SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 356

RJEŠAVANJE PROBLEMA TRGOVAČKOG PUTNIKA UZ POMOĆ GENETSKOG ALGORITMA

Ivana Dasović

Zagreb, lipanj 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 356

RJEŠAVANJE PROBLEMA TRGOVAČKOG PUTNIKA UZ POMOĆ GENETSKOG ALGORITMA

Ivana Dasović

Zagreb, lipanj 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 11. ožujka 2022.

ZAVRŠNI ZADATAK br. 356

Pristupnica:	**Ivana Dasović (0036522719)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentor:	prof. dr. sc. Marin Golub

Zadatak: **Rješavanje problema trgovačkog putnika uz pomoć genetskog algoritma**

Opis zadatka:

Definirati problem trgovačkog putnika (engl. Traveling Salesman Problem, TSP). Opisati genetski algoritam pogodan za rješavanje problema trgovačkog putnika. Posebnu pažnju posvetiti vrsti prikaza rješenja. Ostvariti programski sustav koji genetskim algoritmom rješava problem trgovačkog putnika i ispitati rad algoritma na primjerima slobodno dostupnih problema trgovačkog putnika s najboljim poznatim rješenjima. Radu priložiti algoritme, izvorne tekstove programa i rezultate uz potrebna objašnjenja te citirati korištenu literaturu.

Rok za predaju rada: 10. lipnja 2022.

Zahvaljujem mentoru, prof. dr. sc. Marinu Golubu, na pruženoj pomoći i savjetima tijekom izrade ovog rada.

Sadržaj

1. Uvod

Tema ovog završnog rada je rješavanje problema trgovačkog putnika uz pomoć genetskog algoritma. Problem trgovačkog putnika jako je popularan i poznat problem u algoritamskoj teoriji brojeva. Rješenje ovog problema ima široku praktičnu primjernu jer se na njega svodi puno drugih problema. Potencijalno rješenje problema trgovačkog putnika daju genetski algoritmi. Genetski algoritmi mogu pronaći rješenja dovoljno dobra za primjenu koja samo nekoliko posto odstupaju od optimalnog rješenja. Posljednjih godina obilježava se značajan razvoj genetskih algoritama. Ideja genetskih algoritama dolazi od procesa biološke evolucije te se temelji na odabiru roditelja, križanju, mutacijama i funkciji dobrote koja računa prilagodljivost jedinke okolini.

U nastavku je dan uvod u problem trgovačkog putnika i njegove varijante te pojašnjenje genetskih algoritama i njihovih operatora.

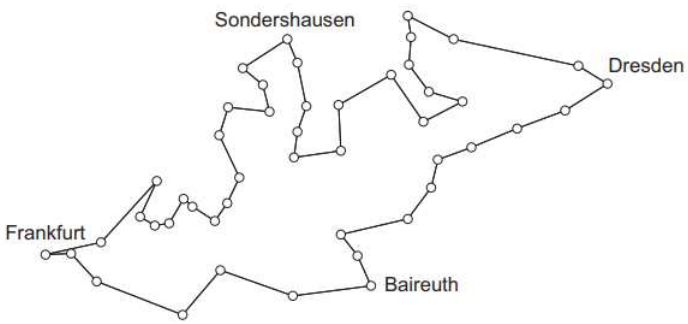
Za potrebe ovog završnog rada napisan je program u programskom jeziku Python koji na temelju ulazne matrice udaljenosti među gradovima traži optimalni put i računa njegovu duljinu. U 4. poglavlju nalazi se opis navedenog programskog rješenja problema trgovačkog putnika uz pomoć eliminacijskog genetskog algoritma. U konačnici, u 5. poglavlju nalaze se eksperimentalni rezultati dobiveni programskim rješenjem.

2. Problem trgovačkog putnika

2.1. Povijest i ideja problema trgovačkog putnika

Problem trgovačkog putnika (*eng. Travelling Salesman Problem, TSP*) interpretacija je problema pronalaska ciklusa minimalne težine u težinskom grafu koji sadrži sve njegove vrhove, tj. pronalaženje Hamiltonovog ciklusa najmanje duljine. Problem trgovačkog putnika je, dakle, minimizacija ukupne udaljenostu koju trgovački putnik treba prijeći krećući se iz jednog grada u drugi, a da pritom posjeti svaki grad točno jednom i vrati se u početni grad. Iako se problem ne čini toliko kompleksnim, on ulazi u probleme diskretne i kombinatorne optimizacije[1]. Pripada kategoriji NP- teških algoritama te ima složenost O(n!)[10]. Rješenje problema ima široku praktičnu primjenu u logistici, planiranju, proizvodnji mikročipova, problemu raspoređivanja različitih vrsta rasporeda i dizajniranju globalnih navigacijskih satelitskih sustava. Upravo zbog velike kompleksnosti i široke primjene problem trgovačkog putnika vrlo je zanimljiv znanstvenicima iz različitih područja.

Nepoznato je tko je i kada definirao problem trgovačkog putnika. Taj problem bio je poznat ljudima puno prije nego li je postao popularan matematički problem. Oblik matematičkog problema trgovačkog putnika proučavao je i Euler koji je na primjeru skakača na šahovskoj ploči razmatrao kako bi skakač mogao posjetiti sva 64 polja samo jednom. Prvo poznato spominjanje problema trgovačkog putnika je u priručniku njemačkog trgovačkog putnika Commis-Voyageur iz 1832. godine u kojemu je problem detaljno opisan riječima, ali ne i matematički. Priložio je 5 ruta kroz Njemačku i Švicarsku, no 4 rute su ipak sadržavale povratak u neki već posjećeni grad.



Slika 2.1 Cammis- Voyageur, 1832. [2]

Karl Menger je počeo proučavati opći oblik problema trgovačkog putnika te ga je 1930. godine i matematički opisao, a Hassler Whitney je prvi uveo pojam „problem trgovačkog putnika“. Tijekom 1940-ih problem su proučavali brojni matematičari te 1950-ih i 1960-ih problem je postajao sve popularniji među europskim i američkim znanstvenicima. 1954. godine Dantzig, Fulkerson i Johnson objavili su opis metode za rješavanje problema i dali primjer od 49 instanci. Popularnost problema nastavljala je rasti i u sljedećim godinama te su se problemom počeli baviti istraživači iz različitih područja znanosti poput matematike, fizike, kemije i računarstva.

2.2. Vrste problema trgovačkog putnika

Postoji više vrsta problema trgovačkog putnika. Osnovne varijance problema trgovačkog putnika su:

1. Simetrični problem trgovačkog putnika koji kao model koristi neusmjereni težinski graf, tj. cijena puta između dva grada A i B bez obzira na smjer obilaska uvijek je ista.
2. Asimetrični problem trgovačkog putnika koji kao model koristi težinski graf za koji vrijedi da duljina puta ima različite vrijednosti ovisno o smjeru obilaska. Primjerice duljina puta od grada A do grada B je manja od duljine puta od grada B do grada A.
3. Višestruki model trgovačkog putnika koji opisuje problem m trgovačkim putnika koji kreću iz određenog grada te moraju obići sve gradove točno jednom, a da ukupni prijeđeni put bude minimalan.

2.3. Metode rješavanja problema trgovačkog putnika

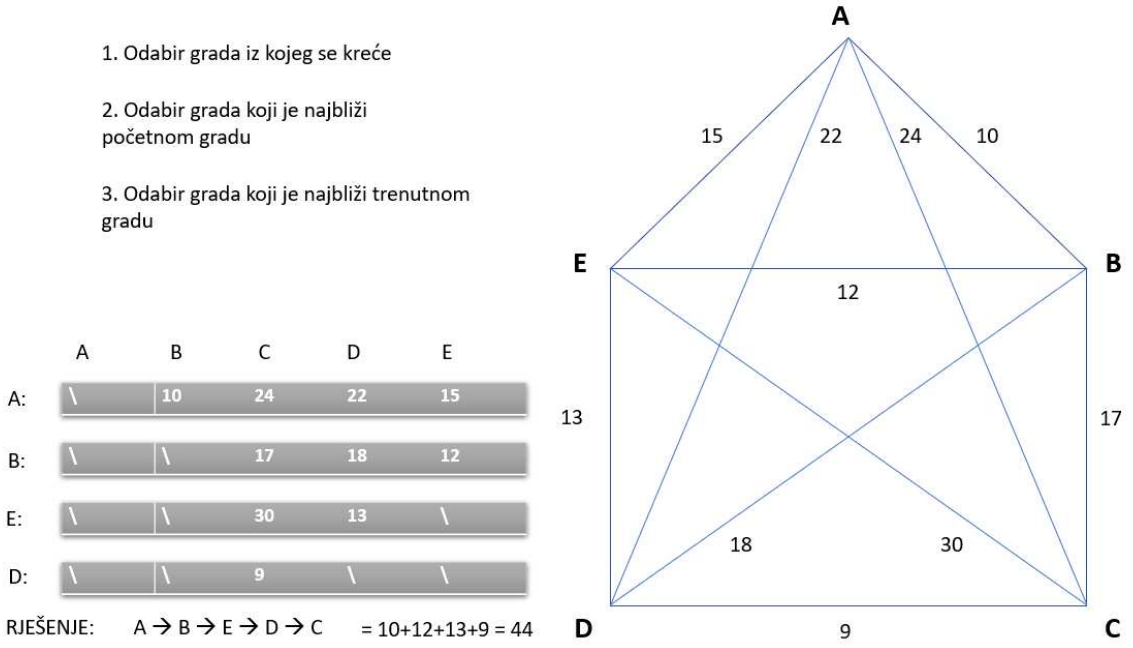
Postoji više metoda rješavanja ovog problema. Prva metoda je iscrpno pretraživanje koje obuhvaća pronalazak svih mogućih rješenja što podrazumijeva obilazak svih puteva i odabir najboljeg. Ova metoda je najprimitivniji i najsporiji način pronalaska najboljeg puta zbog faktorijalne složenosti problema pa se ovo rješenje može iskoristiti samo za relativno mali broj gradova [13].

