk velük, többek között a logisztikában, szemétszállításnál, nyomtatott áramköri lapok fúrása [1] kapcsán, vagy akár a röntgen krisztallográfia [2] esetén is. Ezek a problémák olyan kérdésekkel foglalkoznak, mint például a legrövidebb körutak meghatározása különböző pontok vagy helyszínek között, valamint az erőforrások, például a járművek és az idő optimalizálása.

Az egyik alapvető útvonaltervezési feladat a Karl Menger osztrák matematikus és közgazdász által az 1930-as években formalizált utazó ügynök problémája (Traveling Salesman Problem - TSP) [3], melynél a cél a minimális költségű Hamilton-kör megtalálása. Vagyis az ügynöknek adott városokat pontosan egyszer meg kell látogatnia a lehető legrövidebb útvonalon visszatérve a kiindulási pontba. Erre egy példa a 2.1 ábrán látható, ahol a fekete négyzettel jelölt csúcs a kiindulási pont, és kék színnel jelöltem a városokat összekötő útvonalat.

Richard Karp amerikai informatikus 1972-ben [4] mutatta be, hogy NP-teljes problémáról van szó. A feladat könnyen értelmezhető, de már kis méretű feladat esetén is nagyon sok lehetséges megoldás létezik, így az optimum megtalálása rendkívül nehéz. Ha a kiindulási pont adott és van n város, akkor az összes kombináció $(n-1)!$. Ez azt jelenti, hogy már 15 településsel foglalkozó példában $14! = 87178291200$ variáció létezik, mely közül meg kell határozni a legjobb megoldást. A városok számának növekedésével pedig ez az érték rendkívüli mértékben nő.



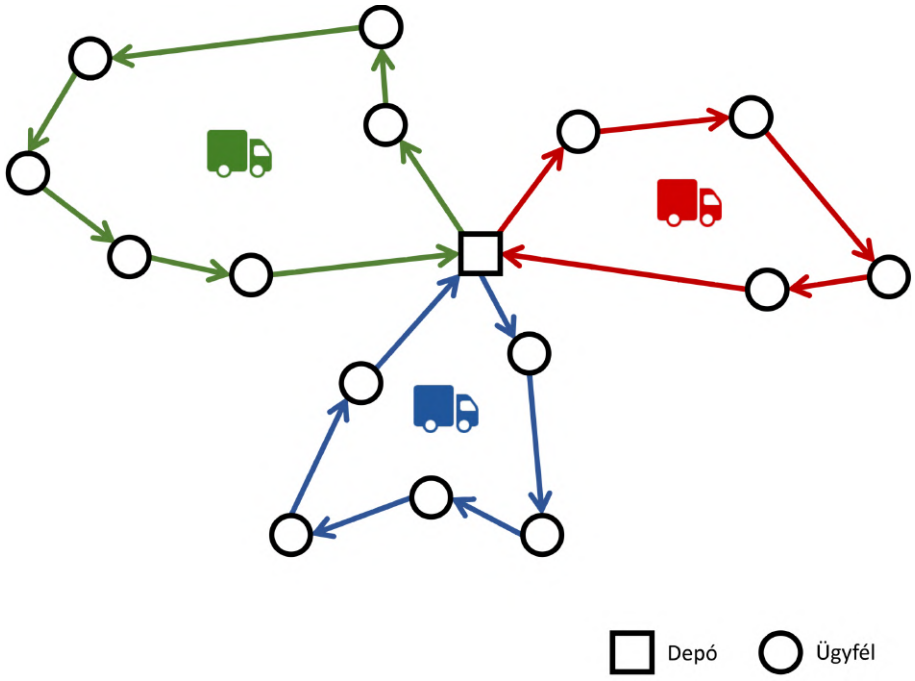
2.1. ábra. Példa az utazó ügynök problémára

A TSP egy általánosított változata a dolgozatomban is vizsgált jármű-útvonaltervezési probléma (Vehicle Routing Problem - VRP), mely nem csupán egyetlen ügynök vagy jármű útvonalát optimalizálja, hanem figyelembe veszi a több jármű alkalmazásának lehetőségét is.

2.2. A jármű-útvonaltervezési probléma

A szakirodalomban először Dantzig és Ramser 1959-ben [5] publikált a problémáról, melyet a teherautó irányítási problémájaként (Truck Dispatch Problem) definiál. A feladat során a cél az volt, hogy kiszolgálják a különböző benzinkutakat úgy, hogy a járművek a lehető legrövidebb úton járják be az összes megállót. Későbbiekben a probléma általánosított változatát fogalmazta meg Clarke és Wright 1964-ben [6]. Ezzel egy olyan lineáris optimalizálási probléma került megfogalmazásra, mellyel gyakrabban találkozhatunk a szállítás és a logisztika területén.

A 2.2 ábrán látható egy példa az általánosabb feladatosztályra. A gráf csúcsai lehetnek depó- és ügyfélcsúcsok. A depócsúcs, melyet fekete négyzettel jelöltem szimbolizálja a járművek kiindulópontját. Ebből a pontból különböző körutakat járhatnak a járművek az ügyfelek között. A példán ezeket az ügyfélcsúcsokat összekötő élek csoportja reprezentálja.



2.2. ábra. Példa a jármű-útvonaltervezési problémára

Az optimális utak meghatározása érdekében egy cél megfogalmazása szükséges, melyet a feladat speciális körülményei befolyásolnak. Ilyen cél lehet például a járművek által megtett távolság minimalizálása, vagy a körutak számának minimalizálása.

A szakirodalomban számos olyan optimalizálási megoldó módszer létezik, melyekkel az útvonaltervezési problémák hatékonyan megoldhatók. A témában széleskörű áttekintést nyújt Laporte [7]. A megfelelő eljárás kiválasztása során az egyik elsődleges szempont a feladat komplexitása, mérete. Egy kisebb probléma esetén érdemes lehet egzakt módszereket alkalmazni, ahol elfogadható időn belül garantálható az optimum megtalálása. Ilyen módszerek gyakori példái a vegyes-egész lineáris programozási modellek (Mixed Integer Linear Programming - MILP), vagy a dinamikus programozás.

Minél bonyolultabb egy feladat annál több időt vesz igénybe a legjobb megoldásnak a megtalálása, mely bizonyos eseteben kivitelezhetetlen a jelenkor technológiai eszközeivel. Ennek elkerülése érdekében gyakran alkalmaznak heurisztikus módszereket, melyek közeli, de nem feltétlenül optimális megoldásokat kínálnak. Ezek az eljárások olyan approximációs stratégiákat alkalmaznak, melyek nem garantálják a globális optimum megtalálását, de gyors és hatékony közelítő megoldásokat adnak. Fő előnyeik közé tartozik, hogy kevesebb számítási erőforrást igényelnek és olyan nagy méretű problémákra is alkalmazhatók, ahol az egzakt módszerek már

nem adnak elfogadható időn belül megoldást. Lenstra és Rinnooy Kan 1981-ben [8] bemutatta, hogy a VRP megoldása NP-nehéz feladat. Emiatt a VRP és annak kiterjesztett még komplexebb variánsai esetén is bevált gyakorlat a különböző heurisztikus módszertanok alkalmazása a fentebb említett előnyök miatt.

Ilyen eljárások lehetnek a különböző evolúciós algoritmusok, melyek az evolúció mechanizmusait veszik alapul. Ennek egy speciális változata a genetikus algoritmus (Genetic Algorithm - GA) [9]. A VRP feladatosztályra Baker és szerzőtársai 2003-ban [10] javasoltak egy ilyen megoldást. Gyakran alkalmazott heurisztikus módszertan a lokális keresés, mely lényege, hogy egy adott megoldásból kiindulva kis változtatásokkal próbáljuk javítani azt. Ennek egy csoportja a szomszédsági keresési eljárások, mely az előre meghatározott megoldás "szomszédjait" kutatja, hogy megtalálhassa a legjobb megoldást. Az egyik ilyen lokális keresés a nagy szomszédsági keresés (Large Neighborhood Search - LNS) [11], melyet Shaw és szerzőtársai 1998-ban javasoltak a VRP problémaosztályra. Az LNS módszer ezen túlmenően hatékonyabb kiegészítéseit is megtalálhatjuk a szakirodalomban. Ilyen többek között az úgynevezett "Adaptive Large Neighborhood Search" (ALNS), melynek a lényege, hogy a keresés során súlyozottan választjuk ki a különféle szomszédságot bejáró metódusokat. Ezzel a módszertannal Pisinger és Ropke 2006-ban [12] oldott meg egy VRP feladatot. Ezenfelül számos heurisztikával találkozhatunk még az irodalomban a jármű-útvonaltervezési problémák esetén, ilyen például a tabu keresés (Tabu Search) [13], vagy a szimulált hűtés (Simulated Annealing) [14].

2.3. Jármű-útvonaltervezési probléma kiterjesztése

Az elmúlt évek kutatásaiban a VRP gyakorlati alkalmazását vizsgálva a különböző területek sajátosságait is figyelembe vették a megoldások során [15]. Többek között ezek olyan kiegészítéseket vontak maguk után, melyek jelentősen befolyásolják a probléma komplexitását, azonban ezáltal a megoldások jóval közelebb állnak a valós esetekhez. A szakirodalom leggyakoribb kiterjesztései az alábbiak:

Capacity Constraints - CVRP Egy alapvető kiegészítésnek tekinthető a kapacitáskorlátozás [16] figyelembevétele, ahol minden árucikkhez egy értéket rendelünk, illetve a járművekhez egy teherbírást. A feladat megoldása során olyan utakat kell definiálnunk, ahol az árufelvétel vagy kiszállítások alatt a járművek teherbírása nem

haladhatja meg a megengedett értéket. Ezen felül adhatunk idő vagy távolság korlátot a körutakra, vagy bevezethetjük a benzinkutat, ahol a járművek feltölthetik az üzemanyagtartályukat, mely adott távolság után kiürül.

Multi-Depot - MDVRP A VRP komplexitása növelhető a több depócsúcs bevezetésével. Ugyanis ilyenkor a járművek nem egyetlen kiindulópontból járhatnak be körutakat, hanem döntést kell hozni arról is, hogy melyik depóból induljanak el. Ezt a kiterjesztést az irodalomban több depós jármű-útvonaltervezési problémának nevezik [17]. A plusz döntési változó jóval bonyolultabb problémát jelent, azonban a gyakorlatban gyakori példa, hogy több telephelyről végzik a kiszállítást, ezzel hatékonyabb tervezést biztosítva.

Pickups and Deliveries - VRPPD A gyakorlatban sokszor találkozhatunk olyan feladattal, ahol nem csak az áruk kiszállításával kell foglalkozni, hanem azok felvételével is. Az ilyen megkötésekkel kiterjesztett VRP-t hívjuk angolul "Vehicle Routing Problem with Pickup and Delivery" feladatosztálynak [18]. Tipikus előfordulás lehet a taxis szolgáltatásoknál, fuvarszállításnál vagy akár a csomagszállítás területén is, ahol a termék vagy utas felvétele és elszállítása különböző helyszínekre történik.

Time Windows - VRPTW A tervezés esetén gyakran kell számolni azzal is, hogy az ügyfelek csak adott intervallumba érhetőek el, például, ha egy üzlet nyitvatartásához kell igazodni. Így a kihívás, hogy a körutak sorrendjét úgy kell optimalizálni, hogy a járművek az ügyfelekhez rendelt időablakon belül érjenek oda. Ezt a problémát hívjuk időablakos jármű-útvonaltervezési problémának [19].

Split Delivery - SDVRP Előfordulhat, hogy egy ügyfélnek több árut kell kiszállítani. Ebben az esetben megoldható az is, hogy egy járművel szállítsuk azt ki, azonban, ha már figyelembe vesszük annak kapacitásait, akkor ez nem feltétlenül eredményez optimális megoldás. Léteznek esetek, mikor jobb megoldás érhető el, ha egy ügyfélnek több rendelkezésre álló jármű külön részletekben juttatja el az árut. Ezt nevezzük angolul "Split Delivery Vehicle Routing Problem" problémának [20].

A gyakorlati problémák gyakran megkövetelik a fentebb említett kiterjesztések különböző kombinációinak alkalmazását. Ezen túlmenően a különféle alkalmazási területek további kiterjesztések bevezetését vagy a klasszikus kiterjesztések speciális

eseteit követelik meg. Ilyen példának tekinthető a hulladékgyűjtési problémánál lévő lerakodó telephelyek bevezetése, mely egy speciális esetét eredményezi a VRPPD-nek. Erre egy hatékony megoldást 2012-ben Buhrkal és szerzőtársai [21] publikáltak.

2.4. Periodikus jármű-útvonaltervezési problémák

A valós útvonaltervezési problémák esetén gyakran hatékonyabb tervezések érhetőek el azzal, hogy a körutakat nem egynapos időintervallumra határozzuk meg, hanem egy hosszabb tervezési időszakra. A feladatdefiníció során ezt az időszakot kisebb periódusokra bontjuk, ahol minden periódusban egy útvonaltervezési feladatot kell megoldani. Ez releváns lehet a TSP esetén is, melyre két heurisztikus megoldást javasolt Paletta az 1992-es publikációjában [22].

A periodikus útvonaltervezés több alkalmazási területen is fontos szempontot jelent. Ilyen tipikusan a szemétszállítás, otthoni egészségügyi ellátás, csomagszállítás vagy akár a biztonsági szolgálat, ahol az őröknek több körutat megtéve kell ellenőrizniük a különböző helyszíneket. Az eltérő alkalmazási területek különféle megkötéseket jelenthetnek, így gyakran másfajta megközelítést igényelnek.

A probléma egy egyszerűbb változatát fogalmazhatjuk meg olyan kiterjesztés bevezetésével, hogy az ügynöknek vagy járműveknek több körutat is meg kell tenniük. Ezt az "M-tour" problémát oldotta meg Russell [23]. A megoldás nem feltételezi, hogy ez különböző napokon történik, így a körutak bármikor megvalósulhatnak a tervezés alatt, így nincs külön döntés arról, hogy mikor melyik ügyfelet szolgáljuk ki.

