Dinamikus járműútvonal-tervezés

HATALA IMRE
Alkalmazott matematikus MSc
Operációkutatás specializáció

Témavezető: HORVÁTH MARKÓ
Mérnöki és Üzleti Intelligencia Kutatólaboratórium, HUN-REN SZTAKI

Belső konzulens: KIS TAMÁS
Operációkutatási Tanszék, ELTE TTK
Mérnöki és Üzleti Intelligencia Kutatólaboratórium, HUN-REN SZTAKI

Eötvös Loránd Tudományegyetem, Természettudományi Kar
Budapest, 2024.

Köszönetnyilvánítás

Ez a diplomamunka nem jöhetett volna létre a sok-sok támogatás nélkül, amelyet a megírása során kaptam. Hálával tartozom mindazoknak, akik lélekben támogattak, mert ezen a megtartó alapon nyugszik minden. Köszönetem szeretném kifejezni továbbá témavezetőmnek, Horváth Markónak a rászánt időért és energiáért, a rengeteg segítségért. Köszönöm a türelmét, útmutatásait és tanácsait.

Tartalomjegyzék

1. Bevezetés

A mindennapi életben észrevétlenül is részesedünk a modern matematika vívmányainak pozitív hatásából. A logisztikai optimalizálás egyik központi témája, a járműútvonal-tervezés ebből a szempontból különösen forró terület. A szállítási folyamatok hatékonysága ugyanis meghatározó a gazdaság működésében – legyen szó akár nagy léptékű ipari folyamatokról, akár olyan hétköznapi tevékenységekről, mint a csomagküldés vagy az ételkiszállítás. Ki ne szeretné, hogy az online rendelt ebédje gyorsan megérkezzen, vagy hogy ne kelljen napokat várni az új íróasztalra? Hasonlóan, melyik vállalat ne akarná csökkenteni a logisztikai költségeit, ezzel növelve a profitrátát és versenyelőnyét a riválisaival szemben?

De nem csak gazdasági és kényelmi szempontok miatt fontos az optimalizálás, hanem – napjainkban egyre hangsúlyosabban – környezetvédelmi szempontból is. Ha egy vállalat, tegyük fel, 10%-kal kisebb járműflottával is teljesíteni tudja az ügyfelek megrendeléseit, akkor az arányosan kevesebb környezeti terheléssel is jár. Sőt, a tágabb értelemben vett logisztika és ellátásilánc-menedzsment területén egyre elterjedtebb az a megközelítés, hogy az optimalizálásnak kifejezetten célja a környezetre gyakorolt hatások, például a karbonlábnyom minimalizálása is [33].

Jelen dolgozatban egy dinamikus járműútvonal-tervezési problémát vizsgálunk, amely az ICAPS 2021 konferencia optimalizálási versenyén jelent meg [17]. A feladat a Huawei infokommunikációs vállalat logisztikai rendszerén alapul, amelyben a gyáregységekből álló hálózaton kell az áruszállítást megtervezni. Egy felvétel-lerakási problémáról van szó, ahol a fel- és lerakodás a LIFO elv szerint történik. A szállítási kérelmek dinamikusan, tíz perces időközönként érkeznek be, és határidőre kell teljesíteni őket. A feladat további különlegessége, hogy a gyárak kiszolgáló kapacitása korlátozott, így az egyes lokációkban – az aktuálisan ott tartózkodó járművek számától függően – várakozással is számolni kell.

A dolgozat felépítése a következő. A 2. fejezetben először ismertetjük a klasszikus járműútvonal-tervezési feladatot (VRP) és kitérünk egyes speciális változatokra is. Ezután egy általános megoldási módszert, a lokálist keresés metaheurisz-

tikát mutatjuk be. A fejezet második részében a dinamikus járműútvonal-tervezési feladatokat (DVRP) tárgyaljuk. Ezek olyan problémák, ahol az input nem adott előre teljes egészében, hanem az információk egy része menet közben érkezik – így a tervezésnek valós időben kell történnie. A dinamikus problémák speciális megközelítést igényelnek, ennek megfelelően bemutatunk egy alkalmas modellezési keretrendszert is. Végül pedig különböző megoldási módszereket tárgyalunk.

A 3. fejezetben ismertetjük a dolgozat témájául szolgáló feladatot, amelyet az említett keretrendszer segítségével modellezünk. A fejezet végén áttekintjük az ismert megoldásokat, az eredeti verseny dobogós helyezettjeinek munkájától az azóta megjelent cikkekig. Ezután a 4. fejezetben ismertetjük a saját megközelítéseinket a feladat megoldására, amelyek közül a legjobb eredményt egy lokális keresésen alapuló módszer szolgáltatja. Végül bemutatjuk a számítási eredményeinket, és összehasonlítjuk más megoldásokkal.

2. Dinamikus járműútvonal-tervezés

A *járműútvonal-tervezési feladat* (*Vehicle Routing Problem*, VRP) az operációkutatás hőskorából származik. 1959-ben fogalmaztak meg először [10], a közismert *utazó ügynök probléma* (*Traveling Salesman Problem*, TSP [12]) általánosításaként. Ebben a fejezetben röviden ismertetjük a klasszikus VRP feladatot és néhány speciális változatát, majd áttérünk dinamikus VRP problémák témakörére. Mindkét esetben megoldási módszereket is tárgyalunk, elsősorban azokra a megközelítésekre koncentrálva, amelyek a dolgozatban később szerepelni fognak.

2.1. VRP feladatok

A VRP feladat során járművek optimális útvonalát kell meghatározni annak érdekében, hogy teljesítsük egy ügyfélkör megrendelésit. A megrendelések tipikusan áruk leszállítását jelentik, de vannak olyan problémák ahol utasok szállítását vagy szolgáltatások (földrajzilag elosztott) teljesítését foglalják magukba. A szállítási terv optimalizálása során különböző megkötéseket és célokat kell figyelembe vennünk a feladat típusától függően.

A klasszikus alapfeladatban adottak az ügyfelek lokációi és megrendelései, egy kitüntetett lokáció (a *depó*), ahol a szállítmány felvétele történik, valamint a járműflotta és a szállítási költségek. A járművek a depóból indulnak és oda térnek vissza, teljesíteniük kell az összes megrendelést, a cél pedig a megtett út (illetve a felmerülő költségek) minimalizálása. Vannak olyan változtok is, ahol több depó is szerepet kap, külön-külön saját járműflottával. A *felvétel-lerakási probléma* (*Pickup and Delivery Problem*, PDP) esetében a csomagok felvétele nem egyetlen kitüntetett lokáción történik, hanem minden megrendeléshez egyedi felvételi és lerakási pont tartozik. Ebben a feladatban a csomagok felvételét és kézbesítését egyaránt be kell tervezni a járművek útvonalába. Ha nem követeljük meg, hogy a járművek visszatérjenek a kezdeti lokációjukra, akkor *nyílt járműútvonal-tervezési feladatról* (*Open Vehicle Routing Problem*, OVRP) beszélünk.

Az egyes speciális problémák karakterisztikáját a feladatban szereplő korlátok határozzák meg. Ezek kapcsolódhatnak a járművekhez, az útvonalakhoz, a depóhoz (a felvételi és lerakási pontokhoz) és az ügyfelekhez is. A *kapacitáskorlátos* feladatban (*Capacitated Vehicle Routing Problem*, CVRP) például figyelembe kell venni a járművek terhelhetőségének korlátait. A rendelkezésre álló járműflotta lehet *homogén*, vagyis egyforma járművekből álló, illetve *heterogén*, amikor járművenként változó tulajdonságokkal (kapacitás, költség, sebesség stb.) kell számolni. Még az is előfordulhat – például egy vegyesen gépjárművekből és drónokból álló eszközpark esetén –, hogy az "úthálózat" is különbözik.

A PDP feladatok esetén gyakori megkötés, hogy a rakodásnak a *LIFO* (*Last In, First Out*) elvet kell követnie: először azt árucikket kell lerakodni, amelyik a legkésőbb került be a csomagtérbe. Ilyenkor az útvonalakat, vagyis a felvételi és lerakási pontok sorrendjét is ennek megfelelően kell megtervezni. A LIFO korlát tipikusan olyan feladatok során merül fel, ahol a járművek raktere csak egy irányból érhető el és az átrakodás nem megengedett – például törékeny áru esetén, vagy mert túl költséges lenne.

A depóra, illetve a felvételi és lerakási pontokra is vonatkozhat korlátozás, például ha a rakodási kapacitások limitáltak. Ilyenkor előfordulhat, hogy a létesítmény nem tud egyszerre kiszolgálni minden járművet, így néhánynak várakoznia kell. De az ügyfelek kapcsán is felmerülhetnek megkötések, például az *időablakos* variánsban (*Vehicle Routing Problem with Time Windows*, VRPTW) az egyes csomagok kézbesítése adott idősávban kell megtörténjen. A VRPTW speciális változata a *határidős* feladat (*Vehicle Routing Problem with Due Dates*, VRPDD), ahol a megrendeléseket egy előírt határidőre kell teljesíteni.

A klasszikus VRP feladatokban az optimalizálási cél a felmerülő költségek minimalizálása. Ezenkívül van olyan változat is, ahol a kézbesítés után bevétel termelődik, nem kötelező minden ügyfelet kiszolgálni, a cél pedig a maximális profit. Léteznek olyan problémák is, ahol több – gyakran egymásnak ellentmondó – célkitűzést is figyelembe kell vennünk (*Multi-objective Vehicle Routing Problem*, MOVRP). Ide tartoznak az olyan feladatok, amikor a költségek mellett az ügyfél

oldalán felmerülő kényelmetlenségeket is minimalizálni szeretnénk, például a kiszállítási idő vagy – határidős feladat esetén – a késés minimalizálásával. Ezek egymásnak ellentmondó célok, mert a minimális költség az erőforrások takarékos felhasználásával érhető el, a gyorsabb teljesítéshez viszont több erőforrást kell igénybe vennünk.

2.1.1. VRP feladatok megoldása lokális kereséssel

A VRP problémák megoldására számos egzakt és heurisztikus módszert ismerünk [9]. Mivel a VRP feladat – a TSP általánosításaként – NP-nehéz, így a gyakorlatban felmerülő nagy feladatokat jellemzően nincs lehetségünk egzakt módszerekkel megoldani, ezért a heurisztikus megközelítések kerülnek előtérbe. A problémaspecifikus heurisztikák közé tartoznak például a különböző dekompozíciós [25] és beillesztéses [6] technikák. Népszerűek továbbá az általános metaheurisztikák is [13], mint a lokális keresés és különböző változatai.

Most röviden bemutatjuk a lokális keresést, amelyet később az 4. fejezetben alkalmazni fogunk. A módszerről általános leírást adunk Talbi [27] alapján, majd Carrabs et al. [7] cikke alapján bemutatjuk a módszer alkalmazását olyan felvétellerakási feladatokra, ahol a LIFO szabály van érvényben.

A *lokális keresés* (*Local Search*, LS) egy széles körben alkalmazott metaheurisztika, amely számos optimalizálási probléma esetén alkalmazható, köztük VRP feladatok megoldására is. Alapelve, hogy az aktuális megoldás lokális környezetében keres javítást. A keresés egy adott kezdeti megoldásból indul, majd iterációkat hajt végre úgy, hogy minden iterációban az inkumbens megoldást megpróbálja egy jobbra cserélni. Ehhez azonban a keresési térnek csak egy szűk részhalmazát tekinti át, amelyet egy meghatározott szomszédsági feltétel határoz meg. A lokális keresés megáll, ha a lehetséges alternatív megoldások egyike sem javít a célfüggvényen. Ekkor az algoritmus egy – a szomszédsági feltételre nézve – lokális optimumot talált.

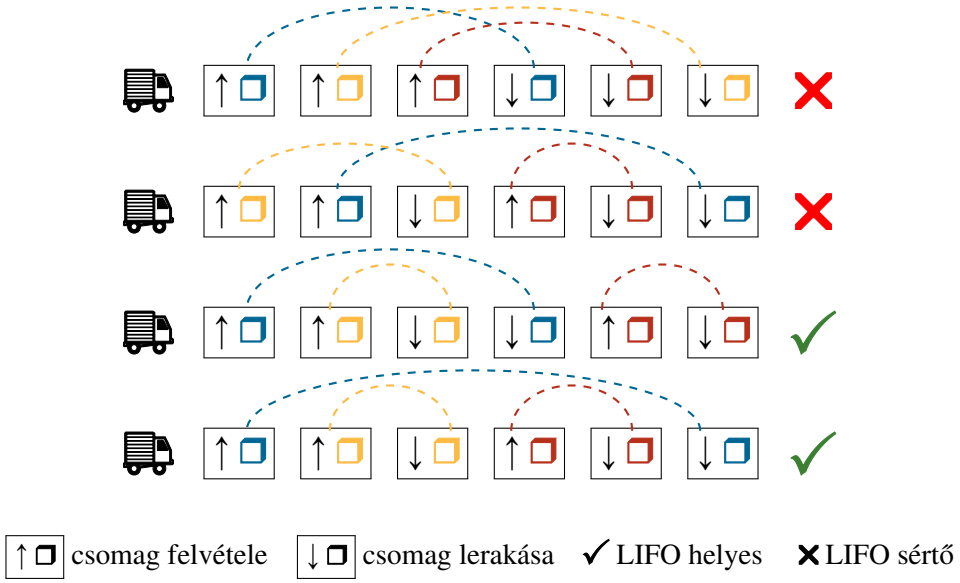
Egy konkrét feladatot megoldó lokális keresés algoritmus megalkotásához általánosan a következőkre van szükség. Meg kell határozni

1. egy szubrutint, ami kiszámít egy kezdeti megoldást,
2. a szomszédsági relációt,
3. az alternatívák közti választás stratégiáját,
4. a leállási feltételt.

