SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1034

RASPOREĐIVANJE VOZILA U DINAMIČKIM UVJETIMA RADA

Josip Feliks

Zagreb, lipanj 2015.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA DIPLOMSKI RAD PROFILA

Zagreb, 26. veljače 2015.

Predmet: **Analiza i projektiranje računalom**

DIPLOMSKI ZADATAK br. 1034

Pristupnik: **Josip Feliks (0036445515)**
Studij: Računarstvo
Profil: Programsko inženjerstvo i informacijski sustavi

Zadatak: **Raspoređivanje vozila u dinamičkim uvjetima rada**

Opis zadatka:

Opisati problem raspoređivanja vozila i primjenu ovog modela raspoređivanja. Istražiti postojeće postupke raspoređivanja i definirati potrebne podatke za različite varijante problema. Odabrati relevantne ispitne podatke za usporedbu. Definirati način ocjene učinkovitosti ovisno o ispitivanom parametru. Pretražiti parametre s ciljem poboljšanja učinkovitosti algoritma genetskog programiranja. Ostvariti sustav za demonstraciju učinkovitosti u dinamičkim uvjetima i proizvoljnom kriteriju optimizacije uz pomoć genetskog programiranja. Ispitati učinkovitost različitih postupaka poboljšanja konačnog rješenja. Radu priložiti izvorne tekstove programa, dobivene rezultate uz potrebna objašnjenja i korištenu literaturu.

Zadatak uručen pristupniku: 13. ožujka 2015.
Rok za predaju rada: 30. lipnja 2015.

Mentor:

Izv. prof. dr. sc. Domagoj Jakobović

Djelovođa:

Doc. dr. sc. Igor Mekterović

Predsjednik odbora za diplomski rad profila:

Prof. dr. sc. Krešimir Fertalj

Zahvaljujem se svojoj djevojci Ivani što je uvijek bila uz mene. Hvala ti Ive što si mi pružala utjehu u teškim trenucima i iskreno se veselila u onim sretnim.

Zahvaljujem se svojim roditeljima što su uvijek vjerovali u mene i žrtvovali se da bi meni bilo lakše. Bez Vas ovo ne bi bilo moguće.

Zahvaljujem se svojem mentoru, prof. dr. sc. Domagoju Jakobovića koji mi je svojim savjetima i konzultacijama puno pomogao tijekom izrade ovog rada, kao i tijekom ostatka studija.

Sadržaj

1. Uvod

Problem raspoređivanja vozila (engl. *Vehicle Routing Problem - VRP*) poznati je optimizacijski problem koji je moguće susresti svakodnevno. Cilj je poslužiti skupinu kupaca sa skupinom vozila i pri tome prijeći najkraći put. Problem trgovačkog putnika instanca je problema raspoređivanja vozila u kojoj postoji samo jedno vozilo koje poslužuje kupce (obilazi lokacije).

Neki od primjera iz svakodnevnice su:

- upravljanje taxi službom,
- posluživanje trgovina robom iz skladišta (na primjer pekarska industrija),
- kurirska služba,
- izrada rasporeda vožnje školskih autobusa i sl.

U ovom radu ostvaren je sustav koji demonstrira učinkovitost genetskog programiranja u pronalasku rješenja za različite inačice problema raspoređivanja vozila. U drugom poglavlju detaljno je opisan problem raspoređivanja vozila. U trećem poglavlju opisano je genetsko programiranje općenito, način korištenja te sastavni dijelovi. U četvrtom poglavlju detaljno je opisan ostvareni sustav za demonstraciju. U petom poglavlju opisane su obuhvaćene inačice problema raspoređivanja vozila. U šestom poglavlju opisani su načini optimizacije koji su korišteni prilikom implementacije kako bi se postigli bolji rezultati. U sedmom poglavlju prikazani su ulazni parametri te rezultati ispitivanja raznih kombinacija parametara s ciljem pronalaska onih koji daju najbolje rezultate. U osmom poglavlju nalazi se zaključak rada. Navedena literatura korištena prilikom izrade ovog rada nalazi se u devetom poglavlju.

2. Problem raspoređivanja vozila

Problem raspoređivanja vozila (engl. *Vehicle routing problem, VRP*) formalno je prvi put predstavljen u radu [7] autora G. B. Dantzig i J. H. Ramser 1959. godine kao generalizacija problema trgovačkog putnika [4]. Problem raspoređivanja vozila formalno je definiran kao graf $G = (V, \mathcal{E}, C)$ gdje je $V = \{v_0, \dots, v_n\}$ skup vrhova, $\mathcal{E} = \{(v_i, v_j) \mid (v_i, v_j) \in V^2, i \neq j\}$ skup bridova i $C = (c_{ij})_{(v_i, v_j) \in \mathcal{E}}$ težinska matrica definirana nad skupom $\mathcal{E}$ koja predstavlja udaljenost između vrhova, vrijeme potrebno da se prevali brid ili cijena brida. Tradicionalno, vrh v_0 predstavlja skladište, a ostali vrhovi predstavljaju kupce, lokacije ili zahtjeve koje je potrebno poslužiti. Problem raspoređivanja vozila sastoji se od pronalaska skupa ruta za K identičnih vozila, koja se u početku nalaze u skladištu, takvih da je svaki kupac (vrh) posjećen točno jednom i da je pritom cijena bridova po kojima se prolazilo minimalna.

Postoji više različitih inačica problema raspoređivanja vozila [1]. Neki od njih su:

- Kapacitivni problem raspoređivanja vozila (engl. *Capacitated Vehicle Routing Problem - CVRP*)
- Kapacitivni problem raspoređivanja vozila s vremenskim okvirima (engl. *Capacitated Vehicle Routing Problem with Time Windows - CVRPTW*)
- Problem raspoređivanja vozila s više skladišta (engl. *Multiple Depot Vehicle Routing Problem - MDVRP*)
- Problem raspoređivanja vozila s više skladišta i vremenskim okvirima (engl. *Multiple Depot Vehicle Routing Problem with Time Windows - MDVRPTW*)
- Otvoreni problem raspoređivanja vozila (engl. *Open Vehicle Routing Problem - OVRP*) – vozila se ne moraju vratiti u skladište
- Problem raspoređivanja vozila s kupljenjem i dostavljanjem (engl. *Vehicle Routing Problem with Pickup and Delivery - VRPPD*) – roba mora biti prebačena s jednog mjesta na drugo
- I sl.

Osim formalne definicije, često je potrebno uključiti dodatne dvije dimenzije: tijek i kvaliteta informacije. Tijek informacije odnosi se na činjenicu da kod nekih inačica

problema dostupna informacija koju ima raspoređivač vozila može biti promjenjiva tijekom izvođenja, na primjer, dolazak novih zahtjeva korisnika. Kvaliteta informacije opisuje vjerojatnost točnosti dostupne informacije, na primjer, kad je poznat samo raspon potražnje pojedinog kupca, a ne konkretna vrijednost. Dodatno, ovisno o problemu i dostupnoj tehnologiji, rute mogu biti dodijeljene ili statički (a-priori) ili dinamički. Tablica 1 prikazuje četiri kategorije problema raspoređivanja vozila.

Tablica 1: Podjela problema raspoređivanja vozila po tipu

		Kvaliteta informacije	
		Deterministički ulaz	Stohastički ulaz
Tijek informacije	Ulaz poznat na početku	Statički i deterministički	Statički i stohastički
	Ulaz promjenjiv tijekom vremena	Dinamički i deterministički	Dinamički i stohastički

Kod statičkih i determinističkih problema sve informacije dostupne su u početku i nakon što se odrede rute, one se više ne mijenjaju.

Statički i stohastički problem raspoređivanja vozila karakteriziran je na način da su neke od ulaznih informacija slučajne i njihova vrijednost postaje poznata tijekom izvođenja. Pretpostavljeno je da su rute dodijeljene vozilima a-priori i da se smiju dogoditi samo male promjene kasnije, na primjer, preskakanje nekog kupca ili odlazak u skladište.

Kod dinamičkih i determinističkih problema, dio ili sve ulazne informacije su nepoznate u početku i postaju poznate tek tijekom izvođenja. Kod ovakvih inačica problema rute se dodjeljuju tijekom izvođenja i potrebna je stalna komunikacija između vozila i raspoređivača.

Slično kao i kod dinamičkih i determinističkih problema, kod dinamičkih i stohastičkih problema dio ili sve ulazne informacije postaju poznate tek tijekom izvođenja. Dodatno, vrijednosti nekih ulaznih informacija su slučajne. Ovdje je također potrebna stalna komunikacija između vozila i raspoređivača.

Različite inačice problema mogu imati različite razine dinamičnosti, a koje mogu biti karakterizirane kroz dvije dimenzije: frekvencija promjena i hitnost

zahtjeva [8]. Frekvencija promjena je brzina kojom nove informacije postaju dostupne, a hitnost zahtjeva razlika između vremena pristizanja zahtjeva i očekivanog vremena posluživanja. Tri različita izraza predložena su za mjerenje razine dinamičnosti problema. U radu [9] definiran je stupanj dinamičnosti δ kao omjer broja dinamičkih zahtjeva n_d i ukupnog broja zahtjeva n_{tot} i prikazan je izrazom (1).

$$\delta = \frac{n_d}{n_{tot}} \tag{1}$$

Temeljeno na činjenici da je vrijeme dospijeća zahtjeva također bitno, u radu [10] je predložen efektivni stupanj dinamičnosti δ^e. Ako se pretpostavi da zahtjevi poznati na početku imaju vrijeme dospijeća jednako 0, δ^e se može prikazati izrazom (2).

$$\delta^e = \frac{1}{n_{tot}} * \sum_{i \in R} \frac{t_i}{T} \tag{2}$$

T predstavlja duljinu planiranog intervala, R skup zahtjeva, a t_i vrijeme dospijeća zahtjeva $i \in R$.

U radu [10] također je proširen efektivni stupanj dinamičnosti na probleme s vremenskim okvirima da bi se uzela u obzir razina hitnosti zahtjeva. Definirano je vrijeme reakcije kao razlika vremena dospijeća zahtjeva i kraja vremenskog okvira zahtjeva. Stoga, proširen efektivni stupanj dinamičnosti prikazan je izrazom (3). t_i predstavlja dospijeće zahtjeva, a l_i kraj vremenskog okvira zahtjeva i.

$$\delta^e_{TW} = \frac{1}{n_{tot}} * \sum_{i \in R} (1 - \frac{l_i - t_i}{T}) \tag{3}$$

Oznaka da je neka inačica dinamičkog tipa prikazana je slovom „D“ na početku svake skraćenice inačice problema. U nastavku su pobliže objašnjene svaka od ostvarenih inačica problema u ovom radu.

Kod dinamičkog kapacitivnog problema raspoređivanja vozila (DCVRP) postoji samo jedno skladište u kojem se na početku nalaze sva vozila. Svako vozilo ima jednak ograničen kapacitet dobara koja može prevoziti. Nakon što količina dobara koje vozilo prevozi padne ispod određene razine, vozilo se vraća u skladište po još.

Nakon što vozilo postane dostupno, raspoređivač na temelju dostupnih informacija javlja vozilu njegovu sljedeću destinaciju. Ovdje ne postoje vremenski okviri unutar kojeg vozilo mora poslužiti kupca. Cilj je da svaki kupac bude poslužen, da se sva vozila vrate u skladište i da se pritom prijeđe što kraći put. Ukupni put dobija se zbrajanjem puteva svakog vozila. Podatak o minimalnom dopuštenom preostalom kapacitetu u vozilu prije nego što vozilo ide u skladište dostupan je i podesiv na početku izvođenja.

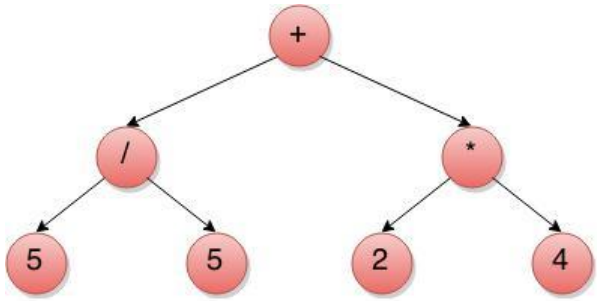
Kod dinamičkog problema raspoređivanja vozila s vremenskim okvirima (DVRPTW) također postoji samo jedno skladište u kojem se na početku nalaze sva vozila. Ovdje vozila imaju neograničen kapacitet i kao takva nisu potrebna dolaziti u skladište tijekom izvođenja. Postoji jedno drugo ograničenje, a to je da kupci moraju biti posluženi unutar određenog vremenskog okvira, inače se smatra da nisu posluženi. Isto kao i kod prethodne inačice problema, i ovdje vozilo, nakon što postane dostupno, od strane raspoređivača saznaje svoju sljedeću destinaciju.

Najsloženija inačica od tri ostvarene u ovom radu je dinamički kapacitivni problem raspoređivanja vozila s vremenskim okvirima (DCVRPTW). To je inačica problema koji je kombinacija prethodne dvije inačice. Vozila imaju ograničen kapacitet i postoje vremenski okviri unutar kojih kupac mora biti poslužen, inače se smatra neposluženim. Kao i do sad, problem je dinamičkog tipa, stoga vozilo, nakon što postane dostupno, od raspoređivača saznaje svoju sljedeću destinaciju.

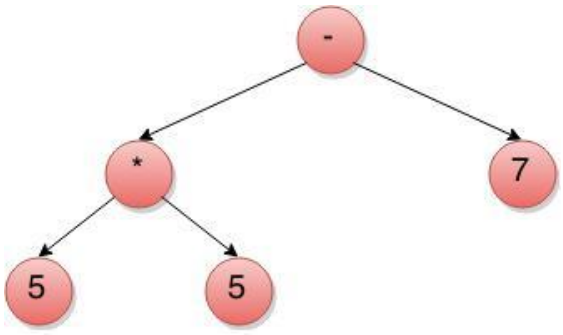
3. Genetsko programiranje

Tehnologija korištena za određivanje parova vozilo – zahtjev u ovom radu je genetsko programiranje. Genetsko programiranje spada u skupinu evolucijskih algoritama. Način na koji funkcionira je taj da genetsko programiranje simulira prirodu, tj. koristeći operatore križanja i mutacije, jedinka kroz generacije evoluira. Struktura rješenja kod genetskog programiranja je stablo. Kao i kod svake stablaste strukture, i ovdje postoje unutarnji i vanjski čvorovi. Vanjski čvorovi se ujedno nazivaju i listovi. Svaki čvor sadrži informaciju, tj. dodijeljen mu je jedan primitiv. Primitiv se dijeli na funkcije i terminale. Unutarnji čvorovi stabla su uvijek funkcije, a vanjski (listovi) su terminali. Stoga se u praksi često koristi naziv funkcijski čvor i terminalni čvor. Funkcijski čvor, kao što mu i ime kaže, sadrži kao informaciju neku funkciju, na primjer zbrajanje, oduzimanje, množenje, dijeljenje, dok terminalni čvor sadrži vrijednost neke varijable, ili jednostavno broj, na primjer 1, 0.5, -245 i sl..

Način funkcioniranja je taj da se u početku stvori inicijalna populacija od nekoliko stabala (jedinki). Postoje dvije metode za stvaranje jedinke: *full* metoda i *grow* metoda. Kod *full* metode svaka grana ima jednaku dubinu, dok kod *grow* metode to nije nužno. Izgled izgrađenog stabla *full* metodom prikazan je na slici 1, a *grow* metodom na slici 2. Nakon što se stvori inicijalna populacija, potrebno je evaluirati svaku jedinku iz populacije.



Slika 1: Stablo izgrađeno full metodom



Slika 2: Stablo izgrađeno grow metodom

3.1. Selekcija

Selekcija je postupak koji služi za odabir jedinki unutar populacije koje će svojim križanjem stvoriti jedinku boljeg genetskog materijala od njih samih. Selekcija je vrlo bitan dio evolucijskih algoritama jer način na koji se odabiru jedinke iz populacije direktno utječe kakav će genetski materijal biti proslijeđen u sljedeću generaciju, a samim time kakva će biti i kvaliteta rješenja u sljedećoj generaciji. Selekcija se može podijeliti na generacijsku i eliminacijsku selekciju.

