SVEUČILIŠTE U SPLITU
PRIRODOSLOVNO-MATEMATIČKI FAKULTET

PROBLEM TRGOVAČKOG PUTNIKA

Marija Vištica

Mentor:
prof.dr.sc. Slavomir Stankov

Neposredni voditelj:
dr.sc. Ani Grubišić

Split, listopad 2012.

Sadržaj

1 Uvod

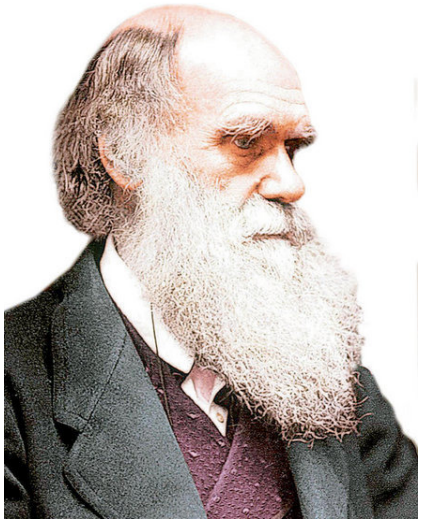
Razvojem računarstva i povećanjem procesne moći računala, ljudi su počeli rješavati izuzetno kompleksne probleme koji su do tada bili nerješivi – i to naprosto tehnikom grube sile (engl. *brute-force*, koristi se još i termin *iscrpnom pretragom*), tj. pretraživanjem cjelokupnog prostora rješenja. S obzirom da su danas računala ekstremno brza, nije rijetka situacija da se njihovom uporabom u sekundi mogu istražiti milijuni potencijalnih rješenja, i time vrlo brzo pronaći ono najbolje. Naravno, ovo vrijedi ako rješavamo problem koji se dade riješiti grubom silom. Međutim, postoji čitav niz problema koje susrećemo u svakodnevnom životu, a ne spadaju u ovu skupinu. [ČUPIĆ, 2009] Jedan od primjera je i *problem trgovačkog putnika*, problem koji spada u razred *NP-teških problema* – problema za koje još uvijek nemamo efikasnih algoritama kojima bismo ih mogli riješiti.

Na sreću, optimalno rješenje često nam nije nužno, obično smo zadovoljni i s rješenjem koje je dovoljno dobro. Algoritme koji pronalaze rješenja koja su zadovoljavajuće dobra, te koji imaju relativno nisku računalnu složenost nazivaju se približni algoritmi, heurističke metode ili jednostavno *heuristike*. U današnje doba posebno su nam zanimljive metaheuristike. *Metaheuristika* je skup algoritamskih koncepata koji koristimo za definiranje heurističkih metoda primjenjivih na širok skup problema. Jedan od primjera metaheuristike je evolucijsko računanje u čije područje spadaju genetski algoritmi, *evolucijske strategije* te evolucijsko programiranje čija je zajednička ideja da rade s populacijom rješenja nad kojima se primjenjuju evolucijski operatori čime populacija iz generacije u generaciju postaje sve bolja i bolja. [ČUPIĆ, 2009]

U ovom radu su opisan je način rješavanja problema trgovačkog putnika putem evolucijskih strategija. U prvom dijelu opisane su evolucijske strategije. Zatim je opisan problem trgovačkog putnika i zbog čega predstavlja toliki izazov. U zadnjem dijelu slijedi praktični rad u okviru kojeg je napravljena aplikacija koja pokušava naći najbolje rješenje za problem trgovačkog putnika pomoću evolucijskih strategija.

2 Evolucijske strategije

Evolucijske strategije su optimizacijske, prirodom inspirirane, strategije koje naglasak stavljaju na evoluciji i prilagodbi generacija. Zasnovane su na ideji natjecanja rješenja i uvelike se oslanjanju na Darwinovu teoriju evolucije: preživljavaju samo najbolji, koju je on i prvi razvio u svoj djelu *O podrijetlu vrsta*, 1859. godine. Temelje se prvenstveno na metodama selekcije i rekombinacije, sa ciljem da se unaprjeđuju najbolje jedinke iz generacije te se na taj način pokušava stvoriti „bolja“ generacija čime dobivamo najbolje rješenje za dani problem.



Slika 1. Charles Darwin [http://www.legalbytes.com/]

2.1 Prikaz rješenja

Populacija se sastoji od određenog broja jedinki koje predstavljaju računalne programe, odnosno strukture koje se mogu jednoznačno preslikati u oblik pogodan za izvođenje na računalu. Svi podaci koji obilježavaju jednu jedinku zapisani su u kromosomu i mogu biti pohranjeni na razne načine. Kromosom se može sastojati od niza bitova (binarni prikaz kromosoma), ili se može sastojati od realnog broja ili vektora realnih brojeva.

Za evolucijske strategije karakteristično je prikazivanje rješenja pomoću vektora realnih brojeva. Broj na određenom mjestu unutar vektora opisuje neku karakteristiku samog rješenja. [DRAGOJEVIĆ, 2008]

2.2 Funkcija dobrote

Kriterij kvalitete je poznat kao funkcija dobrote ili funkcija ocjene kvalitete jedinke (engl. *fitness*) te pomoću njega odlučujemo koje ćemo rješenje odabrati i o njemu najviše ovisi cijeli proces. Dobrota je obično određena nedosljednošću između rezultata dobivenog od jedinke i željenog rezultata. Što je pogreška manja, program je bolji, odnosno što je dobrota jedinke veća, jedinka ima veću vjerojatnost preživljavanja, a time i veću mogućnost prijenosa svojih gena na sljedeću populaciju. U kreiranju efikasnog algoritma, definiranje funkcije dobrote predstavlja ključni, a često i najteži korak, kako bi funkcija vjerno održavala problem koji se rješava. [BESPALJKO, 2009]

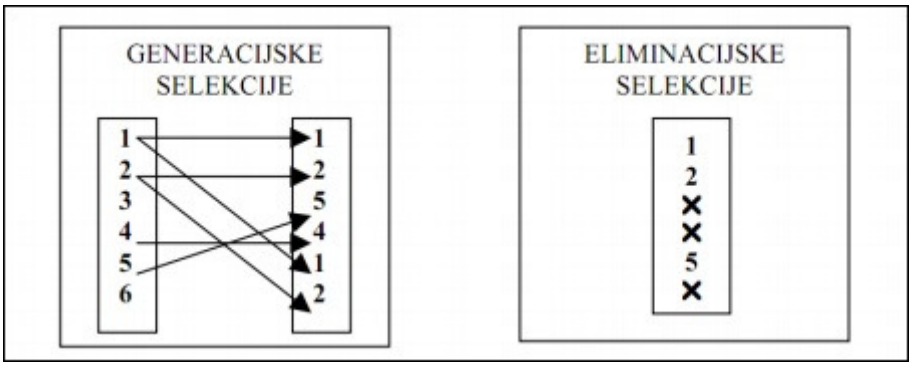
2.3 Selekcija

Nakon što smo odredili kvalitetu jedinke pomoću funkcije dobrote, moramo odlučiti hoćemo li primijeniti genetske operatore na tu jedinku, zadržati je u populaciji ili ju zamijeniti. Selekcija je posljedica natjecanja između jedinki populacije.

Postupak selekcija bi se mogao ostvariti odabirom N najboljih jedinki (gdje je N veličina populacije), ali takav odabir bi doveo do prerane konvergencije rješenja – proces optimizacije završi u svega nekoliko iteracija, te postoji opasnost od „zaglavljivanja“ procesa u nekom lokalnom optimumu[1]. Problem je u tome što se takvim odabirom gubi dobar genetski materijal koji mogu sadržavati loše jedinke pa iz tog razloga i one moraju imati određenu vjerojatnost preživljavanja (veću od nule). [PIELIĆ, 2010]

Vrste operatora selekcije dijelimo na generacijske i eliminacijske. Selekcije koje spadaju u generacijske, odabiru iz trenutne generacije najbolje jedinke i njihovim kopiranjem stvaraju novu generaciju. Nedostatak ovakve vrste selekcije je što u sljedećoj generaciji može postojati više potpuno jednakih jedinki, čime se smanjuje vrijedna genetska raznolikost. Eliminacijske selekcije su usredotočene na biranje jedinki koje će biti odstranjene iz populacije. Lošije jedinke imaju manju vjerojatnost preživljavanja. Nakon njihovog uklanjanja, populacija se nadopuni novim jedinkama koje se dobiju križanjem novih kromosoma.

[1] **Lokalni optimum** je element prostora rješenja sa svojstvom da nijedan niz mutacija tog elementa ne može imati isključivo padajuće vrijednosti evaluacijske funkcije.



Slika 2. Podjela postupka selekcije prema načinu prenošenja genetskog materijala boljih jedinki u novu generaciju

Postoji mnogo načina odabira roditelja (engl. *parent selection shemes*) za sljedeću generaciju, a najpoznatiji od njih su jednostavna selekcija, turnirska selekcija i eliminacijska selekcija koje ćemo pojasniti u nastavku. [PIELIĆ, 2010]

2.3.1 Jednostavna selekcija

Jednostavna selekcija (engl. *roulette wheel parent selection*) je osnovni primjer generacijske selekcije. Cilj je odabrati roditelja čija je vrijednost selekcije proporcionalna njihovoj dobroti. Jedinka sa najvećom dobrotom ima najveće šanse da bude izabrana, dok najlošija jedinka ima najmanje šanse preživljavanja. Vjerojatnost preživljavanja ostalih jedinki je negdje između. [GOLUB, 2004]

Jednostavna selekcija ima nekoliko nedostataka: budući da je vjerojatnost odabira jedinke proporcionalna njenoj dobroti, funkcija dobrote ne smije poprimati negativne vrijednosti. Ovaj nedostatak je moguće riješiti translacijom funkcije dobrote. Drugi nedostatak jednostavne selekcije je i pojavljivanje dupliciranih kromosoma. Eksperimenti su pokazali da i do 50% populacije mogu biti duplikati kromosoma, čime se značajno smanjuje učinkovitost algoritma. [BOŽIKOVIĆ, 2000]

2.3.2 Turnirska selekcija

Turnirska selekcija (engl. *tournament selection*) može biti generacijska ili eliminacijska. Odabire se slučajan broj jedinki iz populacije, a najbolje od njih postaju roditelji sljedeće generacije. Ovaj se selekcija često naziva i k-turnirski odabir, gdje k označava broj jedinki čije se dobrote uspoređuju. Taj se broj naziva veličina prozora.

U slučaju generacijske selekcije, među izabranim jedinkama se traži najbolja koja se zatim kopira u sljedeću generaciju, a postupak je potreno ponoviti onoliko puta dok broj odabranih jedinki na postane jednak veličini populacije.

Ako se radi o eliminacijskoj selekciji, najlošija jedinka od odabranih se uklanja, a zamjenjuje je nova jedinka nastala križanjem dvije slučajne jedinke iz trenutnog turnira. Ova selekcija, a posebno eliminacijska inačica, je pogodna za probleme koji imaju složene kromosome i kod kojih je izračun funkcije dobrote vremenski zahtjevan. [PIELIĆ, 2010]

2.3.3 Eliminacijska selekcija

Eliminacijska selekcija (engl. *steady state selection*) je eliminacijska varijanta jednostavne selekcije. Za razliku od jednostavne selekcije, ne biraju se kromosomi koji će preživjeti već oni loši kromosome koje treba eliminirati i reprodukcijom ih zamijeniti novima. Dakle, loši kromosomi umiru, a njih zamjenjuju djeca nastala reprodukcijom roditelja, tj. preživjelih kromosoma. [GOLUB, 2004]

Vjerojatnost selekcije jedinke je to veća što je dobrota manja. Umjesto funkcije dobrote treba definirati funkciju kazne, koja ima veću vrijednost za lošije jedinke i predstavlja vjerojatnost uklanjanja jedinke iz populacije. Dakle, njena je vrijednost odabira obrnuto proporcionalna njenoj dobroti. Najbolja jedinka će na ovaj način imati vjerojatnost selekcije jednaku nuli, čime se osigurava očuvanje najbolje jedinke, što se naziva *elitizam*.