Az általános tapasztalat szerint a lokális keresés rövid idő alatt is relatíve jó eredményt ad. Emellett viszonylag egyszerű megtervezni és implementálni is. Alapvető tulajdonsága azonban, hogy csupán lokális optimumot talál. Ha ez a megoldás "túl messze" van a valódi optimumtól, akkor nem lehetünk elégedettek az eredménnyel. A probléma megoldására, azaz hogy az algoritmus kilépjen a lokális optimumból, különböző technikák ismertek. Ezek közül most azzal a módszerrel foglalkozunk, amikor az algoritmus futása közben változtatjuk a szomszédsági feltételt, így érve el egyre jobb megoldásokat. Ennek a módszernek *változó szomszédságú keresés* (*Variable Neighborhood Search*, VNS [21]) a neve.

A szomszédsági relációt általában egy *operátor* segítségével definiáljuk, két különböző megoldást akkor tekintve szomszédosnak, ha az egyik megkapható a másikból az adott operátor alkalmazásával. (Az így kapott reláció nem feltétlenül szimmetrikus, de ez a lokális keresés működését nem befolyásolja.) Most Cassani and Righini [8] alapján bemutatunk négy olyan operátort, amelyeket az általunk vizsgált problémára alkalmazhatunk. Ehhez először azonban tisztázni kell, hogy miként néz ki egy megoldás a mi esetünkben.

A VRP feladat megoldása egy útvonalhalmaz, amely a járművek útvonalait írja le. Ha PDP feladatról van szó, akkor a felvételi pontnak minden csomag esetén meg kell előznie a lerakási pontot, továbbá azt is meg kell adni, hogy az útvonal egyes állomásain mely csomagokat kell felvennie vagy leraknia az adott járműnek. Ha ezen túl még a LIFO szabályt is be kell tartanunk, akkor egy megengedett



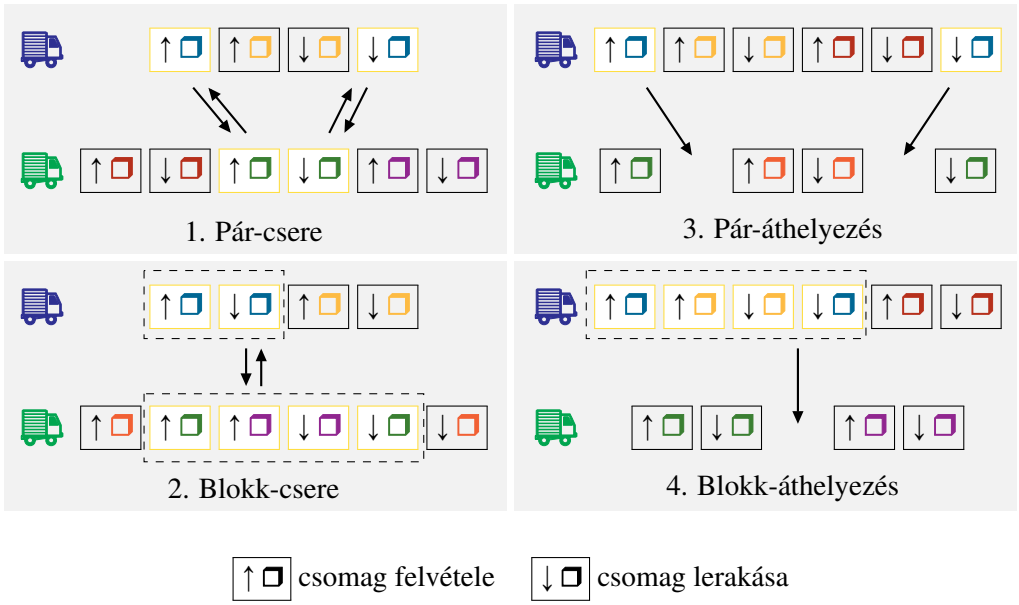
1. ábra. Példa LIFO helyes és LIFO sértő útvonalakra.

megoldásban a járművek útvonalaira egy extra feltétel is érvényes: egy A csomag felvételi pontja pontosan akkor szerepel egy másik B csomag felvételi és lerakási pontja között, ha A lerakási pontja is.

Az általunk alkalmazott operátorok úgy változtatják meg az útvonalakat, hogy az előbbi kitételeket nem sértik meg. Ezek a következők.

1. ***Pár-csere*** (*couple-exchange*)
   Két felvétel-lerakási feladatpár pozícióját (amelyek lehetnek azonos vagy különböző útvonalak részei) megcseréljük, azaz megcseréljük a két felvételi feladatot és megcseréljük a két lerakási feladatot is. Ez a változtatás a LIFO szabályt nyilvánvalóan nem sérti meg.

2. ***Blokk-csere*** (*block-exchange*)
   Egy jármű útvonaltervének egy szakaszát *blokknak* nevezzük, ha a szakasz első és utolsó állomása összetartozó felvétel-lerakási feladatpár. A blokk-csere operátor két diszjunkt blokkot cserél meg (amelyek lehetnek azonos vagy különböző útvonalak részei). Könnyen látható, hogy ha az eredeti



2. ábra. Példa a négy operátor alkalmazására.

útvonalak teljesítették a LIFO szabályt, akkor az új útvanalak is megfelelők lesznek.

3. ***Pár-áthelyezés*** (*relocate-couple*)
   Egy jármű útvonaltervéből egy felvétel-lerakási feladatpárt kiemelünk és egyenként beillesztjük őket egy másik helyre (ami lehet egy másik jármű útvonalterve is). Itt külön figyelni kell rá, hogy a beillesztést LIFO helyesen végezzük el.

4. ***Blokk-áthelyezés*** (*relocate-block*)
   Egy blokkot áthelyezzük egy másik helyre (ami lehet egy másik jármű útvonalterve is). Az új útvonalak itt is mindig megfelelnek a LIFO szabálynak.

Most pedig rátérhetünk a VNS eljárás ismertetésére. A VNS több operátort alkalmaz, amelyek között van egy elsődleges, és egy vagy több másodlagos. Az algoritmus lokális keresést futtat a fő operátorral mindaddig, amíg lokális optimumba nem jut. Ekkor egy iteráció erejéig egy másodlagos operátorra vált és

azzal keres jobb megoldást. Ha talál, akkor újra az elsődleges operátorral folytatja a keresést, ha pedig nem, akkor egy másik másodlagos operátorral próbálkozik. A VNS akkor áll le, ha már egyik operátor sem tud javítani a megoldáson. Ekkor az aktuális megoldás lokális optimum az összes szomszédsági relációra nézve.

Megemlítjük Carrabs et al. [7] munkáját, ahol egy még szofisztikáltabb megoldást alkalmaznak a TSP feladat LIFO korlátos felvétel-lerakási változatára. Noha az általunk vizsgált probléma nem TSP, az idézett cikk kiváló példa a VNS technika alkalmazására LIFO korlátos PDP feladat esetén.

2.2. Dinamikus VRP feladatok

A *dinamikus járműútvonal-tervezési feladatok* (Dynamic Vehicle Routing Problem, DVRP) fő jellemzője, hogy bizonyos információk menet közben érkeznek vagy megváltozhatnak. Az új információkra lehetőség van valós időben reagálni, menet közben is módosíthatjuk a járművek útvonaltervét.

Egy hagyományos, statikus VRP feladatot megoldó algoritmus egy útvonalhalmazt határoz meg, a szállítási feladat pedig ezután a megadott útvonalak szerint végrehajtható. Ezzel szemben egy DVRP esetén a tervezés és a kiszállítás párhuzamosan zajlik, a probléma megoldása pedig egy "szabályzat" – ún. policy –, amely meghatározza, hogy egy adott helyzetben milyen döntést hozzunk. A megoldó algoritmus egy ilyen policyt valósít meg: minden újratervezési helyzetben meghatározza az új útvonalakat.

Dinamikus tényezők a VRP feladat bármely komponensében előfordulhatnak, így az igények, az erőforrások és a (logisztikai) környezet esetében is [26], de ebben a dolgozatban csak a dinamikus igények esetét vizsgáljuk. A dinamikus tényező jellemzően természetes módon adódik a megoldandó gyakorlati probléma sajátosságaiból. Például az *aznapi kiszállítás feladat* (Same-day Delivery Problem, SDDP [34]) esetén, ahol folyamatosan érkező megrendelések teljesítését kell valós időben optimalizálni. Ehhez szorosan kapcsolódik a *dinamikus*

		Az információ minősége szerint	
		Determinisztikus input	Sztochasztikus input
Az információ változása szerint	Az input előre ismert és nem változik	Statikus és determinisztikus	Statikus és sztochasztikus
	Az input változik a feladat során	Dinamikus és determinisztikus	Dinamikus és sztochasztikus

1. táblázat. VRP feladatok osztályozása az információ változása és minősége szerint (Pillac et al. [22] alapján).

felvétel-lerakási feladat (*Dynamic Pickup and Delivery Problem*, DPDP [1]), ahol a rendelések szintén folyamatosan érkeznek, és a tervezés ezzel párhuzamosan zajlik.

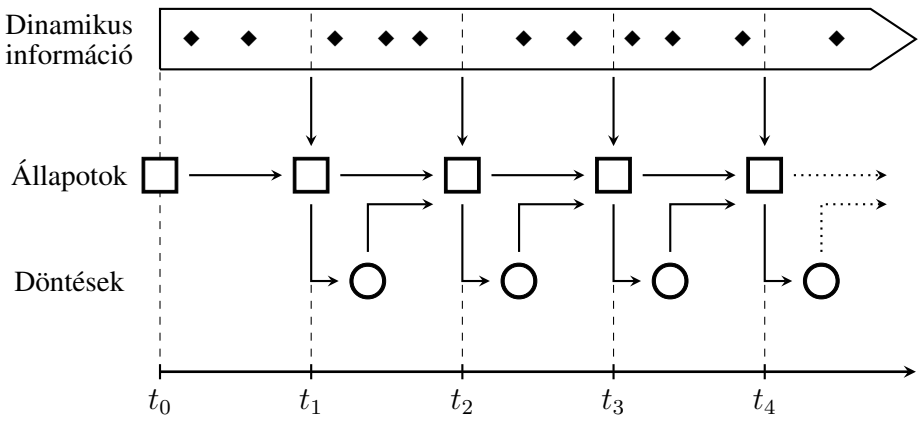
Megjegyezzük, hogy a dinamikus jelleg mellett sztochasztikus tényezők is befolyásolhatják a tervezést. Itt olyan problémákra kell gondolni, amikor a várható történésekről rendelkezünk valamilyen (sztochasztikus) információval. Például Ulmer et al. [31] olyan ételkiszállítási problémát vizsgálnak, ahol az ételek elkészítési ideje véletlenszerű, de ismert eloszlást követ, valamint a megrendelések várható időbeli és földrajzi eloszlása is ismert. Technikailag ez úgy jelenik meg a feladatban, hogy az input egy része valószínűségi változók formájában adott. Azokat a problémákat, ahol a dinamikus és sztochasztikus jelleg egyaránt megjelenik, *sztochasztikus-dinamikus járműútvonal-tervezési feladatnak* (*Stochastic Dynamic Vehicle Routing*, SDVRP) nevezzük.

A fentiek alapján és a szakirodalommal összhangban (vö. Pillac et al. [22]) a VRP feladatokat osztályozhatjuk az 1. táblázatban ábrázolt módon. Az *információ változása* szerint a probléma lehet *statikus* vagy *dinamikus*, az elérhető *információ minősége* szerint pedig megkülönböztetjük a *determinisztikus* és a *sztochasztikus* input esetét. A témakörben való alaposabb elmélyüléshez Pillac et al. [22], Psaraftis et al. [24] és Ulmer [29] összefoglaló cikkeit ajánljuk.

2.2.1. DVRP feladatok modellezése

A DVRP feladatokat *szekvenciális döntési folyamatként* (*Sequential Decision Process*, SDP) modellezzük [26]. Az SDP döntési pontok sorozatából áll, ahol lehetőség van az útvonalak módosítására. A döntési pontokhoz tartozik egy-egy állapot, amelyek a rendszer aktuális helyzetét írják le. A következő állapotot a meghozott döntések és a dinamikus tényezők együttesen határozzák meg.

Megjegyezzük, hogy SDVRP problémák esetén az állapotok közötti átmenet sztochasztikus, és mivel a döntési folyamat rendelkezik a Markov-tulajdonsággal – miszerint az átmenetet a megelőző állapotok nem befolyásolják –, ezért *Markov döntési folyamatról* (*Markov Decision Process*, MDP) is szokás beszélni. Az MDP-n alapuló modellezéssel kapcsolatban remek áttekintést nyújt Ulmer et al. [30] cikke, mi most azonban az általánosabb SDP modellt használjuk. Az alábbiakban áttekintjük a szekvenciális döntési folyamat komponenseit és jellemzőit Soeffker et al. [26] alapján. További, részletesebb forrásként Powell [23] könyvét ajánljuk.



3. ábra. Szekvenciális döntési folyamat.

Döntési pontok

A tervezés folyamán a $k = 1, \ldots, K$ *döntési pontokban* van lehetőség reagálni az új információkra és beavatkozni a kiszállítás folyamatába. A döntési pontokat a következőkben néha azonosítjuk a hozzájuk tartozó időponttal, ilyenkor a $k.$ döntési pontra a t_k időpontként hivatkozunk. A döntési pontok jellemzően periodikusan (adott időközönként) vagy valamilyen eseményhez kötötten merülnek fel, mint például új megrendelés beérkezése vagy egy jármű megérkezése egy lokációra [15]. Az utolsó, K-adik döntési időpont lehet egzaktan adott, feltételesen meghatározott – például addig tart a folyamat, amíg van függőben lévő megrendelés –, vagy valószínűségi változó is.

Dinamikus információ

A dinamikusan változó információkat az Ω *információs modell* írja le. A $k.$ döntési ponthoz tartozó ω_k *realizáció* tartalmazza a t_{k-1} és t_k időpontok között felmerült új információkat. Például dinamikus megrendelések esetén ω_k reprezentálja az új rendeléseket.

