Lokalna optimizacija problema putujućeg trgovca s iznajmljivanjem automobila

Španić, Matija

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Matija Španić

LOKALNA OPTIMIZACIJA PROBLEMA PUTUJUĆEG TRGOVCA S IZNAJMLJIVANJEM AUTOMOBILA

DIPLOMSKI RAD

Varaždin, 2023.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Matija Španić

Matični broj: 45064/16-R

Studij: Informatika u obrazovanju

LOKALNA OPTIMIZACIJA PROBLEMA PUTUJUĆEG TRGOVCA S IZNAJMLJIVANJEM AUTOMOBILA

DIPLOMSKI RAD

Mentor:

Izv. prof. dr. sc. Nikola Ivković

Varaždin, rujan 2023.

Matija Španić

Izjava o izvornosti

Izjavljujem da je moj diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

_____

Sažetak

U ovom se radu istražuje problem putujućeg trgovca s iznajmljivanjem automobila (CaRS), koji je NP-težak kombinatorni računalni problem. Cilj istraživanja je implementirati algoritam lokalne optimizacije za rješavanje problema CaRS te eksperimentalno istražiti karakteristike algoritma na različitim primjerima problema. U uvodu se ukazuje na NP-težak karakter problema CaRS i nedostatak egzaktne metode za rješavanje u razumnom vremenu. Stoga se istraživanje fokusira na metodama iterativne lokalne pretrage i tehnike mekog računarstva kao mogućim pristupima rješavanju problema.

U metodologiji istraživanja se daje pregled metoda lokalne optimizacije, odabire se odgovarajuća metoda za implementaciju algoritma te opisuju podaci korišteni u eksperimentima. Teorijski aspekt rada analizira karakteristike problema putujućeg trgovca, izazove koji proizlaze iz uključivanja iznajmljivanja automobila, matematički model problema CaRS te različite algoritme lokalne optimizacije primjenjive na taj problem. Nakon toga slijedi analiza teorijskih metoda i njihovih prednosti i nedostataka.

Implementacija odabrane metode lokalne optimizacije opisuje se u poglavlju o praktičnoj primjeni, zajedno s eksperimentalnom metodologijom. Rezultati eksperimenata se analiziraju, a također se uspoređuju s drugim metodama rješavanja problema CaRS. Na kraju, zaključak sumira glavne rezultate istraživanja, ističe važnost rada i njegov doprinos te sugerira mogućnosti za daljnje istraživanje i poboljšanje algoritma.

Ključne riječi: algoritam, eksperimentalno istraživanje, iznajmljivanje automobila, NP-težak problem, optimizacija problema putujućeg trgovca

Sadržaj

1. Uvod

U kontekstu problema putujućeg trgovca, javlja se varijanta koja uključuje iznajmljivanje automobila. Problem putujućeg trgovca s iznajmljivanjem automobila (CaRS) dodaje dodatne složenosti, kao što su dodatni troškovi iznajmljivanja, kapacitet vozila te vremenska ograničenja za povratak automobila. Integriranje ovih dodatnih varijabli u matematički model problema predstavlja izazov koji zahtijeva razvoj učinkovitih algoritama za rješavanje CaRS problema.

U radu usredotočit će se se na primjenu metoda lokalne optimizacije za rješavanje problema putujućeg trgovca s iznajmljivanjem automobila. Metode lokalne optimizacije temelje se na iterativnom poboljšavanju trenutnog rješenja putem pretrage susjedstva. Takve metode često postižu dobre rezultate za različite probleme kombinatorne optimizacije. Nadalje, razvijat će se novi algoritam lokalne optimizacije specifično prilagođen za CaRS problem. Kroz ovaj algoritam, cilj je postići što bolje rješenje putujućeg trgovca s iznajmljivanjem automobila, uzimajući u obzir sve troškove, ograničenja kapaciteta vozila i vremenske aspekte.

Svrha ovog rada je ne samo implementirati i testirati algoritam lokalne optimizacije, već i usporediti dobivene rezultate s neoptimiziranim rješenjima, kao i s rezultatima dobivenim primjenom drugih metoda rješavanja CaRS problema. Kroz detaljno testiranje, analizu rezultata i usporedbu s postojećim metodama, moći ćemo bolje razumjeti učinkovitost i prednosti implementiranog algoritma lokalne optimizacije u rješavanju CaRS problema.

Također, razmotrit će se mogućnosti daljnjeg unaprjeđenja algoritma te njegova primjenjivost na različite veličine i složenosti problema iznajmljivanja automobila. U tom smislu, ovaj rad će pružiti doprinos u polju kombinatorne optimizacije i otvoriti nove perspektive za daljnja istraživanja u ovom području.

1.1. Predmet istraživanja

Predmet istraživanja ovog rada je primjena metoda lokalne optimizacije u rješavanju problema putujućeg trgovca s iznajmljivanjem automobila (CaRS). Problem putujućeg trgovca s iznajmljivanjem automobila zahtijeva pronalazak najkraćeg mogućeg puta koji putnik treba proći kako bi posjetio određeni skup gradova i vratio se u polazni grad, uz dodatne faktore poput troškova iznajmljivanja automobila, kapaciteta vozila te vremenskih ograničenja za povratak automobila. Ova varijanta problema predstavlja izazov koji zahtijeva razvoj prilagođenih algoritama i strategija za njegovo rješavanje.

1.2. Ciljevi istraživanja i istraživačka pitanja

Ciljevi istraživanja uključuju:

- Proučavanje metoda lokalne optimizacije i njihove primjenjivosti na problem CaRS;
- Analizu izazova koji se pojavljuju prilikom integriranja iznajmljivanja automobila u problem putujućeg trgovca;
- Evaluaciju performansi različitih metoda lokalne optimizacije u rješavanju problema CaRS;
- Razvoj učinkovitih strategija i algoritama za pronalazak optimalnih rješenja problema CaRS.

Ovaj diplomski rad će pridonijeti boljem razumijevanju složenosti i specifičnosti problema putujućeg trgovca s iznajmljivanjem automobila te pružiti smjernice za primjenu metoda lokalne optimizacije u rješavanju ovog problema.

U okviru istraživanja problema putujućeg trgovca s iznajmljivanjem automobila, postavljaju se sljedeća istraživačka pitanja:

1. Kako se metode lokalne optimizacije mogu prilagoditi za rješavanje problema putujućeg trgovca s iznajmljivanjem automobila?
2. Koji su specifični izazovi i ograničenja koji proizlaze iz integracije iznajmljivanja automobila u problem putujućeg trgovca?
3. Koje metode lokalne optimizacije su najprikladnije za rješavanje problema CaRS?
4. Kako se performanse različitih metoda lokalne optimizacije uspoređuju u kontekstu problema putujućeg trgovca s iznajmljivanjem automobila?
5. Kako se mogu razviti efikasne strategije i algoritmi koji će pružiti optimalna rješenja za problem CaRS?

1.3. Struktura rada

Rad je strukturiran na sljedeći način kako bi se sistematično istražile metode lokalne optimizacije za rješavanje problema putujućeg trgovca s iznajmljivanjem automobila (CaRS) i evaluirale njihove performanse:

1. Uvod: U ovom dijelu rada postavljeni su kontekst istraživanja, predmet istraživanja, ciljevi i istraživačka pitanja. Također je navedena struktura rada kako bi čitatelj dobio pregled sadržaja.
2. Metode i tehnike rada: Ovo poglavlje pruža pregled metoda lokalne optimizacije koje su relevantne za rješavanje problema putujućeg trgovca s iznajmljivanjem automobila. Također se istražuje odabir metode za implementaciju i opisuje skup podataka koji će se koristiti za eksperimente.
3. Optimizacija problema putujućeg trgovca: U ovom poglavlju analiziraju se karakteristike problema putujućeg trgovca i specifični izazovi koji proizlaze iz integracije iznajmljivanja automobila u problem. Također se istražuju algoritmi lokalne optimizacije primjenjivi na problem CaRS.
4. Implementacija odabrane metode lokalne optimizacije: U ovom dijelu rada opisuje se eksperimentalna metodologija koja je korištena za implementaciju odabrane metode lokalne optimizacije.
5. Implementacija algoritma lokalne optimizacije: Prikazuje se konačno rješenje te naposlijetku testiranje i rezultati istraživanja.
6. Zaključak: U zaključnom dijelu rada sumiraju se glavni rezultati istraživanja i daju se odgovori na postavljena istraživačka pitanja.

2. Metode i tehnike rada

Opisane su metode i tehnike rada koje će se koristiti za pisanje diplomskog rada.

2.1. Pregled metoda lokalne optimizacije

U okviru istraživanja problema putujućeg trgovca s iznajmljivanjem automobila (CaRS), u ovom radu se koriste metode lokalne optimizacije za pronalaženje boljih rješenja. Metode lokalne optimizacije su popularan pristup u rješavanju kombinatoričkih problema poput problema putujućeg trgovca.

Analiziraju se različite metode lokalne optimizacije koje su prikladne za rješavanje problema CaRS. Metode lokalne optimizacije usredotočuju se na pretraživanje i poboljšanje pojedinačnih rješenja unutar prostora pretraživanja. One nastoje pronaći optimalno rješenje ili lokalni optimum u blizini trenutno promatranog rješenja.

Pregled metoda lokalne optimizacije obuhvaća različite tehnike kao što su simulirano kaljenje, lokalna pretraga, iterativno poboljšavanje, tabu pretraživanje, genetsko lokalno pretraživanje, itd. Svaka od ovih metoda ima svoje karakteristike, prednosti i nedostatke.

U kontekstu problema putujućeg trgovca s iznajmljivanjem automobila, posebna pažnja će biti posvećena metodi lokalne optimizacije koja se naziva memetički algoritam. Memetički algoritam kombinira evolucijski pristup genetskih algoritama s lokalnom optimizacijom kako bi se postigla bolja kvaliteta rješenja. Ova kombinacija omogućuje raznolikost pretrage prostora rješenja putem genetskih operatora, dok se istovremeno primjenjuje lokalna optimizacija za daljnje poboljšanje rješenja.

Kroz detaljan pregled metoda lokalne optimizacije, istražuje se njihova primjenjivost na problem CaRS i uspoređuju se njihove performanse u pronalaženju kvalitetnih rješenja. Na temelju analize rezultata eksperimenata, odabrat će se najprikladnija metoda za implementaciju i daljnje istraživanje

.

2.2. Odabir metode za implementaciju

Pri odabiru metode za implementaciju za rješavanje problema CaRS, važno je uzeti u obzir karakteristike problema, dostupne resurse i ciljeve istraživanja. U radu će se analizirati prednosti i ograničenja metode i odabrana je ona koja najbolje odgovara postavljenim

zahtjevima. Također se uzima u obzir efikasnost algoritma, skalabilnost, robusnost te mogućnost prilagodbe specifičnim uvjetima problema putujućeg trgovca s iznajmljivanjem automobila.

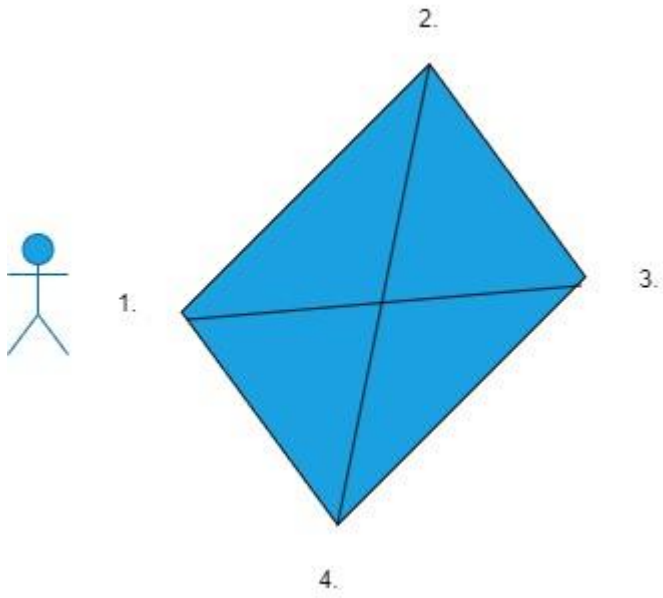
2.3. Opis podataka za eksperimente

Za eksperimentiranje je određen specifičan broj automobila i broj gradova koji su definirani kao datoteke s nastavkom *.car* (Github, repozitorij Elvisa Popovića). Za primjer s manje automobila pogodan je ACO_Cars (eng. *Ant Colony Optimization*) koji je implementiran u praktičnom dijelu ovog rada, a za primjere s više automobile pogodniji je trasngenetski algoritam.

