SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 2855

Rješavanje problema usmjeravanja električnih vozila korištenjem genetskih algoritama

Marin Ovčariček

Zagreb, rujan 2022.

Ovom prilikom želim se zahvaliti mentoru doc. dr. sc. Marku Đuraseviću na savjetima i strpljenju za vrijeme preddiplomskog i diplomskog studija.
Hvala mojoj obitelji na podršci bez koje cijeli put od osnovne škole do diplome ne bi bio moguć.
Želim se zahvaliti svim prijateljima i kolegama uz koje život za vrijeme studija bio zabavan, manje stresan i nezaboravan.
Također, hvala i kolegama iz firme Tacta koji su svojim savjetima i znanjem pomogli u izradi ovog rada.

SADRŽAJ

1. Uvod

U modernom biznisu cilj je poslovati efikasno, odnosno smanjiti troškove na minimum dok se istovremeno pokušava maksimizirati zarada. To uključuje optimiziranje korištenja različitih resursa kao što su materijali, vrijeme, ljudstvo te alati. Jedna od djelatnosti u kojoj se ti procesi koriste je usmjeravanje vozila. Zadnjih godina pojavilo se puno kompanija kao što su Wolt, Bolt, Glovo, Uber i mnogi drugi kojima je cilj što prije i efikasnije poslužiti svoje klijente. Postoje i dostavne usluge u kojima jedno vozilo poslužuje više korisnika između povrataka u skladište te je za njih također bitno optimalno izabrati rutu kako bi se izbjegli nepotrebni troškovi. S obzirom na to da transport robe ima značajan udio u cijeni samog proizvoda, uštede i od nekoliko postotaka mogu biti značajne. Također, kako se zadnjih desetljeća budi svijest o ljudskom utjecaju na okoliš te cijene energenata značajno rastu problem usmjeravanja vozila postaje sve važniji u poslovnom svijetu. U ovom radu će se obraditi tema usmjeravanja flote električnih vozila (eng. *Electric vehicle routing problem E-VRP*) te se za primjer uzima homogena flota (sva vozila su jednakih karakteristika) koja poslužuje skup klijenata. Bitna svojstva problema koji će se rješavati su:

- Vozilo poslužuje klijente dok više nema dovoljno resursa te se onda vraća u skladište
- Da bi vozilo poslužilo klijenta treba imati jednako ili više resursa nego što on zahtijeva
- Kada vozilo stane na postaji za punjenje uvijek se puni do kraja

1.1. Vehicle Routing Problem (VRP)

Problem usmjeravanje vozila je generalizacija problema trgovačkog putnika (eng. *Traveling salesman problem* -TSP) gdje je cilj naći optimalan skup ruta za flotu vozila kako bi se poslužio skup klijenata. U osnovi parametar po kojem se optimira može biti količina utrošenog goriva, vremena, broja vozila ili kombinacija više kriterija. Sam problem može se nadopuniti dodatnim ograničenjima i svojstvima koja se mogu međusobno kombinirati koje stvaraju nove varijante problema. Neke od vrsta nadopunjenih problema su:

- VRPP (Vehicle Routing Problem with Profits) - u ovom slučaju svi klijenti ne moraju biti posluženi ako bi to maksimiziralo profit. Dodatne specijalizacije ovog problema su TOP (Team Orienteering Problem), te TOP uz ograničen kapacitet CTOP (Capacitated-TOP) ili uz određene vremenske okvire u kojima klijent treba biti poslužen TOPTW (TOP-with Time Windows).
- VRPPD (Vehicle Routing problem with Pickup and Delivery) - vozila moraju obići određene postaje na kojima skupljaju pakete i onda ih razvoze klijentima. Dodatno ograničenje koje se može razmatrati je LIFO (eng. *Last In First Out*) što znaci da predmet koji se predaje mora biti onaj koji je zadnji stavljen u spremnik.
- VRPMT (Vehicle Routing Problem with Multiple Trips) - vozila mogu odraditi više ruta.
- OVRP (Open Vehicle Routing Problem) - vozila se ne moraju vratiti u postaju.
- MDVRP (Multiple Depots Vehicle Routing Problem) - postoji više čvorova iz kojih vozila mogu krenuti i u kojima mogu završiti.

1.1.1. Electric Vehicle Routing Problem (E-VRP)

E-VRP Kucukoglu et al. (2021) je specijalizacija problema usmjeravanja vozila gdje se koristi isključivo flota električnih vozila. Ovaj problem je sve aktualniji zbog rasta industrije električnih vozila te njihove sve veće zastupljenosti među korisnicima. Električna vozila uvode nova ograničenja: domet električnih vozila je puno manji od onih s motorom s unutarnjim izgaranjem. Također, u stvarnom svijetu postaja za punjenje električnih vozila ima puno manje nego benzinskih crpki te električna vozila mogu imati nelinearnu brzinu punjenja baterije.

1.2. Genetski algoritam

Genetski algoritam Verma i Kumar (2014) je metaheuristika za rješavanje optimizacijskih problema. Spada u skup Evolucijskih algoritama te je inspirirana procesom prirodne selekcije. Rješenja se prvo kodiraju u "jedinke" u skupu koji se naziva populacija. Kroz iteracije algoritma sama populacija "evoluira". Jedinke odnosno rješenja se miješaju i mutiraju te one najbolje opstaju. Najbitniji dijelovi genetskog algoritma (operatori) su:

- Križaje - Operator križanja (eng. *crossover*) zaslužan je za stvaranje novih jedinki od izabranih roditelja kako bi se dobre gene proslijedilo novoj generaciji
- Mutacija - Mutacija je operator koji služi za održavanje raznolikosti u populaciji rješenja. To potencijalno može imati pozitivan učinak ispadanja iz lokalnih optimuma. Mutacija se obično vrši nad jedinkama dobivenim križanjem s nekom vjerojatnošću, ali može se vršiti i nad cijelom populacijom u iteraciji.
- Selekcija - Selekcija je operator koji služi za izdvajanje jedinki koje će operatorom crossovera stvoriti nove jedinke s potencijalno boljim rasporedom gena. Obično u selekciji se uzimaju najbolje jedinke jer analogno s bićima u prirodi najbolje jedinke stvaraju najzdravije i uspješne potomke.

Logika i tok genetskog algoritma je prikazana pseudokodom iz rada na slici 1.1 .

GA(S)

parameter(s): S – set of blocks
output: superstring of set S

Initialization :
$t \leftarrow 0$
Initialize P_t *to random individuals from* S^*
EVALUATE-FITNESS-GA(S, P_t)

while *termination condition not met*
do
- *Select individuals from* P_t *(fitness proportionate)*
- *Recombine individuals*
- *Mutate individuals*
- EVALUATE-FITNESS-GA$(S, modified\ individuals)$
- $P_{t+1} \leftarrow$ *newly created individuals*
- $t \leftarrow t + 1$

return *(superstring derived from best individual in* P_t*)*

procedure EVALUATE-FITNESS-GA(S, P)
S – *set of blocks*
P – *population of individuals*
for each *individual* $i \in P$
do
- *generate derived string* $s(i)$
- $m \leftarrow$ *all blocks from* S *that are not covered by* $s(i)$
- $s'(i) \leftarrow$ *concatenation of* $s(i)$ *and* m
- $fitness(i) \leftarrow \frac{1}{\|s'(i)\|^2}$

Slika 1.1: Pseudokod genetskog algoritma

2. Implementacija genetskog algoritma

U sklopu rada napravljen je mali framework za rješavanje EVRP problema genetskim algoritmom u jeziku Java. Framework omogućuje učitavanje instanci iz različitih skupova podataka koji mapiraju te rješavaju kroz zajedničko sučelje. Također, moguće je pokretanje više instanci genetskog algoritma kroz proizvoljan broj dretvi kako bi se ubrzalo dobivanje većeg broja rezultata. Pseudokod implementacije algoritma prikazan je na slici 2.1, te su njegove komponente objašnjene u narednim poglavljima.

```
while (!isConverging(population)) {

            parents = select(population);

            population = eliminate(population);

            offspring = []
            for (parents : selected) {
                offspring.add(crossover.(parents)


                offspring = mutation(offspring)
            }
            population.add(offspring)

            population.addRandomSolutionsUntilPopulationFull()
}
```

Slika 2.1: Pseudokod prilagođenog genetskog algoritma

2.1. Kodiranje rješenja

Rješenja su prikazana kao niz identiteta lokacija klijenata. Vozilu se pridružuju klijenti redoslijedom kako su zapisani u rješenju dok god nisu posluženi i dok god je vozilo u mogućnosti poslužiti ga. Ovakav pohlepni pristup izabran je jer iako smanjuje mogući

prostor rješenja u isto vrijeme znatno olakšava implementaciju samog algoritma. U zapisu rješenja nisu zapisana skladišta jer se za svako vozilo zna koja je početna i odredišna lokacija. Cijeli proces enkodiranja je dodatno pojašnjen pseudokodom na slici ispod.

```
currentVehicle = new Vehicle
for (location : locations) {

    if(!vehicle.canServiceCustomer(location)){
        currentVehicle.goToDepot()
        currentVehicle = new Vehicle
    }
    passIntermediateChargingStations(currentVehicle, location)
    currentVehicle.travelTo(location)

   }
 currentVehicle.goToDepot()
```

Slika 2.2: Primjer rješenja

```
currentVehicle = new Vehicle
for (location : locations) {

    if(!vehicle.canServiceCustomer(location)){
        currentVehicle.goToDepot()
        currentVehicle = new Vehicle
    }
    passIntermediateChargingStations(currentVehicle, location)
    currentVehicle.travelTo(location)

   }
 currentVehicle.goToDepot()
```

