SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 5350

Usmjeravanje vozila evolucijskim algoritmom

Vjeko Kužina

Zagreb, lipanj 2018.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA ZAVRŠNI RAD MODULA

Zagreb, 9. ožujka 2018.

ZAVRŠNI ZADATAK br. 5350

Pristupnik: **Vjeko Kužina (0036491223)**
Studij: Računarstvo
Modul: Računarska znanost

Zadatak: **Usmjeravanje vozila evolucijskim algoritmom**

Opis zadatka:

Detaljno opisati problem usmjeravanja vozila. Odabrati i definirati podvrste problema koji će se rješavati u okviru ovog rada. Navesti preporučene algoritme za rješavanje problema usmjeravanja vozila iz literature. Opisati evolucijske algoritme te odabrati i programski ostvariti inačicu algoritma koja je prikladna za rješavanje problema usmjeravanja vozila. Usporediti dobivene rezultate s rezultatima iz literature i slobodno dostupnih baza problema usmjeravanja vozila na Internetu. Uz rad priložiti izvorne tekstove programa i citirati korištenu literaturu.

Zadatak uručen pristupniku: 16. ožujka 2018.
Rok za predaju rada: 15. lipnja 2018.

Mentor:

Prof. dr. sc. Marin Golub

Predsjednik odbora za završni rad modula:

Prof. dr. sc. Siniša Srbljić

Djelovođa:

Doc. dr. sc. Tomislav Hrkać

Zahvaljujem mentoru, prof. dr. sc. Marinu Golubu, na pruženim savjetima tijekom izrade ovoga rada.

SADRŽAJ

1. Uvod

Mnogi optimizacijski problemi mogu se riješiti determinističkim algoritmima koji su vremenski vrlo zahtjevni, što je često neprihvatljivo. U tom se slučaju optimizacijski problemi mogu pokušati riješiti nedeterminističkim algoritmima kao što su evolucijski algoritmi.

Evolucijski algoritmi uzimaju uzor u teoriji evolucije Charlesa Darwina, pritom se oslanjajući na prirodnu selekciju. Stvaraju jedinke, koje predstavljaju moguća rješenja problema, od kojih iz generacije u generaciju bolje prilagođene jedinke imaju veću vjerojatnost preživjeti i generirati potomstvo. Za svakog potomka postoji također vjerojatnost da će dio njega mutirati te tako steći drugačije osobine. Tako se čuvaju dobra rješenja i u isto vrijeme istražuju drugačija rješenja.

Evolucijski algoritam uz pažljivo odabrane parametre i načine izvođenja većinom uspijeva doći do vrlo dobrih i prihvatljivih, a ponekad i do najboljih mogućih rješenja u relativno kratkom vremenu.

2. Optimizacijski problem

2.1. Motivacija

Razne tvrtke koje kao dio svojih usluga nude dostavu svojih dobara do potrošača se susretnu s problemom kako to učiniti, a da potroše što manje novaca. Možda to naizgled izgleda kao mala svota, no pokazano je da se korištenjem optimizacijskih algoritama može uštedjeti oko 5% troškova na prijevoz, a kako na prijevoz proizvoda otpada prosječno 10% njegove ukupne cijene, to sveukupno čini značajnu uštedu[6].

2.2. Opis problema

Problem usmjeravanja vozila (eng. *Vehicle routing problem*, VRP) je NP složen kombinatorni problem koji generalizira Problem trgovačkog putnika (eng. *Traveling salesman problem*, TSP) u kojemu jedan putnik ima zadatak posjetiti sve dane gradove točno jednom i vratiti se u početni grad.

VRP definiran je kao skup vozila *V* = *{*$v_1, v_2, ..., v_n$*}* gdje je *t* broj vozila i graf G = (N, E). Graf G se sastoji od lokacija *N* = *{*$n_0, n_1, ..., v_n$*}* gdje je n_0 skladište, a ostalih *n* lokacija su potrošači kojima dobra trebaju biti dostavljena i od bridova E = e_{ij} gdje $0 \leq i, j \leq$ n, e_{ij} = *(*n_i, n_j*)*.

Cilj osnovnog problema usmjeravanja vozila je sa zadanim brojem vozila V dostaviti najkraćim putem sva potrebna dobra na dane lokacije. Također svako od vozila mora započeti i završiti svoju putanju u skladištu te niti jedna lokacija ne smije biti posjećena više puta.

2.3. Proširenja problema usmjeravanja vozila

Proširenja osnovnog VRP-a proizlaze iz potreba i ograničenja koje se javljaju u stvarnome svijetu.

2.3.1. Problem usmjeravanja vozila s ograničenim kapacitetom

Tako je najpoznatije proširenje, kojime se ovaj rad i bavi Kapacitetni problem usmjeravanja vozila (*Capacitated vehicle routing problem(CVRP)*) koji na osnovni problem nadodaje ograničenje svakog vozila na količinu koju to vozilo može prenijeti. Pri postavljanju problema se većinom zadaju vozila s jednakim kapacitivnim ograničenjem, no moguće je napraviti algoritam i za različite kapacitete ako se za takvu situaciju pokaže potreba.

2.3.2. Problem usmjeravanja vozila s vremenskim ograničenjima

Proširenje Problem usmjeravanja vozila s vremenskim ograničenjima (*Vehicle routing problem with time windows (VRPTW)*) nadodaje ograničenje vremena dostave koje se može izraziti na 4 različita načina.

1) *[t_1, ∞)* gdje je omogućeno preuzimanje stvari tek nakon trenutka t_1 te se u slučaju ranijeg dolaska mora čekati trenutak t_1.

2) *[0, t_2]* gdje je omogućeno preuzimanje stvari do trenutka t_2.

3) *[t_1, t_2]* gdje je omogućeno preuzimanje u vremenskom rasponu od t_1 do t_2.

4) Kombinacija prijašnja 3 ograničenja.

2.3.3. Problem usmjeravanja vozila s prikupljanjem i dostavom

Proširenje Problem usmjeravanja vozila s prikupljanjem i dostavom (*vehicle routing problem with pick-up and deliveries (VRPPD)*) nadodaje mogućnost uzimanja dodatnih stvari kod potrošača te odnošenje istih do drugih potrošača ili samog skladišta.

2.3.4. Ostala proširenja

Osim prije navedenih mogu se dodavati još velik broj različitih proširenja kao na primjer:

1)*Multiple depot VRP (MDVRP)* s više skladišta.

2)*Split delivery VRP (SDVRP)* više vozila može poslužiti istog potrošača.

3)*In a dial-a-ride problem (DARP)* prijevoz ljudi uz prijevoz stvari.

2.4. Pristupi rješavanju

Kako je VRP NP-složen problem on sadrži velik broj mogućnosti i za veće brojeve lokacija ga je veoma teško riješiti pa su veliki napori uloženi u rješavanje toga problema metaheuristikama. Tako su neke od njih genetski algoritam, tabu pretraga i simulirano kaljenje od kojih će ovaj rad koristiti genetski algoritam kao način rješavanja.

3. Genetski algoritam

Genetski algoritmi primjenjuju procese iz prirode i to na konkretnim problemima. Prvo je potrebno odrediti prikaz jedinki (rješenja) problema kojeg želimo riješiti.

Stvara se populaciju apstraktnih jedinki (kromosoma). Svaka pojedina jedinka predstavlja neko od rješenja problema, a sve jedinke imaju istu strukturu.

Svakoj od jedinki se pridružuje ocjena koja pokazuje koliko je dobro napravila posao. Ta ocjena se naziva funkcija dobrote ili cilja i služi predočavanju kvalitete rješenja.

Nakon toga se obavlja selekcija nad jedinkama te se primjenjuju genetski operatori u svrhu stvaranja novih jedinki za iduću generaciju. Postoje dvije vrste genetskih operatora, prva su unarni koji na individualnoj bazi mijenjaju male dijelove, a drugi su operatori višeg reda, koji iz više jedinki (roditelja) tvore nove jedinke (djecu).

Proces ocjenjivanja i kreiranja novih generacija se ponavlja do nekog uvjeta prekidanja.