3. Optimizacija problema putujućeg trgovca

Problem putujećeg trgovca (eng. Traveling Salesman Problem, TSP) je dobro proučavan i poznat kombinatorni optimizacijski problem koji spada u skupinu NP-teških problema. Ovaj problem predstavlja izazov za tradicionalne metode rješavanja. Matematički problemi slični problemu trgovačkog putnika počeli su se razmatrati već u 18. stoljeću, a pojam "trgovački putnik" prvi put je upotrijebljen 1932. godine (Goldbarg i sur., 2012). „Problem putujućeg trgovca sastoji se u odabiru najkraćeg puta za obilazak N gradova i povratak u početni grad. Za svaki grad definirana je njegova lokacija tako da je moguće izračunati udaljenost između gradova“ (Korenić, 2019, str. 40).

Slika 1 prikazuje jednostavni primjer problema putujućeg trgovca. Na slici je četverokut s četiri točke koje predstavljaju četiri različita grada, označena brojevima 1, 2, 3 i 4. Pored četverokuta se nalazi crtež čovjeka, koji simbolizira putujućeg trgovca. Cilj problema putujućeg trgovca je pronaći najkraću moguću putanju koja će putujućem trgovcu omogućiti da posjeti sve gradove jednom, i na kraju se vrati nazad u početni grad. Ova putanja treba minimizirati ukupnu dužinu putovanja, uzimajući u obzir udaljenosti između gradova. Slika ilustrira jedan od osnovnih scenarija TSP-a s četiri grada, ali problem se može proširiti na veći broj gradova i složenije mreže puteva. Rješavanje problema putujućeg trgovca predstavlja izazov za optimizacijske algoritme i ima široku primjenu u različitim područjima poput logistike, planiranja rute, proizvodnje i raspoređivanja resursa.



Slika 1: Problem putujućeg trgovca

Izvor: vlastita izrada

Problem trgovačkog putnika može se formulirati kao pronalaženje najkraćeg puta koji posjećuje određeni broj gradova točno jednom, a zatim se vraća u početni grad. Ovaj problem ima široku primjenu u različitim područjima, poput logistike, planiranja ruta, raspoređivanja poslova i analize podataka. Broj mogućih redoslijeda obilaska gradova raste faktorijelno s brojem gradova, što dovodi do ogromnog prostora pretraživanja (Goldbarg i sur., 2012).

Zbog svoje složenosti, genetski algoritmi su postali popularni pristup rješavanju problema trgovačkog putnika. Genetski algoritmi su stohastičke metode pretraživanja koje oponašaju prirodni tijek biološke evolucije. Oni započinju s populacijom rješenja i iterativno generiraju nove generacije rješenja koristeći genetske operatore poput križanja i mutacije. Kroz generacije, genetski algoritmi konvergiraju prema boljim rješenjima, ali ne jamče optimalno rješenje (Goldbarg i sur., 2012).

Uz problem trgovačkog putnika, postoje i slični problemi koji se mogu svesti na njega. Primjeri uključuju problem pronalaženja najmanjeg Hamiltonovog ciklusa u težinskom grafu, problem trgovačkog putnika s uskim grlom i opći problem trgovačkog putnika (Goldbarg i sur., 2012). Ovi problemi imaju različite varijacije i uvjete, ali dijele zajedničku karakteristiku potrage za optimalnim putem ili ciklusom koji posjećuje određene čvorove s određenim ograničenjima.

Složenost problema trgovačkog putnika čini ga izazovnim za rješavanje, ali i dalje je predmet istraživanja kako bi se pronašle učinkovite metode i strategije. Genetski algoritmi su jedan od pristupa koji je pokazao obećavajuće rezultate u rješavanju problema trgovačkog putnika i njemu sličnih problema, ali i dalje ostaje otvoreno pitanje pronalaska idealnog rješenja (Goldbarg i sur., 2012).

3.1. Karakteristike problema putujećeg trgovca

Problem putujećeg trgovca (Traveling Salesman Problem, TSP) je dobro proučavan i poznat kombinatorni optimizacijski problem (Goldbarg et al., 2012). U ovom poglavlju istražuju se ključne karakteristike problema putujećeg trgovca, poput definicije problema, cilja optimizacije i ograničenja. TSP uključuje pronalaženje najkraćeg mogućeg puta koji posjećuje određeni broj gradova točno jednom, a zatim se vraća u početni grad. U kontekstu problema putujećeg trgovca s iznajmljivanjem automobila (CaRS), dodaje se dodatni aspekt iznajmljivanja automobila koji može utjecati na optimalno rješenje.

Ključne karakteristike problema putujećeg trgovca uključuju (Goldbarg i sur., 2012):

1. Definicija problema: Problem putujećeg trgovca sastoji se od skupa gradova i udaljenosti između njih. Potrebno je pronaći optimalno rješenje, odnosno redoslijed posjeta gradovima koji minimizira ukupno putovanje.
2. Cilj optimizacije: Cilj je pronaći najkraći mogući put koji posjećuje sve gradove i vraća se u početni grad. To znači minimizirati ukupnu duljinu puta, odnosno sumu udaljenosti između gradova.
3. Ograničenja: Ograničenje problema putujećeg trgovca je da svaki grad mora biti posjećen točno jednom, osim početnog grada koji se posjećuje na početku i na kraju puta. Također, ne smiju se posjećivati više puta isti grad.

U kontekstu problema putujećeg trgovca s iznajmljivanjem automobila (CaRS), dodaje se dodatni aspekt iznajmljivanja automobila koji može utjecati na optimalno rješenje. Osim pronalaženja najkraćeg puta između gradova, potrebno je uzeti u obzir i vrijeme i trošak iznajmljivanja automobila, što dodaje složenost u postizanju optimalnih rješenja.

Kroz empirijsku analizu, Johnson i McGeoch (1997) prikazuju rezultate eksperimenata koji pokazuju kako se različite metode lokalne optimizacije ponašaju u praksi. To uključuje usporedbu vremenske složenosti, kvalitete pronađenih rješenja i njihove udaljenosti od globalnog optimuma.

3.2. Izazovi uključivanja iznajmljivanja automobila

Uključivanje iznajmljivanja automobila u problem putujećeg trgovca predstavlja dodatne izazove (Goldbarg i sur., 2012). Iznajmljivanje automobila uključuje dodatne troškove, ograničenja kapaciteta vozila i vremenska ograničenja za povratak automobila. Ovi faktori mogu značajno utjecati na optimalno rješenje problema CaRS. Važno je razumjeti kako integrirati ove dodatne varijable u matematički model kako bismo razvili učinkovite algoritme za rješavanje problema CaRS.

Prema Rios i sur. (2017) iznajmljivanje automobila donosi dodatne troškove koji treba uzeti u obzir prilikom traženja optimalne rute. Svako iznajmljivanje automobila ima svoju cijenu, a ovi troškovi trebaju biti minimizirani u cilju pronalaska najekonomičnije rute. Integracija ovih troškova u matematički model problema CaRS zahtijeva razmatranje kako dodatni trošak utječe na odabir puta i kako se može optimizirati. Kapacitet vozila također je faktor koji treba uzeti u obzir prilikom planiranja rute. Vozilo ima ograničen kapacitet i treba pravilno rasporediti posjete gradovima kako bi se osiguralo da se ne prekoračuje kapacitet vozila. Ovo predstavlja

izazov jer je potrebno pronaći optimalno raspoređivanje gradova s obzirom na njihove udaljenosti i zahtjeve za kapacitetom.

Vremenska ograničenja za povratak automobila također su važna za uzeti u obzir. Iznajmljivanje automobila obično ima određeno vremensko ograničenje za povratak vozila. Ovo vremensko ograničenje može ograničiti mogućnosti rute i zahtijeva pravilno planiranje kako bi se vozilo vratilo u vremenskom okviru. Ova ograničenja dodatno otežavaju problem putujećeg trgovca s iznajmljivanjem automobila i zahtijevaju razvoj sofisticiranih algoritama za efikasno rješavanje problema. Razumijevanje i integracija ovih dodatnih varijabli u matematički model problema CaRS ključno je za razvoj efikasnih algoritama za rješavanje problema putujećeg trgovca s iznajmljivanjem automobila. Kroz analizu i istraživanje ovih izazova, može se pružiti dublji uvid u kompleksnost problema CaRS i razviti napredne strategije i metode koje će omogućiti pronalazak optimalnih rješenja, navode Rios i sur. (2017)

U tablici 1 su prikazani glavni izazovi koji proizlaze iz uključivanja iznajmljivanja automobila u problem putujećeg trgovca. Svaki izazov ima svoj opis koji ukazuje na specifičnost problema i kako utječe na rješenje problema CaRS. Utjecaj na rješenje CaRS odnosi se na to kako svaki izazov može utjecati na pronalazak optimalnog rješenja, ekonomičnost rute, raspoređivanje gradova ili planiranje rute.

Tablica 1: Izazovi uključivanja iznajmljivanja automobila u problem putujećeg trgovca

Izazovi uključivanja iznajmljivanja automobila	**Opis**	**Utjecaj na rješenje CaRS**
Dodatni troškovi	Troškovi iznajmljivanja automobila	Utjecaj na ukupni trošak putovanja
Ograničenja kapaciteta vozila	Ograničenja kapaciteta vozila	Utjecaj na optimalnu raspodjelu resursa
Vremenska ograničenja za povratak automobila	Ograničenja vremena za povratak vozila	Utjecaj na efikasnost rute i vremenski plan
Dinamičke promjene cijena i dostupnosti	Promjene cijena i dostupnosti automobila	Utjecaj na planiranje i optimizaciju ruta
Lokacije za preuzimanje i vraćanje vozila	Različite lokacije za preuzimanje i vraćanje vozila	Utjecaj na dizajn rute i fleksibilnost

Izvor: vlastita izrada; prema Rios i sur. (2017)

3.4. Algoritmi lokalne optimizacije primjenjivi na problem CaRS

Prikazat će poznati algoritma lokalne optimizacije, memetički algoritam i simulirano kaljenje koji su primjenjivi na problem CaRS.

3.4.1. Memetički algoritam

U kontekstu problema CaRS, memetički algoritam ističe se kao metoda lokalne optimizacije koja kombinira genetski algoritam s lokalnom pretragom (Goldbarg i sur, 2012). Memetički algoritam koristi kombinaciju globalne pretrage putem genetskog algoritma i lokalne pretrage koja poboljšava lokalnu kvalitetu rješenja. Ovaj pristup omogućuje istraživanje različitih rješenja u prostoru pretraživanja te pronalaženje optimalnog rješenja za problem CaRS. Memetički algoritam za rješavanje problema putujućeg trgovca s iznajmljivanjem automobila (CaRS) predstavlja inovativan pristup koji se temelji na prirodnom procesu evolucije. Genetski algoritam, kao dio memetičkog algoritma, koristi operatore selekcije, križanja i mutacije kako bi generirao populaciju potencijalnih rješenja. Ta populacija predstavlja skup putanja putujućeg trgovca koje uključuju iznajmljivanje automobila. Svaki član populacije predstavlja jedno rješenje problema CaRS.

Nakon stvaranja početne populacije, genetski algoritam primjenjuje operator selekcije kako bi odabrao najbolje jedinke koje će preživjeti i biti korištene za generiranje sljedeće generacije. Kroz iterativni postupak, genetski algoritam kombinira informacije iz prethodnih generacija kako bi se poboljšala kvaliteta rješenja. Međutim, genetski algoritam sam po sebi može biti ograničen u pronalaženju lokalno optimalnih rješenja. Upravo tu dolazi do izražaja lokalna pretraga u memetičkom algoritmu. Nakon primjene genetskog algoritma, najbolje jedinke se odabiru za daljnju optimizaciju putem lokalne pretrage. Lokalna pretraga fokusira se na poboljšanje kvalitete rješenja u okolini trenutnih rješenja. Kroz primjenu različitih heurističkih postupaka, kao što su lokalno pretraživanje susjedstva ili kombinacija s drugim metaheuristikama, memetički algoritam nastoji pronaći bolja lokalna rješenja (Goldbarg i sur, 2012).

Tablica 2 prikazuje pregled karakteristika memetičkog algoritma (MA) u kontekstu rješavanja problema putujućeg trgovca s iznajmljivanjem automobila (CaRS). Tablica ističe prednosti, mane i ključne karakteristike memetičkog algoritma. Prednosti uključuju efikasno kombiniranje genetskog algoritma (GA) i lokalne optimizacije (LO), mogućnost diverzifikacije i intenzifikacije pretraživanja, prilagodljivost i skalabilnost te potencijal za postizanje visokih kvaliteta rješenja.

