Holló-Szabó Ákos* és **Botzheim János****

Jármű úttervezési probléma megközelítése bakteriális memetikus algoritmussal

Bevezetés

A logisztikai költségek, avagy a szállítás és raktározási költség az egész gazdaságot egyenletesen terheli. Hiszen bármely fizikai termék gyártásánál és forgalmazásánál jelen van. Minden százalék, ami lefaragható belőle több profitot hagy a forgalmazóknak, míg a termékeket olcsóbbá teszi. Ráadásul a logisztika egyre nagyobb szerephez jut, ahogy nő az online kereskedelem népszerűsége. A COVID-19 járvány is jelentősen hozzájárult ehhez a tendenciához, hiszen sokkal többen rendeltek házhoz, hogy kerüljék a kontaktust.

A jármű úttervezési probléma (vehicle routing problem, VRP)[3] a megfelelő kiterjesztésekkel egy valós szállítás optimalizálás legpontosabb matematikai modellje. Figyelembe veszi, a járművek, a sofőrök, az ügyfelek közötti útszakaszok és a szállítandó csomagok fizikai tulajdonságait is. A szállítás költsége ekkor első sorban a sofőr fizetéséből és a jármű fogyasztásából származik, így könnyedén számítható az üzemanyag liter ára és a sofőr bérezésének ismeretében. Az egyszerűsítés kedvéért kutatásunk során egy szállítási központ egy napjára tervezünk, ahol minden helyszínre egy csomag kerül szállításra.

A VRP egy optimalizációs probléma. Létezik egy egyszerűbb feladat, például, hogy minden ügyfél legyen kiszolgálva, és a megoldáshalmazának egy részhalmazát keressük, ami megfelel egy szigorúbb követelménynek. Ráadásul tudunk rendelni a tágabb megoldáshalmaz minden eleméhez egy olyan konstanst, ami minimális, ha az adott megoldás megoldása az optimalizációs problémának is. Ez a konstans bármi lehet, amíg jól megfogalmazza az elvárásainkat, és a mi esetünkben sem egyértelmű ennek az értéknek a meghatározása. Milyen költségeket tekintünk

* *Holló-Szabó Ákos, mester hallgató, Automatizálási és Alkalmazott Informatikai Tanszék, Villamosmérnöki és Informatikai Kar, Budapesti Műszaki és Gazdaságtudományi Egyetem, Budapest*

** *Dr. Botzheim János, tanszékvezető egyetemi docens, Mesterséges Intelligencia Tanszék, Informatikai Kar, Eötvös Loránd Tudomány Egyetem, Budapest*

elhanyagolhatónak? Mi az adott cégnél a szállítás, adminisztráció menete. Egyáltalán mit minimalizálunk? Alapértelmezetten ez pénzben kifejezett költség, de lehet a szállítás környezetre gyakorolt negatív hatásának is a mértéke vagy az ügyfelek elégedettsége.

Ebben a publikációban egy olyan algoritmust közlünk a VRP-re, ami rugalmasan személyre szabható bármely cég szokásaira, elvárásaira és hatékony megoldást nyújt. A bemutatott bakteriális memetikus algoritmus kellően hatékony ahhoz, hogy jól alkalmazható legyen nagyobb feladat méretekre is. A továbbiakban beszámolunk az általunk tapasztalt korlátairól is a módszernek, melyeket későbbi publikációkban kívánunk kiküszöbölni.

A probléma elméleti háttere

A VRP egy közismert problémára, az utazó ügynök problémára (travelling salesman problem, TSP)[2] vezethető vissza. Mivel a TSP NP-teljes, ezért a VRP is NP-beli[2], tehát túlzott futási ideje miatt nem számítható egy adott bemenet hossz felett az emberiség számára rendelkezésre álló erőforrásokkal. A TSP-ben egy utazó ügynök szeretné kiszolgálni az összes ügyfelét egyetlen útja alatt. Az ügynök célja, hogy ez az út hossza minimális legyen. Rendelkezésére áll egy súlyozott, teljes gráf, aminek csúcsai a helyszínek és élein a súly pedig a helyszínek közötti útszakaszok hosszai. Tulajdonképpen ezen a gráfon keres egy olyan kört, ami minden csúcsot pontosan egyszer érint és a súlya minimális, avagy egy minimális összsúlyú Hamilton kört.