Do kraja algoritma dolazi se pri ispunjenju predodređenog uvjeta zaustavljanja.

```
EvolucijskiAlgoritam {
    t = 0;
    generirajpopulaciju P(0);
    do uvjeta zaustavljanja {
        t++;
        ocijeni P(t - 1);
        selektiraj P'(t) iz P(t - 1);
        primijeni operator višeg reda P'(t) i spremi rezultate u P''(t);
        primijeni unarni operator P''(t) i spremi rezultate u P(t);
    }
    Ispis rješenja;
}
```

Slika 3.1 – Pseudokod evolucijskog algoritma[1].

3.1. Prikaz rješenja

Izbor prikaza rješenja znatno utječe na učinkovitost algoritma utoliko što pristupi podatcima i operacije koje se izvršavaju na njima mogu biti olakšane organizacijom i prikazom rješenja.

Prikaz rješenja uvelike ovisi o problemu koji je u pitanju te će se za svaku novu vrstu problema morati odrediti najbolja reprezentacija. Tako to na primjer može biti binarni kod, lista cijelih brojeva, mapa brojeva s pomičnom točkom ili pak kombinacija raznih struktura podataka.

U ovome problemu jedinka je prikazana kao lista vozila u kojoj svako vozilo sadrži svoju listu indeksa lokacija koje treba posjetiti.

3.2. Inicijalizacija

Pri inicijalizaciji nudi se više opcija od kojih su neke :

1) Generiranje pseudo slučajnih jedinki (rješenja) iz domene.

2) Uniformno generiranje jedinki (sve su iste na početku). - Neučinkovito jer se neće moći dobro istražiti veliko područje mogućih rješenja te će rješenja u populaciji pre brzo konvergirati.
3) Generiranje jedinki prema rješenju nekog drugog algoritma.

Većinom je cilj držati veličinu populacije konstantnom kroz iteracije.

U ovome problemu se inicijalizacija svodi na određivanje valjanih putanja za vozila pri tome stavljajući pažnju na maksimalni dozvoljeni kapacitet vozila. Kako bi se to učinilo na učinkovit način potrebno je osmisliti heuristiku kojom će se stvarati jedinke, no koja će također imati izraženu nasumičnosti kako se ipak ne bi stvarala jednaka rješenja.

3.3. Funkcija dobrote

Najbitniji dio stvaranja evolucijskog algoritma često je određivanje funkcije dobrote. Ona mora vjerno evaluirati koliko je pojedino rješenje dobro te mu, sukladno tome, pridijeliti dobrotu. Cilj funkcije dobrote je postići što bolji pokazatelj uspješnosti, kako bi ukupna i prosječna dobrota populacije kroz selekciju i genetske operatore mogla rasti.

Ponekad je teško dobro odrediti funkciju dobrote jer se u pojedinim problemima apstraktnim vrijednostima mora pridijeliti brojka koja će ih opisati.

Ovdje je funkcija dobrote jednostavna, predočena je kao suma udaljenosti koje sve jedinke moraju proći u svojim putanjama.

3.4. Selekcija

Cilj selekcije je većinom odabirati bolje jedinke za stvaranje potomstva kako bi rješenja konvergirala prem optimumima, a ponekad i odabrati lošija rješenja koja potencijalno mogu imati neku dobru osobinu kako bi se dobilo na raznolikosti.

Postoji više vrsta selekcije koje to pokušavaju postići na različite načine, a neke od njih su:

1) Jednostavna selekcija
U jednostavnoj selekciji roditelji se biraju proporcionalno svojoj dobroti (što je veća dobrota, to je veća vjerojatnost za odabir). Pri odabiru ovog načina selekcije treba biti pažljiv jer ponekad se zna pojaviti problem skale u kojem se može dogoditi da su sve dobrote jako velike pa im razlike neće značiti mnogo pri odabiru ili dobrote mogu biti jako male pa će i male razlike u dobroti uzrokovati jako velike promjene u vjerojatnosti odabira.

2) Turnirska selekcija
Turnirska selekcija uzima nasumično nekoliko jedinki iz trenutne populacije te najbolju od njih odabire (eliminacijska turnirska selekcija najgoru briše).

3) Eliminacijska selekcija
Eliminacijska selekcija radi upravo suprotno od jednostavne selekcije utoliko što bira jedinke obrnuto proporcionalno dobroti te odabrane briše iz populacije.

4) Elitizam
Elitizam je većinom samo dodatak jednom od prije navedenih vrsta selekcije time što, jedno ili više rješenja s najvećom dobrotom uvijek sačuva u iduću generaciju.

U ovome problemu su na izbor stavljene 2 vrste selekcije, od kojih je prva elitistička turnirska, a druga eliminacijska turnirska.

3.5. Genetski operatori

3.5.1. Operatori višeg reda

Najčešći operator višeg reda je križanje. Ideja križanja je izrada nove generacije jedinki(djece) od 2 ili više selektiranih jedinki iz trenutne generacije(roditelja). Pri tome se čuvaju osobine koje su jednake kod roditelja, a za osobine koje su različite, predodređenim se algoritmom određuje čiju će osobinu uzeti dijete.

Križanje pokušava konvergirati populaciju prema lokalnim optimumima jer ono čuva slične dijelove rješenja te izmjenjuje samo različite dijelove čime će jedinke unutar populacije sve više i više sličiti jedne na druge. Ako se tome ne postavi nekakva protumjera koja pospješuje divergenciju u populaciji, populacija će pre brzo konvergirati

u nekom lošem rješenju.

U ovome problemu programski su ostvarene 3 vrste križanja koje će kasnije biti detaljnije objašnjene:

1) Čvorovi:
Pokušava objediniti putanje s najvećim preklapanjem čvorova.

2) Lukovi:
Pokušava objediniti putanje s najvećim preklapanjem lukova.

3) Sekvence:
Pokušava objediniti putanje s najduljim preklapanjem sekvenci.

3.5.2. Unarni operatori

Najčešći unarni operator je mutacija. Mutacija uvodi mogućnost da se jedna ili više osobina jedinke promijeni. Ovisno o tome koliku vjerojatnost mutacije postavimo, događaju se velike razlike u tome kako algoritam radi i što pokušava napraviti.

Jedna od protumjera križanja i konvergencije koja s time dolazi je divergencija koja se pruža mutacijom. Mutacija nam omogućava da pobjegnemo iz lokalnih optimuma u potrazi za boljim rješenjima tako što se nasumični dio jedinke promijeni.

Kako vjerojatnost mutacije teži prema jedinici, naš algoritam postaje ustvari algoritam slučajnog isprobavanja rješenja, a ne konstantno poboljšanje. Nasuprot tome, ako teži prema 0, onda algoritam postaje traženje lokalnog optimuma, a ne više potraga za globalnim optimumom.

Programski su ostvarene 3 vrste mutacija koji izbacuju čvorove, a kasnije ih nanovo stavljaju u rješenje umetajućom heuristikom:

1) Nasumična
Nasumično odabrani čvorovi se izbace iz jedinke.

2) Blizinska

Nasumično se odabere čvor te se izbace on i njemu najbliži čvorovi.

3) Zaobilazna

Izbacuje čvorove za koje je potrebno napraviti najveći zaobilazak.

4. Programsko ostvarenje algoritma

Algoritam izrađen u sklopu ovog završnog rada izrađen je u programskom jeziku Java.

Algoritmu je potrebno preko naredbenog retka predati šest ili sedam argumenata, od kojih ako se želi eliminacijska turnirska selekcija se predaju samo 6, a ako se želi obična turnirska selekcija je potrebno dodati još 1 argument. Argumenti su redom putanja do datoteke s problemom, veličina populacije, broj iteracija, broj iteracija bez poboljšanja za prekid, veličinu mutirane populacije, broj iteracija za mutiranu populaciju i vjerojatnost mutacije.

Razlog dodatnog argumenta je taj što je u eliminacijskom turnirskom algoritmu ostvareno dinamičko izračunavanje vjerojatnosti mutacije, a u selekcijskoj turnirskoj selekcije se koristi zadana vrijednost.

4.1. Dohvat podataka

Datoteka koju je potrebno predati programu mora biti specifičnog formata. Ta datoteka mora sadržavati međusobne udaljenosti svih lokacija te zahtjev dobara svih lokacija. Također se mora zadati broj vozila te kapacitetna ograničenja vozila.