Állapotok

Az egyes döntési pontokhoz tartozó s_k *állapotok* tartalmazzák az összes aktuálisan elérhető információt a rendszer helyzetéről: a rendeléseket – beleértve az újonnan beérkezőket is – és azok státuszát, a járművek helyzetét stb. Minden állapot egyben a *döntési modell* egy példánya is, ahol lehetőség van módosítani az útvonalakon.

Döntések

Az x_k *döntés* a döntési modell s_k példányának egy megoldása, például a járművek (újratervezett) útvonalainak halmaza. A döntésnek természetesen megengedett megoldásnak kell lennie, vagyis az útvonalaknak teljesíteniük kell a feladatban szereplő korlátokat.

Nyereség

Az s_k állapotban hozott x_k döntés $R(s_k, x_k)$ *azonnali nyereséget* generál, amely közvetlenül hozzájárul a célfüggvényértékhez. A nyereség lehet például a következő döntési pontig teljesülő kiszállítások értéke. Megjegyezzük, hogy az SDP modellt általában maximalizálási problémaként fogalmazzák meg, de a szokásos módon átfordítható minimalizálási feladatra is, a költségeket negatív nyereségnek tekintve. Ebben az esetben egy tipikus példa a következő döntési pontig felmerülő költségek összege.

Átmenet

Az x_k döntés és az információs modell ω_{k+1} realizációja határozzák meg, hogy milyen állapotba jut a rendszer, vagyis az *átmenetet*. A t_k időponthoz tartozó állapot $s_{k+1} = T(s_k, x_k, \omega_{k+1})$, ahol a T *átmenetfüggvény* írja le a rendszer változását, például a járművek új pozícióját vagy a rendelések státuszának változását.

Célfüggvény

A DVRP feladatokban általában kétféle globális célfüggvény szokott előfordulni: a lehető legtöbb rendelés teljesítése, vagy az összes rendelés teljesítése mellett az összköltség minimalizálása. Az SDP modellben egy s_k állapothoz tartozó döntési feladat célfüggvénye azonban két tagból áll: az $R(s_k, x_k)$ azonnali nyereségből és a jövőbeli nyereségek várható összegéből. A kettő közötti egyensúly kérdése a rövid- és hosszútávú előnyök konfliktusának problémája. Elindítsunk-e minden rendelkezésre álló járművet az aktuális rendelések kézbesítése érdekében, amely gyors teljesítéssel kecsegtet, de lefoglalja az összes erőforrást? Elfogadjunk-e egy nagy értékű rendelést, amit azonban hosszú időbe telik teljesíteni, ha több kis értékű, de gyorsan kiszolgálható rendelés is befuthat a közeljövőben? Az ilyen dilemmák feladatonként külön alapos mérlegelést igényelnek.

Megoldás és optimalitás

Az SDP megoldása egy π *policy*, formálisan egy függvény, amely egy s_k állapothoz az $x_k = \pi(s_k)$ döntést rendeli. A policy tehát meghatározza, hogy a körülményektől függően hogyan módosítsuk a járművek útvonalait. Az SDP célja egy optimális policy meghatározása, ami maximalizálja a nyereségek összegét. Egy π^* optimális policy minden s_k állapotban maximalizálja az $R(s_k, x_k)$ azonnali nyereséget és a jövőbeli nyereségek várható összegét (π^* folyamatos alkalmazása mellett), vagyis kielégíti az ún. *Bellman-egyenletet*:

$$\pi^*(s_k) = \arg\max_{x_k} \left\{ R(s_k, x_k) + \mathbb{E}\left[\sum_{j=k+1}^{K} R\big(s_j, \pi^*(s_j)\big) \,\middle|\, (s_k, x_k) \right] \right\}.$$

Az összeg második tagja a jövőbeli nyereségek várható összege, amelyet a megfelelő állapot-döntés pár *értékének* nevezünk és $V(s_k, x_k)$-val jelöljük. Az összes állapot-döntés pár halmazán értelmezett V *értékfüggvény* ismeretében meghatározható az optimális policy. SDVRP problémák esetében rendelkezünk információval a $V(s_k, x_k)$-ban foglalt várható értékről, a sztochasztikus információt nélkülöző DVRP feladatoknál azonban nem.

Megjegyezzük, hogy a döntési helyzetek általában leírhatók vegyes egészértékű programmal. Ekkor az adott optimalizálási feladat a következő.

$$\max \; R(s_k, x_k) + V(s_k, x_k) \tag{1}$$

$$\text{s.t.} \; A(s_k)x_k \leqslant b(s_k) \tag{2}$$

$$x_k \in \{0, 1\}^{n(s_k)} \tag{3}$$

Az (1) célfüggvény a Bellman-egyenletnek felel meg, a (2) egyenlőtlenség pedig általánosan jelöli az s_k állapottól függő korlátokat, amelyek meghatározzák a megengedett döntések halmazát: az útvonalterveknek meg kell felelniük a feladat megkötéseinek, úgy mint a járművek kapacitása, időablakok betartása, csomag felvétele kézbesítés előtt stb. Az x_k döntési változók írják le a járművek útvonalát és feladatait. Problémától függően tartalmazhatnak valós értékeket is, most az egyszerűség kedvéért a teljes vektort egészértékűnek jelöltük.

2.2.2. Megoldási módszerek

A következőkben Soeffker et al. [26] alapján bemutatunk néhány megközelítést dinamikus VRP feladatok megoldására, amelyeket a szekvenciális döntési folyamatban alkalmazhatunk.

Az első megközelítés a *gördülő tervezés* (*Rolling Horizon Optimization*, RHO) módszer, ahol minden döntési pontban úgy tekintünk a feladatra, hogy már nem lesz változás, és így optimalizálunk. Ehhez az aktuális s_k állapotot statikus VRP feladatnak tekintjük, ahol a célfüggvény az aktuálisan elérhető információk alapján számított jövőbeli *teljes nyereség* (minimalizálási probléma esetén teljes költség), amelyet $R^\star(s_k, x_k)$ jelöl. Például dinamikus igények és periodikus döntési időpontok esetén minden döntési pontban megtervezzük az aktuálisan ismert megrendelések kiszállítását. Ekkor a t_k és t_{k+1} időpontok között az $R^\star(s_k, x_k)$ célfüggvényre optimalizált megoldás szerint zajlik az s_k állapotban ismert megrendelések szállítása, a t_{k+1} és t_{k+2} időpontok között az $R^\star(s_{k+1}, x_{k+1})$ szerint optimalizált útvonalak alapján az s_{k+1} állapotban fennálló megrendelések szállítása, és így tovább.

A gördülő tervezés viszonylag egyszerűen megvalósítható módszer, amellyel kezelhetővé válik a probléma dinamikus jellege. Hátránya viszont, hogy az egyes iterációkban "rövidlátó" döntések születnek, amelyek nincsenek tekintettel a jövőbeli változások lehetőségére. Ez megnehezíti a dinamikus változásokra való rugalmas reagálás lehetőségét és negatívan befolyásolja a globális feladat tekintetében nyújtott teljesítményt.

Más megközelítések feláldozzák az egyes állapotokban ismert információk szerinti optimalitást a flexibilitásért cserébe, amely globálisan jobb hatékonysághoz vezethet. A *policy approximáció* (*Policy Function Approximation*, PFA) módszer esetén előre definiált döntési elvek – jellemzően intuitív stratégiák – kapnak prioritást $R^\star$-gal szemben. Ide tartoznak például az olyan megoldások, amikor az egyes járműveket kijelölt területekre irányítjuk [32], vagy amikor erőforrástartalékolási megfontolásból kötelező várakozást írunk elő a járműveknek bizonyos körülmények esetén [28].

A *költség approximáció* (*Cost Function Approximation*, CFA) módszer a gördülő tervezés hátrányainak kiküszöbölését célozza. Az $R^{\star}(s_k, x_k)$ teljes nyereményfüggvény vagy az $A(s_k)x_k \leqslant b(s_k)$ korlátok célzott módosításával elérjük, hogy az s_k állapothoz tartozó döntési feladatban rugalmas megoldások szülessenek, amelyek hosszú távon, a globális feladat szempontjából előnyösek. Például új korlátok bevezetésével előírjuk, hogy "biztonsági tartalékként" mindig álljon rendelkezésre szabad szállítási kapacitás, vagy a célfüggvény módosításával előnyben részesítjük az olyan megoldásokat, amelyek nem kötnek le egyszerre minden kapacitást [31]. Így közvetett módon érjük el, hogy a megoldás igazodjon a globálisan kritikus szempontokhoz.

SDVRP problémák esetén további megközelítések is ismertek, amelyek felhasználják a rendelkezésre álló sztochasztikus információt is. Ide tartoznak a lehetséges szcenáriókat vizsgáló módszerek, valamint a *megerősítéses tanulás* (*Reinforcement Learning*, RL) különböző válfajai is. RL alapú megoldást javasolnak például a Huawei kutatói (Li et al. [20]) egy a dolgozatunkban vizsgált feladathoz nagyon hasonló DPDP probléma kapcsán.

További kérdés, hogy az adott megközelítésen belül hogyan oldjuk meg az egyes állapotokban fennálló döntési feladatokat. Ezek lokálisan már statikus VRP problémák, azonban – mint korábban kifejtettük – általában nincs lehetőségünk egzakt módon megoldani őket. Különösen, hogy valós idejű tervezés esetén a megfelelő gyorsaság alapfeltétel. Egzakt módszerek helyett ezért heurisztikus megközelítések kerülnek előtérbe, mint a 2.1.1. fejezetben bemutatott lokális keresés.

3. Az ICAPS-DPDP feladat

Ebben a fejezetben bemutatjuk a dolgozat témájául szolgáló feladatot, amely a Huawei Technologies Co. Ltd. valós logisztikai problémáján alapul. A feladat az *International Conference on Automated Planning and Scheduling* (ICAPS) konferencia 2021-ben meghirdetett *The Dynamic Pickup and Delivery Problem* versenyén jelent meg [14, 17].

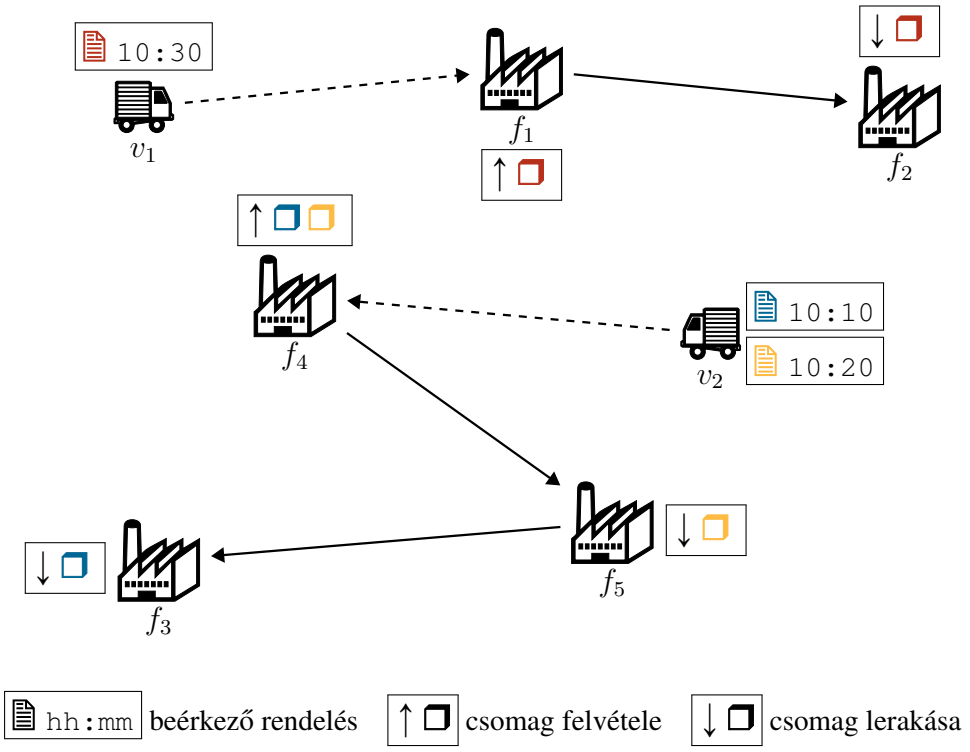
A Huawei az információs és kommunikációs technológiai ipar jelentős szereplőjeként IT- és okoseszközök millióit gyártja le minden évben. A komplex gyártási folyamatok telephelyek, üzemek és gyárak hálózatán zajlanak, ahol kulcsfontosságú az egyes helyszínek közti áruszállítás (alapanyagok, alkatrészek, félkész- és késztermékek szállítása). A vizsgált feladat ennek hatékonyságát célozza: a gyáregységek közötti szállítás folyamatát kell optimalizálni. A következőkben a *gyár*, *állomás* és *lokáció* elnevezéseket használjuk a logisztikai hálózat pontjaira, illetve a *(meg)rendelés* és *csomag* elnevezéseket szinonimaként használjuk.

A probléma formálisan egy dinamikus felvétel-lerakási feladat (DPDP), amelyben a megrendelések dinamikusan érkeznek. Minden megrendelésnek rögzített határideje van, a késést büntetőköltség terheli. A cél a járművek által átlagosan megtett távolság és a késések (az összesített büntetőköltség) minimalizálása. A problémának a szokásos megkötéseken túl része két extra feltétel is: a csomagok fel- és lerakodásának a LIFO szabály szerint kell történnie, valamint a gyárak egyszerre csak korlátozott számú járművet tudnak kiszolgálni, a többi járműnek ilyenkor várakozni kell. A feladatot a dolgozat folyamán az ICAPS-DPDP rövidítéssel fogjuk jelölni.