Kod generacijske selekcije potomci potpuno zamjenjuju populaciju roditelja u svakoj generaciji. Potrebno je stvoriti onoliko jedinki koliko ima i roditelja. Jedinke se stvaraju križanjem i mutacijom roditelja. S obzirom da bi na taj način sljedeća generacija imala manje jedinki, dodaju se još duplikati jedinki da bi se dobila ista veličina populacije kao i u prethodnoj generaciji. To je loše jer duplikati smanjuju genetsku raznolikost, a time i područje pretraživanja.

Eliminacijska selekcija češće je korištena nego generacijska. Ovdje bolje jedinke preživljavaju, a lošije bivaju zamijenjene jedinkama koje nastaju križanjem i mutacijom boljih jedinki. Nema potrebe za stvaranjem duplikata i na taj način se ne smanjuje genetska raznolikost i područje pretraživanja. Potrebno je napomenuti da je kod eliminacijske selekcije ugrađen elitizam. Elitizam je proces u kojem najbolja jedinka uvijek prelazi u sljedeću generaciju i time se čuva najkvalitetnije rješenje. Elitizam nije implicitno ugrađen unutar generacijske selekcije već se mora dodatno ugraditi.

Osim podjele selekcije na generacijsku i eliminacijsku, postoji podjela na turnirsku selekciju i proporcionalnu selekciju.

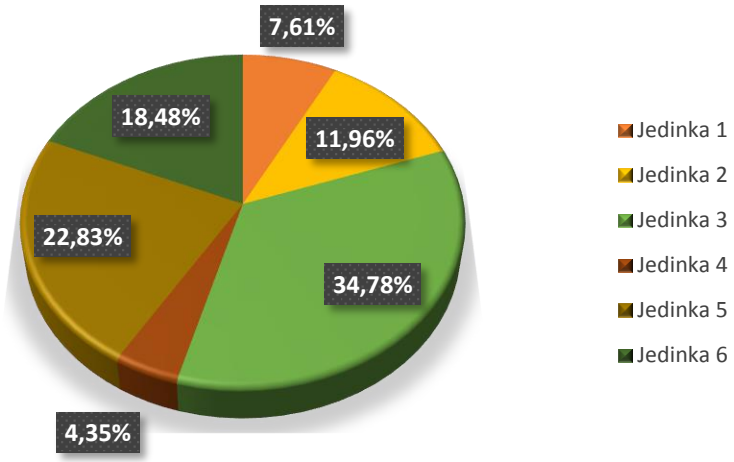
Turnirski odabir ili turnirska selekcija je najčešće korištena u praksi. Ideja je da se na početku iz populacije odabere *k* slučajnih jedinki koje ulaze u turnir. Veličina turnira određena je brojem *k*, a najčešće se za *k* uzima broj tri. Nakon što se odabere *k* jedinki koje ulaze u turnir, one se sortiraju po dobroti. Dvije najbolje jedinke križaju se i stvaraju novu jedinku. Nakon što je stvorena, ovisno o vjerojatnosti mutacije, jedinka će mutirati. Najlošija jedinka u turniru odbacuje se i na njeno mjesto dolazi novonastala jedinka. Razlog zbog kojeg se najčešće za broj *k* postavlja broj tri je taj što u takvom turniru i lošije jedinke imaju šansu biti odabrane za križanje čime se njihov genetski materijal prenosi u sljedeću generaciju. Pokazalo se da s većom disperzijom postoji veća vjerojatnost za pronalazak kvalitetnog rješenja. Pseudo-kôd algoritma koji koristi turnirski-eliminacijsku selekciju prikazan je na slici 3.

```
single generation {
 repeat(deme size times) {
  randomly add <nTournament_> individuals to the tournament;
  select the worst one in the tournament;
  randomly select two parents from remaining ones in the tournament;
  replace the worst with crossover child;
  perform mutation on child;
 }
}
```

Slika 3: Pseudo-kôd turnirski eliminacijske selekcije (preuzeto s [5])

Proporcionalni odabir ili proporcionalna selekcija funkcionira na način da se zbroje sve vrijednosti evaluacije jedinki i da ta suma iznosi 100%. Potom se za svaku jedinku odredi postotak koliko je ona pridonijela ukupnoj sumi. Na taj način jedinke koje više pridonose ukupnoj sumi, odnosno kvalitetnije jedinke, imaju veću šansu biti odabrane. Jedinke se zatim nasumično odabiru iz populacije uzimajući u obzir vjerojatnost njihova odabira. Grafički prikaz proporcionalne selekcije prikazan je na slici 4.

Jedinke	Dobrota
Jedinka 1	7
Jedinka 2	11
Jedinka 3	32
Jedinka 4	4
Jedinka 5	21
Jedinka 6	17



Slika 4: Proporcionalna selekcija

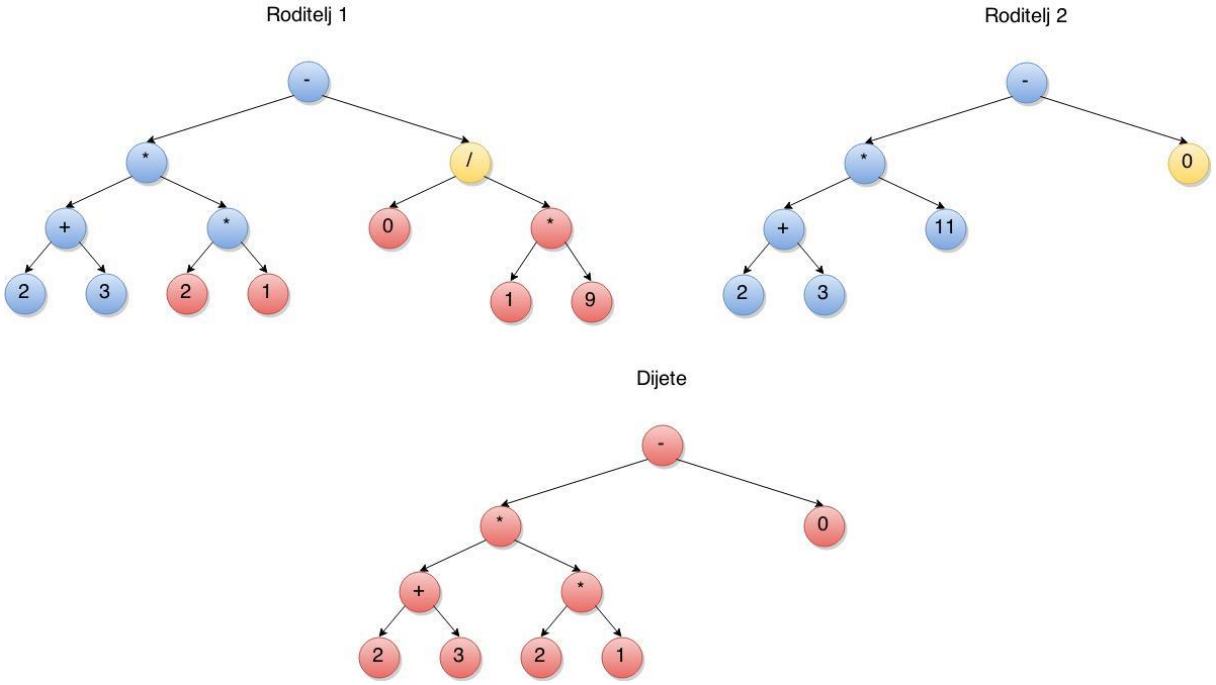
3.2. Križanje

Križanje je proces u kojem se genetski materijal dvije jedinke na određeni način spaja i time stvara novu jedinku. Nova jedinka se naziva dijete, a jedinke koje su se križale da bi nastala nova jedinka nazivaju se roditelji. Cilj je da novonastala jedinka, to jest dijete, bude bolja od roditelja. To nije zajamčeno jer postoji mogućnost da dijete naslijedi lošije gene od roditelja. Postoji više različitih načina križanja, a u nastavku su opisani samo neki.

3.2.1. Križanje s jednom točkom

Križanje s jednom točkom (engl. *one-point crossover*) [6] je križanje kod kojeg se nastoji križati čvorove koji se nalaze na istoj dubini. Zajednička regija pojam je koji se veže uz ovu vrstu križanja. Prvo je potrebno odrediti zajedničku regiju između čvorova koji su ušli u proces križanja. U zajedničku regiju spadaju oni parovi čvorova koji imaju jednak broj operanada. Ako to vrijedi, čvor i njegova djeca ulaze u zajedničku regiju. Ako to ne vrijedi, čvor svejedno ulazi u zajedničku regiju, ali se njegova djeca preskaču. Nakon što je određena zajednička regija, iz nje se

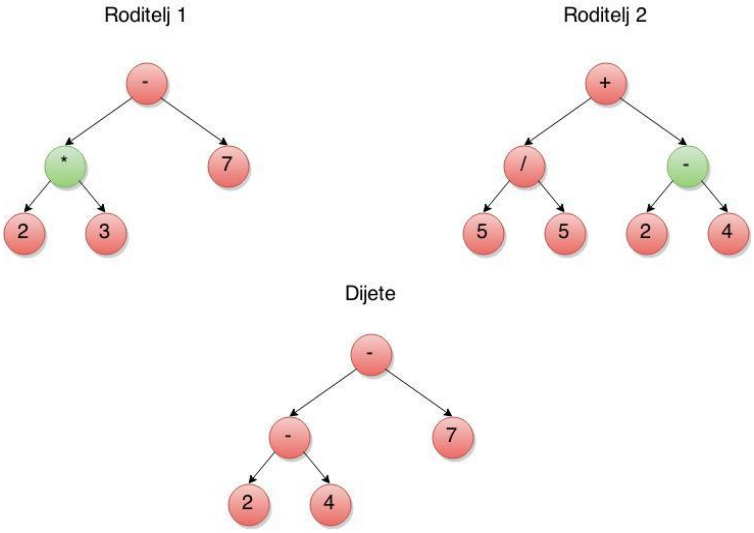
nasumično odabire jedan čvor. Taj čvor je istovjetan za oba roditelja i naziva se točka križanja. Nakon toga je proces generiranja djeteta jednostavan. Dijete je kopija prvog roditelja osim točke križanja i podstabla kojem je točka križanja korijen. Genetski materijal tog dijela uzima se od drugog roditelja. Slika 5 prikazuje primjer križanja s jednom točkom. Plavom bojom označena je zajednička regija kod roditelja. Žutom bojom označen je odabrani čvor. Dijete je identično roditelju 1 osim pripadajuće točke križanja i njenog podstabla. Taj dio preuzet je od roditelja 2. U ovom slučaju, od roditelja 2 preuzet je samo terminalni čvor vrijednosti nula.



Slika 5: Primjer križanja s jednom točkom

3.2.2. Križanje podstabla

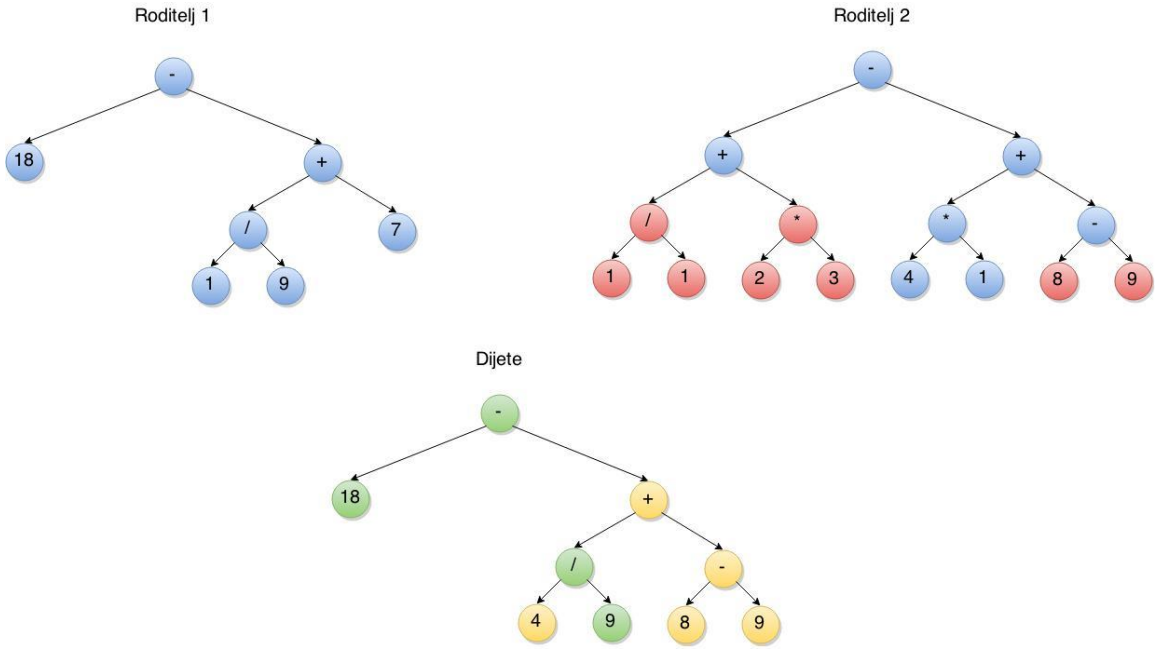
Križanje podstabla (engl. *subtree crossover*) [6] križanje je kod kojeg se u svakom roditelju nasumično odabere čvor koji se smatra točkom križanja. Nakon toga, dijete se generira tako da se uzme genetski materijal prvog roditelja i na mjesto točke križanja postavi podstablo drugog roditelja čiji je korijen točka križanja kod drugog roditelja. Primjer križanja podstabla prikazan je na slici 6. Zelenom bojom označene su točke križanja kod oba roditelja.



Slika 6: Primjer križanja podstabla

3.2.3. Uniformno križanje

Uniformno križanje (engl. *uniform crossover*) [6] križanje je kod kojeg se, isto kao i kod križanja s jednom točkom, prvo određuje zajednička regija. Nakon toga se za svaki čvor, koji se nalazi u zajedničkoj regiji, nasumično određuje hoće li se uzeti od prvog ili drugog roditelja. Ukoliko se dogodi da je čvor na granici zajedničke regije i čvor nije list, tada se osim njega preuzimaju i sva podstabla kojima je ovaj čvor korijen. Slika 7 prikazuje primjer uniformnog križanja. Kod roditelja su plavom bojom označene zajedničke regije. Kod djeteta su zelenom bojom označeni čvorovi preuzeti od prvog roditelja, a žutom od drugog roditelja.



Slika 7: Primjer uniformnog križanja

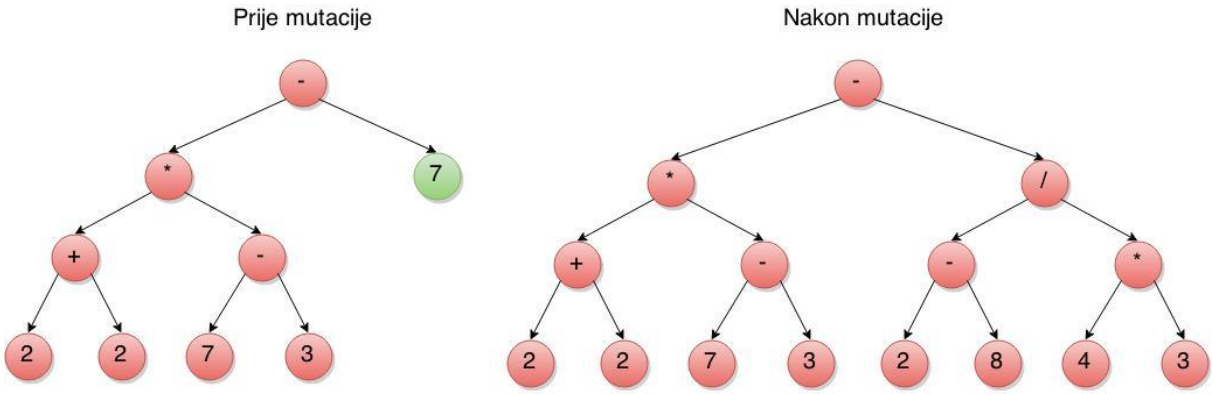
3.3. Mutacija

Mutacija je unarni operator i, za razliku od križanja gdje su potrebne dvije jedinke, potrebna je jedna jedinka na kojoj se događa mutacija. Ovisno o vjerojatnosti, mutacija može i ne mora nastupiti. Kao i kod križanja, postoji više načina mutacije. Često se događa da prilikom evolucije jedinka zapne u neki lokalni optimum. Kad ne bi bilo mutacije, jedinka bi ostala u lokalnom optimumu i prikazala ga kao najbolje rješenje. Zato se koristi mutacija kao operator jer ona, ili unošenjem novog genetskog materijala ili permutacijom čvorova u stablu, može izbaviti jedinku iz lokalnog optimuma. Međutim, nigdje nije zajamčeno da mutacija popravlja kvalitetu jedinke.