S druge strane, memetički algoritam zahtijeva pravilno podešavanje parametara, postoji mogućnost zaglavljivanja u lokalnim optimumima te zahtijeva veće računalne resurse u usporedbi s jednostavnijim metodama. Razumijevanje ovih karakteristika pomaže u procjeni primjenjivosti i potencijala memetičkog algoritma u rješavanju složenih problema putujućeg trgovca s iznajmljivanjem automobila.

Tablica 2: Analiza memetičkog algoritma

Memetički algoritam	**Prednosti**	**Mane**	**Karakteristike**
Efikasno kombiniranje GA i LO	Kombinira globalno i lokalno pretraživanje, što može rezultirati bržim konvergencijama i boljim rješenjima	Potrebno je pravilno podešavanje parametara za postizanje optimalnih rezultata	Kombinacija evolucijskog procesa (GA) i lokalne optimizacije (LO)
Diverzifikacija i intenzifikacija	Omogućava istraživanje širokog prostora pretraživanja i fokusiranje na lokalne područja s visokom kvalitetom rješenja	Postoji mogućnost zaglavljivanja u lokalnim optimumima	Kombinacija globalne i lokalne pretrage
Prilagodljivost i skalabilnost	Može se prilagoditi različitim vrstama problema i promjenjivim uvjetima	Zahtijeva prilagodbu parametara ovisno o problemu i uvjetima	Može se primijeniti na različite domene i veličine problema
Potencijal za postizanje visokih kvaliteta rješenja	Može dovesti do pronalaska visokokvalitetnih rješenja u usporedbi s drugim metodama	Zahtijeva veću računalnu snagu i resurse u usporedbi s jednostavnijim metodama	Može doseći globalne optimume u nekim slučajevima

Primjena memetičkog algoritma na problem CaRS donosi brojne prednosti. Kombinacija globalne pretrage genetskog algoritma i lokalne pretrage omogućuje istraživanje šireg prostora rješenja i pronalaženje optimalnih putanja putujućeg trgovca s iznajmljivanjem automobila.

Isječak kôda 1: Primjer pseudokôda za memetički algoritam

```
Algoritam MemetičkiAlgoritam():
    Inicijaliziraj populaciju P
    Postavi broj iteracija na 0

    Sve dok kriterij zaustavljanja nije zadovoljen:
        Evaluiraj kvalitetu svakog rješenja u populaciji P
```

```
        Odaberi roditelje za reprodukciju iz populacije
        Generiraj nove potomke koristeći operator križanja
        Primijeni operator mutacije na potomke

        Primijeni lokalnu optimizaciju na potomke

        Zamijeni najslabija rješenja u populaciji P s novim potomcima

        Povećaj broj iteracija za 1
    Vrati najbolje rješenje u populaciji P
```

Izvor: izrađeno prema Mavrovouniotis i sur. (2016)

3.4.2. Simulirano kaljenje

U kontekstu problema CaRS, simulirano kaljenje je jedan od algoritama lokalne optimizacije koji se može primijeniti za pronalaženje lokalno optimalnog rješenja. Ovaj algoritam se temelji na principu simuliranja procesa kaljenja metala, gdje se postupno poboljšava rješenje problema kroz iterativni process (Teodorović, 2007).

Simulirano kaljenje koristi analogiju s kaljenjem metala kako bi pronašlo najbolje rješenje u prostoru pretraživanja. Proces simuliranog kaljenja započinje inicijalizacijom početnog rješenja. Nakon toga, algoritam vrši iteracije tijekom kojih se vrši pretraživanje susjedstva i odabir novog rješenja.Glavna karakteristika simuliranog kaljenja je da prihvaća lošija rješenja s određenom vjerojatnošću kako bi izbjeglo zaglavljivanje u lokalnim optimumima. Ta vjerojatnost prihvaćanja lošijeg rješenja postupno se smanjuje tijekom iteracija, što omogućuje algoritmu da se kreće prema boljim rješenjima. Ova strategija omogućuje simuliranom kaljenju da pretražuje širi prostor rješenja i da ima veću vjerojatnost pronalaska globalno optimalnog rješenja. U kontekstu problema CaRS, simulirano kaljenje se može primijeniti na optimizaciju putanja putujećeg trgovca s uključenim iznajmljivanjem automobila. Cilj je pronaći optimalnu putanju koja minimizira ukupno vrijeme putovanja, uz uzimanje u obzir ograničenja iznajmljivanja automobile (Teodorović, 2007).

3.4.3. Transgenetski algoritam

Transgenetski algoritam (TA) predstavlja naprednu metodu optimizacije koja kombinira princip genetskih algoritama s konceptom transgenetskog prenosa znanja. Ovaj algoritam ima za cilj

poboljšati performanse u pronalaženju optimalnih rješenja za složene probleme kao što je problem putujućeg trgovca s iznajmljivanjem automobila (CaRS) (Sandou, 2013).

Transgenetski algoritam temelji se na konceptu prenosa znanja između različitih jedinki unutar populacije. Ideja je da se informacija o dobrom rješenju ili karakteristikama koje doprinose kvaliteti rješenja prenese na druge jedinke kako bi se poboljšala njihova sposobnost pretraživanja prostora rješenja.

Glavne komponente transgenetskog algoritma uključuju, prema Sandou (2013):

- Populacija jedinki: Algoritam koristi populaciju koja se sastoji od različitih rješenja problema. Svaka jedinka predstavlja putni redoslijed gradova u problemu putujućeg trgovca s iznajmljivanjem automobila.
- Genetski operatori: Transgenetski algoritam koristi genetske operatore kao što su selekcija, križanje i mutacija. Oni omogućuju stvaranje novih jedinki iz postojećih, čime se provodi globalno pretraživanje prostora rješenja.
- Transgenetski operator: Ova komponenta predstavlja ključni element transgenetskog algoritma. Transgenetski operator prenosi znanje o dobrim rješenjima ili karakteristikama s jedne jedinke na druge. To se postiže kombiniranjem genetskog operatora s mehanizmom razmjene genetskog materijala između odabranih jedinki.
- Lokalna optimizacija: Nakon primjene transgenetskog operatora, nad dobivenim jedinkama primjenjuje se lokalna optimizacija kao dodatni korak. Lokalna optimizacija poboljšava kvalitetu rješenja unutar svake jedinke, što dovodi do boljih rezultata.
- Kriterij zaustavljanja: Algoritam se zaustavlja kada se ispuni određeni kriterij zaustavljanja, kao što je dostizanje maksimalnog broja generacija ili postizanje zadovoljavajućeg rješenja.

Transgenetski algoritam ima potencijal za rješavanje problema putujućeg trgovca s iznajmljivanjem automobila jer kombinira globalno pretraživanje prostora rješenja putem genetskog operatora s lokalnom optimizacijom koja poboljšava kvalitetu rješenja. Transgenetski operator omogućuje prenošenje znanja između jedinki, što može ubrzati konvergenciju algoritma i povećati vjerojatnost pronalaska optimalnog rješenja (Sandou, 2013).

Primjer transgenetskog algoritma prikazan je isječkom kôda 2. U ovom primjeru, funkcija TransgenetskiAlgoritam prima populaciju jedinki i broj generacija koje će se izvesti. Algoritam započinje inicijalizacijom populacije i evaluacijom jedinki kako bi se odredila njihova kvaliteta. Zatim, algoritam iterira kroz generacije. U svakoj generaciji, za svaku jedinku u populaciji, odabiru se roditelji putem selekcije. Nakon toga, primjenjuju se genetski operatori križanja i mutacije na roditeljske jedinke kako bi se generirala nova djeca. Nova djeca se dodaju u novu

populaciju.Nakon što su sva djeca generirana, nova populacija se evaluira kako bi se odredila njihova kvaliteta. Zatim se nova populacija postaje trenutna populacija, i postupak se ponavlja za sljedeću generaciju. Na kraju, algoritam vraća najbolje rješenje (najbolju jedinku) iz konačne populacije.

Isječak kôda 2: Primjer transgenetskog algoritma

```
Funkcija TransgenetskiAlgoritam(populacija, broj_generacija):
Inicijaliziraj populaciju
Evaluacija(populacija)

Za svaku generaciju od 1 do broj_generacija:
Nova_populacija = [ ]

Za svaku jedinku u populaciji:
Roditelji = Selekcija(populacija)
Djeca = Križanje(Roditelji)
Djeca = Mutacija(Djeca)
Nova_populacija += Djeca

Evaluacija(Nova_populacija)
populacija = Nova_populacija
Vrati najbolje rješenje iz populacije
```

Izvor: izrađeno prema Sandou (2013)

U implementaciji ovog istraživanja, transgenetski algoritam će se primijeniti kao jedan od algoritama lokalne optimizacije za rješavanje problema putujućeg trgovca s iznajmljivanjem automobila. Kroz iterativno izvođenje genetskih operatora, transgenetskog operatora i lokalne optimizacije, očekuje se postizanje boljih rezultata u pronalaženju optimalnih putnih redoslijeda i smanjenju ukupne udaljenosti putovanja.

3.4.4. Tabu algoritam

Istraživanje Stankovića (2013) o primjeni algoritma tabu pretraživanja u rješavanju problema kombinatorne optimizacije pruža korisne uvide za problem putujećeg trgovca. Autor ističe važnost liste tabu kao sredstva za izbjegavanje ponovnog posjećivanja prethodno posjećenih

rješenja, što je ključno za istraživanje novih područja pretraživanja. Ova ideja se može primijeniti u kontekstu problema putujućeg trgovca s iznajmljivanjem automobila (CaRS).

Kroz primjenu algoritma tabu pretraživanja, moguće je izbjeći ponovno posjećivanje već ispitanih ruta i fokusirati se na istraživanje novih mogućnosti. To može rezultirati pronalaženjem optimalnih ili poboljšanih rješenja za problem CaRS. Osim toga, analiza performansi algoritma tabu pretraživanja u usporedbi s drugim algoritmima optimizacije može pružiti relevantne informacije o učinkovitosti i primjenjivosti ovog algoritma u kontekstu problema putujućeg trgovca s iznajmljivanjem automobila. Važno uzeti u obzir i mogućnosti za poboljšanje i prilagodbu algoritma tabu pretraživanja specifičnim zahtjevima problema CaRS. Promišljeno podešavanje parametara algoritma može biti ključno za postizanje optimalnih rezultata. Također, identificirane prednosti i ograničenja algoritma tabu pretraživanja mogu biti korisne smjernice prilikom evaluacije i usporedbe s drugim metodama lokalne optimizacije u istraživanju lokalne optimizacije problema putujećeg trgovca.

3.5. Analiza opcije iznajmljivanja automobila

U ovom potpoglavlju provodi se detaljna analiza opcije iznajmljivanja automobila u kontekstu problema putujućeg trgovca s iznajmljivanjem automobila (CaRS). Izbor iznajmljivanja automobila ima značajan utjecaj na rješenje problema putujućeg trgovca, jer utječe na troškove, kapacitete vozila i vremenska ograničenja za povratak automobila.

Jedan od ključnih aspekata koji se razmatra u analizi opcije iznajmljivanja automobila su troškovi. Izbor određene tvrtke za iznajmljivanje automobila može utjecati na ukupne troškove putovanja. Različiti pružatelji usluga iznajmljivanja automobila mogu imati različite cijene i politike naplate, uključujući cijene po satu, kilometru ili fiksnih dnevnih naknada. Potrebno je pažljivo procijeniti troškove kako bi se odabrala najekonomičnija opcija koja će minimizirati ukupne troškove putovanja (Goldbarg i sur., 2012).

Integracija iznajmljivanja automobila u problem putujućeg trgovca također zahtijeva razmatranje kapaciteta vozila. Svako vozilo koje se iznajmljuje ima ograničeni kapacitet, određen brojem putnika ili količinom tereta koje može prevoziti. Ograničenja kapaciteta vozila moraju se uzeti u obzir prilikom generiranja rješenja problema putujućeg trgovca, kako bi se osiguralo da ukupna količina tereta ili broj putnika ne premašuje kapacitete vozila. Kada se automobil iznajmljuje za putovanje putujućeg trgovca, važno je uzeti u obzir vremenska ograničenja za povratak automobila. Mnoge tvrtke za iznajmljivanje automobila imaju određeno vrijeme do kojeg se automobil mora vratiti kako bi se izbjegle dodatne naknade ili

kazne. Ova vremenska ograničenja moraju se uzeti u obzir prilikom planiranja putovanja i generiranja rješenja problema putujućeg trgovca (Goldbarg i sur., 2012).