Slika 2.3: Pseudokod pridruživanja klijenata vozilu

Prije slanja vozila na neku lokaciju koja nije skladište poziva se metoda provjerava se uvjet canReachLocationAndNearestChargingStation ima li vozilo dovoljno goriva da dođe do lokacije i najbliže postaje za punjenje. Ovaj uvjet se postavlja i ispituje kako bi se osiguralo da vozilo nikada ne dođe u situaciju u kojoj, nakon posjete nekoj lokaciji, nema dovoljno goriva za pomak do najbliže postaje za punjenje. Ovaj uvjet osigurava da se vozilo sigurno transportira bez čekanja i eventualnog ostanka bez goriva na spomenutoj lokaciji. Ako taj uvjet nije zadovoljen, vozilo će se kretati između odabranih postaja sve dok uvjet ne bude zadovoljen. Postaje se odabiru po principu da se uzima najbliža postaja krajnjoj lokaciji, a da u isto vrijeme zadovoljava ranije spomenut uvjet. Postaje za punjenje nisu u zapisu rješenja jer su jednoznačno određene prethodno objašnjenom heuristikom.

```
public void passIntermediateChargingStations(vehicle,destination){

    while(!canReachLocationAndNearestChargingStation(vehicle, destination)) {
    Location chargingStation =
        chooseIntermediateChargingStation(vehicle, destination)

    currentVehicle.travelTo(chargingStation)
    }
}
```

Slika 2.4: Pseudokod prolaska postajama za punjenje na putu do destinacije

```
chooseIntermediateChargingStation(Vehicle vehicle, Location targetLocation) {
    return getChargingStations()
        .filter(canReachLocationAndNearestChargingStation(vehicle, location))
        .min(location -> euclideanDistanceTo(location, targetLocation)))
    }
```

Slika 2.5: Pseudokod odabira postaje za punjenje

2.1.1. Struktura zapisa

Svaka lokacija se pridružuje genu. Gen je objekt koji sadrži lokaciju te težinsku vrijednost koja može poslužiti u različitim vrstama zapisa npr. Random keys Wester (1993). Geni se grupiraju u listu te se stvara kromosom koja predstavlja jedinku, odnosno moguće rješenje. Osim gena, jedinka sadrži i izračunatu vrijednost svoje funkcije dobrote (eng. *fitness*). U ovom radu se koristi funkcija kazne, odnosno što je vrijednost manja jedinka ima veći fitness. Jedinke se grupiraju u populaciju koja se mijenja kroz iteracije algoritma. Svaka populacija ima zadan maksimalni broj jedinki koji se održava konstantnim kroz svaku iteraciju.

2.2. Operatori

Operatori su funkcije koje navode genetski algoritam prema rješenju, generalno se dijele na tri glavne skupine: crossover, mutaciju i selekciju. Inspirirani su životnim ciklusom bića u prirodi. U implementaciji za ovaj rad postoji i operator eliminacije kako bi se omogućilo dodatno podešavanje ponašanja algoritma.

2.2.1. Selekcija

Neki od često korištenih operatora selekcije su *Roullete wheel*, *Rank selection* te *Stohastic universal sampling* dok je u ovom radu korištena turnirska selekcija s veličinom

turnira 3. U turnirskoj selekciji se uzima N nasumičnih jedinki te se od njih uzima ona najbolja kao jedinka "roditelj".

```
public interface Selection {

    List<Pair<GAChromosome>> select(Population population);
}
```

Slika 2.6: Sučelje selekcije

2.2.2. Eliminacija

U ovom radu uključuje se i novi operator eliminacije kojim se postiže bolja kontrola nad populacijom. Eliminacija se vrši odmah nakon selekcije te prije crossovera. Eliminacijom se miču nepoželjne jedinke kako bi se oslobodilo mjesta za potencijalno bolja rješenja. Za rješavanje instanci korištena je eliminacija s elitizmom. Ovaj operator prima decimalanu vrijednost kao parametar elitizma koji određuje udio najboljih jedinki koji će sigurno preživjeti eliminaciju.

```
public interface Elimination {

    Population eliminate(Population population);
}
```

Slika 2.7: Sučelje eliminacije

2.2.3. Crossover

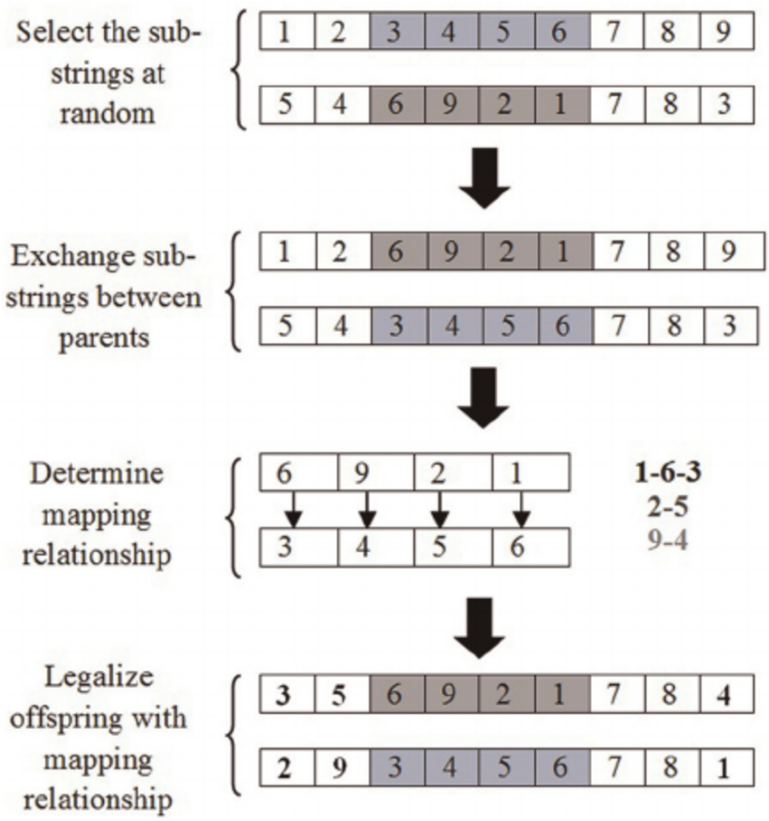
U ovom radu su korišteni operatori crossovera specifični za TSP (eng. *Travelling salesman problem*) preuzeti iz Umbarkar i Sheth (2015), odnosno osigurano je da su geni lokacije jedinstveni prema svojem identitetu te svaka lokacija posjećuje samo jednom. Trenutna implementacija također podržava korištenje više crossovera gdje se svakim pozivom funkcije odabere jedan od crossovera iz liste s jednakom šansom.

```
public interface Crossover {

    Collection<List<Allele>> crossover(List<Allele> parent1, List<Allele> parent2);

    String getName();
}
```

Slika 2.8: Sučelje crossovera

Djelomično mapirano križanje

Djelomično mapirano križanje (eng. *Partially mapped crossover* - PMX) započinje tako da se odredi interval prekida na oba roditelja. Na cut intervalu se tada stvaraju mapiranja koja određuju transformacije gena iz jednog roditelja u drugog, transformacija su translativne te se ih može nizati proizvoljno mnogo. Nakon toga dijete 1 dobiva cut interval roditelja 2 dok dijete 2 dobiva cut interval roditelja 1. Zatim se prazna mjesta, dok god ih ima u djetetu 1 nadopunjuju s genima iz roditelja 1 redoslijedom kako su zapisani, ako taj gen već postoji u djetetu radi se transformacija prema već zapisanim mapiranjima. Prema primjeru sa slike 2.8 prvo dijete je dobilo gene (6-9-2-1) iz intervala prekida drugog roditelja. Prilikom preslikavanja prvog gena prvog roditelja uočeno je da gen 1 već postoji u zapisu te se mapiranjem dobilo da na to mjesto treba staviti gen 6, kako i gen 6 već postoji preko mapiranja se na to mjesto stavio gen 3. Analogan postupak se radi i za drugo dijete.



Slika 2.9: Djelomično mapirano križanje

Križanje modificiranog redoslijeda

Križanje modificiranog redoslijeda (eng. *Modified order crossover* - MOC) započinje odabirom nasumične točke presjeka na kromosomima roditelja koja ga dijeli na lijevi i desni dio. Kromosom dijete zadržat će gene jednog roditelja na istim mjestima, ako su oni sadržani u lijevom dijelu drugog roditelja. Ostali kromosomi nadopunjuju se onim redoslijedom kako su zapisani u desnom dijelu drugog roditelja.

For example with the following parents and crossover point

s1 = (1 2 3 4 | 6 9 8 5 7) and

s2 = (2 1 9 8 |5 6 3 7 4)

After position selection

s1 = (1 2 * * * 9 8 * *) and

s2 = (2 1 * * * * 3 * 4)

Now obtain the generated pair of children as

b1 = (1 2 5 6 3 9 8 7 4) and

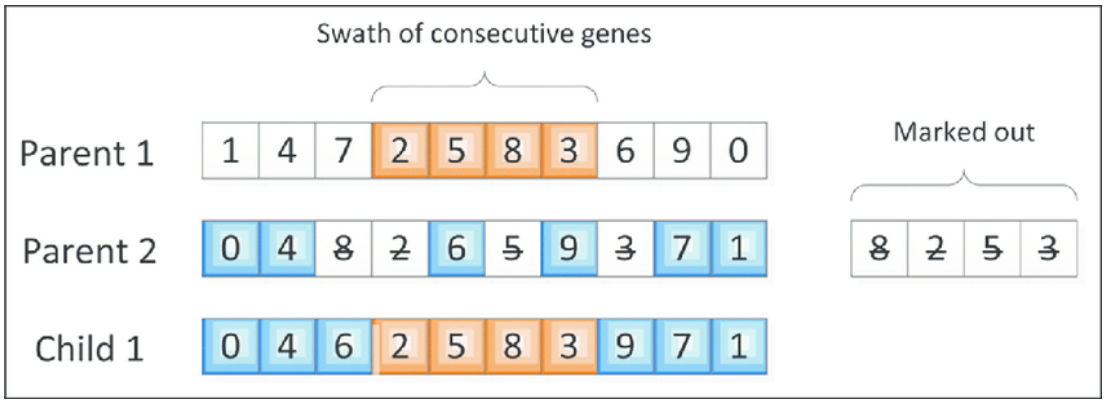
b2 = (2 1 6 9 8 5 3 7 4)

Clearly this method allows only the generation of valid strings.

