Dinamikus jármű útvonaltervezés

Önálló projekt, szakmai gyakorlat III.
2023/24 II. félév
Hatala Imre

Témavezető: Horváth Markó (SZTAKI)

1. Bevezetés

Ebben a félévben új témával foglalkoztam, ami azonban több szálon is kapcsolódik az első két projektmunkámhoz. Az egyik szál a logisztikai optimalizálás tárgyköre, a másik pedig a lokális keresés alkalmazása, ami jelen munkának központi eleme. A vizsgált feladat a dinamikus felvétel-lerakási problémák közé tartozik (Dynamic Pickup and Delivery Problem, DPDP). A projekt során egy konkrét, való életből vett problémát vettem górcső alá, amely az ICAPS és a Huawei által 2021-ben szervezett nemzetközi versenyről ([**1**]) származik.

A feladat a technológiai óriáscég egyik logisztikai problémáját járja körül: a termékek és alkatrészek különböző telephelyek közötti szállítását kell dinamikusan – azaz a valós idejű változásokra reagálva – optimalizálni. Ehhez figyelembe kell venni a logisztikai környezet és erőforrások korlátait, köztük olyan speciálisakat is, mint a LIFO rakodási szabály vagy a dokkolás és várakozás. Ez utóbbi akkor merül fel, ha egy telephelyen a dokkok számánál több járművet kellene kiszolgálni egyidejűleg; ilyenkor egyes járműveknek várakozniuk kell, és csak egy dokk felszabadulása után kezdődhet meg a kiszolgálásuk.

A beszámoló felépítése

A 2. fejezetben a szakirodalom alapján röviden áttekintem a témakört, a 3. fejezetben pedig bemutatom a dolgozat alapjául szolgáló logisztikai feladatot. A 4. fejezet tartalmazza a problémára adott megoldásokat: két egyszerű algoritmussal kezdünk, majd egy okosabb megközelítéssel folytatva végül eljutunk a lokális keresést alkalmazó megoldáshoz, ami jelen dolgozat fő eredménye. A fejezet végén az egyes módszerek eredményeit összehasonlítom az említett verseny első három helyezettjének eredményeivel. Végül az 5. fejezetben felvázolok továbbfejlesztési lehetőségeket.

2. A témakör áttekintése

Röviden ismertetem a felvétel-lerakási problémát és a dinamikus jármű útvonaltervezési problémák lényegét, mint a DPDP alapjait.

2.1. A felvétel-lerakási probléma

A logisztikai optimalizálás egyik alapfeladata a jól ismert utazó ügynök probléma (Travelling Salesman Problem, TSP), aminek általánosított változatai a különböző jármű útvonaltervezési

feladatok (Vehicle Routing Problem, VRP). Utóbbi témakör egyik sokat kutatott szegmense a felvétel-lerakási probléma (Pickup and Delivery Problem, PDP), ahol termékeket megadott felvételi pontokról kell elszállítani a megfelelő lerakási pontokra a rendelkezésre álló járműflotta segítségével. Az optimalizálás célja tipikusan a megtett távolság, a felhasznált járművek száma és az eltelt idő minimalizálása. Közismert, hogy a TSP probléma NP-nehéz, ennek megfelelően az általánosított feladatok is NP-nehezek. **[2]**

2.2. Dinamikus jármű útvonaltervezési problémák

A VRP témakörben egyre nagyobb szerepet kap a sztochasztikus-dinamikus jármű útvonaltervezési problémák (Stochastic Dynamic Vehicle Routing Problem, SDVRP) vizsgálata, amikor valamilyen sztochasztikus bizonytalanságot is figyelembe kell vennünk az optimalizálás során. Ebben az esetben statikus, illetve determinisztikus megközelítés helyett érdemesebb valamilyen adaptív módszert alkalmazni a tervezéshez. Más szóval: a tervezés nem egy egyszeri döntéshozatal lesz, ahol egyetlen, előre rögzített szállítási terv készül (statikus tervezés), hanem időben dinamikusan történik a döntéshozatal, az újabb és újabb információk beérkezésével bizonyos időközönként módosíthatjuk, kiegészíthetjük a szállítási tervet (dinamikus tervezés). A bizonytalanság, ami miatt a probléma dinamikus tervezést igényel, egyaránt jelen lehet az igények (pl. folyamatosan érkező megrendelések), a logisztikai környezet (pl. dugók, útlezárások) és az erőforrások (pl. jármű vagy sofőr rendelkezésre állás) tekintetében is. **[3]**