Jednom odabrani kromosom se više ne može odabrati. Ovaj način selekcije je veoma brz i daje dobre rezultate. [FABRIS, 2005]

2.4 Genetski operatori

U evolucijskom postupku važno je imati nekakav mehanizam varijacije kako bi stvorili razliku i bili sigurni da sljedeće generacije potomaka ne postanu identične kopije roditelja. U slučaju da takav mehanizam ne postoji, daljnja poboljšanja ne bi bila moguća. Dva moguća operatora varijacije u evolucijskim algoritmima su mutacija i križanje, odnosno razmjena genetskog materijala između jedinki. Mutacija (*unarni operator*) mijenja mali dio jedinke dok križanje (*operator višeg reda*) miješa genetski materijal između dviju jedinki kako bi kreiralo potomka koji je kombinacija svojih roditelja. [GOLUB, 2004]

2.4.1 Križanje

Križanje (engl. *crossover*), koje se često naziva i rekombinacija, je proces kojim iz dvije jedinke (roditelji) nastaje nova jedinka (dijete). Najvažnija karakteristika križanja jest da djeca nasljeđuju svojstva svojih roditelja. Novonastalo dijete posjeduje genetski materijal oba roditelja i ima jednak broj gena kao i oni, što znači da je vjerojatnost nasljeđivanja od

pojedinog roditelja obično jednaka i iznosi 0.5. Cilj bi nam, naravno, bio dobiti djecu bolju od svojih roditelja. [PETROVIĆ, 2007]

Križanje se definira proizvoljnim brojem prekidnih točaka. Ovisno o izboru tih točaka postoji nekoliko vrsta križanja:

križanje u jednoj točki (engl. *one-point crossover*),
križanje u dvije točke (engl. *two-point crossover*),
križanje rezanjem i spajanjem (engl. *cut and splice crossover*),
uniformno križanje (engl. *uniform crossover*) i
polu-uniformno križanje (engl. *half-uniform crossover*).

Metode križanja korištene u ovom radu su opisane u poglavlju 4.1.5. *Operatori križanja*.

Kao što smo već spomenuli, potrebno je obratiti pažnju na različitost oba roditelja. Ukoliko su oba jednaka, raditi križanje tada je očito suvišno; tako dobivamo samo još jedan duplikat nekog kromosoma, što nam nije u interesu. U praksi se zato često prije križanja provjerava različitost koja ukoliko nije prisutna, jedan od roditelja se mutira pa se potom obavlja križanje. [PETROVIĆ, 2007]

2.4.2 Mutacija

Mutacija je slučajna promjena jednog ili više gena kako bi se dobila genetska raznolikost sljedeće generacije rješenja.

Kod evolucijskih strategija postoji važno pravilo „1/5 uspjeha“ koje je definirao Rechenberg prilikom svojih istraživanja. Samo pravilo predviđa optimalne performanse za vrijeme trajanja mutacija i to kada od svih mutacija koje se provedu, 20% njih daje uspješne potomke. To znači da kvocijent broja uspješnih mutacija i ukupnog broja mutacija umutar neke populacije mora biti približno 1/5. [PETROVIĆ, 2007]

Mutacijom se pretražuje prostor rješenja što daje mutaciji jednu od najvažnijih osobina. Naime, ovim postupkom se omogućava izbjegavanje lokalnih minimuma[2]. Ako cijela populacija završi u nekom lokalnom minimumu, mutacija će slučajnim pretraživanjem prostora pronaći i bolje rješenje. [PETROVIĆ, 2007]

Postoje razne metode mutacija (slučajna, inverzna, indukcijska, uniformna, Gaussova mutacija, itd.), a one korištene u ovom radu su opisane u poglavlju 4.1.4. *Operatori mutacije*.

[2] **Lokalni minimum** je točka na grafu u kojoj funkcija prelazi iz pada u rast.

2.5 Vrste evolucijskih strategija

Pretpostavimo da je broj roditelja u nekoj generaciji γ označen sa μ, a broj potomaka u generaciji γ označen sa λ. Postoji sedam različitih tipova evolucijskih procesa koje koriste evolucijske strategije [ČERI, MALOVIĆ, 2007].

$(1 + 1)$-ES

U populaciji postoje dvije jedinke. Jedna je jedinka roditelj iz kojeg nakon reprodukcije procesom mutacije nastaje potomak. Proces selekcije se obavlja između te dvije jedinke, a u sljedeću generaciju ide bolja jedinka, tj. koja ima bolji faktor dobrote [ČERI, MALOVIĆ, 2007].

$(\mu + 1)$-ES

Populacija se sastoji od μ jedinki roditelja. Mutacijom jedne jedinke nastaje jedan potomak koji se konstantno reproducira. Iz spojenog seta potomaka izabrane jedinke i trenutne populacije odbacuje se jedinka koja ima najmanji faktor dobrote [ČERI, MALOVIĆ, 2007].

$(\mu + \lambda)$-ES

U ovom slučaju iz μ jedinki roditelja nastaje λ potomaka procesom mutacije. Uvjet za ovaj proces je da je nastalo više djece nego što je roditelja, dakle μ<λ. Svaki od λ potomaka ima svoj faktor dobrote, kao što imaju i svi roditelji. Najboljih μ jedinki iz skupa roditelja i potomaka zajedno prelazi u sljedeću generaciju [ČERI, MALOVIĆ, 2007].

(μ, λ)-ES

Populacija se sastoji od μ jedinki roditelja, koji procesom mutacije daje λ potomaka, s time da vrijedi μ< λ. Svaki od λ potomaka ima svoj faktor dobrote. Za razliku od prethodnog slučaja, ovdje se za prijelaz u novu generaciju promatraju samo potomci, dakle roditelji ne ulaze u izbor. Iz λ potomaka izabire se μ najboljih jedinki koje prelaze u sljedeću generaciju [ČERI, MALOVIĆ, 2007].

$(\mu/\rho, \lambda)$-ES

Ova strategija je (μ, λ) strategija uz dodatak parametra ρ. Ovaj parametar označava broj jedinki roditelja koji se upotrebljava u procesu reprodukcije. Ukoliko je ρ=1, ova strategija je analogna strategiji (μ, λ)-ES, tj. prilikom reprodukcije koristi se samo jedna jedinka iz populacije roditelja, što znači da se upotrebljava operator mutacije. Ukoliko je ρ=2, prilikom reprodukcije se koriste dvije jedinke iz populacije roditelja, što znači da se upotrebljava operator rekombinacije. Selekcija je analogna selekciji kod (μ, λ)-ES [ČERI, MALOVIĆ, 2007].

$(\mu/\rho + \lambda)$-ES

Ova strategija je (μ + λ) strategija uz ponovni dodatak parametra ρ. Za reprodukciju vrijede ista pravila kao kod (μ/ ρ, λ) strategije, a selekcija je analogna selekciji kod (μ + λ)-ES [ČERI, MALOVIĆ, 2007].

$(\mu', \lambda', (\mu, \lambda)\gamma)$-ES

Kod ove strategije se iz populacije roditelja veličine μ' kreira λ' potomaka i izolira na γ generacija. U svakoj od γ generacija stvara se λ potomaka od kojih samo μ najboljih prelazi u sljedeću generaciju. Nakon γ generacija izabiru se najbolje jedinke od γ izoliranih populacija i krug kreće ponovno sa λ' novih jedinki potomaka [ČERI, MALOVIĆ, 2007].

2.6 Algoritmi evolucijskih strategija

Cilj evolucijske strategije je stvoriti populaciju potomaka uz pomoć procesa rekombinacije ili mutacije iz početne populacije roditelja. Postupak mutacije i rekombinacije se ponavlja i nastaju nove generacije potomaka, sve dok se ne dođe do optimuma nekog zadanog problema. Ovaj proces je prikazan pseudokodom na sljedećoj slici. [ČERI, MALOVIĆ, 2007]

```
Evolucijska strategija{
        t = 0;
        generiraj početnu populaciju P(0);
        evaluiraj P(0); //provjeri dobrotu inicijalne populacije
        sve dok nije zadovoljen uvjet završetka evolucijskog procesa{
                izaberi najboljih μ roditelja iz P(t) i stavi ih u Pr(t);
                iz Pp(t) reproduciraj λ potomaka i stav ih u Pr(t);
                mutiraj Pr(t);
                evaluiraj Pr(t);
                ako se koristi plus strategija P(t+1) = Pr(t) u P(t);
                        inače P(t+1) = Pr(t);
                t = t+1;
        }
}
```

Slika 3. Pseudokod procesa evolucijske strategije

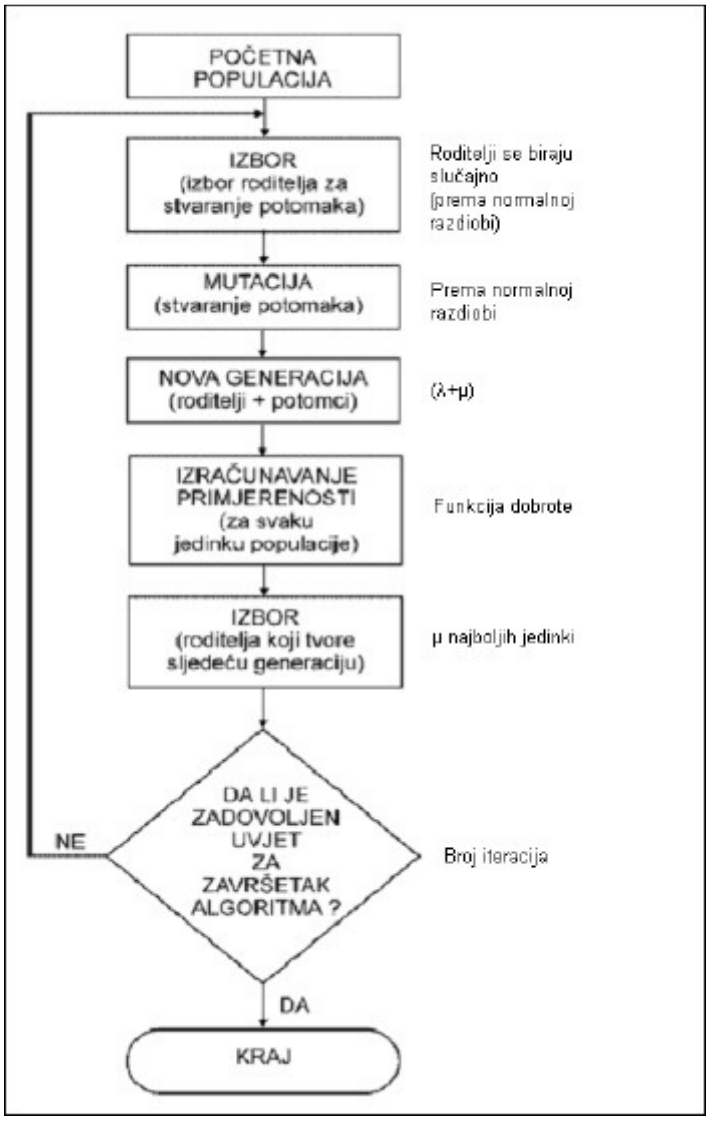
Dakle, postupak generiranja nove generacije populacije ponavljat će se sve dok ne bude zadovoljen uvjet zaustavljanja. Najčešće korišteni uvjeti zaustavljanja su:

- pronađeno je dovoljno dobro rješenje (kriteriji „dovoljno dobrog rješenja moraju biti definirani),
- potrošeni su resursi,
- broj generiranja novih generacija je premašio unaprijed definiranu granicu,
- istekao je unaprijed definiran interval vremena,
- zaključeno je da daljnje iteracije neće dati poboljšanje rezultata.