A feladatot először egy hulladékszállítási problémaként definiálta Beltrami és Bodin 1974-ben [24]. A későbbiekben több publikáció készült a probléma általánosabb változatáról. 1979-ben Russel és Igo [25] fejlesztette tovább Beltrami és Bodin munkáját és 3 heurisztikus megoldást vizsgáltak. Ezen cikk feladatdefiníciójában megfogalmaztak egy külön paramétert, mellyel meghatározott szállítási napot rendeltek az ügyfelekhez. Ezzel szemben legtöbb kutatásban az van kiemelve, hogy hányszor kell kiszolgálni az adott ügyfelet, így a tervezési időszakban különféle lehetséges kombinációkkal dolgoznak. Ilyen korai gyakran hivatkozott mű a témakörben, Christofides és J.E. Beasley 1984-es [26] publikációja. A későbbiekben a problémára több hatékonyabb módszert is kidolgoztak. Többek között ilyen heurisztikát publikáltak Chao és szerzőtársai [27]. Vidal és szerzőtársai 2012-ben [28] egy hibrid

módszert javasoltak, mely a genetikus algoritmus és a szomszédsági keresés kombinációját alkalmazza. Ezek mind hatékony megoldások, azonban nagy különbséget jelent, ha a probléma periodicitását további paraméterekkel kiegészítjük.

Egy másik megfogalmazást vezetett be a problémára Gaudioso és Paletta 1992-ben [29]. Ebben a kutatásban figyelembe vették, mint korlát, hogy hány nap telhet el két látogatás között. Ez egy olyan tényező mely valós problémák esetén sokszor egy fontos igény. Ilyen példa lehet a konzisztens szolgáltatás garantálása. A kis csomagszállító vállalatok általában igyekeznek a lehető legjobb ügyfélkiszolgálást biztosítani. Ez több tényezőt is magával von, mely közül egyik, hogy a sofőrök minél jobb kapcsolatot alakítsanak ki a megrendelőikkel. Ennek érdekében a tervezés során különös hangsúlyt kell fektetni arra, hogy ugyanazok a sofőrök ugyanazokat az ügyfeleket szolgálják ki, nagyjából ugyanabban az időben. A problémát Groër és szerzőtársai 2009-ben [30] konzisztens jármű-útvonaltervezési problémaként (Consistent Vehicle Routing Problem - ConVRP) definiálták. A konzisztens szolgáltatás azonban nem csak a csomagszállító vállaltok esetén lehet fontos, hanem olyan gyakorlati esetekben is, mint például a házi betegápolásnál [31] [32], ahol fontos, hogy a betegeket ugyanazok az ápolók lássák el.

Léteznek problémák, ahol nem kell ilyen szigorú konzisztenciát biztosítani, hanem engedélyezett egy bizonyos mértékű rugalmasság. Ez a rugalmasság például egy ajánlott időponttól való eltérésben jelenik meg. Ilyen többperiódusú járműútvonaltervezési problémát (Multiperiod Vehicle Routing Problem - MPVRP) vizsgáltak Estrada-Moreno, A. és szerzőtársaik [33], ahol az ügyfelek által preferált kiszállítási napokon történő szállítás mellett lehetőség nyílik, hogy a kiszállításokat más napokra is tervezzék, cserébe az ügyfelek kedvezményt kapnak a szállítási díjból. Ez a megközelítés plusz lehetőségeket biztosít, melyek nagyban hozzájárulhatnak a szállítási folyamatok hatékonyabb optimalizációjához. A kutatásukban vizsgált módszertan egy kétszakaszos metaheurisztikus megoldás volt.

3. fejezet

Probléma definíció

A dolgozatomban vizsgált probléma informáliasan a 3.1 fejezetben kerül definiálásra. A megoldásom kidolgozása előtt a feladatot formálisan is definiáltam, melyet a 3.2 szövegrészben taglalok.

3.1. Informális probléma definíció

A dolgozatom a MPVRP egy speciális aspektusát tárgyalja, mely eddig nem került részletes vizsgálatra a szakirodalomban. A kutatás keretében különös figyelmet fordítottam az ügyfelekhez rendelt preferált napokra és az azoktól megengedett eltérés kezelésére.

Az általam vizsgált probléma a hosszútávú tervezésre fókuszál. A szakirodalomban a periodikusságnak is különböző definícióit találhatjuk meg, és emiatt apró eltérések léteznek a többperiodusú VRP meghatározásai között. Az általam vizsgált feladatosztály esetén a multi-periodikusságot a teljes tervezési időtáv kisebb periódusokra bontása jelenti, amik között korlátozások állhatnak fent.

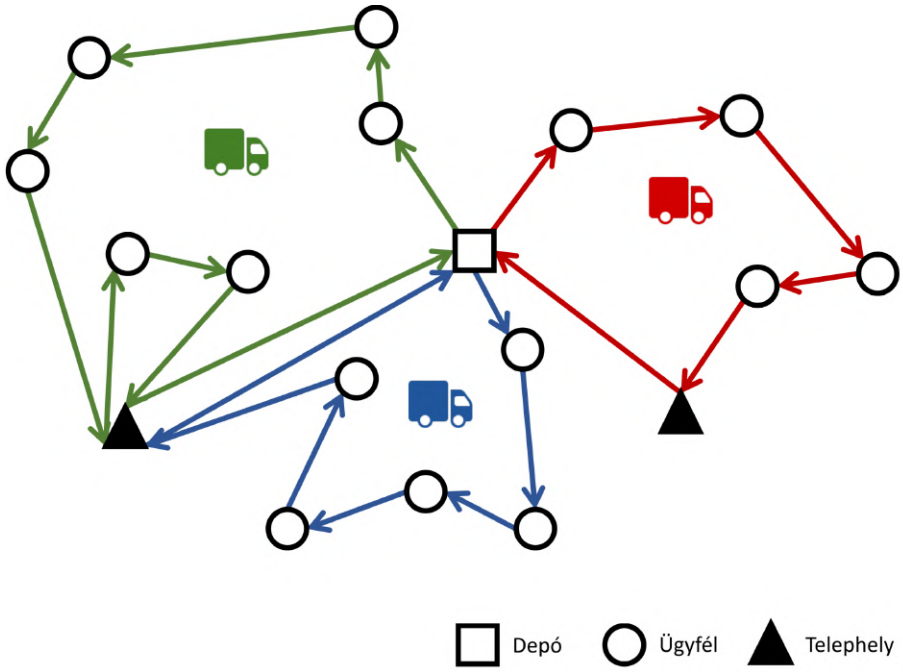
Egy periódusra vonatkozóan bizonyos korlátokat definiálunk az ügyfelekre nézve. Ilyen a preferált nap meghatározása, mely Russel és Igo [25] megoldásában is szerepel. A kutatás céljai közé tartozik, hogy megvizsgáljam, milyen eredmények érhetőek el a rendszerességtől való eltérés megengedésével, figyelembe véve, hogy ez plusz költséget eredményezhet. A preferált naptól történő eltérést vizsgálták Estrada-Moreno, A. és szerzőtársaik [33] is, akik figyelembe vették a rugalmasság lehetőségét az ügyfeleknek biztosított kedvezményekért cserébe. Az ilyesfajta eltérés engedélyezése olyan esetekbe lehet releváns, ahol nem elsődleges cél a konzisztens szolgáltatás. Ilyen lehet

többek között a hulladékszállítás, magába foglalva a háztartási és ipari hulladékok gyűjtését is. Ipari példák esetén gyakran előfordulhat, hogy nem fontos mely napon történik a begyűjtés, viszont fontos tényező lehet a két begyűjtés között eltelt idő. Ennek kezelése érdekében Gaudioso és Paletta [29] féle feladatosztályban definiált maximum eltérés korlátot használjuk a periódusok közötti begyűjtési napokra vonatkozóan. Az általam vizsgált feladatosztály esetén a perióduson belüli döntési tényezőt az ügyfél látogatási napjának meghatározása jelenti, periódusok között pedig korlátot vezettem be egy adott ügyfél két egymás utáni látogatása között eltelt maximális időre.

A hosszútávú periodikus tervezésen túlmenően a feladatosztály másik fontos aspektusa az egy napra levetített optimális útvonaltervezés. A munkám során olyan egynapos VRP feladatot vizsgáltam, mely elsősorban a hulladékgyűjtésre fókuszál. Ilyen specifikus tényező a hulladéklerakó telephelyek, melyek olyan létesítmények, ahol a hulladékot tárolják, kezelik és feldolgozzák. Az útvonaltervezés szempontjából ez olyan korlátokat von maga után, mint például, hogy amennyiben a jármű eléri a maximum begyűjtési kapacitását, akkor meg kell hogy látogasson egy ilyen telephelyet. Ezenfelül fontos szem előtt tartani, hogy a depó nem funkcionálhat lerakóhelyként, így oda a jármű nem térhet vissza a begyűjtött hulladékkal. Ezért minden járműnek a műszak befejezésekor utolsó állomásként egy telephelyet kell meglátogatnia. Ezen hulladékgyűjtési VRP feladatra a 3.1 ábrán látható egy példa, ahol a hulladéklerakó telephelyeket a háromszög alakzat reprezentálja, a különböző színek meg az egyes járművek körútjai.

A probléma definíciója a begyűjtési kapacitáson túlmenően magába foglalja, hogy a napi tervezést, egy műszak időtartamára kell megoldani. Ez a kapacitáskorlát nem csupán a körút hosszát jelenti, hanem egy ügyfél vagy telephely kiszolgálására fordított időt is.

Az általam vizsgált probléma leginkább az Estrada-Moreno, A. és szerzőtársaik [33] által publikált feladatosztályhoz hasonlít. Bár ők is hozzám hasonlóan kezelik a preferált napoktól való eltérés lehetőségét, a munkámban más számos, általuk nem vizsgált tényező is definiálásra került. Többek között nem vizsgálták a periódosuk közötti összefüggést, azaz a két látogatás közötti maximális eltérési idő korlátját, és az olyan megszorításokat, melyek a körutakra vonatkoznak (pl. kapacitáskorlát), illetve a hulladékgyűjtő telephelyek látogatásának a lehetőségét. Ezen specifikációk együttesét alkotó problémaosztályra nem találtam megoldást az irodalomban.



3.1. ábra. Példa VRP telephelyekkel

3.2. Formális probléma definíció

A problémaosztályt formálisan az alábbiakban definiáltam. A tervezési horizontot a T jelöli, melynek hossza napokban a $|T|$. Továbbá bevezetésre kerül a P periódus. A T időszakot $|T|/|P|$ periódusra osztjuk fel. A tervezési időszak egy konkrét i-adik periódusa a P_i, ahol a kezdő nap a $1 + (i-1)|P|$ és vége a $i|P|$.

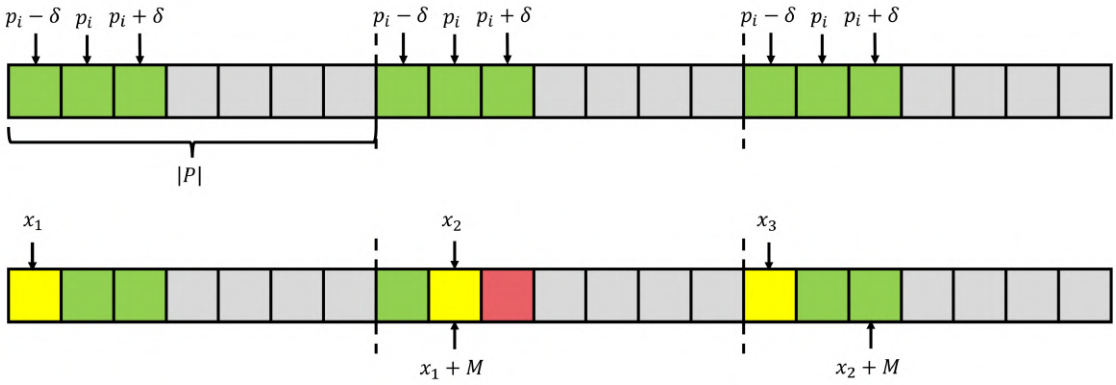
A napi tervezést egy $G = (V, A)$ gráfon határozzuk meg, ahol $V = \{v_0, v_1, ..., v_{n+k}\}$ a csúcsok halmaza. Ezen belül v_0 a depó, míg az $1..n$ az ügyfelek és $n+1..n+k$ a hulladéklerakó telephelyek. A gráfban $A \subseteq V \times V$ az élek halmaza, mely a lehetséges útvonalakat reprezentálja a csúcsok között. Ha $a = (v_i, v_j) \in A$, akkor v_i és v_j között létezik útvonal. Az útvonal távolságát v_i és v_j csúcs között $d(v_i, v_j)$ jelöli.

Minden i-edik ügyfélhez tartozik egy p_i preferált begyűjtési nap, ahol $1 \leq p_i \leq |P|$. A preferált naptól megengedett eltérést δ-val definiáljuk, ahol az eltérés mértékét napokban fejezzük ki. A korlátozást, hogy hány nap térhet el két begyűjtés között az M értéke határozza meg.

A rendelkezésre álló járműflottát a K halmaz reprezentálja. Egy körútra vonatkozóan a H paraméter határozza meg az út maximális időtartamát. Minden ügyfél és telephelycsúcshoz tartozik egy t szolgáltatási idő, melyek összege és az útvonal

hosszára kiszámolt út időtartama nem haladhatja meg a H értékét. Ezenfelül minden ügyfélhez tartotik egy w súly, mely a begyűjtött hulladék mennyiségét jelöli. A körút során külön korlátozást határoz meg a q maximális teherbírási érték, melyet az útvonal során nem léphetünk túl.

A 3.2 ábrán a probléma periodikus aspektusa látható. Az ábra felső részén az adott i ügyfél p_i preferált kiszolgálási napjai és az azoktól való megengedett δ eltérés zöld négyzetekkel van jelölve. Az alsó részen sárga négyzettel kiemelt x_j reprezentálja a j-edik periódusra vonatkozó döntést, hogy mikor történjen a kiszolgálás. Ezenfelül a begyűjtési napok közötti M maximum eltérés korlátját piros négyzet szemlélteti, melyet az előző periódusban lévő begyűjtési nap figyelembevételével határozunk meg.



3.2. ábra. Példa a preferált napok szerinti periodikus tervezésre

4. fejezet

Megoldó módszer

A fejezetben bemutatásra kerül egy heurisztikus megoldás az előzőekben definiált feladatosztályra. A megoldás általános keretrendszeréről a 4.1. alfejezetben esik szó. Arról, hogy egy adott megoldást, hogyan értékeltem ki a 4.2. alfejezetben írok, míg egy új megoldás elfogadásáról a 4.3. alfejezetben. A feladatosztályhoz specifikus megoldás inicializálásáról a 4.4. alfejezetben, míg a heurisztikához tartozó romboló és építő metódusokról a 4.5. és a 4.6. alfejezetekben taglalok részletesen. A fejezet befejezéseképpen a 4.7. alfejezetben a módszer adaptív mechanizmusa kerül bemutatásra.