Az ilyen adaptív, dinamikus tervezési módszerek sokkal precízebben és életszerűbben képesek modellezni a valóságot, mint determinisztikus társaik, ennek köszönhetően sokkal jobban is teljesítenek komplex, a való életből vett logisztikai problémák megoldásában.

3. A feladat

A versenyen szereplő feladat a Huawei valós logisztikai problémájára keres megoldást: termékek és alkatrészek különböző gyárak közötti szállítását kell megtervezni, figyelembe véve a logisztikai környezet és erőforrások korlátait. A megrendelések (adott csomagok szállítási feladatai) folyamatosan, tíz perces időközönként érkeznek be. A feladatban tehát az igények dinamikusak, a logisztikai környezet és az erőforrások statikusak. A problémát először statikus formában írom le (mintha a megrendelések előre ismertek lennének) és majd azután fogalmazom meg a feladat dinamikus változatát.

3.1. A statikus feladat

Input

- $D = (F, A)$ útvonalhálózat, mint teljes irányított gráf, ahol F a gyárak és A az útvonalak halmaza
- $F = \{F_i | \ i = 1, \ldots, M\}$ gyárak
  - n^i_{dock}: dokkok száma

    - t^i_{dock}: dokkolási idő
- $A = \{(i,j) | \ i,j \in F\}$ útvonalak
    - d_{ij}: távolság
    - t_{ij}: utazási idő
- $O = \{O_i | \ i = 1, \dots, N\}$ rendelések, ahol $O_i = \left(F^i_p, F^i_d, q^i, t^i_p, t^i_d, T^i_\alpha, T^i_\omega\right)$ és
    - F^i_p: felvételi pont
    - F^i_d: lerakási pont
    - q^i: összesített méret
    - t^i_p: felrakodási idő
    - t^i_d: lerakodási idő
    - T^i_α: létrehozás időpontja
    - T^i_ω: kézbesítési határidő
- $V = \{V_i | \ i = 1, \dots, K\}$ járművek
    - c_i: kapacitás
    - p_i: kezdő pozíció (egy gyár)

Output

Teljes szállítási terv, azaz az egyes járművek útvonaltervei, amelyben minden állomásnál jelölve van, hogy mely csomagokat veszi fel, illetve rakja le az adott jármű.

Korlátok

- Minden rendelést kézbesíteni kell.
- A szállítás folyamata (általános esetben) a következő: érkezés a felvételi pontra, esetleges várakozás dokk felszabadulására, dokkolás, felrakodás, szállítás, érkezés a lerakodási pontra, esetleges várakozás dokk felszabadulására, dokkolás, lerakodás.
- Be kell tartani a járművek kapacitás korlátját.
- Az egy rendeléshez tartozó árucikkeket együtt kell szállítani. Több csomagra való felosztás csak akkor engedélyezett, ha a rendelés mérete meghaladja a jármű kapacitását.
- LIFO (Last In, First Out) rakodási szabály. Először azt az árucikket kell kirakodni, amelyik a legkésőbb került be a csomagtérbe. Más szóval, a járművekről való lerakodás sorrendje a felrakodási sorrend fordítottja. Például, ha az O_1 és O_2 rendeléseket egy jármű teljesíti, akkor az $\left(F^1_p, F^2_p, F^1_d, F^2_d\right)$ útvonal megsérti a LIFO korlátot, az $\left(F^1_p, F^2_p, F^2_d, F^1_d\right)$ útvonal viszont teljesíti a LIFO korlátot.
- A dokkok száma, ahol a járművek elvégezhetik a fel- illetve lerakodást, korlátozott. Ha az összes dokk megtelt, akkor a soron következő járműnek várakoznia kell, amíg egy dokk fel nem szabadul.