A most következő 3.1. fejezetben megadjuk a probléma leírását, majd a 3.2. fejezetben felvázoljuk a feladat SDP modelljét, ami formálisan leírja a dinamikus folyamatot. Ezután a 3.3. részben a rendelkezésre álló benchmark eszközöket mutatjuk be. Végül a 3.4. fejezetben röviden ismertetjük a problémára korábban született megoldásokat. A saját munka bemutatása ezt követően, a 4. fejezetben kapott helyet.

3.1. Feladatleírás

A probléma leírását Hao et al. [14] és a verseny honlapja [18] alapján adjuk meg. A feladatot első körben úgy mutatjuk be, mint egy statikus VRP problémát, azzal a kiegészítéssel, hogy a rendelések csak a létrehozás időpontjában válnak ismertté. Ez a dinamikus feladatnak még nem a precíz megfogalmazása, csak a probléma áttekinthetőségét és az általános megértést szolgálja. Megjegyezzük, hogy néhány, pusztán technikai részlet bemutatásától eltekintünk, de a matematikai probléma szempontjából minden fontos komponensre kitérünk.



4. ábra. Egyszerű példa az ICAPS-DPDP feladatra.

Paraméterek

- $\mathbf{D} = (\mathbf{F}, \mathbf{A})$ logisztikai hálózat, mint teljes irányított gráf, ahol $\mathbf{F}$ a gyárak és $\mathbf{A}$ az utak halmaza.
- $\mathbf{F} = \{f_i \mid i = 1, \ldots, N_F\}$ gyárak, és gyáranként a következő paraméterek:
  - n_i^{dock}: dokkok száma,
  - T_i^{dock}: dokkolási idő.
- $\mathbf{A} = \{(f_i, f_j) \mid f_i, f_j \in F\}$ utak, és minden úthoz a következő adatok:
  - $\mathrm{dist}(f_i, f_j)$: távolság,
  - $\mathrm{time}(f_i, f_j)$: utazási idő.
- $\mathbf{O} = \{o_i \mid i = 1, \ldots, N_O\}$ rendelések, $o_i = (f_i^p, f_i^d, q_i, T_i^p, T_i^d, t_i^\alpha, t_i^\omega)$ ahol
  - $f_i^p \in \mathbf{F}$: felvételi pont,
  - $f_i^d \in \mathbf{F}$: lerakási pont,
  - q_i: a csomag mérete,
  - T_i^p: felrakodási idő,
  - T_i^d: lerakodási idő,
  - t_i^α: létrehozás időpontja,
  - t_i^ω: kézbesítési határidő.

  Az o_i rendelés csak a t_i^α időpontban válik ismertté.
- $\mathbf{V} = \{v_i \mid i = 1, \ldots, N_V\}$ járművek, a következő paraméterekkel:
  - c_i: kapacitás,
  - $f_i^{start} \in \mathbf{F}$: kezdőpozíció.

Korlátok

- **Kezdeti állapot**
  Kezdetben minden jármű üres és a kezdőpozíciójának megfelelő lokáción tartózkodik. A feladat végén a járműveknek nem kell visszatérniük a kiindulási pontra.

- **A szállítás folyamata**
  Egy rendelés kiszállításának folyamata a szállító jármű szempontjából, ahol f_i jelöli a felvételi pontot, f_j a lerakási pontot és o a rendelést:
  - érkezés a felvételi pontra,
  - várakozás dokk felszabadulására (lásd: dokk-korlát),
  - dokkolás (T_i^{dock} idő),
  - felrakodás (T_o^p idő),
  - elindulás a felvételi pontról,
  - szállítás (utazási idő),
  - érkezés a lerakási pontra,
  - várakozás dokk felszabadulására (lásd: dokk-korlát),
  - dokkolás (T_j^{dock} idő),
  - lerakodás (T_o^d idő).

  A jármű természetesen több rendelést is felvehet, illetve útközben más gyárakat is meglátogathat. A felvett rendelést azonban csak a kézbesítési ponton rakhatja le, átrakodás nem engedélyezett.

- **Rendelések teljesítése**
  Minden rendelést kézbesíteni kell.

- **Járműkapacitás**
  A járművön lévő csomagok összesített mérete semelyik pillanatban sem haladhatja meg a jármű kapacitását.

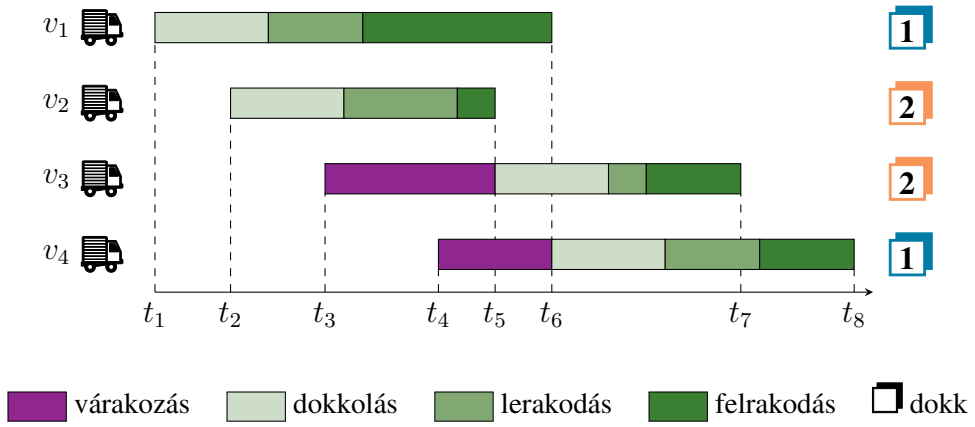
- **Nem osztható rendelések**
  Az egy rendeléshez tartozó árucikkeket együtt kell szállítani. Több csomagra való felosztás csak akkor engedélyezett, ha a rendelés mérete meghaladja a jármű kapacitását.

- **Teljesítési határidő**
  Ha egy rendelést nem sikerül a megadott határidőre teljesíteni, akkor az a késés mértékével arányos büntetőköltséget eredményez. A teljesítés időpontja a szállító jármű lerakási pontra való érkezésének időpontjával egyezik meg, tehát a várakozás, dokkolás és lerakodás már nem számít bele a késésbe. Ha egy rendelést a kapacitáskorlát miatt fel kellett osztani, akkor a rendelés teljesítési ideje az utoljára beérkező tétel érkezési időpontja.

- **LIFO szabály**
  A rakodásnak a LIFO szabály szerint kell történnie: először azt a csomagot kell kézbesíteni, amelyik a legkésőbb került be a raktérbe. Így a csomagok kézbesítésének sorrendje a felvételi sorrend fordítottja lesz.

- **Dokk-korlát**
  Az egyes gyárakban a dokkok száma, ahol a járművek kiszolgálása – felilletve lerakodás – zajlik, korlátozott. Ha az összes dokk megtelt, akkor a soron következő járműnek várakoznia kell, amíg egy dokk fel nem szabadul. A járművek kiszolgálása érkezési sorrendben történik.

Az 5. ábra szemlélteti a dokk-korlátot és a várakozás folyamatát két dokk esetén. A v_1 jármű a t_1 időpontban érkezik, amikor mindkét dokk szabad, így v_1 megkezdi a dokkolást az elsőhöz. A t_2 időpontban érkező v_2 jármű a szabadon maradt második dokkot foglalja el. A v_3 jármű érkezésekor nincs szabad dokk, így a t_3 és t_5 időpontok között várakoznia kell. Amikor a t_5 időpontban befejeződik v_2 kiszolgálása, és elhagyja a kettes dokkot, akkor v_3 megkezdi a dokkolást. A t_4 időpontban érkező v_4 jármű esetén ugyanígy zajlik a folyamat: várakozni kell a t_6 időpontig, amikor is felszabadul az első dokk, és megkezdheti a dokkolást.



5. ábra. Példa a járművek kiszolgálására két dokk esetén.

Célfüggvény

A célfüggvény két tagból áll. Az első a késések miatti büntetőköltség:

$$\mathbf{f}_1 = \sum_{i=1}^{N_O} \max\{0, t_i^c - t_i^\omega\},$$

ahol t_i^c jelöli az o_i rendelés teljesítési időpontját. A második tag a járművek által átlagosan megtett távolság:

$$\mathbf{f}_2 = \frac{1}{N_V} \sum_{i=1}^{N_V} \sum_{j=1}^{\ell_i - 1} \mathrm{dist}(f_i^j, f_i^{j+1}),$$

ahol a v_i jármű által megtett útvonalat a lokációk $\Pi_i = (f_i^1, f_i^2, \ldots, f_i^{\ell_i})$ sorozata jelöli, és ℓ_i az útvonal hossza.

A teljes célfüggvény végül

$$\min \ \mathbf{f} = \lambda \cdot \mathbf{f}_1 + \mathbf{f}_2$$

alakban áll elő, ahol λ egy nagy pozitív konstans. Az optimalizálás tehát elsősorban a késések minimalizálására fókuszál.

Megoldás

A tervezési folyamat célja, hogy a járművekhez úgy rendeljünk útvonalakat és felvétel-lerakási feladatokat, hogy – az előírt korlátok betartása mellett – minden rendelést teljesítsenek, minél kevesebb költséget generálva. A dinamikus feladat megoldásának precíz leírása a folyamat formális megfogalmazását követeli meg, amelyet a következőkben adunk meg.

3.2. SDP modell

Az ICAPS-DPDP feladatot most szekvenciális döntési folyamatként modellezzük Horváth et al. [16] alapján. Mivel minimalizálási problémáról van szó, ezért nyereség helyett költségről fogunk beszélni.

Döntési pontok

A feladat kezdő időpontja $t_0 = 0$, a döntési időpontok pedig periodikusan, vagyis adott időközönként merülnek fel: $t_k = t_{k-1} + \Delta$, ahol $k = 1, \ldots, K$ és Δ rögzített pozitív szám. A továbbiakban a döntési pontokat azonosítjuk a hozzájuk tartozó időponttal. A megrendelések egy véges tervezési idősávon belül érkeznek be, a döntési folyamat azonban ennél tovább is tarthat, amennyiben nem sikerül a végéig minden rendelést teljesíteni. A folyamat akkor ér véget, ha minden rendelést kiszállítottunk.

Dinamikus információ

Az információs modell a menet közben felmerülő megrendeléseket foglalja magában. Az információs modell t_k időpontbeli ω_k realizációja tartalmazza a dinamikus információt, vagyis a t_{k-1} és t_k időpontok között beérkezett rendeléseket: $\omega_k = \{o_i \in \mathbf{O} \mid t_{k-1} < t_i^{\alpha} \leqslant t_k\}$.

Állapotok

Az egyes t_k időpontokhoz tartozó s_k állapotokat a $(t_k, \mathcal{O}_k, \mathcal{V}_k)$ hármas jellemzi, ahol $\mathcal{O}_k$ a nyitott rendelések listája, $\mathcal{V}_k = \{\mathcal{V}_{k,v} \mid v \in \mathbf{V}\}$ pedig a járművek státuszát leíró adathalmaz. Az $\mathcal{O}_k$ lista tartalmazza a ω_k realizáció szerinti új rendeléseket és azokat a korábbi rendeléseket, amelyeket a t_k időpontig még nem vett fel egyik jármű sem. A v járműre vonatkozó dinamikus információkat a $\mathcal{V}_{k,v} = (\Phi_{k,v}, \mathcal{L}_{k,v}, \theta_{k,v})$ hármas írja le, amelyet a következő pontokban részletezünk.

- $\Phi_{k,v} = (f^{curr}_{k,v}, t^{leave}_{k,v})$ a jármű pozíciója. Ha a jármű állomáson tartózkodik, akkor $f^{curr}_{k,v}$ az aktuális gyár, $t^{leave}_{k,v}$ pedig a legkorábbi lehetséges elindulás időpontja. Ha a jármű úton van, akkor $\Phi^{curr}_{k,v} = \varnothing$.

- $\mathcal{L}_{k,v}$ a járművön szállított csomagok listája, amely a berakodás sorrendje szerint rendezett. Ha $\Phi_{k,v} \neq \varnothing$, azaz a jármű állomáson tartózkodik, akkor az $\mathcal{L}_{k,v}$ listán már nem szerepelnek azok a csomagok, amelyeket a jármű az $f^{curr}_{k,v}$ állomásra szállított, de tartalmazza azokat a csomagokat, amelyeket a járműnek ott kell felvennie.

- $\theta_{k,v}$ jelöli a jármű útvonaltervét. Az útvonalterv állomások sorozatából áll, ahol egy állomást és a jármű ottani tevékenységét egy öt elemű vektor ír le. A j. állomás esetében $f_{k,v,j}$ jelöli a megfelelő gyárat, $t^{arr}_{k,v,j}$ az érkezési idő és $t^{dep}_{k,v,j}$ az állomás elhagyásának időpontja, továbbá $\mathcal{D}_{k,v,j}$ az adott gyárba kézbesítendő csomagok listája, $\mathcal{P}_{k,v,j}$ pedig azon csomagok listája, amelyeket a járműnek fel kell vennie. Az útvonal első állomása, vagyis $f_{k,v,1}$ a jármű *célállomása*, $\ell_{k,v}$ pedig a jármű útvonalának hossza. Az útvonalterv ezek alapján

$$\theta_{k,v} = \big((f_{k,v,j}, t^{arr}_{k,v,j}, t^{dep}_{k,v,j}, \mathcal{D}_{k,v,j}, \mathcal{P}_{k,v,j}) \mid j = 1, \ldots, \ell_{k,v}\big).$$

  Az útvonalaknak mindig megengedettnek kell lenniük, azaz meg kell felelniük az 3.1. fejezetben részletezett korlátoknak: a kapacitáskorlátnak, a

rendelések nem oszthatóságának és a LIFO szabálynak, továbbá konzisztensnek kell lenniük a szállítás folyamatával és a dokk-korláttal is. Emellett a járművek útvonalterveinek együttesen is konzisztensnek kell lenniük – például egy adott csomag felvétele nem szerepelhet egynél több jármű útvonaltervében.