Egy ilyen kör ugyanakkor jól reprezentálható egy c hosszú permutációval, ahol c az ügyfelek száma. Ez a reprezentáció ideális, hiszen bijektíven megfeleltethetőek a lehetséges körök és permutációk. Ebből viszont az következik, hogy $c!$ lehetséges körről van szó, melyeknek bejárása $O(c \cdot c!)$ számítási időt vesz igénybe. Ez azt jelenti, hogy a vizsgálandó megoldás halmaz túl nagy, száz ügyfél esetén is az emberiség összes ideje és erőforrása kevés lenne a kiszámításához.

A TSP ugyanakkor jelentős egyszerűsítése a VRP-nek.

A valóságban egy szállításnál az útköltségek irány függőek, avagy a helyszínről b helyszínre nem ugyanaz az út hossza, mint b helyszínről a helyszínre. Ebben az esetben a gráfunk irányított. Az irányított gráfos TSP-t aszimmetrikusnak is nevezzük (ATSP). A vizsgálandó megoldáshalmaz kétszer akkora, hiszen TSP-nél az ellentétes bejárási sorrend

költsége azonos, mivel ugyanazok az élek szerepelnek benne, és így a köröknek csak a felét kell ellenőrizni.

A szállítási egységeknek kapacitásai vannak. Súlyban és térfogatban is korlátozott a raktér, és a munkaerőnek munkaideje is korlátozott. Tehát a szállítási egység nem feltétlen képes minden ügyfelet kiszolgálni és adott ügyfeleket sem minden sorrendben. Ez azt is jelenti, hogy ha a szállítás meg is oldható, nem minden kiszolgálási sorrend elfogadható. Az invalid körökre mondhatnánk azt, hogy a költségük végtelen, de ez megnehezíti a teljesíthető szállítások keresését. Ezért inkább úgy szoktuk tekinteni, hogy a szállítás megoldható és egy büntető költséget vezetünk be. A büntető költségnek az a célja, hogy a kiszolgálási sorrend költsége magasabb legyen, mint bármely hozzá hasonló, de megvalósítható sorrendnek. Például a nem kiszolgálható ügyfelekhez tartozó élek súlyát vehetjük maximálisra ez által bármely kör súlya kisebb lesz, ami több ügyfelet szolgál ki. Úgy is megoldhatjuk ezt a problémát, hogy ekkor a szállító egység több körben szolgálja ki az ügyfeleket. Ekkor a problémát kapacitásosnak is nevezzük (CTSP).

A VRP több ágenses is (MTSP), avagy nem egy ügynökünk van, hanem több szállító egységünk. Nem csak egy kiszolgálási sorrendet szeretnénk optimalizálni, hanem az elérhető szállító egységek közötti kiosztását is az ügyfeleknek. Ekkor $c! \cdot \binom{c+s-1}{s-1}$ lehetséges kiosztás-permutáció páros létezik, ahol s a szállító egységek száma. Ekkor a gráfon nem egy Hamilton kört, hanem több kört keresünk, melyek mind érintik a szállítási központot és minden más helyszínt pontosan egyszer együttesen. Az összes eset vizsgálata ekkor $O\left((c+s)\cdot c!\cdot\binom{c+s-1}{s-1}\right)$ időt vesz igénybe.