Célfüggvény

Ha egy rendelést nem sikerül a megadott határidőre teljesíteni, akkor az a késés mértékével arányos büntetést eredményez. Ez adja a célfüggvény első tagját. Ha egy rendelést a kapacitáskorlát miatt fel kellett osztani, akkor a késést a legutolsó csomag beérkezési ideje alapján kell számítani.

A célfüggvény másik tagja a járművek által megtett átlagos távolság. Tehát a teljes célfüggvény

$$min\, f = \lambda \times f_1 + f_2$$

ahol

- $\lambda \geq 1$ konstans (a szimulátorban $\lambda = \frac{10000}{3600} \approx 2{,}78$ van rögzítve),
- f_1: összesített késés, azaz

  $$f_1 = \sum_{i=1}^{N} max\left(0, T_c^i - T_\omega^i\right)$$

  ahol T_c^i jelöli az $i.$ rendelés teljesítési időpontját,
- f_2: a járművek átlagos megtett távolsága, azaz

  $$f_2 = \frac{1}{K} \sum_{i=1}^{K} \sum_{j=1}^{l_i - 1} d_{n_j^i, n_{j+1}^i}$$

  ahol az $i.$ jármű útvonala $\Pi_i = (n_1^i, n_2^i, \dots, n_{l_i}^i)$.

Mivel versenyen a probléma valósághű instanciái szerepelnek, és a késés mértékegysége másodperc, a távolságé pedig kilométer, ezért az f_1 tag (késés) tipikusan nagyságrendekkel nagyobb, mint az f_2 (átlagos megtett távolság). Ennek megfelelően az optimalizálás fő célja a késés minimalizálása.

3.2. A dinamikus feladat

A dinamikus változatban a statikus alapfeladatot vesszük, annyi különbséggel, hogy a rendelések nem ismertek előre, csak a létrehozás időpontjában válnak ismertté – ahogy a valóságban is. A versenyhez biztosított egy szimulátor, ami iterációsan modellezi a rendelések beérkezését és a szállítás folyamatát is. Egy iterációban, amely virtuálisan tíz percig tart, a szimulátor kiszámítja a (virtuálisan) eltelt tíz perc történéseit és az új iteráció elején visszaadja a feladat aktuális állapotát, valamint az időközben beérkezett új rendelésekkel frissíti a rendelések listáját. Ezután lehetőség van a szállítási tervek módosítására vagy kiegészítésére.

A dinamikus feladat minden iterációjában tehát további input adatok érkeznek. Ezenkívül egy új korlát is bevezetésre kerül.

További input iterációnként

- Úton lévő járművekhez
  - F_{dest}^i: célállomás (egy gyár)
  - T_{arr}^i: érkezési idő a célállomásra

    - L_d^i: célállomásra vonatkozó lerakodási lista (csomagok listája)
    - L_p^i: célállomásra vonatkozó felrakodási lista (csomagok listája)
- Állomáson lévő járművekhez
    - F_{curr}^i: aktuális állomás (egy gyár)
    - T_{dep}^i: indulási idő (amikor leghamarabb elindulhat)
- Minden járműhöz
    - L_{curr}^i: a járművön lévő csomagok listája (ez a gyárba érkezés pillanatában frissül, azaz amit oda kellett vinni, lekerül a listáról, amit ott kell felvenni, felkerül a listára)
    - $\mathcal{L}^i$: a jármű meglévő útvonalterve, amely tartalmazza az egyes állomásokra vonatkozó fel- és lerakodási listákat is

További korlátok

- Úton lévő jármű célállomását nem lehet megváltoztatni.

Output

Szállítási terv iterációnként, azaz minden iterációban meg kell adni az egyes járművek tervezett útvonalát, valamint az egyes gyáraknál esedékes fel- és lerakodási feladatokat.

A megoldás kiértékelése

A szimulátor ellenőrzi a korlátok teljesülését, szimulálja a következő 10 perc eseményeit, majd előállítja az inputot a következő iteráció számára. A teljes szimuláció végén pedig kiértékeli a globális célfüggvényt.