A kezdeti s_0 állapotban $\mathcal{L}_{k,v}$ és $\theta_{k,v}$ minden jármű esetén üres, $f_{k,v}^{curr}$ rendelkezik értékkel és $t_{k,v}^{leave} = t_0$. Azaz a járművek gyár lokációján tartózkodnak és azonnal el tudnak indulni.

Döntések

Döntéseket a $t_1, \ldots, t_K$ időpontokban lehet hozni. Egy $x_k = (\theta_{k,v}^{\star} \mid v \in V)$ döntés megengedett útvonaltervek halmaza. A korábban részletezett korlátokon túl még egy megkötés szerepel a feladatban: ha egy járműnek már van célállomása, akkor azt nem lehet megváltoztatni, vagyis ha $\theta_{k,v} \neq \varnothing$, akkor $f_{k,v,1}^{\star} = f_{k,v,1}$. Továbbá ebben az esetben a $\mathcal{D}_{k,v,1}$ és $\mathcal{D}_{k,v,1}^{\star}$ listáknak is meg kell egyezniük, azonban a $\mathcal{P}_{k,v,1}$ és $\mathcal{P}_{k,v,1}^{\star}$ listák különbözhetnek.

Költség

A költséget az $R(s_k, x_k)$ azonnali költség helyett az $R^{\star}(s_k, x_k)$ teljes költséggel írjuk le, vagyis amit az s_k állapot statikus feladatként való értelmezése esetén az x_k döntés szerinti útvonalak alapján kapnánk. A 3.1. fejezetben ismertetett célfüggvény egyes tagjait most az SDP jelöléseivel írjuk fel. Jelölje $\mathbf{f}_1$ az útvonaltervek szerint számolt késések összesített idejét, azaz

$$\mathbf{f}_1(s_k, x_k) = \sum_{v \in \mathbf{V}} \sum_{j=1}^{\ell_{k,v}^{\star}} \sum_{o_i \in \mathcal{D}_{k,v,j}^{\star}} \max\{0, t_{k,v,j}^{\star arr} - t_i^{\omega}\}.$$

Továbbá jelölje $\mathbf{f}_2$ a járművek által megtett átlagos távolságot, azaz

$$\mathbf{f}_2(s_k, x_k) = \sum_{v \in \mathbf{V}} \text{dist}(f_{k,v}^{curr}, f_{k,v,1}^{\star}) + \sum_{v \in \mathbf{V}} \sum_{j=2}^{\ell_{k,v}^{\star}} \text{dist}(f_{k,v,j-1}^{\star}, f_{k,v,j}^{\star}),$$

ahol dist$(f^{curr}_{k,v}, f^{\star}_{k,v,1}) = 0$, ha $\Phi_{k,v} = \varnothing$ vagy $\ell^{\star}_{k,v} = 0$. Az $R^{\star}$ teljes költségfüggvény így

$$R^{\star}(s_k, x_k) = \lambda \cdot \mathbf{f}_1(s_k, x_k) + \mathbf{f}_2(s_k, x_k).$$

Átmenet

Az x_k döntés kiválasztása után a döntési folyamat a következő, s_{k+1} állapotba kerül. A járművek és rendelések státusza az s_k állapot, az x_k döntés és a dinamikus információ ω_{k+1} realizációja szerint változik meg.

Tekintsük először a járműveket. A $(\Phi_{k+1,v}, \mathcal{L}_{k+1,v}, \theta_{k+1,v})$ hármas értékeit a t_k és t_{k+1} közti időköz történései határozzák meg.

- Ha egy jármű megérkezett a célállomására, akkor $f^{curr}_{k+1,v} = f^{\star}_{k,v,1}$ és $t^{leave}_{k+1,v} = t^{\star dep}_{k,v,1}$ lesz, a járművön szállított csomagok listájáról pedig lekerülnek $\mathcal{D}^{\star}_{k,v,1}$ elemei, míg $\mathcal{P}^{\star}_{k,v,1}$ elemei felkerülnek rá. Ha $\ell^{\star}_{k,v} \geqslant 2$, akkor a jármű új útvonaltervét $\theta^{\star}_{k,v}$-ból kapjuk az első elem eltávolításával, célállomása pedig $f^{\star}_{k,v,2}$ lesz. Ellenkező esetben $\theta_{k+1,v} = \varnothing$.

- Ha egy jármű elhagyta a gyárat, ahol tartózkodott, akkor $\Phi_{k+1,v} = \varnothing$, útvonalterve $\theta_{k+1,v} = \theta^{\star}_{k,v}$ lesz, a szállított csomagok listája pedig nem változik. Ha a jármű végzett az adott lokáción, de nincs következő állomása, akkor $f^{curr}_{k+1,v} = f^{curr}_{k,v}$ és $t^{leave}_{k,v} = t_{k+1}$, az útvonalterve pedig üres lesz.

- Ha a jármű állomáson tartózkodott és a kiszolgálása még nem fejeződött be, vagy úton volt és még nem érkezett meg, akkor az x_k döntésnek megfelelően $\theta_{k+1,v} = \theta^{\star}_{k,v}$ lesz, más nem változik.

- Ha az eltelt időközben egy jármű több eseményben is érintett volt, akkor értelemszerűen az egymás utáni változások végeredménye lesz az új státusz.

A rendelésekre térve, ezek státusza ω_{k+1} és az előbbi események szerint változik. A nyitott rendelések listája ω_{k+1} elemeivel frissül, viszont lekerülnek róla azok a csomagok, amelyeket az eltelt időszakban valamelyik jármű felvett, így

kapjuk a $\mathcal{O}_{k+1}$ listát. Ha $\mathcal{O}_{k+1} = \varnothing$ és a járműveken sincs már csomag, akkor minden rendelés kiszállításra került és az SDP véget ér.

Célfüggvény

Jelölje θ_v a v jármű tényleges útvonalát, azaz a jármű által az SDP folyamán végigjárt állomások és teljesített feladatok sorozatát. A θ_v útvonalak megengedettek, együttesen konzisztensek és kiszolgálják az összes rendelést, vagyis $x = (\theta_v \mid v \in V)$ megoldja a szállítási feladatot. A megoldást a

$$C(x) = \lambda \cdot \mathbf{f}_1(s_0, x) + \mathbf{f}_2(s_0, x)$$

globális költségfüggvény alapján értékeljük ki, azaz kiszámítjuk a θ_v útvonalak költségét az s_0 állapot szerint.

Megoldás

Az SDP megoldása egy π policy, továbbá olyan policyt keresünk, amely minimalizálja a $C(x)$ költségfüggvényt. Mivel a dinamikus változásokról nem rendelkezünk előzetes információval, így a célunk optimális megoldás helyett minél jobb policy meghatározása lesz. Ennek a lehetőségeit vizsgáljuk a következő fejezetekben.

3.3. Benchmark eszközök

A verseny szervezői egy benchmark eszköztárat is biztosítottak ahhoz, hogy az ICAPS-DPDP feladatra született megoldások összehasonlíthatók legyenek [14]. Ez két komponensből áll: tesztadatokból és egy szimulátorból.

A tesztadatok valós adatokon alapuló feladatpéldányok, amelyeket a Huawei biztosított. A gyárak és az úthálózat minden esetben ugyanazok, a többi input adat változó. A 3.1. fejezet jelöléseivel élve tehát $\mathbf{D} = (\mathbf{F}, \mathbf{A})$ rögzített, $\mathbf{O}$ és $\mathbf{V}$ változó. Egy feladatpéldány tartalmazza rendeléseket, a járműveket, illetve ezek jellemzőit, többek között az egyes rendelések létrehozásának időpontját.

Csoport	Feladatpéldányok	Rendelések száma	Járművek száma
1	1 – 8	50	5
2	9 – 16	100	5
3	17 – 24	300	20
4	25 – 32	500	20
5	33 – 40	1000	50
6	41 – 48	2000	50
7	49 – 56	3000	100
8	57 – 64	4000	100

2. táblázat. A tesztcsoportok és jellemzőik.

A probléma paraméterei közül is néhány az összes tesztpéldányon átívelően rögzített: a dokkok n_f^{dock} száma minden f gyár esetén 6, a T_i^{dock} dokkolási idő egységesen 1800 (vagyis 30 perc), és minden v jármű kapacitása $c_v = 15$. Továbbá a tervezési horizont minden esetben egy teljes nap, a rendelések 00:00:00 és 23:59:59 között érkeznek be, és a teljesítésre a létrehozástól számítva mindig 4 óra áll rendelkezésre.

Az összesen 64 tesztpéldány 8 csoportra van osztva, a probléma komplexitása csoportról-csoportra növekszik. A komplexitás esetünkben a feladat méretével arányos, amelyet a rendelések és járművek száma határoz meg. A 2. táblázatban feltüntettük ezeket jellemzőket a 8 tesztcsoport esetében.

A szimulátor a 3.2. fejezetben bemutatott szekvenciális döntési folyamatot valósítja meg: meghatározza az állapotokat és az iterációnként felmerülő dinamikus információt; szimulálja a döntéseket és az átmeneteket; ellenőrzi a tervezési korlátokat és kiértékeli a megoldást. A feladat megoldása céljából a szimulátor tartalmaz egy demó algoritmust, illetve a felhasználó által – C++, Java vagy Python nyelven – írt megoldó algoritmus is hozzácsatolható. A szimulátor így objektív biztosítéka annak, hogy a felhasználó algoritmusa valóban megoldja a feladatot, és a megoldás független kiértékelését is biztosítja.

A szimulátor alapbeállításai szerint $\Delta = 600$, vagyis 10 perces iterációkat szimulál, és $\lambda = 10000/3600$, azaz a késést óránként 10000 egység büntetőköltség terheli. Ezek a beállítások, bár elvben megváltoztathatók, a benchmark részét képezik. A későbbiekben bemutatásra kerülő eredmények egységesen ezen beállítások mellett születtek.

3.4. Ismert megoldások

Ebben a fejezetben röviden tárgyaljuk azokat az ismert megoldásokat, amelyek az ICAPS-DPDP feladatra korábban születtek. Legelőször is a versenyen díjazott első három helyezett csapat algoritmusait mutatjuk be a verseny honlapján [19] elérhető leírás és prezentáció alapján. Közülük az első és a harmadik csapat később továbbfejlesztette a munkáját – ezt is bemutatjuk röviden. Végül két további cikket foglalunk össze. A megoldásokat a 2.2.2. fejezetben bemutatott módszerek szerint osztályozzuk.

A győztes csapat megoldása kevert RHO és PFA megközelítésen alapul. Első lépésben az új rendelések egymás után beillesztésre kerülnek valamely jármű útvonalába aszerint, hogy a döntés pillanatában melyik pozíció jár a legkevesebb költséggel. Az így kialakult kezdeti útvonalakat második lépésben egy VNS eljárás javítja, amely a lépésenkénti javításhoz négy operátort használ – amelyeket mi is bemutattunk a 2.1.1. fejezetben. Ezenkívül egy további operátor bizonyos feltételek teljesülése esetén perturbációt hajt végre az inkumbens megoldáson. Utolsó lépésként egy várakozási stratégiát alkalmaznak: a teljes útvonaltervből csak azokat az útvonalakat adják ki ténylegesen, amelyek tartalmaznak – egy meghatározott kritérium szerint – sürgős rendelést. A következő iterációban azonban a kiszámított teljes útvonaltervből indul ki az algoritmus, így az első lépésnek valóban csak az új rendelések beillesztését kell elvégeznie.

Az algoritmus némileg továbbfejlesztett változatát részletesen bemutatják egy cikkben, amelyet a csapat tagjai jegyeznek: Cai et al. [2]. A szerzők a [3, 4] cikkekben egy másik, még jobb megoldást írnak le, ahol evolúciós algoritmust és

dekompozíciós heurisztikákat használnak, melyeket egy VNS eljárás egészít ki. Később még tovább javítottak a számítási eredményeiken az [5] cikkben bemutatott algoritmussal, ahol a VNS eljárást tabu kereséssel (*tabu search*) ötvözik.

A második csapat PFA megközelítést alkalmaz, ahol egy beillesztéses eljárás kap kiemelt szerepet. Az algoritmus a DPDP feladatból egy speciális hátizsák-feladatot készít és ennek segítségével csoportosítja a rendeléseket. A csoportokat ezután járművekhez rendeli figyelembe véve a rendelések sürgősségét és a járművek kihasználtságát. A fő elv, hogy egy jármű csak akkor induljon el, ha sürgős rendelést teljesít, vagy ha a kihasználtsága elér egy bizonyos korlátot. Az algoritmus emellett figyelembe veszi a megrendelések területi eloszlását is és speciálisan kezeli a sokat látogatott lokációkat.

A bronzérmes csapat CFA típusú módszert alkalmaz. Minden iterációban a kezdeti útvonaltervet egy beillesztéses eljárás alakítja ki, ahol először a sürgős rendelések kerülnek beillesztésre valamely jármű útvonalába, majd ezt követően szállítási idő szerint csökkenő sorrendben a maradék. Végül egy speciális szomszédsági reláción alapuló lokális keresés javítja a megoldást. Az algoritmus sajátossága, hogy mind a kezdeti, mind pedig a lokális keresést alkalmazó fázisban módosított költségfüggvény segítségével értékeli ki az útvonalakat. Az új célfüggvény

$$\mathbf{f}' = \lambda \cdot \mathbf{f}_1 + \mathbf{f}_2 + \lambda \cdot \mathbf{f}_3, \text{ ahol}$$

$$\mathbf{f}_3 = \sum_{f=1}^{N_F} T_f^{dock} \cdot \max\{0, s_f - (n_f^{dock} + 3)\}^2, \text{ ahol}$$

s_f az f gyárat meglátogató járművek száma az aktuális útvonaltervek szerint. A módosítás célja, hogy egy-egy gyárat ne látogasson túl sok jármű, elkerülendő a torlódást és az abból fakadó kényszerű várakozást. A módosított célfüggvény előnyben részesíti az olyan útvonalterveket, ahol egy gyárat legfeljebb hárommal több jármű látogat, mint a dokkok száma.