Druga metoda podrazumijeva korištenje egzaktnih i aproksimacijskih algoritama. Egzaktni algoritmi će uvijek dati optimalno rješenje, no zbog velike složenosti samih algoritama oni nisu primjenjivi na veće primjerke NP- teških problema poput problema trgovačkog putnika. Primjeri egzaktnih algoritama su:

1. Algoritam Branch and bound koji za traženje rješenja koristi stablo čime se prostor rješenja razdvaja na manje fragmente, tj. grane. Na početku se definiraju gornja i donja granica te se zatim grana uspoređuje sa svakom od granica. Ako se u određenoj grani ne može dohvatiti rješenje bolje od trenutnog najboljeg pretraživanje te grane se tada ne nastavlja. Na svakoj razini odlučuje se koji čvor se uzima u set rješenja te se pretražuju djeca tog čvora.
2. Algoritam Cut and branch koristi Branch and bound algoritam i podrezivanje kojim se dodaju ograničenja na prostor rješenja čime se više grana odbacuje i ne pretražuje.

Aproksimacijski algoritmi nazivaju se još i heurističke metode, tj. heuristike. To su algoritmi koji ne pronalaze uvijek optimalno rješenje, ali pronalaze ono „dovoljno dobro“, odnosno zadovoljavajuće, te nemaju veliku računarsku složenost. Heuristička funkcija svakom stanju dodjeljuje procjenjenu udaljenost od konačnog stanja. Na temelju heurističke funkcije odabire se smjer daljnjeg pretraživanja prostora stanja. Optimistična heuristika nikad ne daje veću vrijednost od stvarne udaljenosti do ciljnog stanja. Ako heuristička funkcija nije optimistična konačno rješenje ne mora biti optimalno rješenje. No, upravo zato što nije uvijek potrebno optimalno rješenje nije bitno je li heuristika optimistična. Korištenjem neoptimistične heuristike se smanjuje broj generiranih stanja koja vode ka rješenju te se radi kompromis između kvalitete rješenja i složenosti pretraživanja. Aproksimacijski algoritmi dijele se na konstrukcijske algoritme i algoritme lokalne pretrage. Konstrukcijski algoritmi grade rješenje postupno: dio po dio. Primjeri konstrukcijskih algoritama su:

1. Algoritam najbližeg susjeda koji obuhvaća pretragu koja započinje u određenom gradu te se za sljedeći grad bira onaj do kojega je udaljenost najmanja a da još nije bio posjećen. Obzirom na to da se ne gleda do tad ukupna cijena puta algoritam najbližeg susjeda ne daje optimalno rješenje.



Slika 2.2 Ilustracija algoritma najbližeg susjeda

2. Algoritam umetanja najbližeg koji podrazumijeva pretragu koja započinje određenim gradom pa se ruta proširi njemu najbližim gradom te se sljedeći grad bira tako da se odabere onaj do kojeg je udaljenost najmanja od jednog od ta dva grada, ubaci se u rutu i pretraga se nastavlja.
3. Algoritam umetanja najudaljenijeg koji obuhvaća umetanje najviše udaljenog grada od skupa gradova koji se već nalaze u listi (na početku je u listi samo početni grad) te se on dodaje na optimalno mjesto u listi.

Algoritmi lokalnog pretraživanja traže rješenja počevši od nekog početnog rješenja, a zatim traže poboljšanja tog rješenja. Jednostavnim izmjenama početnog rješenja dobivaju se susjedna rješenja te se najbolje susjedno rješenje može odabrati za sljedeće početno rješenje. Primjer algoritma lokalne pretrage je algoritam uspona na vrh koji pripada algoritmima pohlepne heuristike. Algoritam za odabir sljedećeg stanja uvijek uzima ono stanje koje se trenutno čini najboljim ne uzimajući u obzir do tad ukupni prijeđeni put čime se znatno smanjuje prostorna složenost. Ovaj algoritam nije potpun što znači da i ako postoji rješenje algoritam ga možda neće pronaći stoga on nije ni optimalan. No, ovaj algoritam ima relativno malu vremensku složenost i minimalnu prostornu složenost što ga čini bržim i jednostavnijim za izvođenje.

Zbog složenosti problema trgovačkog putnika tradicionalne metode nisu dovoljno dobre. S razvojem računarstva i poboljšanjem računalne procesorske snage, ljudi su počeli rješavati izuzetno složene probleme, kao što je problem trgovačkog putnika, koristeći različite programski ostvarene algoritme. Unatoč stalnom rastu performansa računala, potrebno je tražiti bolje algoritme za rješavanje problema pošto se i dalje ne mogu efikasno riješiti problemi poput problema trgovačkog putnika[12]. U teoriji, do danas nije pronađen algoritam za pronalazak idealnog rješenja ovog problema, no postoje algoritmi koji dovode do zadovoljavajućeg rješenja. Unatoč tome što ni genetski algoritmi ne pronalaze uvijek optimalno rješenje, njihovo korištenje pri pronalasku rješenja problema trgovačkog putnika nije rjetkost. Odstupanje od optimalnog rješenje pri uporabi genetskih algoritama je svega nekoliko posto što daje prihvatljivo rješenje.

3. Genetski algoritmi

3.1. Povijest i ideja genetskih algoritama

Genetski algoritmi su vrsta evolucijskih algoritama zasnovanih na Darwinovoj teoriji evolucije. Ova vrsta algoritama oslanja se na prirodnu selekciju. Evolucijski algoritmi također obuhvaćaju evolucijske strategije, genetsko programiranje i evolucijsko programiranje.

Evolucijski algoritmi pojavili su se 50-ih godina prošlog stoljeća te se razvijaju i dalje.

Genetski algoritmi kao vrsta evolucijskih algoritama su robusniji i najčešće se koriste za aproksimaciju funkcija i kao optimizacijske metode, ali mogu se primijeniti i na mnoge druge probleme[15].

Američki znanstvenik Johna H. Holland prvi je 1970-ih iznio temelje genetskih algoritama, a 1975. godine objavio je knjigu „Adaptation in Natural and Artificial Systems“ u kojoj je opisao te temelje.

Osnova genetskih algoritama su metode selekcije, rekombinacije, tj. križanja, i mutacije čime se oponaša prirodni tijek biološke evolucije. Prirodna selekcija je dio procesa evolucije koji vodi ka boljoj prilagođenosti populacije okolini. Prirodnom selekcijom eliminiraju se jedinke koje su slabije prilagođene uvjetima života zbog čega se nove generacije razvijaju iz jedinki bolje prilagođenih uvjetima. Osnovna ideja genetskih algoritama je simulacija procesa evolucije i njena primjena na rješavanje raznih optimizacijskih problema. U biološkoj evoluciji postoji generacija (naraštaj) čije jedinke mogu stvarati potomke razmnožavanjem pri čemu nove jedinke nasljeđuju karakteristike svojih roditelja. Kod jedinke populacije može doći do mutacija. Mutacije su slučajne promjene značajki do kojih dolazi kod određenog postotka gena. Zbog mutacija potomci mogu imati neke karakteristike koje njihovi roditelji nisu imali. Sve jedinke populacije ne mogu preživjeti, već preživljavaju samo one bolje prilagođene okolini pa se značajke „jačih“ jedinki prenose u sljedeće generacije, tj. ostaju dijelom populacije, koje time generalno postaju sve bolje prilagođene okolini.

Značajke genetskih algoritama su: prilagodljivost jedinki okolini u kojoj se populacija nalazi, robusnost, oponašanje postupaka iz prirode, široko područje primjene te mogućnost dobivanja dobrih približnih rješenja[15].