4. Megoldások a feladatra

A következőkben bemutatom a problémára adott megoldásokat. Először a szimulátorhoz csatolt *demo* algoritmust írom le, majd ennek egy kicsit okosabb változatát, a *naiv* algoritmust. Ezek után a *best insert* algoritmus már egy lépés a lokális keresés – röviden *LS* – felé. A fejezet végén összehasonlítom az egyes módszerek eredményeit a verseny első három helyezettjének megoldásaival (**1. ábra**).

Az összehasonlítás alapja a verseny szervezői által biztosított 64 teszt instancia, amelyek egyre nehezedő nyolcas csoportokra vannak felosztva. Fontos, hogy ebből csak az első két instanciacsoportot vizsgáltam a projekt során; az algoritmusaim ezekre íródtak, és az összehasonlítás is csak ezeken történik. Ezen instanciák esetében a dokkok száma mindig nagyobb lesz a járművek számánál, ezért a várakozással nem kell számolni, és így az optimalizálási feladat leegyszerűsödik. Azonban még így is elég komplex és érdekes marad.

A következőkben bemutatott algoritmusokat Python nyelven implementáltam. A megvalósítás technikai részletei közül csak a legfontosabbat említem meg, hogy az egyes járművek útvonalait láncolt listában tároljuk. A láncolt lista struktúra lehetővé teszi az útvonalak egyszerű és gyors

módosítását, különösen az útvonal-részletek kivágása és beillesztése esetében, ami kulcsfontosságú a lokális keresésen alapuló megoldások hatékonysága érdekében. Az alábbi algoritmusok mindegyikét láncolt listában tárolt útvonalakkal implementáltam.

Megjegyzés

A túlméretes rendeléseket csak több darabra osztva lehet teljesíteni. Az alábbi algoritmusok mindegyike elvégzi ezt a felosztást, ha szükséges. Az egyszerűség kedvéért ezt nem részletezem külön és a „rendelés", illetve „feladat" alatt egyaránt olyan felvételi-lerakási feladatot fogok érteni, ami már megfelelő méretű csomagra vonatkozik.

4.1. A *demo* algoritmus

A szimulátorhoz csatolt *demo* algoritmus esetében a feladatkiosztás úgy néz ki, hogy a járművek egyesével kapnak felvételi-lerakási feladatokat: az üres jármű elmegy a felvételi pontra, felveszi a hozzá rendelt egyetlen csomagot, azt elszállítja a lerakási pontra, majd megy tovább a következő neki kiosztott rendelésért. Tehát itt egy jármű mindig vagy üres, vagy pedig egyetlen csomagot szállít.

Az algoritmus lépései

1. →A járművek már kiosztott feladatai nem változnak, tehát minden járműnek a meglévő útvonalterv lesz az aktuális iterációban is a kezdeti útvonalterve.
2. →Ha van ki nem osztott megrendelés, akkor ezek közül az $i.$ megrendelést az $i \bmod K$ sorszámú járműhöz allokáljuk: a jármű útvonalához hozzáfűzzük a rendelés felvételi és lerakási pontját (a megfelelő felvételi és lerakási listákkal együtt).

Az algoritmus helyes megoldást ad abban az értelemben, hogy kiszállít minden rendelést, nem sérti meg a kapacitás korlátot, 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, sőt aránytalanul jobban terheli a kisebb sorszámú járműveket. Ez nagy mértékű összesített késéshez, így nagyon rossz globális célfüggvényértékhez vezet.

4.2. A *naiv* algoritmus

A *naiv* algoritmus a *demo* algoritmuson alapul, hasonlóan mohó elven működik, de a mohó elv itt 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.

Az algoritmus lépései

1. →A *demo* algoritmus első lépése.
2. ↻ Ciklus: sorban tekintjük az újonnan beérkezett rendeléseket.
3. →Minden járműre kiszámítjuk, hogy mikor végez a meglévő feladataival, valamint az utolsó (lerakodási) feladat lokációja és a kiosztandó rendelés felvételi pontja közti utazási időt.
4. →Azt a járművet választjuk, amelyik minimalizálja a kettő összegét.
5. →A kiválasztott jármű útvonalához hozzáfűzzük a rendelés felvételi és lerakási pontját.