3.3.1. Mutacija podstabla

Mutacija podstabla (engl. *subtree mutation*) vrsta je mutacije koja može uvelike promijeniti genetski materijal jedinke. Radi na principu da se unutar stabla nasumično odabere čvor koji će mutirati. Zatim se nasumično generira podstablo s korijenom u prije odabranom čvoru. Potrebno je paziti da se ne naruši ograničenje maksimalne dubine stabla. Ovo je jedna od najčešće korištenih mutacija upravo

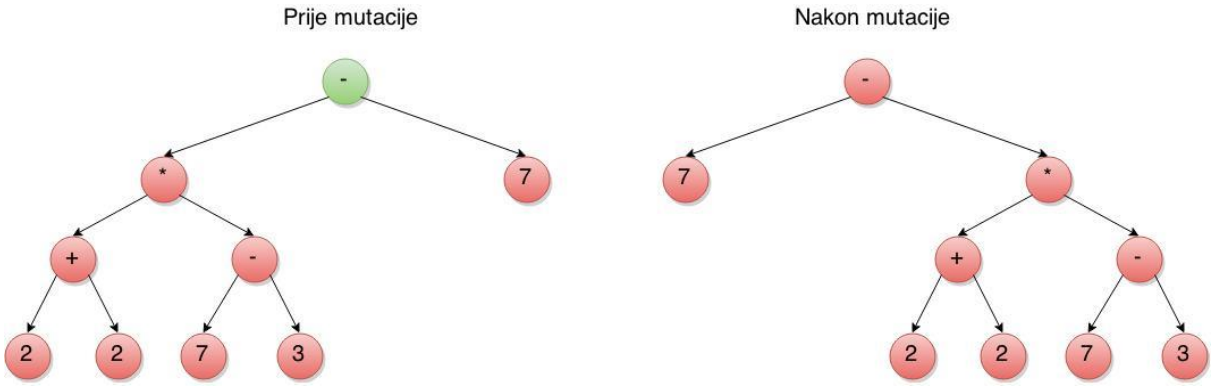
zato što može donijeti velike promjene u jedinku i na taj način izaći iz lokalnog optimuma ukoliko se jedinka našla u njemu. Primjer mutacije podstabla prikazan je na slici 8. Zelenom bojom označen je čvor odabran za mutaciju.



Slika 8: Primjer mutacije podstabla

3.3.2. Permutirajuća mutacija

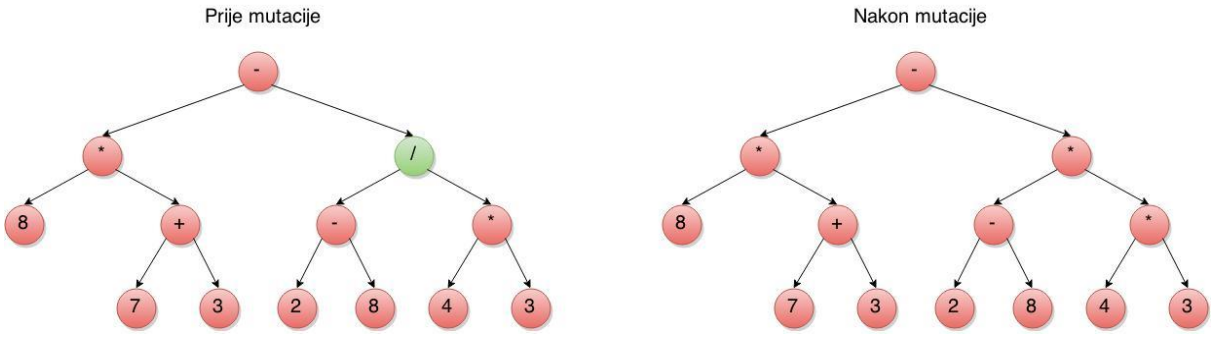
Permutirajuća mutacija (engl. *permutation mutation*) vrsta je mutacije koja ne donosi novi genetski materijal. Jedinka mutira tako što se nasumično odabere čvor u stablu i njegova se djeca ispremiješaju. Da bi ova mutacija imala smisla, čvor koji je odabran za mutaciju mora imati barem dvoje djece, to jest, mutacija nije primjenjiva na listove ili funkcije koje imaju samo jedan operand. Primjer permutirajuće mutacije prikazan je na slici 9. Korijen stabla je odabran za mutiranje. Jedina promjena koja se dogodila je da je lijevo dijete korijena postalo desno dijete i obrnuto. Treba napomenuti da iako mutacija nije unijela novi genetski materijal, kvaliteta dobivene jedinke može se bitno razlikovati od kvalitete jedinke prije mutacije.



Slika 9: Primjer permutirajuće mutacije

3.3.3. Nadomještajuća mutacija

Nadomještajuća mutacija (engl. *node replacement mutation*) vrsta je mutacije koja unosi novi genetski materijal [2]. Jedinka mutira na način da se nasumično odabere čvor koji će mutirati i na njegovo mjesto stavi novi čvor koji sadrži drugi funkcijski ili terminalni operator. Uvjet koji mora biti zadovoljen je da novi čvor mora imati jednaki broj djece kao i čvor koji mutira. Na primjer, unutarnji čvor koji sadrži funkciju zbrajanja može biti zamijenjen sa čvorom koji sadrži funkciju množenja, ali ne i sa čvorom koji sadrži sinus kao funkciju. Primjer nadomještajuće mutacije prikazan je na slici 10. Zelenom bojom označen je odabrani čvor za mutaciju.



Slika 10: Primjer nadomještajuće mutacije

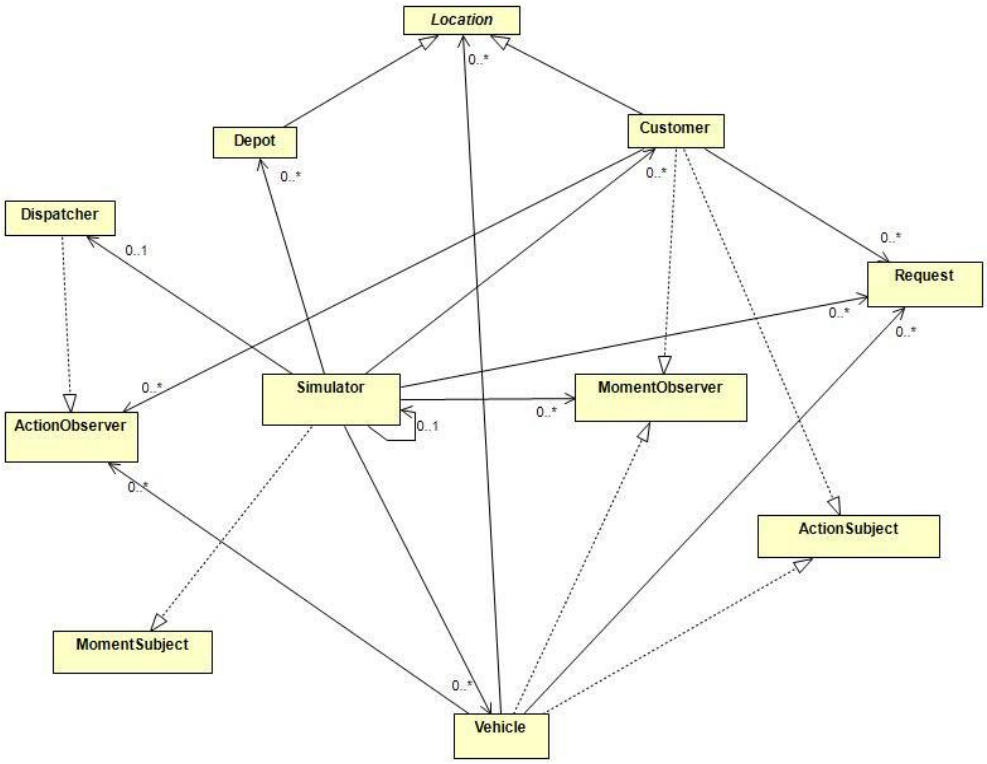
3.4. Kriterij zaustavljanja

Osim opisanih operatora, potrebno je još odrediti kriterij zaustavljanja algoritma. Postoje više različitih kriterija, a ovdje će biti navedeni samo neki. Kao kriterij zaustavljanja može se postaviti broj uzastopnih generacija bez poboljšanja kvalitete

rješenja. Ako se nakon, na primjer 50 generacija, trenutno najbolje rješenje ne poboljša, zaključak je da je algoritam došao u lokalni, ili u najboljem slučaju globalni, minimum ili maksimum, i trenutno najbolje rješenje se uzima kao konačno rješenje izvršavanja. Kao kriterij zaustavljanja algoritma može se postaviti maksimalno vrijeme izvođenja nakon kojeg se prekida izvođenje i trenutno najbolje rješenje se uzima kao konačno. Još jedan kriterij koji se dosta često koristi je postavljanje zahtjeva za određenom kvalitetom rješenja. Ukoliko rješenje u nekom slučaju bude jednako zahtijevanom, ili pak bolje, izvođenje se prekida.

4. Sustav za demonstraciju učinkovitosti u dinamičkim uvjetima i proizvoljnom kriteriju optimizacije

Sustav za demonstraciju učinkovitosti središnja je točka ovog rada. Pojednostavljeni UML dijagram cjelokupnog sustava prikazan je na slici 11.



Slika 11: Pojednostavljeni UML dijagram cjelokupnog sustava

Entiteti koji se nalaze unutar sustava su vozilo (engl. *Vehicle*), lokacija (engl. *Location*), zahtjev (engl. *Request*), raspoređivač (engl. *Dispatcher*) i simulator (engl. *Simulator*). Funkcija vozila, lokacije i zahtjeva je sama po sebi jasna. Unutar simulatora nalaze se svi potrebni i aktualni podaci uz pomoć kojih se obavlja uparivanje vozila i zahtjeva. Potrebno je napomenuti da tijekom izvođenja jedna lokacija šalje više od jednog zahtjeva. S obzirom da vrijeme unutar simulatora nije kontinuirano nego postoje diskretni vremenski trenuci, u simulatoru su pohranjeni podaci aktualni za svaki vremenski trenutak. Vremenski trenuci su spremljeni u listu koja se tijekom izvođenja algoritma stalno osvježava. Novi vremenski trenutak

nastaje kad neki zahtjev postane aktivan ili neko vozilo postane dostupno. Prilikom pokretanja simulatora iz datoteke se čitaju svi zahtjevi koji će u budućnosti postati aktivni. Lista vremenskih trenutaka osvježava se svaki put kad vozilo postane dostupno. Ti vremenski trenuci nisu na početku nigdje zapisani i ovise o načinu dodjele zahtjeva vozilu.

Sustav koji stvara parove vozila i zahtjeva naziva se raspoređivač. Odluku o tome kojem vozilu će biti dodijeljen zahtjev raspoređivač donosi uz pomoć genetskog programiranja. Pseudo-kôd raspoređivača prikazan je na slici 12.

```
novi vremenski trenutak;
ako je stigao novi zahtjev {
      ako postoje dostupna vozila {
            dodijeli vozilo novom zahtjevu;
      }
} inače je vozilo postalo dostupno {
      ako postoji zahtjev na čekanju {
            dodijeli zahtjev vozilu;
      }
}
```

Slika 12: Pseudo-kôd raspoređivača

Prilikom primitka nove informacije, raspoređivač provjerava je li ona stigla od strane vozila ili kupca. Ako je stigla od strane vozila, znači da vozilo javlja da je dostupno, a ako od kupca, kupac šalje novi zahtjev. Na temelju te informacije, raspoređivač provjerava postoji li mogućnost uparivanja nekog od dostupnih vozila i nekog od aktivnih zahtjeva. Ako postoji, otpremnik dodjeljuje novi zahtjev vozilu.

Unutar cjelokupnog sustava na dva mjesta je implementiran oblikovni obrazac *Singleton*. *MomentSubject* i *MomentObserver* sučelja su koja su implementirana radi praćenja vremenskog trenutka. Simulator upravlja vremenskim trenucima i kod svake promjene javlja vozilima i kupcima, koji su implementirali sučelje *MomentObserver*, novu informaciju. Vozila i kupci implementiraju osim sučelja *MomentObserver* i sučelje *ActionSubject*. Ono služi za javljanje nove informacije raspoređivaču koji implementira sučelje *ActionObserver*.

5. Ostvarene inačice problema raspoređivanja vozila

U ovom radu ostvarene su tri različite inačice problema raspoređivanja vozila – dinamički kapacitivni problem raspoređivanja vozila (DCVRP), dinamički problem raspoređivanja vozila s vremenskim okvirima (DVRPTW) te dinamički kapacitivni problem raspoređivanja vozila s vremenskim okvirima (DCVRPTW). Sve tri inačice problema detaljno su opisane u drugom poglavlju. U ovom poglavlju prikazan je skup funkcijskih i terminalnih čvorova te funkcija cilja. Skup funkcijskih čvorova jednak je za sve tri inačice i prikazan je tablicom 2.

Tablica 2: Prikaz funkcijskih čvorova

Oznaka	Opis
+	Zbrajanje
-	Oduzimanje
*	Množenje
/	Dijeljenje
Abs	Apsolutna vrijednost
Avg	Prosječna vrijednost
Log10	Logaritam po bazi 10
Sqrt	Drugi korijen
Ifgt	Uvjetni operator >
Iflt	Uvjetni operator <

Funkcija cilja jednaka je za sve tri inačice i prikazana je izrazom (4).

$$rezultat = 100000 * a + b \tag{4}$$

a – broj neposluženih zahtjeva

b – ukupna prijeđena udaljenost

S obzirom da je primarni cilj poslužiti sve zahtjeve i, uz to prijeći najkraći put, rješenje kod kojeg postoji neposluženi zahtjev biva strogo kažnjeno, što je vidljivo po skalaru uz varijablu *a* unutar izraza (4).

Terminali se razlikuju kod ostvarenih inačica. Inačica DCVRP za terminale ima udaljenost između dvije lokacije (engl. *distance*) koja se mjeri u sekundama, te potražnja kupca (engl. *demand*) i preostali kapacitet dobara u vozilu (engl. *remaining capacity*) koji se mjere u kilogramima. Inačici DVRPTW su na raspolaganju drukčiji terminalni čvorovi. S obzirom da se u ovoj inačici ne gleda na kapacitet, to jest, smatra se da vozila imaju beskonačan kapacitet, dostupni terminali su samo oni čija je mjerna jedinica sekunda, a to su udaljenost, vrijeme početka vremenskog okvira (engl. *ready time*), vrijeme završetka vremenskog okvira (engl. *due date*) i vrijeme potrebno da se posluži kupac (engl. *service time*). Inačica DCVRPTW na raspolaganju ima uniju terminalnih čvorova inačica DCVRP i DVRPTW.

Popis terminala prikazan je tablicom 3 koja se nalazi u sljedećem poglavlju.

6. Optimizacije

Razni načini optimizacije algoritma mogu poslužiti za poboljšanje konačnog rješenja. U ovom radu ostvarene su dvije vrste optimizacije – semantičko genetsko programiranje i iterativno raspoređivanje. Oba načina optimizacije detaljno su opisana u nastavku.