Jedinka	Moguće rješenje problema
Populacija	Skup svih mogućih rješenja problema
Gen	Parametar (element matematičkog objekta kromosoma)
Kromosom	Matematička reprezentacija jedinke
Razmnožavanje i mutacija	Matematički operatori pretraživanja prostora mogućih rješenja (roditelj- rješenje na temelju kojega se stvaraju nova rješenja (djeca))
Prikladnost	Kvaliteta rješenja problema
Prirodna selekcija	Uklanjanje loših rješenja problema višestrukim korištenjem dobrih mogućih rješenja problema

Tablica 3.1 Usporedba elemenata prirodne evolucije i evolucijskih algoritama

Prednosti korištenja genetskih algoritama u rješavanju problema su širok prostor rješenja i paralelizam u smislu da se više iteracija genetskog algoritma može istovremeno odvijati na više jezgri istog procesora ili na više procesora. No, genetski algoritmi imaju poteškoće poput ograničenja u odabiru parametara i identifikacije funkcije kvalitete rješenja. Osnovne parametre genetskog algoritma čine veličina populacije, vjerojatnost križanja i vjerojatnost mutacije te broj iteracija. Veličina populacije uglavnom se unaprijed određuje te svaka nova generacija ili izmijenjena populacija ima isti broj jedinki kao i početna populacija. Veći broj jedinki ne znači nužno i bolje konačno rješenje, ali broj jedinki populacije može utjecati na kvalitetu konačnog rješenja stoga se on određuje ovisno o konkretnom problemu koji se rješava genetskim algoritmom. Premali broj jedinki znači i manji prostor potencijalnih rješenja te potencijalno manji broj reprodukcija i mutacija koje inače vode ka većem prostoru potencijalnih rješenja. Međutim, preveliki broj jedinki u populaciji usporava izvođenje algoritma, a ponekad ni ne vodi ka nekim novim rješenjima. Vjerojatnost križanja određuje koji postotak jedinki populacije će nastati križanjem. Ako je kod generacijskog genetskog algoritma vjerojatnost križanja jednaka 1, populacija nastaje samo križanjem. Ovaj problem

rješava se uvođenjem elitizma. Elitizmom se najbolja rješenja prenose u sljedeću generaciju. No, ako je postotak križanja i mutacija premalen, premalen je i prostor potencijalnih rješenja koji se pretražuje te se velik broj jedinki prenosi iz prethodne generacije zbog čega izmijenjene populacije ili nove generacije puno sporije vode ka boljim rješenjima. Vjerojatnost mutacije određuje koji će postotak gena jedinki nastalih križanjem mutirati. Mutacijom nastaju nove jedinke koje se djelomično razlikuju od stare jedinke stoga je mutacija u određenom postotku gena povoljna jer povećava broj potencijalnih rješenja problema te može voditi ka boljem rješenju. Ako se vjerojatnost mutacije postavi na 0, sve novonastale jedinke nastaju samo križanjem. Vjerojatnost mutacije koja je jednaka broju 1 rezultira nasumičnim stvaranjem novih jedinki te križanje ne dolazi do izražaja, a algoritam se pretvara u algoritam slučajnog pretraživanja. Vjerojatnost mutacije i vjerojatnost križanja također se uobičajeno unaprijed određuju. Iznimka je adaptivni genetski algoritam koji tijekom rada algoritma mijenja vrijednosti parametara. Prema unaprijed zadanim formulama određuju se vjerojatnost mutacije i vjerojatnost križanja ovisno o promjenama u populaciji.

3.2. Operatori genetskih algoritama

3.2.1. Funkcija kvalitete rješenja

Funkcija kvalitete rješenja, takozvana fitness funkcija ili funkcija dobrote, je funkcija čiji je ulaz jedinka, a izlaz broj koji opisuje koliko je dobro jedinka prilagođena okolini. Funkcija dobrote mora kvantitativno dati odgovor na pitanje „Koliko je dobro ovo rješenje sustava?“. Evaluacijom rješenja se daje ocjena kvalitete jedinki te se na temelju evaluacije biraju jedinke za razmnožavanje ili jedinke za eliminaciju. Funkcija kvalitete rješenja zaslužna je za poboljšanje sljedeće generacije jedinki. Što bolje funkcija kvalitete rješenja evaluira prilagođenost jedinke okolini, to algoritam vodi ka boljim rješenjima.

3.2.2. Odabir roditelja

Odabir roditelja vrlo je važan korak u rješavanju problema genetskim algoritmima. Postoji više načina na koje se roditelji mogu odabrati. Za odabir najboljih jedinki koristi se fitness funkcija, tj. funkcija kvalitete rješenja koja se računa za svaku jedinku.

Mogući način odabira roditelja je uz pomoć jednostavne selekcije (eng. *roulette-wheel parent selection*). Ovisno o funkciji kvalitete rješenja pojedina jedinka zauzima određeni postotak kruga. Nasumičnim odabirom točaka unutar tog kruga biraju se roditelji. Naravno, jedinka s boljom vrijednost funkcije kvalitete rješenja ima veću šansu postati roditelj i razmnožavati se jer zauzima veći dio tog kruga.

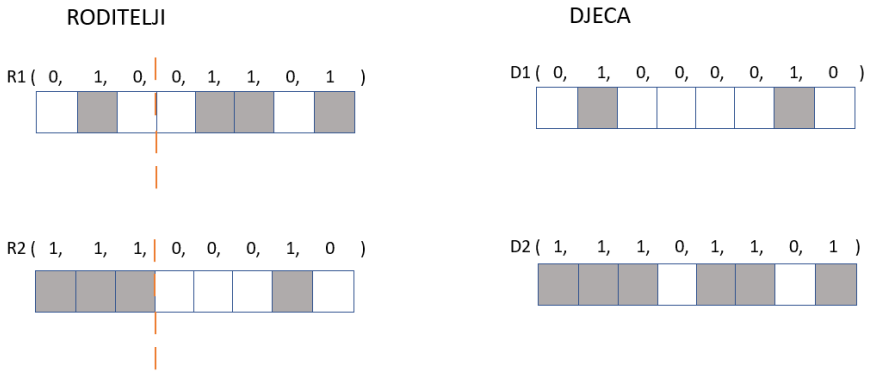
Sljedeći način za odabir roditelja je turnir. U zaseban skup se nasumično izdvoji k jedinki te se za novog roditelja odabere najbolja od njih. Proces se ponavlja dok se ne odabere zadovoljavajući broj roditelja. Naravno, iz skupa od k jedinki može se izdvojiti i više jedinki za razmnožavanje.

Posljednji način je nasumičan odabir roditelja. Neovisno o funkciji kvalitete rješenja bira se zadovoljavajući broj roditelja nasumično. Obzirom na to da se roditelji ne biraju ovisno o funkciji kvalitete rješenja ovaj način se ne preporuča.

3.2.3. Križanje

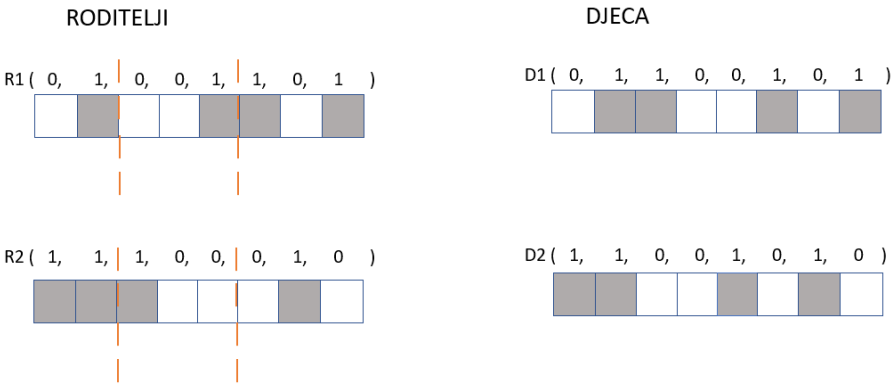
Nakon odabira roditelja provodi se križanje kojim se stvaraju nove jedinke. Križaju se dvije jedinke koje nazivamo roditeljima i na taj način se stvaraju nove jedinke, tj. djeca. Roditelji djeci prenose svoj genetski materijal i tako genetski materijal roditelja, tj. njihova svojstva, prelaze u novu generaciju. Svaka jedinka nastala križanjem sadrži genetski materijal oba roditelja. Postoji mogućnost da dijete postane bolje rješenje od roditelja što je povoljno za razvoj daljnjih potencijalnih rješenja. Više je načina križanja te se često i kombinira više načina križanja pri stvaranju novih populacija.

Prvi način je križanje u jednoj točki. Odabere se nasumično ili unaprijed određena pozicija u kromosomu te prvo dijete sadrži gene prvog roditelja do te pozicije i gene drugog roditelja od zadane pozicije nadalje, dok za drugo dijete vrijedi obrnuto: ono sadrži gene drugog roditelja do te pozicije i gene prvog roditelja od te pozicije.