Proces stvaranja nove generacije se može opisati kroz sljedeće korake:

1. Stvori λ novih jedinki:
    a. odaberi slučajnih ρ≤μ roditelja
    b. križaj ρ roditelja kako bi dobio potomka
    c. odredi novu vrijednost parametra ρ prema normalnoj razdiobi
    d. mutiraj dijete pomoću nove vrijednosti parametra ρ
2. Odaberi novu populaciju na temelju funkcije dobrote:
    a. iz populacije djece ukoliko se radi o (λ, μ)-ES
    b. iz populacije djece i roditelja ukoliko se radi o (λ + μ)-ES

Dijagram tijeka evolucijske strategije (λ + μ) prikazan je na slici 4.



Slika 4. Dijagram tijeka (λ + μ) evolucijske strategije

2.7 Primjena evolucijskih strategija

Evolucijske strategije imaju široku primjenu u rješavanju raznih optimizacijskih problema (uključujući optimizacijske probleme u kontinuiranom i diskretnom vremenu, vektorsku optimizaciju i kombinatorne probleme sa i bez ograničenja). Koriste se u dubinskoj analizi podataka[3], u izradi kompliciranih vremenskih rasporeda, u kemiji, za geometriju (npr. kod izrade mostova), za rješavanje raznih verzija problema trgovačkog putnika, u optici i obradi slike, u aproksimaciji polinomnih funkcija, itd... [ČERI, MALOVIĆ, 2007]

[3] Podatkovno rudarenje ili dubinska analiza podataka (eng. *Data mining*) je sortiranje, organiziranje ili grupiranje velikog broja podataka i izvlačenje relevantnih informacija.

3 Problem trgovačkog putnika

Problem trgovačkog putnika (engl. *Traveling Salesman Problem, TSP*) je problem diskretne i kombinatorne optimizacije. Spada u skupinu NP-teških problema, a složenost mu je O(N!).

Porijeklo ovog problema seže daleko u povijest. Matematičke probleme slične problemu trgovačkog putnika je počeo razmatrati Euler, kojeg je zanimalo kako bi skakač na šahovskoj ploči posjetio sva 64 mjesta samo jednom. Prva doslovna pojava problema trgovačkog putnika bila je u knjizi njemačkog trgovačkog putnika B.F. Voighta, 1832. Autor knjige sugerira kako je pokrivanje što više lokacija bez posjećivanja ijedne od njih dvaput najvažniji apsekt planiranja putovanja. No, ni on nije problem trgovačkog putnika tako imenovao. . [PIELIĆ, 2008]

Početkom 20.st. matematičari William Rowan Hamilton i Thomas Kirkman su razmatrali probleme koji se svode na problem trgovačkog putnika. Hamilton je osmislio igru (slika 5.) koja je od igrača tražila da odredi stazu između 20 točaka uz korištenje definiranih veza između njih. Opća forma ovog problema se pojavljuje 30-ih godina 20.stoljeća zahvaljujući Karl Mengeru kada je i sam pojam „trgovački putnik“ prvi put upotrijebljen. [PIELIĆ, 2008]

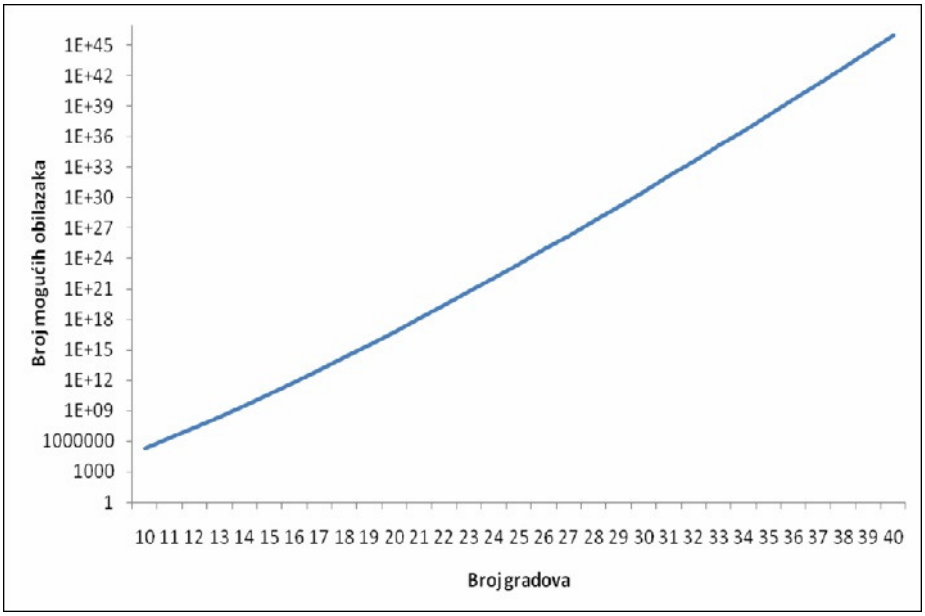
Slika 5. Hamiltonova igra (engl. icosian game) [http://4.bp.blogspot.com/]

3.1 NP-teški problemi

U kombinatornoj optimizaciji često se pojavljuju NP-teški problemi (engl. *non-polynomial hard problems*). Riječ je o zadaćama koje se prema općem uvjerenju ne mogu rješavati u polinomskom vremenu t = Nk, gdje je N dimenzija problema, a k konstanta. Najčešće su složenosti O(N!), O(2N) i sl. Klasičan algoritam je jednostavan: provjeri sva moguća rješenja da bi našao pravo, ali je neizvediv osim za male N. Na primjer, iz grupe od 400 ljudi treba odabrati 100 prema zadanom kriteriju. Broj kombinacija je ~ 1096, što je više od broja atoma u cijelom svemiru. NP-teški problemi ne omogućuju jednostavnu provjeru dobivenog rješenja i ne znamo da li postoji bolje. [VELJAN, 2001]

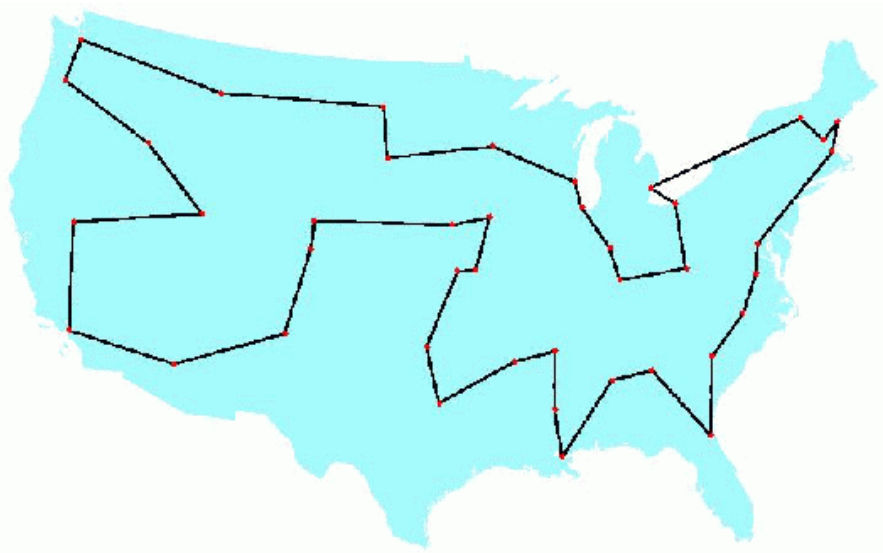
3.2 Opis problema trgovačkog putnika

Sam problem je vrlo jednostavan – trgovački putnik ima unaprijed definirane gradove i sve međusobne udaljenosti među njima, te mora posjetiti svaki grad samo jednom i vratiti se u početni grad. Pitanje je kojim bi redoslijedom trgovački putnik morao obilaziti gradove, a da ukupna duljina puta bude minimalna. Na prvi pogled ovaj se problem i ne čini preteškim, ali ako se uzme u obzir da ima faktorijelnu složenost, tj. da se najkraći put između N gradova nalazi negdje unutar prostora stanja koji je velik N!/2N, odnosno ((N-1)!)/2 (slika 6.), računanjem se dobije da već za 11 gradova postoji 1 814 400 mogućih redoslijeda obilazaka (kombinacijska eksplozija). [PIELIĆ, 2008]



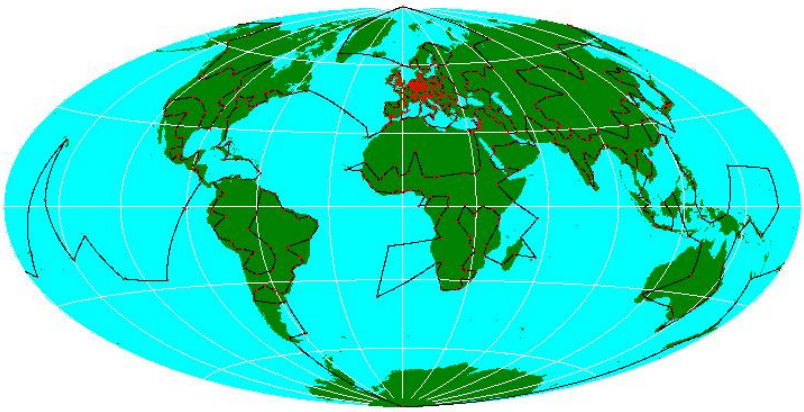
Slika 6. Složenost problema trgovačkog putnika

Prvi veliki problem trgovačkog putnika koji je bio riješen je bio problem sa 49 gradova, koje su riješili Dantzig, Fulkerson i Johnson (slika 7.). [GUTIN, PUNNEN, 2004]

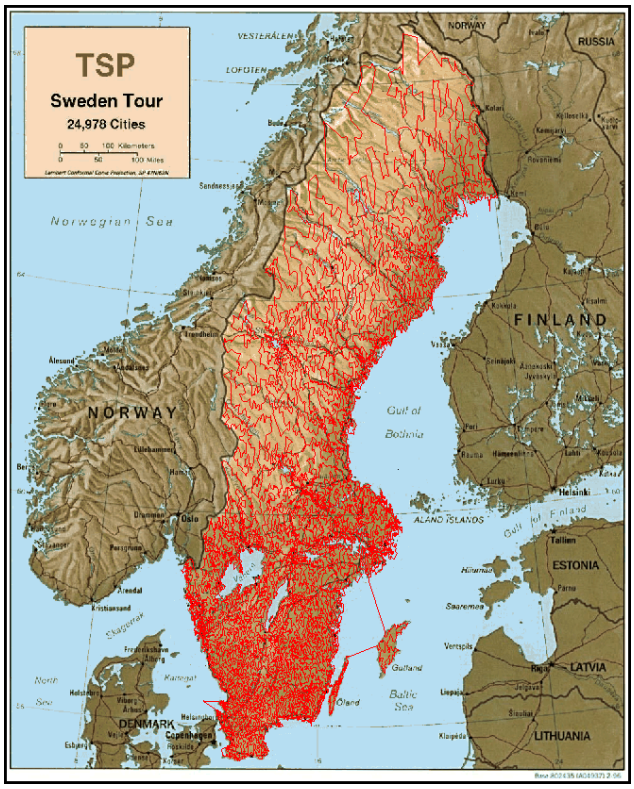


Slika 7. Problem trgovačkog putnika sa 49 gradova [www.tsp.gatech.edu/history/img/dantzig.gif]

Grötschel i Holland su 1987. riješili problem sa 666 gradova (slika 8.), a 2004. su godine su Applegate, Bixby, Chvátal, Cook, and Helsgaun riješili problem sa 24 978 gradova u Švedskoj (slika 9.).



