Pronalaženje optimalnog ili najbržeg puta pomoću genetskog algoritma

Ognjen Zelenbabić

1. Definicija problema

Problem: Izabrati mapu jedne (stvarne ili virtuelne) države i definisati lokaciju većih gradova u njoj. Definisati rastojanja između pojedinih gradova, postojanje puteva, njihovih dužina i kvaliteta. Definisati kriterijumsku funkciju kao funkciju više promenljivih koja će uzeti u obzir postojanje, dužinu i kvalitet puta na odgovarajući način. Primenom genetskog algoritma odrediti optimalnu trajektoriju koja povezuje dva proizvoljna grada na usvojenoj mapi.

2. Genetski algoritmi

Genetski algoritam je heuristička metoda koja oponaša prirodni proces evolucije. Ovaj algoritam se uglavnom koristi za probleme pretraživanja i optimizacije a pripada većoj klasi takozvanih evolutivnih algoritama. GA (Genetski Algoritmi) su u svojoj osnovi zasnovani na populaciji horomozoma koji su najčešće predstavljeni nizovima binarnih cifara. Oni rešavaju problem pronalaska dobrih hromozoma manipulišući materijalom unutar hromozoma, naslepo, ne znajući ništa o tipu problema koji se rešava.

Jedinu informaciju koju GA poseduju je evaluacija svakog hromozoma koji proizvode. Tu evaluaciju koriste da usmere selekciju hromozoma tako da se oni hromozomi sa najboljom evaluacijom reprodukuju mnogo češće od onih sa lošom evaluacijom.

Koraci od kojih se sastoji svaki GA su:

Korak 1: Inicijalizuje se populacija hromozoma;
Korak 2: Proceni se svaki hromozom u populaciji;
Korak 3: Stvaraju se novi hromozomi parenjem trenutnih hromozoma nakon čega se primenjuje mutacija i rekombinacija (prilikom parenja roditeljskih hromozoma);
Korak 4: Brišu se određeni članovi populacije kako bi se napravilo mesta za nove hromozome;
Korak 5: Procenjuju se novi hromozomi i ubacuju u populaciju;
Korak 6: Ukoliko je kriterijum zaustavljanja zadovoljen tada stajemo i „vraćamo“ najbolji hromozom, ukoliko nije, tada idemo ponovo na korak 3.

Ne postoji jedan GA koji predstavlja univerzalno rešenje optimizacionih problema ili problema pretraživanja, već se mehanizmi genetskih algoritama prilagođavaju trenutnom problemu. Mehanizmi enkodiranja i funkcija evaluacije stvaraju vezu između GA i specifičnog problema koji se rešava.

Evaluacione funkcije imaju istu ulogu u GA kao što ima okolina u procesu prirodne evolucije.

Svaki niz predstavlja tačku u prostoru pretrage a samim tim i moguće rešenje problema. Nizovi su dekodirani od strane evaluatora kako bi dobili objektivnu vrednost funkcije individualne tačke u prostoru pretraživanja. Dobijena vrednost funkcije je tada konvertovana u vrednost fitnesa koja određuje verovatnoću da na tu individuu deluju genetski operatori. Populacija tada evoluira od generacije do generacije kroz upotrebu genetskih operatora. Ukupan broj nizova koji čine populaciju se ne menja kroz promene generacija.

GA u svojoj najprostijoj formi koristi tri operatora:

1. reprodukciju,
2. krosover (ukrštanje) i
3. mutaciju.

3. Pristup rešavanju

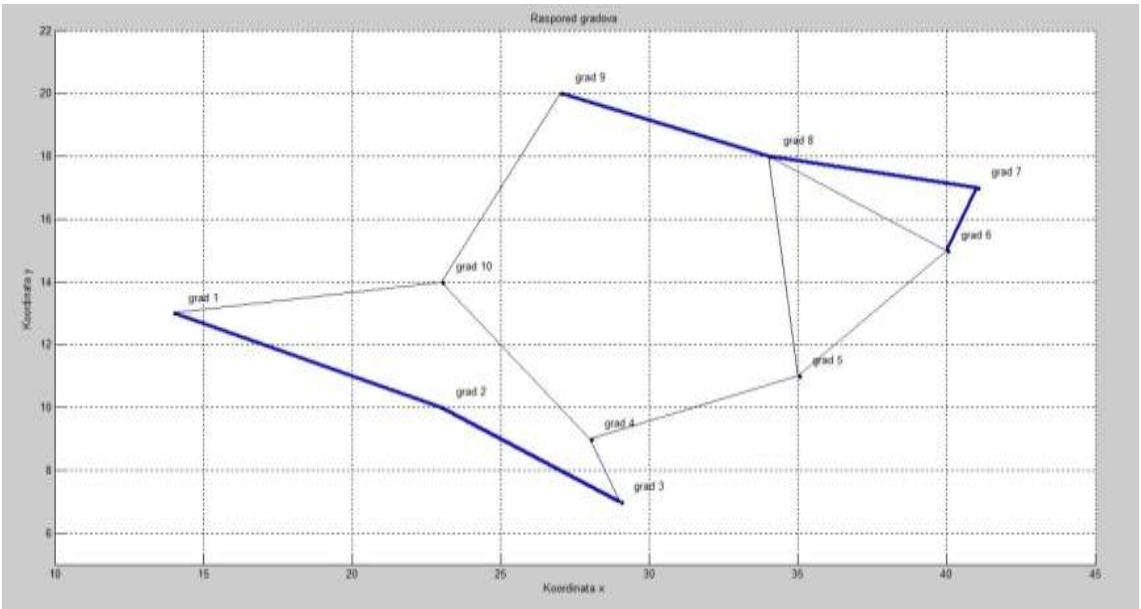
Pre nego što počnem da opisujem postupak rešavanja problema hteo bih napomenuti da sam sve odvojene celine programa smeštao u zasebne funkcije pa ih potom pozivao u glavnom programu. Ovaj pristup sam koristio u izradi svih zadataka kako bi preglednost algoritma bila veća i da bih lakše locirao i otklonio probleme koje sam imao prilikom kucanja koda.