A csapat tagjai később egy másik, szintén CFA megközelítésen alapuló megoldást is adtak a problémára (Horváth et al. [16]), ahol az előbbi $\mathbf{f_3}$ helyett két másik

komponenst adtak hozzá a költségfüggvényhez. Ezek közül a nagyobb jelentőségű komponens explicit módon bünteti a várakozással töltött időt, azaz amikor a jármű dokk felszabadulására vár. A másik, kevésbé meghatározó komponens a szabadon álló, feladatot nem végző járművek számával arányos büntetőköltséget ad a célfüggvényhez. Az egyes iterációkban a kezdeti megoldást itt egy VNS eljárás javítja, ahol a blokk-áthelyezés és blokk-csere operátorok mellett egy új operátort is bevezetnek. Az ismert megoldások közül ez az algoritmus adja átlagosan a legjobb eredményeket az ICAPS-DPDP közepes és nagy feladatpéldányain (5-8. tesztcsoportok).

Egy további megoldást prezentál Du et al. [11] cikke, ahol PFA módszert használnak. Itt heurisztikus eljárások alapján rendelik a csomagokat járművekhez, majd az egyes járművek útvonalait (az állomások meglátogatásának sorrendjét) lokális kereséssel optimalizálják. Ez a munka a második helyezett csapat megoldásával mutat hasonlóságokat: a heurisztikák intuitív stratégiákon alapulnak és külön kezelik a sokat látogatott lokációkat.

Végül megemlítjük Zhou et al. [35] munkáját, amely RHO megközelítésen alapul. Itt az egyes iterációk feladatpéldányait memetikus (*memetic*) algoritmus segítségével oldják meg, amely egy genetikus (*genetic*) algoritmus és lokális keresés kombinációja. Az algoritmus értékelését azonban megnehezíti, hogy a szerzők valójában nem pontosan az ICAPS-DPDP feladatot vizsgálták, hanem annak egy relaxált változatát, amelyben nincs dokk-korlát.

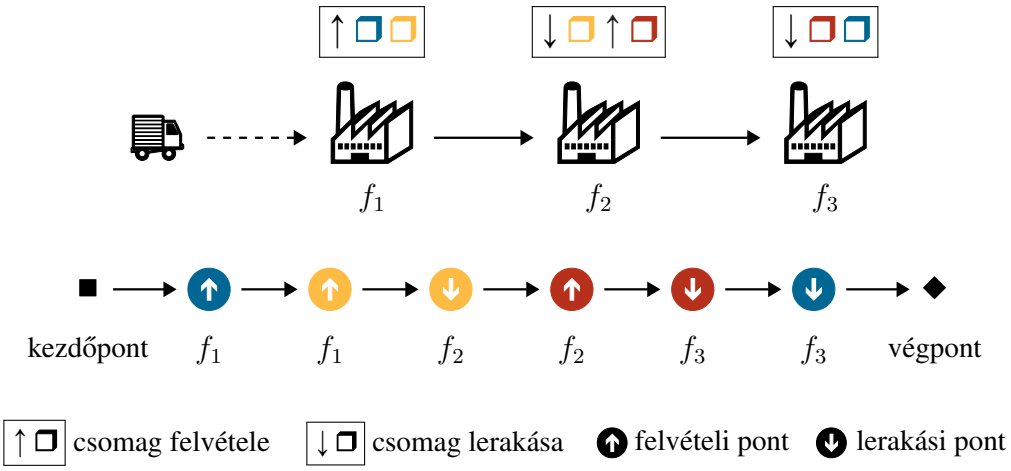
4. Megoldások az ICAPS-DPDP feladatra

A következőkben bemutatjuk a problémára adott saját megoldásokat. Először a technikai háttér meghatározó elemeit ismertetjük, majd ezután következnek az egyes algoritmusok. A tárgyalt megoldások mindegyike a 2.2.2. fejezetben bemutatott gördülő tervezés megközelítést (RHO) alkalmazza, vagyis az egyes iterációkban az $R^\star$ teljes költség szerint optimalizálunk.

Elsőként a szimulátorhoz csatolt *demó* algoritmust írom le, valamint ennek egy okosabb változatát, a *naív* algoritmust. Ezek egyszerű mohó megközelítésen alapulnak és elsősorban referenciaszerepük van: összehasonlítási alapként szolgálnak a többi módszerhez. A következő megoldás a *beillesztő* algoritmus, amely az új rendeléseket egymás után illeszti be valamely jármű útvonalába, az aktuálisan legjobb pozícióba. Végül pedig a *VNS* algoritmus kerül bemutatásra, amely a beillesztő algoritmus eredményéből indul ki, majd lokális kereséssel javítja a megoldást. A fejezet legvégén pedig összehasonlítjuk az egyes módszerek számítási eredményeit a verseny első három helyezettjének megoldásaival.

4.1. Technikai háttér

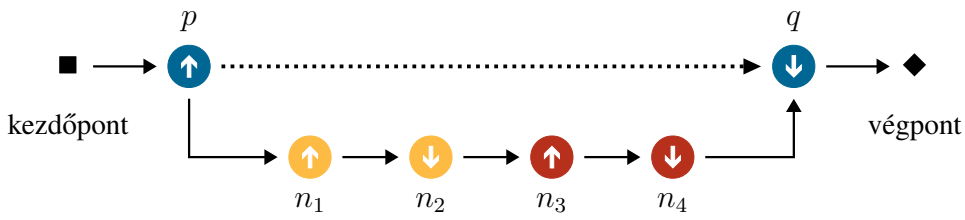
Ebben a részben tárgyaljuk a fejezetben bemutatott algoritmusok implementációjának fontosabb technikai részleteit. Az első lényeges elem az útvonalak reprezentációja, amelyet a programkódban kétszeresen láncolt listával valósítunk meg. A lista elemei egységnyi felvételi vagy lerakási feladatok, azaz egyetlen rendelés felvétele vagy kézbesítése. Így egy jármű adott lokációhoz tartozó különböző feladatait különböző pontok reprezentálják. Ez a struktúra lehetővé teszi az útvonalak gyors és egyszerű módosítását, azaz részútvonalak kivágását és beillesztését. Ez később kulcsfontosságú lesz a lokális keresés hatékonysága szempontjából. Az algoritmusok végén a megoldásként kapott útvonalakat visszaalakítjuk a szimulátor által megkövetelt formára: összevonjuk a szomszédos feladatokat, ha közös a lokációjuk.



6. ábra. Egy példa útvonalterv és láncolt listás változata.
(Az egyszerűség kedvéért csak a menetiránnyal megegyező irányú pointert tüntettük fel.)

A láncolt listás reprezentáció egy szemléltető példáját illusztrálja a 6. ábra. A kezdőpont és a végpont nem valódi állomást jelölnek, hanem speciális pontok, amelyek megkönnyítik a láncolt listák kezelését és az algoritmusok implementációját. Az egyes felvételi és lerakási feladatok a láncolt lista egy-egy belső pontjának felelnek meg.

A 7. ábra pedig egy útvonal módosítását szemlélteti ebben a speciális struktúrában. Egy $n_1, n_2, \ldots, n_k$ pontsorozat (speciálisan, ha $k = 1$, akkor egyetlen pont) beillesztése az p és q pontok közé: az utóbbi két pont egymásra mutató pointerei helyett az p és n_1, valamint az n_k és q pontokat kapcsoljuk össze. Egy pontsorozat eltávolítása pedig értelemszerűen az előbbi műveletek fordítottjaként valósul meg. A kivágás és beillesztés ezzel a megoldással gyorsabb és könnyebben implementálható, mintha az útvonalakat egyszerű listákban tárolnánk.



7. ábra. Útvonal módosítása a láncolt lista struktúrában. Ebben a példában $k = 4$. (Az egyszerűség kedvéért csak a menetiránnyal megegyező irányú pointert tüntettük fel.)

1. Algoritmus Kiértékelés

/* Inicializálás */
Kezdetben $R^\star := 0$, valamint Q és $L(f)$ minden f gyárra üres.
for all v úton lévő jármű **do**
    Beillesztünk egy érkezési eseményt Q-ba az érkezési időponttal.
for all v állomáson tartózkodó jármű **do**
    if v kiszolgálásra vár vagy kiszolgálása folyamatban van **then**
        Kiszámítjuk az indulási időpontot és beillesztjük v-t az $L(f_v)$ listába.
        Beillesztjük Q-ba az indulási eseményt.
/* Események feldolgozása */
while $Q \neq \varnothing$ **do**
    Kivesszük Q-ból az első e eseményt.
    if e indulási esemény **then**
        Eltávolítjuk e-t az $L(f_e)$ listából.
        if a v_e járműnek van következő állomása **then**
            Kiszámítjuk az érkezési időpontot és beillesztjük Q-ba az új eseményt.
    if e érkezési esemény **then**
        $R^\star := R^\star + c_e$, ahol c_e az előző állomás távolságából és a kézbesített rendelések késéséből számított költségösszeg.
        Kiszámítjuk az indulási időpontot és beillesztjük v_e-t az $L(f_e)$ listába.
        Beillesztjük Q-ba az indulási eseményt.
Output: $R^\star$

A másik fontos technikai részlet az 1. algoritmusban bemutatott kiértékelési eljárás, amely adott megoldás esetén meghatározza az útvonalterv $R^\star$ teljes költségét. A kiértékelés események szimulációján alapszik, ahol egy esemény egy jármű állomásról való elindulása vagy megérkezése. Ehhez fenntartunk egy Q globális eseménylistát, amelyet elsőbbségi sor adatszerkezetben tárolunk, ahol a prioritást az események időpontja határozza meg. Q elemei $e = (y_e, t_e, f_e, v_e)$ alakú események, ahol y_e jelöli az esemény típusát (érkezési vagy indulási esemény), t_e az esemény időpontja, f_e a kapcsolódó gyár és v_e a kapcsolódó jármű.

Számon tartunk továbbá minden f gyárra egy $L(f)$ foglalási listát, ahol az adott gyárban tartózkodó járművek elindulási időpont szerint növekvő sorrendben szerepelnek. Ez alapján tudjuk kiszámolni, hogy egy új jármű mikor kezdheti meg a dokkolást, illetve ennek megfelelően mikor fog végezni. Egy jármű elindulási időpontjában – amikor végzett a gyárhoz kapcsolódó feladataival –, az általa használt dokk felszabadul.

4.2. Megjegyzés az algoritmusokhoz

A feladat egyes iterációiban az aktuális megoldás a járművek útvonalterveit tartalmazza, amelyeknek teljesíteniük kell a 3.1. fejezetben bemutatott korlátokat, valamint a célállomásra vonatkozó – a 3.2. fejezet döntéseket leíró részében ismertetett – korlátot. Ennek megfelelően az algoritmusoknak az útvonalak meghatározása során biztosítaniuk kell

- a rendelések felosztásával kapcsolatos korlátok betartását,
- a célállomásra vonatkozó feltétel teljesülését,
- a kapacitáskorlát betartását,
- a LIFO szabály betartását,
- a rendelések teljesítését.

A rendelések több csomagra való felosztása csak akkor engedélyezett, ha a rendelés mérete meghaladja a jármű kapacitását – ekkor viszont szükségszerű. A fejezetben bemutatott algoritmusok mindegyike elvégzi ezt a felosztást, amennyiben indokolt. Az egyszerűség kedvéért a leírásban ezt külön nem részletezzük, és a továbbiakban *rendelés*, *csomag*, illetve *feladat* alatt egyaránt olyan felvételi vagy lerakási feladatot fogunk érteni, ami már megfelelő méretű csomagra vonatkozik.

A célállomásra érvényes megkötés, a kapacitáskorlát és a LIFO szabály közül mindig ellenőrizzük azokat, amelyeket az adott algoritmus valamely lépése megsérthet, a pszeudokódban azonban ezt sem fogjuk az ellenőrzés tényénél jobban részletezni. A rendelések teljesítésének is eleget fog tenni minden algoritmusunk, mert az összes rendelést kiosztjuk valamelyik járműnek. (Továbbá a szimulátor sem áll le, amíg van függőben lévő rendelés.)

A feladatban szerepelő többi korlát teljesülését pedig a szimulátor automatikusan biztosítja: a szállítás folyamatát a feladatleírásnak megfelelően modellezi, betartja a dokk-korlátot, és a kézbesítési határidőt nem teljesítő rendelések késését is kiszámítja.

4.3. A demó és naív algoritmusok

A szimulátorhoz csatolt *demó* algoritmus (2. algoritmus) esetében a járművek egyesével kapnak felvételi-lerakási feladatokat. Az üres jármű elmegy a felvételi pontra, felveszi a hozzá rendelt egyetlen csomagot, elszállítja a lerakási pontra, majd megy tovább a következő neki kiosztott rendelésért.

Az egyes iterációkban a járművek aktuális célállomásainak megfelelő feladatok képezik a kezdeti útvonalhalmazt. Ha egy jármű célállomása lerakási pont, akkor az lesz a kezdeti terv része. Ha felvételi pont, akkor az a megfelelő lerakási ponttal együtt lesz az útvonal része. Ezután a többi rendelés járművek közti kiosztása ciklikusan történik.