Slika 8. Problem trgovačkog putnika sa 666 gradova [www.tsp.gatech.edu/history/img/gr666.gif]



Slika 9. Problem trgovačkog putnika sa 24 978 gradova [www.tsp.gatech.edu/history/img/sw_relief1.gif]

Matematička formulacija problema trgovačkog putnika je sljedeća:

> Zadan je težinski graf G=(V;E), gdje je težina c_{ij} između čvorišta *i* i *j* ne-negativna vrijednost. Potrebno je pronaći Hamiltonov – razapinjujući – put, odnosno put koji sadrži sve vrhove grafa, na način da je ukupna cijena puta minimalna.

Ovako obrađen problem trgovačkog putnika nije ograničen izborom postojećih puteva, odnosno, pretpostavlja se da su svaka dva grafa međusobno povezana putovima. Također, problem je u ovom primjeru simetričan, odnosno, put iz grada i u grad j je jednako „težak“ kao i put iz grada j u grad i. To ne mora biti slučaj u stvarnoj primjeni – primjerice, težina prolaska određenog puta može ovisiti o dobu dana ili godine što nam zapravo pokazuje da postoje podvrste problema trgovačkog putnika. [VELJAN, 2001]

Jedini potpuno siguran način za rješavanje ovog problema je sveobuhvatno pretraživanje kompletnog prostora rješenja, odnosno provjeravanje svakog rješenja iz skupa rješenja koje se bazira na linearnom programiranju ili *branch-and-bound* principu, što je zbog „prostranosti“ prostora rješenja nemoguće pretražiti u polinomskom vremenu pa su se metode za rješavanje problema trgovačkog putnika razvijale u smjeru približnih metoda koje se osnivaju na principu lokalnog pretraživanja ili koriste neki metaheuristički algoritam, primjerice algoritam najbližeg susjeda (engl. *Nearest neighbour algorithm*), pohlepni algoritam (engl. *Greedy algorithm*), simulirano kaljenje, i dr.

Jedna od algoritama i genetski algoritmi zasnovani na evolucijskim strategijama koji su već objašnjene, a u poglavlju 3 ćemo pokazati kako se oni realiziraju na našem zadatku.

3.3 Varijante problema trgovačkog putnika

U literaturi [HJERTENES, 2002] se spominje niz varijanti problema trgovačkog putnika u odnosu na opći predhodno definiran problem. Ove varijante su nastale modeliranjem realnih problema, a u nastavku su predstavljene neke od najčešćih.

MAX TSP – razlika između MAX TSP i standardno definiranog problema trgovačkog putnika se sastoji samo u definiciji granične težnje. Naime, u standardnom problemu se određuje ciklus sa najmanjom vrijednosti kriterija, a kod MAX TSP se određuje ciklus sa najvećom vrijednosti kriterija.

TSP za izbjegavanje uskog grla (engl. *bottleneck TSP*) – kod ove varijante, cilj je da se pronađe takav Hamiltonov ciklus u grafu G za koji je najveća cijena svih njegovih grana što manja.

TSP sa višestrukim posjetama (TSPM) – ova varijanta zahtijeva da se odredi takva staza iz grafa G kod koje se svaki čvor grafa posjeti barem jednom, uz postizanje minimalne vrijednosti kriterija.

Problem glasnika – ova varijanta predstavlja problem kod kojeg je za dva data čvora x i y grafa G potrebno odrediti Hamiltonovu stazu koja polazi iz čvora x i završava u čvoru y, uz postizanje minimalne vrijednosti kriterija.

Klasterizirani TSP – kod ove varijante čvorovi grafa G su podijeljeni u klastere V_1, V_2, ..., V_k. Potrebno je odrediti Hamiltonov ciklus koji daje minimalnu vrijednost kriterija, uz ograničenje da se svi čvorovi klastera moraju posjetiti jedan za drugim.

TSP sa m trgovačkih putnika – neka se u polaznim čvoru grafa G nalazi m trgovačkih putnika. Svaki od njih će obići podskup X_i čvorova grafa i vratiti se u polazni čvor. Suma cijena svih prijeđenih grana treba biti minimalna.

3.4 Primjena problema trgovačkog putnika

Osim što predstavlja veliki izazov za rješavanje, problem trgovačkog putnika je široko primjenjiv na različite probleme usmjeravanja i uspoređivanja. [FABRIS, 2009]

Jedna od prvih primjena bilo je raspoređivanje putova školskih autobusa koji su vozili djecu, te je poslužila kao motivacija Merrilu Floodu, pioniru u proučavanju problema trgovačkog putnika, za daljnje istraživanje. Sljedeća primjena 1940-ih, odnosila se na transport poljoprivredne opreme, i dovela je do matematičkih istraživanja Mahalanobisa u Bengalu i Jessena u Iowi. Još davno se metodologija rješavanja problema trgovačkog putnika koristila

za djelotvorno raspoređivanje kovanica po telefonskim govornicama diljem gradova. U novije vrijeme ovim problemom se modelira raspoređivanje servisnih posjeta, dovoženje hrane osobama koje su nepokretne, usmjeravanje transportnih kamiona, itd.

Jednostavnost modela dovela je i do njegove primjene na drugim područjima. Klasičan primjer za to je raspored bušenja rupa na tiskanoj pločici. Stroj koji obavlja bušenje rupa mora obići veliki broj mjesta na pločici da bi obavio bušenje. Ako se kao funkcija udaljenosti definira utrošak energije ili vrijeme za micanje glave stroja od jednog do drugog mjesta, onda je cilj taj utrošak ili to vrijeme što više smanjiti – naći obilazak čiji su utrošak ili potrebno vrijeme najmanji. U industriji ovo može višestruko smanjiti troškove proizvodnje. Sličan primjer je raspored pomicanja glave stroja koja otiskuje spojeve na tiskanoj pločici.

Grupa istraživača iz Houstona i sa sveučilišta Brigham Young koristili su ulančanu Lin-Kernighan metodu kako bi optimirali redoslijed udaljenih nebeskih tijela koja bi trebalo slikati u okviru NASA-inog *Starlight* svemirskog programa. Cilj je bio minimizirati utrošak goriva pri manevrima usmjeravanja dvaju satelita. Ovdje su gradovi bili nebeska tijela koja treba snimiti, a udaljenost između njih utrošak goriva.

Proizvođači poluvodičke tehnologije koristili su metode za rješavanje problema trgovačkog putnika kako bi optimirali testne puteve (*scan chains*) na integriranom krugu s ciljem što manjeg utroška energije i bržeg rada čipa.

Da problem ima primjenu i u bioinformatici pokazuje činjenica da je grupa istraživača iz AT&T laboratorija koristila metodologiju rješavanja problema trgovačkog putnika kako bi pronašla najkraći DNA niz dobiven ugrađivanjem DNA nizova (koji se mogu preklapati) iz unaprijed određenog skupa.

Ovaj problem ima mnogo primjena, a ovdje su navedene samo neke od njih. Postoji još niz primjera u kojima se problem trgovačkog putnika pojavljuje kao potproblem ili se drugi problemi mogu svesti na njega.

4 Programsko ostvarenje

4.1 Evolucijske strategije za rješavanje problema trgovačkog putnika

Zbog faktorijelnog rasta prostora rješenja s porastom broja gradova, metoda evolucijskih strategija mogla bi dati dobre rezultate u rješavanju problema trgovačkog putnika. Uz pomoć evolucijskih strategija nije potrebno pretraživati cijeli prostor rješenja, nego se evolucijom iz generacije u generaciju algoritam sve više približava optimalnom rješenju pomoću posebno definiranih operatora križanja i mutacija.

4.1.1 Prikaz kromosoma

Iako postoji više mogućih načina prikaza (vrijednosni prikaz, stablasti prikaz, i dr.), za ovaj rad odabran je najjednostavniji prikaz i prikaz koji se najčešće koristi kod problema čije je rješenje neka vrsta redoslijeda (npr. gradova, zadataka) – prikaz permutiranim nizom:

(0 , 3 , 5 , 2 , 1 , 6)

U ovom prikazu postoji 6 gradova koje treba obići, a redoslijed obilazaka je sljedeći: kreće se iz grada 0, pa u grad 3, nakon njega 5, itd. Dakle, mjesto u šestorki označava kada će taj grad biti posjećen, a broj koji je na tom mjestu označava sam grad. Ukratko, gradovi su poredani u smjeru u kojem se posjećuju. Ovaj način prikaza je uobičajen u implementaciji problema trgovačkog putnika.

U programu je za prikaz kromosoma korištena klasa `Kromosom.cs`, čiji se kod može vidjeti u Dodatku B.

4.1.2 Funkcija dobrote

Funkcija dobrote jednaka je jednostavnoj formuli koja zbraja udaljenosti između gradova, dajući tako za svaki kromosom ukupnu duljinu puta. Naravno, što je duljina puta kraća, kromosom je bolji i ima veću vjerojatnost preživljavanja.

4.1.3 Operatori selekcije

Selekcija se obavlja tako da se odaberu dva proizvoljna grada i formiraju roditelji u odnosu na grad prije prvog odabranog grada i drugi roditelj – u odnosu na drugi grad i grad koji ide poslije drugog odabranog grada.

4.1.4 Operatori mutacije

U algoritmu su korištene tri vrste mutacija.

Jednostavna (slučajna) mutacija

Slučajnim odabirom odabiru se dva različita grada u kromosomu (odabiru se dva grada da bi se izbjeglo ponavljanje brojeva u nizu, tj. da bi postigli raznolikost), nakon čega se obavlja njihova zamjena. Npr. (2 3 4 1 5 8 6 7) postaje (2 5 4 1 3 8 6 7).

Mutacija Gaussovom (normalnom) razdiobom

Ideja mutacije je sljedeća: slučajnim odabirom odabrati jedan grad, a drugi grad s kojim će se ovaj zamijeniti izabrati tako da veću vjerojatnost odabira ima grad koji se u kromosomu nalazi na poziciji bližoj prvom odabranom gradu.

Postupak je dan u točkama koje slijede:

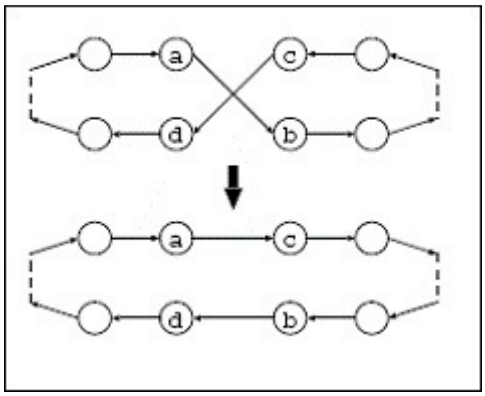
1. Prije početka izvođenja algoritma radi se matrica vjerojatnosti A.
2. Slučajnim odabirom izabire se jedan grad u kromosomu i pamti njegova pozicija
3. Odredi se za koliko se maksimalno mjesta u kromosomu može pomaknuti od pozicije prvog odabranog grada.
4. Računa se koji redak matrice će biti potreban za određivanje drugog grada prema sljedećoj formuli:
5. redak=Udaljenost-[(brojGradova/2)-1]
6. Slučajnim odabirom generira se broj q iz intervala [0,1], i traži se za koji stupac, odnosno j vrijedi: $q \in \langle a_{redak,j-1}, a_{redak,j} \rangle$. Redni broj stupca, odnosno j označava za koliko mjesta u kromosomu se treba pomaknuti u lijevu ili desnu stranu kako bi se odabrao drugi grad. Ovim postupkom dobit će se dva moguća grada s kojima bi se prvi odabrani grad mogao zamijeniti. Napravit će se zamjene za oba slučaja, a na kraju će se uzeti rješenje s boljom dobrotom.