Slike 4.1 i 4.2 predstavljaju primjer jedne takve datoteke, a sama datoteka dostupna je u literaturi[7]. Primjeri datoteka za različite vrste problema usmjeravanja vozila također su dana u literaturi[5].

```
NAME : A-n32-k5
COMMENT : (Augerat et al, No of trucks: 5, Optimal value: 784)
TYPE : CVRP
DIMENSION : 32
EDGE_WEIGHT_TYPE : EUC_2D
CAPACITY : 100
NODE_COORD_SECTION
 1 82 76
 2 96 44
 3 50 5
 4 49 8
 5 13 7
 6 29 89
 7 58 30
 8 84 39
 9 14 24
 10 2 39
 11 3 82
 12 5 10
 13 98 52
 14 84 25
 15 61 59
 16 1 65
 17 88 51
 18 91 2
 19 19 32
 20 93 3
 21 50 93
 22 98 14
 23 5 42
 24 42 9
 25 61 62
 26 9 97
 27 80 55
 28 57 69
 29 23 15
 30 20 70
 31 85 60
 32 98 5
```

Slika 4.1: Datoteka s podatcima(prvi dio)[7]

```
DEMAND_SECTION
1 0
2 19
3 21
4 6
5 19
6 7
7 12
8 16
9 6
10 16
11 8
12 14
13 21
14 16
15 3
16 22
17 18
18 19
19 1
20 24
21 8
22 12
23 4
24 8
25 24
26 24
27 2
28 20
29 15
30 2
31 14
32 9
DEPOT_SECTION
 1
 -1
EOF
```

Slika 4.2: Datoteka s podatcima(drugi dio)[7]

Slika zadaje vrstu problema, broj lokacija, način računanja udaljenosti, kapacitet vozila, pozicije lokacija te zahtijevanu količinu dobara od svake lokacije.

U sklopu rada programski je ostvaren i parser ovakvih datoteka.

4.2. Prikaz rješenja i funkcija dobrote

Prikaz rješenja(jedinke) napravljen je kao prikaz svih korištenih vozila za dostavu te putanja kojima se svako od tih vozila kreće. Tako konkretna jedinka sadrži listu vozila od kojih svako vozilo posjeduje listu cijelih brojeva koji predstavljaju čvorove/lokacije koje treba posjetiti.

Funkcija dobrote programski je ostvarena tako da pokazuje ukupnu udaljenost koja će sva vozila morati proći po svojim putanjama unutar jedinke.

Primjer rješenja može se vidjeti na slici 4.3.

```
Vozilo 1: 0 18 8 9 22 15 29 10 25 5 20 0
Vozilo 2: 0 28 11 4 23 3 2 6 0
Vozilo 3: 0 26 7 13 17 19 31 21 0
Vozilo 4: 0 14 24 27 0
Vozilo 5: 0 30 16 1 12 0
790
```

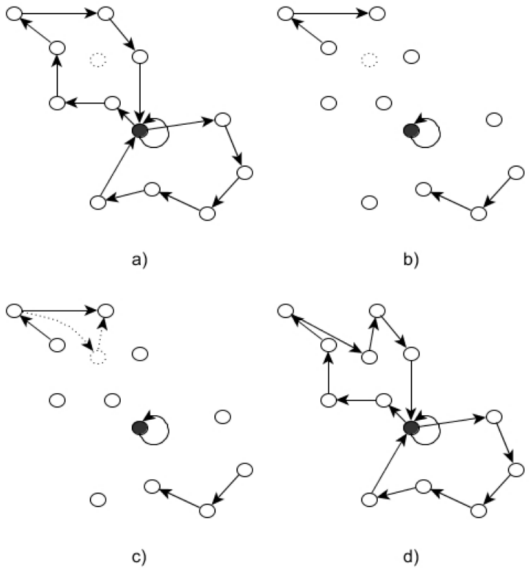
Slika 4.3: Prikaz rješenja

4.3. Umetajuća heuristika

Kada bi samo dodavali čvorove nasumično u pojedine putanje dobivali bi izuzetno loša rješenja koja nam vrlo vjerojatno neće pomoći pri pronalasku optimalnog rješenja, a češće će se i stvarati nemoguća rješenja. Pomoglo bi nam kada bi umetali čvorove tako da minimiziramo udaljenost koja će se prijeći dodavanjem čvora u putanje.

Tako je u okviru ovoga rada ostvarena umetajuća heuristika koja nasumično odabire čvorove iz liste ne umetnutih čvorova te ih pokušava dodati u postojeće putanje tako da minimizira udaljenost koja će se morati dodatno prijeći, pri tome pazeći da se ne prekorači maksimalni dozvoljeni kapacitet vozila koje je zaduženo za tu putanju. Ako ne postoji mjesto u postojećoj putanji na koje se može dodati, ovisno o tome postoji li još slobodnih vozila se dodaje novo vozilo sa svojom putanjom ili se prekida stvaranje jedinke i kreće u novi pokušaj.

Ova se heuristika koristi pri stvaranju jedinku te pri bilo kakvoj modifikaciji jedinki (mutacijama, križanjima).



Slika 4.4: Umetajuća heuristika: a) Trenutna rješenja; b) Bridovi u koje je moguće dodati čvor; c) Potraga za minimalnom cijenom umetanja; d) putanje s ubačenim čvorom [2]

4.4. Evolucija algoritma

Glavni algoritam ovoga programa izvodi evoluciju kroz generacije tako što u svakoj generaciji stvara novu populaciju iste veličine kao i prošla. Stvara ju tako da odabere roditelje, križa ih nasumično odabranim operatorom križanja te tako stvorenu djecu doda u novu populaciju.

S određenom vjerojatnošću će prvo dijete biti mutirano tako što će se nasumično odabranim operatorom mutacije izbaciti nekoliko čvorova iz putanja toga djeteta te će se od ostataka putanja i izbačenih čvorova generirati jedna cijela mala populacija mutiranih jedinki. Ta nova populacija će kroz manji broj iteracija prolaziti istim procesom evolucije kao i cjelokupna populacija.

Ovaj proces prikazan je ovim pseudokodom:

Pop_1 - generiraj inicijalnu populaciju veličine PS_1 umetajućom heuristikom.
while broj iteracija bez napretka < IL_1 i broj iteracija < NOI_1
    sort(Pop_1)
    for i = 1..PL_1
        x_{p11}, x_{p12} – odredi roditelje iz Pop_1

$x_{c11} = crossover(x_{p11}, x_{p12})$ - križaj roditelje i generiraj potomka
$x_{c12} = crossover(x_{p12}, x_{p11})$ - križaj roditelje i generiraj potomka
dodaj x_{c11} i x_{c12} u Pop_2
if random(0,1) < MP - primjeni mutaciju s vjerojatnošću MP
(x'_{m1}, N'_{m1}) - procesom mutacije izbaci neke čvorove i generira novo lomično rješenje s ne izbačenim čvorovima
$MutPop_1$ – generiraj populaciju veličine PL_2 umetanjem N'_{m1} u x'_{m1}
call(MutPop) - pokreni proces $MutPop_1$
x_{m1} = odredi najbolje rješenje iz $MutPop_1$
dodaj x_{m1} u Pop_2
end if
end for
Pop_1 = Pop_2
clear(Pop_2)
end while
najbolje rješenje je Pop_1 [0]

PROCESS $MutPop$: **while** broj iteracija bez poboljšanja < IL_2 i broj iteracija < NOI_2
sort($MutPop_1$)
for j=1...PL_2
x_{p21}, x_{p22} - odredi roditelje iz $MutPop_1$
x_{c21} = crossover(x_{p21}, x_{p22})
x_{c22} = crossover(x_{p22}, x_{p21})
dodaj x_{c21} iz x_{c22} u $MutPop_2$
if random(0,1) < MP
x_{m2} - mutiraj x_{c21}
dodaj x_{m2}u $MutPop_2$
end if
end for
$MutPop_1$ = $MutPop_2$
clear($MutPop_2$)
end while