Slika 2.10: Križanje modificiranog redoslijeda

Redoslijedno križanje

Redoslijedno križanje (eng. *Ordered crossover* - OX) održava raspored gena za vrijeme stvaranja novih jedinki. Podniz gena jednog roditelja mapira se na iste pozicije u djetetu dok se ostali geni, odnosno lokacije koje još nisu posjećene mapiraju na slobodna mjesta u kromosomu u redoslijedu kako su raspoređeni u drugom roditelju. Za drugo dijete crossover radi se analognim postupkom, jedino se zamijene uloge dvaju roditelja.



Slika 2.11: Ordered crossover

2.2.4. Mutacija

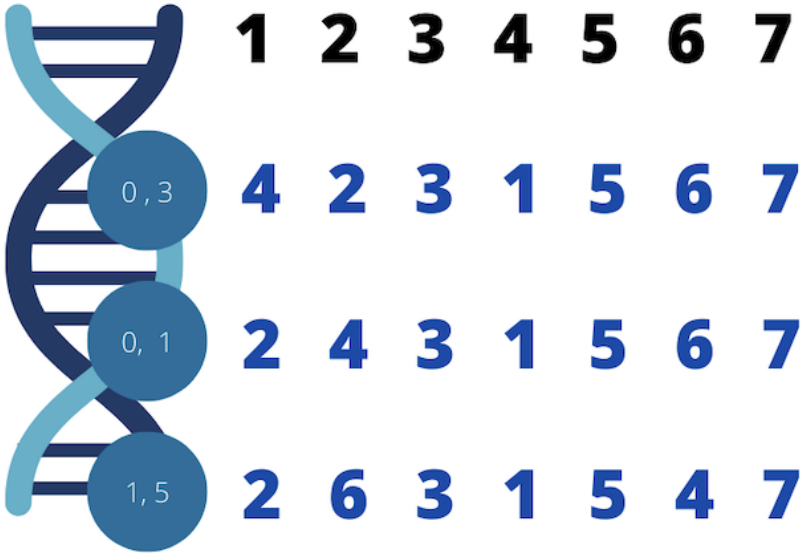
Implementacije mutacija su također prilagođene problemu putujućeg trgovca što osigurava da klijenti budu posjećeni samo jednom. Sve implementacije primaju dodatan parametar stope mutacije kojom se određuje učestalost mutacija po iteraciji.

```
public interface Mutation {

    List<Allele> mutate(List<Allele> alleles);

    String getName();
}
```

Slika 2.12: Sučelje mutacije

Nasumična zamjena gena

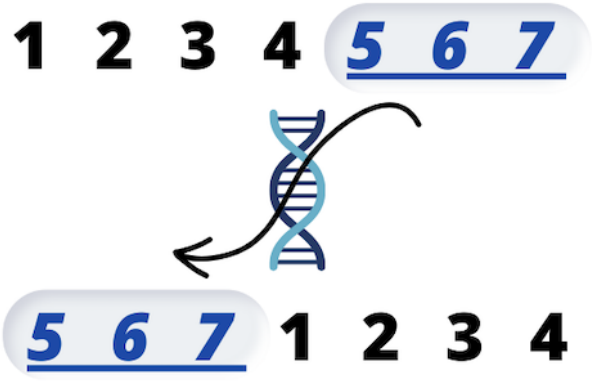
Prilikom ove mutacije nasumično se radi zamjena N parova gena po mjestu na kromosomu, odnosno mijenja se redoslijed posjećivanja za dvije lokacije. Jedan od mogućih slučajeva mutacije prikazan je na slici 2.12. Nasumična zamjena u općenitom slučaju vrši male promjene gena na više lokacija.



Slika 2.13: Nasumična zamjena gena

Ciklička mutacija

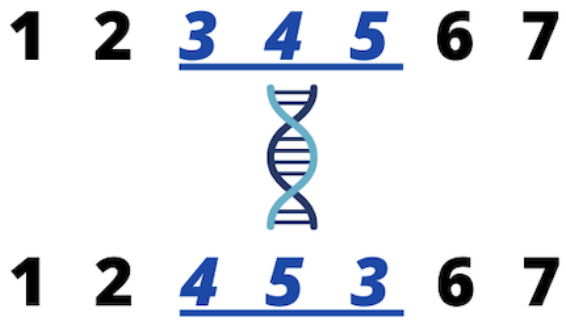
Ciklična mutacija prima dodatan parametar iznos posmaka. Kromosom se modificira tako da se niz gena kružno posmakne u nekom smjeru. U implementaciji ovog rada radi se posmak udesno. Ova mutacije efektivno mutira cijeli kromosom, ali općeniti redoslijed lokacija ostaje isti.



Slika 2.14: Ciklička mutacija

Perturbacija gena u sekvenci

Nasumično se nalaze dva mjesta na kromosomu koji su udaljeni za unaprijed zadan iznos gena, te se geni između ta dva mjesta perturbiraju, dok ostali geni ostaju na istom mjestu. Kod peturbacije kromosom se mijenja samo lokalno.



Slika 2.15: Perturbacija gena

2.3. Evaluacija rješenja

Evaluatori računaju funkciju dobrote pojedinog rješenja. Kao ulaznu vrijednost primaju SolutionContext. To je objekt koji sadrži informacije o svojstvima vozila i lokacija, položaju lokacija te broju korištenih vozila i njihovim stanjima (pređen put, posjećene lokacije, proteklo vrijeme itd.) U ovom radu implementirani su vremenski evaluator koji zbraja proteklo vrijeme za svako vozilo, njemu sličan evaluator za udaljenost koji zbraja pređen put svakog vozila te evaluator vremenskih okvira koji računa fitness na temelju točnost dostave vozila za svaku lokaciju. Svi implementirani evaluatori zapravo računaju funkciju kazne, što znači da je fitness jedinke veći što joj je kazna manja.

```
public interface SolutionEvaluator {

    double evaluate(SolutionContext solution);

}
```

Slika 2.16: Sučelje SolutionEvaluator

2.3.1. Evaluator vremenskih okvira

Evaluator vremenskih okvira (eng. *Time window evaluator* je nadogradnja vremenskog koji uz to što zbraja ukupno vrijeme dodatno kažnjava posluživanje klijenata prerano ili prekasno prema formuli tako da se na utrošeno vrijeme nadoda vrijeme koliko je vozilo uranilo ili zakasnilo.

2.4. Pomoćne komponente

Implementirani Genetic algorithm framework osim samog rješavača sadrži i pomoćne komponente prikazane sučeljima kako bi se framework mogao prilagoditi različitim datasetovima i formatima rezultata.

2.4.1. InstanceLoader

Uloga InstanceLoadera je učitavanje instance problema preko tekstualne datoteke te mapiranje lokacija, njihovih koordinata te svih svojstava lokacija, vozila i samog problema u objekt tipa Instance kojeg program zna koristiti. Neka od mogućih svojstava koja se mapiraju su:

- Vozila - prosječna brzina, brzina punjenja, funkcija punjenja, kapacitet prtljažnika, kapacitet rezervoara
- Lokacije - tip lokacije, koordinate, količina zahtijevanih resursa, vrijeme istovara, očekivani vremenski interval za dostavu

```
public interface InstanceLoader {

    Instance load(String path) throws IOException, JDOMException;
}
```

Slika 2.17: Sučelje InstanceLoader

```
public interface Instance {

    Set<Location> getLocations();

    Set<InstanceProperty>  getInstanceProperties();

    Set<VehicleProperty> getVehicleProperties();

}
```

Slika 2.18: Sučelje Instance

2.4.2. CallbackAction

CallbackAction je sučelje koje omogućava korisniku da obavlja akcije nakon svakog koraka u iteraciji algoritma. Trenutno je moguće odraditi akciju nakon selekcije, crossovera, mutacije, iteracije te na kraju samog algoritma. Također postoji opcije kloniranja koja služi za stvaranje CallbackActiona istog tipa s drugim parametrima, a koristi

se kako bi svaka dretva koja izvršava genetski algoritam imala akcije s posebnim postavkama. U predanalizi se koristio CallbackAction koji je zapisivao fitness najbolje jedinke na kraju algoritma u file jedinstven za trenutne hiperparametre.

```java
public interface CallbackAction {

    void resultAction(Population population);

    void crossoverAction(Population population);

    void selectionAction(Population population);

    void mutationAction(Population population);

    void iterationAction(Population population);

    CallbackAction cloneWithDifferentConfiguration(String configuration);
}
```