Čitav kod ovog postupka se može vidjeti u dodatku II.

2opt mutacija ili mutacija „kraći put“

2opt metoda jedna je od najpoznatijih metoda lokanog pretraživanja koje se koriste kod problema trgovačkog putnika. Unapređuje turu brid po brid okrećući poredak gradova u podturi. Algoritam je prikazan na slici 10., a može se opisati jednostavnim primjerom. Neka

postoji put od grada A do grada B, i put od grada C do grada D. Uspoređuje se nejednakost AB + CD > AC + BD. Ako je istina, dolazi do zamjene kako je prikazano na slici. Postupak se ponavlja za sve bridove dok je god moguće skratiti rutu. Dakle, ova metoda odabire neku stazu unutar grafa i okreće poredak gradova u podstazi ako se time dobiva kraća staza. Veliki problem 2opt metode je taj što može zapeti u nekom lokalnom optimumu iz kojega, zbog načina usporedbe, više ne može izaći. No, uz pravilno odabrane operatore križanja, koji će unijeti dovoljnu raznolikost, ova mutacija postaje najefikasnija. U dodatku II, prikazan je kod 2opt mutacije.



Slika 10. 2opt metoda [www.zemris.fer.hr/~golub/]

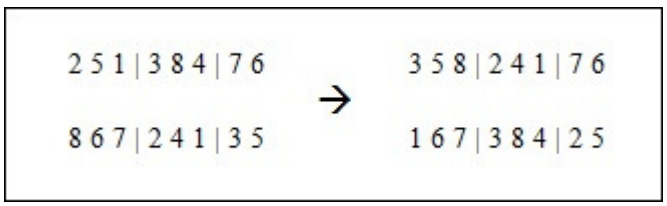
4.1.5 Operatori križanja

U strategijama (μ/ ρ, λ)-ES i (μ/ ρ + λ)-ES koristi se operator križanja. Vrste križanja koje su implementirane u algoritmu su sljedeće:

- PMX križanje (engl. *Partially Matched Crossover*)
- GX križanje (engl. *Greedy Crossover*)
- GSX križanje (engl. *Greedy Subtour Crossover*)

PMX križanje

U oba roditelja označe se dvije točke prekida (u našem primjeru točke prekida su označene znakom ' | '. Geni između tih točaka zamijene se između roditelja. Ostatak kromosoma se popunjava tako da se gradovi izvan točaka prekida vraćaju na svoje mjesto ukoliko već ne postoje kao rezultat zamjene. Ako grad već postoji na nekom mjestu između točaka prekida, umjesto njega se upisuje onaj grad kojeg je zamijenio novi grad.



Slika 11. Primjer PMX križanja

U primjeru na slici 11. geni između točaka prekida zamjene mjesta. Dakle iz prvog roditelja 3 se zamijeni sa 2, dok se u drugom roditelju 2 zamijeni sa 4, i tako za ostala 2 gena. Sada se puni ostatak prvog kromosoma. Na prvoj poziciji više ne može biti broj 2 jer se on već nalazi na poziciji 4, pa se iz tog razloga umjesto broja 2 upisuje broj koji je prije bio na poziciji 4, a to je broj 3. Na sljedećoj poziciji je bio broj 5, a pošto nije iskorišten u zamjeni, slobodno ga se piše natrag na poziciju 2. Za broj 1 je isti slučaj kao na poziciji 1. Umjesto tog broja bi trebalo upisati broj 4, no on je već upisan, te se umjesto njega upisuje broj 8. Nadalje, brojevi 7 i 6 nisu iskorišteni u zamjeni pa ih se upisuje na njihovo staro mjesto u kromosomu. Isti postupak se izvodi za drugi kromosom.

Kod PMX križanja se također može vidjeti u dodatku II.

GX križanje

Greedy crossover uzima prvi grad iz jednog roditelja, uspoređuje gradove u koje se dolazi iz tog grada u oba roditelja i uzima onog čiji je put kraći. Ako se jedan grad u djetetu već pojavio, onda se uzima drugi. Ako su se oba grada već pojavila, tada se slučajnim odabirom odabire jedan neodabrani grad. Postupak je vrlo jednostavan, a kod se nalazi u dodatku II.

GSX križanje

Ovo križanje radi tako da iz oba roditelja uzima što je moguće dulji dio genetskog materijala, tj. što dulji podskup gradova. Pseudokod je prikazan na slici.

```
Ulaz: kromosom k = (a0, a1, ..., an-1) i k = (b0, b1, ..., bn-1)
       Izlaz: dijete k
       procedura krizaj (ka, kb) {
              fa <- true
              fb <- true
              izaberi jedan slucajan grad g
              izaberi x gdje je ax = t
              izaberi y gdje je by = t
              k <- t
              čini{
                     x = (x - 1) mod n
                     y = (y - 1) mod n
                     ako fa = true onda{
                            ako ax nije u k onda
                                   k <- ax * k
                            inače
                                   fa <- false
                     }
                     ako fb = true onda{
                            ako by nije u k onda
                                   k <- k * by
                            inače
                                   fb <- false
                     }

              } dok fa = true ili fb = true
              ako staza nije potpuna onda
                     dodaj ostale gradove slucajnim redoslijedom u k
              vrati k
       }
```

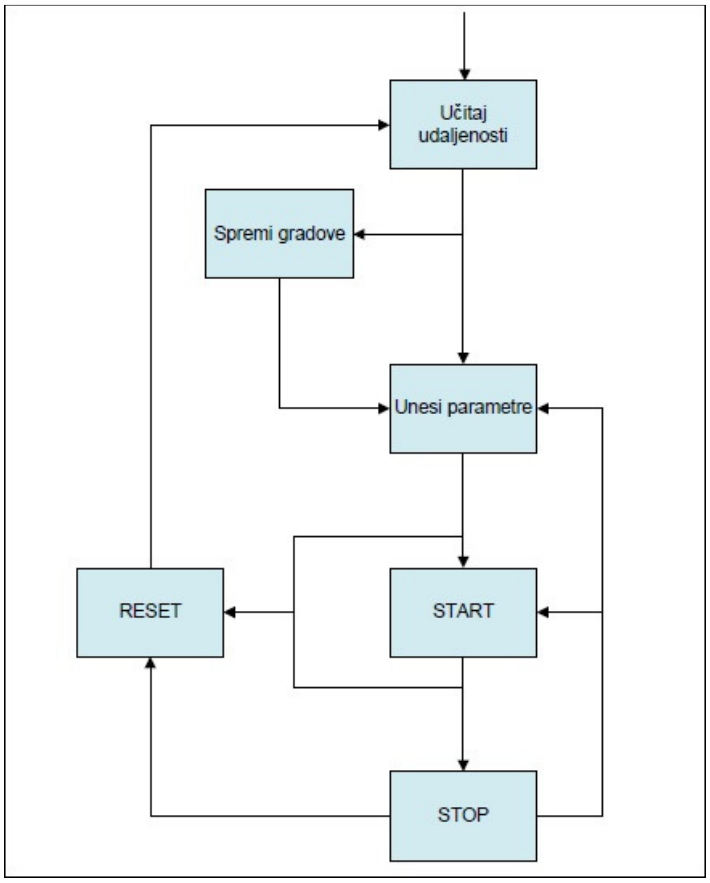
Slika 12. Pseudokod GSX križanja

Na slici n je broj gradova, a * je znak konkatencije (lijepljenja) nizova. GSX križanje je najefikasnija vrsta križanja u odnosu na ovdje opisane vrste.

4.2 Rad s aplikacijom

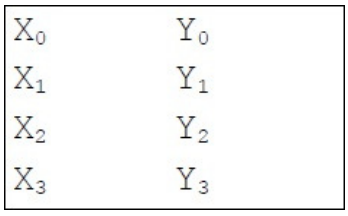
Aplikacija je izrađena u okruženju Microsoft Visual Studio 2010, u programskom jeziku C#.

Tijek rada aplikacije prikazan je na slici 13.



Slika 13. Tijek izvođenja aplikacije

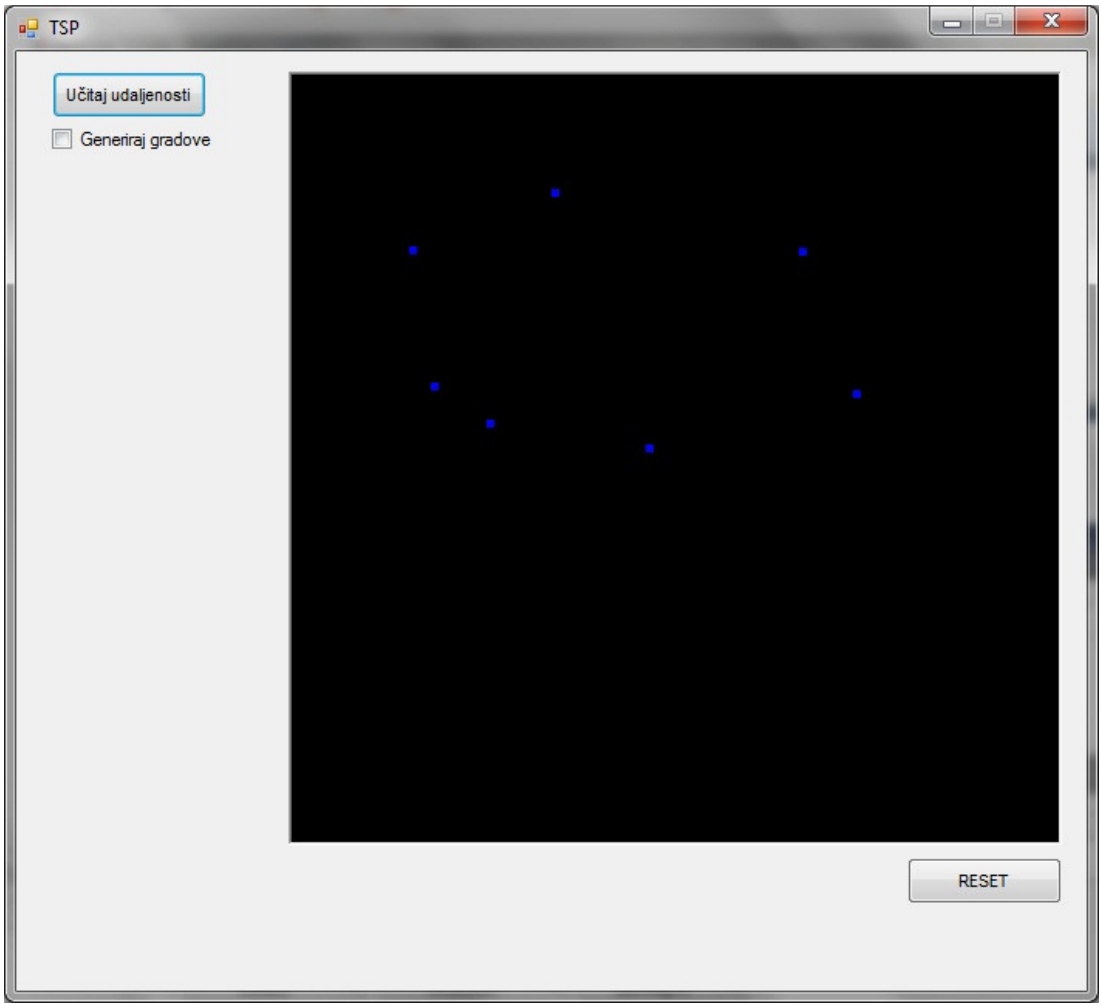
Učitavanje udaljenosti moguće je na dva načina. Korisnik može učitati koordinate točaka iz datoteke - pritiskom tipke *Učitaj udaljenosti*, otvara se dijalog za otvaranje datoteke čiji zapis treba izgledati ovako:



Slika 14. Format zapisa točaka u datoteci

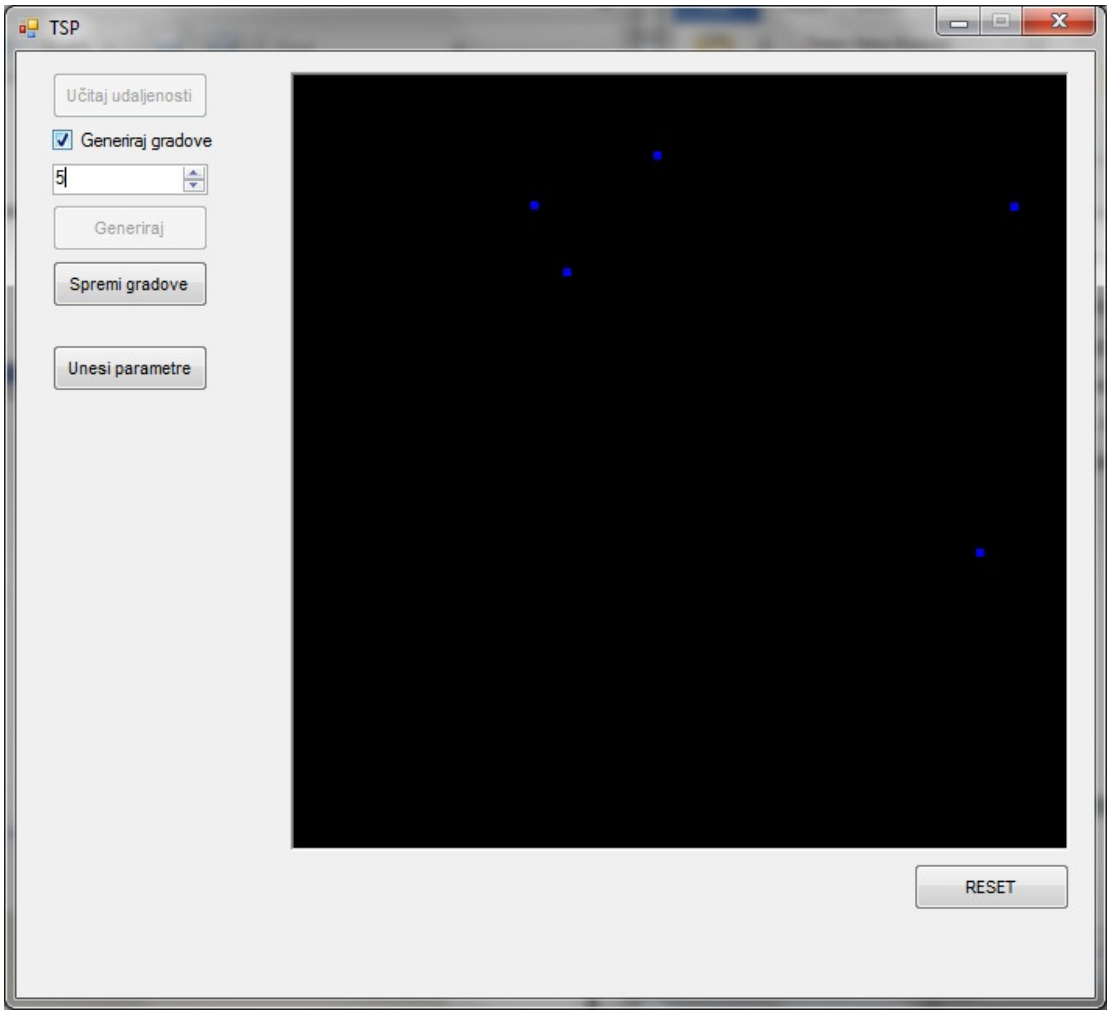
X_i i Y_i predstavljaju x, odnosno y koordinatu točke i. Koordinate x i y moraju biti cjelobrojne nenegativne vrijednosti i u zapisu moraju biti razmaknute tabulatorom.

Drugi način učitavanje točaka je postavljanje točaka na panelu za crtanje putem miša. Pritiskom tipke *Učitaj udaljenosti*, program učita koordinate postavljenih točaka.



Slika 15. Izgled sučelja – unos točaka putem panela za crtanje

Osim ovakvog načina zadavanja problema, korisnik može odabrati opciju da program samostalno generira problem zadan brojem gradova – stavljanjem kvačice na *Generiraj gradove*, dobiva se opcija za odabir broja gradova. Pritiskom tipke *Generiraj* stvara se zadani broj točaka. Program će slučajnim odabirom koordinata unutar dimenzija panela za crtanje odabrati koordinate točaka i prikazati ih na panelu (slika 16.).

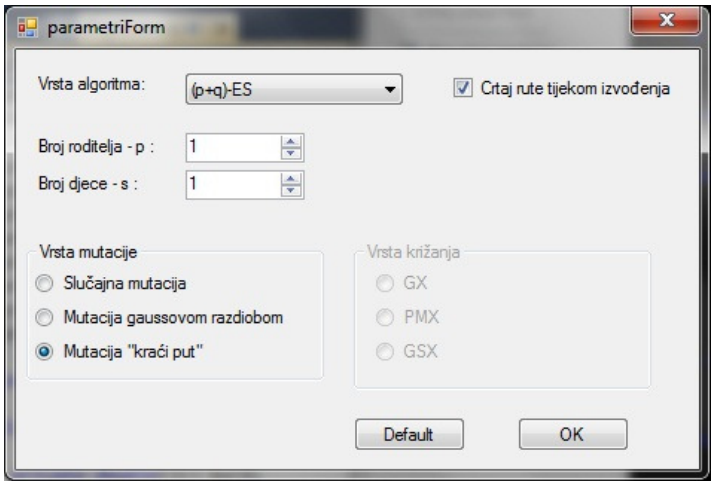


Slika 16. Sučelje nakon generiranja 5 točaka

Za slučajeve učitavanja udaljenosti putem miša (preko panela) ili slučajnim generiranjem, postoji mogućnost spremanja koordinata nastalih točaka, pritiskom tipke *Spremi gradove*.

Nakon učitavanja udaljenosti, slijedi zadavanje parametara. Pritiskom tipke Unesi parametre otvara se dijalog za izbor parametara (slika 17.). Moguće je odabrati postavke za sljedeće parametre:

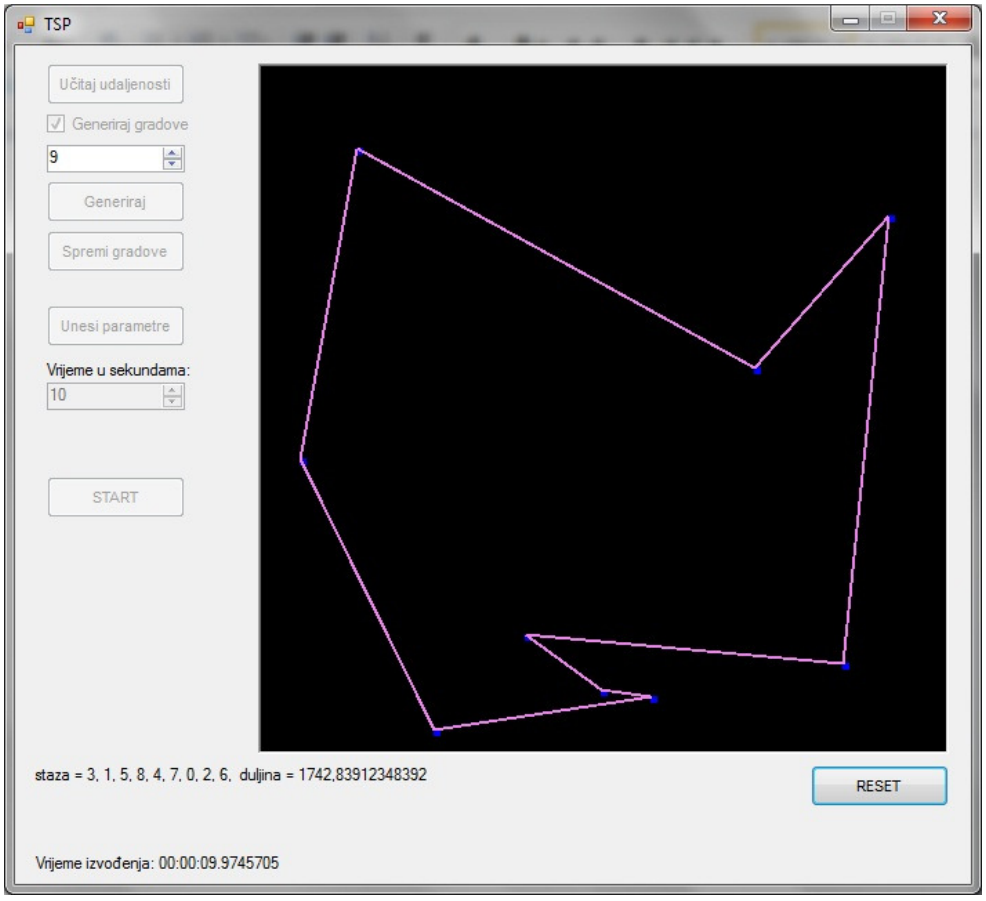
- Vrsta algoritma ((1+1) – ES, (μ, λ) – ES, (μ + λ) – ES, (μ + 1) – ES, (μ/2, λ) – ES, (μ/2 + λ) – ES)
- Broj roditelja i djece (ovisi o odabranom algoritmu)
- Vrsta mutacije (jednostavna mutacija, mutacija s normalnom razdiobom, pohlepna mutacija (2opt mutacija))
- Vrsta križanja – za strategije (μ/2, λ) – ES i (μ/2 + λ) – ES: GX, PMX i GSX
- Uvjet zaustavljanja: vremensko ograničenje, ukupni broj generacija, broj generacija sa nepromijenjenom najboljom jedinkom
- Mogućnost crtanja ruta tijekom izvođenja programa (usporava program)



Slika 17. Dijalog za unos parametara

Pritiskom tipke *Default* postavljaju se predviđeni parametri. Nakon pritiska tipke *OK*, dijalog se zatvara i fokus se vraća na glavni prozor. Sada je moguće pokrenuti program pritiskom tipke *START*.

Tijekom izvođenja programa, u lijevom dijelu prozora ispod panela za crtanje, nakon završetka izvođenja se ispisuje vrijeme izvođenja, te najbolja ruta i njen put (slika 18.). Najbolja ruta se zbog preglednosti ispisuje samo ako je broj gradova manji ili jednak 100.



Slika 18. Izgled sučelja nakon završetka izvođenja

Program se zaustavlja nakon što je zadovoljen uvjet zaustavljanja zadan u parametrima, ili pritiskom tipke *STOP*. Pritiskom tipke *RESET*, parametri i točke na panelu se brišu, te je moguće zadati novi problem i unijeti nove parametre za ponovno pokretanje programa.

4.3 Rezultati eksperimentiranja

Računalo na kojem je testirana aplikacije je Intel(R) Core(TM) i3 2.40GHz s 3.00 GB RAM-a. Eksperimenti su izvođeni na problemu od 200 gradova za koji postoji 1.97e+372 mogućih rješenja. Ako pretpostavimo da za stvaranje jedne staze od 200 gradova treba stotinka milisekunde, tada bi nam za pretraživanje cjelokupnog prostora rješenja trebalo 6.25e+359 godina.