Parametri:
Pop_1 - trenutna populacija
PS_1 - veličina populacije
IL_1 - broj iteracija bez napretka nakon kojeg se zaustavlja algoritam
NOI_1 - Ukupan broj iteracija
x_{p1y} - Roditelj broj y
x_{c1y} - Dijete broj y
Pop_2 - nova populacija (za iduću generaciju)
MP - vjerojatnost mutacije
x'_{m1} - parcijalna jedinka
N'_{m1} - čvorovi izbačeni iz parcijalne jedinke
$MutPop_1$ - trenutna populacija sastavljena od mutiranih jedinki
PL_2 - veličina mutirane populacije

x_{m1} - najbolje rješenje iz mutirane populacije
IL_2 - broj iteracija bez napretka nakon kojeg se zaustavlja proces mutiranja
NOI_2 - ukupan broj iteracija mutirane populacije
x_{p2y} - Roditelj broj y u mutiranoj populacij
x_{c2y} - Dijete broj y u mutiranoj populacij
$MutPop_2$ - nova populacija mutiranih jedinki (za iduću generaciju)
x_{m2} - mutirana jedinka u procesu evolucije mutiranih jedinki

4.5. Selekcija

Odabrana su 2 vrste selekcije od kojih je prva elitistička turnirska selekcija koja nakon što prebaci 2 najbolje jedinke u iduću populaciju bira roditelja tako što nasumično odabere unaprijed određeni broj jedinki (*tournamentSize*) te onu s najvećom dobrotom postavi kao roditelja, a druga je 3-turnirska eliminacijska selekcija koja nasumično odabere 3 jedinke, od njih 2 bolje postavlja kao roditelje, a treću miče iz populacije.

Ovakvi operatori su odabrani jer je za ovaj problem važno sačuvati najbolja rješenja iz generacije u generaciju.

4.6. Križanje

U programskom ostvarenju operatora križanja ideja je bila sačuvati zajedničke dijelove roditelja. Prvo je potrebno odlučiti što će se uopće smatrati zajedničkim unutar 2 jedinke, nakon toga potrebno je identificirati te zajedničke dijelove, sačuvati ih te ostatak rekonstruirati u djetetu koristeći umetajuću heuristiku.

U nastavku je prikazan pseudokod koji gore upisano izvodi:

x_i - prvi roditelj
x_j - drugi roditelj
$x_0^{'}$ - generiraj dijete
for svaku putanju $r_i \in R_i$
    dodaj (**call** (**Pronađi_Zajednički_Dio** (x_i, x_j))) u $x_0^{'}$
end for
dodaj(**Neposjećeni**(x_i)) u $x_0^{'}$

Parametri:
x_y - roditelj broj y

$x_0^{'}$ - dijete
r_i - putanja
R_i - lista svih putanja

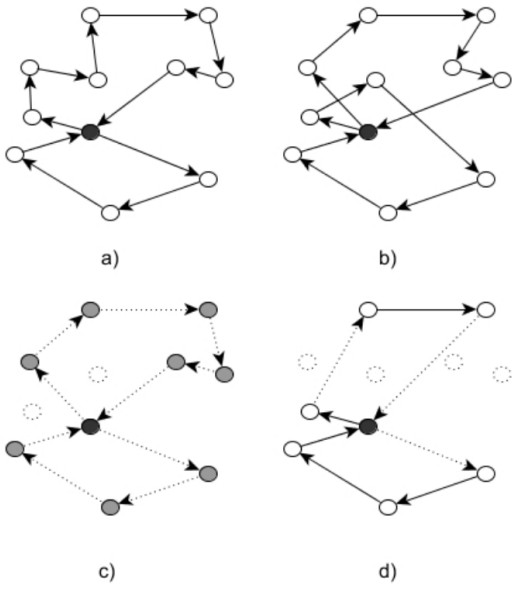
Programski su ostvarena 3 operatora križanja koji drugačije određuju što se smatra zajedničkim unutar dvije jedinke:

1) Križanje zajedničkih čvorova(eng. *Common nodes crossover*, CNX)

Ideja je pronaći putanje unutar roditelja koje imaju najveći broj zajedničkih čvorova, sačuvati tu putanju i predati ju djetetu te ostatak te putanje i ostale putanje popuniti umetajućom heuristikom.

2) Križanje zajedničkih lukova(eng. *Common arcs crossover*, CAX)

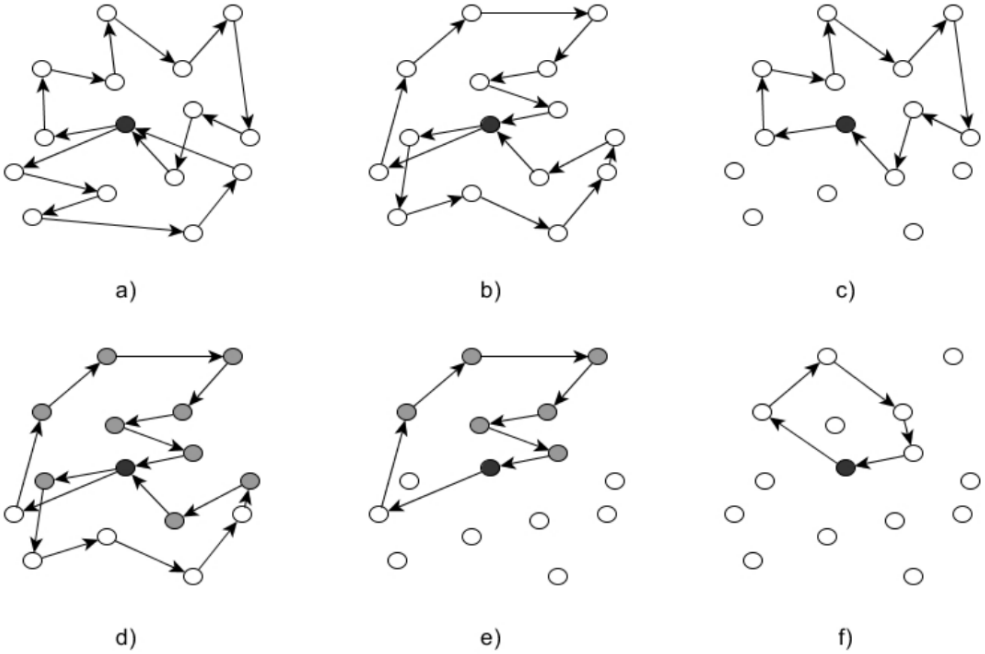
Lukovi prvog roditelja će se zadržati istim redoslijedom u djetetu ako isti takvi lukovi postoje i u drugome roditelju. Time se podrazumijeva da imaju isti početni i završni čvor te povezuju čvorove istim redoslijedom.



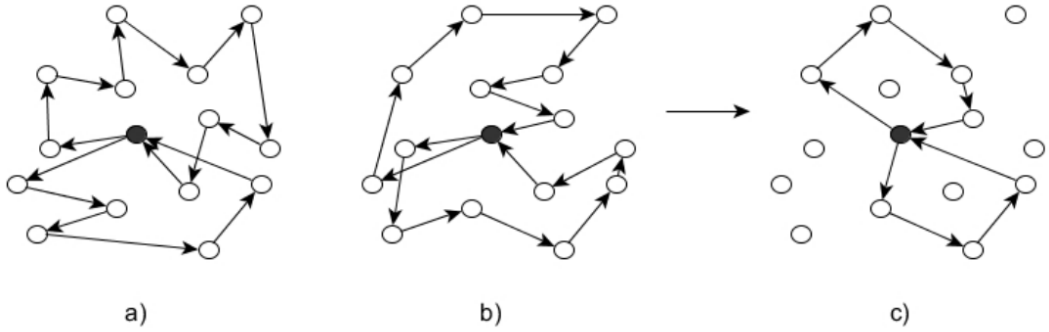
Slika 4.5: Križanja zajedničkih čvorova i lukova: a) Prvi roditelj; b) Drugi roditelj; c) Križanje zajedničkih čvorova; d) Križanje zajedničkih lukova [2]

3) Križanje zajedničkih najdužih sekvenci(eng. *Longest common sequence crossover*, LCSX)

Za svaku putanju prvog roditelja pronađe se putanja u drugom roditelju s najviše istih čvorova, te se ti zajednički čvorovi prenesu djetetu istim redoslijedom kojim su prisutni u prvom roditelju.