6.1. Semantičko genetsko programiranje

Semantičko genetsko programiranje posebna je vrsta genetskog programiranja. Kod klasičnog genetskog programiranja dovoljno je da stablo bude sintaksno ispravno. Sintaksno ispravno znači da stablo ima ispravnu strukturu i da se može prevesti u ispravnu matematičku formulu. Pod ispravnom strukturom smatra se da ukoliko funkcijski čvor sadrži funkciju koja ima dva operanda, na primjer zbrajanje, onda taj čvor mora imati dvoje djece. Da bi algoritam uopće pravilno radio stablo mora biti sintaksno ispravno.

Semantički ispravna stabla, osim što moraju biti sintaksno ispravna, moraju zadovoljavati i dodatna pravila. Semantički ispravno stablo ispravno je i unutar domene problema, a ne samo matematički. Dodatna informacija koja je ostvarena je mjerna jedinica čvora, odnosno primitiva unutar pripadnog čvora. Osim same mjerne jedinice, potrebno je voditi evidenciju i o potenciji mjerne jedinice, inače stablo neće biti semantički ispravno. Neke funkcije zahtijevaju da im operandi imaju jednaku mjernu jedinicu uključujući i potenciju, na primjer funkcija zbrajanja, dok kod nekih funkcija to nije bitno, na primjer funkcija množenja. Unutar sustava postoje dva različita tipa informacije – informacija koja se mjeri sekundama i informacija koja se mjeri kilogramima. Tablica 3 prikazuje sve terminale i njihove mjerne jedinice, a tablica 4 ograničenje mjernih jedinica kod različitih funkcija.

Tablica 3: Popis terminala i njihovih mjernih jedinica:

Terminal	Mjerna jedinica
Udaljenost (engl. *distance*)	Sekunda (broj sekundi potreban za prijeći određenu udaljenost)
Potražnja (engl. *demand*)	Kilogram
Preostali kapacitet dobara u vozilu (engl. *remaining capacity*)	Kilogram
Vrijeme početka vremenskog okvira (engl. *ready time*)	Sekunda
Vrijeme završetka vremenskog okvira (engl. *due date*)	Sekunda
Vrijeme trajanja posluživanja kupca (engl. *service time*)	Sekunda

Tablica 4: Popis implementiranih funkcija

Funkcija	Ograničenje	Računanje mjerne jedinice	Broj argumenata
Zbrajanje; +	Lijevo i desno dijete moraju imati iste potencije	Potencija lijevog ili desnog djeteta (neovisno kojeg jer su jednake)	2
Oduzimanje; -	Lijevo i desno dijete moraju imati iste potencije	Potencija lijevog ili desnog djeteta (neovisno kojeg jer su jednake)	2
Množenje; *	Nema	Zbroj potencija lijevog i desnog djeteta	2
Dijeljenje; /	Nema	Razlika potencija lijevog i desnog djeteta	2
Drugi korijen; $\sqrt{\ }$	Nema	Potencija djeteta	1
Uvjetna funkcija >	Prvo i drugo dijete moraju imati iste potencije	Potencija trećeg ili četvrtog djeteta (ovisno o uvjetu)	4
Uvjetna funkcija <	Prvo i drugo dijete moraju imati iste potencije	Potencija trećeg ili četvrtog djeteta (ovisno o uvjetu)	4
Logaritmiranje; $\log_{10}$	Nema	Potencija djeteta	1
Apsolutna vrijednost	Nema	Potencija djeteta	1
Prosječna vrijednost	Lijevo i desno dijete moraju imati iste potencije	Potencija lijevog ili desnog djeteta (neovisno kojeg jer su jednake)	2

Semantičko genetsko programiranje generirati će uvijek fizikalno ispravne izraze, dok klasično genetsko programiranje neće. Iz tog razloga potrebno je modificirati način na koji se stvara stablo prilikom inicijalizacije te operatore križanja i mutacije. S obzirom da naglasak ovog rada nije na semantičkom genetskom programiranju već je to samo jedan oblik optimizacije, ostvarena je samo *full* metoda izgradnje semantički ispravnog stabla. Kao operator križanja ostvareno je križanje s jednom točkom, a umjesto klasične mutacije ostvarena je mutacija podstabla. I novo ostvareno križanje i mutacija na izlazu daju semantički ispravno stablo.

U nastavku su detaljno opisani način izgradnje, križanja i mutacije semantički ispravnog stabla.

6.1.1. Izgradnja semantički ispravnog stabla

Izgradnja semantički ispravnog stabla bitno se razlikuje od izgradnje klasičnog stabla. Ovo je najvažniji dio semantičkog genetskog programiranja jer operatori križanja i mutacije pretpostavljaju da je stablo na kojem se oni izvode semantički ispravno. Ako stablo u početku nije semantički ispravno, neće biti ni nakon križanja ili mutacije. Iz tog razloga osmišljen je algoritam za izgradnju semantički ispravnog stabla. Kao što je već naglašeno, algoritam izgradnje temelji se na *full* metodi izgradnje stabla.

Da bi se algoritam mogao što jednostavnije predočiti potrebno je krenuti od jednostavnijih primjera. Za početak, neka je potrebno izgraditi semantički ispravno stablo dubine 1. Numeracija dubine kreće od 0, stoga će ovo stablo imati jedan funkcijski čvor, koji je ujedno i korijen, i jedno, dvoje ili četvero djece ovisno o funkciji koja je sadržana u korijenu. Ako korijen sadrži funkciju zbrajanja, njegova djeca moraju imati jednake mjerne jedinice. Redoslijed generiranja čvorova je takav da se prvo generira korijen, zatim lijevo dijete pa desno dijete. Na početku, korijenu nije dodijeljena niti jedna mjerna jedinica. Nakon što se generira lijevo dijete, koje je ujedno list, i njegova mjerna jedinica, i s obzirom da korijen trenutno ne sadrži nikakvu mjernu jedinicu, ona se preslikava s lijevog djeteta na korijen. Sada korijen sadrži jednaku mjernu jedinicu kao i lijevo dijete. S obzirom da korijen sadrži funkciju zbrajanja, a ograničenje kod funkcije zbrajanja je da oba djeteta moraju imati jednaku mjernu jedinicu, ta mjerna jedinica se prosljeđuje desnom djetetu kao uvjet. Desno dijete sada mora generirati terminalni čvor koji ima mjernu jedinicu jednaku

proslijeđenom uvjetu. Konkretno, ako je lijevo dijete generiralo terminalni čvor s informacijom o potražnji kupca (engl. *demand*) koja se mjeri u kilogramima, onda i desno dijete mora generirati terminalni čvor s informacijom koja se mjeri u kilogramima, to jest, nasumično odabrati između potražnje kupca i preostalog kapaciteta unutar vozila (engl. *remaining capacity*). U slučaju da je u korijenu sadržana funkcija koja nema nikakvih ograničenja, terminalni čvorovi se generiraju nasumično.

Složeniji primjer je izgradnja semantički ispravnog stabla veće dubine. Redoslijed generiranja čvorova je i dalje jednak – djeca se generiraju s lijeva na desno s tim da lijevo dijete mora biti potpuno generirano prije nego se krene na desno. Ovdje se misli na izgradnju podstabla kojemu je lijevo dijete korijen. Nakon što se izgradilo lijevo podstablo provjerava se trenutni čvor i funkcija koju on sadrži. Ako ne postoji ograničenje na funkciju, to jest, potrebno je obaviti množenje, dijeljenje, odrediti drugi korijen, odrediti apsolutnu vrijednost ili logaritmirati, desno dijete ne mora imati jednaku potenciju kao i lijevo. Ovo je jednostavniji slučaj jer se funkcija za korijen desnog podstabla može odabrati nasumično. Kompliciraniji slučaj je ako postoji uvjet i desno dijete mora imati jednaku potenciju kao i lijevo. Onda se gleda je li potrebna mjerna jedinica jednostavna ili složena. Pod jednostavnom mjernom jedinicom smatra se $s,\ s^2,\ kg,\ kg^{-3}$ i slično. Pod složenom mjernom jedinicom smatra se skg^2, s^4kg^{-2} i slično.

Ako je mjerna jedinica jednostavna izračunavaju se maksimalna i minimalna potencija koje se mogu postići na trenutnoj dubini i označavaju kao gornja i donja granica. Ako tražena mjerna jedinica ima potenciju unutar tih granica, odabire se nasumično funkcija. Ako tražena mjerna jedinica ima potenciju ispod dolje granice kao funkcija se odabire funkcija dijeljenja. Ako tražena mjerna jedinica ima potenciju iznad gornje granice kao funkcija se odabire funkcija množenja. Množenjem i dijeljenjem nastoji se osigurati da potencija bude unutar gornje i donje granice i u konačnici da podstablo ima traženu mjernu jedinicu.

Ako je mjerna jedinica složena, postupak je drukčiji. Najprije se za svaku mjernu jedinicu gleda potencija je li veća ili manja od nule. Ako su potencije svih mjernih jedinica veće od nule, kao funkcija se odabire funkcija množenja. Ako su potencije

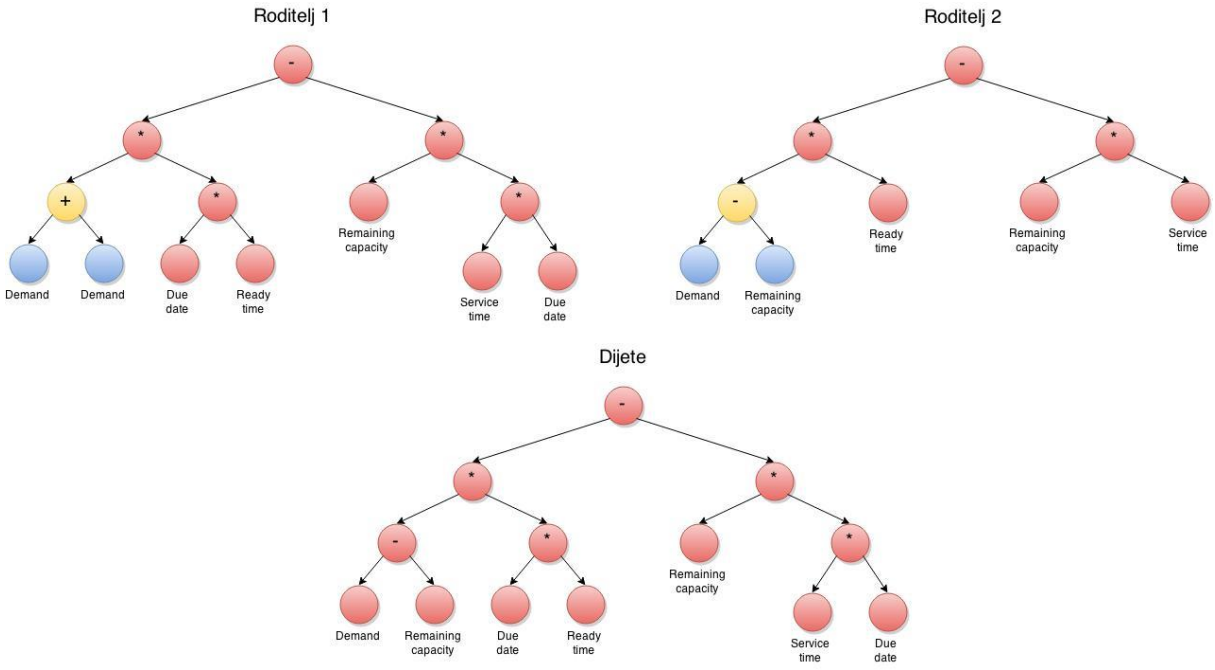
svih mjernih jedinica manje od nule, kao funkcija se odabire funkcija dijeljenja. Ako postoje i pozitivne i negativne potencije, odabire se funkcija dijeljenja.

Slijedeći korak je određivanje načina na koji će se tražena mjerna jedinica, jednostavna ili složena, podijeliti na djecu čvora. Ako odabrana funkcija ima samo jedan operand, na primjer funkcija logaritmiranja, tada se tražena mjerna jedinica u cijelosti predaje jedinom djetetu čvora. Ako je odabrana funkcija iz skupa zbrajanja, oduzimanja ili određivanja prosječne vrijednosti, tada se mjerna jedinica u cijelosti predaje i jednom i drugom djetetu. Ako je odabrana funkcija iz skupa uvjetnih funkcija, tada se svakom djetetu u cijelosti predaje tražena mjerna jedinica. Iako kod uvjetnih funkcija postoji ograničenje da prvo i drugo dijete moraju imati istu potenciju te u slučaju potrebe za konkretnom mjernom jedinicom treće i četvrto dijete također, ovdje je radi jednostavnosti implementirano da sva djeca imaju jednaku mjernu jedinicu. Na taj način nije narušena semantika stabla. Ako je odabrana funkcija množenja, potencija mjerne jedinice rastavlja se na dva dijela. Prvi dio će biti tražena mjerna jedinica prvom djetetu, a drugi dio drugom djetetu. Potencija se podijeli na dva i rezultat dijeljenja je nova potencija mjerne jedinice namijenjena prvom djetetu. Ostatak dijeljenja nova je potencija mjerne jedinice koja je namijenjena drugom djetetu. U slučaju da je tražena potencija jednaka nuli, gleda se trenutna dubina. Ako je trenutna dubina za jedan manja od maksimalne dubine, i jednom i drugom djetetu se kao tražena mjerna jedinica prosljeđuje trenutno tražena mjerna jedinica s potencijom nula. U slučaju da je trenutna dubina za više od jedan manja od maksimalne dubine, prvom djetetu se prosljeđuje tražena mjerna jedinica s potencijom 1, a drugom djetetu tražena mjerna jedinica s potencijom -1. Ako je odabrana funkcija dijeljenja, postupak je vrlo sličan postupku kod funkcije množenja osim nakon što se potencija podijeli na dva, ostatak dijeljenja se množi s -1, a tek onda prosljeđuje drugom djetetu. Ako tražena mjerna jedinica ima potenciju nula, i jednom i drugom djetetu se prosljeđuje tražena mjerna jedinica s potencijom 1.

Objašnjeni algoritam izgrađuje semantički ispravno stablo koje se kasnije koristi kod križanja i mutacije.

6.1.2. Križanje semantički ispravnih stabala

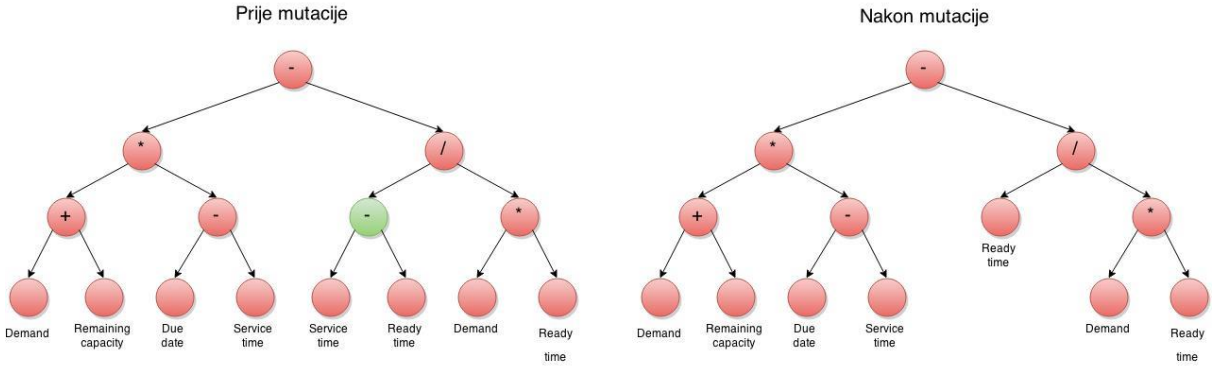
Kao što je već napomenuto, fokus ovog rada nije na semantičkom genetskom programiranju, stoga je implementiran samo jedan tip križanja, a to je križanje s jednom točkom semantički ispravnih stabala. Razlika u odnosu na klasično križanje s jednom točkom je način na koji se određuje zajednička regija. Kod klasičnog križanja u zajedničku regiju spadali su oni čvorovi koji su imali jednak broj operanada. Dodatan uvjet koji je potrebno zadovoljiti da bi jedinka nastala križanjem bila semantički ispravna je da u zajedničku regiju ulaze oni čvorovi roditelja koji, osim što imaju jednak broj operanada, imaju i jednaku semantičku informaciju, to jest, jednaku mjernu jedinicu. Potrebno je napomenuti da, na primjer, s i s^2 nisu jednake mjerne jedinice. Znači, mora se apsolutno sve poklapati da bi se čvorovi mogli naći u zajedničkog regiji. Dalje je postupak jednak kao i kod klasičnog križanja s jednom točkom. Slika 13 prikazuje primjer križanja s jednom točkom koje zadovoljava semantička pravila. Plavom bojom označena je zajednička regija kod roditelja, a zatim je unutar zajedničke regije nasumično odabran čvor (žuta boja) kao točka križanja. Dijete dobiveno križanjem također je semantički ispravno.