Prvi korak u rešavanju problema bio je odabir mape na kojoj ću definisati gradove, postojanje i kvalitet puteva između njih. Za mapu koju ću koristiti sam izabrao jedan region Srbije na kojem sam definisao 10 čvorova gde će se putevi račvati. Mapu, kao i koordinate gradova sam našao na sajtu www.serbiamap.net. Koordinate gradova sam koristio da bi definisao poziciju gradova kako bi mogao iscrtati gradove i puteve između gradova na početnoj mapi a takođe i da bi iscrtao dobijeno rešenje problema.

Gradove koje sam usvojio kao moguće lokacije kao i putevi između njih su prikazani na sledećoj slici (slika 1).

Slika 1: Čvorovi su označeni crvenim tačkama.

Mapa koju sam iscrtao u programskom paketu MATLAB je prikazana na sledećoj slici (slika 2).



Slika 2: Debljom linijom je označen autoput.

Koordinate gradova, dobijene sa slike 1, su:

```
x=[14 23 29 28 35 40 41 34 27 23]; %koordinata x
y=[13 10  7  9 11 15 17 18 20 14]; %koordinata y
```

Algoritam sam pisao tako da ga je moguće koristiti i za bilo koji drugi skup gradova, pri čemu broj gradova ne mora biti isti. Da bi obezbedio da mi algoritam radi i pri većem (ili manjem) broju gradova uveo sam promenljivu br_grd koja predstavlja broj gradova a definisana je sa `br_grd=length(x)`.

Ako bih imao želju da proširim svoj algoritam tako da računa optimalni put između bilo koja dva grada u Srbiji, na primer, trebao bi jednostavno samo da uvedem nove matrice koordinata i da definišem nove matrice postojanja puta i kvaliteta puta.

Na osnovu početne slike definisao sam matricu postojanja puta (pp) i matricu kvaliteta puta (kp) na osnovu kojih sam iscrtao prikazani grafik na slici 2. Pomenute matrice sam definisao na sledeći način:

```
% Matrica postojanja puteva izmedju gradova - pp
pp=[...
    0 1 0 0 0 0 0 0 0 1;
    1 0 1 0 0 0 0 0 0 0;
    0 1 0 1 0 0 0 0 0 0;
    0 0 1 0 1 0 0 0 0 1;
    0 0 0 1 0 1 0 1 0 0;
    0 0 0 0 1 0 1 1 0 0;
    0 0 0 0 0 1 0 1 0 0;
    0 0 0 0 1 1 1 0 1 0;
    0 0 0 0 0 0 0 1 0 1;
    1 0 0 1 0 0 0 0 1 0];

% Matrica kvaliteta puteva izmedju gradova - kp
kp=[...
    0 2 0 0 0 0 0 0 0 1;
    2 0 2 0 0 0 0 0 0 0;
    0 2 0 1 0 0 0 0 0 0;
    0 0 1 0 1 0 0 0 0 1;
    0 0 0 1 0 1 0 1 0 0;
    0 0 0 0 1 0 2 1 0 0;
    0 0 0 0 0 2 0 2 0 0;
    0 0 0 0 1 1 2 0 2 0;
    0 0 0 0 0 0 0 2 0 2;
    1 0 0 1 0 0 0 0 2 0];
```

Kao što se može videti iz matrice kvaliteta puta, usvojio sam da je autoput dva puta bolji od „običnog“ puta. Naravno, mogao sam uvesti još kategorija kvaliteta puta, ali je slučajnost odlučila da sam na odabranoj slici imao samo dva različita tipa puteva. Lako je uvesti dodatne razlike u kvalitetu puteva i to prostom izmenom vrednosti u matrici kvaliteta puteva.