Slika 4.6: Pronalazak najdulje sekvence u svim putanjama: a) Prvi roditelj; b) Drugi roditelj; c) Prva određena putanja prvog roditelja za evaluaciju; d) Identificirane putanje s istim čvorovima kao u putanji u c); e) Određena putanja s najviše zajedničkih čvorova iz d); f) Najdulja zajednička sekvenca iz c) i e) je pronađena [2]



Slika 4.7: Križanje zajedničkih najdužih sekvenci: a) Prvi roditelj; b) Drugi roditelj; c) Pronađene najdulje sekvence [2]

4.7. Mutacija

U programskom ostvarenju mutacije princip je bio suprotan križanju, uz to što se izvodi na jednom rješenju, po nekom kriteriju izdvojiti čvorove koji će se preurediti, a ostatak rješenja ostaviti istim.

Vjerojatnost mutacije u algoritmu selekcije predaje se u argumentima te će zbog načina programskog ostvarenja algoritma biti podosta veća nego u tradicionalnim evolucijskim algoritmima. Naime inače dva odabrana roditelja križanjem stvaraju dijete koje još može i mutirati te se tek onda dodaje u populaciju, a u ovome algoritmu se stvara dvoje djece te se odmah oboje dodaju u populaciju. Nakon toga postoji vjerojatnost da prvo dijete mutira te u slučaju da se to dogodi se i mutirana jedinka dodaje u populaciju. Zbog toga je udio mutacije puno manji nego u tradicionalnim algoritmima pa je vjerojatnost mutacije veća kako bi to nadoknadila.

U algoritmu eliminacije vjerojatnost mutacije se dinamički mijenja ovisno o tome koliko je populacija konvergirala. Tako će vjerojatnost mutacije biti veća ako populacija više konvergira, a manja ako populacija više divergira[4].

Prema kriteriju izdvajanja čvorova programski su ostvarena 3 algoritma:

1) Mutacija nasumičnih čvorova

Nasumično se biraju čvorovi koji će biti izbačeni iz jedinke te naknadno dodani umetajućom heuristikom.

2) Mutacija bliskih čvorova

Nasumično se odabere jedan čvor koji se izbaci te se za daljnje izbacivanje odabiru čvorovi najbliži prvom izbačenom čvoru.

3) Mutacija najvećeg obilaska

Izabiru se čvorovi koji maksimiziraju funkciju $l_r(n_r) = l(n_{r-1}, n_r) + l(n_r, n_{r+1}) - l(n_{r-1}, n_{r+1})$ gdje l označava udaljenost. Time se izdvajaju čvorovi koji uzrokuju najveći zaobilazak u trenutnom rasporedu čvorova.

Količina izabranih čvorova ograničena je izrazom **0.5z|N|** gdje je **z** nasumično odabran broj iz intervala **[0,1]**.

4.8. Programsko ostvarenje

U ovome radu izrađen je program za cjelokupnu obradu i rješavanje problema usmjeravanja vozila s kapacitivnim ograničenjem. To uključuje programsko ostvarenje parsera za dokumente koji zadaju problem, umetajuće heuristike, cjelokupnog genetskog algoritma koji uključuje programsko ostvarenje 2 operatora selekcije, 3 operatora križanja i 3 operatora mutacije.

5. Eksperimentalni rezultati

Rezultatima statistike pokušavaju se pokazati utjecaji parametara algoritma na njegovo izvođenje te tako ustanoviti najbolje parametre za korištenje algoritma.

Kako za male probleme za od 30 do 70 lokacija algoritam s dobrim parametrima većinom pronađe najbolje rješenje relativno brzo, nije imalo smisla izvoditi algoritam na tim problemima, jer će se razlika bolje vidjeti na problemima u kojima teško pronalazi optimum.

Tako je problem na kojem su izvođeni eksperimenti, problem usmjeravanja vozila s kapacitivnim ograničenjem koji na raspolaganje daje 10 vozila i zahtijeva ispostavu dobara na 79 lokacija. Problem ima zatvorenost od 94% što znači da ukupna količina zahtijevanih dobara iznosi 94% sume kapaciteta svih vozila.

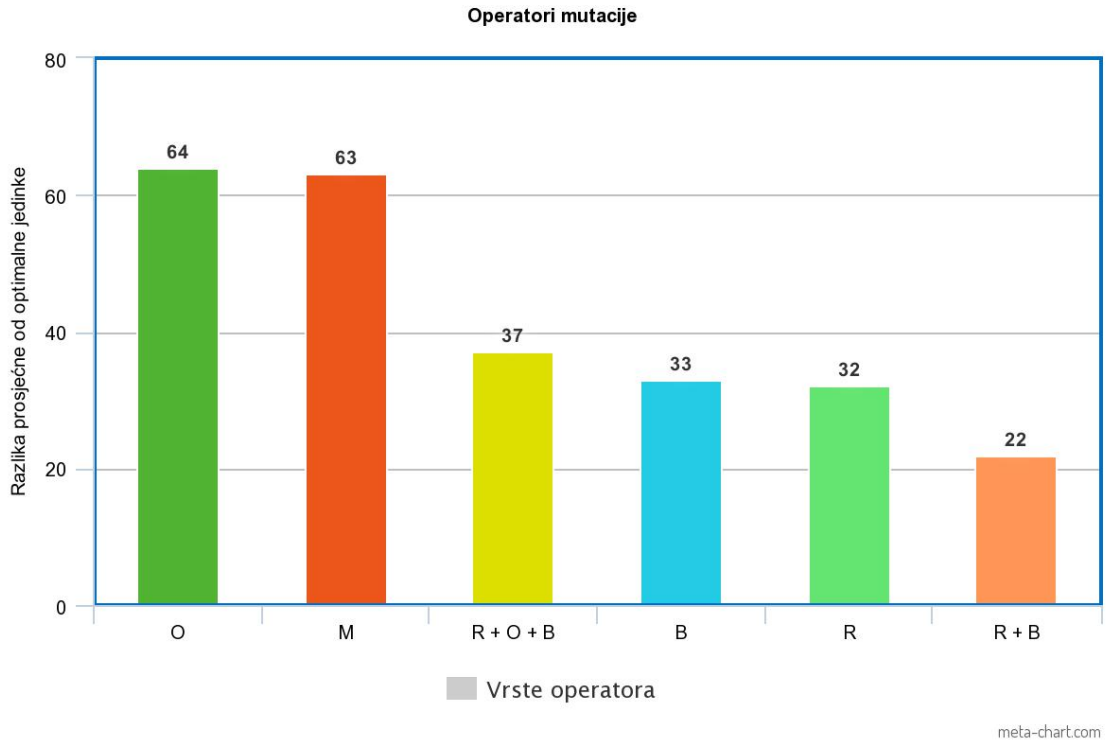
Dokument koji predstavlja ovaj problem dostupan je u dokumentaciji[9].

5.1. Operatori mutacije

U svim mjerenjima svi parametri osim korištenih operatora mutacije su držani fiksni. Tako su mjerenja izvođena na populaciji od 100 jedinki u 1000 iteracija glavnog algoritma s ustaljenjem od 200 iteracija, mutacijska populacija je veličine 10 jedinki i izvodi se kroz 10 iteracija, a vjerojatnost mutacije je 0.5. Kao operator selekcije korištena je 3-turnirska selekcija, a kao operatori križanja korišteni su CAX i LCSX.

Svaki od prikazanih podataka izračunat je iz 30 rezultata izvođenja programa pod is-

tim uvjetima. Podatci prikazuju razliku prosječnog rješenja algoritma u 30 iteracija od optimalnog rješenja.