Slika 13: Primjer semantičkog križanja s jednom točkom

6.1.3. Mutacija semantički ispravnog stabla

Iz istog razloga zašto je ostvaren samo jedan tip križanja semantički ispravnih stabala, ostvaren je i jedan tip mutacije i to mutacija podstabla semantički ispravnog stabla. Kod korištenja mutacije podstabla jedinke, unutar jedinke se odabire čvor koji će biti korijen podstablu koje će mutirati. Da bi jedinka nakon mutacije i dalje bila semantički ispravna, novo generirano podstablo mora imati jednaku mjernu jedinicu kao što je imalo i podstablo prije mutacije. To je jedina prilagodba u odnosu na klasičnu mutaciju podstabla. Slika 14 prikazuje semantičku mutaciju podstabla. Zelenom bojom označen je čvor koji je odabran za mutaciju. Mjerna jedinica čvora prije mutacije je, isto kao i nakon mutacije, s.



Slika 14: Primjer semantičke mutacije podstabla

6.2. Iterativno raspoređivanje

Iterativno raspoređivanje druga je od dvije ostvarene optimizacije u ovom radu. Princip rada je da se uvode nove informacije koje pospješuju kvalitetu odabira sljedećeg vozila ili zahtjeva. Nove informacije implementirane su kao novi terminalni čvorovi. Vrijednost novih terminalnih čvorova je promjenjiva, a u početku je postavljena na vrlo veliku vrijednost. Na kraju svake iteracije ponovno se izračunavaju vrijednosti novih terminalnih čvorova. Cilj je iterirati sve dok je novo rješenje bolje od prethodnog. Tablica 5 prikazuje novo dodane informacije.

Tablica 5: Popis novih informacija

Oznaka čvora	Mjerna jedinica
Ukupno vrijeme čekanja	Sekunda
Ukupno dostupno vrijeme	Sekunda
Ukupni preostali kapacitet	Kilogram

Ukupno vrijeme čekanja (engl. *total wait time*) suma je vremena čekanja svih vozila. Ukoliko vozilo dođe do kupca prije otvaranja vremenskog okvira, ono mora čekati. Cilj je smanjiti vrijeme čekanja jer je vozilo u tom trenutku nedostupno, a ne obavlja posao. Ova nova informacija koristi se kod dviju ostvarenih inačica: DVRPTW i DCVRPTW. Ukupno dostupno vrijeme (engl. *total available time*) suma je vremena dostupnosti svih vozila. Nakon što je neko vozilo poslužilo određenog kupca, ono je dostupno sve dok mu se ne dodijeli novi zahtjev. Cilj je rasporediti vozila na način da su podjednako opterećena. U tom slučaju smanjiti će se ukupno vrijeme dostupnosti svih vozila. Ova nova informacija koristi se kod dviju ostvarenih inačica: DVRPTW i DCVRPTW. Ukupan preostali kapacitet (engl. *total remaining capacity*) suma je preostalih kapaciteta dobara u svim vozilima. Razlog zašto je uvedena ova informacija je jer se želi minimizirati broj odlazaka u skladište, a samim time i prijeđena udaljenost. Cilj je rasporediti vozila na način da su podjednako opterećena i da preostali kapacitet dobara na kraju izvođenja kod svakog vozila bude što manji. Ova nova informacija koristi se kod dviju ostvarenih inačica: DCVRP i DCVRPTW.

Korištenjem iterativnog raspoređivanja, minimalno dva puta će se izvoditi raspoređivanje za jednu jedinku. Razlog je taj što se u prvoj iteraciji za vrijednost dobrote jedinke postavlja vrlo velika vrijednost. Nakon što završi prva iteracija i rasporede se vozila, izračunava se vrijednost dobrote jedinke. Kako je cilj minimizirati vrijednost, ona će uvijek biti bolja od vrijednosti koja je postavljena na početku. Nakon što se vidi da je vrijednost manja, ponovno se izračunavaju vrijednosti novih informacija (u početku su bile postavljene na vrlo velike vrijednosti) i ponovno se kreće u postupak raspoređivanja. Broj iteracija ovisi o poboljšanju vrijednosti dobrote jedinke iz iteracije u iteraciju. Ako se dogodi da je vrijednost dobrote jedinke veća od trenutno najbolje, proces se zaustavlja i u jedinku se sprema najbolja (minimalna) vrijednost dobrote.

7. Rezultati

U ovom poglavlju prikazani su rezultati ispitivanja kvalitete rješenja uz različite parametre genetskog programiranja. U obzir je uzet utjecaj dubine stabla, utjecaj duljine stagnacije, utjecaj veličine populacije, utjecaj broja generacija, utjecaj vjerojatnosti mutacije. Također će biti prikazani rezultati dobiveni klasičnim genetskim programiranjem te rezultati dobiveni bez utjecaja iterativnog raspoređivanja.

Cjelokupni sustav ostvaren je programskim jezikom *Java* koristeći skup alata za evolucijsko računanje ECF (*Evolutionary Computation Framework*).

S obzirom da trenutno ne postoje globalno priznati ispitni primjeri za dinamičko programiranje, autor je bio primoran uzeti jedan statični ispitni primjer iz [3] i uz pomoć njega generirati datoteku sa zahtjevima. Odabran je ispitni primjer u kojem postoji 200 kupaca i svaki kupac ima jedan zahtjev. Kupci su, gledajući koordinate, smješteni u grozdove. U ovom radu generirana je datoteka unutar koje svaki kupac tijekom vremena izvođenja šalje tri zahtjeva. Generirana datoteka stoga ima 600 zahtjeva. Broj vozila je postavljen na 50, a kapacitet vozila na 200. Ovaj ispitni primjer služi za učenje algoritma. Ispitni primjer za ocjenu učinkovitosti, tj. za ispitivanje utjecaja pojedinog parametra također je generiran s obzirom da ne postoje globalno priznati ispitni primjeri. U ovom slučaju uzet je ispitni primjer u kojem postoji 400 kupaca i svaki kupac ima jedan zahtjev. Naknadno je generiran ispitni primjer kod kojeg svaki kupac ima tri zahtjeva, stoga postoji ukupno 1200 zahtjeva. Broj vozila je 100, a svako vozilo ima kapacitet 700. Kupci su, kao i kod ispitnog primjera za učenje, smješteni u grozdove. Razlog zašto se ne ispituje učinkovitost na istom ispitnom primjeru je što su ovdje ostvarene dinamičke inačice problema raspoređivanja vozila i cilj je pronaći vrijednosti parametara koje nisu specifične za pojedini slučaj nego se mogu globalno koristiti.

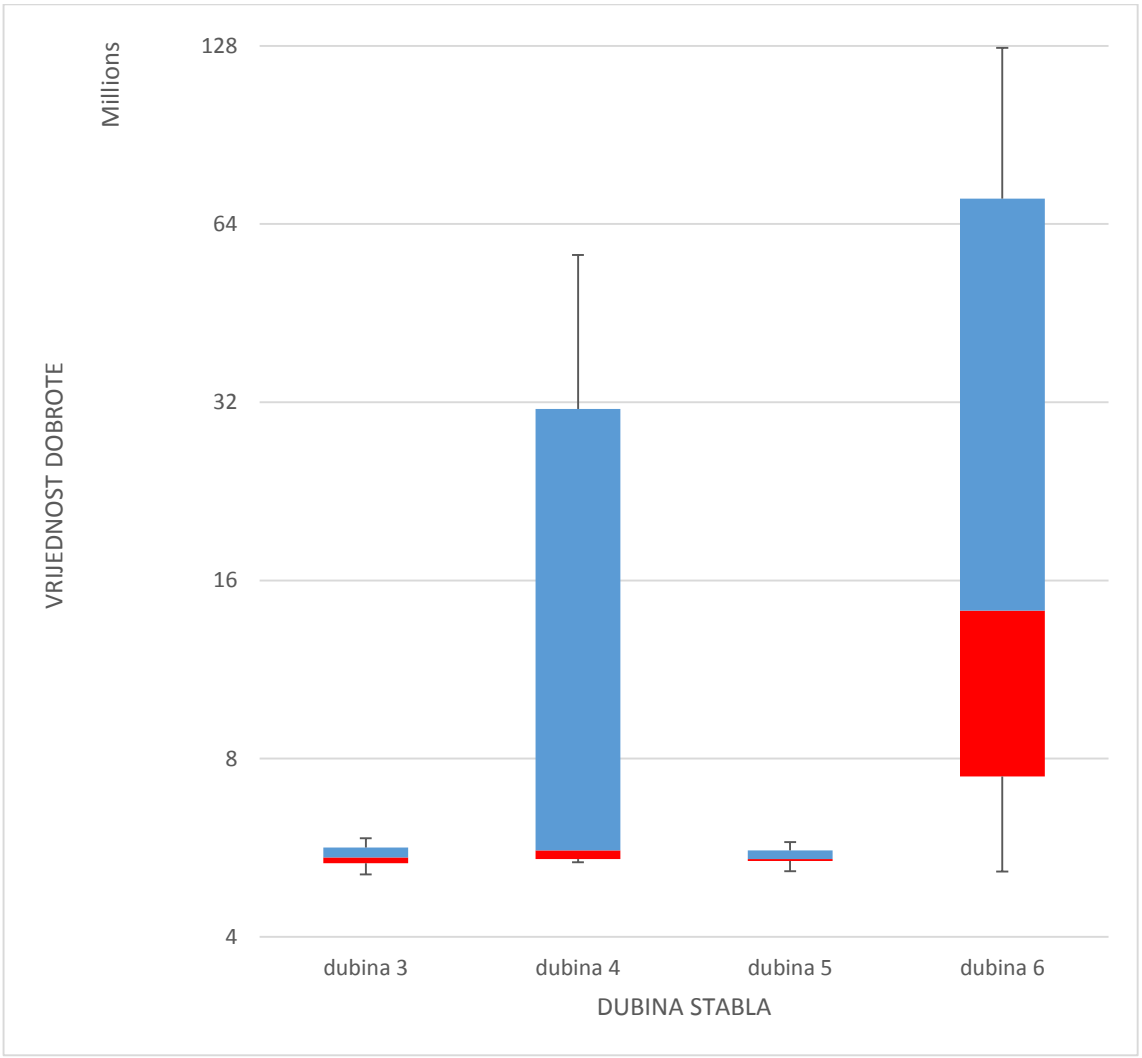
Nakon što je algoritam naučen na ispitnom primjeru za učenje, proslijeđen je ispitnom primjeru za ocjenu učinkovitosti iz kojeg su proizašli rezultati prikazani u narednim poglavljima. Sva ispitivanja izvodila su se na najsloženijoj inačici, DCVRPTW. Početna konfiguracija algoritma genetskog programiranja slijedi:

- Dubina stabla = 3

- Duljina stagnacije = 25
- Veličina populacije = 50
- Broj generacija = 500
- Vjerojatnost mutacije = 0.3
- Korišteno semantičko genetsko programiranje
- Korišteno iterativno raspoređivanje

7.1. Utjecaj dubine stabla na konačno rješenje

Dubina stabla drastično može povećati ili smanjiti skup mogućih rješenja. U ovom radu ispitivanja su se izvodila na četiri različite dubine stabla. Ispitivanje je pokrenuto dvadeset puta. Box plot za različite dubine stabala prikazan je na slici 15. Korištena je logaritamska skala zbog nemogućnosti prikaza svih vrijednosti.

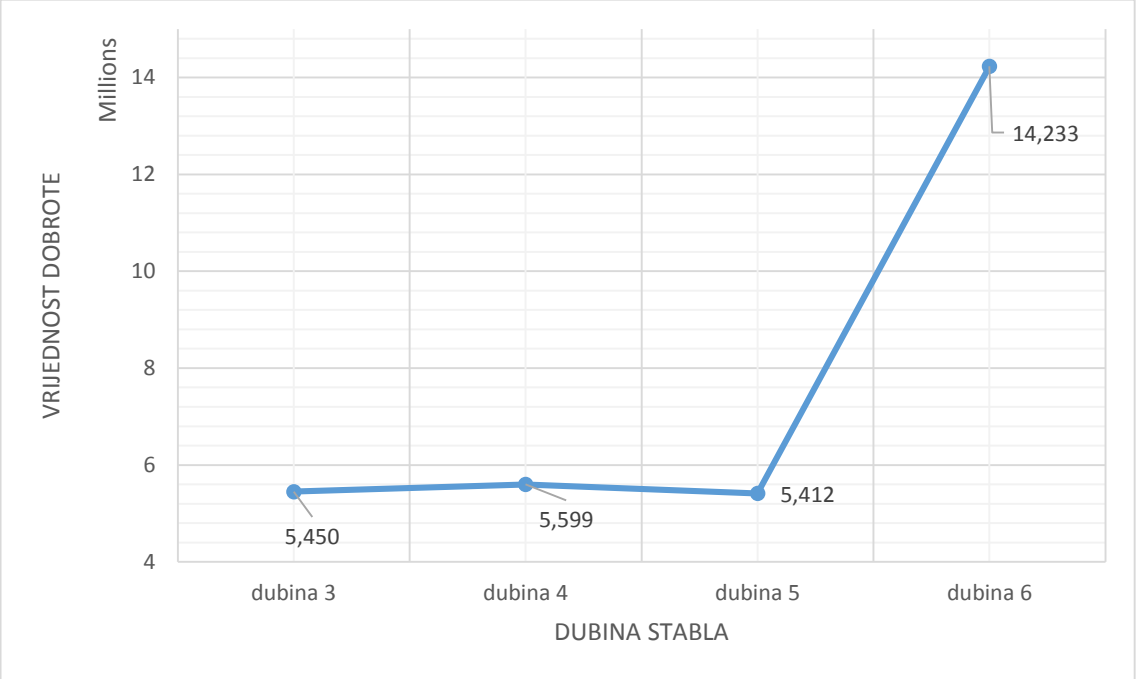


Slika 15: Box plot za različite dubine stabla

Vrijednost dobrote za svaku ispitanu dubinu stabla prikazana je tablicom 6 i na slici 16.