Rastojanje između gradova sam izračunao kao Euklidsko rastojanje koje je definisano kao rastojanje

$$d = [(x_1 - x_1^{'})^2 + ... + (x_n - x_n^{'})^2]^{1/2}$$

između dva vektora $x = (x_1, x_2, ..., x_n)^T$ i $x^{'} = (x_1^{'}, x_2^{'}, ..., x_n^{'})^T$.

Kod kojim sam obezbedio računanje daljina između gradova je sledeći:

```
% Racunam daljine izmedju svih gradova
d=zeros(br_grd,br_grd);
f=zeros(br_grd,br_grd);
for i=1:br_grd
    for j=1:br_grd
        d(i,j)=sqrt((x(1,i)+x(1,j))^2+(y(1,i)+y(1,j))^2);
        f(i,j)=1/d(i,j); %reciprocna vrednost daljine
    end
end
```

Bitno je primetiti da sam mesto daljina koristio recipročnu vrednost daljina jer sam pretpostavio da je putanja između dva proizvoljna grada bolja ako je put između njih kraći.

4. Izbor populacije

Kao što sam već naveo u prethodnom tekstu, uobičajeno je da populaciju sačinjavaju hromozomi koji su predstavljeni kao nizovi binarnih cifara. Međutim, taj pristup se prilikom rešavanja problema trgovačkog putnika ne može koristiti. Znamo takođe da populacija predstavlja skup mogućih rešenja datog problema.

Dakle, ako bih kodirao populaciju kao nizove binarnih brojeva javili bi se brojni problemi. Na primer, ukrštanjem dva roditelja (koji su predstavljeni nizom binarnih cifara) dobio bi novi niz binarnih cifara koji ne bi nosio nikakvu informaciju. Mutacija nekog potomka bi takođe dovela do gubitka informacija a ne bih dobio novi kvalitet kako je to zamišljeno teorijom genetskih algoritama. Zaključak je da, ako bi hteo binarno da kodiram populaciju, ne bi smeo da koristim operatore ukrštanja i mutacije čime bi bio prinuđen da problem rešavam analitički. Onda se postavlja pitanje šta će mi uopšte genetski algoritam.

Naravno, stvari nisu tako crne kao što sam počeo da pišem. Upravo zbog ovakvih problema i postoje različiti načini kodiranja populacije.

Ja sam u svom algoritmu koristio takozvano ***permutaciono kodiranje***. To jest, svaki hromozom ima onoliko elemenata koliko ima gradova u zadatom skupu a aleoli (elementi hromozoma) pripadaju skupu [1, br_gradova]. Na primer, hromozom [1 4 2 8 5 9 6 3 10 7] se sastoji od deset članova i govori mi kojim su redosledom gradovi posećeni. Krećemo iz prvog grada ka četvrtom pa zatim drugom itd. dok se put ne završi u sedmom gradu.

Ovo je rezon koji bi koristio u klasičnom problemu trgovačkog putnika. Zavisno od broja gradova definisao bi svakog člana populacije kao nasumične permutacije niza decimalnih cifara kojih ima onoliko koliko ima gradova. Vidimo da je kreiranje populacije u ovom slučaju veoma prosto.

U problemu koji sam ja imao da rešim ovakav rezon nisam mogao da koristim.

Pošto sam ja morao da napišem algoritam koji bi mi tražio optimalan put između dva proizvoljna grada pristup koji sam koristio bio je drugačiji. Svaki put kada bi pokrenuo program dobijao sam nova dva nasumična grada. To znači da sam populaciju svaki put morao nanovo da pravim za konkretan slučaj.

Prvi korak mi je bio odabir dva nasumična grada što sam rešio sledećim kodom:

```
% Biram nasumicno pocetni i krajnji grad
poc_grd=fix(1+br_grd*rand(1));
krj_grd=fix(1+br_grd*rand(1));
% Uveravam se da nisam izabrao dva ista grada
while (poc_grd==krj_grd)
    krj_grd=fix(1+br_grd*rand(1));
end
```

Za dva odabrana grada sam napravio populaciju tako da svaki hromozom predstavlja jednu od mogućih putanja između ovih gradova. Logično je da mi je dužina hromozoma kod različitih putanja bila različita ali to nije pravilo nikakav problem.

Kao što sam već napomenuo, različite celine programa sam smeštao u različite funkcije pa sam tako i formiranje populacije realizovao u zasebnoj funkciji koju sam nazvao *formiram_pop*.

Ideja kojoj sam se služio sastojala se u sledećem. Kako sam već odabrao dva nasumična grada tako su mi početni i krajnji grad bili unapred poznati. Dakle kretao sam iz grada *poc_grd*. Za odabir narednog grada bilo je potrebno da obezbedim dva uslova. Prvi je bio da naredni grad koji odaberem nije već posećen a drugi uslov je da do tog grada postoji put. U početku sam dosta „petljao“ oko realizacije ove ideje dok nisam našao prosto i elegantno rešenje (mada sam siguran da postoji i jednostavnije rešenje).