A *demo* algoritmushoz hasonlóan a *naiv* algoritmus is helyes megoldást ad, azonban sokkal jobb eredménnyel: az első tesztcsoport példányain nagyságrendekkel jobb értékeket ér el és a második tesztcsoport esetén is javít az eredményeken (**1. ábra**).

4.3. A *best insert* algoritmus

Ez az algoritmus az új rendeléseket egymás után beilleszti járművek útvonaltervébe oly módon, hogy minden lépésben minimalizálja a keletkező terv célfüggvényértékét. Ehhez minden soron következő rendelésnél végigpróbálja az összes lehetséges beillesztést és kiválasztja közülük a legjobbat. Ez a módszer már egy köztes lépés a lokális keresés felé, sőt, tulajdonképpen tekinthető a lokális keresés egy egyszerű megvalósításának is.

Az algoritmus lépései

1. →A *demo* algoritmus első lépése.
2. ↻ Ciklus: sorban tekintjük az újonnan beérkezett rendeléseket.
3. →A legjobb beillesztést tároló változót üresre inicializáljuk.
4. ↻ Ciklus: sorban tekintjük a járműveket.
5. ↻ Ciklus: sorban tekintjük a feladat összes lehetséges beillesztését a jármű útvonalába.
6. →Végrehajtjuk a beillesztést.
7. →Ellenőrizzük a tervezési korlátokat (úton lévő jármű célállomása nem változhat, kapacitáskorlát, LIFO szabály).
8. →Ha a tervezési korlátok nem sérülnek, akkor kiértékeljük a célfüggvényt. Ha ez jobb, mint az eddigi legjobb beillesztés eredménye, akkor ezt a beillesztést tároljuk el új legjobbként.
9. →Eltávolítjuk a feladatot a jármű útvonalából.
10. →Alkalmazzuk a legjobb beillesztést.

A *best insert* algoritmus az első tesztcsoport esetén is javít az eredményeken a *naiv* algoritmushoz képest, a második tesztcsoport példányain pedig nagyságrendekkel jobb értékeket ér el (**1.ábra**).

4.4. Lokális keresés

A lokális keresés (Local Search, LS) egy általános optimalizálási módszer, melynek alapötlete, hogy megengedett megoldásból kiindulva lépesenként igyekszünk javítani a megoldáson oly módon, hogy minden iterációban az aktuális megoldás egy lokális környezetét (ezt nevezzük keresési térnek) fésüljük át jobb megoldást keresve. A módszer alapos bemutatása megtalálható az első projektmunkához kapcsolódó beszámolómban ([**4**]), ennek részletezésétől most eltekintek.

A keresési teret definiálhatjuk operátorok segítségével, amelyek egy (megengedett) megoldást módosítanak úgy, hogy egy másik (megengedett) megoldást kapjunk: a keresési tér azon megoldások halmaza lesz, amelyek az aktuális állapotból előállíthatók a rendelkezésre álló operátorok alkalmazásával. Számos operátort tárgyal a szakirodalom különböző problémák lokális keresésen alapuló megoldásai esetében, amelyek közül sokat jó eredményekkel alkalmaztak PDP problémák esetén és LIFO korlát mellett is. [**5**]

Ezek közül a következő négy operátort vizsgáltam meg és alkalmaztam a lokális kereséshez. Minden esetben csak olyan módosításokat tekintünk, amelyek megengedett megoldást eredményeznek.

- *Block relocation*: egy jármű útvonaltervének egy szakaszát (egy ún. blokkot) áthelyezzük egy másik helyre (ami lehet egy másik jármű útvonalterve is).
- *Couple relocation*: egy jármű útvonaltervéből egy felvétel-lerakási feladatpárt kiemelünk és egyenként beillesztjük őket egy másik helyre (ami lehet egy másik jármű útvonalterve is).
- *Block exchange*: két diszjunkt blokkot (amelyek lehetnek azonos vagy különböző útvonalterv részei) megcserélünk.
- *Couple exchange*: két felvétel-lerakási feladatpárt (amelyek lehetnek azonos vagy különböző útvonalterv részei) megcserélünk, azaz megcseréljük a két felvételi feladatot és megcseréljük a két lerakási feladatot is.