Tablica 6: Utjecaj dubine stabla na konačno rješenje

	DUBINE STABALA			
	Dubina 3	Dubina 4	Dubina 5	Dubina 6
VRIJEDNOST DOBROTE (MEDIJAN)	$5{,}450 * 10^6$	$5{,}599 * 10^6$	$5{,}412 * 10^6$	$14{,}233 * 10^6$



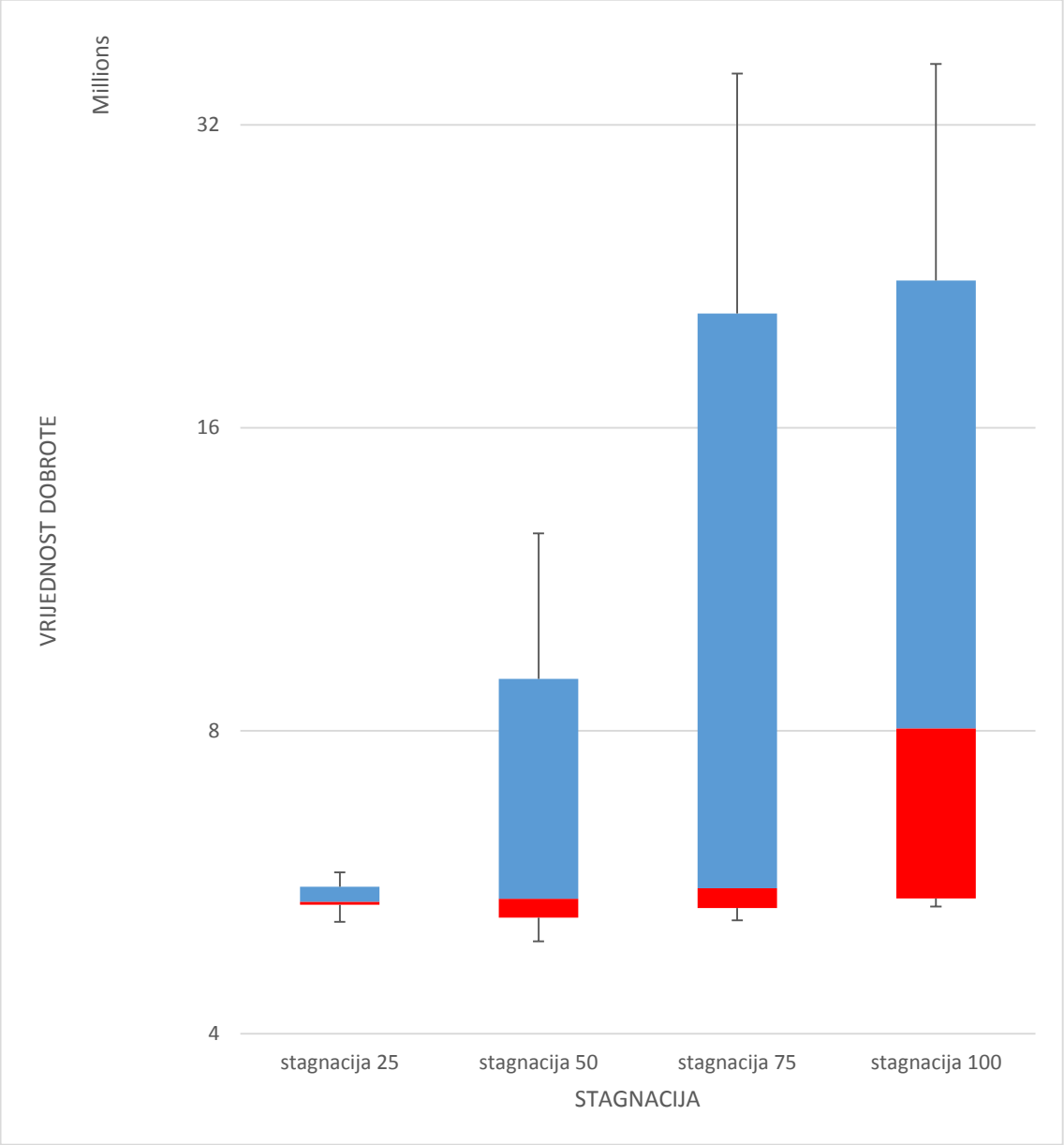
Slika 16: Grafički prikaz vrijednosti dobrote za različite dubina stabala

Vidljivo je da dubina 5 daje najbolje vrijednosti stoga se u nastavku ispitivanja koristi upravo ta dubina stabla. Iako najveći raspon mogućnosti, najlošiji rezultati su bili kad je algoritam imao mogućnost stvoriti stablo dubine 6.

7.2. Utjecaj duljine stagnacije na konačno rješenje

Duljina stagnacije bitan je parametar evolucijskih algoritama jer većom duljinom algoritam ima veće šanse izvući se iz lokalnog minimuma ili maksimuma. S druge strane, veća duljina stagnacije znači i dulje izvođenje algoritma. U ovom radu

ispitano je nekoliko različitih duljina stagnacije. Box plot za različite duljine stagnacije prikazan je na slici 17.

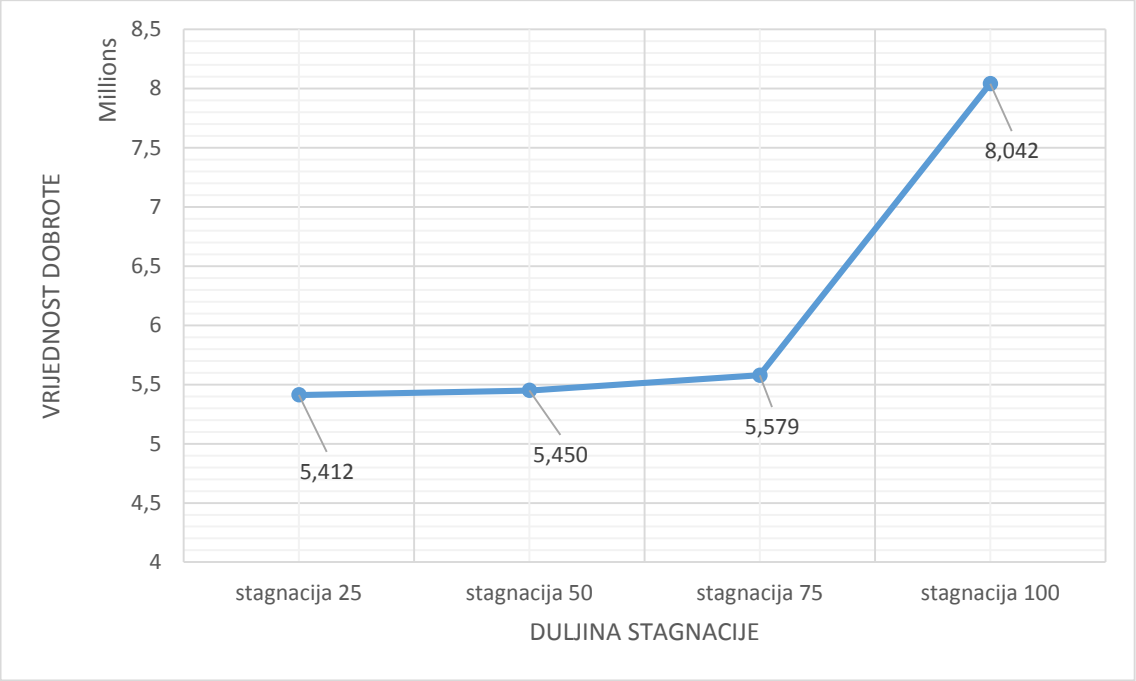


Slika 17: Box plot za različite duljine stagnacije

Vrijednost dobrote kod različitih duljina stagnacije prikazana je tablicom 7, a grafički prikaz slikom 18.

Tablica 7: Utjecaj duljine stagnacije na konačno rješenje

	DULJINA STAGNACIJE			
	Stagnacija 25	Stagnacija 50	Stagnacija 75	Stagnacija 100
VRIJEDNOST DOBROTE (MEDIJAN)	$5{,}412 * 10^6$	$5{,}450 * 10^6$	$5{,}579 * 10^6$	$8{,}042 * 10^6$



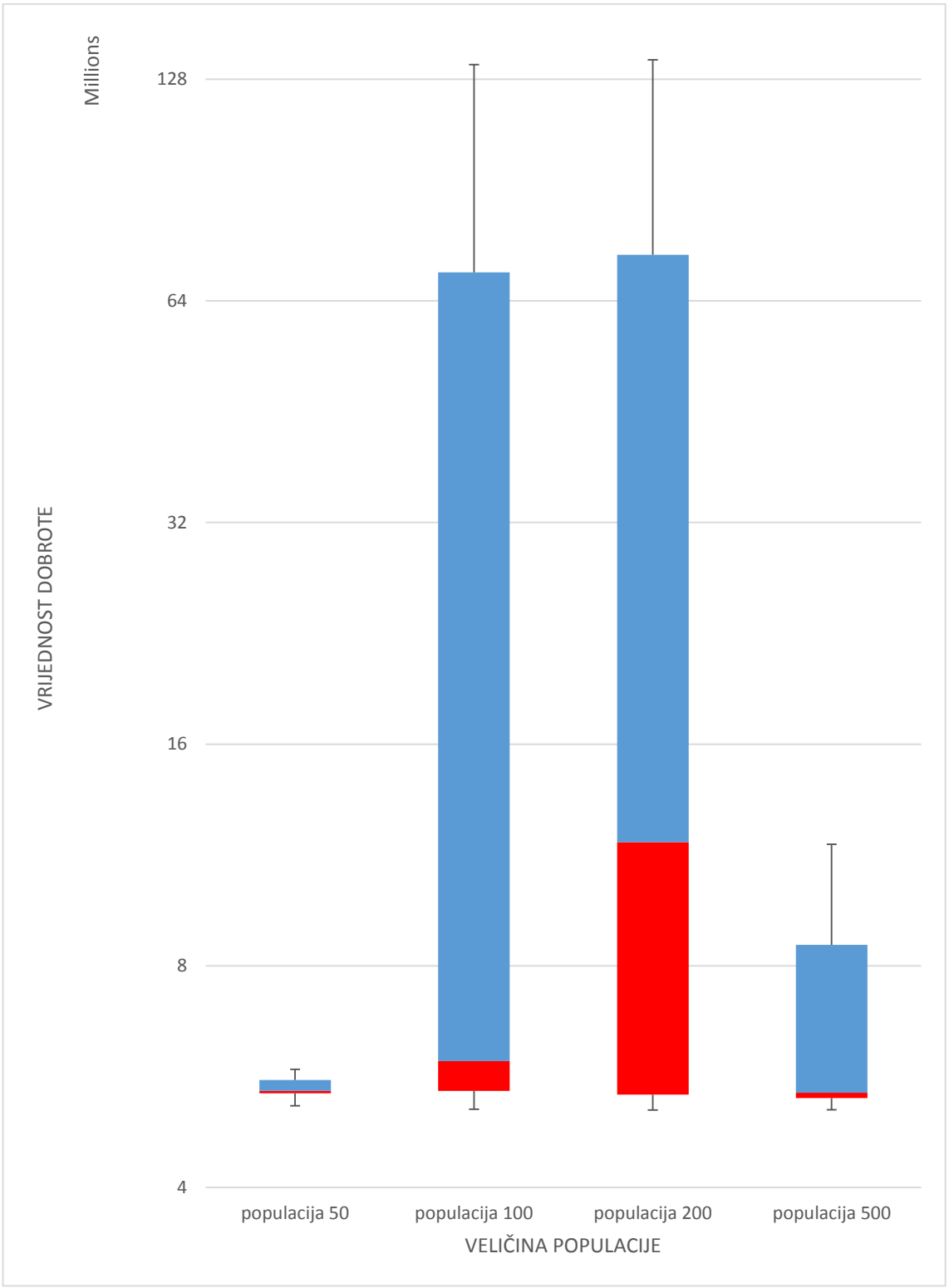
Slika 18: Grafički prikaz vrijednosti dobrote za različite duljine stagnacije

Najbolji rezultati dobiveni su kad je za duljinu stagnacije postavljena vrijednost 25. Kad je bila postavljena vrijednost 100, odnosno, ukoliko se u 100 generacija ne dogodi promjena na bolje, algoritam završava, dobiveni su najlošiji rezultati. Za svaku duljinu algoritam je pokrenut dvadeset puta. U nastavku se koristi duljina stagnacije 25.

7.3. Utjecaj veličine populacije na konačno rješenje

Veličina populacije također je bitan parametar evolucijskih algoritama. Veći broj jedinki unutar populacije znači i veća raznolikost, što je bitno jer rješenje neće odmah na početku konvergirati prema nekoj vrijednosti, nego će imati više prostora

za pretraživanje. U ovom radu ispitano je ponašanje algoritma kod četiri različite veličine populacije, a dijagram je prikazan na slici 19.

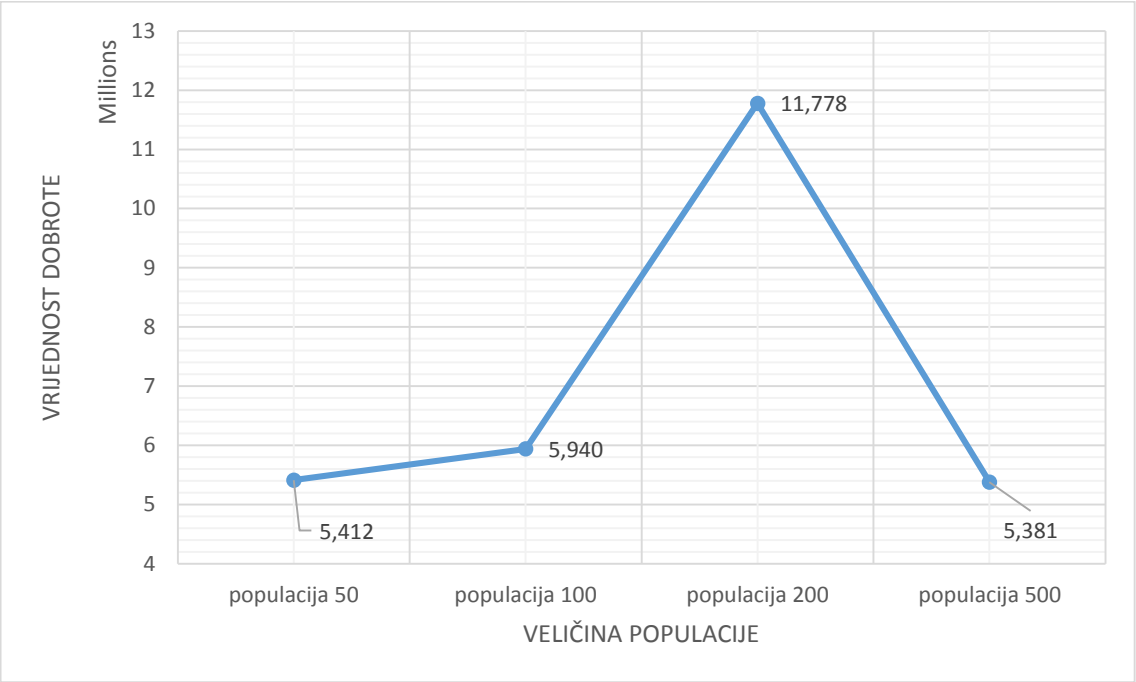


Slika 19: Box plot za različite veličine populacije

Tablica 8 i slika 20 prikazuju vrijednost dobrote za svaku ispitanu veličinu populacije.

Tablica 8: Utjecaj veličine populacije na konačno rješenje

	VELIČINA POPULACIJE			
	Populacija 50	Populacija 100	Populacija 200	Populacija 500
VRIJEDNOST DOBROTE (MEDIJAN)	$5{,}412 * 10^6$	$5{,}940 * 10^6$	$11{,}778 * 10^6$	$5{,}381 * 10^6$



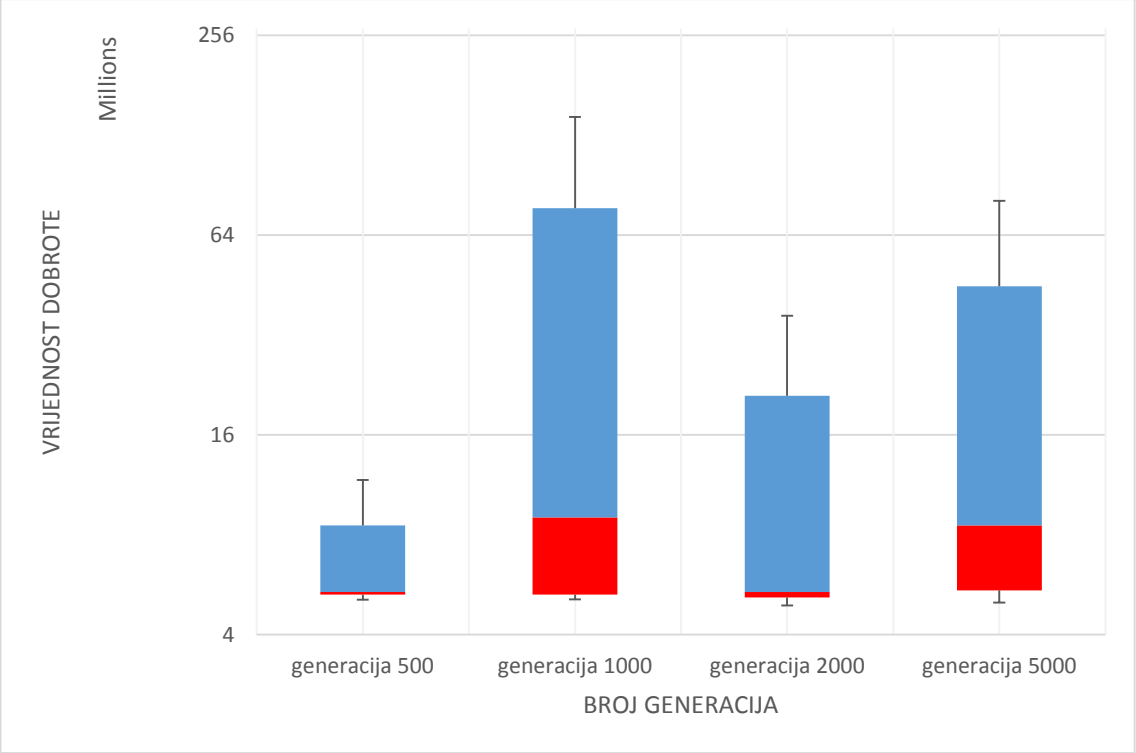
Slika 20: Grafički prikaz vrijednosti dobrote za različite veličine populacije

Najbolji rezultati dobiveni su prilikom korištenja populacije od 500 jedinki. Ovo je logično jer više jedinki unutar populacije znači i veća raznolikost. U nastavku se koristi populacija od 500 jedinki.