Slika 2.19: Sučelje CallbackAction

2.4.3. ConvergenceChecker

Sučelje kojim se određuje trenutak terminiranja algoritma. Nakon svake iteracije implementacija ovog sučelja prima trenutnu populaciju te prema nekom kriteriju određuje treba li zaustaviti izvršavanje. Trenutne implementacije podržavaju zaustavljanje nakon određenog broja iteracija ili nakon proteklog vremena u kojem se može odrediti u različitim vremenskim jedinicama (milisekunde, sekunde, minute, sati). Također je moguće koristiti i provjeru koji će logirati kretanje fitnessa najbolje jedinke kroz iteracije.

```java
public interface ConvergenceChecker {

    boolean isConverging(Population population);

    default String evaluate(Population population) throws OperationNotSupportedException {
        throw new OperationNotSupportedException();
    }

    ConvergenceChecker deepCopy();

    String getName();
}
```

Slika 2.20: Sučelje ConvergenceChecker

3. Rezultati

3.1. Skup podataka

Instance problema koje će se rješavati u ovom radu su iz skupa podataka predstavljenog u Goeke (2019). Radi se o 93 instance s postajama za punjenje i vremenskim okvirima u kojima klijenti žele biti posluženi. Instance se dijele u više kategorija (C1, C2, R1, R2, RC1, RC2). C označava instance čiji su klijenti generirani prema već poznatom rješenju, u R su klijenti nasumično raspoređeni na kvadratnom području. RC sadržava kombinaciju nasumično raspoređenih i klasteriranih klijenata. Instance označene s 1 imaju uske intervale za klijente i mali kapacitet vozila dok one označene s 2 imaju znatno manja ograničenja. Format zapisa instanci je sljedeći: redci u datoteci do prvog praznog retka označuju lokacije te imaju svojstva podijeljenih u osam stupaca.

- Jedinstveni slijed znakova (identitet lokacije)
- Tip lokacije : d-skladište (eng. Depot), f- postaja za punjenje (eng. *Recharging station*), c- klijent (eng. *customer*)
- x koordinata
- y koordinata
- Količina resursa koje lokacija zahtijeva
- Trenutak u kojem je lokacija spremna da primi resurse
- Trenutak do kojeg lokacija očekuje resurse
- Vrijeme koje vozilo provede na lokaciji dok radi istovar

Nakon lokacija u pet redaka se opisuju svojstva vozila:

- Q - Kapacitet spremnika goriva
- C - Kapacitet prtljažnika vozila
- r - Stopa potrošnje goriva u jednoj jedinici vremena

– g - Stopa inverznog punjenja, broj jedinica vremena potrebnih za napuniti jednu jedinicu energije

– v - Prosječna brzina, jednaka na svim bridovima

3.2. Predanaliza

Predanaliza se sastoji od računanja optimalnih vrijednosti hiperparametara za dani skup podataka. Hiperparametri se optimiraju sljedećim redoslijedom: veličina populacije, vjerojatnost mutacije, vrsta crossovera, vrsta jedinke. Dok se optimira parametar ostali se drže na istim vrijednostima, nakon što se pronađe optimalna vrijednost iz preddefiniranog skupa ona se "zamrzava" te se prelazi na sljedeći parametar. Predanaliza se odvila na nasumično odabranoj petini ukupnog dataseta. Instance koje su se koristile u predanalizi su: [rc107_21, c106C15, rc208_21, rc106_21, c101_21, c208_21, r105C15, r209C15, r112_21, rc205C10, c103C15, r102_21, r103C10, r208_21, r201_21, c104_21, r202_21, rc103C15, rc103_21]. Hiperparametri koji su u početku fiksirani su:

– Svaka instanca problema izvodi se 10 puta s vremenskim trajanjem 10 minuta

– Selekcija je turnirska, te je veličina turnira 3. U selekciji se odabire broj jedinki "roditelja" koji odgovara 60% od ukupne veličine populacije

– Eliminacija je elitistička, odnosno udio najboljih jedinki uvijek preživljava, u ovom radu se koristi elitizam od 20%

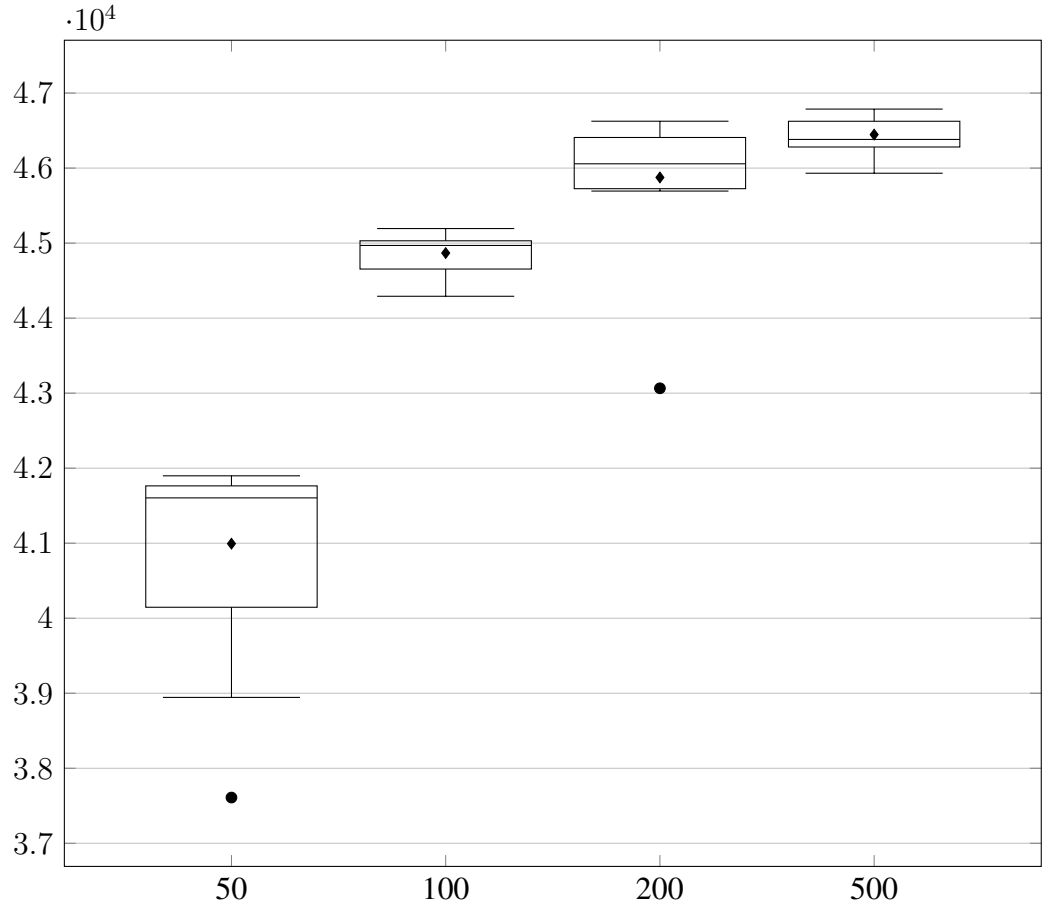
– Kod cikličke mutacije posmak se radi udesno te je veličina posmaka 3

– Kod mutacije zamjene parova radi se zamjena 3 para gena

Rezultati za svaku kombinaciju parametara se računaju tako da se svaka instanca pokrene N puta (u ovom slučaju 10), na kraju algoritma se ispiše fitness najbolje jedinke te se radi suma po rednom broj izvođenja. Iz N suma se stvara kutijasti dijagram kojim se može lijepo prikazati minimalna, maksimalna, medijan te raspršenost rezultata. Za predanalizu korišten je evaluator koji računa ukupan put koji vozila prijeđu dok poslužuju klijente. Manja vrijednost predstavlja bolje rješenje.

3.2.1. Veličina populacije

Za veličinu populacije izabran je skup mogućih vrijednost [50,100,200,500]. Za vrijeme optimiranje veličine populacije korišteni su Ordered crossover te Ciklička mutacija sa stopom mutacije 0.3. Rezultati optimiranja su prikazani na box plotu ispod.