Slika 5.1: O - Mutacija najvećeg obilaska; M - Mini mutacija; R - Nasumična mutacija; B - Mutacija bliskih čvorova

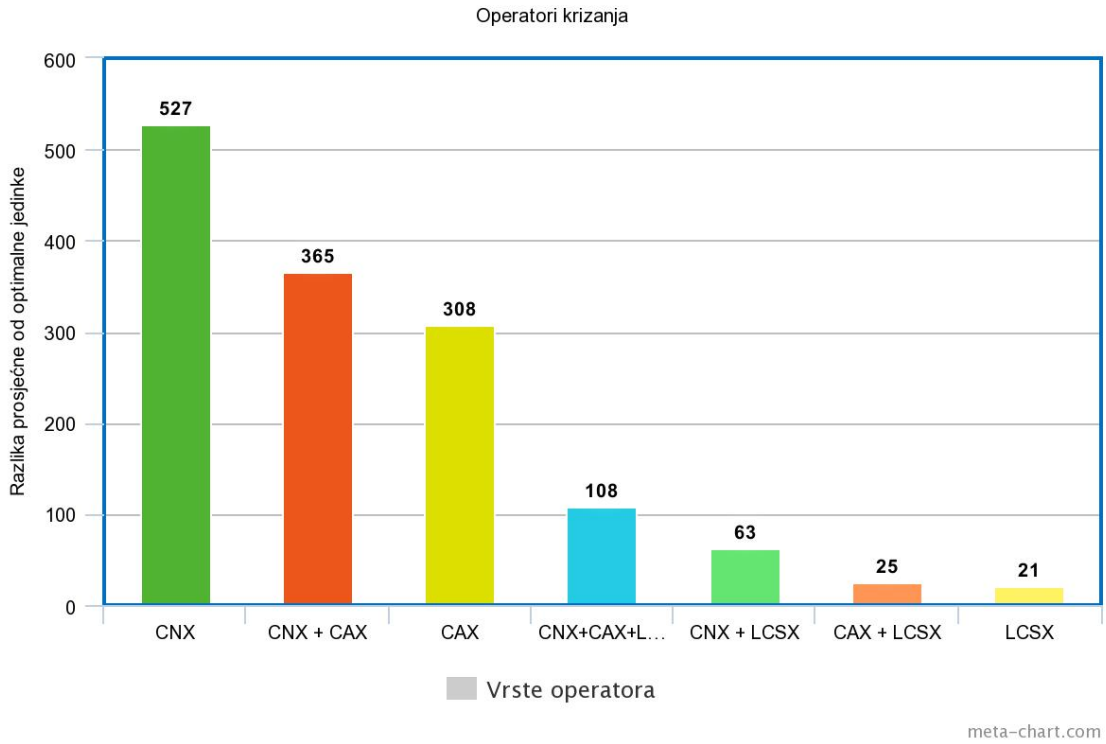
Mini mutacija nasumično izbacuje samo jedan čvor te ga ponovno dodaje umetajućom heuristikom, a ostale mutacije su opisane u odjeljku mutacija. Mini mutacija radi minimalne promjene te kao takva na dobiva pretjerano dobre rezultate uslijed toga se ni mutacija najvećeg obilaska nije pokazala kao dobar izbor, dok se kombinacija nasumične mutacije i mutacije bliskih čvorova pokazala kao najbolje rješenje.

5.2. Operatori križanja

U svim mjerenjima svi parametri osim korištenih operatora križanja su držani fiksni. Tako su mjerenja izvođena na populaciji od 100 jedinki u 1000 iteracija glavnog algoritma s ustaljenjem od 200 iteracija, mutacijska populacija je veličine 10 jedinki i izvodi se kroz 10 iteracija, a vjerojatnost mutacije je 0.5. Kao operator selekcije kori-

štena je 3-turnirska selekcija, a kao operatori mutacije korišteni su nasumična mutacija i mutacija bliskih čvorova.

Svaki od prikazanih podataka izračunat je iz 30 rezultata izvođenja programa pod istim uvjetima. Podatci prikazuju razliku prosječnog rješenja algoritma u 30 iteracija od optimalnog rješenja.



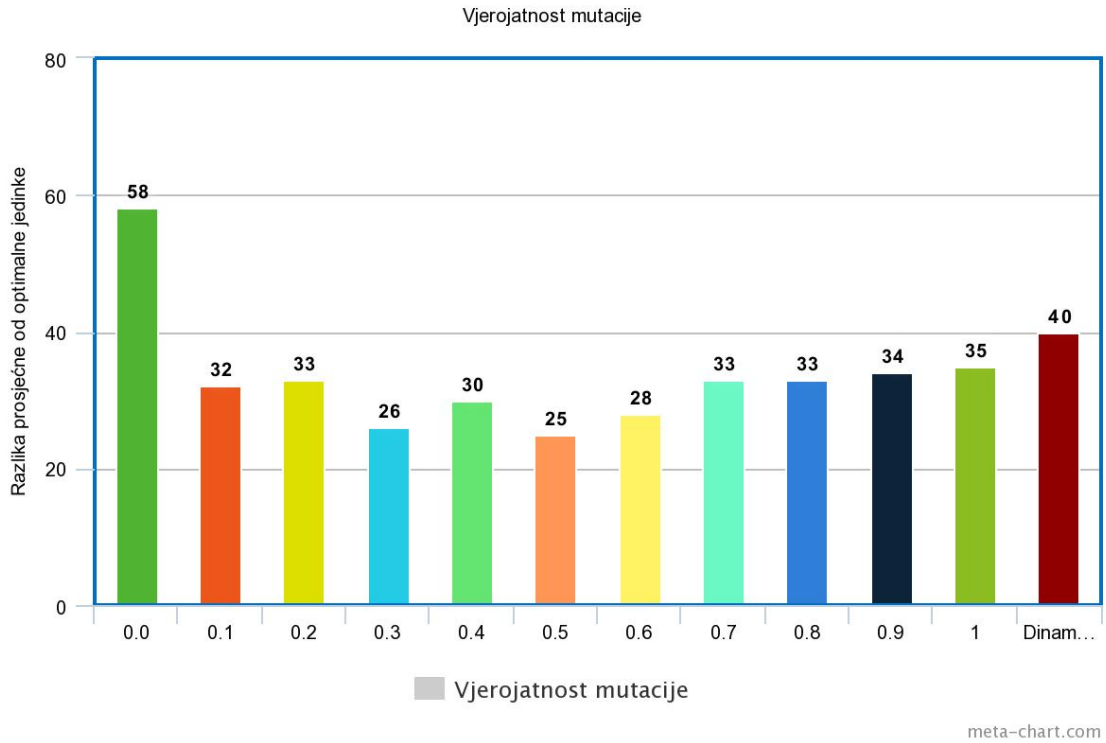
Slika 5.2: CNX - Križanje zajedničkih čvorova; CAX - Križanje zajedničkih lukova; LCSX - Križanje zajedničkih najduljih sekvenci

Kao što se može vidjeti po razlici grafa 5.1 i 5.2, način križanja ipak utječe više na kvalitetu rezultata nego mutacija. Tako se pokazalo da je LCSX definitivno najbolji način križanja te također dobro funkcionira i u kombinaciji s CAX-om, no čim se CAX koristi bez LCSX-a ili se uopće koristi CNX algoritam ne uspijeva uspješno pronalaziti kvalitetna rješenja. Iz toga se može zaključiti da je vjerojatno najbolja taktika čuvanja najduljih sekvenci unutar roditelja te da oni nose najbitnije dobre informacije.

5.3. Vjerojatnost mutacije

U svim mjerenjima svi parametri osim korištene vjerojatnosti mutacije su fiksni. Tako su mjerenja izvođena na populaciji od 100 jedinki u 1000 iteracija glavnog algoritma s ustaljenjem od 200 iteracija, mutacijska populacija je veličine 10 jedinki i izvodi se kroz 10 iteracija. Kao operator selekcije korištena je 3-turnirska selekcija i u jednom slučaju 3-turnirska eliminacija, kao operatori mutacije korišteni su nasumična mutacija i mutacija bliskih čvorova, a kao operatori križanja LCSX i CAX.

Svaki od prikazanih podataka izračunat je iz 30 rezultata izvođenja programa pod istim uvjetima. Podatci prikazuju razliku prosječnog rješenja algoritma u 30 iteracija od optimalnog rješenja.