Slika 3.1 Križanje u jednoj točki

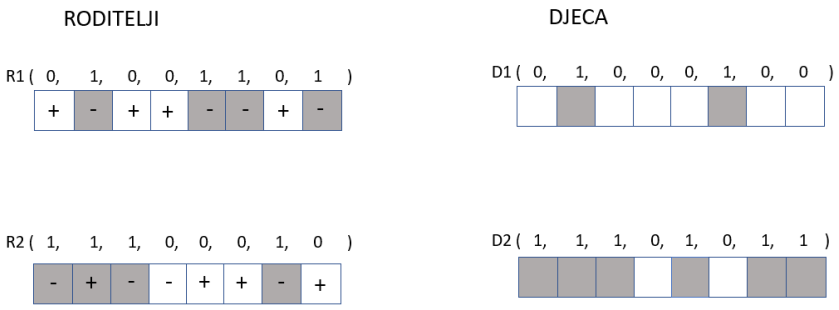
Drugi način je križanje između dvije nasumično unaprijed određene pozicije u kromosomu. Prvo dijete tada sadrži gene prvog roditelja do prve određene pozicije, gene drugog roditelja od prve pozicije do druge pozicije te gene od prvog roditelja od druge pozicije nadalje. Drugo dijete se stvara obrnutim postupkom: prvi dio gena do prve pozicije sadrži od drugog roditelja, zatim gene između prve i druge pozicije od prvog roditelja te gene od druge pozicije nadalje od drugog roditelja.



Slika 3.2 Križanje između dvije pozicije

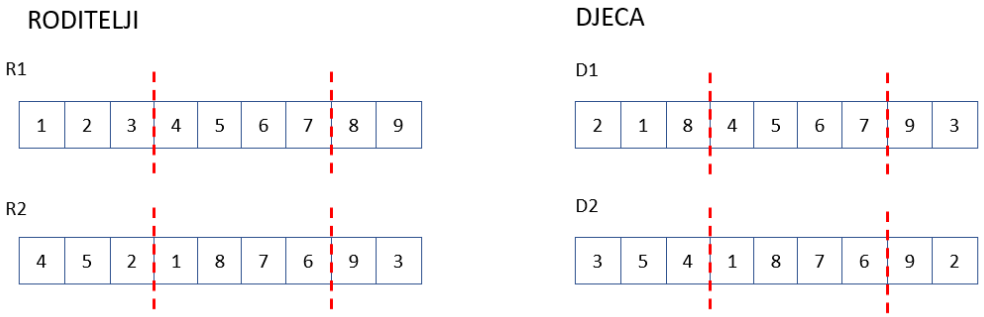
Na treći način križanje se može napraviti tako da se odaberu određene pozicije na kojima će se zamijeniti geni roditelja pa će prvo dijete imati sve gene prvog roditelja osim gena na odabranim pozicijama na kojima će se nalaziti geni drugog roditelja, a drugo dijete će imati

sve gene drugog roditelja osim gena na odabranim pozicijama gdje će biti geni prvog roditelja.



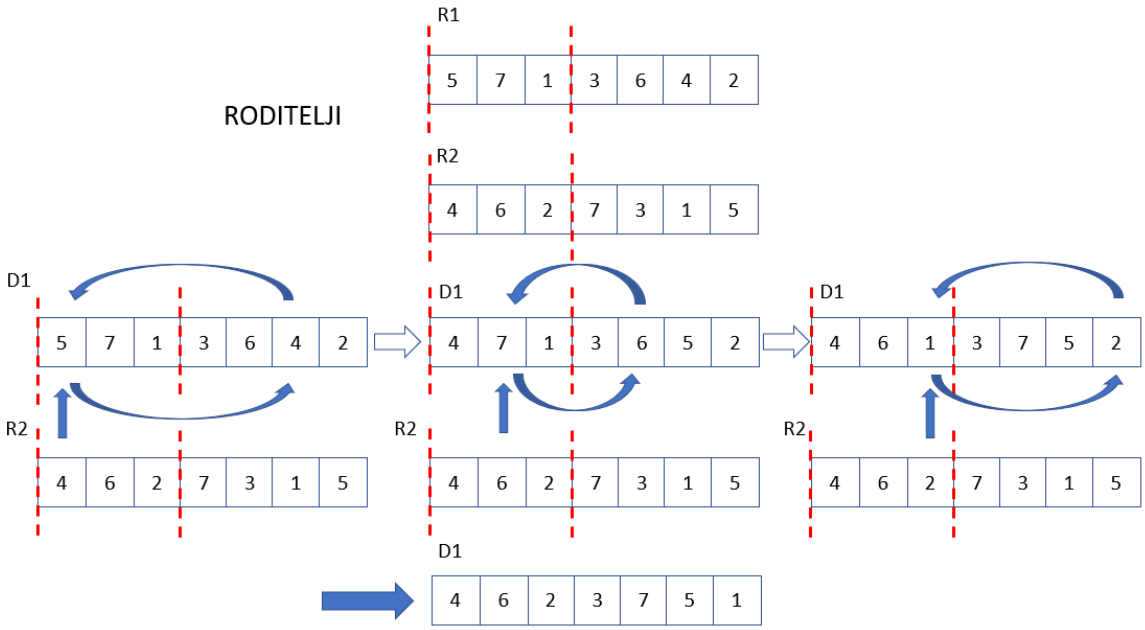
Slika 3.3 Križanje na određenim pozicijama

Sljedeći način križanja je takozvano OX1 križanje, tj. poredano križanje. Odaberu se dvije fiksne pozicije te dijete između te dvije pozicije nasljedi gene prvog roditelja, a preostale gene uzme od drugog roditelja redom uz uvjet da ne uzima gene koji su već uzeti od prvog roditelja.



Slika 3.4 OX1 križanje

Naredni način križanja je PMX križanje, tj. križanje s djelomičnim preklapanjem. Dijete nasljedi redom sve gene od jednog od roditelja. Za određeni raspon pozicija gena gleda koji gen se nalazi na toj poziciji kod drugog roditelja pa se kod djeteta zamijene pozicije ta dva gena kao što je prikazano na slici:



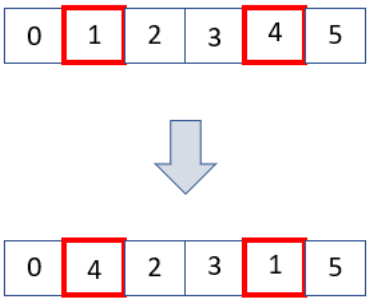
Slika 3.5 PMX križanje

Treba napomenuti da ne nastaje baš svaka jedinka nove populacije samo križanjem, ponekad se „elitne jedinke“, tj. roditelj, prenose u sljedeću populaciju, a ponekad se u sljedeću populaciju prenose nove jedinke s mutiranim genima.

3.2.4. Mutacije

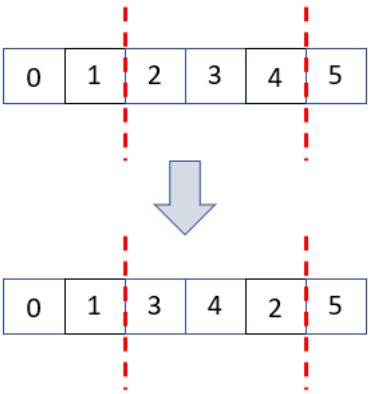
Mutacije se definiraju kao promjene gena u kromosomu. One su vrlo bitne jer stvaraju novo moguće rješenje problema, doprinose raznolikosti jedinki u populaciji, obavljaju ulogu potpunog pretraživanja te sprječavaju populaciju da „zapne“ u lokalnom ekstremu[4]. Do mutacija dolazi samo u određenom postotku gena u jedinkama nakon križanja. Najčešće se bira mali postotak jedinki u kojima će doći do mutacije jer u suprotnom genetski algoritam vodi ka nasumičnom stvaranju rješenja.

Postoji više načina ostvarenja mutacija. Prvi od načina je zamjena gena jedinke na određenim pozicijama.



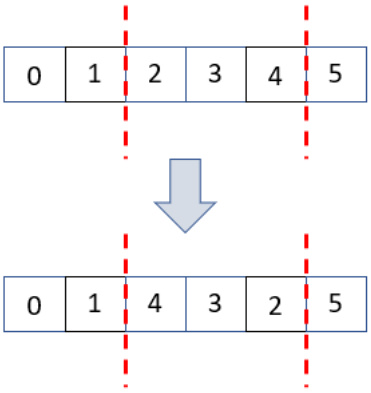
Slika 3.6 Mutacija zamjenom dva gena

Sljedeći način na koji se ostvaruju mutacije je odabir dvije pozicije između kojih će geni međusobno nasumično zamijeniti mjesta.



Slika 3.7 Mutacija zamjenom pozicija gena u određenom dijelu kromosoma

Treći način je odabir dvije pozicije između kojih će geni obrnuti svoj redoslijed i time stvoriti jedinku koja se razlikuje od početne.



Slika 3.8 Preokretanje redoslijeda gena u određenom dijelu kromosoma

3.3. Kriterij završetka i odabir najboljeg rješenja

Nove generacije stvaraju se sve dok se ne zadovolji kriterij završetka. Kriterij završetka može se odabrati tako da se stvori samo određen broj novih generacija ili određen broj izmjena u populaciji te se u konačnici po funkciji kvalitete rješenja kao rješenje bira najbolja jedinka iz posljednje generirane populacije. Sljedeći način odabira kriterija završetka je unaprijed određivanje vrijednosti funkcije kvalitete rješenja i zaustavljanje stvaranja nove generacije ili izmjena trenutne populacije kada se u tadašnjoj populaciji pojavi jedinka sa zadovoljavajućom vrijednosti funkcije kvalitete rješenja. Ta jedinka se onda odabere kao rješenje zadanog problema. Nadalje, kriterij završetka može se definirati i kao postotak promjene najboljeg rješenja pri izmjeni trenutne populacije ili generiranju nove generacije. Ukoliko je promjena kvalitete rješenja najbolje jedinke manja od zadanog postotka proces se zaustavlja te se kao konačno rješenje uzima tadašnja najbolja jedinka. Također, uvjet zaustavljanja može biti određeno vrijeme koliko se algoritam vrti, a kada zadani broj sekundi prođe odabire se najbolja jedinka kao rješenje problema.