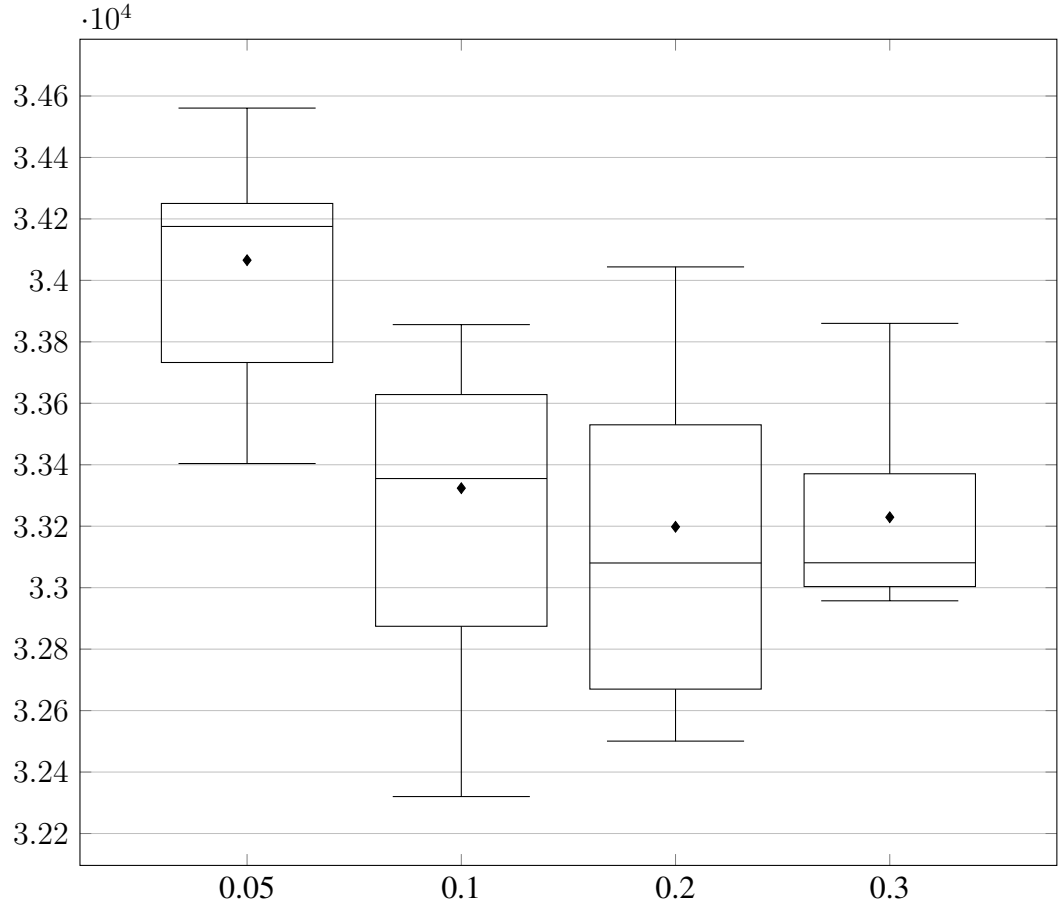


Slika 3.1: Rezultati optimiranja veličine populacije

Prema rezultatima može se uočiti logaritamski rast funkcije kazne s porastom veličine populacije. Iako je veličina 50 najmanje stabilna daje daleko najbolje rezultate. Uspješnost male populacije može se objasniti kroz vremenski uvjet konvergencije od 10 minuta. Jedna iteracija algoritma s manjom populacijom će se odviti puno brže nego one s većom te joj dalje prednost u pronalasku boljeg rješenja. Također veličina populacija može ovisi i veličini instance (u ovom slučaju broj lokacija). Prema Gotshall i Rylander optimalna veličina populacije raste logaritamski s veličinom instance. Također dolaze do zaključka da s porastom populacije raste i broj potrebnih generacija (iteracija) kako bi postigla konvergencija što je u skladu s dobivenom rezultatima.

3.2.2. Stopa mutacije

Skup stopa mutacije koje će se testirati je [0.05, 0.1, 0.2, 0.3]. U prethodnom koraku određena je veličina populacije 50, dok je za ostale parametre ostavljen Ordered crossover te Ciklička mutacija.

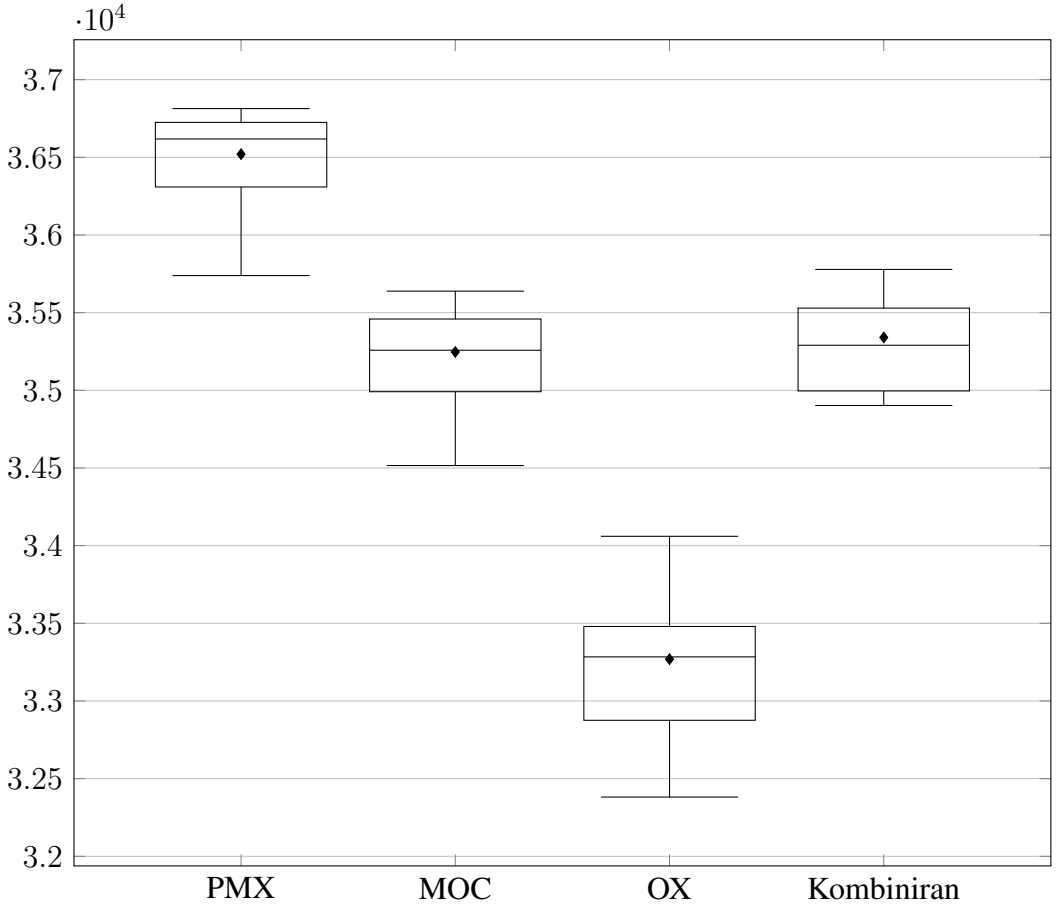


Slika 3.2: Rezultati optimiranja stope mutacije

Dobiveni rezultati ne daju jasnu sliku optimalne stope mutacije. Stopa od 0.05 se pokazala najgorom dok ostale variraju po raspršenosti whiskera i kvartila u oba smjera. Za daljnje optimiranje izabrana je stopa od 0.2 zbog najniže srednje vrijednosti i zadovoljavajućeg donjeg kvartila.

3.2.3. Vrsta crossovera

Kod predanalize crossovera korišteni su PMX, OX, MOC te crossover operator koji nasumično s istom vjerojatnošću bira jedan od prethodna tri operatora. Za mutaciju je i dalje korištena Ciklička mutacija.

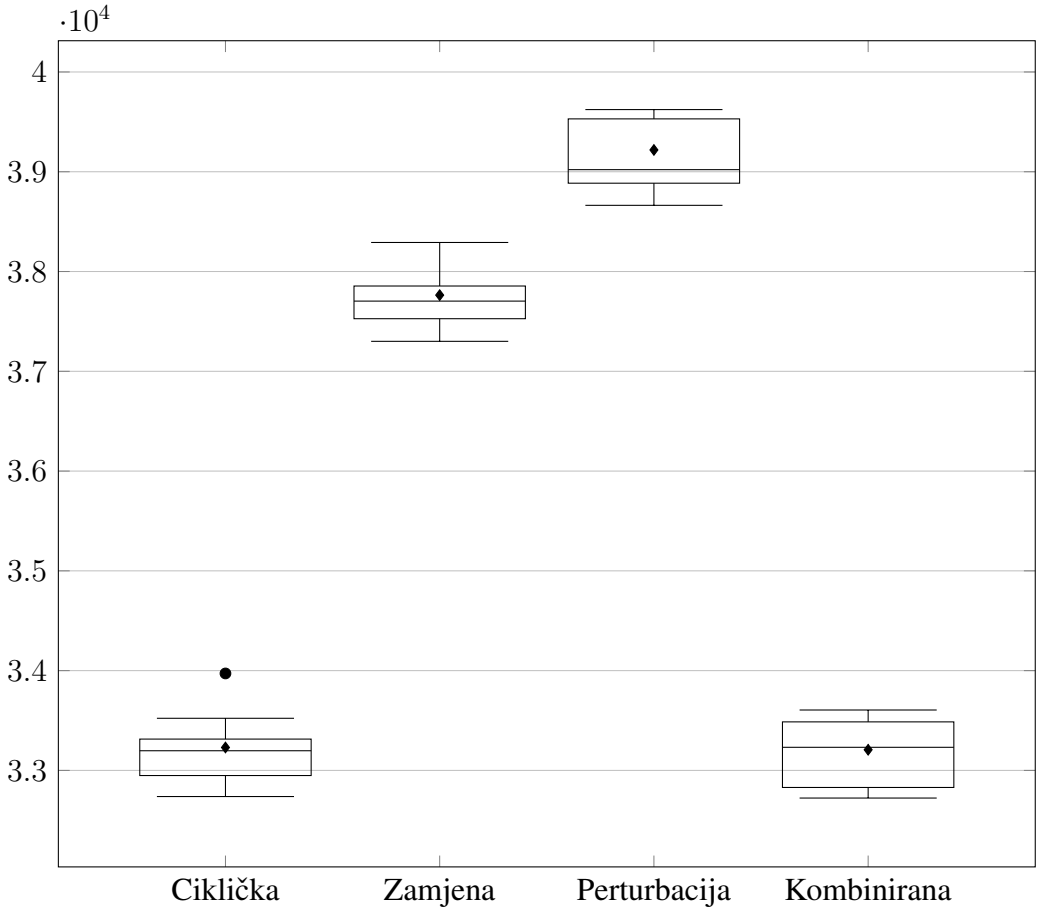


Slika 3.3: Rezultati optimiranja vrste crossovera

Za konkretni problem Ordered crossover se pokazao kao najbolja opcija. Postoji mogućnost inherentne pristranosti s obzirom na to da se i prethodne optimizacije koristile taj isti crossover te kako hiperparametri ne moraju biti nužno nezavisni.