A legnagyobb különbség ugyanakkor az, hogy a TSP-ben a Hamilton kör súlyát akarjuk minimalizálni, ami közvetlenül az élekből jön. Ezzel szemben a VRP esetében az élek az út fizikai tulajdonságát, forgalmi adatait, forgalmi szabályait rögzítik. A forgalmi adatok és a szabályzás ráadásul idő függő is lehet. A távolság helyett időre vagy közvetlenül pénzre optimalizálunk. Ez a költség számítható a kiszolgálási sorrendből és kiosztásból egy polinomiális idejű szimulációval. A mi általunk használt szimuláció relatív egyszerű így $O(c)$ idő alatt lefut egy szállítási tervre.

Algoritmus tipológia

Úgynevezett heurisztikákat kell alkalmaznunk, melyek úttervezési problémák esetén két főbb csoportra oszthatóak. Az egyik csoport az útterv generáló algoritmusok, a másik pedig az iteratív optimalizációk.

A szállítás generálók egyetlen szállítási tervet generálnak, melyeket lokálisan optimális döntések alapján építenek fel. Ilyen például a „legközelebbi szomszéd”[8] vagy a „mohó”[8] módszer. Előbbi az összes kamiont elindítva úgy választ ki ügyfelet a kiszolgálásban következőnek és úgy rendeli szállítási egységhez, hogy a szállítás addigi költsége minimálisan növekedjen. Utóbbi veszi az ügyfelek közötti útszakaszokat és mindig beveszi a következő olyan útszakaszt, melynek költsége minimális és nem eredményez olyan kört, ami ne menne át a szállítási központon.

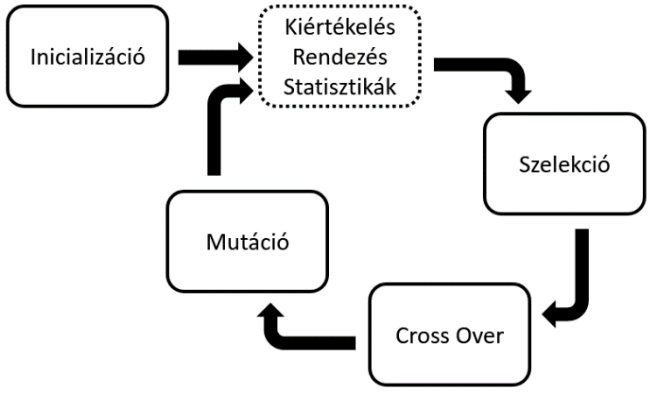
Az iteratív optimalizációk további két csoportra bonthatóak. Az egyik a lokális keresések, a másik pedig a globális keresések. A lokális keresések egyszerre egy szállítási tervet tartanak számon és azon igyekeznek javítani végig vizsgálva ahhoz hasonló szállításokat és lecserélve az eredetit annál kisebb költségűre. Itt is megjelenik a lokális minimumok problémája, avagy, ha a szállítási tervet csak nála jobb szállítási tervre cseréljük, akkor könnyen juthatunk olyan szállítási tervhez, amin olyan drasztikusan kell változtatni a javításhoz, hogy az algoritmus tévesen optimumnak ítéli. Ezért a fejlettebb algoritmusok valamilyen eséllyel elfogadnak költségesebb terveket is. Ilyen algoritmus például a K-Opt[11] csoport, a Lin-Kernighan[10] és a „Branch and Bounds”[9] is.

A globális keresések jelentős részét teszik ki a természeti jelenségeken alapuló algoritmusok. Ezeknek két nagyobb családja az evolúciós és a raj intelligenciát alkalmazó algoritmusok. A raj intelligenciát[4] [5] alkalmazó algoritmusoknál több ágens folytat keresést párhuzamosan, és egymással megosztott részinformációk alapján javítanak eredményeiken. Mi az evolúciós algoritmusokra helyeztük a hangsúlyt, mivel ígéretes teljesítményt nyújtottak az utazó ügynök problémán, amin a jármű úttervezési probléma is alapszik.