7.4. Utjecaj broja generacija na konačno rješenje

Utjecaj broja generacija u pronalasku kvalitetnog rješenja velik je, ali uz uvjet da za duljinu stagnacije i maksimalno vrijeme izvođenja algoritma nisu odabrane male vrijednosti. Većim brojem generacija algoritam više vremena pretražuje prostor

mogućih rješenja čime mu se povećava mogućnost pronalaska najkvalitetnijeg. U ovom radu ispitano je ponašanje algoritma ako je za broj generacija odabrano 500, 1000, 2000 i 5000. Prikaz dobivenih rješenja vidljiv je na slici 21.

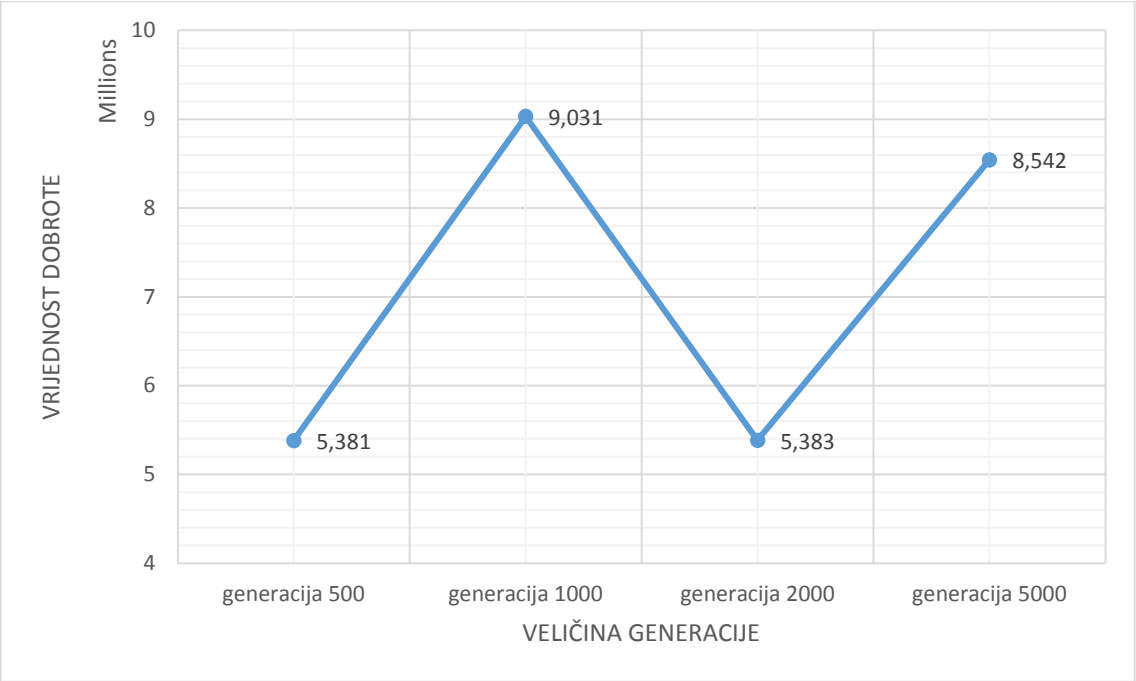


Slika 21: Box plot za različiti broj generacija

Tablica 9 i slika 22 prikazuju vrijednost dobrote za svaku ispitani broj generacija.

Tablica 9: Utjecaj broja generacija na konačno rješenje

	BROJ GENERACIJA			
	Generacija 500	Generacija 1000	Generacija 2000	Generacija 5000
VRIJEDNOST DOBROTE (MEDIJAN)	$5{,}381 * 10^6$	$9{,}031 * 10^6$	$5{,}383 * 10^6$	$8{,}542 * 10^6$

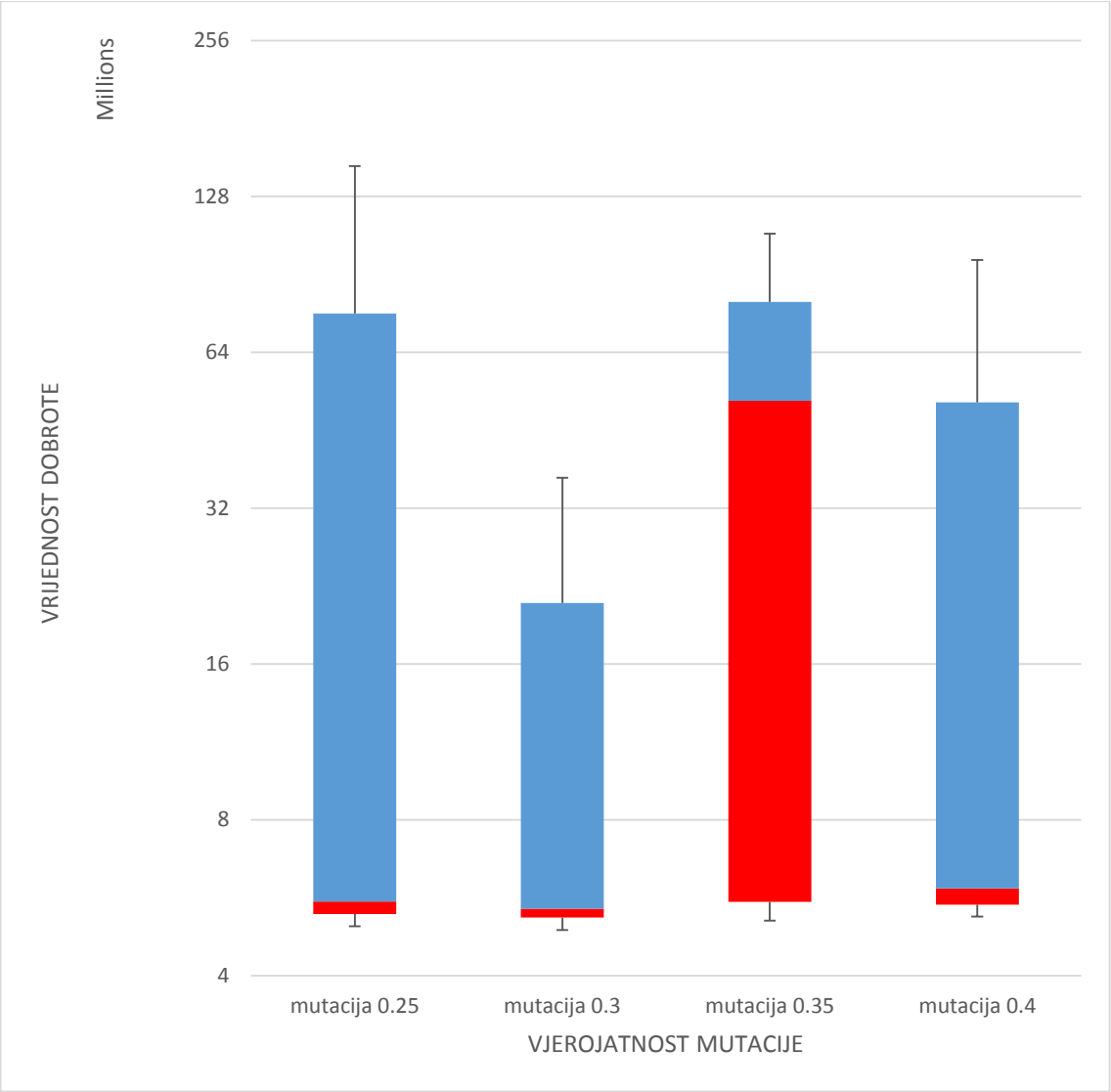


Slika 22: Grafički prikaz vrijednosti dobrote za različiti broj generacija

Iako je medijan najbolji (minimalan) kad je za broj generacija uzet broj 500, autor je odlučio da u nastavku ispitivanja koristi za broj generacija 2000 iz razloga što je razlika vrlo mala, a četiri najbolje vrijednosti dobrote kod broja generacija 2000 su bolje od najbolje vrijednosti dobrote kod broja generacija 500.

7.5. Utjecaj vjerojatnosti mutacije na konačno rješenje

Mutacija je jako bitan segment evolucijskih algoritama. U ovom radu ispitivane su vjerojatnosti mutacije čija vrijednost nije prelazila vrijednost 0,5 jer vjerojatnost mutacije veća od 50% svodi algoritam da nasumično pretražuje čitav prostor pretraživanja. Ispitane vrijednosti vjerojatnosti mutacije prikazane su na slici 23.

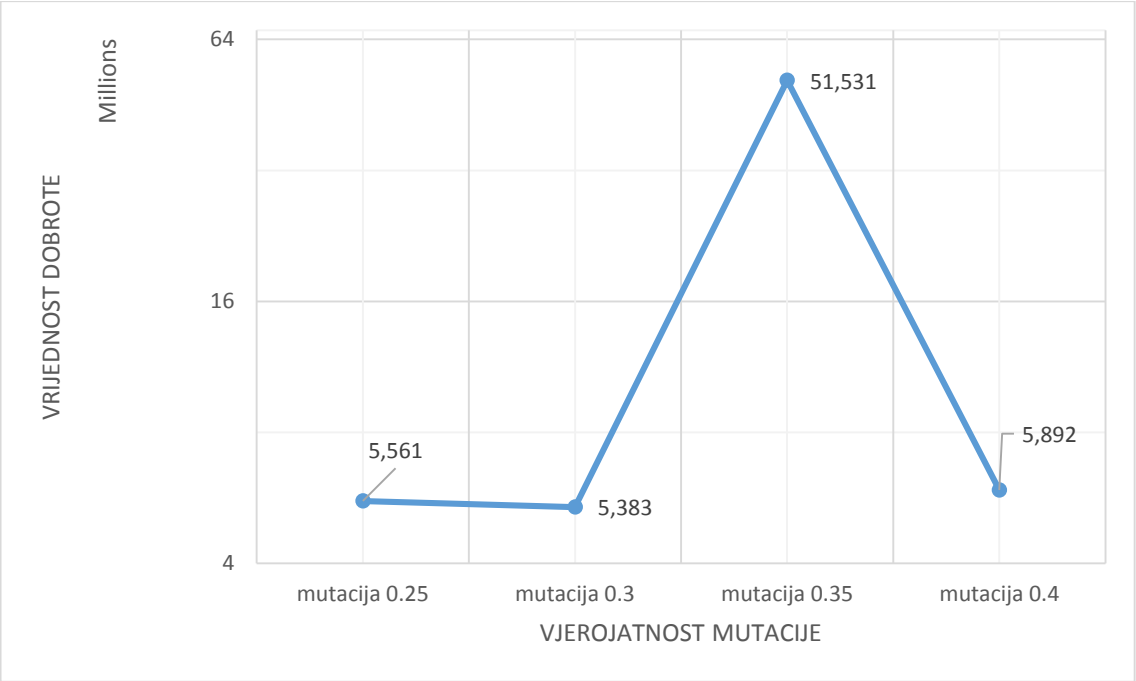


Slika 23: Box plot za različite vjerojatnosti mutacije

Tablica 10 i slika 24 prikazuju vrijednost dobrote za svaku ispitanu vjerojatnost mutacije.

Tablica 10: Utjecaj vjerojatnosti mutacije na konačno rješenje

	VJEROJANOST MUTACIJE			
	Mutacija 0,25	Mutacija 0,30	Mutacija 0,35	Mutacija 0,40
VRIJEDNOST DOBROTE (MEDIJAN)	$5{,}561 * 10^6$	$5{,}383 * 10^6$	$51{,}531 * 10^6$	$5{,}892 * 10^6$

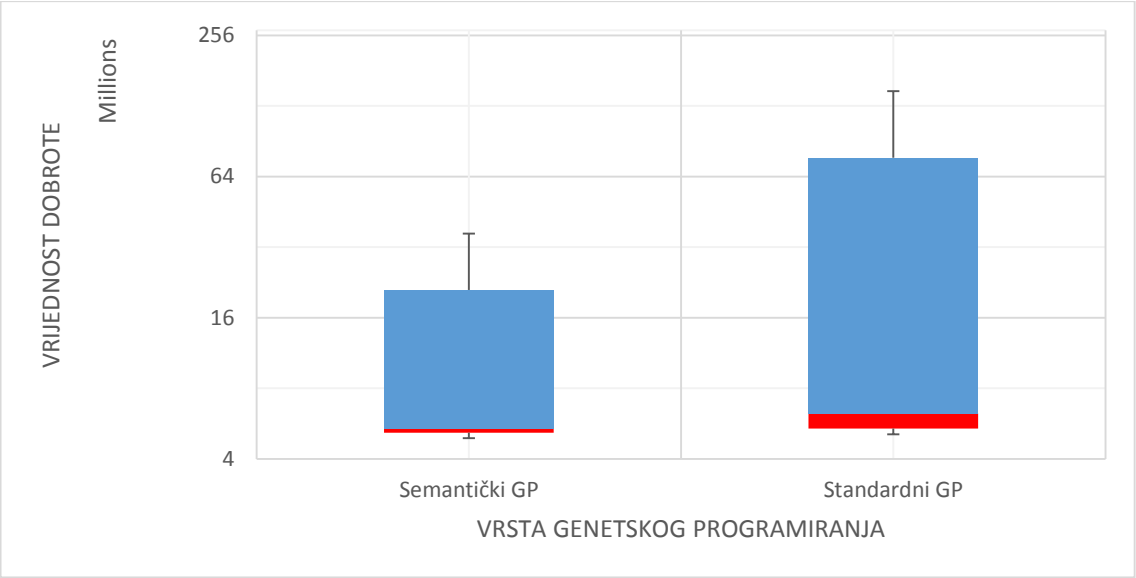


Slika 24: Grafički prikaz vrijednosti dobrote za različite vjerojatnosti mutacije

Vjerojatnost mutacije koja je postavljena na početku ispitivanja daje najbolju vrijednost stoga se ona koristi kod sljedećih ispitivanja. Kod vjerojatnosti mutacije od 0,35 vrijednosti dobrote su iz nepoznatog razloga vrlo visoke.

7.6. Utjecaj optimizacije semantikom na konačno rješenje

Do sad su sva ispitivanja koristila semantičko genetsko programiranje. U ovom poglavlju ispitivanja su pokrenuta koristeći standardno genetsko programiranje. Za operator križanja odabrano je uniformno križanje, a kao mutacija odabrana je nadomještajuća mutacija. Selekcija je turnirsko – eliminacijska uz veličinu turnira od tri jedinke. Rezultati su prikazani na slici 25.