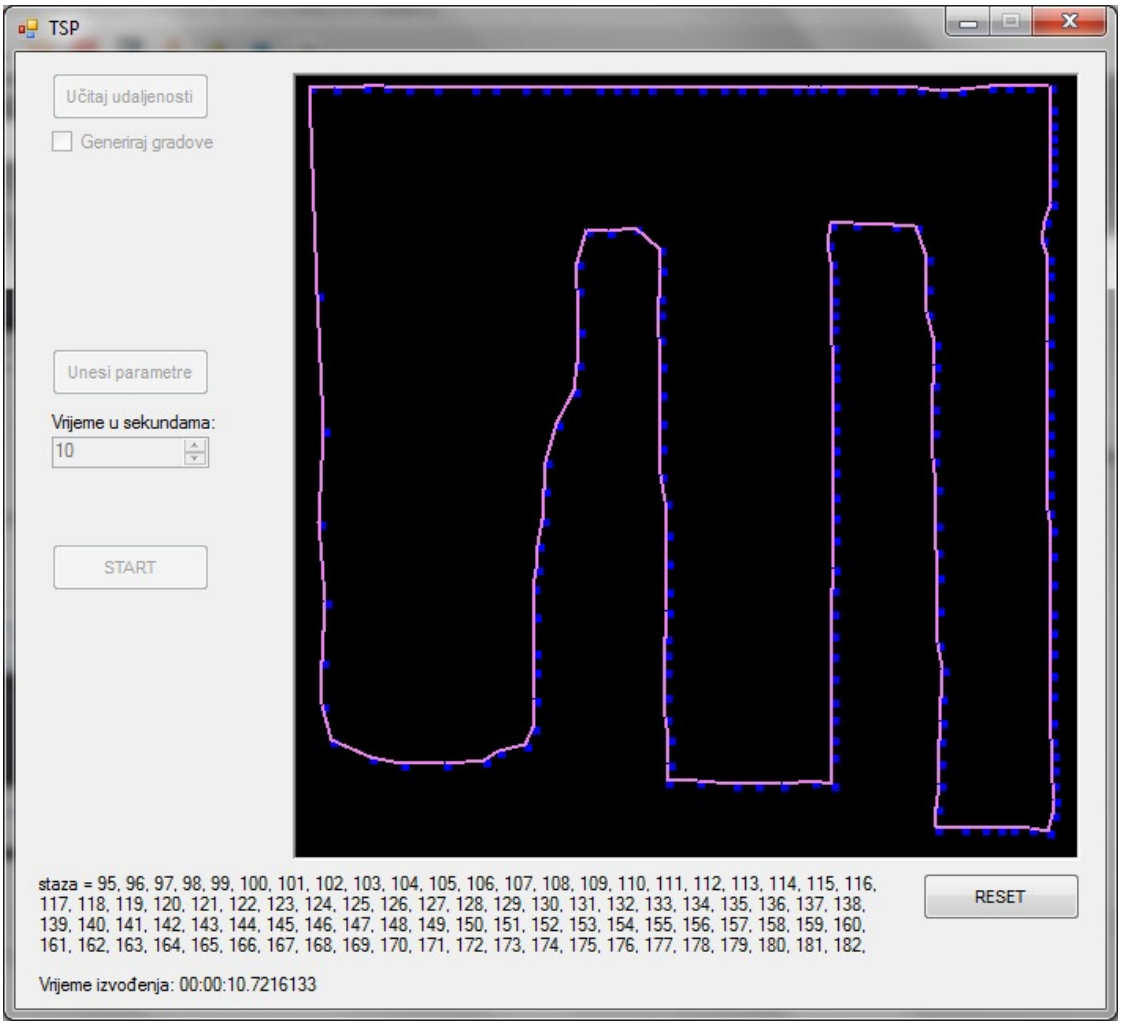
4.3.1 Rezultati i usporedbe rada genetskih operatora

Razmotrili smo sve rezultate kombinacija pojedinih strategija i genetskih operatora.

Od testiranih strategija koje koriste isključivo mutaciju, najbolja se pokazala (μ + λ) –ES, dok je najlošija bila (μ, λ) – ES. Kod (μ, λ) strategije se prilikom povećavanja broja jedinki uz konstantan omjer jedinki i roditelja poboljšava rješenje. Strategija (μ + λ) najbolje rezultate daje uz što manji broj jedinki. Promjena omjera jedinki roditelja i djece nije pokazala nikakve promjene u rezultatima. (μ + 1) strategija se pokazala lošijom od (μ + λ) strategije.

Od strategija koje koriste križanje najbolja se pokazala (μ/2, λ) strategija. Za GX križanje i jednostavnu mutaciju rješenje je ovisilo o ukupnom broju jedinki kako kod zarez tako i kod plus strategije, dok kod svih ostalih kombinacija križanja i mutacija takva ovisnost nije bila izražena.

Najbolja metoda koja je gotovo uvijek došla do optimalnog rješenja je 2opt mutacija. Do točnog rješenja se uspjelo doći korištenjem mutacije 2opt i bilo koje vrste strategije, dok je god ukupni broj jedinki u populaciji veći od 2. Budući da 2opt metoda može zapeti u lokalnom minimumu, korištenje većeg broja jedinki umanjuje taj nedostatak.



Slika 19. Izgled rješenja: (2+6)-ES 2OPT mutacijom (mutacijom kraćeg puta)

Eksperimenti su pokazali od kolike je važnosti ispravan odabir genetskih operatora i broja jedinka u populaciji. Pri odabiru broja jedinki u populaciji treba uzeti u obzir što je više jedinki to će algoritam napraviti manje iteracija u konstantnom vremenu. No, veći broj jedinki proširuje prostor pretraživanja i tako omogućava brže nalaženje optimuma. Ako se ovi parametri ispravno odrede, algoritam daje izvrsne rezultate.

5 Zaključak

Već kroz opise primjena kod problema trgovačkog putnika lako je vidjeti kako je njegova primjena vrlo široka i kako postoje razne verzije tog problema. Evolucijske strategije su zapravo samo princip, ideja, smjernica kako taj problem rješiti na drukčiji način od klasičnih metoda, jer sve je na korisniku da sam odluči, da li će razvijati svoj vlastiti algoritam ili će probati svoj problem prilagoditi već nekom postojećem algoritmu koji rješava neku sličnu klasu problema.

Također, vidi se da su evolucijske strategije, korisne za one klase problema koje se ne mogu rješiti na klasične načine zbog veličine područja koje pretražuju, što smo uočili i na našem primjeru trgovačkog putnika za dvjesto gradova uz uporabu potrebnih osnovnih operatora – selekcije i mutacije.

Dobiveni rezultati kod uporabe 2opt mutacije su više nego zadovoljavajući. Donekle zadovoljavajuće rezultate davala je (μ/2, λ) strategija uz korištenje GX križanja i jednostavne mutacije, te nešto većeg broja jedinki roditelja i djece. Ukoliko nepromišljeno biramo operatore, algoritam će najvjerojatnije završiti rad u nekom lokalnom optimumu, bliže ili dalje od pravog optimuma, ovisno o parametrima.

6 Literatura

[ČUPIĆ, 2009]

Čupić, Marko (2009) - Prirodom inspirirani optimizacijski algoritmi, Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu (http://www.fer.unizg.hr/_download/repository/Cupic2009PrirodomInspiriraniOptimizacijskiAlgoritmi.pdf)

[DRAGOJEVIĆ, 2008]

Dragojević, Ognjen (2008) – Semantičko genetsko programiranje, Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu

[BESPALJKO, 2009]

Bespaljko, Igor (2009) – Izvođenje pravila raspoređivanja uz pomog genetskog programiranja, Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu

[PIELIĆ, 2010]

Pielić, Marko (2010) – Učinkovitost automatski definiranih funkcija u evolucijskim algoritmima, Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu

[GOLUB, 2004]

Golub, Marin (2004) – Genetski algoritam (Prvi dio) – Jednostavni genetski algoritam, Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu (http://www.zemris.fer.hr/~golub/ga/ga_skripta1.pdf)

[BOŽIKOVIĆ, 2000]

Božiković, Marko (2000) – Globalni paralelni genetski algoritam, Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu

[FABRIS, 2005]

Fabris, Ognjen (2005) – Primjena genetskih algoritama za raspoređivanje poslova u višeprocesorskom sustavu, Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu

[PETROVIĆ, 2007]

Petrović, Juraj (2007) – Evolucijski algoritmi, Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu

[ČERI, MALOVIĆ, 2007]

Čeri, Malović (2007) – Evolucijske strategije, Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu (http://www.zemris.fer.hr/~golub/ga/studenti/projekt2007/es/PR_Ceri_Malovic.pdf)

[PIELIĆ, 2008]

Pielić, Marko (2008) – Rješavanje problema trgovačkog putnika uz pomoć genetskih algoritama, Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu

[VELJAN, 2001]

Veljan (2001), Kombinatorna i diskretna matematika, Algoritam, Zagreb 2001.

[GUTIN, PUNNEN, 2004]

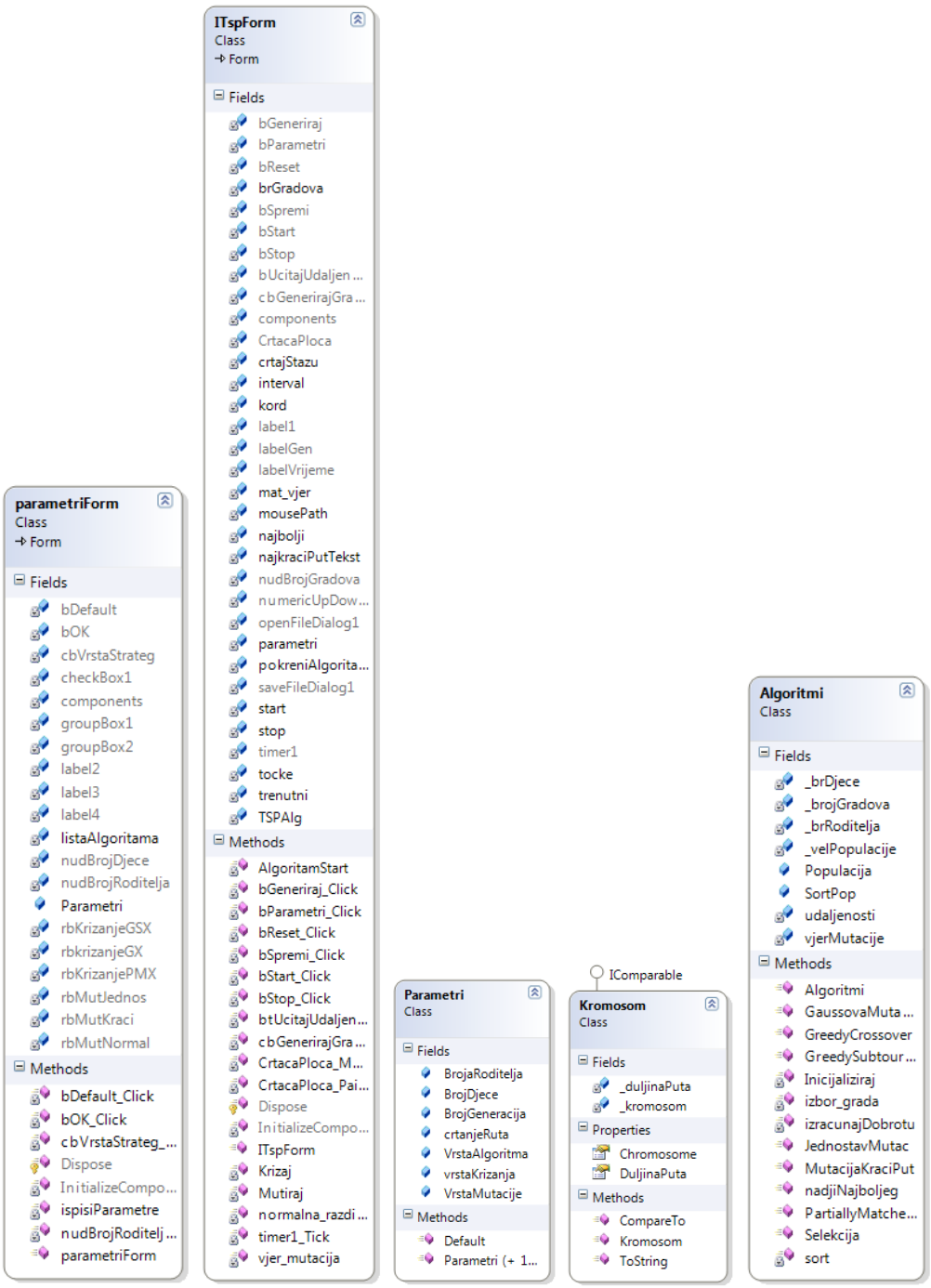
Gutin, Gregory, Punnen, Abraham P., – The Traveling Salesman Problem and Its Variations", Kluwer Academic Publishers, 2004