Az evolúciós algoritmusok[12] lényege, hogy a megoldás halmazt úgy kezeljük, mint ha egy faj lenne. Ekkor az egyszerre számontartott megoldásokat úgy kezeljük, mint ennek a fajnak a populációja. Minden iterációban a populáción szelekciót hajtunk végre, ahol annak egy részét megsemmisítjük. Ekkor nagyobb túlélési esélyt biztosítunk a kisebb költségű megoldásoknak. Az iteráció másik lépése a mutáció, ahol új példányokat képzünk és meglévőket módosítunk véletlenszerűen. A szelekciós

lépés biztosítja azt, hogy a folyamat egy optimumhoz konvergáljon. Ha nem halnak ki a populációból azok az elemi tulajdonságok, amik az optimális példányt alkotják, akkor jó eséllyel megtalálja azt, csupán idő kérdése.

A genetikus algoritmusok[6] [7] az evolúciós algoritmusok legelterjedtebb csoportja. A szelekció és mutáció közé egy új lépést hoz be az ötvözést (crossover). A crossover lépésben a megtartott példányokból generálódnak újak azok tulajdonságainak ötvözésével, ekkor a szelektált példányokat szülőknek, míg a generáltakat azok gyerekeinek is nevezzük. Az új példányok jó valószínűséggel hasonló költségűek, mint a szüleik, hiszen azok tulajdonságaiból lettek alkotva. A standard esetben minden két példányból két új generálódik a populációt megduplázva. A legnagyobb probléma a genetikus algoritmusokkal azok sebessége. Mivel mint a szelekció, crossover, és a mutáció véletlen folyamatok, a javulás egyre kisebb valószínűséggel következik be. Ezért szokták az evolúciós algoritmusokat kiegészíteni lokális kereséssel, ami biztosítja az állandó konvergenciát. Az így kiegészített evolúciós algoritmusokat hívják memetikus algoritmusoknak.



1. ábra Genetikus algoritmus

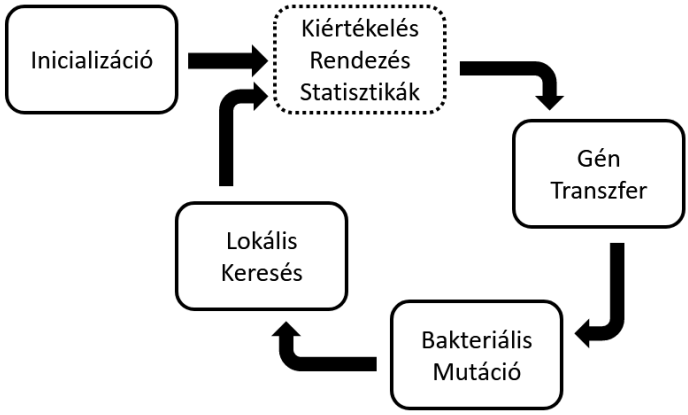
Alkalmazott algoritmus

Az általunk alkalmazott algoritmus is egy memetikus algoritmus, de nem genetikus, hanem bakteriális. A bakteriális evolúciós algoritmusok[1] a genetikus algoritmusoknál modernebbek, de még közel sem annyira elterjedtek. Megjelenésük nem csak a genetikusok továbbfejlesztését, de szemlélet váltást is jelentenek. A bakteriális evolúciós algoritmusokban ugyanis eltűnik a szelekció lépése és a crossovert felváltja a gén transzfer.

A gén transzfer lépésben nem több szülőből generálódnak új példányok, hanem gyengébben teljesítő példányokba szúrunk elemi tulajdonságokat a jobban teljesítő példányokból. Ez azt eredményezi, hogy a legjobban teljesítő példányok nem kerülhetnek felülírásra és húzóerőként hatnak a populáció másik felére.

A mutáció is jelentősen módosult, úgynevezett bakteriális mutációra lett cserélve. Ennek során minden mutálódó példányból másolatok úgynevezett klónok készülnek, majd mindegyik másolaton véletlen mutációt hajtunk végre. Az algoritmus vagy a legjobb vagy a második legjobb klónnal[1] írja fölül az eredetit. Utóbbit azért engedi meg valamilyen valószínűséggel, hogy elkerülje a lokális minimumot.