Ova analiza opcije iznajmljivanja automobila ima za cilj procijeniti utjecaj izbora iznajmljivanja automobila na optimalno rješenje problema putujućeg trgovca. Promjene u troškovima, kapacitetima vozila i vremenskim ograničenjima mogu rezultirati promjenama u kvaliteti rješenja, optimalnoj putanji ili ukupnom trošku putovanja. Evaluacija i usporedba različitih opcija iznajmljivanja automobila omogućit će odabir najpogodnije opcije koja će pridonijeti pronalasku optimalnog rješenja problema putujućeg trgovca.

Navedena analiza opcije iznajmljivanja automobila bit će provedena kroz prikupljanje relevantnih podataka, usporedbu cijena i politika različitih tvrtki za iznajmljivanje automobila, te modeliranje i evaluaciju utjecaja tih opcija na rješenje problema putujućeg trgovca. Rezultati analize bit će kritični za razumijevanje mogućnosti i ograničenja integracije iznajmljivanja automobila u rješavanje problema CaRS.

4. Algoritam optimizacije kolonijom mrava

U ovom poglavlju će se razmatrati primjena algoritma optimizacije kolonijom mrava kao preduvjet za implementaciju metode lokalne optimizacije. Kolonija mrava je metaheuristička tehnika inspirirana ponašanjem stvarnih kolonija mrava u potrazi za najkraćim putem od izvora hrane do mravinjaka. Algoritam kolonijom mrava pokazao se vrlo uspješnim u rješavanju problema optimizacije kombinatornih problema. Za potrebe ovog diplomskog rada koristi se algoritam kolonijom mrava koji je razvijen i implementiran od strane Elvisa Popovića. Spomenuti algoritam je prilagođen i optimiziran kako bi se rješavao odabrani problem, a njegova implementacija dostupna je na online repozitoriju Githuba (Github, repozitorij Elvisa Popovića).

Algoritam kolonijom mrava je baziran na ponašanju stvarnih kolonija mrava, koji putujući između izvora hrane i mravinjaka, ostavljaju feromone na stazama kojima prolaze. Ovaj prirodni mehanizam komunikacije omogućava mravima da slijede staze s više feromona, što rezultira otkrivanjem kraćih i boljih putanja do izvora hrane. U kontekstu optimizacije, algoritam koristi analogiju feromona za odabir boljih rješenja u pretraživanju prostora mogućih rješenja problema. Svaki mrav predstavlja jedno moguće rješenje, a svaka staza koju mrav slijedi predstavlja element rješenja. Kako mrav prolazi stazama, ostavlja feromone koji se povećavaju ili smanjuju ovisno o kvaliteti rješenja (Grković i Bulatović, 2012).

Prolaskom kroz više iteracija, algoritam omogućuje mravima da postupno konvergiraju prema boljim rješenjima, jer staze s više feromona privlače više mrava. Istovremeno, feromoni isparavaju s vremenom, što omogućava algoritmu da istražuje različite staze i izbjegava zarobljavanje u lokalnim optimumima (Grković i Bulatović, 2012).

Algoritam se izvodi kroz niz koraka iterativnog pretraživanja prostora rješenja, gdje svaki korak obuhvaća sljedeće ključne elemente (Grković i Bulatović, 2012):

1. Inicijalizacija feromona.
2. Kreiranje kolonije mrava i njihov prolazak kroz staze uz ostavljanje feromona.
3. Evaluacija kvalitete svakog rješenja i ažuriranje feromona na temelju pronađenih boljih rješenja.
4. Ponavljanje koraka 2 i 3 kroz više iteracija, uz moguću primjenu heurističkih pristupa za usmjeravanje pretraživanja.

U ovom diplomskom radu, algoritam kolonijom mrava će biti korišten kao osnova za lokalnu optimizaciju kako bi se poboljšala kvaliteta rješenja u konkretnom problemu. Rezultati dobiveni

primjenom ove tehnike bit će temelj za daljnje uspoređivanje s drugim optimizacijskim algoritmima i analizu učinkovitosti rješenja.

Kako bi se implementirala metoda lokalne optimizacije potrebno je kao prvo implementirati algoritam kolonije mrava nad kojim će se vršiti postupak lokalne optimizacije. U svrhu ovog diplomskog rada korišten je implementirani novi algoritam Elvisa Popovića, dostupan na online repozitoriju Githuba (Github, repozitorij Elvisa Popovića). Prilagođena je *main.cpp* funkcija kako bi se program pravilno pozivao. Glavni element koji je potreban osim pomoćnih funkcija za čitanje, pisanje i pokretanje programa je *createSolution.cpp* , potrebno zaglavlje (eng. h*eader*) te funkcije *CreateSolution.h* te algoritam kolonije mrava *ant_run.cpp*.

4.1. Opis algoritma

Algoritam optimizacije kolonijom mrava (*Ant Colony Optimization*, ACO) koristi se za rješavanje problema optimizacije. ACO je nedeterministička metaheuristika koja simulira ponašanje kolonije mrava u potrazi za hranom (Eftimov i sur, 2018). U ovom algoritmu koriste se dva ključna postupka:

- postupak konstrukcije rješenja;
- postupak ažuriranja tragova feromona.

Postupak konstrukcije rješenja koristi se za generiranje rješenja korak po korak. Algoritam započinje s praznim rješenjem, a zatim dodaje komponente rješenja temeljem vjerojatnosti koje se izračunavaju na temelju tragova feromona i heurističkih informacija. Ova procedura omogućuje postupno stvaranje kvalitetnih rješenja. Vjerojatnost odabira određene komponente, poput sljedećeg čvora ili automobila, ovisi o vrijednostima feromona i heuristici. Izračunavanje vjerojatnosti provodi se prema formuli (Popović i sur., 2022):

$$p_{ij}^{k} = \frac{[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{l \in \mathcal{N}_i^k} [\tau_{il}]^{\alpha}[\eta_{il}]^{\beta}}, \text{ if } j \in \mathcal{N}_i^k$$

U ovoj formuli, τij predstavlja trag feromona između čvora i i čvora j, α i β su parametri algoritma koji kontroliraju utjecaj tragova feromona i heuristike, ηij predstavlja heurističku informaciju o atraktivnosti prijelaza između čvorova i i j, a N_i^kredstavlja skup dostupnih susjeda čvora i u koraku k.

Nakon generiranja rješenja, slijedi postupak ažuriranja tragova feromona. Ovaj postupak ima za cilj prilagoditi tragove feromona na temelju kvalitete pronađenih rješenja. Ažuriranje se provodi taloženjem novih tragova feromona koji su obrnuto proporcionalni trošku najboljeg rješenja pronađenog od strane elitnog mrava. Izraz za ažuriranje tragova feromona između čvora i čvora j je:

$$\Delta\tau_c^{(best)} = \begin{cases} \frac{1}{f(s^{best})}, & \text{if } c \in s^{\text{best}} \\ 0 & \text{else} \end{cases}$$

U ovoj formuli, f (sbest) predstavlja kvalitetu najboljeg rješenja pronađenog od strane mrava. Ažuriranje tragova feromona provodi se kombinacijom taloženja novih tragova feromona i isparavanja postojećih tragova feromona.

Važno je naglasiti da se odabir najboljeg rješenja i ažuriranje tragova feromona provode u skladu s odabranom strategijom jačanja tragova. Moguće strategije uključuju strategiju najbolje iteracije i globalnu strategiju. Također se može koristiti općenitije strategije poput κ-najbolje i maks-κ-najbolje kako bi se fino podešavalo ponašanje algoritma. Osim taloženja tragova feromona, također se provodi isparavanje postojećih tragova feromona prema slijedećim izrazima. Trag feromona između čvora i i čvora j ažurira se kombinacijom isparavanja i taloženja prema izrazu:

$$\tau_{ij}^n \leftarrow (1-\rho_n)\tau_{ij}^n, \forall (i,j) \in L$$

Slično, trag feromona između čvora i i automobila ik ažurira se prema izrazu:

$$\tau_{ik}^c \leftarrow (1-\rho_c)\tau_{ik}^c, \forall ik \in C$$

U ovim izrazima, ρ_n i ρ_c su parametri algoritma, a L i C predstavljaju skupove svih čvorova grana ili čvorova automobila, redom. Općenito, trag feromona na putanji najboljeg mrava ažurira se kombinacijom isparavanja i taloženja prema izrazu:

$$\tau_c = (1-\rho)\cdot\tau_c + \Delta\tau_c^{(\text{best})}$$

Nakon određenog broja iteracija, obično se javlja razdvajanje tragova feromona zbog ažuriranja tragova feromona (Ivković i sur., 2011). Trag feromona dobrih komponenti iz prethodnih najboljih rješenja bit će puno veći od tragova feromona ostalih komponenti.

Vrijednosti tragova feromona komponenata koje nisu dio najboljeg rješenja tijekom mnogih iteracija eksponencijalno se smanjuju prema izrazima za ažuriranje feromona. To može dovesti do stagnacije algoritma jer se vjerojatnost odabira neke nove komponente može postati izuzetno mala, pa čak i nula zbog ograničene preciznosti prikaza brojeva s pomičnim zarezom na računalima. Kako bismo izbjegli stagnaciju algoritma, tragovi feromona su ograničeni između τ_{max} i τ_{min} kao kod MAX-MIN. Gornja granica određuje se prema izrazu Ant System (Ivković i sur., 2011):

$$\tau_{max} = \frac{1}{\rho \cdot f(s^*)}$$

U ovom izrazu, f(s∗) predstavlja trošak optimalnog rješenja s∗. Za određene probleme poput Problema trgovačkog putnika, asimetričnog Problema trgovačkog putnika i Problema kvadratne raspodjele, postoje analitički izrazi koji se koriste za određivanje donje granice (Ivković i sur., 2011). Međutim, za različite vrste problema i varijante algoritma, ovi izrazi mogu biti različiti. Za ACOCaRS, kao i za većinu drugih ACO algoritama, analitički izraz nije poznat. U takvim situacijama, donja granica τmin se izračunava množenjem gornje granice s algoritamskim parametrom θ, prema sljedećem izrazu:

$$\tau_{min} = \vartheta \cdot \tau_{max}$$

4.2. Pseudokod i implementacija algoritma

U nastavku je opisan pseudokod i implementaciju algoritma ACOCaRS (Ant Colony Optimization for Car Renter Problem with Stochastic Demand) (Popović i sur., 2022). Isječak kôda 3 prikazuje algoritam prima za ulaz problem koji treba riješiti i parametre algoritma koji su nužni za izvođenje. Za izlaz se generira najbolje rješenje bestSolution.

Isječak kôda 3: Pseudokod ACOCaRS

```
Input: (Problem, Param)
Output: (bestSolution)
1: (τ, η, sol, bestSolution) ←Init(Problem, Param)
2: for iteration = 0 to P aram.n ants do
3: for ant = 0 to P aram.M iter do
4: sol[ant] ← ConstructSolution(Problem, Param, τ , η)
5: end for
6: τ ←UpdatePheromones(τ , Param, sol)
7: bestSolution ←FindBestSol(sol)
```

```
8: end for
9: return bestSolution
```

Izvor: Prema Github repozitoriju Elvisa Popovića

Ovaj pseudokod omogućuje implementaciju i izvršavanje algoritma ACOCaRS i pronalaženje najboljeg rješenja problema putem optimizacije kolonijom mrava.

U prvom koraku, vrši se inicijalizacija algoritma. Tragovi feromona (τ), heurističke vrijednosti (η), rješenje (sol) i varijabla za najbolje rješenje (bestSolution) inicijaliziraju se prema ulaznim parametrima. Nakon inicijalizacije, u drugom koraku izvršava se petlja iteracija mrava. Za svaku iteraciju mrava, unutar petlje za pojedinačnog mrava, konstruira se rješenje (sol[ant]) koristeći postupak konstrukcije rješenja na temelju problema, parametara, tragova feromona (τ) i heuristike (η). U petom koraku ažuriraju se tragovi feromona. Koristi se postupak ažuriranja tragova feromona na temelju dobivenih rješenja (sol) i parametara. Nakon ažuriranja tragova feromona, u šestom koraku pronalazi se najbolje rješenje (bestSolution) među svim dobivenim rješenjima (sol). Konačno, u poslijednjem koraku se kao rezultat algoritma vraća najbolje pronađeno rješenje (bestSolution).

Implementacija ovog algoritma zahtijeva definiranje funkcija za inicijalizaciju, konstrukciju rješenja, ažuriranje tragova feromona i pronalaženje najboljeg rješenja, što je prikazano u isječku kôda 4.