4.1. Large Neighborhood Search heurisztika

A dolgozatomban felvetett feladatosztályra egy Large Neighborhood Search (LNS) heurisztika keretét vettem alapul. A módszertan egy olyan lokális keresésen alapuló heurisztika, mely egy kezdeti megoldásból indulva iteratívan jelentős méretű változtatásokat hajt végre a megoldás javítása érdekében. A legtöbb szomszédsági kereséshez képest az LNS jóval nagyobb méretű keresési térben mozog, aminek köszönhetően a nagyobb eséllyel találja meg a globális optimumot. A megközelítés általános működését a 4.1. pszeudokód mutatja be.

Az algoritmus első lépéseként egy kezdeti megoldást határozunk meg (1. sor), melyet a metódus további részében iteratívan szeretnénk megjavítani. A metódusba dolgozhatunk egy paraméterként előre meghatározott megoldással, vagy alkalmazhatunk különféle kezdeti megoldást generáló technikát is. A dolgozatomban két ilyen módszert mutatok be a 4.4. alfejezetben.

4.1. Algoritmus. LNS heurisztika

```
1: Kezdeti megoldás: s_best ← s
2: while kilépési feltétel nem teljesül do
3:    s' ← s
4:    q ügyfelek eltávolítása s'-ből
5:    Eltávolított ügyfelek visszaszúrása s'-be
6:    if f(s') < f(s_best) then
7:       s_best ← s'
8:    end if
9:    if elfogadás(s', s) then
10:      s ← s'
11:   end if
12: end while
13: return s_best
```

A módosítást végző iterációt ismételjük (2. sor) mindaddig, amíg a kilépési feltétel nem teljesül, mely lehet egy adott iteráció szám elérése vagy egy speciálisabban meghatározott feltétel. A ciklusmag első lépéseként (3. sor) lemásoljuk a kezdeti s megoldást, majd ezt követően meghívunk s'-re egy romboló és egy építő metódust (4. és 5. sorok). A rombolás során valamilyen módon eltávolítunk q ügyfelet a megoldásból, míg az építés során ezeket az ügyfeleket egy választott módszer alapján visszaszúrjuk a megoldásba. Mindkét művelet számos módon kivitelezhető, az általam alkalmazott megoldásokról a 4.5. és a 4.6. alfejezetekben írok részletesen.

A módosított s'-re és az eddigi legjobbnak bizonyult s_{best}-re kiszámítunk egy úgynevezett "fitness" értéket (6. sor), mely a megoldás jóságának mértékét mutatja. Ez az $f(s)$ függvény reprezentálja az adott feladatosztályhoz tartozó célfüggvényt, mely jelenesetben az útvonalhosszok és a preferált napoktól való eltérésért számított büntetés értékek összegének minimalizálását jelenti. Amennyiben s' jobb eredményt ért el, mint s_{best}, akkor s_{best}-et felülírjuk a jobb megoldással (7. sor).

Az iteráció végén meghívásra kerül egy elfogadási metódus (9. sor), mely eldönti, hogy a módosított megoldást elfogadjuk-e. Amennyiben elfogadásra kerül s', akkor s-et felülírjuk azzal (10. sor). Ezáltal a következő iterációban már frissített s megoldásból indulunk ki. Majd ciklus befejezését követően visszaadjuk az eddigi legjobb megoldást (13. sor), ezzel befejezve a heurisztikát.

A módszertan kidolgozása során a Ropke és Pisinger [12] féle megoldást vettem alapul, mely egy adaptív megoldása az LNS heurisztikának. A keresés során adaptívan történik a romboló és építő metódusok kiválasztása. Pontosabban az iterációk során a metódusok súlyozott valószínűsége változik, melyet a keresési térben elért

előző eredmények alapján számolunk ki. Ennek részleteiről a 4.7. alfejezetben írok részletesen.

4.2. Megoldás kiértékelése

Egy megoldás kiértékelésére szolgál az $f(s)$ függvény, melyet fitness függvénynek szokás hívni. Bemeneti paramétere egy s megoldás. Az én esetemben ez a metódus számolja ki az útvonalak hosszát és a preferált napoktól való eltérésért számított büntetés értéket. Ezt a számítást a következő 4.1. képlet reprezentálja.

$$f(s) = \sum_{d \in T} \sum_{l \in K} t_{d,l} + \sum_{p \in P} \sum_{v \in V} c_{p,v} \tag{4.1}$$

A képletben szereplő $t_{i,l} \in \mathbb{R}^+$ érték az d nap l jármű útvonalának hosszát jelenti. Az útvonalak hosszának kiszámításához a járművek útvonalait bejárva összeadjuk az egyes szakaszok hosszát. A megoldások tartalmazhatnak olyan ügyfél kiszolgálásokat, melyek egy úgynevezett virtuális járművel történnek. Ezek csak az LNS iteráción belül léteznek és akkor fogjuk csak használni őket, ha az előre megadott járműflottával nem tudunk megvalósítható megoldást találni. Ennek eseteit a későbbi alfejezetekben részletezem. Fontos viszont a fitness függvény számításánál kezelni, hogy ne kerüljön be legjobb megoldásként olyan eset, ahol ez a virtuális jármű használva van. Emiatt az $f(s)$ függvény számításánál, ha l jármű virtuális, akkor a $t_{d,l} = \infty$, így $f(s)$ értéke is ∞ értéket vesz fel, ezzel elkerülve, hogy a s bekerüljön a legjobb megoldások közé.

A $c_{p,v}$ értéke jelzi, hogy a v ügyfél a p periódusban történő kiszolgálása mekkora költséggel jár az ügyfélhez rendelt nap alapján. Ez a kiszolgálási nap és a preferált nap közötti különbséget foglalja magába, vagyis ha p' napon látogatjuk meg az ügyfelet, akkor $|p - p'|$ érték az eltérés, melyet egy büntető tényezővel szorzunk meg. Amennyiben a kiszolgálás a preferált napon történik, akkor $c_{p,v}$ nulla értéket vesz fel.

4.3. Megoldás elfogadása

A Ropke és Pisinger [12] által javasolt megoldást felhasználva a következőképpen definiáltam az elfogadás(s', s) függvényt, mely egy szimulált hűtéses elv alapján

működik. Első lépésnek meghatározunk egy elfogadási valószínűséget, melyet felhasználva döntjük el, hogy elfogadjuk-e az új megoldást. Minden iterációban választunk egy véletlenszerű számot 0 és 1 között, majd ezt összehasonlítjuk P_{accept} elfogadási valószínűséggel, melyet a következő 4.2. képlet alapján számíthatunk:

$$P_{\text{accept}} = e^{-\frac{f(s')-f(s)}{T}} \tag{4.2}$$

Az $f(s')$ jelöli az új megoldás fitness értékét, $f(s)$ az eddigi megoldását, T pedig a hőmérsékletet. A T értéke a hőmérséklet csökkenésével egyre kisebb lesz, így az elfogadási valószínűség is csökken. A kezdeti T érték számítása még az iterációk előtt, a kezdeti s megoldással a következő 4.3. egyenlet alapján történik:

$$T = -\frac{w \cdot f(s)}{\log(0.5)} \tag{4.3}$$

A képletben T jelöli a hőmérsékletet, w pedig egy konstans, melyet jelen megoldásban 0.05 értékre állítottam. Ez a kezdeti T hőmérséklet azt fogja garantálni, hogy a módszer az első iterációban pontosan 0.5 valószínűséggel fogad el egy olyan megoldás, ami w százalékkal rosszabb az LNS kezdeti megoldásánál, majd T értékének csökkenésével ("hűtésével") ez az elfogadási valószínűség is csökkenni fog. Minden iterációban ezt a T hőmérsékletet megszorzom egy c konstanssal, melyet 0.99975 értékre definiáltam, így az iterációk során a hőmérséklet folyamatosan csökken.

4.4. Kezdeti megoldás meghatározása

Lokális kereső algoritmusok esetén szükség van egy kezdeti megoldásra, melyből a keresés kiindulhat. Dolgozatomban olyan módszereket vizsgáltam kezdeti megoldás előállítására, melyek az ügyfeleket a preferált napjaikon látogatják meg. Ezen inicializálási folyamat során az algoritmus először minden ügyfelet a preferált napjára oszt, majd az ezáltal kialakult egynapos VRP feladatokat külön oldja meg. Az alábbiakban két módszert mutatok be ezeknek a VRP-knek a megoldására.

A 4.4.1. alfejezetben bemutatott vegyes-egész lineáris programozási modell egy egzakt módszer, azaz garantálja az optimum megtalálását. Gyakran azonban ezek az egy napra kialakult példák is rendkívül nagy méretűek, így a globális optimum megtalálása nagyon sok időt vehet igénybe. Emiatt ezen megoldásokat csak az eredmények kiértékeléséhez fogom használni.

A fentiek miatt kidolgoztam egy másik megoldást is, mely csak lokális optimumot garantál, azonban jóval gyorsabb futási időt biztosít a nagyobb feladatok esetén is. Ezt a mohó algoritmuson alapuló heurisztikát a 4.4.2. alfejezetben definiálom.

4.4.1. MILP modell az egy napos VRP megoldására

Az alábbi modell egy Buhrkal és szerzőtársai által publikált [21] megoldáson alapszik. Az a 3.2. alfejezetben megfogalmazott problémaosztály egy részfeladatát, az egy napra levetített hulladékgyűjtésre fókuszáló VRP-t oldja meg. A Buhrkal és szerzőtársai féle megoldás figyelembe veszi a hulladéklerakó telephelyeket, melyek olyan létesítmények, ahol a hulladékot tárolják.

Az alábbi megoldásban nem vesszük figyelembe a hivatkozott modellben található, ügyfelekhez tartozó időablakokat, mivel a kiszolgálás időpontját elsősorban periodikusan kezeljük. Viszont az előbb említett szakirodalmi megoldáson túlmenően bevezetésre került egy maximum idő korlát egy körút hosszára vonatkozóan, illetve külön korlátozással kezeljük, hogy a járművek a műszak végén üresen térjenek vissza a depóba. Így azt feltételezve, hogy a depó nem funkciónál lerakodóhelyként, azaz a jármű oda nem térhet vissza a begyűjtött hulladékkal.

A megközelítés az alábbiakban taglalt változókból, korlátozásokból és célfüggvényből áll.

Változók

$x_{i,j,l} \in \{0,1\}$ az i csúcsból a j csúcsba megy-e az l jármű, ezzel meghatározva a jármű útvonal sorrendjét. $i,j \in V', l \in K$, ahol $V' = \{v_0, v_{0'}\} \cup V^c \cup V^d$ a kiinduló és visszatérési depó, az adott napra kiszolgálandó ügyfelek és a hulladéklerakó telephelycsúcsok halmaza és K a járművek halmaza.

$d_{i,l} \in \mathbb{R}^{0,+}$ az i csúcs után a l járművön lévő összes összegyűjtött hulladék mennyiséget határozza meg. $i \in V', l \in K$.

$w_{i,l} \in \mathbb{R}^{0,+}$ minden i csúcsba történő érkezési időt tárolja minden l járműre vonatkozóan. $i \in V', l \in K$.

Korlátozások

A megoldásba minden jármű a depóból indul és oda is tér vissza. Ennek megfelelően az $x_{i,j,l}$ változók kezeléséhez a következő két korlátozás bevezetése szükséges. A 4.4 úgynevezett "big-M" korlátozás felel az indulásért, ahol $\mathbf{M}$ paraméter egy kellően nagy konstanst jelent a relaxáció biztosítására. Ezenfelül a 4.5 megszorítás biztosítja a körút végén lévő visszatérést.

$$\sum_{i \in V'} \sum_{j \in V' \setminus \{i\}} x_{i,j,l} \leq \mathbf{M} \cdot \sum_{j \in V' \setminus \{0\}} x_{0,j,l} \quad \forall l \in K \tag{4.4}$$

$$\sum_{i \in V^d} x_{i,0',l} = \sum_{j \in V^c} x_{0,j,l} \quad \forall l \in K \tag{4.5}$$

Annak érdekébe, hogy az előre meghatározott ügyfél halmaz minden elemét pontosan egyszer szolgáljuk ki, a 4.6 megszorítást alkalmazzuk.

$$\sum_{i \in V' \setminus \{j\}} \sum_{l \in K} x_{i,j,l} = 1 \quad \forall j \in V^c \tag{4.6}$$

A jármű útvonalak sorrendjének kezeléséhez a 4.7 egyenletet, egy úgynevezett "flow conservation" korlátot alkalmazunk, mely biztosítja, hogy a depó kivételével minden csúcsból pontosan annyi jármű távozzon, mint amennyi odaérkezik.

$$\sum_{i \in V' \setminus \{j\}} x_{i,j,l} = \sum_{k \in V' \setminus \{i\}} x_{j,i,l} \quad \forall j \in V' \setminus \{0\}, l \in K \tag{4.7}$$

A járművek kapacitásának kezeléséhez a következő négy korlátozást vezettem be. A 4.8 korlátozás biztosítja, hogy a depóból induló járművek üresen induljanak. A hulladéklerakó telephelyeken történik a járművek kiürítése, mely többször is megtörténhet egy körút során. Egy ügyfél kiszolgálása során összegyűjtött hulladék mennyiség hozzáadását a jármű által felhalmozott értékhez a 4.9 korlátozást alkalmaztam. Ezenfelül az egy járműre vonatkozó egy időpontban maximálisan összegyűjthető hulladék korlátozását a 4.10 megszorítás garantálja, ahol a q paraméter jelöli a jármű maximális kapacitását.

$$d_{i,l} = 0 \quad \forall l \in K, i \in \{0, 0'\} \tag{4.8}$$

$$d_{i,l} + q_i \leq d_{j,l} + \mathbf{M} \cdot (1 - x_{i,j,l}) \quad \forall i \in V' \setminus V^d, j \in V', l \in K \tag{4.9}$$

$$d_{i,l} \leq q \quad \forall i \in V', l \in K \tag{4.10}$$

Az útvonalak tervezése során számolni kell mind az út időtartamával, mind a megállókon eltöltött idővel. Ennek összegét tárolja a $w_{i,l}$ változó, melynek frissítését a 4.11 formula biztosítja. Ezenfelül a 4.12 korlátozással biztosítottam, hogy a járművek az előre meghatározott H műszakhosszon belül teljesítsék a körutakat.

$$w_{i,l} + s_i + t_{i,j} \leq w_{j,l} + \mathbf{M} \cdot (1 - x_{i,j,l}) \quad \forall i \in V', j \in V', l \in K \tag{4.11}$$

$$w_{i,l} \leq H \quad \forall i \in V', l \in K \tag{4.12}$$

Célfüggvény

Az optimum megtalálásához a 4.13 célfüggvényt definiáltam, mely az útvonalak összesített időtartamát minimalizálja.

$$\min \sum_{l \in K} \sum_{i \in V'} \sum_{j \in V' \setminus \{i\}} t_{i,j} \cdot x_{i,j,l} \tag{4.13}$$

4.4.2. Mohó algoritmus az egy napos VRP megoldására

A lentebb bemutatott mohó algoritmusnak a fő célja, hogy képes legyen inicializálni egy kezdeti megoldást, mint ahogy a 4.4.1. alfejezetben bemutatott MILP modell is. Ellentétben azonban az előbbivel, ennek a megközelítésnek nem az a célja, hogy megtalálja a globális optimumot, hanem hogy egy hatékony módon meghatározzon egy megoldást, melyet később az LNS javítani tud.