A mutációt követően kerül sor a lokális keresésre. Erre a 2-Opt algoritmus egy egyszerűsített változatát alkalmaztuk. A lokális keresés végrehajtódik a legjobb példányon és az összes többire 10% eséllyel. Azért fontos, hogy a többi példányt is javítsuk, hogy ne legyen az első példány túl domináns. Ha túl sokáig ugyanaz a példány a legerősebb, akkor egyre több példány fog rá hasonlítani és a hibái is dominálni fogják a populációt. Erre a jelenségre vezettük be a *belterjesség* fogalmat.



2. ábra Bakteriális Memetikus Algoritmus

A továbbiakban a lépések működését részletesebben is leírjuk. Ehhez szükséges az adat formátumról értekeznünk. A mi implementációnkban a permutáció és kiosztás úgynevezett egy kromoszómás reprezentációját használtuk. Ennek a lényege, hogy ahelyett, hogy egy *c* hosszú permutációt felvágtunk volna *s* szakaszra, törés pontokat szúrtunk a permutációba, melyeket *c*-től számoztunk. Így lényegében egy nagy *c*+*s* hosszú permutációra egészítjük ki. Ez ugyan azt eredményezi, hogy több reprezentáció (akár *s*!) megfeleltethető egy szállítási tervnek, de messze menően könnyebbé teszi a kezelést. Felmerül a kérdés, hogy ekkor mit tekintünk elemi tulajdonságnak, génnek. Alapértelmezetten a permutáció elemeit szoktuk, de jogos megfigyelés az is, hogy fontosabb az, hogy melyik elem melyiket követi közvetlenül, minthogy mi a pozíciójuk.

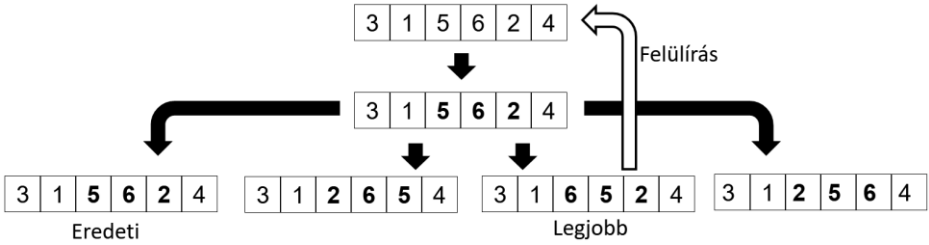
	4	2	3	6	5	1	
4	2	*	3	6	*	5	1
4	2	**7**	3	6	**8**	5	1

3. ábra Egy kromoszómás reprezentáció

A gén transzfer során a populációt ketté választjuk költség szerint, és párokba rendezzük úgy, hogy a populáció legkisebb költségű elemét a legköltségesebb elemmel párosítjuk, a második legkisebb költségűt a második legköltségesebbel és így tovább. Ekkor a kisebb költségűt *donornak* és a költségesebbet *alanynak* is nevezzük. Ezt követően egy véletlen

szegmenset választunk adott hosszal a donorból. A szegmenset azonos pozícióba szúrjuk be az alanyba. Fontos, hogy a szegmens összefüggő kell legyen, hiszen a költség az útszakaszoktól függ jobban, nem az elemek pozíciójától.

A mutáció során minden példányt mutálunk. Adott a klónok száma és a mutált szegmens mérete. Minden mutációban választunk egy véletlen pozíciót, ami minden klónban a kiválasztott szegmens pozíciója lesz. Minden klónban a szegmens elemeit megkeverjük. Ezt követően minden klónt kiértékelünk és kiválasztunk egyet közülük, amivel felülírjuk az eredeti példányt. Minden példányon többször is végrehajtjuk a mutációt.