3.4. Vrste genetskih algoritama

Prepoznajemo dvije temeljne vrste genetskih algoritama s obzirom na vrstu selekcije: eliminacijski i generacijski. Razlika između ove dvije vrste je u stvaranju „nove generacije“, tj. „nove populacije“.

3.4.1. Generacijski genetski algoritam

Generacijski genetski algoritam nakon koraka selekcije u potpunosti stvara novu generaciju te staru generaciju briše. Novu generaciju čine „elitne jedinke“ koje imaju dobru funkciju dobrote te njihovim križanjem nastaju nove jedinke koje upotpunjuju novonastalu populaciju. No, nije nužno da se baš svaka „elitna jedinka“ križa i stvara djecu. Stvara se onoliko novih jedinki koliko je potrebno da populacija bude veličine početne populacije. Također, nove jedinke mogu mutirati i time doprinijeti raznolikosti populacije i širenju skupa potencijalnih rješenja.

3.4.2. Eliminacijski genetski algoritam

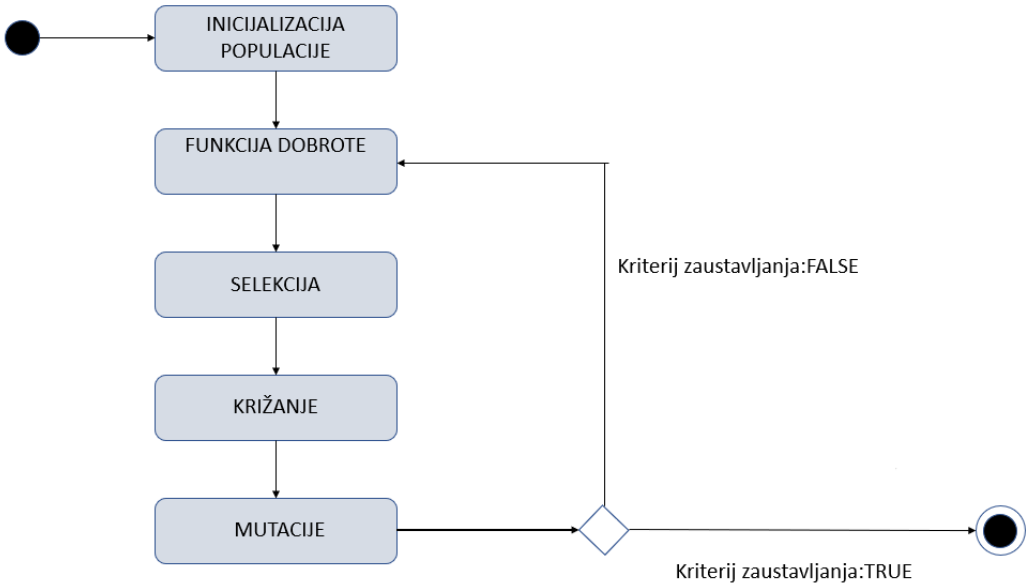
Eliminacijski genetski algoritam usredotočen je na eliminaciju „slabijih“ jedinki iz populacije. U koraku selekcije ne stvara u potpunosti novu generaciju nego se nekom od metoda selekcije biraju „slabije“ jedinke koje se zatim eliminiraju iz populacije. Jedinke s manjom funkcijom dobrote smatraju se lošijima te imaju manju šansu za reprodukciju i opstankom u populaciji. Križanjem „boljih“ jedinki stvaraju se nove koje potom postaju dio populacije. Određeni postotak gena novonastalih jedinki mutira te tako mogu nastati jedinke koje već postoje u populaciji. Takve „duplicirane“ jedinke mogu ostati u populaciji ili se mogu brisati i zamijeniti novim jedinkama.

```
Eliminacijski genetski algoritam{
   generiraj početnu populaciju;
   sve dok nije zadovoljen uvjet završetka evolucijskog procesa{
      izračunaj vjerojatnost eliminacije za svaku jedinku;
      M puta jednostavnom selekcijom odaberi i izbriši jedinku;
      parenjem preživjelih jedinki nadopuni populaciju;
   }
}
```

Slika 3.9 Pseudokod eliminacijskog genetskog algoritma [4]

3.5. Programski način ostvarenja genetskog algoritma

Svaki program koji implementira genetski algoritam započinje inicijalizacijom vjerojatnosti mutacije, veličine populacije i populacije s nasumičnim rješenjima, tj. jedinkama. Zatim se provodi postupak selekcije i evaluacije određenih jedinki. Ako je implementiran generacijski algoritam stvara se nova populacija, a ako je implementiran eliminacijski algoritam biraju se jedinke koje se eliminiraju. Križanjem određenog postotka jedinki stvaraju se nove jedinke koje upotpunjuju populaciju. Određen postotak gena novih jedinki mutira stvarajući novo potencijalno rješenje. Sve dok se ne zadovolji uvjet zaustavljanja ovaj postupak se ponavlja. Nakon ispunjivanja uvjeta zaustavljanja odabire se jedinka koja na temelju funkcije dobrote predstavlja najbolje rješenje zadanog problema.



Slika 3.10: Prikaz postupaka kod provođenja genetskog algoritma

4. Programsko rješenje

Program je napisan u programskom jeziku Python. Programsko rješenje nudi rješenje za dvije vrste problema trgovačkog putnika: simetrični problem trgovačkog putnika i asimetrični problem trgovačkog putnika.

Program korištenjem funkcije loadtext() iz biblioteke numpy učitava matricu udaljenosti između gradova iz datoteke. Grad predstavlja pozicija u prvom retku ili u prvom stupcu dobivene matrice.

Programski je ostvaren troturnirski eliminacijski algoritam koji pri svakoj iteraciji nasumično bira 3 jedinke te ih sortira po funkciji dobrote. Ona jedinka čije je rješenje evaluirano kao najlošije se eliminira iz populacije te se stvara nova jedinka križanjem preostale dvije koja se zatim mutira i ubacuje u populaciju.

Jedinku predstavlja lista koja na svakom mjestu ima određen broj koji predstavlja grad. Trgovački putnik prolazi gradove redom kojim su navedeni u toj listi te se na kraju vraća u početni grad.

```
[3, 0, 2, 1, 4]
[0, 4, 2, 3, 1]
[2, 1, 3, 4, 0]
```

Slika 4.1 Primjer 3 različite jedinke za rutu od 5 gradova

Funkcije initialise_population() inicijalizira populaciju od 60 jedinki pozivajući funkciju initialise_individual(). Funkcija initialise_individual() inicijalizira jedinku tako da napravi listu s nasumičnim redoslijedom svih gradova.

```
def initialise_population(city_count):
    population = [ ]
    for i in range(POPULATION_SIZE):
        population.append(initialise_individual(city_count))
    return population


def initialise_individual(city_count):
    initial = list(range(city_count))
    random.shuffle(initial)
    return initial
```

Kod 4.1 Funckije initialise_population() i initialise_individual()

Funkcija fitness_function() izračunava kvalitetu rješenja za pojedinu jedinku tako da zbraja udaljenost između gradova redom kojim su oni navedeni u jedinki i udaljenost posljednjeg grada od prvog grada obzirom na to da se trgovački putnik mora vratiti u početni grad.

```
def fitness_function(individual, tsp_base):
    fit_result = 0
    for i in range(len(individual) - 1):
        fit_result += tsp_base[ individual[ i ] ][ individual[ i + 1 ] ]
    fit_result += tsp_base[ individual[ -1 ] ][ individual[ 0 ] ]
    return fit_result
```

Kod 4.2 Funkcija fitness_function()

Funkcija tournament_picking() nasumično odabire 3 jedinke koje potom evaluira i sortira po kvaliteti rješenja. Jedinka koja je po funkciji fitness_function() ocijenjena kao bolja ima veću šansu za preživljavanje i reprodukciju.

```
def tournament_picking(population, tsp_base):
    tournament = [ ]
    for j in range(3):
        index = random.randint(0, len(population) - 1)
        tournament.append((fittness_function(population[ index ], tsp_base), index))
    tournament.sort()

    return tournament[ 0 ][ 1 ], tournament[ 1 ][ 1 ], tournament[ 2 ][ 1 ]
```

Kod 4.3 Turnirski odabir jedinki

Programski je ostvareno križanje s djelomičnim preklapanjem, tj. PMX križanje. U funkciji pmx_crossover() stvara se nova jedinka tako da se od jednog roditelja uzme prvih n gradova te se zamjene mjesta tim gradovima ovisno o tome gdje se nalaze u listi drugog roditelja.

```
def pmx_crossover(parent_1, parent_2):
    n = (len(parent_1) // 3)
    parent_1_copy = parent_1.copy()
    for position in range(n):
        element = parent_2[ position ]
        parent_1_copy[ position ], parent_1_copy[ parent_1.index(element) ] \
            = parent_1_copy[ parent_1.index(element) ], parent_1_copy[ position ]
```