Isječak kôda 4: Implementacija funkcije kalkuliranja riješenja ACOCaRS

```
double CalculateSolution(double (*calculateSolutionCost)(ProblemData
&problem, Solution &solution), Solution &solution)
{
    int i, it, a;
    double result = 0.0;
    Solution antSolution, iterSolution, kappaMinSolution;
    std::deque<Solution>::iterator kappaIt;

    resetSolution(antSolution); //ant solution
    resetSolution(iterSolution); //best ant solution in one iteration
    resetSolution(solution); //best solution of all iterations

    solution.problemName = solData.problemData.name;
    solution.cost = numeric_limits<double>::max();
    for(it=0; it<solData.parameters.nIterations; it++)
    {
        iterSolution.cost = numeric_limits<double>::max();
```

```
        for(a=0; a<solData.parameters.nAnts; a++)
        {
            if(AntRun(antSolution))
            {
                result = calculateSolutionCost(solData.problemData,
antSolution);
                antSolution.iteration = it;
                if(result < iterSolution.cost) //best ant
                    CopySolution(antSolution, iterSolution);
            }
        }
        AddKappa(iterSolution);
        kappaMinSolution = GetMinKappa();
        UpdatePheromones(kappaMinSolution); //best ant solution =
iterSolution
        if(iterSolution.cost < solution.cost)
        {
            CopySolution(iterSolution, solution);
            CalculateMaxMin(solution.cost);

            std::cout << "Iter: " << solution.iteration << ", best solution
" << solution.cost << ", max car: " <<
            solData.carPheroMax << ", max node: " << solData.nodePheroMax
<< std::endl;
            int nodeListSum = 0;
            for(i=0; i<solData.problemData.dimension; i++)
            {
                nodeListSum +=solution.nodeList[i];
                std::cout << "(" << solution.carList[i] << ":" <<
solution.nodeList[i] << ") ";
            }
            std::cout << " | " << nodeListSum << std::endl;
        }
        else
            std::cout << "Iter: " << it << ", cost: " << iterSolution.cost
<< ", best: " << solution.cost << ", kappa best: " << GetMinKappa().cost <<
std::endl;
    }
    return result;
}
```

Izvor: Prema Github repozitoriju Elvisa Popovića

Navedeni kod predstavlja implementaciju algoritma ACOCaRS (Ant Colony Optimization for Car Renter Problem with Stochastic Demand) u jeziku C (Github, repozitorij Elvisa Popovića). Ova implementacija sadrži funkciju CalculateSolution koja izračunava rješenje problema na temelju dane ulazne konfiguracije.

Ulazni parametri funkcije su:

- *calculateSolutionCost*: Pokazivač na funkciju koja izračunava trošak rješenja za određeni problem i rješenje;
- *solution*: Struktura koja predstavlja rješenje problema.

Implementacija započinje inicijalizacijom različitih varijabli, uključujući antSolution, iterSolution, i kappaMinSolution, koje se koriste za pohranjivanje privremenih rješenja. Također, inicijaliziraju se varijable solution, antSolution, i iterSolution na početne vrijednosti.

Nakon inicijalizacije, slijedi petlja iteracija koja se izvodi prema broju definiranih iteracija (solData.parameters.nIterations). Unutar ove petlje, izvršava se petlja mrava koja se izvodi prema broju definiranih mrava (solData.parameters.nAnts). Za svakog pojedinačnog mrava, izvršava se funkcija AntRun koja konstruira rješenje za tog mrava te računa njegov trošak pomoću funkcije calculateSolutionCost. Ako je trošak rješenja manji od trenutno najboljeg rješenja za tu iteraciju (iterSolution.cost), kopira se to rješenje u iterSolution.

Nakon petlje mrava, dodaje se iterSolution u deque kappa pomoću funkcije AddKappa, a zatim se iz dequea kappa dobiva rješenje s najmanjim troškom pomoću funkcije GetMinKappa i pohranjuje u kappaMinSolution. Slijedi ažuriranje tragova feromona pomoću funkcije UpdatePheromones, pri čemu se koristi najbolje pronađeno rješenje (kappaMinSolution) kao referenca za ažuriranje tragova feromona.

U sljedećem koraku provjerava se je li trenutno najbolje rješenje (iterSolution) bolje od ukupno najboljeg rješenja (solution). Ako je to slučaj, kopira se iterSolution u solution i vrši se izračun vrijednosti maxMin feromona pomoću funkcije CalculateMaxMin. Također, ispisuje se trenutno najbolje rješenje, maksimalna vrijednost feromona za automobile (solData.carPheroMax) i čvorove (solData.nodePheroMax), te se ispisuje lista čvorova i broj posjećenih čvorova u tom rješenju. Ako trenutno najbolje rješenje nije bolje od ukupno najboljeg rješenja, ispisuje se trenutni trošak rješenja (iterSolution.cost), ukupno najbolje rješenje (solution.cost) i najbolje rješenje iz dequea kappa (GetMinKappa().cost). Na kraju petlje iteracija, vraća se izlazni rezultat rješenja.

Kako bi se usporedila rješenja te pronašlo optimalno rješenje potrebno je implementirati funkciju calculateSolutionCost koje je opisana isječkom kôda 5 (Popović i sur. 2022).

Isječak kôda 5: Pseudokod konstrukcije rješenja

```
Input: (Prob, Param, τ, η)
Output: (AntSol)
1: AntSol ← Ø AntSol starts as empty partial solution
2: currN ode ← 0 Each route start with node 0
3: currCar ← SelectCar(Prob, Param, currNode, currCar)
4: AntSol ←AddToSol(AntSol, 0, currentCar, currentNode)
5: for i = 1 to P rob.numOf N odes do
6: currtN ode ← SelectNode(τ ,η, Prob, Param, currtNode, currCar)
7: currCar ← SelectCar(τ ,η, Prob, Param, currNode, currCar)
8: AntSol ←AddToSol(AntSol, i, currCar, currNode)
9: end for
10: return AntSol
```

Izvor: Popović i sur. (2022)

Ulazni parametri funkcije su sljedeći: (Prob) predstavlja problem koji se rješava, (Param) su parametri algoritma, (τ) označava tragove feromona, a (η) označava heurističke vrijednosti. Pseudokod koristi ove ulazne parametre kako bi izvršio konstrukciju rješenja na temelju tragova feromona i heuristike te pružio rješenje AntSol kao izlaznu vrijednost. Pseudokod započinje inicijalizacijom praznog djelomičnog rješenja AntSol. Trenutni čvor postavlja se na početni čvor 0, a trenutni automobil odabire se pomoću funkcije SelectCar. Zatim se trenutno odabrani čvor i automobil dodaju u rješenje pomoću funkcije AddToSol. Nakon inicijalizacije, petljom se prolazi kroz sve čvorove. U svakoj iteraciji, odabire se sljedeći čvor pomoću funkcije SelectNode na temelju tragova feromona (τ) i heuristike (η). Zatim se odabire sljedeći automobil pomoću funkcije SelectCar na temelju trenutnog čvora i automobila. Odabrani čvor i automobil dodaju se u rješenje AntSol pomoću funkcije AddToSol. Na kraju petlje, rješenje AntSol se vraća kao rezultat konstrukcije rješenja.

Implementacija CalculateSolutionCost napisana je u isječku kôda 6.

Isječak kôda 6: Implementacija funkcije konstrukcije rješenja

```
double CalculateSolutionCost(ProblemData &problem, Solution &solution)
{
    // n1 current node, n2 next node
    // c1 last car, c2 current car
    // np number of passengers in the car
```

```
    int i, n1, n2, c1, c2, cp, np;
    double result;
    if(solution.dimension    <    2    ||    solution.nodeList.size()    !=
(size_t)solution.dimension)
        return 0.0;
    if(problem.edge_weight_format        !=        EF_FULL_MATRIX        ||
problem.edge_weight_type != ET_EXPLICIT)
        return 0.0; //not implemented
    cp = solution.nodeList[0];
    c1 = solution.carList[0];
    result = 0.0;
    for(i=0; i<solution.dimension; i++)
    {
        n1 = solution.nodeList[i];
        if(i<solution.dimension-1)
            n2 = solution.nodeList[i+1];
        else
            n2 = solution.nodeList[0];
        if(i>0)
            c1 = solution.carList[i-1];
        c2= solution.carList[i];
        if(solution.passengersNodeList.size()>0)
            np = solution.passengersNodeList[n1].size()+1;
        else
            np = 1;
        result                                                          +=
(double)problem.nonEuclidSection.edgeWeight[c2][n1][n2]/np;
        if(c1 != c2)
        {
            result += problem.nonEuclidSection.returnRate[c1][n1][cp];
            cp = n1;
        }
    }
    result += problem.nonEuclidSection.returnRate[c1][0][cp];
    solution.cost = result;
    return result;
}
```

Izvor: Prema Github repozitoriju Elvisa Popovića

Funkcija CalculateSolutionCost izračunava trošak rješenja na temelju problema i zadane rute. U kodu su definirane sljedeće varijable: n1 i n2 predstavljaju trenutni i sljedeći čvor u ruti, c1 i c2 predstavljaju prethodno i trenutno odabrani automobil, cp predstavlja zadnji posjećeni čvor, np predstavlja broj putnika u automobilu, i se koristi za iteraciju kroz čvorove rute, a result je ukupni trošak rješenja.

Prije izračuna, provjerava se ispravnost dimenzije rute i broja čvorova u listi čvorova. Također, provjerava se odgovarajući format i tip podataka problema. Ako provjere nisu zadovoljene, vraća se vrijednost 0.0. Zatim se inicijaliziraju početne vrijednosti za varijable cp i c1, te se postavlja rezultat na 0.0. U petlji se iterira kroz čvorove rute i izračunava se ukupni trošak rješenja. Trošak se povećava za težinu prijelaza između trenutnog čvora n1 i sljedećeg čvora n2, podijeljenu s brojem putnika np u automobilu. Također, ako se promijeni automobil (c1 != c2), dodaje se povratna vrijednost prijelaza između prethodnog čvora n1 i zadnjeg posjećenog čvora cp. Na kraju se dodaje i povratna vrijednost prijelaza između posljednjeg čvora n1 i početnog čvora 0. Naposljetku, rezultat se pohranjuje u varijablu solution.cost i vraća kao izlaz funkcije.

5. Implementacija algoritma lokalne optimizacije

U ovom poglavlju predstavljen je postupak implementacije algoritma lokalne optimizacije s ciljem poboljšanja kvalitete rješenja problema koji se rješava primjenom algoritma optimizacije kolonijom mrava. Kao što je prethodno opisano, algoritam kolonijom mrava je metaheuristička tehnika koja omogućuje pronalaženje dobrih rješenja za kombinatorne probleme. Algoritam lokalne optimizacije za problem iznajmljivanja automobila ima za cilj poboljšati postojeće rješenje izvršavajući seriju malih izmjena, prilagodbi ili zamjena kako bi se postigla bolja kvaliteta rješenja.

U početku, algoritam koristi inicijalno rješenje koje sadrži redoslijed gradova i informacije o tome s kojim se automobilom izlazi iz pojedinog grada. Na primjer, ako postoji popis gradova 4, 3, 1, 7, 2, n, ..., 5, 6, uz gradove su dodijeljeni automobili, pa dobivamo bilježenje redoslijeda gradova sa sljedećim oznakama automobila: 4A, 3A, 1E, 7E, 2E, NE,..., 5C, 6C.

Glavni korak algoritma je pomicanje mjesta iznajmljivanja automobila kako bi se poboljšalo ukupno rješenje. To se postiže odabirom jednog od automobila i izračunavanjem za koliko bi se promijenila cijena rješenja ako se taj automobil pomakne za jedno mjesto udesno. Na primjer, ako je automobil iznajmljen u čvoru "*j*", a sljedeći čvor je "*k*", tada se računa razlika u cijeni iznajmljivanja automobila na prethodnom i trenutnom čvoru. Ako je izračunata cijena negativna ili nula, vrši se pomicanje automobila za jedno mjesto udesno.

Ovaj postupak pomicanja mjesta automobila ponavlja se sve dok se dobiva poboljšanje rješenja ili dok se ne dođe do kraja segmenta za taj automobil. Ukoliko nema poboljšanja, algoritam prelazi na sljedeći automobil i iznova vrši provjeru mogućih pomaka. Ako se dogodi situacija u kojoj neki automobil ostane samo na jednoj cesti, odnosno iza njega nema drugog automobila, tada se isplati izbaciti taj automobil iz rješenja. Na taj način se izbjegava plaćanje iznajmljivanja s vraćanjem automobila na sljedećem koraku, što može dovesti do boljeg rješenja.

5.1. Primjena programskog rješenja

Za implementaciju lokalne optimizacije korištena je funkcija *LocalOpt*. Ova funkcija predstavlja srž algoritma lokalne optimizacije te se koristi za poboljšanje postojećeg rješenja u problemu CaRS. Ova funkcija je odgovorna za pozivanje manjih funkcija koje obavljaju specifične operacije lokalne optimizacije.