4. ábra Bakteriális mutáció

A lokális keresésnél végig iterálunk minden két elemen a javított példányban. Megcseréljük a két pozíció tartalmát, kiértékeljük a példányt és ha jobb a költsége, akkor felülírjuk az eredeti példányt. Az eredeti 2-Opt algoritmusban a két pozíció közötti szakaszt invertálni kell, de ennek költsége túl magas volt, ezért döntöttünk az egyszerűbb megoldás mellett. Ezen kívül az eredeti algoritmus újra és újra végignéz minden két pozíciót, amíg talál javítást, míg a mi megoldásunk egyetlen egyszer iterál végig. Ha minden elemet teljesen optimalizálnánk, akkor az algoritmus beragadna egy lokális minimumba.

Úgy becsültük[2] és a mérések is azt igazolják, hogy az algoritmus $O((c+s)^{4.5})$ futási idejű. Igaz ezt nagyban befolyásolja a paraméterezés. Mivel az ügyfelek és járművek számától függ a komplexitás, ezért a paramétereket is ennek függvényében választottuk meg. $\sqrt{c+s}$ a populáció mérete. A gén transzfer során $\sqrt{c+s}$ a szekvencia hossza. A mutáció során $\sqrt{c+s}$ a klónok száma és a mutált szekvencia hossza. A tesztek során $c+s$ iterációt futtattunk. Az inicializáció $O((c+s)^{1.5})$ időt vesz igénybe, mivel minden példány $O(c+s)$ idő alatt készül el. A gén transzfer $O((c+s)^{1.5})$ időt vesz igénybe, mivel a populáció felén végig kell iterálni és a donorból az alanyba $O(c+s)$ átírni a géneket,

mivel az alany egészét is érintheti a változás. A bakteriális mutáció már több, $O((c+s)^{2.5})$ processzor időt használ, mivel a populáció minden elemére elkészíti a klónokat, azokat ki is értékeli és ezt megismétli $\sqrt{c+s}$-szer. A legtöbb futási időt mégis a lokális keresés igényli, ahol a $c+s$ pozíció minden párján végig iterál és minden iterációban ki is értékeli az új példányt. Ez $O((c+s)^{3.5})$-ös futási időt jelent. Ez pedig minden iteráció futási idejét is $O((c+s)^{3.5})$-é teszi. Ezt végezetül az iterációk számával szoroztuk fel. Ezt mi viszont $O((c+s)^4)$-re mérsékeltük azzal, hogy a lokális keresés csak minden a $\sqrt{c+s}$ iterációban fut le.

Megfigyelések és következtetések

Az algoritmus jól teljesített kisebb feladat méreteken. Harminc ügyfél esetén mindig megtalálta az optimumot. Ötven ügyfélre már nem tudtunk optimumot számítani, de mindig ugyan azt a szállítási tervet adta eredményül azonos bemeneti adatra így élünk a feltételezéssel, hogy az optimumot találta meg. Ebben az is megerősít minket, hogy más algoritmusokkal sem sikerült jobb eredményre jutnunk. Száz ügyfélnél már változó eredményekre jutottunk. Ha hasonló értékűeket is, de más-más eredményeket kaptunk futtatásonként. Ezer ügyfélen már jelentős volt az algoritmus futási ideje, így nem tudtunk kellő mennyiségű adatot gyűjteni.

Összehasonlítottuk több heurisztikával, mint a legközelebbi szomszéd (Nearest Neighbour), mohó módszer (Greedy), egyszerűsített 2-Opt, 2-Opt. Mindegyiknél jobb eredményeket hozott az összes teszt adaton.

A standard genetikus és a memetikus algoritmussal is összevetettük. Ezekkel azonos minőséget hozott kevesebb iteráció alatt. Az algoritmus gyengeségei csak nagyobb bemeneteken mutatkoztak. A bemenet méretének függvényében ugyanis gyorsabban vesztett hatékonyságából, mint a genetikus algoritmusok. Genetikus algoritmusokkal TSP-n is összevetettük, ahol szintén megfigyelhető volt a jelenség, de sokkal enyhébb volt. Viszont, amikor eltávolítottuk a lokális keresést, akkor mindkét problémára majdnem megszűnt. Arra jutottunk, hogy a problémát első sorban a példányok kiértékelésének, a lokális keresésnek és a bakteriális mutációnak a futási ideje okozza.