Jednostavno sam napravio pomoćnu matricu koja je nosila informaciju koji su gradovi preostali da budu posećeni. Na taj način sam obezbedio uslov da ne posetim isti grad dva puta. Ovo sam rešio na sledeći način:

```
% Pravim matricu gradova koji preostaju da se posete
pom_lok_1=zeros(1,br_grd); % Pomocna matrica neposecenih lokacija
k_1=0;
for i=1:br_grd
    if (i~=poc_grd)
        k_1=k_1+1;
        pom_lok_1(1,k_1)=i; % Pomocna matrica preostalih gradova
    end
end
```

Sledeći korak je bio da napravim matricu koja sadrži iste podatke kao i prethodna matrica s tom razlikom da nova matrica neće u sebi imati nula.

```
pom_lok_2=zeros(1,k_1); % Matrica lokacija
for i=1:k_1
    pom_lok_2(1,i)=pom_lok_1(1,i); % Matrica preostalih gradova (bez nula)
end
```

U stvari, samo sam promenio dimenzije originalne matrice tako da ona sadrži samo nenulte članove. U izvođenju ove ideje koristio sam pomoćni brojač *k_1* koji mi je rekao koliko ima nenultih članova u originalnoj matrici.

Sledeći korak je bio da napravim matricu preostalih gradova do kojih postoji put.

```
% Pravim matricu neposecenih gradova do kojih postoji put
pom_lok_3=zeros(1,k_1);
k_2=0;
for i=1:k_1
    if (pp(poc_grd,pom_lok_2(1,i))==1)
        k_2=k_2+1;
        pom_lok_3(1,k_2)=pom_lok_2(1,i); % Neposeceni gradovi do kojih
                                         % postoji put
    end
end
```

Posle ovoga je opet usledio isti proces redimenzioniranja matrice *pom_lok_3* tako da dobijem matricu neposećenih gradova do kojih postoji put. Ovo sam sve uradio sa ciljem da si pojednostavim izbor sledećeg grada tako što ću sledeći grad da biram nasumično kao jednu od vrednosti dobijene matrice. Matrica koju sam dobio kao krajnje rešenje je:

```
% Pravim matricu neposecenih gradova do kojih postoji put bez nula
lok=zeros(1,k_2); % Matrica neposecenih gradova do kojih postoji put
for i=1:k_2
    lok(1,i)=pom_lok_3(1,i);
end
```

Ne samo što sada imam matricu neposećenih gradova do kojih postoji put, već znam i koliko tih gradova ima. Izbor sledećeg grada je sada bio prost.

```
% Nasumicno biram jedan od neposecenih gradova do kojih postoji put
sl_grd=lok(1,fix(1+k_2*rand(1))); % Sledeci grad
```

Na samom početku ove funkcije sam naveo da mi početnu lokaciju u hromozomu predstavlja grad koji sam nasumično odabrao kao početni.

```
% Postavljam odabrani grad na prvo mesto u pomocnoj matrici populacije
pom_pop=zeros(1,br_grd); % Pomocna matrica populacije
pom_pop(1,1)=poc_grd;
```

Nakon izvršenog izbora sledećeg grada ažurirao sam pomoćnu matricu populacije sa sledećim kodom.

```
% Usnimavam u informaciju hromozoma ovaj sledeci grad
pom_pop(1,br+1)=sl_grd;
```

Što se tiče idejnih rešenja koja sam koristio u ovoj funkciji preostalo je da objasnim samo još jedan korak. Gore navedeni proces sam ponavljao dok god algoritam ne dođe u situaciju da je sledeći grad u stvari poslednji grad koji sam nasumično odabrao. Ovo sam rešio *while* petljom. Bitno je napomenuti da sam pre ove petlje ispisao gore navedeni algoritam za dobijanje matrice neposećenih gradova do kojih postoji put.

```
% Ponavljam postupak dok god ne dodjem do krajnjeg grada
br=1; % Pomocni brojac
while (poc_grd~=krj_grd)
    % Nasumicno biram jedan od neposecenih gradova do kojih postoji put
    sl_grd=lok(1,fix(1+k_2*rand(1))); % Sledeci grad
    % Usnimavam u informaciju hromozoma ovaj sledeci grad
    pom_pop(1,br+1)=sl_grd;
    % Pocetni grad mi je sada sledeci grad
    poc_grd=sl_grd;

    % Opet pravim matricu neposecenih gradova do kojih postoji put

    % Ako vise nema neposecenih gradova do koji postoji put setujem
    % preostale clanove na 11 i izlazim iz petlje
    if (k_2==0)
        for i=1:br_grd
            if (pom_pop(1,i)==0)
                pom_pop(1,i)=11;
            end
        end
        break
    end
    % Inkrementiram brojac
    br=br+1;
end
```

Kao što se može primetiti, dodao sam još jednu liniju koda koja kaže da mi je sada početni grad u stvari sledeći grad koji sam odabrao. Nakon ovih, početnih, linija u *while* petlji sledi isti proces kao što sam već naveo. Opet pravim matricu neposećenih gradova do kojih postoji put.