Az általam javasolt módszer a "best fit" logikán alapul, mely során minden ügyfelet a legjobban illeszkedő jármű útvonalhoz rendelünk. Ennek kivitelezését a specifikált problémára vonatkozóan a 4.2. pszeudokód mutatja be.

A metódus egy rendezéssel kezdődik (1. sor), mely során a V^c halmaz elemeit a depótól mért távolság szerint növekvő sorrendbe rendezzük. Ezzel biztosítva, hogy az utána lévő ciklusban (2. sor) először a legközelebbi ügyfelekre kerüljön sor. A ciklus célja, hogy meghatározzuk, hogy melyik járműhöz rendeljük az adott v ügyfelet. A legjobb járművet (3. sor) és a hozzá tartozó beszúrási költséget (4. sor) jelöli, mely kezdetben végtelen értéket vesz fel.

Ezt követően végignézzük az összes K járművet (5. sor). Ebben a ciklusban megvizsgáljuk, hogy az adott v ügyfél hozzáadható-e az l jármű útvonalához (6. sor). Ennek teljesüléséhez a jármű útvonalának ideje nem haladhatja meg a H maximális műszakhosszt, figyelmebe véve, hogy a járműnek még vissza kell térnie a depóba, illetve ha az útvonal utolsó állomása nem telephely, akkor még a járműnek meg

4.2. Algoritmus. Mohó algoritmus az egy napos VRP megoldására

Tömb: $S \leftarrow$ Rendezd V^c elemeit a depótól mért távolság szerint növekvő sorrendben
for minden elem $v \in S$ tömbre **do**
  $l_{best} \leftarrow$ null {legjobb jármű}
  $c_{best} \leftarrow \infty$ {legkisebb beszúrási költség}
  for minden $l \in K$ halmazra **do**
    if v hozzáadható l jármű útvonalához **then**
      $c \leftarrow$ Költség(l, v)
      if $c < c_{best}$ **then**
        $l_{best} \leftarrow l$
        $c_{best} \leftarrow c$
      end if
    end if
  end for
  if $l_{best} \neq$ null **then**
    v hozzáadása l_{best} útvonalához
    if l_{best} kapacitása a küszöbérték alá csökken **then**
      $d \leftarrow$ Legközelebbi-telephely(l_{best})
      d telephely hozzáadása az l_{best} útvonalához
    end if
  else
    return null {Nem sikerült minden ügyfelet kiszolgálni}
  end if
end for
for minden $l \in K$ halmazra **do**
  if l utólsó állomása nem telephely és megy ügyfélhez **then**
    $d \leftarrow$ Legközelebbi-telephely(l_{best})
    d telephely hozzáadása az l_{best} útvonalához
  end if
end for
return K a járművek útvonalai

kell látogatnia a legközelebbi hulladéklerakót is. Valamint a hozzáadási lehetőségnek garantálásához figyelembe kell venni, hogy a jármű által begyűjtött hulladék mennyisége nem haladhatja meg a q kapacitást. Amennyibe hozzáadható az ügyfél az l jármű útvonalához, kiszámítjuk a c beszúrási költséget (7. sor), mely jelen feladatosztály esetén a v és az előző állomás közötti távolságnak, v állomás kiszolgálási idejének, legközelebbi hulladéklerakóhoz tartozó távolságának és annak kiszolgálási idejének, valamint a depóba való visszatéréshez szükséges távolságának összegét jelenti. Ha ez az újonnan kiszámolt c kisebb, mint az eddigi legjobb beszúrási költség (8. sor), akkor frissítjük a l_{best} legjobb jármű és a c_{best} legjobb beszúrási költség értékét (9. és 10. sorok).

Miután végig iteráltunk minden l járművön, megnézzük, hogy sikerült-e találni olyan körutat, melyhez hozzáadható a v ügyfél (14. sor). Ha nem, akkor a metódus **null** értékkel tér vissza (21. sor), jelezve, hogy nem sikerült minden ügyfelet kiszolgálni. Ellenkező esetben hozzáadjuk az l_{best} jármű útvonalához a v ügyfelet (15. sor). Annak érdekében, hogy biztosíthassuk a jármű kapacitásának megfelelő kihasználását, ellenőrizzük, hogy az l_{best} jármű kapacitása egy előre meghatározott küszöbérték alá csökken-e (14. sor). Ha ez teljesül, akkor a jármű útvonalához hozzáadjuk a legközelebbi hulladéklerakó telephelyet (17. és 18. sorok), ezzel biztosítva, hogy a további ügyfelek kiszolgálását ne akadályozza a jármű kapacitása.

A metódusban az előbbieken felül biztosítani kell, hogy minden körút a depó előtt egy hulladéklerakó telephelyen fejeződjön be. Ennek érdekében egy ciklussal újra végig nézzük az összes l járművet (24. sor). Amennyiben l utolsó állomása ügyfél (25. sor), akkor a jármű útvonalához hozzáadjuk a legközelebbi hulladéklerakó telephelyet (26. és 27. sorok). Így teljesítve minden korábbi feltételt az algoritmus visszatér minden jármű útvonalával (30. sor), ezzel meghatározva a kezdeti megoldást.

4.5. Romboló módszerek

Az LNS heurisztika egyik fő eleme a rombolási lépés, melynek célja, hogy a megoldásból valamilyen módon eltávolítsunk elemeket. Ezt megtehetjük véletlenszerű módokon is, valamint hatékonyabb megközelítéseket alkalmazva is. Fontos továbbá, hogy figyelembe vegyük a feladatosztály sajátosságait, biztosítva, hogy a teljes ke-

resési teret bejárjuk. Rendszerint még jobb megoldásokat érhetünk el, ha különféle módszerek együttesét alkalmazzuk a keresés során.

A fejezet további részében több ilyen eljárást különböztetek meg, melyek különféle módokon képesek az előzőekben definiált VRP feladat esetén a G gráf V csúcsainak egy $R \subseteq V$ részhalmazát eltávolítani. A módszerek meghatározása során igyekeztem, minél nagyobb diverzitást biztosítani a keresési térben, hogy a keresés ne ragadjon lokális optimumokba. Ezek a metódusok különböző szinteken végzik a módosításokat. Vannak módszerek (4.5.1. alfejezet és 4.5.3. alfejezet), melyek kisebb mértékben, egy-egy nap tervezése szempontjából rombolják a megoldásokat. Elkülöníthető szint a teljes periódus szempontjából történő törlés (4.5.2. alfejezet), valamint a nagyobb léptékű rombolás, ahol egy teljes nap kiszolgálásait töröljük (4.5.4. alfejezet).

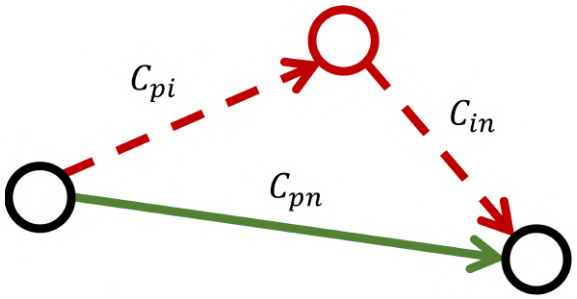
4.5.1. Legrosszabb kiszolgálás törlése

A módszer az úgynevezett "worst removal" logikán alapul, mely során minden iterációban a legrosszabb teljesítményt nyújtó elemeket távolítjuk el. Ezt a teljesítményt egy törlési költség reprezentálja, melyet a következőképpen számítható:

$$\Delta C = (C_{pi} + C_{in}) - C_{pn} + c_i * \delta \tag{4.14}$$

A 4.14 képletben $C_{pi} \in \mathbb{R}^+$ jelenti p-ből i-be való utazásnak költségét, $C_{in} \in \mathbb{R}^+$ az i-ből n-be való utazás költségét, $C_{pn} \in \mathbb{R}^+$ a p-ből n-be való utazás költségét, $c_i \in \{0, 1\}$ pedig az i elem preferált naptól való eltérést reprezentálja, melyet egy büntető tényezővel, δ szorzunk. A számítással meghatározható, hogy az i elem törlése hogyan befolyásolja a megoldás költségének csökkenését. Erre a törlésre az útvonaltervezés szempontjából mutat egy példát a 4.1. ábra, ahol a szaggatott piros vonalak jelölik a törlés előtti utakat és az ahhoz tartozó költség értékeket és a zöld vonal reprezentálja az i elem törlése után keletkező útvonalat és annak költségét.

Egy további fontos kiegészítés lehet a randomizálás alkalmazása, mely növeli az algoritmus diverzitását és segíthet elkerülni, hogy az algoritmus korán lokális optimumokba ragadjon. Ilyen véletlenszerűséggel bővített módszert alkalmazott többek között Ropke és Pisinger [12]. Ezt kutatásom során definiált problémára igazított módszert mutatja be az alábbi a 4.3 pszeudokód, ahol fontos kiterjesztés a periodi-



4.1. ábra. Példa egy csomópont törlésére

kusság kezelése. Ehhez szükséges a megoldásban figyelembe venni, hogy adott ügyfél mely periódusban történő kiszolgálását szeretnénk törölni.

4.3. Algoritmus. Worst removal algoritmus kiszolgálás törlésre

1: Tömb: $L \leftarrow$ Az összes törölhető $(d, l, i) \in T \times K \times V^c$ kombináció, rendezve csökkenő Törlési-költség(d, l, i, s) szerint
2: **while** $n > 0$ és $L \neq \emptyset$ **do**
3: Válassz egy véletlen számot y a $[0, 1]$ intervallumban
4: Kiválasztott kombináció: $(d_r, l_r, r) \leftarrow L[y^p \cdot |L|]$
5: (d_r, l_r, r) kombináció eltávolítása az s megoldásból
6: $L \leftarrow L \setminus \{(d_r, l_r, r)\}$
7: $n = n - 1$
8: **end while**
9: **return** s módosított megoldás

A függvény bemeneteként szolgál s, mint megoldás, $n \in \mathbb{N}$ a törlendő csúcsok száma, illetve $p \in \mathbb{R}^+$ randomizációs kitevő, melyet a törlendő elemek kiválasztásánál használunk fel. A metódus először a $T \times K \times V^c$ nap-járműű-ügyfél kombinációk halmazának elemeit csökkenő sorrendbe rendezi a törlési költségük alapján (1. sor), ahol a költség az előzőleg definiált (4.14. képlet) módon kerül kiszámításra.

Ezt követően addig iterálunk egy ciklussal, amíg nem sikerült n darab elemet eltávolítani a megoldásból és az L tömb nem üres (2. sor). Minden iteráció során egy y véletlen számot választunk a $[0, 1]$ intervallumból (3. sor). Ezután a L tömbből kiválasztjuk a $\lfloor y^p \cdot |L| \rfloor$ indexű (d_r, l_r, r) elemet, ahol r az ügyfél azonosítója (4. sor), majd eltávolítjuk azt az s megoldásból (5. sor). Ezután az r elemet eltávolítjuk az L tömbből (6. sor), majd csökkentjük egyel a törlendő elemek számát (7. sor). Legvégül ciklus után a metódus visszatér a módosított megoldással (9. sor).

4.5.2. Legrosszabb ügyfél törlése minden periódusból

Annak érdekében, hogy hatékonyabban járhassa be az algoritmus a keresési teret, érdemes másfajta elvek alapján is törölni az elemeket. Míg a 4.5.1. fejezetben taglalt mód külön kezelte a nap-jármű-ügyfél kombinációkat, addig a következőekben bemutatott eljárás (4.4. algoritmus) adott ügyfeleket választ és azokat minden periódusból eltávolít, ezzel növelve a diverzitást. A választás során ugyanúgy a "worst removal" elvet követjük, mint ahogy azt az előző módszertannál is tettük (4.5.1. fejezet).

4.4. Algoritmus. Worst removal algoritmus ügyfél törlése minden periódusból

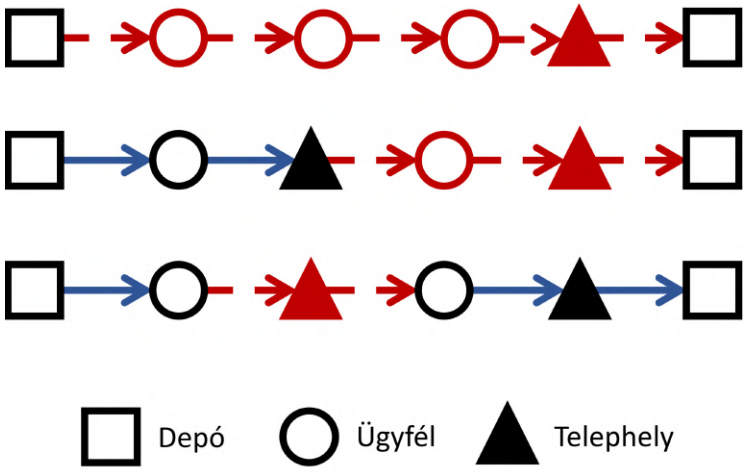
1: Tömb: $L \leftarrow$ Minden $i \in V^c$, az ügyfelek halmaza, rendezve Törlési-költség(i, s) szerint
2: **while** $n > 0$ és $L \neq \emptyset$ **do**
3: Válassz egy véletlen számot y a $[0, 1]$ intervallumban
4: Kiválasztott ügyfél: $r \leftarrow L[y^p \cdot |L|]$
5: **for** $p \in P$ **do**
6: Töröld az r ügyfelet a d_r hozzárendelt napról és l_r járműről az s megoldásban
7: **end for**
8: $L \leftarrow L \setminus \{r\}$
9: $n = n - 1$
10: **end while**
11: **return** s módosított megoldás

Az algoritmus futtatásához adott kell, hogy legyen egy s megoldás és $n \in \mathbb{N}$ a törlendő ügyfelek száma és $p \in \mathbb{R}^+$, mint randomizációs kitevő. Inicializálásként a metódus a V^c halmaz minden elemét hozzárendeli az L tömbhöz, rendezve a törlési költségük alapján. Ez a költség az adott ügyfélre vonatkozó minden periódusban lévő kiszolgálások törlési költségének az összegét jelenti (1. sor).