U početku, *LocalOpt* funkcija se poziva s tri glavna parametra:

- *Solution& sol*: Referenca na objekt klase Solution koji sadrži trenutno rješenje problema.
- *ProblemData& problem*: Referenca na objekt klase ProblemData koji sadrži informacije o problemu CaRS, kao što su matrice troškova i povrata za ceste.
- *int MaxTries:* Broj maksimalnih pokušaja koji se koriste u procesu lokalne optimizacije.

Unutar *LocalOpt* funkcije nalazi se petlja koja se izvodi *MaxTries* puta, kako bi se provjerilo više puta mogu li se postići poboljšanja u rješenju. U svakom koraku petlje, poziva se funkcija *move_right_by1* za provođenje pojedinačne iteracije lokalne optimizacije.

Ova petlja, prikazana isječkom kôda 7, omogućuje višestruko izvršavanje lokalne optimizacije kako bi se povećala vjerojatnost pronalaženja boljeg rješenja. Pomoću poziva *move_right_by1* provode se promjene u rješenju kako bi se provjerilo ima li poboljšanja s desne strane, nakon toga traže se poboljšanja sa funkcijom *move_left_by1*. Na kraju petlje, rješenje se vraća s izračunatom optimiziranom vrijednosti ukupnih troškova, koja je pohranjena u varijabli *sol.cost.*

Isječak kôda 7: Primjena funkcije *LocalOpt*

```
double LocalOpt(Solution& sol, ProblemData& problem, int MaxTries) {
    for (int tryCount = 0; tryCount < MaxTries; tryCount++) {
        move_right_by1(sol, problem.nonEuclidSection.edgeWeight,
problem.nonEuclidSection.returnRate, tryCount, sol.dimension);

      move_left_by1(sol, problem.nonEuclidSection.edgeWeight,
problem.nonEuclidSection.returnRate, tryCount, sol.dimension);

    }


    return sol.cost;
}
```

Izvor: vlastita izrada

Kako bi se uspješno implementirala funkcija LocalOpt za poboljšanje rješenja problema CaRS, ključno je detaljno razraditi implementaciju funkcije move_right_by1. Funkcija move_right_by1 igra ključnu ulogu u pojedinačnom pomicanju automobila u desno unutar rješenja te se koristi za provođenje pojedinačnih iteracija lokalne optimizacije. Funkcija move_right_by1 prima sljedeće parametre:

- *Solution& solution*: Referenca na objekt klase Solution, koji predstavlja trenutno rješenje problema CaRS i u kojem su pohranjeni gradovi i informacije o automobilima.
- *const vector<vector<vector<int>>>& cost*: Referenca na trodimenzionalni vektor koji sadrži troškove putovanja između gradova za svaki automobil.
- *const vector<vector<vector<int>>>& costOfReturningVehicle*: Referenca na trodimenzionalni vektor koji sadrži troškove povratka vozila na ishodište za svaki automobil.
- *int start_segment*: Početni segment u kojem se vrši provjera pomicanja automobila u desno.
- *int end_segment*: Krajnji segment u kojem se vrši provjera pomicanja automobila u desno.

Funkcija *move_right_by1*, prikazana na isječku kôda 8, koristi petlju koja se izvodi sve dok je *start_segment* manji od end_segment i dok funkcija *move_right_by1_single* vraća true. Unutar petlje, svaki put kad se funkcija move_right_by1_single uspješno izvrši, povećava se vrijednost start_segment kako bi se provjerio sljedeći segment. Na isti način funkcionira I *move_left_by1*, razlika je sto funkcija za pomak u desno počinje od prvog segmenta i ide do zadnjeg, a funkcija za pomicanje u lijevo krece od zadnjeg segmenta i ide prema prvom.

Isječak kôda 8: Primjena funkcije void move_right_by1

```
void move_right_by1(Solution& solution, const vector<vector<vector<int>>>&
cost, const vector<vector<vector<int>>>& costOfReturningVehicle, int
start_segment, int end_segment) {
    while (start_segment < end_segment && move_right_by1_single(solution,
cost, costOfReturningVehicle, start_segment)) {
        start_segment++;
    }
}
```

Izvor: vlastita izrada

Posljednja funkcija koju je potrebno implementirati kako bi implementacija bila potpuna je move_right_by1_single. Ova funkcija ima ključnu ulogu u provođenju pojedinačnih pomicanja automobila u desno te se koristi za provjeru i izvršavanje pojedinačnih koraka lokalne optimizacije. Funkcija move_right_by1_single prima sljedeće parametre:

- *Solution& solution*: Referenca na objekt klase Solution, koji predstavlja trenutno rješenje problema CaRS i u kojem su pohranjeni gradovi i informacije o automobilima.

- *const vector<vector<vector<int>>>& cost*: Referenca na trodimenzionalni vektor koji sadrži troškove putovanja između gradova za svaki automobil.
- *const vector<vector<vector<int>>>& costOfReturningVehicle*: Referenca na trodimenzionalni vektor koji sadrži troškove povratka vozila na ishodište za svaki automobil.
- *int start_segment*: Početni segment u kojem se vrši provjera pomicanja automobila u desno.

Unutar funkcije, najprije se inicijaliziraju varijable improvement, *last_index, i* i *j* koje će se koristiti u daljnjem izvršenju funkcije. Varijabla *improvement* predstavlja iznos poboljšanja rješenja koje bi se moglo ostvariti pomakom automobila, dok *last_index* označava indeks zadnjeg segmenta u rješenju. Varijable i i j koriste se za indeksiranje gradova u rješenju.

Nakon toga, funkcija provjerava specifične uvjete za koje se ne smije izvršiti pomicanje automobila. Na primjer, ako je *start_segment* jednak nuli, to znači da je u pitanju početni segment u kojem se uzima prvi automobil, pa u tom slučaju automobil ne smije biti pomaknut u desno jer nema prethodnog automobila.

Ako ne postoje posebni uvjeti koji zabranjuju pomicanje automobila, funkcija nastavlja s izračunavanjem mogućeg poboljšanja rješenja. To se postiže usporedbom troškova putovanja za prethodni automobil (označen s *previous_vehicle*) i trenutni automobil (označen s *current_vehicle*) iz grada *i* u grad *j.* Ako se izračuna pozitivna razlika improvement, to znači da bi se pomakom automobila ostvarilo poboljšanje.

Dodatno, funkcija uzima u obzir i slučaj kada automobil nije vraćen na početno mjesto, što znači da će se morati platiti trošak povratka vozila na ishodište. U tom slučaju, funkcija dodaje taj trošak *(costOfReturningVehicle[prev_vehicle][node_i][node_j])* na novi trošak putovanja automobila. Nakon što se izračuna konačno poboljšanje improvement, provjerava se je li ono pozitivno, što bi značilo da će pomak automobila rezultirati boljim rješenjem. Ako je to slučaj, automobil se pomakne u desni segment, a vrijednosti troška rješenja se ažuriraju u skladu s ostvarenim poboljšanjem. Dodatno, za svrhu inspekcije poboljšanja, ispisuje se iznos *improvement*, *new_cost* i *current_cost.*

Funkcija *move_right_by1_single* (isječak kôda 9) završava s povratkom true ili false, ovisno o tome je li automobil uspješno pomaknut ili ne. U slučaju kada pomicanje automobila nije isplativo i ne donosi poboljšanje, vraća se false, što označava da se neće nastaviti s pomicanjem tog automobila u desno.

Isječak kôda 9: Primjena funkcije *bool move_right_by1_single*

```
bool move_right_by1_single(Solution & solution, const
vector<vector<vector<int>>> & cost, const vector<vector<vector<int>>> &
costOfReturningVehicle, int start_segment) {
    double improvement = 0.0;
    int last_index = solution.dimension - 1;
    int i = start_segment;
    int j = (i + 1) % solution.dimension;

    if (start_segment == 0) {
        // It is the starting point and the first vehicle, so it cannot
move to the right
        return false;
    } else {
        int previous_vehicle = solution.carList.at(start_segment - 1);
        int current_vehicle = solution.carList.at(start_segment);
        improvement =
cost[previous_vehicle][solution.nodeList[i]][solution.nodeList[j]] -
cost[current_vehicle][solution.nodeList[i]][solution.nodeList[j]];
        double current_cost =
cost[current_vehicle][solution.nodeList[i]][solution.nodeList[j]];
        double new_cost =
cost[previous_vehicle][solution.nodeList[i]][solution.nodeList[j]];
        if (current_vehicle != solution.carList[0] && solution.carList[i]
!= solution.carList[j])
        {
            int prev_vehicle = solution.carList[start_segment - 1];
            int node_i = solution.nodeList[i];
            int node_j = solution.nodeList[j];
            new_cost +=
costOfReturningVehicle[prev_vehicle][node_i][node_j];
        }
        improvement = current_cost - new_cost;

        // Check if there is an improvement
        if (improvement > 0) {
            // Move the vehicle to the right node segment
            solution.carList[i] = previous_vehicle;

            // Additional logging to inspect improvements
            cout << "Improvement: " << improvement << ", New Cost: " <<
new_cost << ", Old Cost: " << current_cost << endl;
```

```
            solution.cost -= improvement;
            return true;
        } else {
            return false;
        }
    }
}
```

Izvor: vlastita izrada

Funkcija *AppendToCsv* koja je opisana u isječku koda X ima zadatak dodavanja rezultata analize u CSV (*Comma-Separated Values*) datoteku. Ova funkcionalnost iznimno je korisna u kontekstu analize eksperimentalnih rezultata ili praćenja promjena tijekom vremena, omogućujući organizaciju i jednostavno čitljiv format podataka.

Prilikom pozivanja funkcije, proslijeđuju se sljedeći parametri:

- *filename*: Ime datoteke u koju će se dodavati podaci.
- *problemName*: Naziv problema za koji su analizirani rezultati.
- *nAnts*: Broj mrava koji je korišten u analizi.
- *nIterations*: Broj iteracija koje su provedene u analizi.
- *resultWithoutOpt*: Rezultat problema prije primjene bilo kakvih optimizacija.
- *resultWithOpt*: Rezultat problema nakon primjene određene optimizacije.

Funkcija prvo pokušava otvoriti ciljanu CSV datoteku u režimu dodavanja (ios::app). Ako otvaranje datoteke nije uspjelo, ispisuje se odgovarajuća poruka o grešci. Ukoliko je datoteka prazna, funkcija dodaje zaglavlje u CSV formatu koje sadrži nazive atributa pod kojima će se pohranjivati rezultati. Nakon toga, funkcija dodaje redak podataka u CSV datoteku koristeći znak za razdvajanje polja (;) kako bi svako polje bilo jasno identificirano. Konačno, datoteka se zatvara. Ova funkcija omogućuje sustavno i strukturirano pohranjivanje analitičkih rezultata, što olakšava njihovu kasniju analizu, usporedbu i interpretaciju. Njena upotreba pridonosi organizaciji istraživačkog rada te omogućuje jednostavno dijeljenje podataka s drugima.

Isječak kôda 10: Primjena funkcije bool move_right_by1_single

```
void AppendToCsv(const string& filename, const string& problemName, int
nAnts, int nIterations, double resultWithoutOpt, double resultWithOpt)
    ofstream csvFile(filename, ios::app);
```

```
    if (!csvFile) {
        cerr << "Error opening CSV file: " << filename << endl;
        return;
    }
    if (csvFile.tellp() == 0) {
        csvFile << "Problem Name;Number of Ants;Number of Iterations;Result
without Optimization;Result with Optimization" << endl;
    }
    csvFile << problemName << ";" << nAnts << ";" << nIterations << ";" <<
resultWithoutOpt << ";" << resultWithOpt << endl;
    csvFile.close();
}
```

Izvor: vlastita izrada

5.2. Testiranje i rezultati

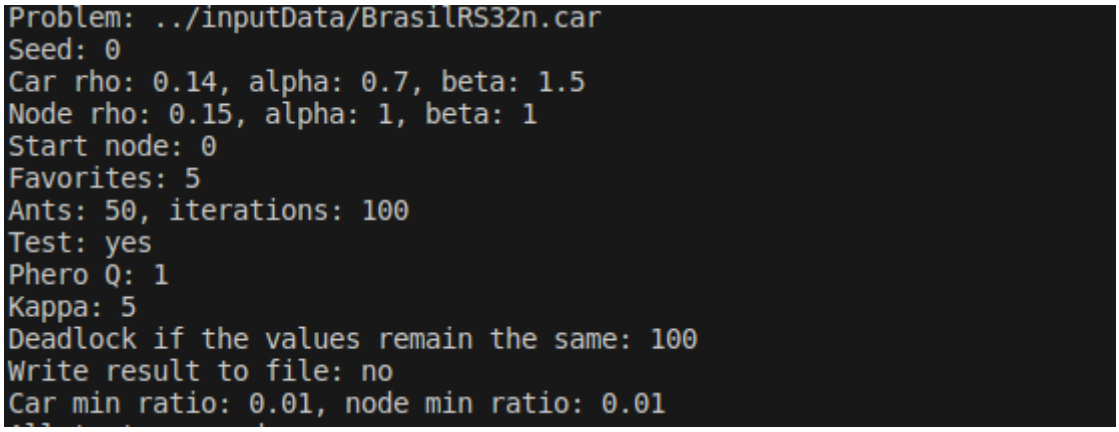
U ovom poglavlju, fokusira se na testiranje i rezultate primjene implementiranog algoritma lokalne optimizacije u svrhu poboljšanja kvalitete rješenja problema iznajmljivanja automobila koji je prethodno riješen algoritmom optimizacije kolonijom mrava. Iako je algoritam kolonijom mrava pokazao uspješnost u pronalaženju rješenja, postoje mogućnosti da ta rješenja nisu potpuno optimalna. Stoga je implementiran algoritam lokalne optimizacije kako bi se dodatno unaprijedila kvaliteta dobivenih rješenja.