Slika 5.3

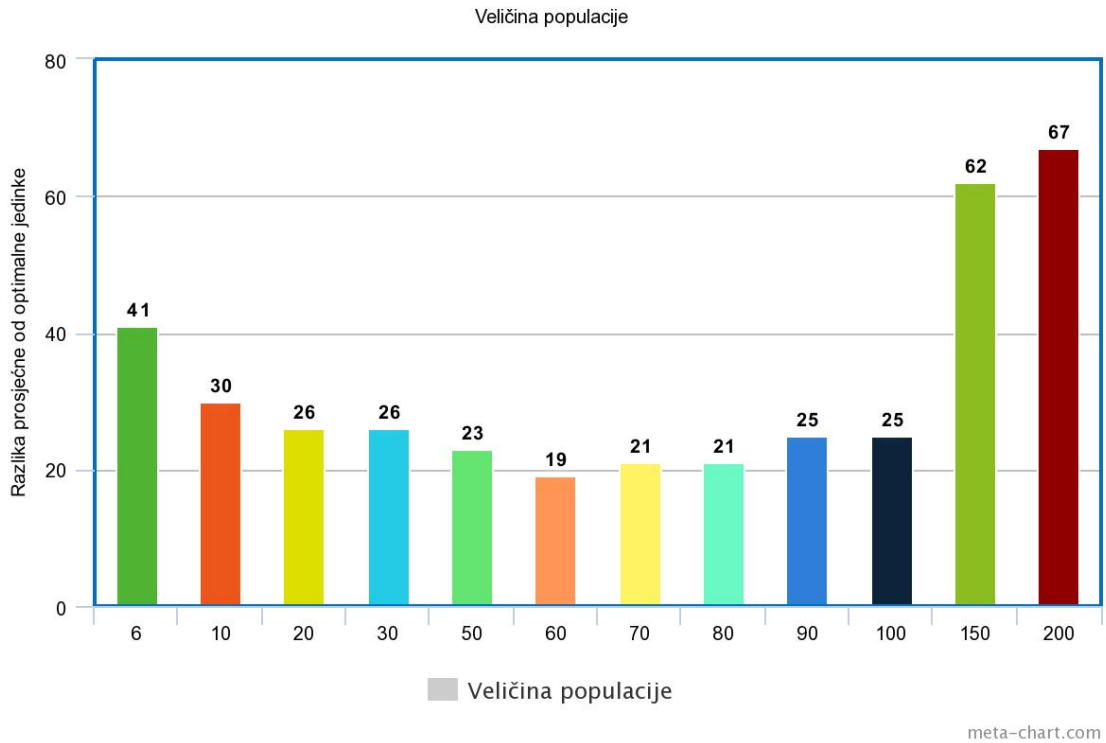
Bez mutacije algoritam pre brzo konvergira i ne uspijeva pronaći dobra rješenja, a zbog načina na koji je algoritam programski ostvaren(objašnjeno u sekciji o mutaciji) sve ostale vrijednost proizvode solidna rješenja gdje se najbolja ističu oko vjerojatnosti

0.5.

5.4. Veličina populacije

U svim mjerenjima svi parametri osim korištene veličine populacije su fiksni. Tako su mjerenja izvođena u 1000 iteracija glavnog algoritma s ustaljenjem od 200 iteracija, mutacijska populacija je veličine 10 jedinki i izvodi se kroz 10 iteracija. Kao operator selekcije korištena je 3-turnirska selekcija i u jednom slučaju 3-turnirska eliminacija, kao operatori mutacije korišteni su nasumična mutacija i mutacija bliskih čvorova, a kao operatori križanja LCSX i CAX.

Svaki od prikazanih podataka izračunat je iz 30 rezultata izvođenja programa pod istim uvjetima. Podatci prikazuju razliku prosječnog rješenja algoritma u 30 iteracija od optimalnog rješenja.



Slika 5.4

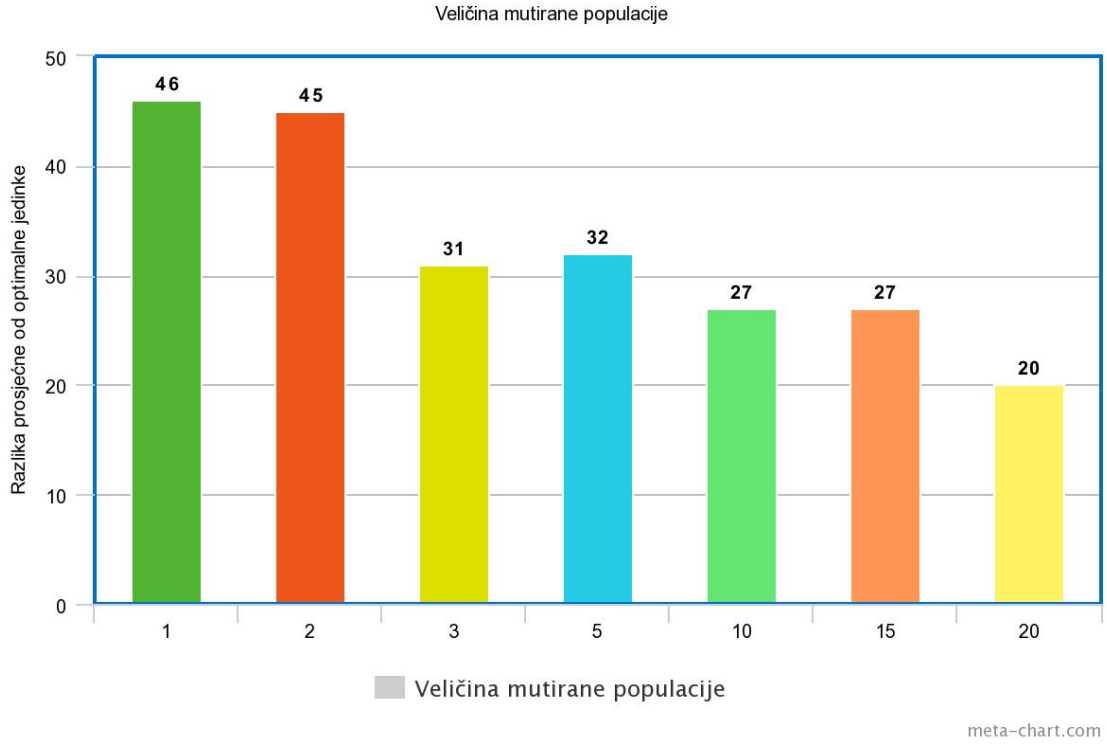
Smanjimo li populaciju na pre malu količinu nećemo imati dovoljno genetskog

materijala za pronaći najbolja rješenja, a ako ju previše povećamo imat ćemo previše jedinki i najbolje će teže dolaziti do izražaja. Upravo ovo se može uočiti i na grafu sa slike 5.4 gdje za populacije manje od 10 i veće od 100 rješenja ne uspijevaju biti zadovoljavajuća. Za konkretni algoritam ispostavilo se kao najbolja populacija od 60-ak jedinki.

5.5. Veličina mutirane populacije

U svim mjerenjima svi parametri osim korištene veličine mutirane populacije su fiksni. Tako su mjerenja izvođena na populaciji od 100 jedinki u 1000 iteracija glavnog algoritma s ustaljenjem od 200 iteracija, mutacijska populacija izvodi se kroz 10 iteracija. Kao operator selekcije korištena je 3-turnirska selekcija i u jednom slučaju 3-turnirska eliminacija, kao operatori mutacije korišteni su nasumična mutacija i mutacija bliskih čvorova, a kao operatori križanja LCSX i CAX.

Svaki od prikazanih podataka izračunat je iz 30 rezultata izvođenja programa pod istim uvjetima. Podatci prikazuju razliku prosječnog rješenja algoritma u 30 iteracija od optimalnog rješenja.



Slika 5.5

Za veličinu mutirane populacije od 1 se algoritam ponaša upravo kao da i nema te populacije već da je samo početna jedinka mutirana pa algoritam ne daje veoma dobra rješenja. Kako raste veličina mutirane populacije tako se i kvaliteta rješenja povećava ali se i duljina izvođenja jako povećava.