Na kraju *while* petlje sam dodao još jedan uslov da ako ne postoji više gradova koje bi mogao odabrati izlazim iz petlje. Ovaj uslov sam dodao jer je moguće da algoritam nasumično odabere grad iz koga ne postoji put do sledećeg grada koji nije posećen iako nije još došao do ciljnog grada. Da bi znao da je došlo do ovog "nedozvoljenog" stanja setovao sam preostale članove hromozoma na 11 kako bi mogao da, pri proračunu fitnesa ovog hromozoma, umanjim fitnes do te mere da ovaj hromozom nema šanse da bude izabran za sledeću generaciju.

5. Proračun fitnes funkcije

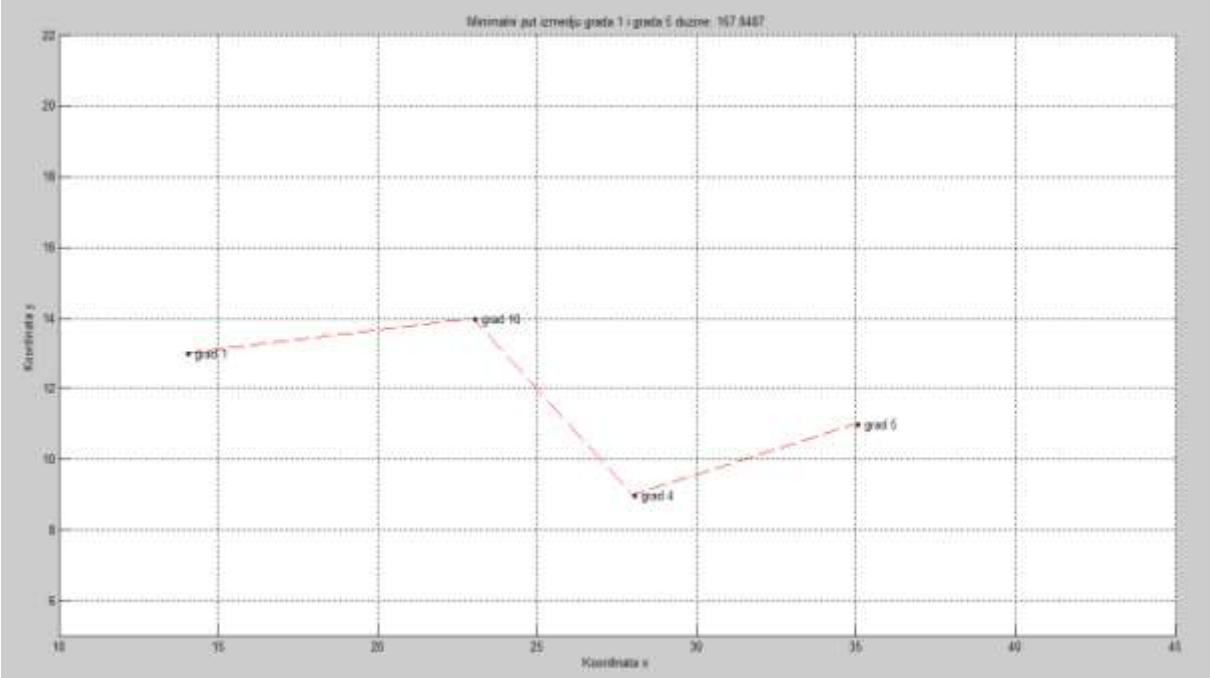
Fitnes je dodeljen svakoj individui u populaciji, gde veliki brojevi označavaju dobar status. Fitnes funkcija može biti bilo koja nelinearna, ne-diferencijabilna, nekontinualna, pozitivna funkcija jer algoritam zahteva samo da fitnes bude dodeljen svakom nizu (hromozomu).

Kod računanja fitnes funkcije nisam imao problema jer je rezon koji sam koristio prost. Kompletan kod za proračun fitnesa sam smestio u zasebnu funkciju koju sam nazvao *fitnes_novo*. Izračunao sam sume dužine i kvaliteta pređenog puta za svaku putanju kao i još jednu dodatnu sumu postojanja puta. Ove tri sume sam iskoristio za formiranje fitnes funkcije koju sam odabrao na sledeći način:

```
% Dobijeni fitnes je jednak
fitnes=zeros(br_grd*br_grd,1);
for i=1:br_grd*br_grd
    fitnes(i,1)=1000*suma_d(i,1)+1*suma_kp(i,1)*suma_pp(i,1);
end
```