Miután a mutált szekvencia méretét, a klónok számát és az ismétlés mennyiségét is konstans értékekre állítottuk, a bakteriális mutációval kapcsolatos sebesség problémák megoldódtak. Ugyanakkor az iterációnkként javulás is jelentősen esett. A futási idő és hatékonyság

egyensúlyának megtalálása további kutatást igényel. Ami a lokális keresést illeti, tovább korlátoztuk, hogy az első pozíció pár után, amire javulást talál álljon le. Ezt továbbfejlesztettük úgy, hogy a pozíciókat véletlen sorrendbe vegye, ezzel elkerülhető, hogy a lokális keresés a példányok elejére korlátozódjon. Ugyanakkor indokolt lehet más lokális keresések vizsgálata és integrációja is.

Összefoglalás

A jármű ütemezési probléma a mai gazdaság egyik legnagyobb központi problémája. Mivel NP-beli, ezért direkt megoldása a legtöbb esetben lehetetlen reális időkorlát mellett.

A bakteriális memetikus algoritmus egy fejlett iteratív optimalizáció, ami hatékony megoldás lehet. Az algoritmust adaptáltuk a problémára és megkezdtük finom hangolását is. Kisebb feladat méretekre hatékony, sőt optimális, ugyanakkor az ezres nagyságrendben már akadályokba ütközik. Ezek többnyire a futásidőt és azzal arányos költség csökkenést érintik.

A célunk a jövőre nézve: Példányok kiértékelésének erőforrásigényének csökkentése. A mutációs lépés továbbfejlesztése és finomhangolása. Megoldások átemelése fejlettebb genetikus algoritmusokból. Alternatív lokális keresések tesztelése és optimalizációja.

Felhasznált Irodalom:

[1] BÓDIS, T. – BOTZHEIM, J. (2018): Bacterial memetic algorithms for order picking routing problem with loading constraints. Expert Systems with Applications, 105:196–220.

[2] CORMEN, T. H. – LEISERSON, C. E. – RIVEST, R. L. – STEIN. C. (2009): Introduction to Algorithms Third Edition, Massachusetts Institute of Technology.

[3] DANTZIG, G. B. – RAMSER, J. H. (1959): The Truck Dispatching Problem. Management Science. 6(1): 80–91.

[4] DORIGO, M. – GAMBARDELLA, L. M. (1997): Ant colony system: A cooperative learning approach to the traveling salesman problem. IEEE Transactions on Evolutionary Computation, 1(1):53–66.

[5] DORIGO, M. – MANIEZZO, V. – COLORNI, A. (1996): Ant system: Optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics-Part B Cybernetics, 26(1):29–41.

[6] GOLDBERG, D. E. (1989): Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley PC.
[7] HOLLAND J. H. (1992): Adaption in natural and artificial systems. The MIT Press.
[8] JOHNSON, D. S. – MCGEOCH, L.A. (1995): The Traveling Salesman Problem: A Case Study in Local Optimization.
[9] LAND, A. H. – DOIG, A. G. (1960): An automatic method of solving discrete programming problems. Econometrica, 28(3):497–52.
[10] LIN, S. – KERNIGHAN, B. W. (1973): An effective heuristic algorithm for the traveling salesman problem. Operations Research, 21(2):498–516.
[11] PAPADIMITRIOU, C. H. – STEIGLITCZ, K. (1982): Combinatorial Optimization: Algorithm and Complexity. Prentice-Hall, Englewood Cliff, NJs.
[12] VIKHAR, P. A. (2016): Evolutionary algorithms: A critical review and its future prospects. Proceedings of the 2016 International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC). Jalgaon: 261–265.