Ezt követően addig fut egy ciklus, míg nem sikerül n darab elemet eltávolítani és az L tömb nem üres (2. sor). Az iterációk során az előzőekhez hasonlóan választunk egy y véletlen számot a $[0, 1]$ intervallumból (3. sor), majd az L tömbből kiválasztjuk a $\lfloor y^p \cdot |L| \rfloor$ indexű r elemet, melyet a törlendő ügyfél azonosítására használunk (4. sor). Ezt követően végig iterálunk az összes $p \in P$ perióduson (5. sor). Minden periódusban eltávolítjuk az r ügyfelet az s megoldásból, megkeresve a hozzárendelt d_r napot és l_r járművet (6. sor). Ezután az r elemet eltávolítjuk az L tömbből (8. sor) és csökkentjük a törlendő elemek számát (9. sor). Végezetül a metódus visszatér a módosított s megoldással (11. sor).

4.5.3. Legrosszabb hulladéklerakó telephely törlése

Az eddigiekben bemutatott romboló heurisztikák az ügyfél csomópontok törlésére fókuszáltak. A vizsgált problémaosztályban azonban ugyanilyen fontos állomások a hulladéklerakó telephelyek is, így érdemes azok törlési lehetőségeit is megvizsgálni. Egy útvonalnál összefüggésben vannak a hulladéklerakó telephely állomások az előtte és utána lévő ügyfél csomópontokkal, hiszen a járműveknek van egy maximális kapacitásuk, melyet ilyen megállásokkal tudnak kezelni. Ezenfelül az utolsó állomásnak is egy hulladéklerakó telephelynek kell lennie. Ezen tényezők miatt a törlésnél különféle eseteket kell megkülönböztetni. Ilyen esetek példái láthatók a 4.2. ábrán, ahol piros színnel jelöltem a csomópontok törlését.



4.2. ábra. Példák a hulladéklerakó telephely törlésre

A legfelső útvonalon látható, hogy egyetlen hulladéklerakó telephelyen áll meg a jármű az utolsó állomásként. Ebben az esetben ha töröljük ezt a telephely állomást, törölni kell az összes többi ügyfél állomást is, hiszen a probléma definíció szerint kell a depóba történő visszatérés előtt üríteni kell a járművet egy telephelyen. Az alatta lévő példa esetén több hulladéklerakás is történik, így az utolsó törlés esetében csak az előtte lévő telephely állomásig kell törölni az ügyfél állomáspontokat is. Amennyiben nem az utolsó kiszolgálást töröljük, akkor az egyetlen befolyásoló tényező, ami magával vonhatja az ügyfél látogatások törlésit is, az a jármű maximum kapacitása. Ilyen esetet szemléltet az ábra utolsó útvonala, ahol nem történik ügyfél csúcs törlés. Ez nagyban függ az adott feladat paramétereitől, de előfordulhat, hogy nem kell törölni egyetlen ügyfél állomást sem csak a telephelyet, mint ahogy az a 4.2. ábrán is látható.

A lerakó telephely választása is történhet különböző módokon, lehetne teljesen véletlenszerűen is akár, de én most az előzőekben is többször használt "worst removal" elvet alkalmazom. Ennek a kerete adaptálható az előzőekben bemutatott metódusokból, csak a telephely csúcsokra alkalmazva. A választást követő lépéseket a 4.5. pszeudokód mutatja be.

4.5. Algoritmus. Worst removal algoritmus hulladéklerakó telephely törlésre

1: **if** Utólsó-hulladéklerakó(d, s) **then**
2: $d_{prev} \leftarrow$ Az előző hulladéklerakó vagy a v_0 depó az adott útvonalon
3: Töröld a d-t és az összes ügyfelet az adott útvonalról d_{prev}-ig az s megoldásban
4: **else**
5: $d_{prev} \leftarrow$ Az előző hulladéklerakó vagy a v_0 depó az adott útvonalon
6: $d_{next} \leftarrow$ A következő hulladéklerakó az adott útvonalon
7: Töröld d-t az s megoldásból
8: Tömb: $L \leftarrow$ Az d_{prev} és d_{next} közötti i ügyfél csúcsok, Törlési-költség(i, s) szerint rendezve
9: **while** Összes-hulladék-mennyiség(L) $\leq q$ és $L \neq \emptyset$ **do**
10: Válassz egy véletlen számot y a $[0, 1]$ intervallumban
11: Kiválasztott ügyfél: $r \leftarrow L[y^p \cdot |L|]$
12: Töröld az r ügyfelet az s megoldásból
13: $L \leftarrow L \setminus \{r\}$
14: **end while**
15: **end if**
16: **return** s módosított megoldás

Az algoritmus paraméterként vár egy s megoldást, melyen a módosításokat végzi, egy $p \in \mathbb{R}^+$ randomizációs kitevőt, valamint egy d hulladéklerakó telephely állomást, melyet törölni szeretnénk. Ez magába foglalja, hogy tartozik ehhez a csúcshoz egy d nap és l jármű is, melyet a megoldás módosításakor figyelembe kell venni.

Az algoritmus első lépésként eldönti, hogy az adott telephely az utolsó az adott jármű útvonalán (1. sor). Amennyiben igen, akkor két eset lehetséges, vagy van előtte egy másik hulladéklerakó telephely, vagy a depóba az egyedüli nem ügyfél csúcs még. Az első esetben meg kell keresni az előző hulladéklerakó telephelyet és addig töröljük az összes csúcsot. Amikor nincs más telephely az útvonalon, abban az esetben a depóig törölni kell az összes csúcsot. Ezt a műveletet végzi el az algoritmus 5. sorban.

Ha a választott d telephely nem az utolsó az útvonalon, akkor meg kell keresni a d_{prev} előző és a d_{next} következő telephelyet is (5. és 6. sorok), ahol a d_{prev} ugyanúgy lehet a depó is, ha az adott útvonalon nincs előtte más hulladéklerakó telephely állomás. Mielőtt az ügyfeleket is hozzáigazítanánk a törléshez, törölni kell a d te-

lephelyet az útvonalból (7. sor). Ezután az d_{prev} és d_{next} közötti ügyfél csúcsokat az adott útvonalon az L tömbbe gyűjtjük, rendezve a törlési költségük alapján (8. sor). Az L tömb elemei reprezentálják azokat az ügyfeleket, melyeket az adott l jármű meglátogat az d_{prev} és d_{next} között. Legtöbb esetben ez az útszakasz alatt felhalmozott hulladék mennyiség meg fogja haladni a jármű kapacitását, így törölni kell annyi ügyfelet, hogy ez a korlát ne sérüljön. Ezt a műveletet a következő ciklus végzi. A ciklus feltétele, hogy az összes ügyfél által felhalmozott hulladék mennyiség ne haladja meg a jármű kapacitását és az L tömb ne legyen üres (9. sor). Minden iterációban egy véletlen számot választunk a $[0, 1]$ intervallumból (10. sor), majd az L tömbből kiválasztjuk a $\lfloor y^p \cdot |L| \rfloor$ indexű r törlendő ügyfél csúcsot (11. sor), melyet az s megoldásból törölünk is (12. sor). Majd eltávoltjuk az L tömbből egyaránt (13. sor). Végül a ciklus után az algoritmus visszatér a módosított s megoldással (16. sor).

4.5.4. Egy teljes nap tervezésének a törlése

Mivel az előző két módszertan tipikusan kisebb méretű változásokat hajt végre és vagy az ügyfelek szempontjából optimalizál vagy egy-egy nap tervezés finomhangolására fókuszál, ezért érdemes lehet ezek mellett nagyobb méretű változást biztosító metódusokat is alkalmazni. Ennek egy lehetséges módja, ha egy teljes napot újra tervezünk, azaz az összes ügyfelet eltávolítjuk az adott napról. A módszer során a leghangsúlyosabb kérdés, hogy melyik napot válasszuk. A diverzitás növelése érdekében több különböző metódust is alkalmaztam, melyeket a következőkben részletezek:

Véletlenszerűen választott nap Egyik legegyszerűbb módja a nap kiválasztásának, ha egyenlő eséllyel választunk egy napot a D halmazból. Ennek használata is fontos, mivel így biztosított, hogy bármelyik nap kiválasztható, így a keresési tér teljes bejárása garantált.

Legköltségesebb nap A költséget elsősorban a törlési költség befolyásolja, mely jelen esetben a nap összes járműjével megtett távolság összege jelenti. Az a nap kerül kiválasztásra, ahol ez az érték a legmagasabb. Amennyiben több nap is ugyanolyan költséggel rendelkezik, akkor kevesebb ügyfelet tartalmazó napot választjuk.

Legforgalmasabb nap A forgalom mérése során két szempont alapján értékeltem a napokat. Az egyik, hogy hány ügyfelet szolgálunk ki az adott nap, a másik pedig, hogy az ügyfelek mekkora törlési költséget generálnak. A módszertannál a két érték összege adja meg, hogy melyik napot válasszuk. És a legnagyobb érték jelenti a legforgalmasabb napot.

4.6. Beszúró módszerek

Az LNS keretalgoritmus következő lépése a törölt ügyfelek visszaillesztése a megoldásba. Miután különféle módon eltávolítottunk kiszolgálásokat a megoldásból, vagyis bizonyos értelemben romboltuk azt, most újra kell építenünk a tervezést. Mindezt úgy kell megtennünk, hogy ne sértsük meg a problémaosztály korlátait, és a megoldás minőségét is javítsuk.

Nem minden esetben lehetséges a javítás a célfüggvény szempontjából, de hosszútávon hatékonyabb eljárást érünk el, ha minél szélesebb keresési térben próbálkozunk. Emiatt előfordulhat, hogy átmenetileg csökken a célfüggvény értéke, de ezáltal olyan megoldásokat is találhatunk, melyek a későbbi iterációkban jobb eredményeket produkálhatnak. Ennek elérése érdekében 2 különböző elven működő beszúró módszert javasoltam a dolgozatomban. Egyik egy mohó algoritmus (lásd 4.6.1), míg a másik egy úgynevezett regret elven működő algoritmus (lásd 4.6.2).

4.6.1. Mohó beszúró algoritmus

A visszaszúrás egyik legegyszerűbb módja, ha valamilyen mohó elvet követünk. Ez azt jelenti, hogy a beszúrásnál valamilyen módon értékelni kell a visszaszúrási lehetőségeket, majd ezek közül a legjobbat választani. Jelen esetben ez a költség az új ügyfél beszúrásával járó plusz távolság és idő. Ez a számítás a következő a 4.15. egyenlettel végezhető el:

$$\Delta C = C_{i-1,i} + C_{i,i+1} + t_i + c_i * \delta \tag{4.15}$$

A C jelöli az összköltséget, mely a távolságok és az idők összege. Az i index a beszúrás helyét jelöli az adott útvonalon. A plusz távolság a $C_{i-1,i}$ és $C_{i,i+1}$ értékek összege, melyek a beszúrás helyétől az előző és a következő ügyfélhez tartozó távolságokat jelentik. A t_i érték az ügyfél kiszolgálására fordított időt reprezentálja, míg

d_i az ügyfél preferált napjától való eltérést, melyet a δ büntetés költséggel szorozunk meg.

A beszúrás esetén figyelembe kell venni, hogy a törlés során előfordulhat olyan eset is ahol az ügyfél csak egyetlen periódusból lett törölve, míg más esetben többből. Mivel a periodikusság szempontjából is vannak korlátaink, így fontos, hogy a beszúrás során ezt is figyelembe vegyük. Hiszen ha több periódusból hiányzik az ügyfél, akkor úgy kell visszaszúrni, hogy kezeljük a periódusok közötti összefüggéseket is. Tovább bonyolítja a megoldást, hogy az új beszúrásokkal bizonyos esetekben újabb hulladéklerakó telephely látogatásokat is be kell iktatni.

Azért, hogy ezeket a megszorításokat megfelelően kezeljük, először minden ügyfélhez kiszámoljuk az összes lehetséges beszúrási lehetőséget. Illetve a beszúrások után frissítjük ezen kombinációk halmazát. Egy ilyen kombináció magába foglalja, hogy az adott ügyfelet, mely periódusokba szúrjuk vissza, és az adott perióduson belül mely napra, útvonalra, pozícióba, valamint ha szükséges, akkor új telephely látogatásokat is hozzárendelhetünk. Egy új telephely kiválasztása során fontos, hogy a pozíció, ahova az adott útvonalon szúrjuk, ott az előtte és utána lévő csúcsokhoz képest a lehető legoptimálisabb a telephely elhelyezkedése, melynek megállapításához felhasználható a 4.15. egyenlet.

Bizonyos esetekben előfordulhat, hogy egy adott napra nézve egy ügyfelet nem lehet a rendelkezésre álló járműflotta egyik útvonalába sem visszaszúrni, mert meghaladnánk a maximális H napi munkaidőt. Ekkor létrehozhatunk egy új virtuális járművet, ahol nincsen korlátozás a napi munkaidőre. Ennek előnye, hogy nem vetjük el az adott megoldást, hiszen lehet ebből folytatva a későbbiekben virtuális jármű nélkül találunk jobb megoldást. Annak érdekében, hogy ne ilyen eset legyen a legjobb megoldás az $f(s)$ számítása során (4.2) az útvonalhosszhoz ∞ értéket rendelünk.

A teljes algoritmus folyamatát a 4.6. pszeudokód mutatja be:

A függvény bemenete egy s megoldás, illetve a törölt ügyfelek halmaza. Mivel azok az ügyfelek, akik a legtöbb periódusból lettek törölve, nagyobb befolyásoltsággal bírnak a megoldásra, ezért először őket próbáljuk visszaszúrni. Így algoritmus első lépéseként (1. sor) a törölt ügyfeleket ez alapján csökkenő sorrendbe rendezzük az R tömbben. Ezt követően a Összes-beszúrás-kombináció(R, s) metódus segítségel minden lehetséges beszúrási kombinációt kiszámolunk (2. sor), melyeket egy C tömbben tárolunk. Itt fontos figyelembe venni, hogy ha egy periódusba visszaszúrunk egy ügyfelet, akkor az befolyással van arra, hogy a többi periódusban mely

4.6. Algoritmus. Mohó beszúró algoritmus

```
1: Tömb: R ← Törölt ügyfelek, aszerint csökkenő sorrendbe rendezve, hogy hány
   periódusból lettek törölve
2: Tömb: C ← Összes-beszúrás-kombináció(R, s)
3: L ← ∅ Beszúrt ügyfelek
4: for c ∈ C do
5:    if r_c ∈ L then
6:       continue
7:    end if
8:    s ← Kombináció-beszúrása(c, s)
9:    L ← L ∪ {r_c}
10:   V ← V ∪ {l_c} Érintett járművek
11:   C ← Összes-beszúrás-kombináció-frissítés(V, C, s)
12: end for
13: return s módosított megoldás
```

napokon szolgálható ki ugyanez az ügyfél. A kombinációk továbbá tartalmazzák a konkrét beszúrási helyeket a járművekhez, ahol legkisebb a beszúrás költsége az adott napon, valamint ha szükséges új hulladéklerakó telephely látogatást is a hozzá tartozó információkkal. Inicializálásként definiálásra kerül egy L halmaz, melybe a visszaszúrt ügyfelek kerülnek (3. sor).