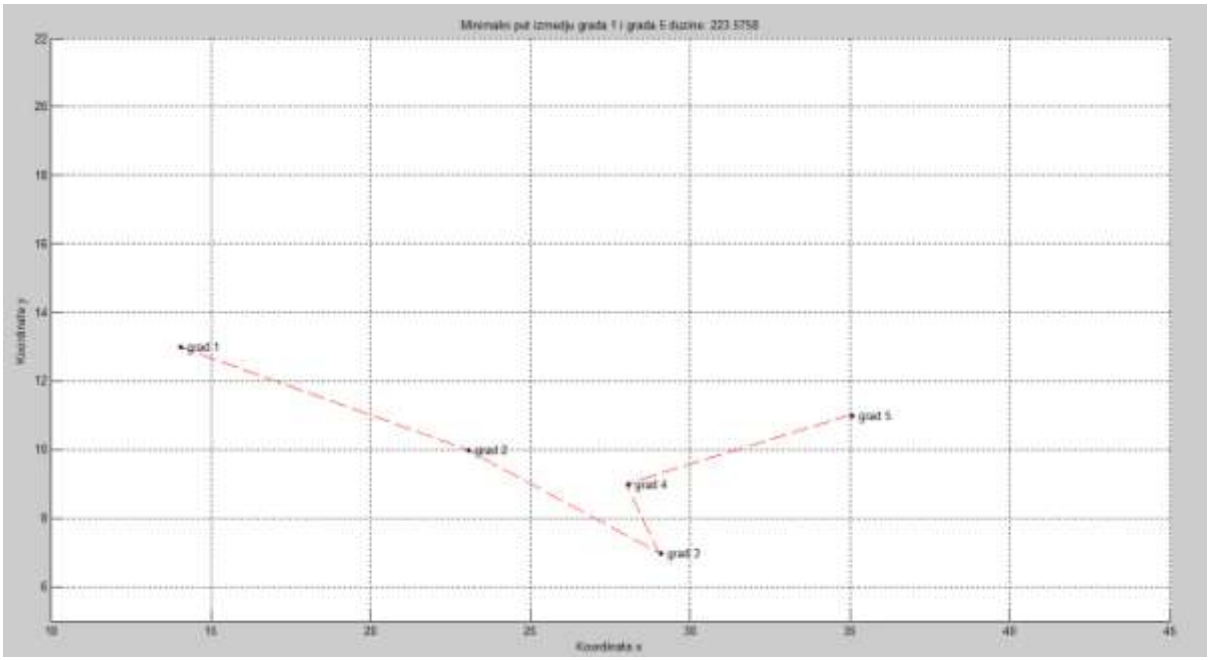
Može se videti da mi pri proračunu fitnes funkcije najviše uticaja ima dužina pređenog puta što sam usvojio empirijski. Uvodeći analogiju sa modernim GPS (Global Positioning System) uređajima gde se kao upit često postavljaju uslovi da se odabere ili optimalan ili najkraći put tako se i u mom slučaju promenom množilaca suma mogu postići različiti rezultati.

U slučaju traženja optimalnog puta između grada 1 i grada 5 dobijam putanju prikazanu na sledećoj slici (slika 3).



Slika 3: Optimalni put između gradova 1 i 5.

Posmatrajući sliku 2 može nam se učiniti da postoji bolji put za izabrane gradove, i to onaj put koji koristi autoput. Ovaj put bi krenuo iz čvora 1 preko čvorova 2,3 i 4 da bi naposletku stigao do čvora 5. Ta putanja je prikazana na slici 4.



Slika 4: Naizgled optimalan put između gradova 1 i 5.

Međutim, razlika između ove dve putanje je očigledna. U prvom slučaju je dužina pređenog puta približno 168km, dok je u drugom slučaju dužina pređenog puta približno 224km. Dakle druga putanja je duža za 56km, tj. duža je za 33% od prve putanje, tako da zaključujem da algoritam radi pravilno.

Dužinu pređenog puta sam izračunao koristeći prethodno izračunata (Euklidska) rastojanja između gradova.

```
suma_d=zeros(br_grd*br_grd,1); % Pomocna suma 1
% Racunam predjeni put svakog clana populacije
for i=1:br_grd*br_grd
    suma=0; % Pomocna suma
    for j=1:br_grd-1
        tr_grd=pop(i,j); % Trenutni grad
        nr_grd=pop(i,j+1); % Naredni grad
        if (nr_grd~=0 && nr_grd~=11)
            suma=suma+d(tr_grd,nr_grd);
        elseif (nr_grd~=0 && nr_grd==11)
            suma=suma+1000;
        end
    end
    suma_d(i,1)=1/suma; % Prva potrebna suma za fitnes (reciprocna duzina puta)
end
```