Az algoritmus lépései

1. →Lefuttatjuk a *best insert* algoritmust.
2. ⟳ Ciklus: amíg le nem állítjuk.
3. →Eltároljuk az aktuális megoldás célfüggvényértékét.
4. ⟳ Ciklus: sorban tekintjük az elérhető operátorokat.
5. ⟳ Ciklus: sorban tekintjük az aktuális operátor szerinti módosításokat.
6. →Végrehajtjuk a módosítást.
7. →Kiértékeljük a célfüggvényt. Ha jobb eredményt kapunk, mint az eltárolt érték, akkor a 2. sorra ugrunk.
8. →Visszavonjuk a módosítást.
9. →Leállítjuk a ciklust.

Az algoritmust többféle konfigurációval is lefuttattam. Külön vizsgáltam azokat az eseteket, amikor csak egyetlen operátort lehet használni, illetve azt az esetet, amikor mind a négy operátor elérhető.

Az eredményeket vizsgálva az látszik, hogy az első tesztcsoport példányain a lokális keresés már nem tud szignifikánsan javítani a *best insert* algoritmushoz képest, viszont a második tesztcsoport esetében már javulás figyelhető meg, ami néhány esetben nagyságrendi különbséggel is jár. Érdekesség, hogy az összes operátort tartalmazó konfiguráció teljesítménye egyáltalán nem kiemelkedő az egy operátoros konfigurációkhoz képest, illetve hogy utóbbiak közül mindegyiknél van olyan instancia, amelyen szignifikánsan jobban teljesítenek a többi módszernél.

#	Demo	Naiv	Best insert	LS – block relocation	LS – couple relocation	LS – block exchange	LS – couple exchange	LS – all operators	Gold	Silver	Bronze
1	157 938.2	198.1	142.5	141.8	138.3	142.4	141.6	132.6	134.0	2 300.0	130.0
2	89 812.9	9 206.1	103.8	89.7	92.1	97.6	96.8	92.1	95.6	30 500.0	91.4
3	33 834.0	157.1	102.6	98.7	104.6	101.0	101.3	102.5	96.8	35 800.0	96.5
4	41 754.2	158.3	102.9	108.1	104.7	101.3	102.7	100.8	94.6	5 490.0	104.0
5	147 364.0	4 403.6	5 448.5	5 451.7	5 451.6	5 448.5	5 448.7	5 451.9	3 310.0	16 900.0	5 450.0
6	52 385.6	190.2	120.6	128.5	105.6	120.8	120.8	120.6	105.0	4 760.0	118.0
7	95 747.5	6 451.3	3 988.1	3 752.6	3 752.6	3 987.7	3 988.1	3 788.6	4 390.0	12 800.0	7 360.0
8	38 768.0	124.3	63.1	63.9	66.7	61.6	61.6	64.3	68.8	797.0	769.0
9	2 121 982.7	1 823 806.3	227.3	170.7	172.2	166.2	3 742.8	6 553.1	152.0	181 000.0	8 480.0
10	5 415 769.6	5 234 380.2	225 918.2	208 801.2	201 300.0	344 534.1	236 255.2	250 489.7	194 000.0	1 770 000.0	187 000.0
11	1 919 632.2	1 168 255.7	9 094.5	2 305.9	2696.9	4 036.4	173.2	5 750.4	198.0	181 000.0	3 040.0
12	3 249 158.9	2 556 355.2	27 872.0	83 011.8	78 175.3	39 146.9	30 224.2	38 807.4	52 900.0	567 000.0	84 200.0
13	2 495 984.3	1 896 729.5	24 634.9	3 817.0	185.0	7 440.5	6 450.3	4 572.0	7 180.0	220 000.0	277.0
14	2 614 084.7	2 257 233.7	6 841.1	145.5	13 570.9	4 546.1	177.9	2 101.5	9 390.0	237 000.0	7 820.0
15	3 362 476.0	2 771 743.4	48 309.0	25 320.9	13 490.5	40 674.2	21 979.3	22 782.8	13 500.0	793 000.0	149 000.0
16	3 239 131.3	1 825 784.3	42 837.9	32 369.4	22 064.2	49 447.1	49 452.8	20 994.7	16 800.0	854 000.0	57 800.0

1. ábra: A dolgozatban szereplő algoritmusok és az eredeti verseny dobogós megoldásainak eredményei ([**6**], [**7**]) az 1-16 sorszámú teszt instancián.