3.2.4. Vrsta mutacije

Analogno kao i za crossover, za optimiranje mutacija korišteni su svi prethodno definirani operatori i uz njih poseban operator koji provodi mutaciju koristeći jedan od tih operatora s jednakom vjerojatnošću.



Slika 3.4: Rezultati optimiranja vrste mutacije

Ciklička mutacija se pokazala daleko najboljom od ostalih opcija, kombinirana mutacija joj dolazi blizu, ali kako ona može u nekim slučajevima koristiti češće dvije lošije mutacije, ciklička mutacija je odabrana kao hiperparametar. Postoji mogućnost kao i kod Ordered crossovera da je ciklička mutacija pobijedila zbog pristranosti uzrokovanom time što se za vrijeme optimiranja drugih hiperparametara koristila ciklička mutacija.

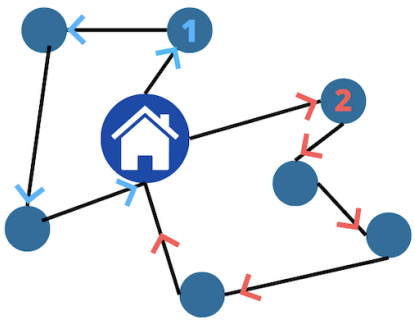
3.3. Analiza

3.3.1. Performanse genetskog algoritma

U prvoj fazi analize mjerit će se uspješnost genetskog algoritma s odabranim hiperparametrima naspram pohlepnog algoritma najbližeg susjeda (eng. *Nearest neighbor*) na svim instancama u skupu podataka prema metrikama prijeđenog puta, utrošenog vremena i metrike vremenskih intervala. Kao i prije genetski algoritam će se pokretati 10 puta po 10 minuta za svaku instancu, dok će se Nearest neighbor pokretati 10 puta za svaku instancu. Uprosječeni rezultati su prikazani na tablici 3.1. Na kutijastim dijagramima prikazani su odnosi performansi genetskog algoritma i najbližeg susjeda. Svaki kutijasti dijagram predstavlja drukčiju metriku prema kojoj se optimiralo te iznose svih metrika u odnosu na to.

Algoritam najbližeg susjeda

Implementirani algoritam najbližeg susjeda radi tako da se u početku za vozila uzme nasumični klijent te se za svakog sljedećeg klijenta uzima onaj najbliži. Vozila i dalje prate heuristiku nalaženja postajanja za punjenje kao i u genetskog algoritmu, nakon što više nisu u mogućnosti poslužiti klijenta vraćaju se u skladište te se za novo vozilo uzima nasumični klijent. Vizualizacija algoritma najbližeg susjeda prikazana je na slici 3.5.

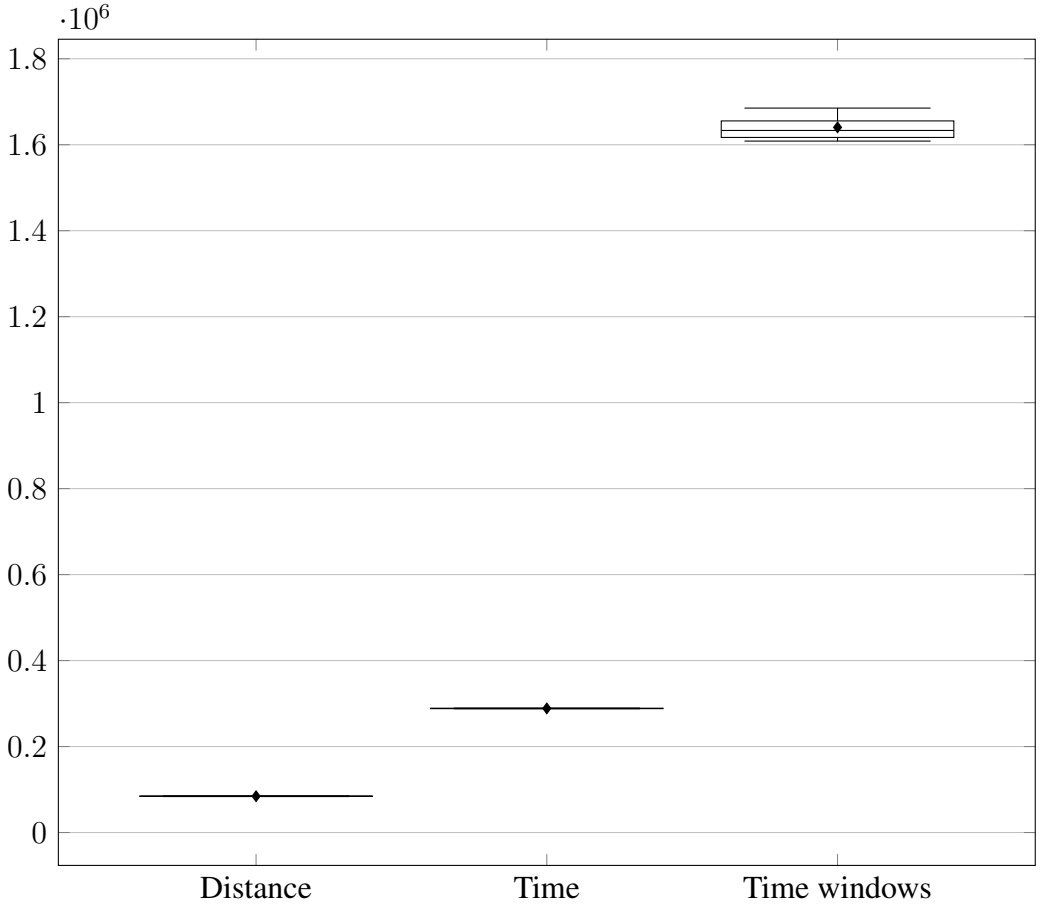


Slika 3.5: Algoritam najbližeg susjeda

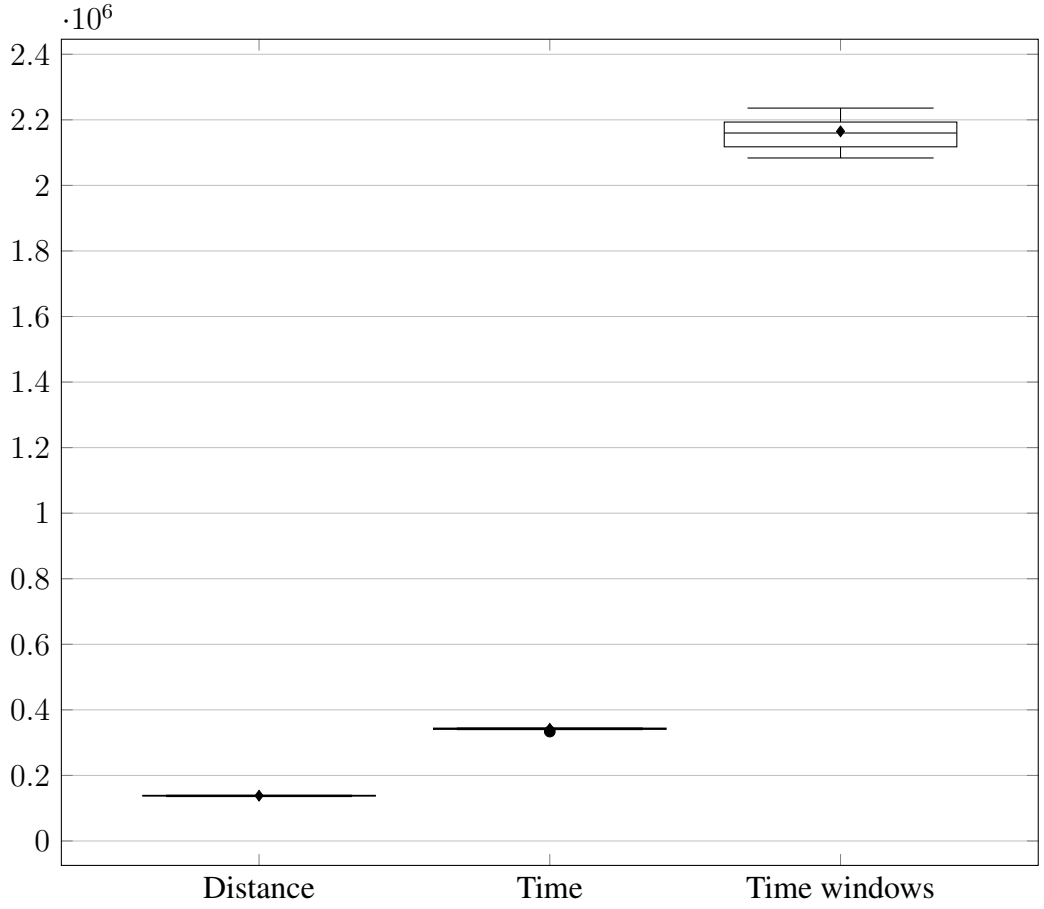
	GA_distance	GA_time	GA_time_windows	Nearest neighbor
Distance	138168	84537	116140	71490
Time	342158	288737	319990	267690
Time windows	2164860	1640435	1936005	1400048