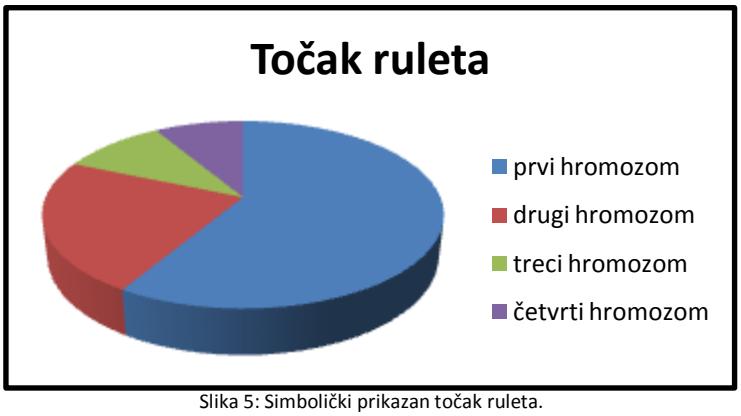
Matrice kvaliteta pređenog puta i postojanja puta sam proračunao na sličan način, s tom razlikom što mesto Euklidske daljine (d) u algoritmu figurišu matrice kvaliteta puta (kp) i matrica postojanja puta (pp). Mesto na kome menjam pomenute matrice je označeno crvenom bojom u prikazanom kodu.

U ove petlje sam uveo dva dodatna uslova kako bi fitnes bio pravilno formiran.

Prvi uslov se tiče problem koji nastaje zbog toga što su hromozomi različitih dužina pa sam naveo da ako je sledeći član hromozoma jednak nuli tada sumu ne računam dalje. Drugi uslov je sličan, s tim što sam kod njega naveo da se suma ne računa na "klasičan" način ako je sledeći član jednak 11 (stanje greške), već dodajem veliku vrednost sumi. Na taj način obezbeđujem uslov da je fitnes hromozoma koji ima nedozvoljeno stanje daleko manji od ostalih hromozoma populacije pa skoro sigurno neće biti izabran za sledeću generaciju.

6. Reprodukcija

Reprodukcija je proces u kome se individualni nizovi kopiraju u skladu sa njihovom vrednošću fitnesa. Ovaj operator je veštačka verzija prirodne selekcije. Proces reprodukcije (biranje roditelja) se može izvesti pomoću više različitih metoda. Ja sam odabrao metod ***točka ruleta*** (Roulette Wheel Selection) gde se proces reprodukcije vrši okretanjem simuliranog točka ruleta čija polja imaju različite veličine proporcionalne vrednosti fitnesa individua. Svaki put kada nam treba potomak dovoljno je okrenuti točak ruleta koji nam daje kandidata za reprodukciju. Simbolični točak ruleta je prikazan na slici 5.



Slika 5: Simbolički prikazan točak ruleta.

Ova tehnika se algoritamski uvodi na sledeći način:

1. Sumiramo fitnes svih članova populacije i nazovemo taj fitnes "total fitnes";
2. Generišemo n, nasumični broj između nule i totalnog fitnesa;
3. Vraćamo prvog člana populacije čiji je fitnes, sabran sa fitnesom svih prethodnih članova, veći ili jednak generisanom broju n.

Funkciju u kojoj sam rešio problem reprodukcije i u kojoj sam uvrstio dobijene potomke u populaciju sam nazvao *dominacija*. Prvi korak sam izveo na sledeći način:

```
% Trazim zbir svih fitnesa (total fitnes)
tot_fitnes=0; % Inicijalizujem total fitnes
for i=1:br_grd*br_grd
    tot_fitnes=tot_fitnes+fitnes(i,1);
end
```

Zatim sam opisane korake dva i tri radio zasebno za oba roditelja. Pošto su koraci identični, i kod je identičan za oba roditelja sa jedinom razlikom u indeksima. Indekse koje sam menjao sam u sledećem kodu označio crvenom bojom.

```
% Trazim jedan nasumican broj od 0 do total fitnesa
n=tot_fitnes*rand(1);
suma=0; % Pomocna suma
br=0; % Pomocni brojac
while (suma<n)
    br=br+1;
    suma=suma+fitnes(br,1);
end
roditelj_1=pop(br,:); % Prvi roditelj
fit_roditelja_1=fitnes(br,1); % Fitnes prvog roditelja
```

Kao što možemo primetiti, reprodukcija je u mom slučaju, takođe, izvedena prosto.

Već sam napomenuo da bi ukrštanje i mutacija u mom slučaju doveli do gubitka informacija, odnosno algoritam ne bi pronašao korektno rešenje. Iz tog razloga sam usvojio da su potomci direktne kopije roditelja što je za rezultat imalo dalje uproštavanje koda. Izabrane roditelje sam odmah uvrštavao u populaciju zajedno sa pripadajućim fitnesima na mestu hromozoma koji imaju minimalni fitnes.