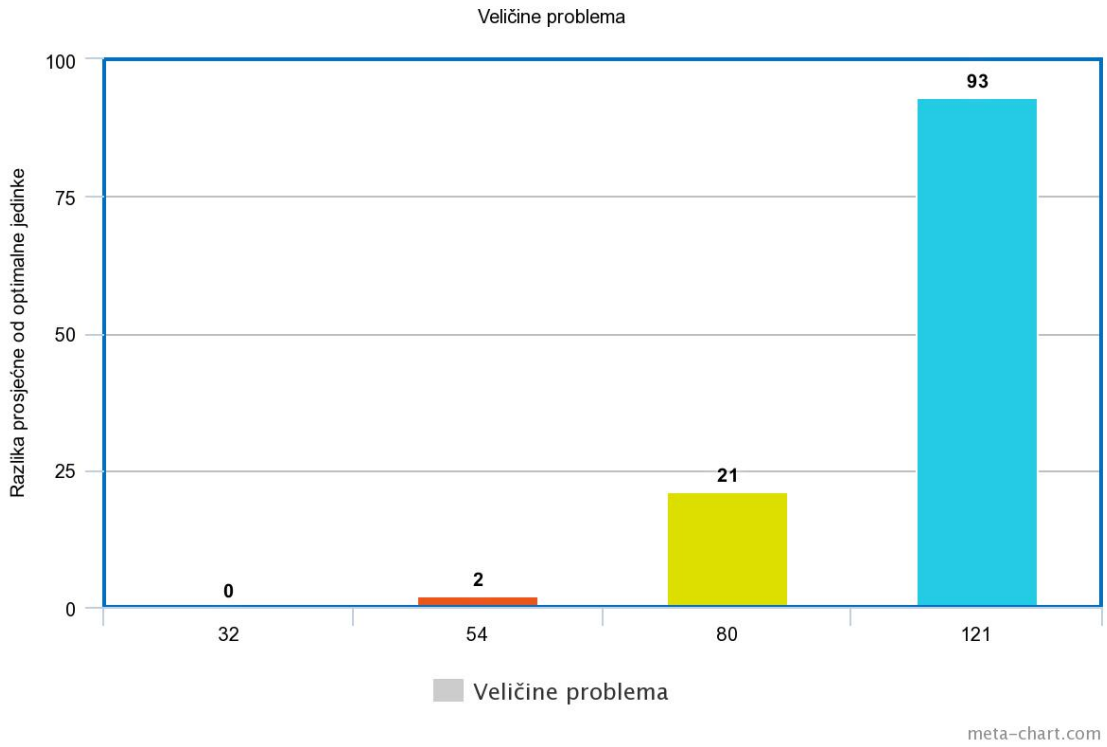
6. Usporedba na različitim veličinama problema

Pri izvođenju programa na različitim veličinama problema usmjeravanja vozila s kapacitivnim ograničenjem većinom su se iskazale 4 skupine:

1) Tako je prva skupina problema do otprilike 50 lokacija ona u kojoj algoritam uvijek uspije pronaći optimalno rješenje.
2) Druga skupina su problemi s između 50 i 70 lokacija u kojoj se u više od 50% slučajeva pronađe optimalno rješenje, a u ostatku je konačno rješenje veoma blizu optimalnome rješenju.
3) Treća skupina su problemi s između 70 i 120 lokacija za koje algoritam s pažljivo određenim parametrima samo u manje od 10% slučajeva pronalazi optimalno rješenje, a u ostatku je većinom blizu optimalnog dok ponekad i zaglavi u nekom ponešto lošijem rješenju .

4) Četvrta vrsta su problemi s više od 120 lokacija u kojima algoritam ne uspijeva pronaći optimalno ili trenutno znano najbolje rješenje, kako za većinu tako velikih problema ni nisu znana optimalna rješenja.

U stvaranju statistike sa slike 6.1 algoritam za svaki problem pokrenut je s istim parametrima te su korišteni prosječni rezultati iz 30 mjerenja za svaki problem.



Slika 6.1: Prosječno odstupanje najboljeg dobivenog rješenja od optimalnog rješenja u odnosu na broj lokacija.

U usporedbi sa slike 6.1 korišteni su:

1) Problem s 32 lokacije, 5 vozila te zatvorenosti od 82%[7].

2) Problem s 54 lokacije, 7 vozila i zatvorenosti od 96%[8].

3) Problem s 80 lokacija, 10 vozila i zatvorenosti od 94%[9].

4) Problem sa 121 lokacijom, 7 vozila i zatvorenosti od 98%[10].

Kako veličine problema rastu, tako raste i prostor mogućih rješenja, duljina izvođenja se povećava, a i zadatak pronalaska optimalnog rješenja se otežava.

7. Zaključak

Problem usmjeravanja vozila je NP složen problem iz stvarnoga svijeta za koji postoji više pristupa rješavanju. U ovome radu odabran je pristup evolucijskim algoritmom. Programski je ostvareno parsiranje ulaznih datoteka s problemom, umetajuća heuristika, ostvarena su 2 operatora selekcije, 3 operatora križanja i 3 operatora mutacije te i sami genetski algoritam za rješavanje.

Kao najbolji parametri algoritma iskazali su se veličina populacije od 60 jedinki, vjerojatnost mutacije od 50%, operator križanja LCSX, kombinacija dvaju operatora mutacije, nasumične i bliskih čvorova, te turnirska selekcija. Takva kombinacija parametara uspijeva u velikoj većini slučajeva pronaći optimalno rješenje u problemima do 70 lokacija, a u problemu s 80 lokacija u 10%slučajeva pronalazi optimalna rješenja, a prosječno je udaljeno oko 1% od optimalnog rješenja.

8. Literatura

[1] Golub, M., Genetski algoritam, prvi dio, Zagreb, 2004.

[2] Vaira, G., Genetic algorithm for vehicle routing problem, Vilinus, 2014.

[3] Gold, H., Carić, T., Vehicle routing problem, Zagreb, 2008.

[4] Jakobović, D., Adaptive genetic operators in elimination genetic algorithm, Zagreb.

[5] Vehicle routing data sets, 3 10 2003, https://www.coin-or.org/SYMPHONY/branchandcut/VRP/data/index.htm , 23 5 2018

[6] Vehicle routing problem, 9 4 2018, https://en.wikipedia.org/wiki/Vehicle_routing_problem, 23 5 2018

[7] Vehicle routing data set A-n32-k5, 3 10 2003, https://www.coin-or.org/SYMPHONY/branchandcut/VRP/data/A/A-n32-k5.vrp, 11 6 2018

[8] Vehicle routing data set A-n54-k7, 3 10 2003, https://www.coin-or.org/SYMPHONY/branchandcut/VRP/data/A/A-n54-k7.vrp, 11 6 2018

[9] Vehicle routing data set A-n80-k10, 3 10 2003, https://www.coin-or.org/SYMPHONY/branchandcut/VRP/data/A/A-n80-k10.vrp, 11 6 2018

[10] Vehicle routing data set M-n121-k7, 3 10 2003, https://www.coin-or.org/SYMPHONY/branchandcut/VRP/data/M/M-n121-k7.vrp, 11 6 2018

Usmjeravanje vozila evolucijskim algoritmom

Sažetak

Opisan je problem usmjeravanja vozila i općeniti genetski algoritam. Programski je ostvaren genetski algoritam za rješavanje problema usmjeravanja vozila. U sklopu toga su ostvarena 3 operatora mutacije, 3 operatora križanja i 2 operatora selekcije, umetajuća heuristika i parser za dokumente koji zadaju problem. Izvedeni su eksperimenti i prikazani rezultati algoritma pri varijaciji pojedinih parametara i operatora. Algoritam je isproban na više različito složenih problema.

Ključne riječi: Problem usmjeravanja vozila, evolucijski algoritam, križanje, mutacija, selekcija, jedinka, populacija.

Evolutionary Algorithm for Vehicle Routing

Abstract

The vehicle routing problem and the general evolutionary algorithm are described. A genetic algorithm for solving the vehicle routing problem was implemented. 3 mutation operators, 3 crossover operators, 2 selection operators, an insertion heuristic and a parser for documents which define the problem were implemented. Statistics have been made showing how successful the algorithm is for variations of parameters. The algorithm was tested on problems of different complexity.

Keywords: Vehicle routing problem, evolutionary algorithm, crossover, mutation, selection, unit, population.