Kod 4.4 PMX križanje

Programski je također ostvareno i križanje u poretku, tj. OX1 križanje. Nasumično se odabiru dvije pozicije između kojih će se prepisati geni od prvog roditelja a zatim će se na preostale pozicije unijeti geni redom kojim se nalaze u drugom roditelju a koji već nisu dio kromosoma djeteta.

```
def ox_crossover(parent_1, parent_2):
    size = len(parent_1)
    child = [-1]*size
    start, end = sorted([ random.randrange(size) for _ in range(2) ])

    for i in range(start, end + 1):
        child[ i ] = parent_1[ i ]

    current_parent_2_position = 0
    for i in range(0, start):
        while parent_2[ current_parent_2_position ] in child:
            current_parent_2_position += 1
        child[ i ] = parent_2[ current_parent_2_position ]
    for i in range(end+1, size):
        while parent_2[ current_parent_2_position ] in child:
            current_parent_2_position += 1
        child[ i ] = parent_2[ current_parent_2_position ]

    return child
```

Kod 4.5 OX1 križanje

Populacija se mijenja turnirskim odabirom 3 jedinke, eliminacijom najlošije jedinke i dodavanjem jedinke koja nastaje križanjem u poretku ili križanjem s djelomičnim preklapanjem dvije jedinke koje su evaluirane kao bolje rješenje na temelju funkcije kvalitete rješenja.

```
def next_population(population, tsp_base, crossover_type):
    # turnir
    p1_index, p2_index, worst_index = tournament_picking(population, tsp_base)
    # pmx ili ox križanje
    parent_1 = population[ p1_index ]
    parent_2 = population[ p2_index ]
    population.pop(worst_index)
    if (crossover_type == "pmx"):
        new_individual = pmx_crossover(parent_1, parent_2)
    else:
        new_individual = ox_crossover(parent_1, parent_2)
    # mutation
    individual_mutation = mutation(new_individual)
    population.append(individual_mutation)
    return population
```

Kod 4.6 Funkcija next_population()

Do mutacije gena dolazi samo kod jedinki koje su nastale križanjem. Vjerojatnost mutacije gena postavljena je na 5% stoga samo 5% gena novih jedinki nastalih križanjem mutira. Prilikom mutacije dva gena zamijene mjesta čime se mijenja redoslijed gradova.

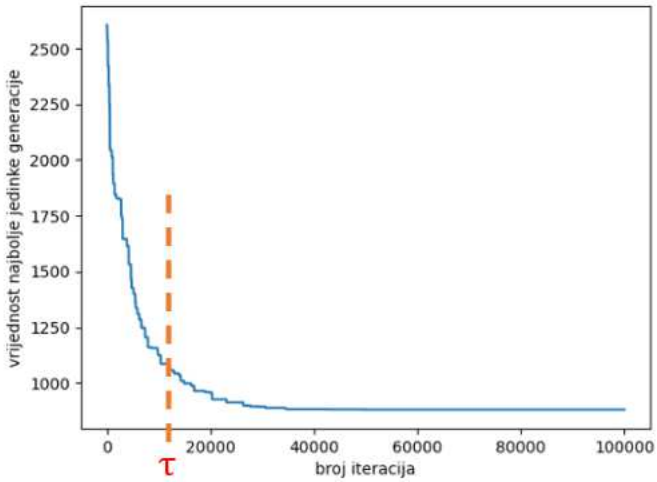
```
def mutation(individual):
    individual_copy = individual.copy()
    for i in range(len(individual) - 1):
        probability = random.randint(0, 10000)
        if probability <= 10000 * MUTATION_PROBABILITY:
            mutation_point_b = random.randint(0, len(individual) - 1)
            individual_copy[i], individual_copy[mutation_point_b] = \
                individual_copy[mutation_point_b], individual_copy[i]
    return individual_copy
```

Kod 4.7 Funkcija mutation()

Nakon stvaranja nove jedinke ponavlja se postupak procjene kvalitete jedinke po funkciji kvalitete rješenja, ponovno se odabiru roditelji i na isti način se mijenja populacija.

Obzirom na to da fitness_function() računa kvalitetu kromosoma kao zbroj udaljenosti između gradova koje prolazi trgovački putnik, vidi se da zaista jedinke s najboljim rasporedom gradova opstaju dulje u populaciji, dok one s lošijim rasporedom gradova se eliminiraju u turniru.

Za uvjet zaustavljanja postavljen je određen broj iteracija, tj. broj evaluacija. Broj iteracija određuje se unaprijed. Obično se za broj iteracija uzima broj koji je približno 5 puta veći od broja iteracija u kojima se dobiveno najbolje rješenje intenzivno mijenja(τ). To je takozvano koljeno funkcije evolucije koja prikazuje najbolju dobivenu jedinku u pojedinoj iteraciji što je vidljivo na **Error! Reference source not found.**.



Slika 4.2 Prikaz vrijednosti τ

Na grafu je vidljivo da funkcija dobrote najbolje jedinke populacije pada s brojem iteracija što ukazuje na to da razmnožavanjem jedinki i mutacijom određenog postotka gena novonastalih jedinki nastaju sve bolje prilagođene jedinke. Vrijednost τ izračunata je korištenjem biblioteke kneed i pripadne funkcije KneedLocator().

5. Eksperimentalni rezultati

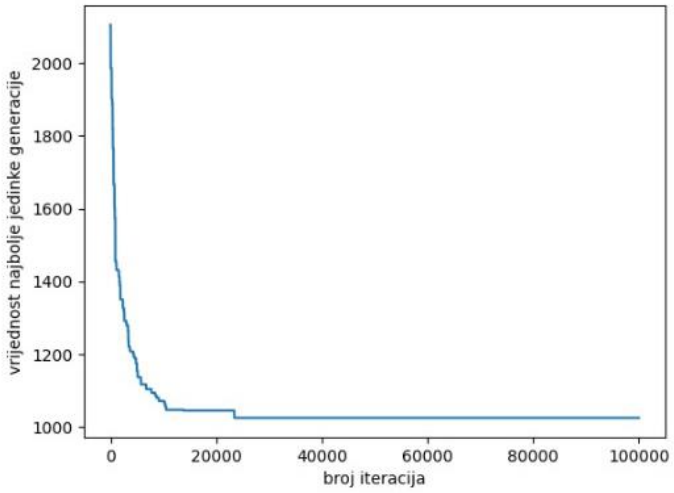
Eksperimenti su provedeni na 4 različita asimetrična problema trgovačkog putnika.

Prvo je provedeno testiranje rute koja sadrži svega 5 gradova i čiji optimalni put ima cijenu 19. Testiranje je obuhvaćalo 10 različitih mjerenja za svaku vrstu križanja. Uvjet zaustavljanja bio je postavljen na 100000 iteracija te je broj jedinki populacije iznosio 60, a vjerojatnost mutacije 5%. Dobiveni rezultati pokazuju su da prosječni τ iznosi 1 što znači da bi nam trebalo svega 5 iteracija kako bismo došli do zadovoljavajućeg rješenja problema. Također, korištenjem obje vrste križanja algoritam je došao do optimalnih rezultata svaki put.

Zatim je provedeno testiranje rute od 26 gradova čiji je optimalan put duljine 937. Uvjet zaustavljanja je također bio postavljen na 100000 iteracija te je broj jedinki populacije iznosio 60 i vjerojatnost mutacije 5%. Dobiveni rezultati prikazani su sljedećom tablicom i grafom:

RBR EKSPERIMENTA	τ1	τ2	NAJBOLJA JEDINKA 1	NAJBOLJA JEDINKA 2	ODSTUPANJE OD OPTIMALNOG PUTA IZ LITERATURE 1		ODSTUPANJE OD OPTIMALNOG PUTA IZ LITERATURE 2	
1	11024	10957	1057	1074	120	12,8 %	137	14,6 %
2	11797	10669	937	1031	0	0,0 %	94	10,0 %
3	11074	11470	1053	1007	116	12,4 %	70	7,5 %
4	12864	13092	1015	975	78	8,3 %	38	4,1 %
5	11396	11143	987	1085	50	5,3 %	148	15,8 %
6	10379	10281	975	1185	38	4,1 %	248	26,5 %
7	10113	11194	1112	1199	175	18,7 %	262	28,0 %
8	10638	1115	953	1105	16	1,7 %	168	17,9 %
9	11818	12225	955	1021	18	1,9 %	84	9,0 %
10	10366	12938	1003	956	66	7,0 %	19	2,0 %