4.5. Konklúzió

A *best insert* és a lokális keresésen alapuló algoritmusok jól teljesítenek az első két tesztcsoport példányain, figyelembe véve, hogy az eredeti verseny dobogós megoldásaival összemérhető minőségű eredményt adnak, sőt esetenként még azoknál is jobbat. Természetesen ez nem jelenti azt, hogy ezek az algoritmusok a teljes feladaton, illetve az összes tesztcsoport példányain ilyen jól teljesítenének. Valószínű, hogy a komplexebb, várakozást is tartalmazó instanciák esetében sokkal szerényebb eredményeket kapnánk.

Annál is inkább, mivel ezek az algoritmusok „mohó” elven működnek abban az értelemben, hogy csak az aktuálisan ismert rendelésekre vannak tekintettel és arra irányulnak, hogy az igényeket a lehető leghamarabb kielégítsék. Ennek a megközelítésnek az a hátulütője, hogy az erőforrásokat

potenciálisan pazarló módon használja fel (pl. a kapacitás töredékét kitevő csomaggal is útnak indítja a járművet) és így az újonnan megjelenő igényekre nem tudnak eléggé rugalmasan reagálni. Ez lehet a magyarázata annak a furcsa jelenségnek is, amit a sporadikusan előforduló, a többi megoldáshoz képest szignifikánsan jobb eredmények jellemeznek: ezek az eredmények bizonyítékok arra, hogy az adott példány esetén létezik „nagyon jó" megoldás, viszont az algoritmusok tipikusan nem találják meg ezeket, mert a „mohóság" eredményeként egyes kritikus döntési helyzetekben nem áll rendelkezésre elegendő mozgástér.

5. További fejlesztési lehetőségek

Az előbbiek alapján valószínűsítem, hogy a rugalmasság lehetőségét fenntartó tervezésben rejlik a legjelentősebb fejlesztési lehetőség. Egyfajta tartalékot kell képezni az erőforrásokból (jelen esetben a rendelkezésre álló szállítási kapacitásból), hogy azt a kritikus döntési helyzetben be lehessen vetni. Ilyen módszer lehet például a távoli határidővel rendelkező megrendelések teljesítésének késleltetése vagy az alacsony töltöttségű járművek várakoztatása, illetve kerülőútra küldése. Azt is érdemes lehet megvizsgálni, hogy vannak-e sűrűn látogatott lokációk, amelyek közelében megéri szabad kapacitást állomásoztatni. Ezek mellett a lokális keresés továbbfejlesztése is megfontolandó, például kifinomultabb operátorok alkalmazásával.

Ezt a témát tervezem folytatni szakdolgozat formájában is, ahol terveim szerint ezeket az ötleteket is meg fogom vizsgálni, valamint továbbfejlesztem az algoritmusokat, hogy a várakozást is képesek legyenek (jól) kezelni. Ezek után lehetőség nyílik majd az összes teszt példánnyal való tesztelésre, illetve az eredeti versenyen született megoldásokkal való teljes értékű összehasonlításra.

Hivatkozások

[1] https://competition.huaweicloud.com/information/1000041411/introduction

[2] J. Cai, Q. Zhu, Q. Lin, L. Ma, J. Li, Z. Ming: *A survey of dynamic pickup and delivery problems.* Neurocomputing 554 (**2023**) 126631

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

[4] https://math-projects.elte.hu/projects/work/127/

[5] F. Carrabs, J-F. Cordeau, G. Laporte: *Variable neighborhood search for the pickup and delivery traveling salesman problem with LIFO loading.* INFORMS Journal on Computing Vol. 19, No. 4, Fall **2007**, pp. 618–632

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

[7] Horváth Markó (SZTAKI), a verseny résztvevője, a harmadik helyezett csapat tagja.