Tablica 3.1: Performanse genetskog algoritma naspram algoritma najbližeg susjeda

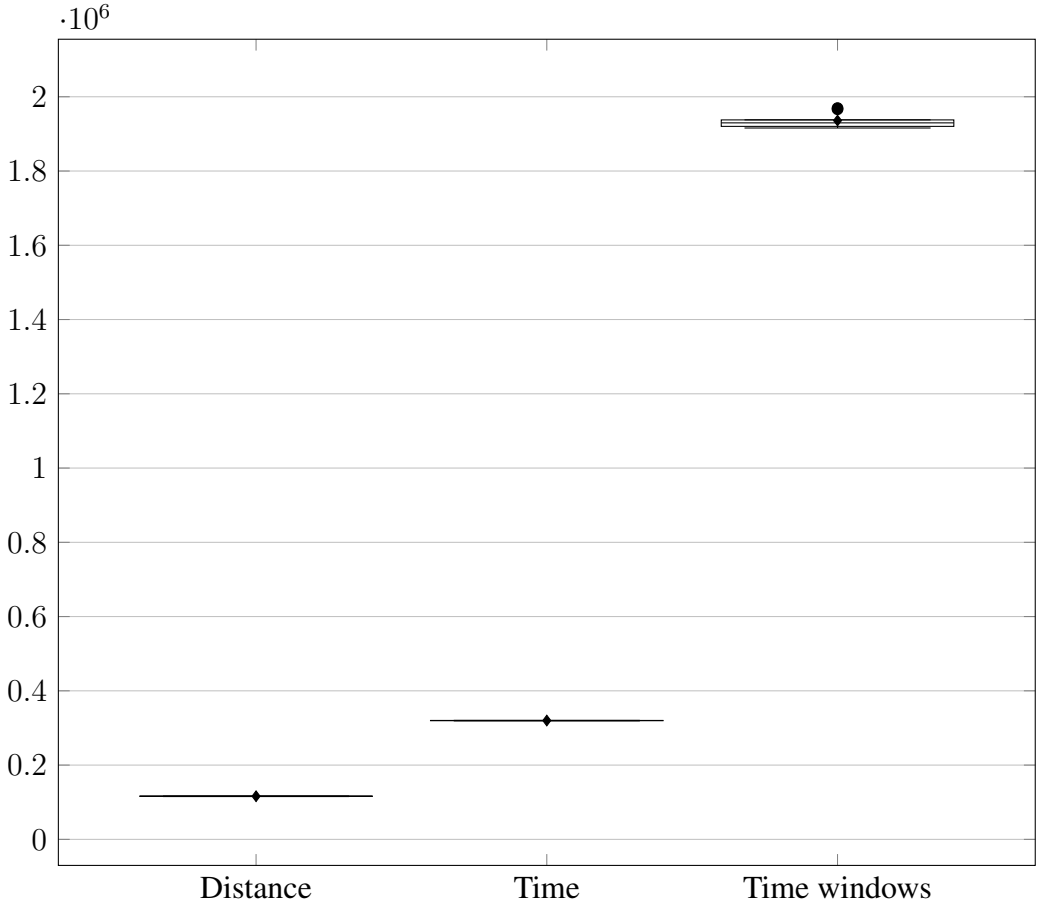
Redci tablice prikazuju metriku po kojoj su se ocjenjivala rješenja dok stupci predstavljaju algoritam te kriterij po kojem se optimiralo. Rezultati ukazuju na problem prerane konvergencije, prvi pokazatelj je superiornost pohlepnog algoritma najbližeg susjeda. Rezultati ukazuju da je optimiranje po vremenu najefikasnije za sve evaluacije dok suprotno vrijedi za optimiranje po prijeđenom putu. Također, optimiranje po vremenu se rezultatima najviše približilo Nearest neighbor algoritmu. Problem konvergencije se potencijalno može riješiti detaljnim predanalizom i odabirom hiperparametara posebice veličine populacije, stope mutacije i elitizma te uvođenjem tehnika kao što su DGCA (eng. *Dynamic Genetic Clustering Algorithm* Malik i Wadhwa (2014)) . Rezultati evaluacija po kriterijima su prikazani na slikama 3.6, 3.7 i 3.8.



Slika 3.6: Evaluacija kod optimiranja po utrošenom vremenu



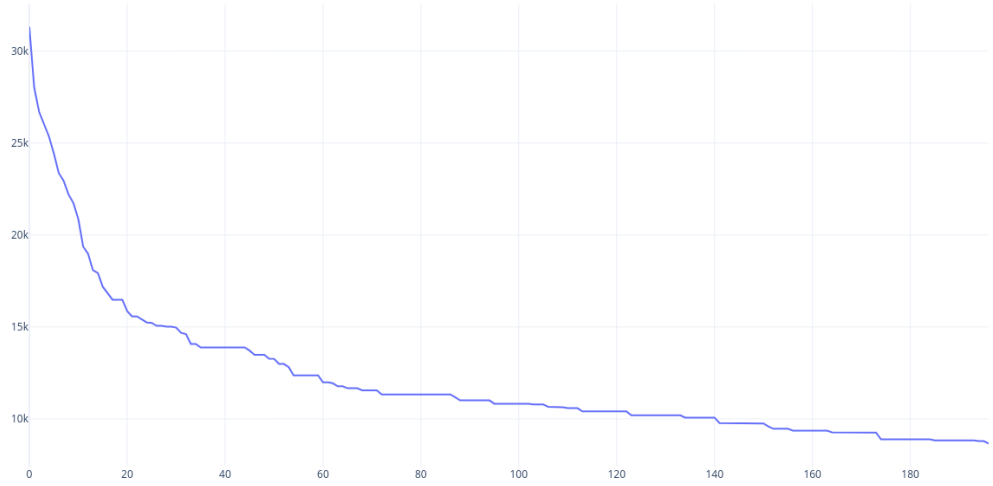
Slika 3.7: Evaluacija kod optimiranja po prijeđenom putu



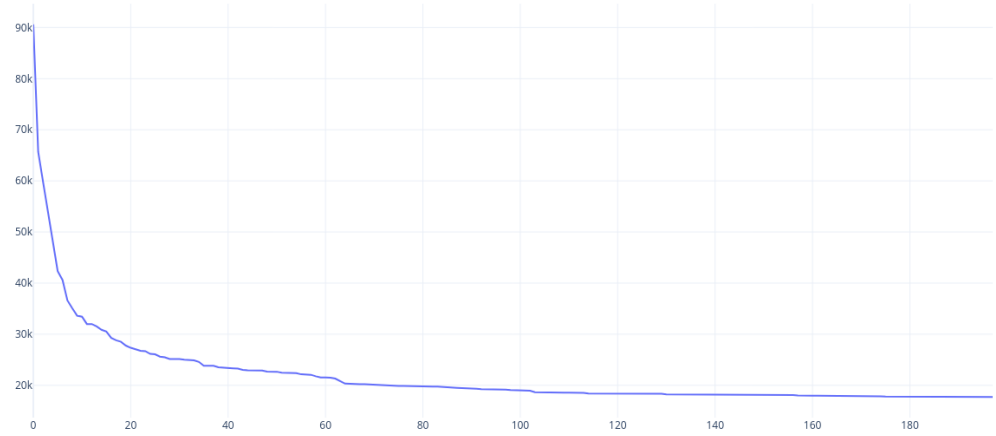
Slika 3.8: Evaluacija kod optimiranja po vremenskim okvirima

3.3.2. Rješavanje EVRP_CTW problema

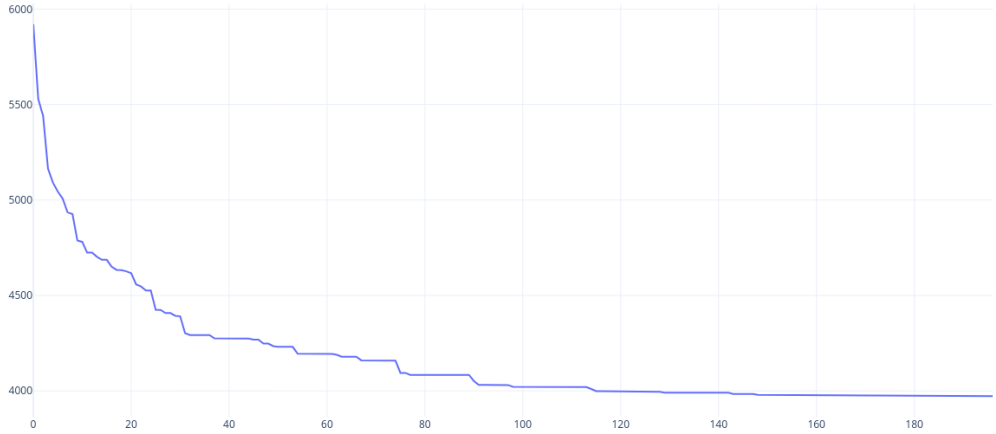
U drugoj fazi analize, genetski algoritam se pokreće jednom na svakoj od 6 vrsti instanci [r,rc,c] x [1,2] kako bi se prikazalo smanjivanje funkcije kazne kroz vrijeme. Uvođenjem time windows metrike genetski algoritam rješava puni problem koji uključuje vremenske intervale posluživanja te električna vozila s ograničenim kapacitetom. Pad funkcije kazne se pratio kroz 200 iteracija. Rezultati su prikazani na grafovima 3.8, - 3.12 Na x osi su označene iteracije dok su na y osi označene vrijednosti funkcije kazne koja uključuje vremenske intervale.