Tablica 5.1 Rezultati eksperimenata na primjeru od 26 gradova



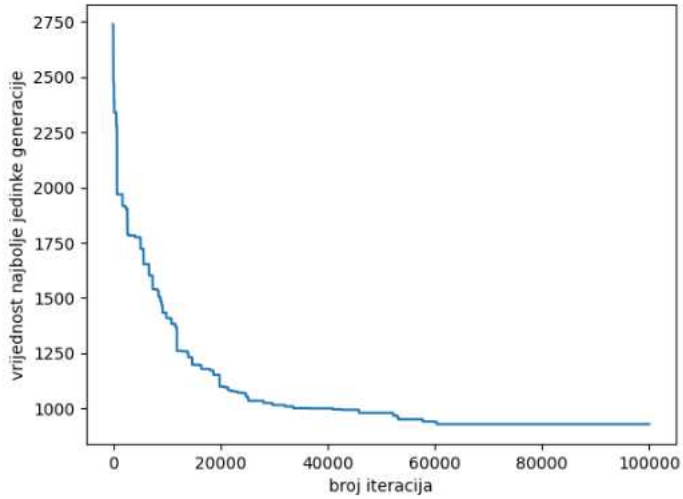
Slika 5.1 Prikaz intenziteta promjene rješenja na primjeru od 26 gradova

Stupci u tablici označeni brojem 1 prikazuju rezultate dobivene poredanim križanjem jedinki, a stupci označeni brojem 2 prikazuju rezultate dobivene križanjem s djelomičnim preklapanjem. Vidljivo je da poredano križanje vodi ka boljim i preciznijim rezultatima od križanja s djelomičnim preklapanjem. Vodeći se teorijom da se do zadovoljavajućih rezultata dolazi nakon 5τ iteracija, na temelju prva dva stupca iz Tablica 5.1 može se vidjeti da je 100000 iteracija dovoljno za pronalaženje rješenja tog problema. Također je vidljivo da su odstupanja od optimalnog rješenja relativno mala.

Sljedeća mjerenja napravljena su na ruti od 42 grada s optimalnim putem cijene 699. Tablicom i grafom prikazani su rezultati 10 provedenih eksperimenata na ovom primjeru. Broj jedinki populacije ponovo je bio postavljen na 60, vjerojatnost mutacije na 5% te je uvjet zaustavljanja bio 100000 iteracija.

RBR EKSPERIMENTA	τ1	τ2	NAJBOLJA JEDINKA 1	NAJBOLJA JEDINKA 2	ODSTUPANJE OD OPTIMALNOG PUTA IZ LITERATURE 1		ODSTUPANJE OD OPTIMALNOG PUTA IZ LITERATURE 2	
1	16808	16111	933	856	234	33,5 %	157	22,5 %
2	13844	15469	973	991	274	39,2 %	292	41,8 %
3	20521	13571	740	844	41	5,9 %	145	20,7 %
4	13511	18242	806	1008	107	15,3 %	309	44,2 %
5	14866	14635	881	1086	182	26,0 %	387	55,4 %
6	16087	14552	982	920	283	40,5 %	221	31,6 %
7	11867	13478	892	892	193	27,6 %	193	27,6 %
8	13404	14802	887	1011	188	26,9 %	312	44,6 %
9	31018	20662	796	942	97	13,9 %	243	34,8 %
10	12965	17964	919	939	220	31,5 %	240	34,3 %

Tablica 5.2 Rezultati eksperimenata na primjeru od 42 grada



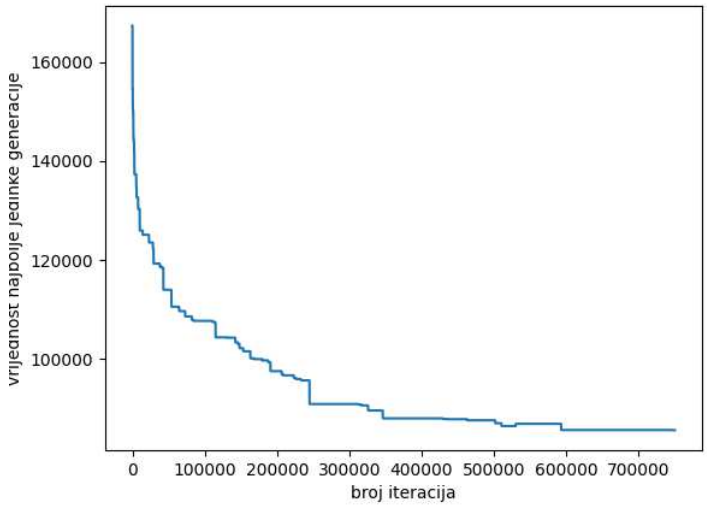
Slika 5.2 Prikaz intenziteta promjene rješenja na primjeru od 42 grada

Stupci u tablici označeni brojem 1 prikazuju rezultate dobivene poredanim križanjem, a stupci označeni brojem 2 prikazuju rezultate dobivene križanjem s djelomičnim preklapanjem. Ponovo je vidljivo da su u prosjeku rezultati dobiveni poredanim križanjem bolji od rezultata dobivenih križanjem s djelomičnim preklapanjem. Iz grafa na Slika 5.2 i u prve dvije kolone tablice možemo vidjeti da je τ približno 15000 što znači da bi 75000 iteracija vodilo do zadovoljavajućeg rješenja. Odstupanja od optimalnog rezultata su u ovom primjeru veća od odstupanja u prethodnom primjeru koji je imao manji broj gradova.

Posljednje mjerenje napravljeno je na primjeru od 124 grada čiji je optimalni put duljine 36230. U prvih 5 mjerenja broj iteracija bio je postavljen na 100000 i broj jedinki populacije na 60 te vjerojatnost mutacije na 5%, no kako rezultati nisu bili zadovoljavajući i τ je u prosjeku bio znatno veći od 100000/5, broj iteracija povećan je na 600000, a broj jedinki populacije na 80. Nakon promjene parametara vidljivo je značajno poboljšanje u pronalasku rješenja. Sljedeća tablica i graf prikazuju rezultate 10 provedenih eksperimenata na ruti od 124 grada.

RBR EKSPERIMENTA	τ1	τ2	NAJBOLJA JEDINKA 1	NAJBOLJA JEDINKA 2	ODSTUPANJE OD OPTIMALNOG PUTA IZ LITERATURE 1		ODSTUPANJE OD OPTIMALNOG PUTA IZ LITERATURE 2	
1	21477	20295	101358	104143	65128	179,8 %	67913	187,5 %
2	28231	30148	106443	101032	70213	193,8 %	64802	178,9 %
3	65452	33322	98881	108736	62651	172,9 %	72506	200,1 %
4	26670	27656	99209	98935	62979	173,8 %	62705	173,1 %
5	35083	19211	99110	107062	62880	173,6 %	70832	195,5 %
6	90088	272353	78927	85578	42697	117,9 %	49348	136,2 %
7	99948	89900	82919	87359	46689	128,9 %	51129	141,1 %
8	61812	93489	85284	82980	49054	135,4 %	46750	129,0 %
9	87940	121735	83844	80133	47614	131,4 %	43903	121,2 %
10	100972	158768	82802	82511	46572	128,6 %	46281	127,7 %
11	160106	83958	83445	82210	47215	130,3 %	45980	126,9 %
12	222920	88535	82480	83953	46250	127,66	47723	131,72

Tablica 5.3 Rezultati eksperimenata na primjeru od 124 grada



Slika 5.3 Prikaz intenziteta promjene rješenja na primjeru od 124 grada

Iz

RBR EKSPERIMENTA	τ1	τ2	NAJBOLJA JEDINKA 1	NAJBOLJA JEDINKA 2	ODSTUPANJE OD OPTIMALNOG PUTA IZ LITERATURE 1		ODSTUPANJE OD OPTIMALNOG PUTA IZ LITERATURE 2	
1	21477	20295	101358	104143	65128	179,8 %	67913	187,5 %
2	28231	30148	106443	101032	70213	193,8 %	64802	178,9 %
3	65452	33322	98881	108736	62651	172,9 %	72506	200,1 %
4	26670	27656	99209	98935	62979	173,8 %	62705	173,1 %
5	35083	19211	99110	107062	62880	173,6 %	70832	195,5 %
6	90088	272353	78927	85578	42697	117,9 %	49348	136,2 %
7	99948	89900	82919	87359	46689	128,9 %	51129	141,1 %
8	61812	93489	85284	82980	49054	135,4 %	46750	129,0 %
9	87940	121735	83844	80133	47614	131,4 %	43903	121,2 %
10	100972	158768	82802	82511	46572	128,6 %	46281	127,7 %
11	160106	83958	83445	82210	47215	130,3 %	45980	126,9 %
12	222920	88535	82480	83953	46250	127,66	47723	131,72

Tablica 5.3 je vidljivo da povećanje broja iteracija znatno utječe na konačan rezultat te da su rezultati dobiveni eksperimentima s većim brojem iteracija znatno bolji od rezultata dobivenih sa 100000 iteracija. No, rezultati ovog eksperimenta su generalno lošiji od rezultata eksperimenata provedenih na manjem broju gradova što se jasno vidi iz stupaca u kojima su prikazani postotci odstupanja od optimalnog rješenja. Brojem 1 označeni su rezultati dobiveni poredanim križanjem, a brojem 2 rezultati dobiveni križanjem s

djelomičnim preklapanjem. Također, uočljivo je da je manja relativna razlika između rezultata dobivenih poredanim križanjem, koji su u ovom slučaju neznatno bolji, i rezultata dobivenih križanjem s djelomičnim preklapanjem.