[HJERTENES, 2002]

Øystein M. Hjertenes – A Multilevel Scheme for the Travelling Salesman Problem, University of Bergen, 2002

7 Prilozi

7.1 Dodatak I – dijagrami klasa



7.2 Dodatak II – dijelovi programskog koda

7.2.1 Klasa Kromosom

```
usingSystem;
usingSystem.Collections;
using System.Collections.Generic;
usingSystem.Text;

namespace TSP_Testing
{
/// Klasa Kromosom definira jedan kromosom, stazu i duljinu puta
publicclassKromosom:IComparable
    {
privateArrayList _kromosom = newArrayList();
privatedouble _duljinaPuta;

public Kromosom(ArrayList jedinka, double duljina)
        {
foreach (int i in jedinka)
            {
                _kromosom.Add(i);
            }
            _duljinaPuta = duljina;
        }
/// Svojstvo koje vraca stazu
publicArrayListChromosome
        {
get
            {
return _kromosom;
            }
        }
/// Svojstvo koje vraca ili postavlja duljinu puta kromosoma
publicdoubleDuljinaPuta
        {
get
            {
return _duljinaPuta;
            }
set
            {
                _duljinaPuta = value;
            }
        }

publicoverridestringToString()
        {
StringBuilder staza = newStringBuilder();

foreach (int i in _kromosom)
            {
staza.AppendFormat("{0}, ", i);
            }
return"staza = " + staza.ToString() + " " + "duljina = " + this._duljinaPuta.ToString();
        }

        #regionIComparableMembers

publicintCompareTo(objectobj)
        {
double d1 = this._duljinaPuta;
double d2 = ((Kromosom)obj)._duljinaPuta;
return d2.CompareTo(d1);
        }
        #endregion
    }
}
```

7.2.2 Mutacija 2opt

```
//2opt mutacija
publicKromosomMutacijaKraciPut(KromosomzaMutirati)
        {
ArrayList elementi = newArrayList();
foreach (int i inzaMutirati.Chromosome)
            {
elementi.Add(i);
            }
Randomrnd = newRandom();
int gen1 = rnd.Next(this._brojGradova);
int n=1;
double put;
intprevious;
do
            {
if (gen1 == 0)
                {
previous = elementi.Count - 1;
                    put = udaljenosti[(int)elementi[previous]][(int)elementi[gen1]];
                }
else
                {
previous = gen1 - 1;
                    put = udaljenosti[(int)elementi[previous]][(int)elementi[gen1]];
                }
List<int> kraci = newList<int>();
foreach (int i in elementi)
                {
if ((udaljenosti[(int)elementi[previous]][i] < put) && ((int)elementi[previous] != i))
                    {
kraci.Add(elementi.IndexOf(i));
                    }
                }
if ((kraci.Count == 0) && (n < _brojGradova))
                {
                    n++;
                    gen1 = (gen1+1)%(_brojGradova-1);
                }
elseif (n >= _brojGradova)
                {
int gen2 = rnd.Next(_brojGradova - 1);
inttemp = (int)elementi[gen1];
                    elementi[gen1] = elementi[gen2];
                    elementi[gen2] = temp;
KromosommutKromosom = newKromosom(elementi, this.izracunajDobrotu(elementi));
returnmutKromosom;
                }
else
                {
int gen2 = rnd.Next(kraci.Count);
                    gen2 = kraci[gen2];
kraci.Clear();
inttemp = (int)elementi[gen1];
                    elementi[gen1] = elementi[gen2];
                    elementi[gen2] = temp;
KromosommutKromosom = newKromosom(elementi, this.izracunajDobrotu(elementi));
returnmutKromosom;
                }
            } while (true);
        }
```

7.2.3 Mutacija normalnom razdiobom

```
//Gaussova mutacija (mutacija normalnom razdiobom)
publicKromosomGaussovaMutacija(KromosomzaMutirati)
        {
ArrayList elementi = newArrayList();
foreach (int i inzaMutirati.Chromosome)
            {
elementi.Add(i);
            }
Randomrnd = newRandom();
int gen1 = rnd.Next(this._brojGradova); //izabran prvi grad
int x = _brojGradova - gen1 - 1;
int y = gen1;
int d = Math.Max(x, y);

int gen2 = izbor_grada(d)+1; //udaljenost pozicije drugog grada

if ((gen2 + gen1) < _brojGradova)
            {
//zamjena gena na poziciji gen1 i gena na poziciji gen1+gen2
inttmp = (int)elementi[gen1];
                elementi[gen1] = elementi[gen2+gen1];
                elementi[gen2+gen1] = tmp;
            }
elseif ((gen1 - gen2) >= 0)
            {
//zamjena gena na poziciji gen1 i gena na poziciji gen1-gen2
inttmp = (int)elementi[gen1];
                elementi[gen1] = elementi[gen1 - gen2];
                elementi[gen1 - gen2] = tmp;
            }
KromosommutKromosom = newKromosom(elementi, this.izracunajDobrotu(elementi));
returnmutKromosom;
        }
```

7.2.4 GX križanje

```
//GX križanje
publicKromosomGreedyCrossover(Kromosom rod1, Kromosom rod2)
        {
ArrayList elementi = newArrayList();
ArrayList stazaRod1 = newArrayList();
ArrayList stazaRod2 = newArrayList();

            stazaRod1 = rod1.Chromosome;
            stazaRod2 = rod2.Chromosome;
elementi.Add(stazaRod1[0]);
Randomrnd = newRandom();

for (int i = 1; i < stazaRod1.Count; i++)
            {
int ind1 = (stazaRod1.IndexOf(elementi[i - 1]) + 1) % this._brojGradova;
int ind2 = (stazaRod2.IndexOf(elementi[i - 1]) + 1) % this._brojGradova;
int ind0 = (int)elementi[i - 1];

if ((elementi.IndexOf(stazaRod1[ind1]) != -1) && (elementi.IndexOf(stazaRod2[ind2]) != -1))
                {
int novi;
do
                    {
                        novi = rnd.Next(this._brojGradova);
                    } while (elementi.IndexOf(novi) != -1);
elementi.Add(novi);
continue;
                }
if (elementi.IndexOf(stazaRod1[ind1]) != -1)
                {
elementi.Add(stazaRod2[ind2]);
```

```
continue;
                }
if (elementi.IndexOf(stazaRod2[ind2]) != -1)
                {
elementi.Add(stazaRod1[ind1]);
continue;
                }
if (udaljenosti[ind0][(int)stazaRod1[ind1]] <= udaljenosti[ind0][(int)stazaRod2[ind2]])
                {
elementi.Add(stazaRod1[ind1]);
continue;
                }
if (udaljenosti[ind0][(int)stazaRod1[ind1]] > udaljenosti[ind0][(int)stazaRod2[ind2]])
                {
elementi.Add(stazaRod2[ind2]);
continue;
                }
            }
Kromosom dijete = newKromosom(elementi, this.izracunajDobrotu(elementi));
return dijete;
        }
```

7.2.5 PMX križanje

```
//PMX križanje
publicKromosomPartiallyMatchedCrossover(Kromosom rod1, Kromosom rod2, int tocka1, int tocka2)
        {
ArrayList elementi1 = newArrayList();
ArrayList elementi2 = newArrayList();

for (int i = 0; i <this._brojGradova; i++)
            {
                elementi1.Add(null);
                elementi2.Add(null);
            }
//dodavanje dijelova izmedjutocaka prekida
for (int i = tocka1 + 1; i <= tocka2; i++)
            {
                elementi2[i] = rod1.Chromosome[i];
                elementi1[i] = rod2.Chromosome[i];
            }
//generiranje ostatka kromosoma prema definiciji PMX krizanje
for (int i = 0; i <this._brojGradova; i++)
            {
if ((i > tocka1) && (i <= tocka2))
                {
continue;
                }
if (elementi1.Contains(rod1.Chromosome[i]))
                {
intindex = rod2.Chromosome.IndexOf(rod1.Chromosome[i]);
if (elementi1.Contains(rod1.Chromosome[index]))
                    {
index = rod2.Chromosome.IndexOf(rod1.Chromosome[index]);
                        elementi1[i] = rod1.Chromosome[index];
                    }
else
                    {
                        elementi1[i] = rod1.Chromosome[index];
                    }
                }
else
                {
                    elementi1[i] = (int)rod1.Chromosome[i];
                }
if (elementi2.Contains(rod2.Chromosome[i]))
                {
intindex = rod1.Chromosome.IndexOf(rod2.Chromosome[i]);
if (elementi2.Contains(rod2.Chromosome[index]))
                    {
index = rod1.Chromosome.IndexOf(rod2.Chromosome[index]);
```

```
                    elementi2[i] = rod2.Chromosome[index];
                }
else
                {
                    elementi2[i] = rod2.Chromosome[index];
                }
            }
else
            {
                elementi2[i] = (int)rod2.Chromosome[i];
            }
        }
Kromosom dijete1 = newKromosom(elementi1, this.izracunajDobrotu(elementi1));
Kromosom dijete2 = newKromosom(elementi2, this.izracunajDobrotu(elementi2));
//vracanje boljeg djeteta
if (dijete1.DuljinaPuta < dijete2.DuljinaPuta)
        {
return dijete1;
        }
return dijete2;
    }
```

7.2.6 GSX križanje

```
//GSX križanje
publicKromosomGreedySubtourCrossover(Kromosom rod1, Kromosom rod2)
    {
ArrayList elementi = newArrayList();
bool fa = true, fb = true;
Randomrnd = newRandom();
intsluc = rnd.Next(this._brojGradova);
int indR1 = rod1.Chromosome.IndexOf(sluc);
int indR2 = rod2.Chromosome.IndexOf(sluc);
elementi.Add(sluc);

do
        {
if (indR1 == 0)
            {
                indR1 = 8;
            }
            indR1 = (indR1 - 1) % this._brojGradova;
            indR2 = (indR2 + 1) % this._brojGradova;
if (fa)
            {
if (elementi.Contains(rod1.Chromosome[indR1]))
                {
                    fa = false;
                }
else
                {
elementi.Add(null);
for (int i = elementi.Count - 1; i > 0; i--)
                    {
                        elementi[i] = elementi[i - 1];
                    }
                    elementi[0] = rod1.Chromosome[indR1];
                }
            }
if (fb)
            {
if (elementi.Contains(rod2.Chromosome[indR2]))
                {
fb = false;
                }
else
                {
elementi.Add(rod2.Chromosome[indR2]);
                }
            }
        } while ((fa) || (fb));
```

```
while (elementi.Count<this._brojGradova)
            {
sluc = rnd.Next(this._brojGradova);
if (elementi.Contains(sluc))
                {
continue;
                }
else
                {
elementi.Add(sluc);
                }
            }
Kromosom dijete = newKromosom(elementi, this.izracunajDobrotu(elementi));
return dijete;
        }
```