Az algoritmus fő ciklusa (4. sor) végigiterál az összes beszúrási kombináción. Első lépésben itt ellenőrzésre, kerül, hogy az adott r_c-vel jelölt ügyfél már szerepel-e az L halmazban (5. sor). Amennyiben igen, akkor a következő iterációra lépünk (6. sor), hiszen az adott ügyfelet már visszaszúrtuk. Ellenkező esetben beszúrásra kerül a c kombináció az s megoldásba (8. sor), majd az L halmazba (9. sor). Az érintett járműveket is frissítjük (10. sor), majd frissítjük a beszúrás kombinációkat (11. sor), figyelmbe véve, hogy mely járművek körútjaiban történt változás. Az ciklus végén visszaadjuk a módosított megoldást (13. sor), ezzel befejezve a beszúró algoritmust.

4.6.2. Regret beszúró algoritmus

A regret alapú módszerek esetén egy olyan elvet követünk, melyben a döntési folyamat során figyelembe veszünk olyan költségeket, melyek akkor merülnek fel, ha nem a leghatékonyabb megoldást választjuk minden lépésben. Ezzel olyan döntéseket hozhatunk, mellyel hosszú távon jobb eredményeket érhetünk el, elkerülve a lokális optimumokba való beragadást. Az általam vizsgált probléma esetén az előző mohó algoritmust (4.6.1) módosítottam úgy, hogy minden lehetséges beszúrási kombinációra kiszámolok egy regret értéket. Ennek módját Ropke és Pisinger [12]

megoldása alapján határoztam meg. A regret érték a következő a 4.16. képlet alapján számoltam:

$$\mathrm{R}_i = \max_{j \in U}(\Delta f_i^{2j} - \Delta f_i^{1j}) \tag{4.16}$$

A Δf_i^{1j} jelöli az i ügyfélre vonatkozó költségnövekedést, ha azt a j-edik útvonalba a legkisebb költséget eredményező pozícióba helyezzük be. A Δf_i^{2j} azt a költségnövekedést fejezi ki, mely akkor keletkezik, ha ugyanazt az i ügyfelet a j-edik útvonalon a második legkevésbé költséges pozícióba illesztjük be. A R_i érték, azaz az i-edik ügyfél regret értéke ezen két költségnövekedési érték különbségének maximális értéke az összes lehetséges j beillesztési lehetőségen keresztül. Ez a különbség szemlélteti, hogy milyen költséget von maga után, ha az optimális helyett a második legjobb beillesztési lehetőség mellett döntünk.

Az algoritmus szempontjából minden beszúrásnál a regret értéket kell minimalizálom, melyhez a 4.6. algoritmust oly módon módosítottam, hogy minden lehetséges beszúrási kombinációra kiszámolom a regret értéket, majd ez alapján rendeztem C elemeit.

4.7. Adaptív keresés

A keresés során alkalmazott romboló és beszúró metódusok kiválasztása történhet teljesen véletlenszerűen, vagy determinisztikusan is egy előre meghatározott sémát követve is akár. Annak érdekében, hogy hatékonyabb keresést érjünk el érdemes lehet az olyan módszereket gyakrabban alkalmazni, melyekkel a legjobb eredményeket értük el korábban. Erre nyújt megoldást a következőekben bemutatott adaptív módszer, ahol a keresés során a romboló és beszúró metódusok egy súlyozott valószínűsége változik az előző iterációk során elért eredmények alapján.

A módszerek súlyozását egy előre meghatározott itárció szám után frissítjük, vagyis a teljes keresést kisebb szegmensekre bontjuk, jelen munkámban 100 iterációra. Ezen szegmensek alatt pontszámokat gyűjtünk a metódusok teljesítménye alapján, majd ezeket a pontszámokat és az adott metódus alkalmazásának a számát felhasználva súlyozunk. A pontszámítás során a következő eseteket különböztetjük meg:

σ_1 Ha új globális optimumot találtunk.

σ_2 Ha az új megoldás elfogadásra került, jobb mint az előző megoldás és nem volt még ilyen megoldás.

σ_3 Ha az új megoldás elfogadásra került, rosszabb mint az előző megoldás és nem volt még ilyen megoldás.

Minden iteráció végén megvizsgáljuk, hogy melyik σ feltétele teljesül, és annak értékét hozzáadjuk az i metódushoz tartozó p_i pontszámhoz. Ezt a pontszámot mindig az adott szegmensre vonatkozóan számoljuk, amint egy új kezdődik, a korábbi pontszámok nullázódnak. A súlyozás során a következő számítást (4.17. egyenlet) hajtjuk végre, ahol H jelöli a heursiztikák halmazát:

$$w_i^{\text{new}} = w_i \cdot (1 - r) + r \cdot \left(\frac{p_i}{n_i}\right) \qquad \forall i \in H \tag{4.17}$$

A képletben w_i^{new} a heurisztika új súlya, w_i az előző súly, α az adaptív súlyozás paramétere, p_i a pontszám, n_i pedig az i metódus alkalmazásának száma az adott szegmens alatt. Ez a súlyozási folyamat segíti elő, hogy a sikeresebb módszerek nagyobb valószínűséggel kerüljenek kiválasztásra a jövőbeli iterációkban, így hosszú távra nézve javítva az eredmények minőségét.

Az adaptív módszer további fontos eleme a metódusválasztás. A választás során minden heurisztika valószínűsége a saját súlyának és a teljes súlyösszeg arányán alapszik. Az egyes heurisztikák kiválasztási valószínűségét az alábbi 4.18. képlet határozza meg:

$$P(i) = \frac{w_i}{\sum_{j \in H} w_j} \tag{4.18}$$

Az egyenletben $P(i)$ jelöli az i módszer kiválasztási valószínűségét, w_i az i módszer súlya, míg a $\sum_{j \in H} w_j$ a teljes súlyösszeg. Ez a képlet biztosítja, hogy a heurisztikák kiválasztási valószínűsége közvetlenül arányos legyen a korábbi teljesítményükkel, így preferálva azokat a módszereket, melyek jobban teljesítenek.

5. fejezet

Tesztelés

Az előzőekben bemutatott módszert valós adatok alapján generált bemeneteken teszteltem, ezeken vizsgálva a módszer hatékonyságát és alkalmazhatóságát. Az 5.1. alfejezetben ismertetem a tesztelés környezetét, ezt követően az adathalmazra vonatkozó információkat és a teszteredményeket részletezem. A tesztadatokat egy valós hulladékszállító társaság adatai alapján generáltam, melyről az 5.2. alfejezetben írok. Ezt követően az 5.3. alfejezetben ismertetem a tesztelés menetét, majd a teszteredményeket az 5.4. alfejezetben. Legvégül pedig az 5.5. alfejezetben az adaptivitás hatását vizsgálom.

5.1. Tesztelés környezete

A dolgozatomban javasolt megoldó módszert Python programozási nyelven implementáltam. A megoldás része egy MILP modell is napi útvonaltervek előállítására. Ennek a modellnek megoldására számos megoldó szoftvercsomag (solver) létezik. lIyen optimalizálási eszköz például a GLPK[1] nyílt forráskódú megoldója, vagy az iparban igen elterjedt CPLEX[2], illetve Gurobi[3] solverek, melyek jóval hatékonyabbak, mint az előbb említett GLPK. A MILP modellt minden esetben a Gurobi 10.0.1-es verziójával oldottam meg, a Gurobi Python API-t (gurobipy) használva. Az implementáció futtatáshoz a Python 3.12.3 verzióját használtam. Az eredmények feldolgozása szintén Pythonban implementált scriptekkel történt, ahol vizuális

[1] `https://www.gnu.org/software/glpk/`
[2] `https://www.ibm.com/products/ilog-cplex-optimization-studio`
[3] `https://www.gurobi.com/`

ábrázoláshoz a Matplotlib külső könyvtárat használtam. A teszteseteket egyetlen számítógépen futtattam, Apple M2 modellű processzorral és 16 GB-os memóriával.

5.2. Tesztadatok

Az implementáció vizsgálatához szükséges bemeneti példákat egy hulladékgyűjtő társaság valós adatai alapján generáltam. A társaság eszközei heti rendszerességgel látogatnak gyűjtőpontokat egy adott régión belül, minden gyűjtőpontot a hét ugyanazon napján. A társaság eredeti, naponta történő tervezésében ez naptól függően kb. 400-900 meglátogatandó pontot jelent.

A bemenetekben a fenti gyűjtőpontokat tekintettem az ügyfeleknek, a látogatás eredeti napját pedig az ügyfél preferált napjának. A társaság egyetlen telephellyel és lerakóhellyel rendelkezett, ezek szerepelnek minden bementi példában depóként és hulladéklerakó telephelyként.

A fenti valós adatokból több különböző méretű bemeneti példahalmazt generáltam. A tervezési időszak minden példában 4 hetet tartalmazott, tehát 28 napból állt. Ezeket az adathalmazokat az egy hétre jutó meglátogatandó ügyfelek számával különböztetek meg. Minden halmazra véletlenszerűen kiválasztva az ügyfeleket 10 különféle bemenetet generáltam 50, 100 és 150 egy hétre jutó ügyféllel, azaz a tervezési időszakban 200, 400 és 600 látogatással. A preferált napok adottak voltak, a két látogatás között maximális eltérési időt 8 napban határoztam meg. Az egy körútra vonatkozó megszorítások közül a maximum úthossznak 500 km-t állítottam be és kiszolgálási idővel nem számoltam, így csak a távolságra definiáltam napi korlátot. Minden ügyfélhez rendelt súly 1 volt, azaz minden ügyfélnek azonos volt a fontossága a megoldás szempontjából. Az egy járműre vonatkozó kapacitás pedig 100 egységre paramétereztem.

5.3. A tesztelés menete

Minden bemenetre elkészítettem a preferált nap szerinti optimális tervezést, ami alatt azt a megoldást értjük, amikor minden ügyfelet pontosan a preferált napján szolgálunk ki. Ezekre a tervezésekre a továbbiakban **PrefOpt**-ként fogok hivatkozni. Ezeket az optimumokat a 4.4.1 fejezetben megadott MILP modellel kaptam. A megoldások által adott költségek összehasonlításként szolgálnak majd az implementált

módszer hatékonyságának kiértékeléséhez. Ezeknek a MILP modelleknek az optimális megoldása már ilyen méretű példák esetén is nehéz, sok esetben 1-2 napot is igénybe vett az optimum meghatározása, így nem mondhatóak gyors és hatékony megközelítésnek a feladat megoldására.

A bemenetekre minden esetben 4 alkalommal futtattam az implementált módszert. A futtatások között csak a preferált naptól való eltérésért járó büntetés mértéke különbözött. A teszteset során megvizsgáltam, hogy milyen módon befolyásolja a megoldás minőségét a δ büntetési érték változtatása. Ezt PrefOpt megoldásokhoz viszonyítottam, ahol minden ügyfél a preferált napján került kiszolgálásra, ezáltal büntetés mértéke nem befolyásolja a célfüggvény értékét. A tesztesetek során a δ értéket a PrefOpt megoldás egy napra eső átlagköltségének a 0%, 5%, 10% és 15%-ára állítottam. Az első esetben értelemszerűen nincs büntetés a preferált naptól való eltérés esetén, a második esetben egy átlagos napi költség 5%-a lesz minden eltérő ügyfél esetén, és így tovább.

A kapott eredményeket a preferált napokon történő optimális kiszolgáláshoz viszonyítva értékeltem ki, azt vizsgálva, hogy milyen büntetési értékek esetén lehetséges-e olcsóbb útvonaltervet ennél, illetve ha túl magas a büntetés, akkor megközelíti-e a PrefOpt értékét az ALNS által adott megoldás.

Az ALNS-t minden esetben a BestFit által adott kezdeti megoldásról indítottam. A módszer paraméterezéséhez a Ropke és Pisinger [12] publikációban megadott értékeket vettem alapul, melyek a következők:

Paraméter	Érték	Alkalmazási hely
p	10	worst removal (4.5.1. fejezet)
r	0.1	adaptív súlyozás (4.7. fejezet)
σ_1	33	heurisztikák jutalmazása (4.7. fejezet)
σ_2	9	heurisztikák jutalmazása (4.7. fejezet)
σ_3	13	heurisztikák jutalmazása (4.7. fejezet)
w	0.05	szimulált hűtés (4.3. fejezet)

5.1. táblázat. A tesztelés során használt paraméterek

A módszert minden példa esetén 25000 iterációszámmal futtattam. A legnagyobb 150 ügyfeles példákra ez futásonként nagyjából 10000 másodpercet jelent. Míg ez a 2,5-3 órás futásidő soknak tűnhet, egy 4 hetes időszakban összesen 600 ügyfélkiszol-

gálás ütemezéséről szól a feladat, a periódusok közti korlátok figyelembevételével. Hasonló méretű (600 ügyfeles), egy napra történő VRP megoldását Ropke és Pisinger [12] átlagosan 2221 másodperc alatt tette meg egy hasonló ALNS módszerrel, míg a PrefOpt meghatározása egzakt módszer segítségével bizonyos bemenetekre több napig is eltarthat.

5.4. Futási eredmények

Először a periódusonként 50 ügyfeles példákra történő tesztelés eredményeit az 5.2. táblázatban mutatom be.