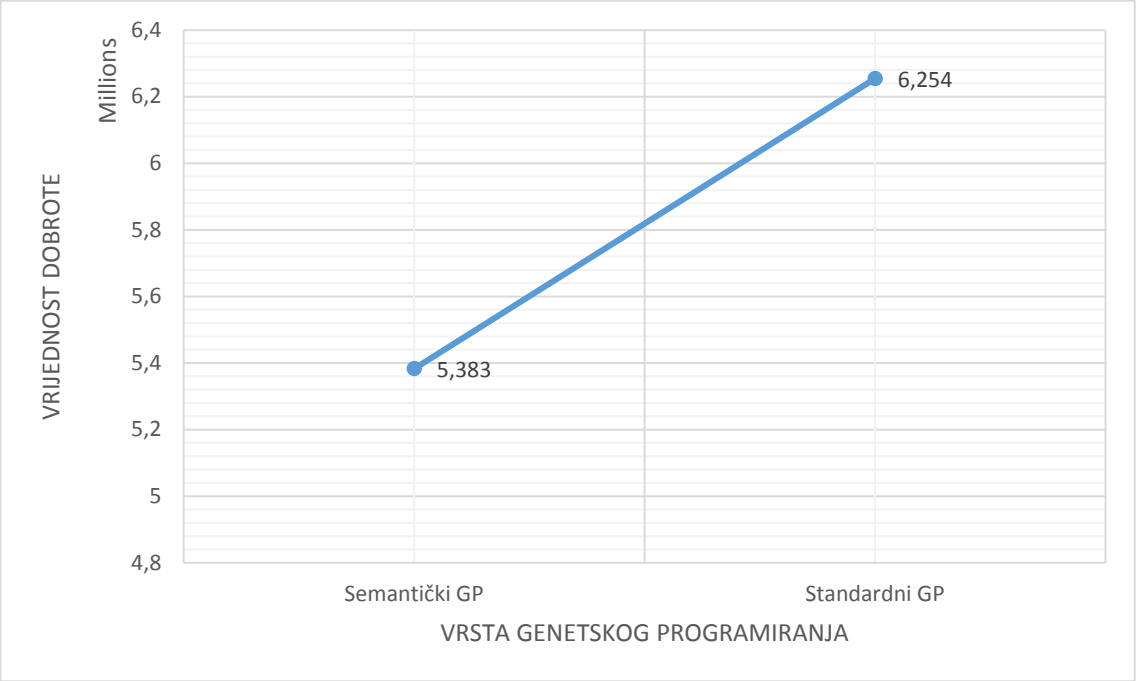


Slika 25: Box plot za usporedbu vrijednosti dobrota dobivenih korištenjem semantičkog i standardnog GP

Tablica 11 i slika 26 prikazuju vrijednost dobrote za svaku ispitanu vjerojatnost mutacije.

Tablica 11: Utjecaj vrste genetskog programiranja na konačno rješenje

	VRSTE GENETSKOG PROGRAMIRANJA	
	Semantički GP	Standardni GP
VRIJEDNOST DOBROTE (MEDIJAN)	$5{,}383 * 10^6$	$6{,}254 * 10^6$

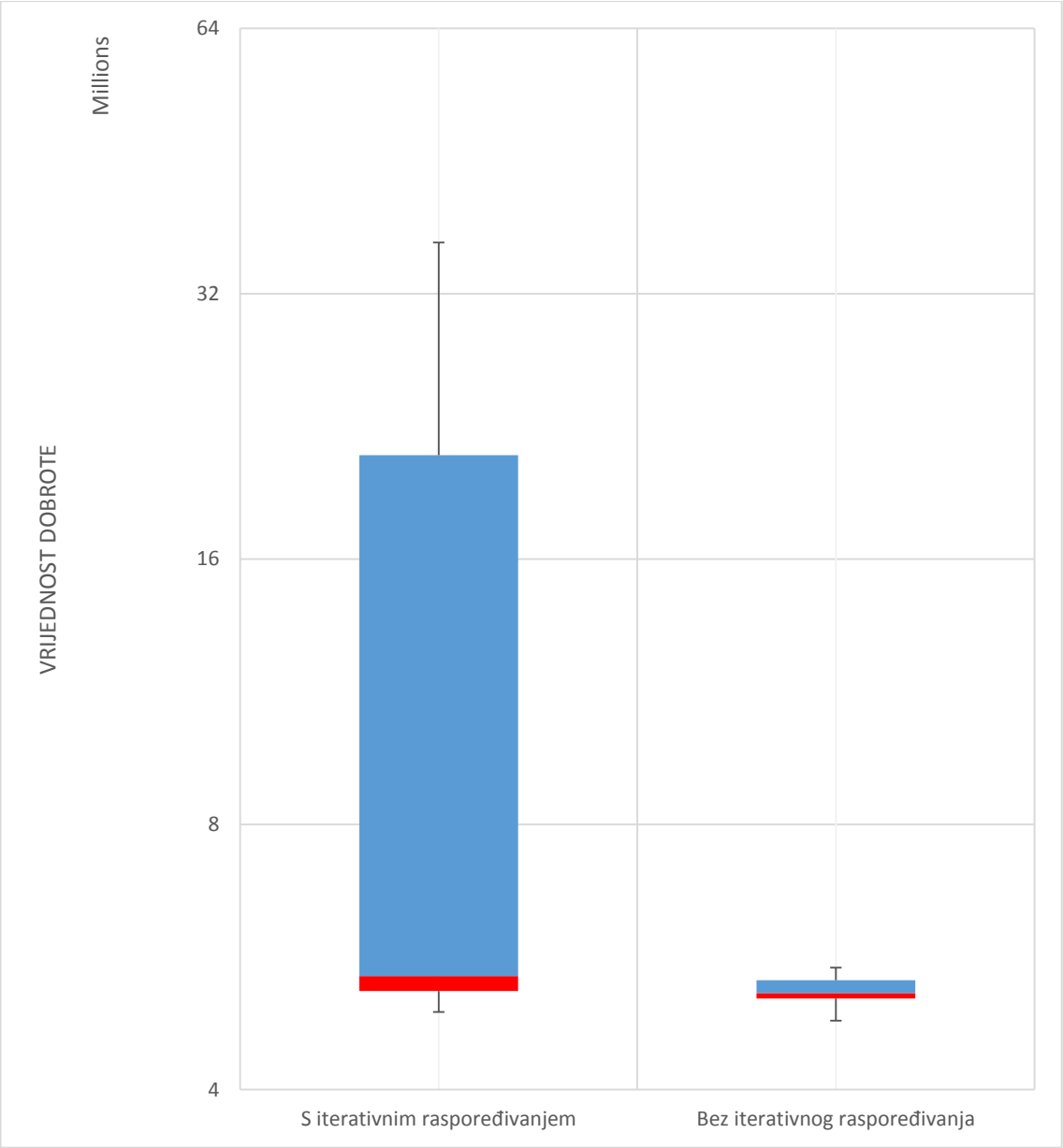


Slika 26: Grafički prikaz vrijednosti dobrote kod različitih vrsta genetskog programiranja

Vrijednosti dobrote dobivene korištenjem semantičkog genetskog programiranja puno su bolje od vrijednosti dobrota dobivenih korištenjem standardnog genetskog programiranja. U nastavku ispitivanja se, kao i do sad, koristi semantičko genetsko programiranje.

7.7. Utjecaj optimizacije iterativnim raspoređivanjem na konačno rješenje

Druga ostvarena optimizacija je iterativno raspoređivanje. U nastavku su prikazani rezultati dobiveni korištenjem iterativnog raspoređivanja i rezultati dobiveni kad se nije koristilo iterativno raspoređivanje. Slika 27 prikazuje vrijednosti dobrota dobivenih nakon što je ispitivanje pokrenuto dvadeset puta.

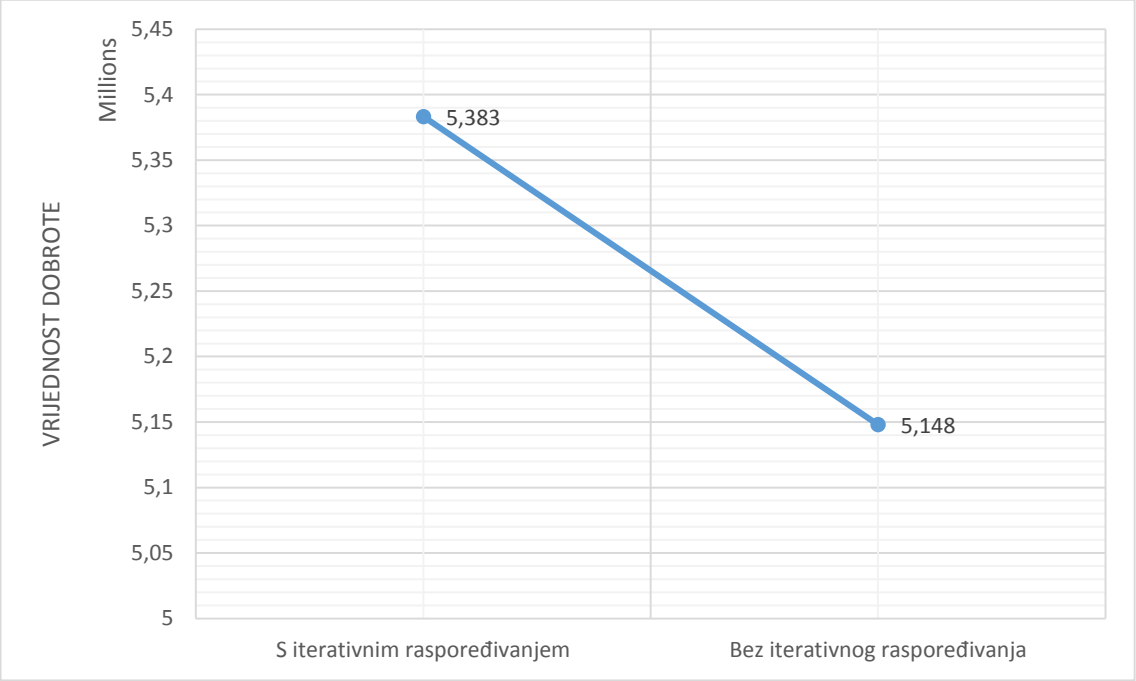


Slika 27: Box plot za usporedbu vrijednosti dobrota dobivenih korištenjem i ne korištenjem iterativnog raspoređivanja

Tablica 12 i slika 28 prikazuju utjecaj optimizacije iterativnim raspoređivanjem na konačno rješenje.

Tablica 12: Utjecaj korištenja optimizacije iterativnim raspoređivanjem

	OPTIMIZACIJA ITERATIVNIM RASPOREĐIVANJEM	
	S iterativnim raspoređivanjem	Bez iterativnog raspoređivanja
VRIJEDNOST DOBROTE (MEDIJAN)	$5{,}383 * 10^6$	$5{,}148 * 10^6$



Slika 28: Grafički prikaz vrijednosti dobrote korištenja i ne korištenja iterativnog raspoređivanja

Iz rezultata je vidljivo da su bolja rješenja dobivena kad se nije koristilo iterativno raspoređivanje. Razlog zašto rješenje dobiveno ovom optimizacijom je lošije od rješenja koje se dobije kad se ne koristi iterativno raspoređivanje može biti nekvalitetan odabir novih informacija.

8. Zaključak

Nakon obavljenih ispitivanja i proučenih rezultata, vidljivo je da neki parametri značajno, a neki manje značajno, utječu na konačno rješenje. Najkvalitetnije rješenje dobiveno je korištenjem dubine stabla 5, duljine stagnacije od 25 generacija, veličine populacije od 500 jedinki, 2000 generacija, vjerojatnosti mutacije od 0,3 i korištenjem semantičkog genetskog programiranja.

Ispitivanja su izvođena koristeći jedan ispitni primjer za učenje algoritma i jedan ispitni primjer za ocjenu učinkovitosti algoritma. Ispitivala se najsloženija od tri ostvarene inačice, DCVRPTW, odnosno dinamički kapacitivni problem raspoređivanja vozila s vremenskim okvirima.

Iako trenutno ne postoje globalno priznati ispitni primjeri za dinamičke probleme raspoređivanja vozila, iz rezultata dobivenih u ovom radu vidljivo je da postoji prostor za napredak. Osim standardnih ispitivanja utjecaja parametara genetskog programiranja na konačno rješenje, moguće je uvesti nove informacije kod iterativnog raspoređivanja, ostvariti nove načine optimizacije, ispitati sustav koristeći neku drugu funkciju cilja, ispitati ponašanje ako se promjeni minimalna količina dobara koju vozilo smije imati prije nego što mora u skladište, i slično.

Sustav za demonstraciju ostvaren u ovom radu omogućuje ugradnju i ispitivanje različitih inačica problema i različitih funkcija ciljeva na jednostavan način. S obzirom da je genetsko programiranje korišteno samo prilikom odluke raspoređivača kojem vozilu će dodijeliti zahtjev, na jednostavan način može se ugraditi neki drugi algoritam.

Kao što je vidljivo, problem raspoređivanja vozila vrlo je težak problem s vrlo velikim prostorom za pretraživanje. Koristeći semantičko genetsko programiranje kao način optimizacije postignuti su kvalitetniji rezultati od rezultata postignutih koristeći standardno genetsko programiranje. Bez obzira na to, svakako postoji još mjesta za napredak isprobavanjem nekih drugih optimizacijskih tehnika.

9. Literatura

[1] http://en.wikipedia.org/wiki/Genetic_programming

[2] „*Optimizacija raspoređivanja u okruženju nesrodnih strojeva*“, Marko Đurasević, Fakultet elektrotehnike i računarstva, 2014

[3] http://www.bernabe.dorronsoro.es/vrp/ Homberger instances

[4] V. Pillac, M. Gendreau, C. Guéret, A. L. Medaglia. „A Review of Dynamic Vehicle Problems“

[5] http://gp.zemris.fer.hr/ecf/algorithms.html

[6] *R. Poli, W.B. Langdon, N.F. McPhee, i Koza J.R. „A Field Guide to Genetic Programming“, http://www0.cs.ucl.ac.uk/staff/wlangdon/ftp/papers/poli08_fieldguide.pdf, 2008*

[7] *„The Truck Dispatching Problem“*, G. B. Dantzig, J. H. Ramser, 1959

[8] *„Planned Route Optimization For Real-Time Vehicle Routing“,* Soumia Ichoua, Michel Gendreau, Jean-Yves Potvin, 2007

[9] *„Vehicle routing problems with varying degrees of dynamism“*, K. Lund, Oli B. G. Madsen, J. Rygaard, 1996

[10] *„The Dynamic Vehicle Routing Problem“*, A. Larsen, 2001

Raspoređivanje vozila u dinamički uvjetima rada

Sažetak

U ovom radu opisan je problem raspoređivanja vozila. Opisan je algoritam genetskog programiranja te sustav koji demonstrira učinkovitost algoritma genetskog programiranja u pronalaženju kvalitetnog rješenja u dinamičkim uvjetima i proizvoljnom kriteriju optimizacije. Opisane su ostvarene inačice problema raspoređivanja vozila te optimizacijske tehnike koje su korištene s ciljem dobivanja kvalitetnijeg rješenja od strane genetskog programiranja. Prikazani su i komentirani rezultati koji su dobiveni koristeći različite parametre genetskog programiranja i optimizacija.

Ključne riječi: problem raspoređivanja vozila, genetsko programiranje, iterativna pravila raspoređivanja, semantičko genetsko programiranje

Vehicle routing in dynamic environments

Abstract

In this thesis the vehicle routing problem is described. The genetic programming algorithm as well as system that demonstrates effectiveness of genetic programming algorithm in finding quality solution in dynamic conditions and using arbitrary optimization criterion are also described. Implemented versions of vehicle routing problem and optimization techniques used in order to obtain quality solutions from genetic programming side are also described. Solutions gotten from using different genetic programming parameters and optimizations are shown and commented.

Keywords: vehicle routing problem, genetic programming, iterative dispatching rules, semantic genetic programming