2. Algoritmus Demó

1: Legyen S az aktuális célállomások feladataiból álló útvonalhalmaz.
2: **for all** r nem allokált rendelés **do**
3: Legyen i az r sorszáma az új rendelések között.
4: Legyen k a járművek száma.
5: Legyen v az $(i - 1 \bmod k) + 1$ sorszámú jármű.
6: Módosítjuk S-et: v útvonalának végére tesszük r felvételét és lerakását.
7: Output: S

A demó algoritmus megengedett megoldást ad, mert kiszállít minden rendelést, nem sérti meg a kapacitáskorlátot, sem a LIFO szabályt (hiszen egy jármű egyszerre csak egy rendelést szállít), és úton lévő jármű célállomását sem módosítja. Azonban semmilyen módon nem veszi figyelembe a célfüggvényt, ami nagy mértékű késésekhez, így nagyon rossz globális célfüggvényértékhez fog vezetni.

A *naív* algoritmus (3. algoritmus) esetében az előző iteráció megoldása lesz a kezdeti útvonalhalmaz. A demó algoritmushoz hasonlóan a naív is mohó elven működik, de már a célfüggvényt is figyelembe veszi. Tudva, hogy egy jó megoldáshoz a késéseket kell leszorítani, ez az algoritmus az új feladatokat úgy osztja ki, hogy a soron következő rendelést mindig ahhoz a járműhöz allokálja, amelyik a meglévő feladatai végeztével a leghamarabb tud odaérni a felvételi pontra.

3. Algoritmus Naív

1: Legyen S az inkumbens útvonalak szerinti megoldás.
2: **for all** r új rendelés **do**
3: Legyen t_v a v jármű utolsó állomásának befejezési időpontja.
4: Legyen f_v a v jármű utolsó állomásának lokációja.
5: Legyen f_r^p az r rendelés felvételi pontja.
6: $v^* := \arg\min_v \left\{ t_v + \text{time}(f_v, f_r^p) \right\}$
7: Módosítjuk S-et: v^* útvonalának végére tesszük r felvételét és lerakását.
8: Output: S

4.4. A beillesztő algoritmus

A 4. algoritmus az előző iteráció által kiadott útvonalak alapján építi fel a kezdeti megoldást. Ezután az új rendeléseket egymás után beilleszti járművek útvonaltervébe oly módon, hogy minden lépésben minimalizálja a keletkező terv költségét. Ehhez a soron következő rendelésnél végigpróbálja az összes lehetséges beillesztést és kiválasztja közülük a legjobbat.

Ebben az algoritmusban már ellenőrizni kell a tervezési korlátokat, mégpedig minden vizsgált beillesztésnél: jármű célállomása nem változhat, valamint be kell tartani a járművek kapacitáskorlátját és a LIFO szabályt. Továbbá itt már alkalmaznunk kell az 1. algoritmusban bemutatott eseményalapú kiértékelési eljárást is a beillesztési opciók összehasonlításához. Az algoritmus leírásában S_r^* az r rendelés beillesztésével kapott aktuálisan legjobb megoldást jelöli.

4. Algoritmus Beillesztő

1: Legyen S az inkumbens útvonalak szerinti megoldás.
2: **for all** r új rendelés **do**
3: $S_r^* := \varnothing$
4: **for all** v jármű és p_1, p_2 lehetséges beillesztési pozíciók **do**
5: Legyen S' az a megoldás, amit S-ből kapunk, ha az r rendelés felvételét és lerakását v útvonalának p_1 és p_2 pozíciójába illesztjük be.
6: **if** az S' megoldásban v útvonala megengedett **then**
7: **if** S' jobb mint S_r^* vagy $S_r^* = \varnothing$ **then**
8: $S_r^* := S'$
9: $S := S_r^*$
10: Output: S

4.5. A VNS algoritmus

Ebben az algoritmusban a 2.1.1. fejezetben leírt VNS eljárást alkalmazzuk. A 5. algoritmusban a kezdeti megoldás a beillesztő algoritmus outputja, a lokális keresés pedig a négy korábban bemutatott operátort használja: pár-csere, blokk-csere, pár-áthelyezés, blokk-áthelyezés. A pár-áthelyezés esetében az alternatív megoldások keresésénél ellenőrizni kell a célállomást, a kapacitáskorlátot és a LIFO szabályt is. A másik három operátor automatikusan LIFO-helyes útvonalat ad, így ezeknél csak a célállomás és a kapacitáskorlát ellenőrzése szükséges.

5. Algoritmus VNS

1: Legyen S a beillesztő algoritmus outputja.
2: **while** a leállási feltétel nem teljesül **do**
3: Legyen S' a *blokk-áthelyezés* operátorral elérhető legjobb megoldás.
4: **if** S' jobb mint S **then**
5: $S := S'$ és ugorjunk a ciklus elejére.
6: Legyen S' a *pár-áthelyezés* operátorral elérhető legjobb megoldás.
7: **if** S' jobb mint S **then**
8: $S := S'$ és ugorjunk a ciklus elejére.
9: Legyen S' a *blokk-csere* operátorral elérhető legjobb megoldás.
10: **if** S' jobb mint S **then**
11: $S := S'$ és ugorjunk a ciklus elejére.
12: Legyen S' a *pár-csere* operátorral elérhető legjobb megoldás.
13: **if** S' jobb mint S **then**
14: $S := S'$ és ugorjunk a ciklus elejére.
15: Output: S

A leállási feltétel két komponensből áll. Egyrészt leállunk akkor, ha a VNS már nem javít a megoldáson, másrészt egy időkorlát elérésekor, hogy beleférjünk az aktuális periódus időtartamába. Kérdés még, hogy hogyan találjuk meg az egyes operátorokkal elérhető legjobb megoldást? Erre a 6. algoritmusban adunk egy

példát, amely a blokk-áthelyezés operátorral végzett lokális keresést írja le. Az algoritmusban e^* az aktuálisan legjobb megoldás értéke, n^* egy felvételi pontra (blokk kezdőpontjára) mutató pointer, n^*_{prev} egy nem-végpontra mutató pointer. A blokkok kivágása és beillesztése a 4.1. fejezetben leírtak szerint történik.

6. Algoritmus Lokális keresés blokk-áthelyezéssel

1: Legyen S az inkumbens útvonalak szerinti megoldás.
2: $e^* := S$ értéke
3: $n^* := NULL$
4: $n^*_{prev} := NULL$
5: **for all** v_1 jármű és n_1 felvételi pont v_1 útvonalában **do**
6: Legyen n_1^{prev} az n_1-et megelőző pont.
7: Az n_1 kezdetű b blokkot eltávolítjuk v_1 útvonalából.
8: **if** v_1 útvonala nem felel meg a tervezési korlátoknak **then**
9: A b blokkot visszaillesztjük az n_1^{prev} utáni pozícióba.
10: **go to** 5
11: **for all** v_2 jármű és n_2 pont (nem végpont) v_2 útvonalában **do**
12: A b blokkot beillesztjük v_2 útvonalába, az n_2 utáni pozícióba.
13: **if** v_2 útvonala nem felel meg a tervezési korlátoknak **then**
14: A b blokkot eltávolítjuk v_2 útvonalából.
15: **go to** 11
16: **if** S értéke $< e^*$ **then**
17: $e^* := S$ értéke
18: $n^* := n_1$
19: $n^*_{prev} := n_2$
20: A b blokkot eltávolítjuk v_2 útvonalából.
21: A b blokkot visszaillesztjük az n_1^{prev} utáni pozícióba.
22: **if** $n^* \neq \varnothing$ **then**
23: Az n^* kezdetű blokkot áthelyezzük az n^*_{prev} utáni pozícióba.
24: Output: S

4.6. Számítási eredmények

A fejezetben tárgyalt algoritmusokat Python nyelven implementáltuk. Most bemutatjuk a számítási eredményeket és összehasonlítjuk a verseny dobogós megoldásaival. Feltüntetjük a 3.4. fejezetben szintén bemutatott CFA-VNS algoritmus [16] eredményeit is, amely a jelenleg ismert legjobb megoldásokat adja közepes és nagy példákra (5-8. tesztcsoportok). A 3. táblázat az átláthatóság kedvéért a tesztcsoportok átlagos eredményeit tartalmazza. Az egyes feladatpéldányok szerinti részletes eredmények a 48. oldalon kezdődő *Függelékben* találhatók.

Az átlagokat megvizsgálva szembeötlő, hogy a demó és naív algoritmusok mennyivel gyengébb eredményeket adnak a többinél. Ez persze amiatt nem meglepő, hogy ezekben a megoldásokban egy jármű egyszerre csak egyetlen csomagot szállít. Meglepő lehet viszont, hogy a naív algoritmus csak az 1. és a 3. csoportban múlja felül lényegesen a demó megoldást, és a feladat méretének növekedésével a két algoritmus közti különbség fokozatosan csökken. Ennek az lehet az oka, hogy az egyre több rendelés összesített késését egyre kevésbé kompenzálja az a tény, hogy míg a demó algoritmus teljesen figyelmen kívül hagyja a célfüggvényt, addig a naív azzal összhangban rendeli a csomagokat járművekhez.

A beillesztő algoritmus már látványosan jobb eredményeket produkál: az első kivételével minden tesztcsoportban 2-3 nagyságrenddel jobb végeredményt ad, mint a naív algoritmus. Ezeket az eredményeket még tovább javítja a VNS eljárás, amely minden iterációban a beillesztő algoritmus megoldásából indít lokális keresést. Ugyanakkor azt látjuk, hogy a VNS már nem jelent nagyságrendi javulást, csak a 4. és 6. csoport esetében add számottevően jobb eredményt.

Emellett viszont az az első látásra furcsa jelenség is előfordul, hogy a VNS eredménye még valamivel rosszabb is. Hogyan lehetséges ez? A válasz az, hogy hiába állít elő jobb megoldást a VNS az *egyes iterációkban*, ha azzal túlságosan bekorlátozza a jövőbeli lehetőségeit, akkor a beillesztő algoritmus globálisan jobb eredményre vezethet. Ugyanezt a jelenséget a VNS algoritmus tesztelése során is megfigyeltük: a korai iterációkban legköltséghatékonyabb megoldás esetenként

#	Demó	Naív	Beillesztő	VNS	Arany*	Ezüst*	Bronz*	CFA–VNS*
1	82 201	2 620	1 260	1 228	2 896	13 676	1 764	1 076
2	3 052 277	2 460 739	53 602	36 774	41 535	599 933	62 180	20 266
3	1 924 049	84 916	910	1 219	5 860	2 311	8 970	888
4	34 062 585	25 139 953	18 164	6 016	6 545	105 049	26 938	7 456
5	44 609 565	26 046 288	11 588	12 292	10 459	17 284	94 795	3 159
6	431 205 664	385 180 542	804 699	186 190	41 494	153 419	651 945	16 178
7	920 531 454	835 171 235	3 166 529	1 946 080	798 241	904 586	1 941 385	632 913
8	1 858 880 618	1 710 245 585	19 068 935	16 054 362	11 359 466	18 678 529	15 122 816	4 466 224

3. táblázat. A tesztcsoportok átlagos eredményei.
(*Forrás: Horváth et al. [16].)

rosszabb végeredménnyel jár. Vagyis a kizárólag gördülő tervezés megközelítést alkalmazó megoldások nem elég rugalmasak a dinamikus változások hatékony kezeléséhez. Ezen a problémán segíthet, ha a gördülő tervezést PFA és CFA típusú módszerekkel ötvözzük.

Még ezzel együtt is azt látjuk azonban, hogy a VNS algoritmus a kis méretű feladatokat tartalmazó 1-4. csoportok esetén lekörözi a verseny dobogós algoritmusait, a közepes (5-6.) és nagy (7-8.) tesztcsoportok esetén pedig nagyjából az ezüst- és bronzérmes megoldások szintjén mozog.

Végül meg kell említenünk a CFA-VNS eredményeit, amelyek szinte kivétel nélkül a legjobbak. Különösen látványos az előny a gyakorlati felhasználás szempontjából igazán releváns 5-8. csoportok esetében. Ez a CFA megközelítés hatásosságát mutatja. Az algoritmusban döntő szerepe van annak, hogy a járművek várakozással töltött idejét célzottan leszorítja. Ez közvetlenül nem befolyásolja a globális költségfüggvényt, közvetve azonban mégis jelentős költségcsökkenéshez vezet.

5. Összefoglalás

A dolgozatban egy dinamikus felvétel-lerakási feladatot (DPDP) vizsgáltunk, amely a Huawei infokommunikációs vállalat valós logisztikai problémáján alapul. A feladat eredetileg az ICAPS 2021 konferencia optimalizálási versenyén jelent meg. A dinamikus jelleg az igények szintjén jelenik meg: a tervezési horizont alatt folyamatosan beérkező rendelések miatt a járművek útvonalait és feladatait is folyamatosan frissíteni kell. A rendelésekhez teljesítési határidő is tartozik, és az optimalizálás elsődleges célja a késések minimalizálása. A feladatban a LIFO szabályt és az állomások kapacitáskorlátjából adódó kényszerű várakozást is figyelembe kell venni.

A problémát szekvenciális döntési folyamat (SDP) formájában modelleztük, és megoldó algoritmust is fejlesztettünk hozzá. Az algoritmus a gördülő tervezés (RHO) megközelítésen alapul. Minden tervezési iterációban az aktuálisan ismert információk alapján döntünk, mintha a jövőben már nem lenne több változás. Az egyes iterációkban először egy beillesztő algoritmus kiosztja az új rendeléseket a járművek között, majd ezt a kezdeti megoldást egy változó szomszédságú keresés (VNS) alkalmazásával javítjuk. Az algoritmusunk jól teljesít a probléma megoldásában: a kis méretű tesztpéldányok esetén lekörözi a verseny dobogós algoritmusait, a közepes és nagy instanciák esetén pedig nagyjából az ezüst- és bronzérmes megoldások szintjén mozog.

Ugyanakkor a számítási eredmények összehasonlításából az is kitűnik, hogy van még tér az algoritmus továbbfejlesztésére. Nagy potenciál van a 2.2.2. fejezetben bemutatott CFA megközelítésben, amelyet a jelenlegi legjobb megoldás is alkalmaz. Emellett a VNS eljárás fejlesztése is előrelépést jelenthet, például más operátorsorrend vagy új operátorok alkalmazásával. Vizsgálatra érdemesek még az erőforrás-tartalékoló stratégiák is, amelyek nagyobb mozgásteret biztosítanak a dinamikus változások kezeléséhez. További lehetőség a népszerű lokációk és sűrű időszakok vizsgálata, habár ezek már a túlillesztés veszélyét is felvetik.