Eksperimentalni rezultati ukazuju na to da je važno dobro odrediti parametre jer oni uvelike utječu na konačno rješenje. U prvom provedenom eksperimentu bilo je nepotrebno imati 100000 iteracija i populaciju od 60 jedinki za problem koji se sastojao samo od 5 gradova. Populacija je sadržavala više istih jedinki obzirom na to da je prostor rješenja relativno malen. No, u posljednjem provedenom eksperimentu uočljivo je da broj iteracija ipak nije bio dovoljno velik obzirom na to da su odstupanja od optimalnog rješenja značajna. Povećanje broja iteracija sa 100000 na 600000 i broja jedinki populacije sa 60 na 80 vodilo je puno boljim konačnim rezultatima, ali je izvođenje programa bilo znatno sporije.

Također, bitan je i odabir načina križanja. U provedenim eksperimentima poredano križanje u većini slučajeva dalo je bolje rezultate od križanja s djelomičnim preklapanjem stoga bi za rješavanje novih problema ovom implementacijom bilo bolje koristiti poredano križanje.

Iz prikazanih grafova je vidljivo da se algoritam puno brže približava optimalnom rješenju pri rješavanju problema trgovačkog putnika s manjim brojem gradova uz jednaku veličinu populacije u svakom eksperimentu. Povećanje populacije i broja iteracija uobičajeno vodi ka boljem konačnom rješenju, ali rezultira usporavanjem algoritma stoga bi trebalo pronaći kompromis između veličine populacije, broja iteracija i brzine izvođenja algoritma.

6. Zaključak

Primjena rješenja problema trgovačkog putnika je velika. Iz eksperimenata provedenih na rutama različitih brojeva gradova vidljivo je da je rješenje dobiveno eliminacijskim genetskim algoritam prihvatljivo. Genetski algoritmi primjenjivi su i pri rješavanju brojnih drugih problema stoga se i ovo programsko rješenje uz nekoliko preinaka može prilagoditi rješavanju problema sličnih problemu trgovačkog putnika. Iako se egzaktnim algoritmima dobiva optimalno rješenje, oni često predugo traju, dok s genetskim algoritmima u razumnom vremenu dobivamo prihvatljivo rješenje. Obzirom na to da se ne pretražuje cijeli prostor rješenja, genetski algoritmi su brži, posebice na većim skupovima problema koji zahtjevaju veći broj iteracija čime se radi kompromis između kvalitete rješenja i brzine. Eliminacijom lošijih jedinki i rekombinacijama jedinki koje predstavljaju bolje rješenje te mutacijama određenog postotka gena jedinki nastalih rekombinacijom algoritam pri svakoj izmjeni populacije vodi ka sve boljim rješenjima problema.

Rezultati dobiveni programskim rješenjem ukazuju na to da je programsko rješenje gotovo idealno za primjere u kojima je broj gradova relativno malen, a povećanjem broja gradova u primjerima sve je veća razlika između optimalnog rješenja i rješenja dobivenog implementacijom genetskog algoritma. Nažalost, genetski algoritmi ne jamče optimalno rješenje, ali namještanjem parametara može se doći do zadovoljavajućeg ili čak optimalnog rješenja. Primjerice ovaj genetski algoritam uvijek daje optimalno rješenje za rutu od 5 gradova, ali povećanjem populacije ili broja iteracija može dati optimalno rješenje i za veće rute. No, broj jedinki populacije i broj iteracija treba povećavati racionalno jer ni veći broj jedinki ni veći broj iteracija ne jamče optimalnost rješenja, a ponekad su čak ti brojevi bespotrebno veliki što samo usporava algoritam. Također, poredano križanje dalo je bolje rezultate od križanja s djelomičnim preklapanjem stoga je prikladnije koristiti poredano križanje pri traženju rješenja problema konkretno u ovoj implementaciji.

7. Literatura

[1] Gutin, G., Punnen, A. P. *The travelling salesman problem and its variations*. New York: Springer Science+Business Media, LLC

[2] Applegate, D. L., Bixby, R. E., Cvatal, V., Cook, W. J. The Traveling Salesman Problem: A Computational Study. New Jersey: Princeton University Press, 2007.

[3] Čupić, M. *Raspoređivanje nastavnih aktivnosti evolucijskim računanjem*. Doktorski rad. Sveučilište u Zagrebu Fakultet elektrotehnike i računarstva, 2011.

[4] Golub, M. *Genetski algoritam: Prvi dio*. Sveučilište u Zagrebu Fakultet elektrotehnike i računarstva, 2004., dostupno na Internet adresi: http://www.zemris.fer.hr/~golub/ga/ga_skripta1.pdf

[5] Umbarkar, A. J., Sheth, P. D. *Crossover operators in genetic algorithms*: *A review*, dostupno na Internet stranici: http://ictactjournals.in/paper/IJSC_V6_I1_paper_4_pp_1083_1092.pdf

[6] Čupić, M. *Prirodom inspirirani optimizacijski algoritmi*. Sveučilište u Zagrebu Fakultet elektrotehnike i računarstva, 2010.

[7] Čupić, M. Evolucijsko računarstvo. 1. izdanje, predavanje iz predmeta: Uvod u umjetnu inteligenciju, Sveučilište u Zagrebu Fakultet elektrotehnike i računarstva, 2019.

[8] Šnajder, J., Dalbelo Bašić, B. Umjetna inteligencija: Heurističko pretraživanje. Sveučilište u Zagrebu Fakultet elektrotehnike i računarstva, 2019.

[9] Jakobović, D. *Adaptive Genetic Operators in Elimination Genetic Algorithm*, Sveučilište u Zagrebu Fakultet elektrotehnike i računarstva, 1999.

[10] De Jong, K. A., Spears, W. M., Using *Genetic Algorithms to Solve NP- Complete problems*, 1989. dostupno na Internet adresi: http://citeseerx.ist.psu.edu/viewdoc/versions;jsessionid=D557F70FAD37EB98502FB06B17677F19?doi=10.1.1.55.841

[11] Saiyed, A. R., The Traveling Salesman Problem, Indiana State University, 2012. dostupno na Internet stranici: http://cs.indstate.edu/~zeeshan/aman.pdf

[12] Cook, W. J., *In Pursuit of the Traveling Salesman: Mathematics at the Limits of Computation*, New Jersey: Princeton University Press, 2014.

[13] Ahmed, Z. H., Al-Dayel, I., *An Exact Algorithm for the Single-Depot Multiple Travelling Salesman Problem*, Riyadh: 2020. dostupno na Internet adresi: http://paper.ijcsns.org/07_book/202009/20200909.pdf

[14] Universität Heidelberg, *TSPLIB: Library of Travelling Salesman Problem*, dostupno na Internet stranici: http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/ , pristupljeno 01.ožujka 2022.

[15] Lucasius, C. B., Kateman, G. Understanding and using genetic algorithms Part 1. Concepts, properties and contex, Amsterdam: 1993. dotupno na Internet adresi: https://www.sciencedirect.com/science/article/abs/pii/016974399380079W

Rješavanje problema trgovačkog putnika pomoću genetskog algoritma

Sažetak

U ovom radu obrađeni su problem trgovačkog putnika i genetski algoritmi. Prva dva poglavlja daju uvid u problem trgovačkog putnika i detaljan opis ideje i vrsta genetskih algoritama te njihovih opetatora. Često korištena metoda rješavanja problema trgovačkog putnika su genetski algoritmi. U svrhu ovog rada napravljeno je programsko rješenje u programskom jeziku Python implementacijom eliminacijskog genetskog algoritma koje je prikazano u 4. poglavlju. Programsko rješenje rješava simetrični i asimetrični problem trgovačkog putnika dajući ne optimalne, ali zadovoljavajuće rezultate. Navedeni su prednosti i nedostaci korištenja genetskih algoritama pri rješavanju problema trgovačkog putnika. Rezultati provedenog eksperimenta dani su grafički i u tablicama.

Ključne riječi: problem trgovačkog putnika, evolucija, genetski algoritam, populacija, jedinka, gen, selekcija, križanje, mutacije, kvaliteta rješenja

Solving the traveling salesman problem using genetic algorithms

Summary

This paper deals with the traveling salesman problem and genetic algorithms. The first two chapters provide an insight into the traveling salesman problem and a detailed description of the idea and types of genetic algorithms and their operators. A commonly used method for solving the traveling salesman problem is genetic algorithms. For the purpose of this paper, a software solution in the Python programming language was developed by implementing the elimination genetic algorithm, which is presented in Chapter 4. The software solution solves the symmetrical and asymmetric traveling salesman problem by giving not optimal, but good enough results. The advantages and disadvantages of using genetic algorithms in solving the traveling salesman problem are listed. The results of the experiment are given graphically and in tables.

Key words: traveling salesman problem, evolution, genetic algorithm, population, individual, gene, selection, crossover, mutations, solution quality