	PrefOpt	**Bestfit**	**0% bünt.**	**5% bünt.**	**10% bünt.**	**15% bünt.**
1	393.48	979.01	**70.6%**	**79.0%**	**90.8%**	100.8%
2	362.35	822.70	**74.1%**	**91.5%**	**98.1%**	111.3%
3	430.14	949.74	**80.7%**	**93.4%**	105.3%	102.8%
4	414.51	898.58	**63.4%**	**73.8%**	**89.6%**	**99.5%**
5	438.41	856.14	**71.4%**	**84.6%**	**97.5%**	100.4%
6	325.83	754.22	**72.8%**	**91.3%**	**98.4%**	102.3%
7	404.67	919.41	**76.6%**	**86.2%**	100.2%	110.9%
8	361.72	852.29	**73.2%**	**89.9%**	**94.8%**	101.4%
9	350.81	893.21	**73.3%**	**93.3%**	107.4%	113.3%
10	311.15	705.14	**66.0%**	**76.6%**	**89.3%**	**89.6%**
Átl.			**72.2%**	**86.0%**	**97.1%**	103.2%

5.2. táblázat. Periódusonként 50 ügyfeles példa vizsgálata különböző büntetési szinteken

A táblázat első oszlopa a példa sorszámát jelöli, majd közvetlenül mellette az első oszlop az adott példához tartozó PrefOpt érték található. A harmadik oszlop tartalmazza a BestFit értéket, melyek az ALNS kezdeti megoldását jelentették. A többi oszlopban láthatóak a különböző büntetési szintekhez tartozó eredmények, melyeket az ALNS futtatásával értem el. Ezek százalékos értékek, melyek azt mutatják, hogy milyen mértékben tér el az ALNS adott példára kapott megoldása a PrefOpt értéktől. A táblázat minden sora egy adott bemeneten történő futtatások eredményeit tartalmazza, az utolsó sorban pedig átlagoltam ezeket minden büntetési

szintre vonatkozóan. Amennyiben az adott eredmény jobb mint a PrefOpt, az adott cella szövegét félkövérrel emeltem ki. Jól látható, hogy a büntetési költség nélküli esetben jelentős javítást sikerült elérni. A 3-as példa kivételével, ahol 19.3% volt a javulás a PrefOpt értékhez képest, minden más esetben több, mint 20%-os. A többi büntetési szintekhez tartozó átlagok alapján látható, hogy a 15% kivételével minden esetben javított a módszer a PrefOpt értékén, ahol pedig a magas büntetés miatt nem, vagy csak nehezen tudott, ott átlagban megtalálta a PrefOpt által adott költségeket, amikor minden ügyfelet a preferált napján keresünk fel. A módszer ezekre a bemeneti példákra átlagosan 6-8 perc alatt találta meg a táblázatban bemutatott megoldásokat.

A periódusonként 100 ügyfeles eredmények az 5.3. táblázatban találhatóak, ahol jelentősen nőtt a feladat mérete és így a futási idő is. A leglasabb futtatásokhoz itt már egy óra is szükséges volt.

	PrefOpt	**Bestfit**	**0% bünt.**	**5% bünt.**	**10% bünt.**	**15% bünt.**
1	536.79	1493.75	**77.6%**	**98.0%**	108.7%	112.2%
2	524.51	1434.98	**74.6%**	104.1%	121.5%	112.1%
3	488.98	1267.34	**71.1%**	108.1%	109.7%	111.3%
4	540.77	1424.99	**81.1%**	104.2%	110.2%	108.4%
5	590.64	1533.43	**74.8%**	**98.6%**	106.0%	101.8%
6	508.09	1333.96	**73.9%**	**95.9%**	114.7%	110.2%
7	541.75	1354.61	**72.7%**	**98.0%**	**93.1%**	101.7%
8	505.20	1403.16	**82.3%**	113.6%	120.0%	111.3%
9	526.59	1335.19	**82.6%**	104.3%	128.3%	112.9%
10	544.88	1326.03	**72.1%**	105.1%	110.7%	110.1%
Átl.			**76.3%**	103.0%	112.3%	109.2%

5.3. táblázat. Periódusonként 100 ügyféles példa vizsgálata különböző büntetési szinteken

A táblázatok felépítése ugyanazt a sémát követi, mint az előző 5.2. táblázatban. Az eredmények azt mutatják, hogy egy ilyen nagyobb példa esetén a büntetési értékekkel nehezebb elérni a PrefOpt értékeket, azonban több példa esetén még mindig sikerül a javulás az 5% büntetésnél, illetve a 7-es sorszámú bemenetnél a 10% bünte-

tésnél 6.9%-os javulást sikerült elérni. Jól látszik azonban, hogy a többi példa esetén is sikeresen közelíti a PrefOpt értékeket az implementált módszer.

A harmadik példahalmaz, a 150 ügyfeles példák eredményei az 5.4. táblázatban láthatóak. A táblázat szintén ugyanazt a sémát követi, mint az előzőekben bemutatottak. Az ügyfél szám növekésével még tovább nőtt a feladat bonyolultsága, ezáltal a futási idő is, mely a legtöbb példa esetén 3 óra körül alakult.

	PrefOpt	**Bestfit**	**0% bünt.**	**5% bünt.**	**10% bünt.**	**15% bünt.**
1	652.98	1831.95	**71.3%**	109.5%	120.9%	110.5%
2	651.19	1764.17	**73.5%**	111.2%	115.7%	107.7%
3	625.02	1874.25	**77.5%**	123.0%	143.8%	117.8%
4	699.49	2012.51	**71.2%**	117.6%	117.8%	114.7%
5	544.55	1670.17	**82.3%**	122.2%	128.2%	125.7%
6	592.34	1829.04	**80.5%**	111.2%	121.7%	122.0%
7	568.33	1727.41	**74.4%**	118.9%	130.9%	116.8%
8	616.18	1719.06	**78.8%**	113.0%	131.0%	112.6%
9	687.58	1728.78	**68.0%**	**95.8%**	105.2%	101.9%
10	609.55	1774.85	**70.1%**	107.6%	119.5%	112.5%
Átl.			**74.8%**	113.0%	123.5%	114.2%

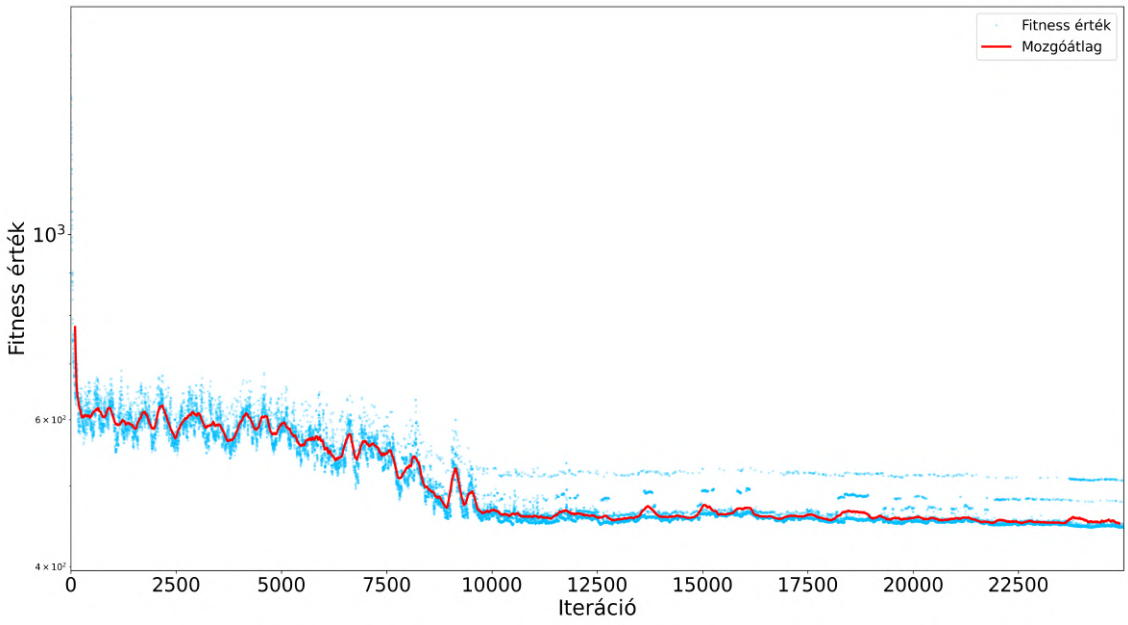
5.4. táblázat. Periódusonként 150 ügyfeles példa vizsgálata különböző büntetési szinteken

A táblázatból látszik, hogy míg 0% büntetés esetén talál a módszer a PrefOpt-nál jobb megoldásokat, már az 5% büntetési értéknél csak egyetlen példa, a 9-es sorszámú esetében sikerült jobb eredményt elérni, ahol 4.2%-os a javulás. A többi eredményről is leolvasható, hogy az ALNS sikeresen közelítette a PrefOpt értékeket, de a példák bonyolultsága miatt már nem volt képes az előzőekben említett paraméterekkel és a 25000-es iterációszámmal megjavítani azokat. Az előzetes tesztelések azt mutatták, hogy az iteráció szám növelése a módszer hatékonyságát növeli, így a továbbiakban érdemes lehet a futtatások számát növelni vagy a kilépési feltételeket módosítani. Ezenfelül nem kisérleteztem a paraméterek finomhangolásával, mely szintén további javulást jelenthetne.

5.5. Az adaptivitás hatása

Az előzőekben bemutatott generált példák során az eredmények azt mutatták, hogy a nagyobb iteráció szám alatt végzett futtatások esetén hatékonyabban funkcionált az implementált módszer. Ez az adaptivitásnak is köszönhető, melynek hosszú távú hatása az, hogy a módszer képes a futás során változtatni az aktuálisan jobban teljesítő romboló és beszúró módszerek kiválasztásának a valószínűségein. Ha egy módszer jobban teljesít az aktuálisan vizsgált szomszédságon, akkor a következő szegmensben (ami 100 iterációnak felel meg) gyakrabban kerül alkalmazásra. Ha egy módszer egyáltalán nem képes javítani az aktuális megoldáson több eltelt szegmens alatt, úgy az egy idő után már egyáltalán nem kerül meghívásra. Az ALNS ilyen módú működését a 9-es sorszámmal jelölt, periódusoként 150 ügyfeles teszteset eredményén vizsgáltam, ahol 25000 volt az iterációk száma.

Az ALNS futása során az aktuális megoldások $f(s)$ fitness értékeinek változását az 5.1. ábrán szemléltetem.

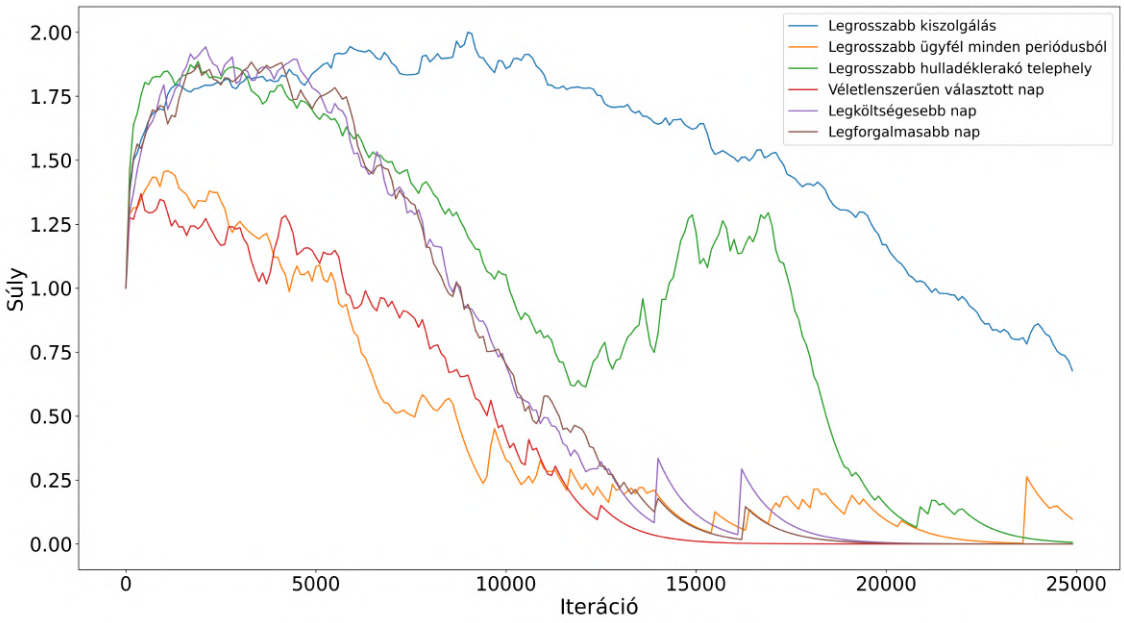


5.1. ábra. A fitness érték változása

Az x tengelyen az iterációk száma, az y tengelyen pedig a fitness érték látható logaritmikus skálázást alkalmazva a jobb átláthatóság érdekében. Az iterációk során kapott ideiglenes megoldások fitness értékeit a kék pontok jelölik, míg a piros vonal ezeknek a mozgó átlagértékét mutatja. Az átlag kezdeti meredek csökkenése azt mutatja, hogy a módszer a keresés elején gyorsan talál egy lokális optimumot, majd

egy-egy nagyobb ugrással kilép belőle, majd újra javítani tud a megoldáson. Ez a viselkedés jól szemlélteti az ALNS működését. Az adaptivitás hatása abban is megmutatkozik, hogy a későbbi iterációk során finomhangolódik a romboló és beszúró heurisztikák alkalmazása, és így kevésbé meredeken, de javul az aktuálisan vizsgált fitnessek átlaga.

Az ALNS a kezdeti 100 iterációs szegmensben azonos valószínűséggel választ romboló és beszúró heurisztikákat, de ezek után az adaptivitás miatt ezek különböző súlyokat kapnak attól függően, hogy hogyan teljesítettek az előző szegmensekben. Ezek a súlyok befolyásolják a heurisztikák kiválasztásának a valószínűségét. Az általam alkalmazott romboló módszerek súlyainak az alakulását az 5.2. ábrán szemléltetem.