Függelék

#	Demó	Naív	Beillesztő	VNS	Arany*	Ezüst*	Bronz*	CFA–VNS*
1	157 938	199	140	142	129	2 304	130	202
2	89 813	9 276	97	93	96	30 479	91	114
3	33 834	157	109	102	97	35 801	97	161
4	41 754	158	108	92	107	5 492	104	135
5	147 364	4 404	5 448	5 450	3 304	16 929	5 446	3 351
6	52 386	190	125	125	105	4 762	118	174
7	95 747	6 451	3 993	3 753	19 261	12 847	7 355	4 366
8	38 768	124	65	66	72	797	769	104
9	2 121 983	1 823 806	176	8 427	7 236	181 180	8 481	193
10	5 415 770	5 234 380	308 847	200 730	204 299	1 766 276	187 123	111 955
11	1 919 632	1 168 256	5 723	192	3 130	180 884	3 042	216
12	3 249 159	2 556 355	18 862	26 616	14 975	567 136	84 158	14 217
13	2 495 984	1 896 729	8 661	178	1 904	220 249	277	213
14	2 614 085	2 250 695	23 703	5 295	1 153	236 921	7 824	2 462
15	3 362 476	2 929 909	18 653	27 473	57 350	792 942	148 731	11 873
16	3 239 131	1 825 784	44 192	25 283	42 236	853 875	57 804	20 996
17	1 124 181	12 077	81	83	83	97	411	166
18	831 217	214	94	80	84	112	8 558	186
19	2 025 354	53 886	114	101	112	478	4 150	190
20	2 303 874	50 948	3 348	3 495	33 237	3 984	16 224	2 605
21	2 317 274	133 431	118	2 255	109	2 341	18 501	178
22	2 348 568	187 257	3 314	3 541	3 307	3 321	21 103	3 426
23	2 710 993	192 460	116	110	106	6 866	809	188
24	1 730 930	49 053	95	89	9 845	1 288	2 001	167
25	36 280 619	26 806 466	10 404	6 128	9 859	92 381	21 451	11 666
26	35 827 416	25 401 041	40 637	8 949	5 687	275 469	53 999	9 260
27	30 148 349	23 567 753	143	130	135	18 440	18 812	190
28	28 373 603	20 502 982	7 119	7 102	7 138	9 445	14 868	7 590
29	36 962 795	27 416 407	16 946	6 552	6 735	162 952	42 398	11 018
30	31 667 849	23 100 558	16 009	119	128	74 027	21 113	167
31	36 370 600	27 212 716	41 979	11 893	16 682	147 256	23 312	12 181
32	36 869 452	27 111 704	12 078	7 256	5 992	60 425	19 554	7 576

4. táblázat. Az 1-32. tesztpéldányok eredményei (1-4. tesztcsoport).
(*Forrás: Horváth et al. [16].)

#	Demó	Naív	Beillesztő	VNS	Arany*	Ezüst*	Bronz*	CFA–VNS*
33	41 372 735	22 335 675	2 810	2 792	71	7 031	95 017	195
34	43 446 471	23 656 850	15 022	12 573	7 314	10 503	54 374	3 200
35	45 742 469	27 870 018	100	3 674	3 442	15 761	103 753	203
36	44 088 153	26 872 728	17 622	19 142	16 362	23 186	112 879	4 666
37	45 121 134	28 275 518	12 107	12 426	7 994	5 142	95 740	205
38	49 146 413	27 255 770	14 958	17 374	14 878	30 136	94 230	3 793
39	45 841 600	28 467 724	19 461	19 749	20 167	22 253	85 226	5 962
40	42 117 545	23 636 021	10 624	10 604	13 444	24 262	117 141	7 050
41	434 824 819	397 038 176	496 715	114 978	45 770	124 253	548 756	7 940
42	429 253 738	383 046 922	524 557	53 613	20 713	164 002	600 982	20 343
43	425 283 510	377 040 487	955 504	352 555	57 092	166 646	550 661	18 836
44	419 494 681	370 006 306	978 508	222 092	62 934	190 604	715 481	34 649
45	431 436 930	377 605 116	1 116 594	481 339	63 793	181 171	616 245	10 245
46	455 753 070	402 833 164	1 310 019	165 182	45 418	176 491	798 331	17 874
47	420 420 264	387 507 786	454 529	80 619	21 321	104 337	602 968	11 152
48	433 178 302	386 366 377	601 168	19 143	14 915	119 848	782 137	8 387
49	928 558 344	852 239 540	3 523 433	2 336 168	852 983	1 184 048	1 594 690	788 852
50	895 425 453	800 188 207	3 669 343	2 340 501	869 247	572 493	2 512 247	828 236
51	895 416 695	831 970 594	2 198 848	1 275 846	215 636	515 467	1 335 194	214 956
52	915 754 348	851 846 894	3 453 795	1 295 238	507 888	447 647	2 030 603	814 458
53	982 553 863	886 272 194	3 264 608	1 116 187	35 138	316 974	1 930 299	435 550
54	911 726 082	814 397 402	3 503 414	2 705 960	1 641 164	1 629 784	1 995 016	621 130
55	890 922 621	787 985 910	1 998 379	1 461 379	337 435	729 537	1 975 596	530 423
56	943 894 225	856 469 135	3 720 409	3 037 359	1 926 434	1 840 740	2 157 437	829 699
57	1 837 682 797	1 719 408 639	20 232 747	16 575 811	11 402 231	22 433 901	16 517 326	4 474 936
58	1 805 309 412	1 697 770 838	17 617 261	15 326 340	9 345 817	13 243 401	11 214 249	4 141 195
59	1 899 532 209	1 679 659 308	18 868 881	15 286 872	8 791 534	12 351 488	9 542 269	4 351 745
60	1 847 871 472	1 708 355 206	19 205 538	18 794 397	10 309 934	21 521 732	13 191 278	5 634 971
61	1 788 153 867	1 636 806 184	17 007 605	13 134 717	5 115 366	9 095 706	7 207 770	3 499 727
62	1 849 395 294	1 710 296 191	19 465 627	16 258 777	11 168 899	14 541 496	10 857 351	3 717 233
63	1 889 175 713	1 758 203 644	19 928 461	15 506 615	19 126 599	30 883 611	25 025 233	5 326 702
64	1 953 924 183	1 771 464 670	20 225 358	17 551 365	15 615 352	25 356 899	27 427 054	4 583 280

5. táblázat. A 33-64. tesztpéldányok eredményei (5-8. tesztcsoport).
(*Forrás: Horváth et al. [16].)

Hivatkozások

[1] G. Berbeglia, J.-F. Cordeau, and G. Laporte. Dynamic pickup and delivery problems. *European journal of operational research*, 202(1):8–15, 2010.

[2] J. Cai, Q. Zhu, and Q. Lin. Variable neighborhood search for a new practical dynamic pickup and delivery problem. *Swarm and Evolutionary Computation*, 75:101182, 2022.

[3] J. Cai, Q. Zhu, Q. Lin, J. Li, J. Chen, and Z. Ming. An efficient multi-objective evolutionary algorithm for a practical dynamic pickup and delivery problem. In *International conference on intelligent computing*, pages 27–40. Springer, 2022.

[4] J. Cai, Q. Zhu, Q. Lin, L. Ma, J. Li, and Z. Ming. A survey of dynamic pickup and delivery problems. *Neurocomputing*, page 126631, 2023.

[5] J. Cai, Q. Zhu, Q. Lin, Z. Ming, and K. C. Tan. Decomposition-based multiobjective evolutionary optimization with tabu search for dynamic pickup and delivery problems. *IEEE Transactions on Intelligent Transportation Systems*, 2024.

[6] A. M. Campbell and M. Savelsbergh. Efficient insertion heuristics for vehicle routing and scheduling problems. *Transportation science*, 38(3):369–378, 2004.

[7] F. Carrabs, J.-F. Cordeau, and G. Laporte. Variable neighborhood search for the pickup and delivery traveling salesman problem with lifo loading. *INFORMS Journal on Computing*, 19(4):618–632, 2007.

[8] L. Cassani and G. Righini. Heuristic algorithms for the tsp with rear-loading. In *35th Annual Conference of the Italian Operations Research Society (AIRO XXXV), Lecce, Italy*, 2004.

[9] J.-F. Cordeau, G. Laporte, M. W. Savelsbergh, and D. Vigo. Vehicle routing. *Handbooks in operations research and management science*, 14:367–428, 2007.

[10] G. B. Dantzig and J. H. Ramser. The truck dispatching problem. *Management science*, 6(1):80–91, 1959.

[11] J. Du, Z. Zhang, X. Wang, and H. C. Lau. A hierarchical optimization approach for dynamic pickup and delivery problem with lifo constraints. *Transportation Research Part E: Logistics and Transportation Review*, 175: 103131, 2023.

[12] M. M. Flood. The traveling-salesman problem. *Operations research*, 4(1): 61–75, 1956.

[13] M. Gendreau, J.-Y. Potvin, O. Bräumlaysy, G. Hasle, and A. Løkketangen. *Metaheuristics for the vehicle routing problem and its extensions: A categorized bibliography*. Springer, 2008.

[14] J. Hao, J. Lu, X. Li, X. Tong, X. Xiang, M. Yuan, and H. H. Zhuo. Introduction to the dynamic pickup and delivery problem benchmark – ICAPS 2021 competition. *arXiv preprint arXiv:2202.01256*, 2022.

[15] M. Horváth and T. Tamási. A general modeling and simulation framework for dynamic vehicle routing. *arXiv preprint arXiv:2411.12406*, 2024.

[16] M. Horváth, T. Kis, and P. Györgyi. A cost function approximation method for dynamic vehicle routing with docking and lifo constraints. *arXiv preprint arXiv:2405.01915*, 2024.

[17] ICAPS 2021. Competitions, 2021. (Az ICAPS 2021 versenyei.) `https://icaps21.icaps-conference.org/Competitions/` (Utolsó letöltés: 2024. 11. 23.).

[18] ICAPS 2021: The dynamic pickup and delivery problem. Problem Description, 2021. (A verseny honlapja.) https://competition.huaweicloud.com/information/1000041411/circumstance (Utolsó letöltés: 2024. 11. 23.).

[19] ICAPS 2021: The dynamic pickup and delivery problem. Winning Team, 2021. (A díjazott munkák.) https://competition.huaweicloud.com/information/1000041411/Winning (Utolsó letöltés: 2024. 12. 02.).

[20] X. Li, W. Luo, M. Yuan, J. Wang, J. Lu, J. Wang, J. Lü, and J. Zeng. Learning to optimize industry-scale dynamic pickup and delivery problems. In *2021 IEEE 37th international conference on data engineering (ICDE)*, pages 2511–2522. IEEE, 2021.

[21] N. Mladenović and P. Hansen. Variable neighborhood search. *Computers & operations research*, 24(11):1097–1100, 1997.

[22] V. Pillac, M. Gendreau, C. Guéret, and A. L. Medaglia. A review of dynamic vehicle routing problems. *European Journal of Operational Research*, 225 (1):1–11, 2013.

[23] W. B. Powell. *Approximate Dynamic Programming: Solving the curses of dimensionality*. Wiley Series in Probability and Statistics: 842. New York: John Wiley & Sons, 2011.

[24] H. N. Psaraftis, M. Wen, and C. A. Kontovas. Dynamic vehicle routing problems: Three decades and counting. *Networks*, 67(1):3–31, 2016.

[25] A. Santini, M. Schneider, T. Vidal, and D. Vigo. Decomposition strategies for vehicle routing heuristics. *INFORMS Journal on Computing*, 35(3):543–559, 2023.

[26] N. Soeffker, M. W. Ulmer, and D. C. Mattfeld. Stochastic dynamic vehicle routing in the light of prescriptive analytics: A review. *European Journal of Operational Research*, 298(3):801–820, 2022.

[27] E. Talbi. Metaheuristics: From design to implementation. *John Wiley & Sons google schola*, 2:268–308, 2009.

[28] B. W. Thomas. Waiting strategies for anticipating service requests from known customer locations. *Transportation Science*, 41(3):319–331, 2007.

[29] M. W. Ulmer. *Approximate dynamic programming for dynamic vehicle routing*, volume 61. Springer, 2017.

[30] M. W. Ulmer, J. C. Goodson, D. C. Mattfeld, and B. W. Thomas. On modeling stochastic dynamic vehicle routing problems. *EURO Journal on Transportation and Logistics*, 9(2):100008, 2020.

[31] M. W. Ulmer, B. W. Thomas, A. M. Campbell, and N. Woyak. The restaurant meal delivery problem: Dynamic pickup and delivery with deadlines and random ready times. *Transportation Science*, 55(1):75–100, 2021.

[32] J. I. Van Hemert and J. A. La Poutré. Dynamic routing problems with fruitful regions: Models and evolutionary computation. In *International Conference on Parallel Problem Solving from Nature*, pages 692–701. Springer, 2004.

[33] P. van Nierop. Best practices to model carbon costs in supply chain network design, 2021. `https://www.aimms.com/story/towards-sustainable-supply-chain-network-design-starting-with-carbon-costs/` (Utolsó letöltés: 2024. 12. 14.).

[34] S. A. Voccia, A. M. Campbell, and B. W. Thomas. The same-day delivery problem for online purchases. *Transportation Science*, 53(1):167–184, 2019.

[35] Y. Zhou, L. Kong, L. Yan, Y. Liu, and H. Wang. A memetic algorithm for a real-world dynamic pickup and delivery problem. *Memetic Computing*, pages 1–15, 2024.