U početku poglavlja, prikazuje se postupak testiranja implementiranog algoritma lokalne optimizacije. Kroz testiranje, procjenjuje se učinkovitost algoritma te uspoređuje kvaliteta optimiziranih rješenja s neoptimiziranim. Detaljno je opisan postupak testiranja, uključujući odabir ulaznih podataka i kriterije za ocjenjivanje kvalitete rješenja. Nakon toga, slijedi analiza rezultata testiranja i usporedbe dobivenih rješenja.

Na kraju poglavlja, donosi se sažetak rezultata i zaključak o učinkovitosti implementiranog algoritma lokalne optimizacije u poboljšanju kvalitete rješenja problema iznajmljivanja automobila. Također, razmatraju se mogući izazovi i prilike za daljnje poboljšanje algoritma te se predstavljaju smjernice za buduća istraživanja u ovom području.

Uspješnost postupka lokalne optimizacije uvelike ovisi o problemskom scenariju i parametrima koji su korišteni za algoritam optimizacije kolonijom mrava. Različiti problemi mogu imati različite strukture i karakteristike, što utječe na to kako će lokalna optimizacija djelovati na konačno rješenje. Također, važno je uzeti u obzir parametre algoritma optimizacije kolonijom mrava jer oni mogu znatno utjecati na kvalitetu pronađenog rješenja. Ako je broj iteracija i

mrava u algoritmu optimizacije kolonijom mrava velik, te ako se pronađe optimalno rješenje, lokalna optimizacija možda neće značajno poboljšati to rješenje. U tim situacijama, algoritam optimizacije kolonijom mrava je već konvergirao prema optimalnom rješenju, te dodatna lokalna optimizacija može imati minimalan utjecaj na rezultate. U konkretnom slučaju korištenja funkcije LocalOpt nakon zapisivanja rješenja algoritma optimizacije kolonijom mrava (ACO), rezultati se ispisuju na ekran prvo za rezultat ACO, a zatim za rezultate nakon primjene lokalne optimizacije.



Slika 2: Korišteni parametri za problem BrasilRS32.car

Izvor: vlastita izrada

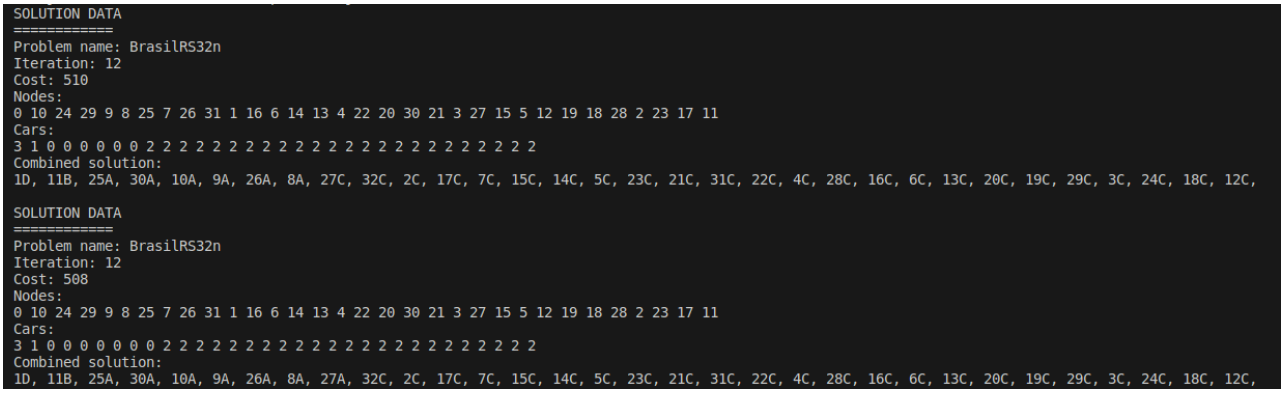
Traži se riješenje za problem BrasilRS23n.car koji Traži se rješenje za problem BrasilRS23n.car koji ima slijedeće parametre:

- NAME: BrasilRS32n -> Naziv problema "BrasilRS32n".
- TYPE: CaRS -> Vrsta problema koji se rješava je tipa "CaRS" (Cars-as-a-Service Problem).
- COMMENT: Instances for the CaRS Problem (Asconavieta & Goldbarg) -> Komentar koji opisuje svrhu i izvor instanci problema "CaRS".
- DIMENSION: 32 -> Broj čvorova (gradova) u problemu, u ovom slučaju ima 32 čvora.
- CARS_NUMBER: 4 -> Broj automobila (vozila) dostupnih u problemu, ukupno je 4 automobila.
- EDGE_WEIGHT_TYPE: EXPLICIT -> Tip težina rubova (cesta) je "EXPLICIT", što znači da su težine cesta eksplicitno navedene.
- EDGE_WEIGHT_FORMAT: FULL_MATRIX -> Format težina rubova (cesta) je "FULL_MATRIX", što znači da su svi parovi težina rubova navedeni u matrici.
- EDGE_WEIGHT_SECTION -> Sadrži stvarne težine cesta u obliku matrice.

Parametri korišteni u ACO algoritmu (slika 2) su postavljeni kako bi omogućili različite postavke i testiranje algoritma. Neki od ključnih parametara su:

- Ants: Broj mrava koji sudjeluju u izgradnji rješenja.
- Iterations: Broj iteracija algoritma.
- Car rho, Node rho: Faktor isparavanja feromona za automobil i čvor.
- Alpha, Beta: Težinski faktori za feromone i heuristiku pri odabiru sljedećeg čvora.

Ovi parametri mogu biti ključni za postizanje boljih rješenja i njihovo podešavanje za određeni problem je važan dio procesa optimizacije.



Slika 3: Rješenje i ispis algoritma

Izvor: vlastita izrada

Može se primijetiti da u rješenju dobivenom pomoću algoritma optimizacije kolonijom mrava (ACO), automobil "A" se koristi od 3. do 8. čvora (grada) u redoslijedu posjećivanja gradova. Međutim, nakon primjene lokalne optimizacije, automobil "A" se sada koristi od čvora 3 do čvora 9. Odnosno, lokalna optimizacija je pronašla bolje rješenje tako da je čvor 27C zamijenjen čvorom 27A, dok je ostatak rješenja ostao nepromijenjen. Kroz ovu promjenu, ukupna cijena rješenja smanjena je sa 510 na 508. Iako se čini da je razlika u cijeni relativno mala, ova promjena pokazuje da je lokalna optimizacija doprinijela boljem rješenju za ovaj konkretan problem. Iako su parametri i rezultati lokalne optimizacije korišteni nakon ACO algoritma, ovakve manje promjene mogu biti značajne za specifične scenarije i mogu poboljšati ukupnu kvalitetu rješenja. Osim toga, važno je napomenuti da uspješnost lokalne optimizacije može varirati ovisno o karakteristikama problema i postavkama algoritma optimizacije kolonijom mrava.

5.3. Analitički prikaz poboljšanja

U ovom poglavlju slijedi analitički prikaz poboljšanja dobivenih primjenom predloženog pristupa optimizacije. Rezultati eksperimenata su sumirani u tabličnom obliku izvedenom iz

generirane CSV datoteke. Ovi podaci pružaju dublji uvid u utjecaj optimizacije na različite primjere problema. U tablici 3 su prikazani relevantni rezultati za svaku varijaciju eksperimenta.

Tablica 3: Rezultati eksperimentalnog postupka

Problem	**Broj mrava**	**Broj iteracija**	**Rezultat bez optimizacije**	**Rezultat sa optimizacijom**	**Poboljšanje**
att48na	5	10	2887	2887	0
att48na	10	10	2886	2815	71
att48na	10	10	2909	2867	42
att48na	10	10	2758	2758	0
att48na	10	10	2621	2502	119
att48na	40	10	2523	2512	11
att48na	40	10	2721	2721	0
att48na	40	10	2022	2013	9
att48na	40	10	2607	2593	14
att48na	40	10	2272	2272	0
att48na	40	40	2399	2274	125
att48na	40	40	2268	2259	9
att48na	40	40	2170	2170	0
att48na	50	100	2248	2237	11
att48na	50	100	2221	2207	14
att48na	3	20	2799	2770	29
att48na	7	49	2255	2244	11
att48na	2	1000	1626	1626	0
att48na	2	1000	1633	1633	0
att48na	4	6	3097	3097	0
att48na	2	6	3137	3137	0
att48na	6	8	2949	2867	82
Brasil16n	6	8	2332	2035	297
Brasil16n	10	24	1899	1495	404
Brasil16n	10	24	2203	2134	69
Brasil16n	4	12	2203	1593	610
Brasil16n	20	30	2647	2647	0
Brasil16n	20	30	2479	2413	66
Brasil16n	400	5	1998	1649	349
Brasil16n	400	8	1910	1910	0
Brasil16n	22	8	2841	2745	96
Brasil16n	2200	800	1344	1344	0
Brasil16n	2	800	1908	1908	0
Brasil16n	7	2	3385	2908	477
Brasil16n	7	20	1814	1795	19
Brasil16n	100000	20	1380	1380	0

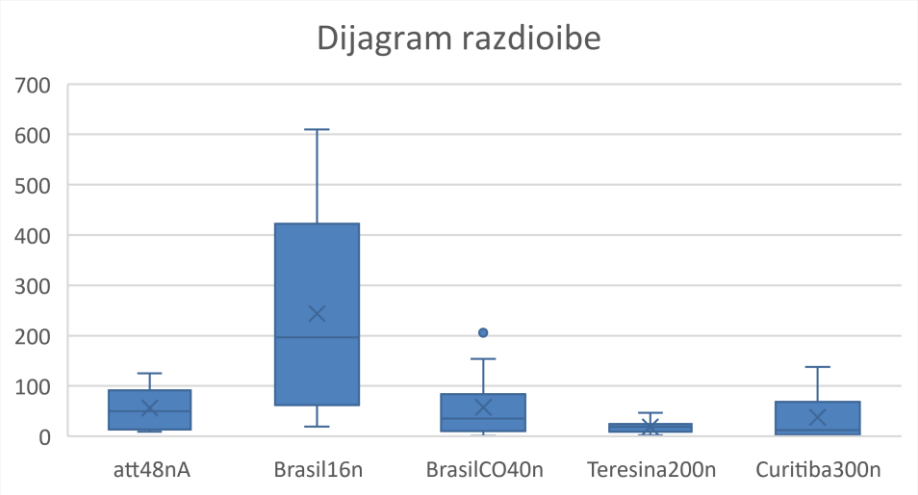
Brasil16n	1000	4	1561	1512	49
BrasilCO40n	1000	4	954	954	0
BrasilCO40n	1000	8	966	760	206
BrasilCO40n	20	8	1141	1140	1
BrasilCO40n	230	3	999	973	26
BrasilCO40n	170	6	1059	999	60
BrasilCO40n	113	5	1026	1026	0
BrasilCO40n	13	3	1152	1152	0
BrasilCO40n	130	3	1134	1090	44
BrasilCO40n	130	5	1109	955	154
BrasilCO40n	130	5	1140	1090	50
BrasilCO40n	150	4	1047	1047	0
BrasilCO40n	43	4	1119	1106	13
Teresina200n	5	10	6240	6229	11
Teresina200n	50	100	5846	5846	0
Teresina200n	50	22	5898	5887	11
Teresina200n	500	8	5729	5682	47
Teresina200n	20	8	5988	5960	28
Teresina200n	200	5	5799	5783	16
Teresina200n	200	50	5656	5638	18
Teresina200n	1000	3	5864	5861	3
Teresina200n	40	199	3083	3067	16
Teresina200n	3	50	6104	6086	18
Teresina200n	200	4	5963	5941	22
Teresina200n	2	20	6144	6121	23
Teresina200n	20	30	5990	5971	19
Teresina200n	15	12	5970	5948	22
Teresina200n	6	3	6464	6462	2
Curitiba300n	6	3	9217	9158	59
Curitiba300n	5	20	9145	9143	2
Curitiba300n	8	20	9064	9064	0
Curitiba300n	80	20	8938	8800	138
Curitiba300n	52	4	8884	8820	64
Curitiba300n	22	40	8981	8917	64
Curitiba300n	18	13	8963	8962	1
Curitiba300n	8	65	8683	8681	2
Curitiba300n	12	3	9076	9066	10
Curitiba300n	4	4	9237	9234	3
Curitiba300n	6	20	9120	9119	1
Curitiba300n	12	4	9184	9179	5
Curitiba300n	33	3	9180	9099	81
Curitiba300n	341	1	9026	9012	14

Izvor: vlastita izrada

Svaki redak u tablici predstavlja određenu konfiguraciju eksperimenta, pružajući informacije o nazivu problema (s obzirom na broj gradova), broju mrava korištenih u algoritmu, broju izvršenih iteracija te početnom i optimiziranom rezultatu. Uočavamo kako optimizacija utječe na svaki problem te koje konfiguracije eksperimenata donose najbolje rezultate.