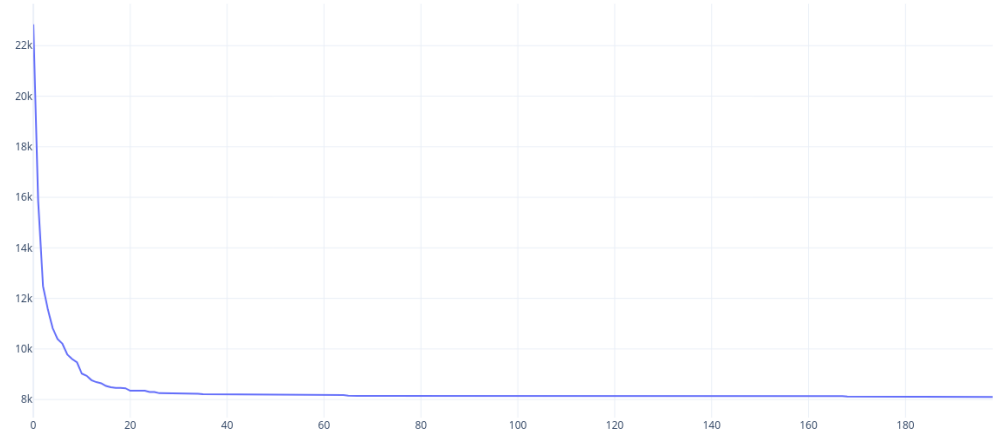
Slika 3.9: Instanca c101_21



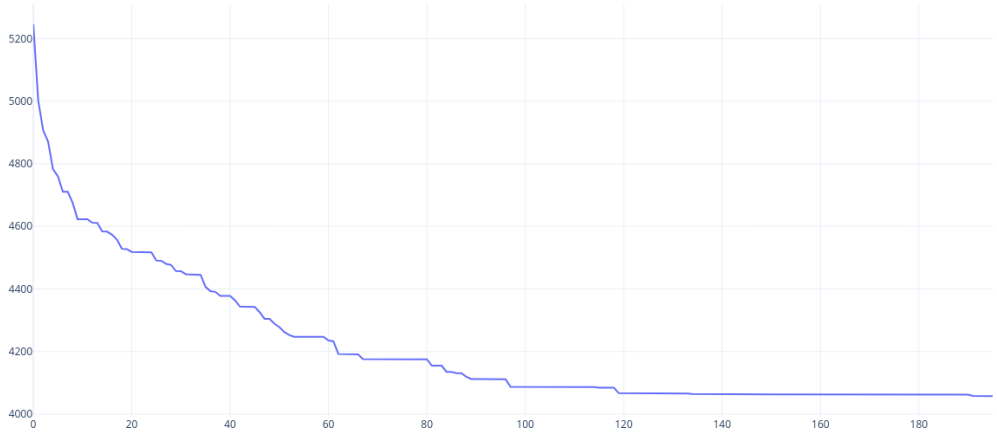
Slika 3.10: Instanca c201_21



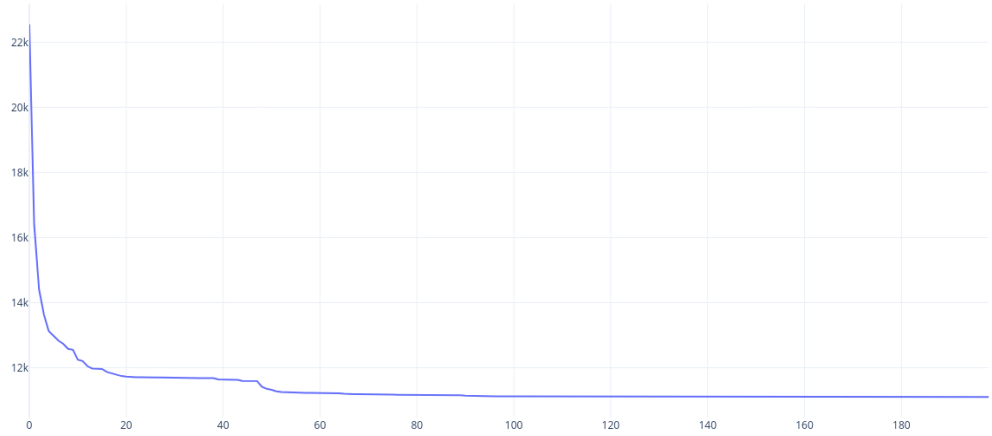
Slika 3.11: Instanca r101_21



Slika 3.12: Instanca r201_21



Slika 3.13: Instanca rc101_21



Slika 3.14: Instanca rc201_21

Prema grafovima se vidi da genetski algoritam brže spušta funkciju kazne za instance tipa 2 zbog širih intervala u kojem klijent želi biti poslužen. Razlog većih iznosa funkcije kazne kod instanci s relaksiranim ograničenjima dolazi zato što vozila imaju veći kapacitet za resurse te samim time poslužuju više korisnika što uzrokuje veća kašnjenja zbog pohlepnog načina odabira klijenata.

4. Zaključak

Rješavanje problema trgovačkog putnika pokazao se bitnim aspektom povećanja profita uslužnih djelatnosti kao što su dostave. U radu se razmatralo rješavanje specifičnijeg problema usmjeravanja flote vozila genetskim algoritmom.

U prvoj fazi tražile su se što bolje vrijednosti hiperparametara: veličine populacije, stope mutacije, mutacije te križanja nad dijelom instanci iz zadanog skupa podataka. U drugoj fazi se genetski algoritam pokretao za dane hiperparametre nad cijelim skupom podataka te su se uspoređivale performanse s obzirom na funkciju prema kojoj se optimiralo. Usporedba se radila metrika utrošenog vremena, prijeđenog puta te utrošenog vremena s dodatnim kaznama kod posluživanja klijenata izvan zadanog vremenskog intervala. Također su se usporedile performanse naspram pohlepnog algoritma najbližeg susjeda. Neočekivano loši rezultati genetskog algoritma ukazuju na problem prerane konvergencije što može biti posljedica loše odabranih hiperparametara.

U zadnjem dijelu rada prikazan je tijek genetskog algoritma kroz pad funkcije kazne u 200 iteracija za razne vrste instanci. Pokazalo se da se događa veći pad funkcije kazne kod instanci s relaksiranim ograničenjima, ali isto tako i veću početnu i krajnju vrijednost kazne s obzirom nego kod instanci sa striktnijim ograničenjima. Uzrok tome je kombinacija većeg kapaciteta vozila te pohlepna strategija dodjeljivanja klijenata vozilima.

Postojeći genetski algoritam se u budućnost može poboljšati detaljnijom analizom i boljim odabirom hiperparametara, drukčijom strategijom dodjeljivanja klijenata vozilima (npr. omogućiti vozilu da se vrati u skladište prije nego što više ne bude u mogućnosti poslužiti iti jednog klijenta, posljedica toga je veći prostor mogućih rješenja, što u općenitom slučaju može dati bolja rješenja pogotovo u situacijama gdje se optimira utrošeno vrijeme). Dodatno se mogu uključiti nove vrste mutacije i križanja te uvođenje dodatnih heuristika kao što su parcijalno punjenje gdje vozilo ne mora puniti bateriju do kraja, što može biti korisno kod rješavanja primjera iz stvarnog svijeta gdje je svojstvo baterija da se sporije pune s porastom nakupljenog naboja.

Alternativno rješavanje problema usmjeravanja vozila se može raditi i drugim algo-

ritmima kao što su Tabu search, VNS (Variable neighborhood search), 2-opt algoritam itd.

LITERATURA

Dominik Goeke. E-vrptw instances", mendeley data, v1,. 2019. doi: 10.17632/h3mrm5dhxw.1.

Stanley Gotshall i Bart Rylander. Optimal population size and the genetic algorithm. doi: 10.1.1.105.2431.

Ilker Kucukoglu, Reginald Dewil, i Dirk Cattrysse. The electric vehicle routing problem and its variations: A literature review. *Computers & Industrial Engineering*, 161:107650, 2021. ISSN 0360-8352. doi: https://doi.org/10.1016/j.cie.2021.107650. URL `https://www.sciencedirect.com/science/article/pii/S0360835221005544`.

Ms. Shikha Malik i Mr. Sumit Wadhwa. Preventing premature convergence in genetic algorithm using dgca and elitist technique. 2014.

Dr. Anantkumar Umbarkar i P. Sheth. Crossover operators in genetic algorithms: A review. *ICTACT Journal on Soft Computing (Volume: 6 , Issue: 1)*, 6, 10 2015. doi: 10.21917/ijsc.2015.0150.

Vijay Kumar Verma i Biresh Kumar. Genetic algorithm: an overview and its application. *International Journal of Advanced Studies in Computers, Science and Engineering*, 3(2):21, 2014.

Vickie Dawn Wester. "a genetic algorithm for the vehicle routing problem. " master's thesis, university of tennessee. 1993.

Rješavanje problema usmjeravanja električnih vozila korištenjem genetskih algoritama

Sažetak

U ovom radu predstavljen je problem usmjeravanja električnih vozila čije je rješavanje bitno za maksimiziranje profita uslužnih djelatnost te genetski algoritam kojime će se taj problem rješavati. Objašnjene su komponente i sam tok općenitog genetskog algoritma te prilagođene implementacije. U predanalizi su optimirani hiperparametri veličine populacije , stope mutacije, vrste križanja i vrste mutacije. S dobivenim vrijednostima uspoređene su performanse genetskog algoritma naspram običnog algoritma najbližeg susjeda po više kriterija. Na kraju se radi analiza i prikaz pada funkcije kazne kroz iteracije algoritma za razne tipove instanci.

Ključne riječi: Genetski algoritam, problem usmjeravanja električnih vozila, vremenski okviri

Solving electric vehicle routing problem using genetic algorithms

Abstract

In this thesis Electric vehicle problem, a problem whose solution is important for maximizing profit of service activities is introduced along with Genetic algorithm which is used for solving it. Components and flow of a general Genetic algorithm and its implementation are explained. In the preanalisys phase population size, mutation chance, crossover and mutation hyperparameters are optimized. Resulting values are then used to compare the Genetic algorithm's performance to one of basic Nearest neighbor algorithm by multiple criteria. Finally, analisys of cost function's decline through iterations is performed on different types of instances.

Keywords: Gentic algorithm, electric vehicle routing problem, time windows