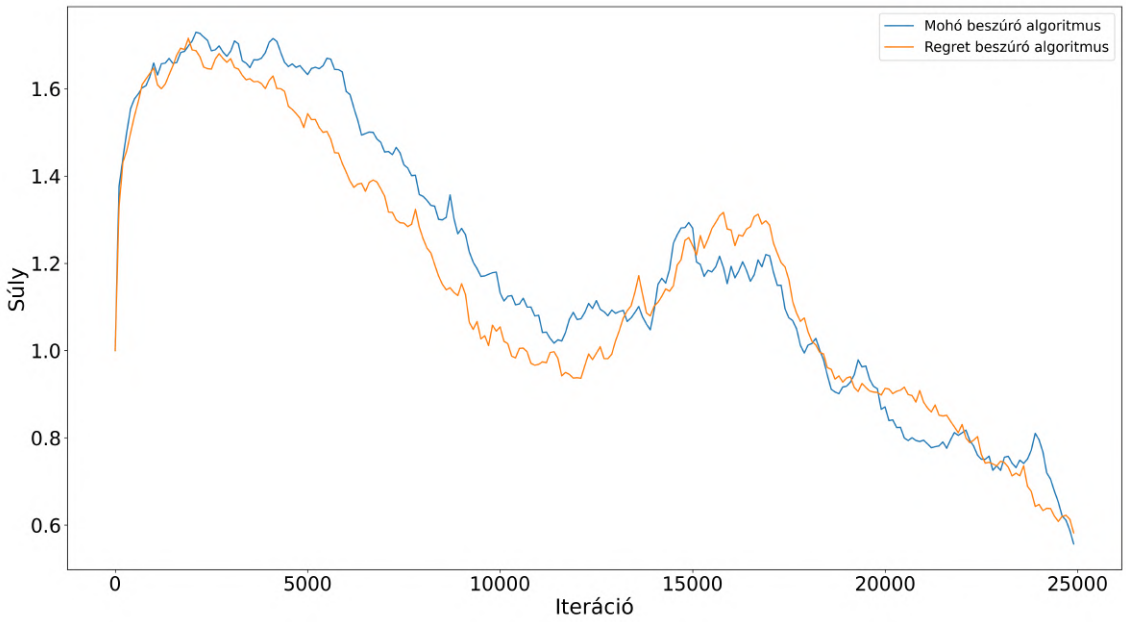


5.2. ábra. A romboló módszerek súlyainak az alakulása

Az ábra y tengelyén az iteráció számokat jelöltem, míg az x tengelyen a súlyokat. Különböző színű vonalak különböztetik meg a romboló módszerek súlyainak az alakulását. Az eredmények azt mutatják, hogy a "Legrosszabb kiszolgálás" heurisztika teljesített a leghatékonyabban. A keresés első feléről leolvasható, hogy a módszerek mindegyike nagyobb valószínűséggel került kiválasztásra. Ez egyértelműen mutatja a 4.3. fejezetben bemutatott elfogadási mechanizmus működését, ahol a szimulált hűtésnek köszönhetően az elfogadási valószínűség a kezdeti iterációk során magasabb, így a súlyok is magasabbak. A keresés vége felé a súlyok egyre inkább csökkennek,

bizonyos módszereket, például a "Véletlenszerűen választott nap törlése"-t egyáltalán nem is használja már az ALNS. Vannak olyan módszerek is, pl a "Legrosszabb hulladéklerakóhely törlése", amik egy kezdeti rosszabb teljesítés után 15000 iteráció környékén hirtelen megint nagyon sokat tudnak javítani az aktuális megoldáson.

A beszúró módszerek súlyainak alakulását is hasonló módon lehet elemezni, melyet az 5.3. ábrán mutatok be.



5.3. ábra. A beszúró módszerek súlyainak az alakulása

Az ábra ugyanolyan sémát követ, mint az előző. Munkám során két beszúró módszert különböztettem meg, egyik a "Mohó beszúró algoritmus", míg a másik a "Regret beszúró algoritmus" módszer. Ugyanúgy, mint a romboló heurisztikák esetében itt is az látható, hogy a keresés elején magasabbak a súlyok, majd a keresés során azok konvergálnak. Nem állapítható meg nagy különbség a két módszer között, hasonlóan teljesítettek a keresés során, bizonyos szegmensekben az egyik, másokban pedig a másik volt picivel jobb.

6. fejezet

Összefoglalás

Munkám során feldolgoztam a VRP és MPVRP problémákkal foglalkozó irodalmat, majd azonosítottam utóbbinak egy olyan aspektusát, melynek külön részeivel már több kutatás is foglalkozott, azonban a teljes probléma még nem került kidolgozásra. A feladat magába foglalta a hulladékgyűjtés során felmerülő útvonaltervezési problémának a sajátosságait, mint például a hosszú távú tervezés, a preferált napok és az azoktól megengedett eltérés, vagy a hulladéklerakó telephelyek bevezetése. A definiált feladatra kidolgozásra került egy ALNS alapú megoldó módszer, mely adaptív módon változtatja a keresés során használt módszereket. A sikeres adaptáció implementálásra került, melyet valós adatokon teszteltem. Megvizsgáltam a módszer hatékonyságát különböző szempontok alapján, és a tesztek eredményei alapján megállapítható, hogy a módszer hatékonyan alkalmazható a vizsgált problémára.

A téma még számos lehetőséget rejt magában, melyeket későbbi kutatásomban szeretnék még részletesebben feltérképezni és körüljárni. Néhány, már most felvetődött kiegészítés:

Az implementáció további finomítása: A jelenlegi implementáció még nem tökéletes, bizonyos pontokon még lehetne növelni a kód hatékonyságát, optimalizálni a futási időt. Tovább gyorsítható lehetne még a keresési folyamat alacsonyabb szintű programozási nyelven történő implementálásával, mint például a C++.

A módszer további tesztelése: A dolgozatomban bemutatott eredmények alapján sejthető, hogy a a javasolt módszer hatékonyan alkalmazható, azonban további tesztekkel teljes körűen meg lehetne erősíteni a módszer hatékonyságát. Érdemes lehetne megvizsgálni, hogy még nagyobb iteráció számokkal milyen eredmények ér-

hetőek el. Ezenfelül jövőbeli terveim között van a szakirodalmi benchmark tesztek kiterjesztése a problémára és az azokon való tesztelés.

További romboló és beszúró módszerek: A bemutatott megoldásban is elég hatékonynak bizonyultak az alkalmazott módszerek, de további heurisztikák bevezetésével lehetne akár még jobb eredményeket elérni, esetlegesen kevesebb iteráció alatt hatékonyabban javítani a megoldásokat. Többek között érdemes lehet egy MILP alapján működő beszúró módszer alkalmazása is, ahol például a visszaszúrás során az adott jármű útvonalát lehetne optimalizálni, mely egy kisebb TSP problémát jelentene.

MILP modell kidolgozása: Mivel a vizsgált bemeneteken MPVRP problémára vonatkozó optimuma nem ismert, érdemes lenne ezeket az értékeket is meghatározni, és összehasonlítani az ALNS által talált megoldásokkal. Ennek első lépese egy MILP modell felírása a többperiódusú problémára. Mivel már kis problémákra se garantálható a modell hatékonysága, mindenképp meg kell vizsgálnom közelítő megoldásokat erre a modellre. A modell definiálásához jó kiindulási pontként szolgálhat a 4.4.1. fejezetben bemutatott MILP modell, melyet kiegészíthetünk a periodikus megoldásokra vonatkozó döntési változókkal és korlátokkal.

Köszönyetnyilvánítás

Ezúton szeretnék köszönetet mondani külső témavezetőimnek, Dr. Dávid Balázsnak, aki időt nem sajnálva támogatta a munkám és rengeteg értékes szakmai tanáccsal látott el. Köszönettel tartozom belső témavezetőmnek Tóth Teklának, hogy rendszeres konzultációk során segítette a munkámat. Továbbá köszönöm az InnoRenew CoE kutatóintézetnek, hogy szakmai gyakorlat keretein belül lehetőséget biztosított a tudományos munkám hatékony elvégzésére, valamint az ott dolgozó kollégáknak, akik támogattak és segítséget nyújtottak a kutatásom során.

Végül szeretnék köszönetet mondani a Szlovén Kutatási és Innovációs Alapnak (ARIS) az N1-0223 projekt támogatásáért, valamint a Szlovén Kutatási és Innovációs Alap (ARIS) és a Gazdasági, Turisztikai és Sportminisztérium (METS) közös támogatásáért a CRP 2023 program V4-2351 projektjén keresztül.

Irodalomjegyzék

[1] M. Grötschel, M. Jünger és G. Reinelt. „Optimal control of plotting and drilling machines: A case study". *ZOR Zeitschrift für Operations Research Methods and Models of Operations Research* 35 (1 1991). ISSN: 03409422. DOI: 10.1007/BF01415960.

[2] Robert G. Bland és David F. Shallcross. „Large travelling salesman problems arising from experiments in X-ray crystallography: A preliminary report on computation". *Operations Research Letters* 8 (3 1989). ISSN: 01676377. DOI: 10.1016/0167-6377(89)90037-0.

[3] William J Cook. *In pursuit of the traveling salesman: mathematics at the limits of computation*. Princeton University Press, 2015.

[4] Richard M. Karp. „Reducibility among Combinatorial Problems". Boston, MA: Springer US, 1972, 85–103. old. ISBN: 978-1-4684-2001-2. DOI: 10.1007/978-1-4684-2001-2_9.

[5] G. B. Dantzig és J. H. Ramser. „The Truck Dispatching Problem". *Management Science* 6 (1 1959). ISSN: 0025-1909. DOI: 10.1287/mnsc.6.1.80.

[6] G. Clarke és J. W. Wright. „Scheduling of Vehicles from a Central Depot to a Number of Delivery Points". *Operations Research* 12 (4 1964). ISSN: 0030-364X. DOI: 10.1287/opre.12.4.568.

[7] Gilbert Laporte. „The vehicle routing problem: An overview of exact and approximate algorithms". *European Journal of Operational Research* 59 (3 1992). ISSN: 03772217. DOI: 10.1016/0377-2217(92)90192-C.

[8] J. K. Lenstra és A. H.G.Rinnooy Kan. „Complexity of vehicle routing and scheduling problems". *Networks* 11 (2 1981). ISSN: 10970037. DOI: 10.1002/net.3230110211.

[9] John H Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[10] Barrie M. Baker és M. A. Ayechew. „A genetic algorithm for the vehicle routing problem". *Computers and Operations Research* 30 (5 2003). ISSN: 03050548. DOI: 10.1016/S0305-0548(02)00051-5.

[11] Paul Shaw. „Using constraint programming and local search methods to solve vehicle routing problems". 1520. köt. 1998. DOI: 10.1007/3-540-49481-2_30.

[12] Stefan Ropke és David Pisinger. „An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows". *Transportation Science* 40 (4 2006). ISSN: 15265447. DOI: 10.1287/trsc.1050.0135.

[13] Michel Gendreau, Alain Hertz és Gilbert Laporte. „Tabu search heuristic for the vehicle routing problem". *Management Science* 40 (10 1994). ISSN: 00251909. DOI: 10.1287/mnsc.40.10.1276.

[14] Ibrahim Hassan Osman. „Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem". *Annals of Operations Research* 41 (4 1993). ISSN: 02545330. DOI: 10.1007/BF02023004.

[15] Paolo Toth és Daniele Vigo. *The vehicle routing problem*. SIAM, 2002.

[16] Jens Lysgaard, Adam N. Letchford és Richard W. Eglese. „A new branch-and-cut algorithm for the capacitated vehicle routing problem". *Mathematical Programming* 100 (2 2004). ISSN: 00255610. DOI: 10.1007/s10107-003-0481-8.

[17] Jean François Cordeau, Michel Gendreau és Gilbert Laporte. „A tabu search heuristic for periodic and multi-depot vehicle routing problems". *Networks* 30 (2 1997). ISSN: 00283045. DOI: 10.1002/(SICI)1097-0037(199709)30:2<105::AID-NET5>3.0.CO;2-G.

[18] Gerardo Berbeglia és tsai. „Static pickup and delivery problems: a classification scheme and survey". *TOP* 15 (1 2007). ISSN: 1134-5764. DOI: 10.1007/s11750-007-0009-0.

[19] Marius M. Solomon. „Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints". *Operations Research* 35 (2 1987). ISSN: 0030364X. DOI: 10.1287/opre.35.2.254.

[20] Moshe Dror és Pierre Trudeau. „Split delivery routing”. *Naval Research Logistics (NRL)* 37 (3 1990). ISSN: 15206750. DOI: 10.1002/nav.3800370304.

[21] Katja Buhrkal, Allan Larsen és Stefan Ropke. „The Waste Collection Vehicle Routing Problem with Time Windows in a City Logistics Context”. *Procedia - Social and Behavioral Sciences* 39 (2012). ISSN: 18770428. DOI: 10.1016/j.sbspro.2012.03.105.

[22] G. Paletta. „A multiperiod traveling salesman problem: Heuristic algorithms”. *Computers and Operations Research* 19 (8 1992). ISSN: 03050548. DOI: 10.1016/0305-0548(92)90018-Z.

[23] Robert A. Russell. „Technical Note—An Effective Heuristic for the M -Tour Traveling Salesman Problem with Some Side Conditions”. *Operations Research* 25 (3 1977). ISSN: 0030-364X. DOI: 10.1287/opre.25.3.517.

[24] E. J. Beltrami és L. D. Bodin. „Networks and vehicle routing for municipal waste collection”. *Networks* 4 (1 1974). ISSN: 10970037. DOI: 10.1002/net.3230040106.

[25] R. Russell és W. Igo. „An assignment routing problem”. *Networks* 9 (1 1979). ISSN: 10970037. DOI: 10.1002/net.3230090102.

[26] N. Christofides és J. E. Beasley. „The period routing problem”. *Networks* 14 (2 1984). ISSN: 10970037. DOI: 10.1002/net.3230140205.

[27] I-Ming -M Chao, Bruce L. Golden és Edward Wasil. „An improved heuristic for the period vehicle routing problem”. *Networks* 26 (1 1995). ISSN: 10970037. DOI: 10.1002/net.3230260104.

[28] Thibaut Vidal és tsai. „A hybrid genetic algorithm for multidepot and periodic vehicle routing problems”. *Operations Research* 60 (3 2012). ISSN: 0030364X. DOI: 10.1287/opre.1120.1048.

[29] M. Gaudioso és G. Paletta. „Heuristic for the periodic vehicle routing problem”. *Transportation Science* 26 (2 1992). ISSN: 00411655. DOI: 10.1287/trsc.26.2.86.

[30] Chris Groër, Bruce Golden és Edward Wasil. „The consistent vehicle routing problem”. *Manufacturing and Service Operations Management* 11 (4 2009). ISSN: 15234614. DOI: 10.1287/msom.1080.0243.

[31] Ashlea R. Bennett és Alan L. Erera. „Dynamic periodic fixed appointment scheduling for home health". *IIE Transactions on Healthcare Systems Engineering* 1 (1 2011). ISSN: 19488319. DOI: 10.1080/19488300.2010.549818.

[32] Kunlei Lian. *Service consistency in vehicle routing*. University of Arkansas, 2017.

[33] A. Estrada-Moreno és tsai. „Biased-randomized iterated local search for a multiperiod vehicle routing problem with price discounts for delivery flexibility". *International Transactions in Operational Research* 26 (4 2019). ISSN: 14753995. DOI: 10.1111/itor.12625.

Ábrák jegyzéke

Táblázatok jegyzéke

Algoritmusok jegyzéke