Primjećuje se da su najveća poboljšanja postignuta kod problema s manjim brojem gradova kao što je Brasil16n, gdje se bilježi impresivno poboljšanje od 610. Analizirajući ovo poboljšanje u kontekstu ostalih eksperimenata, vidljivo je da problem nije blizu optimalnog rješenja, te nisu iskorišteni veliki brojevi mrava ili iteracija. Drugim riječima to je iz razloga što ACO algoritam nije bio blizu optimalnog rješenja pa lokalna optimizacija može rezultirati sa značajnim poboljšanjima. Ovo je iznimno značajno jer naglašava sposobnost algoritma da donese značajne koristi u situacijama gdje su prethodno postignuta rješenja još daleko od optimalnog. Ovaj rezultat naglašava važnost odabira parametara, naročito broja mrava i iteracija, u postizanju većih poboljšanja. Kada se nije blizu optimalnog rješenja i koriste umjereni brojevi mrava i iteracija, algoritam pokazuje veći potencijal za istraživanje i optimizaciju. To dovodi do postizanja većih postignuća, pri čemu se troši manje resursa za generiranje rješenja nego kad bi se koristio povećani broj mrava i iteracija.

Na dijagramu prikazanom slikom 4 vidljivo je kako algoritam reagira na različite probleme, odnosno koliki je raspon poboljšanja algoritma lokalne optimizacije. Evidentno je da je problem Brasil16n imao zapravo najveća poboljšanja, u slučajevima kada je broj mravi i iteracija bio malen kao što je ustanovljeno i s analitičkom analizom tablice 3. Osim ekstrema u navedenim specifičnim okolnostima, vidljiva su poboljšanja i kod kompleksnijih problema s većom količinom iteracija i mrava. Ovisno o parametrima evidentna su poboljšanjima kojima je 100 iznos gornje granice kao rezultat ovog testiranja.



Slika 4: Dijagram razdiobe poboljšanja na različitim problemima

Izvor: vlastita izrada

6. Zaključak

U ovom radu se fokusiralo na problem putujućeg trgovca s iznajmljivanjem automobila (CaRS) te se predstavio i implementirao algoritam lokalne optimizacije s ciljem poboljšanja kvalitete rješenja problema. CaRS problem uključuje dodatne kompleksnosti u odnosu na TSP kao što su troškovi iznajmljivanja, kapaciteti vozila i vremenska ograničenja za povratak automobila na početnu točku. Algoritam lokalne optimizacije se temelji na iterativnom poboljšavanju trenutnog rješenja putem pretrage susjedstva. U početku, koristi se inicijalno rješenje koje sadrži redoslijed gradova i informacije o tome s kojim se automobilom izlazi iz pojedinog grada. Glavni korak algoritma je pomicanje mjesta iznajmljivanja automobila kako bi se poboljšalo ukupno rješenje. Ovaj postupak pomicanja mjesta automobila se ponavlja više puta kako bi se pronašlo lokalno optimalno rješenje.

Implementiran je algoritam lokalne optimizacije specifično prilagođen za CaRS problem, a zatim se testirao na primjeru problema BrasilRS23n.car s različitim postavkama parametara. Testiranje je pokazalo da algoritam lokalne optimizacije može dovesti do značajnih poboljšanja u kvaliteti rješenja. Iako su promjene u cijeni rješenja ponekad bile male, lokalna optimizacija je pokazala sposobnost pronalaženja boljih rješenja za određene scenarije.

Uspješnost algoritma lokalne optimizacije ovisi o problemskom scenariju i parametrima korištenim u algoritmu optimizacije kolonijom mrava. Važno je pravilno odabrati parametre kako bi se postigao najbolji rezultat, ali i razumjeti da postizanje globalnog optimalnog rješenja može biti teško u nekim slučajevima. Algoritam lokalne optimizacije može biti koristan dodatak alatima za rješavanje problema putujućeg trgovca s iznajmljivanjem automobila. Kroz ovaj diplomski rad se pokazuje kako lokalna optimizacija može doprinijeti poboljšanju kvalitete rješenja. Daljnje istraživanje i eksperimentiranje s različitim algoritmima lokalne optimizacije i parametrima moglo bi pružiti još bolje rezultate i otvoriti put ka rješenju složenih problema iznajmljivanja automobila u stvarnom svijetu.

U istraživanju su istaknute prednosti memetičkog algoritma kao perspektivne metode za optimizaciju problema CaRS. Memetički algoritam kombinira globalnu pretragu putem genetskog algoritma s lokalnom pretragom koja poboljšava lokalnu kvalitetu rješenja. Integracija iznajmljivanja automobila u problem putujućeg trgovca predstavlja dodatne izazove, kao što su troškovi, kapacitet vozila i vremenska ograničenja. No, adaptacijom metoda lokalne optimizacije i integriranjem ovih dodatnih varijabli u matematički model, moguće je razviti učinkovite algoritme za rješavanje problema CaRS.

Uspoređene su različite metode lokalne optimizacije za rješavanje problema CaRS, kao što su memetički algoritam i simulirano kaljenje. Analiziraju se njihove performanse u pronalaženju

optimalnih rješenja u kontekstu problema putujućeg trgovca s iznajmljivanjem automobila. Rezultati ukazuju na visoku učinkovitost memetičkog algoritma i njegovu sposobnost pronalaženja optimalnih rješenja u kratkom vremenskom periodu.

Shodno uvodu rada, pronađeni su odgovori na istraživačka pitanja:

1. Kako se metode lokalne optimizacije mogu prilagoditi za rješavanje problema putujućeg trgovca s iznajmljivanjem automobila?

Prilagodba metoda lokalne optimizacije za problem putujućeg trgovca s iznajmljivanjem automobila uključuje kombinaciju globalne i lokalne pretrage te integraciju iznajmljivanja automobila u matematički model radi uzimanja u obzir kapaciteta vozila, vremenskih ograničenja i troškova. Memetički algoritmi i simulirano kaljenje mogu se koristiti za istraživanje rješenja.

2. Koji su specifični izazovi i ograničenja koji proizlaze iz integracije iznajmljivanja automobila u problem putujućeg trgovca?

Integracija iznajmljivanja automobila u problem putujućeg trgovca stvara specifične izazove kao što su dodatni troškovi, kapacitetska ograničenja i vremenska ograničenja vozila. Ovi faktori zahtijevaju prilagodbu matematičkog modela i efikasne algoritme za pretraživanje rješenja.

3. Koje metode lokalne optimizacije su najprikladnije za rješavanje problema CaRS?

Za rješavanje problema putujućeg trgovca s iznajmljivanjem automobila, odabir metoda poput memetičkih algoritama, simuliranog kaljenja, tabu pretraživanja, genetskih algoritama ili mravlji algoritmi ovisi o zahtjevima i ciljevima optimizacije.

4. Kako se performanse različitih metoda lokalne optimizacije uspoređuju u kontekstu problema putujućeg trgovca s iznajmljivanjem automobila?

Usporedba performansi metoda za problem putujućeg trgovca s iznajmljivanjem automobila obuhvaća ocjenu kvalitete rješenja, vremensku učinkovitost i stabilnost algoritama na temelju ukupne duljine puta, troškova te vremena potrebnog za pronalazak rješenja.

5. Kako se mogu razviti efikasne strategije i algoritmi koji će pružiti optimalna rješenja za problem CaRS?

Razvoj efikasnih strategija za problem putujućeg trgovca s iznajmljivanjem automobila zahtijeva teorijski pristup i eksperimentalno istraživanje. Analiza postojećih metoda, modifikacije algoritama te eksperimentalna evaluacija pomažu razviti strategije koje će se nositi s izazovima i ograničenjima problema.

Popis literature

1. Eftimov, T., Korošec, P., Koroušić Seljak, B.: Data-driven preference-based deep statistical ranking for comparing multi-objective optimization algorithms. (2018). Preuzeto 23.7.2023. https://doi.org/10.1007/978-3-319-91641-512
2. Github. Rrepozitorij Elvica Popovića : Feri, preuzeto 15.07.2023. s https://github.com/ElvisPopovic1/FERI
3. Goldbarg, M. C., Asconavieta, P. H., & Goldbarg, E. F. G. (2012). Memetic algorithm for the traveling car renter problem: an experimental investigation. Memetic Computing, 4, 89-108.
4. Grković, V., & Bulatović, R. (2012). Modifikovani algoritam kolonije mrava za rešavanje inženjerskih optimizacionih problema. IMK-14-Istraživanje i razvoj, 18(4).
5. Ivković, N., Golub, M., Malekovic, M.: A pheromone trails model for MAX-MIN ant system. In: Proceedings of 10th Biennal International Conference on Artificial Evolution, Angers, France, pp. 35–46 (2011)Eft
6. Johnson, D. S., & McGeoch, L. A. (1997). The traveling salesman problem: A case study in local optimization. Local search in combinatorial optimization, 1(1), 215-310.
7. Korenić, F. (2019). Metode rješavanja kombinatoričke optimizacije (Doctoral dissertation, University of Pula. Faculty of Informatics in Pula).
8. Mavrovouniotis, M., Müller, F. M., & Yang, S. (2016). Ant colony optimization with local search for dynamic traveling salesman problems. IEEE transactions on cybernetics, 47(7), 1743-1756.
9. Popović, E., Ivković, N., Črepinšek, M. (2022). ACOCaRS: Ant Colony Optimization Algorithm for Traveling Car Renter Problem. In: Mernik, M., Eftimov, T., Črepinšek, M. (eds) Bioinspired Optimization Methods and Their Applications. BIOMA 2022. Lecture Notes in Computer Science, vol 13627. Springer, Cham. Preuzeto 1.7.2023. s https://doi.org/10.1007/978-3-031-21094-5_3
10. Rios, B. H. O., Goldbarg, E. F. G., & Quesquén, G. Y. O. (2017, June). A hybrid metaheuristic using a corrected formulation for the Traveling Car Renter Salesman Problem. In 2017 IEEE Congress on Evolutionary Computation (CEC) (pp. 2308-2314). IEEE.
11. Sandou, G. (2013). Metaheuristic optimization for the design of automatic control laws. John Wiley & Sons.
12. Stanković, M. (2013). Rešavanje nekih problema kombinatorne optimizacije algoritmom tabu pretraživanja. Univerzitet u Beogradu matematički fakultet.

13. Teodorović, D. (2007). Transportne mreže. Univerzitet u Beogradu, Saobračajni fakultet.
14. Wong, L. P., Low, M. Y. H., & Chong, C. S. (2010). Bee colony optimization with local search for traveling salesman problem. International Journal on Artificial Intelligence Tools, 19(03), 305-334.

Popis slika

Popis tablica

Isječci kôda

Korišteni alati

1. Draw.io – izrada grafikona
2. Visual Studio Code – implementacija algortima i lokalna optimizacija
3. Microsoft Excel – izrada izračuna i dijagrama

Kratice

ACO = Ant Colony Optimization, optimizacija kolonijom mrava

ACOCaRS = optimizacija kolonijom mrava na problemu putujećeg trgovca i iznajmljivanjem automobila

CaRS = problem putujućeg trgovca s iznajmljivanjem automobila

GA = Geneteski algoritam

LO = Lokalna opitimizacija

MA = Memetički algoritam

TA = Transgenetski algoritam