Kao i u prethodnom slučaju, algoritam je isti za oba roditelja sa jedinom razlikom u indeksima matrica.

```
% Trazim clana populacije sa minimalnim fitnesom (ubacujem prvog roditelja)
min_fitnes=min(fitnes);
for i=1:br_grd*br_grd
    if (min_fitnes==fitnes(i,1))
        pop(i,:)=roditelj_1; % Ubacujem prvog roditelja na njegovo mesto
        fitnes(i,1)=fit_roditelja_1; % Ubacujem i fitnes roditelja 1
    end
end
```

Zaključno sa funkcijom *dominacija* sam završio projektovanje genetskog algoritma za rešavanje zadatog problema. Sve funkcije koje sam koristio kao i glavni program sam napisao u dodatku ovog dokumenta. Ostalo je da uvrstim ove funkcije u glavni program i iscrtam dobijena rešenja.

7. Glavni program

Program sam počeo inicijalizacijom početnih podataka, odnosno pozivanjem funkcije *pocetne_info*.

```
% Inicijalizujem pocetne podatke
[x,y,br_grd,pp,kp,d,f,poc_grd,krj_grd]=pocetne_info;
% Formiram populaciju za date gradove
```

Zatim sam izvršavanjem funkcije formiranja populacije sto puta (br_grd*br_grd puta) dobio sto hromozoma koji u sebi nose različite putanje između dva odabrana grada.

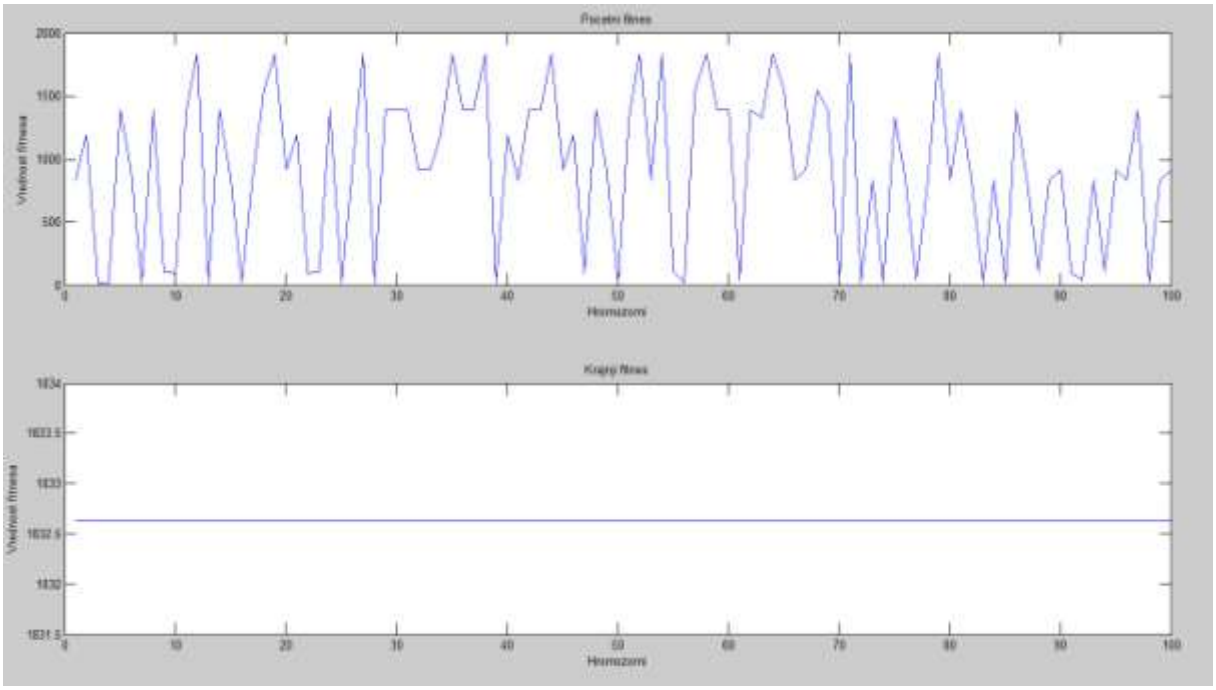
```
% Formiram populaciju za date gradove
pop=zeros(br_grd*br_grd,br_grd);
for i=1:br_grd*br_grd
    [privr_pop]=formiram_pop(poc_grd,krj_grd,br_grd,pp);
```

```
    pop(i,:)=privr_pop(1,:);
end
```

Pozivanjem funkcije za proračunavanje fitnesa sam izračunao fitnes svakog hromozoma u populaciji.

```
% Racunam fitnes za sve clanove date populacije
[fitnes]=fitnes_novo(br_grd,pop,f,pp,kp);
```

Zatim sam uveo operator reprodukcije i uvrštavanje dobijenih potomaka u postojeću populaciju. Nakon reprodukcije se prosečan fitnes promenio u toj meri da sam od skupa mnogo različitih vrednosti fitnesa dobio jedinstven fitnes. Ovo je prikazano na sledećoj slici (slika 6).



Slika 6: Početni i krajnji fitnes.

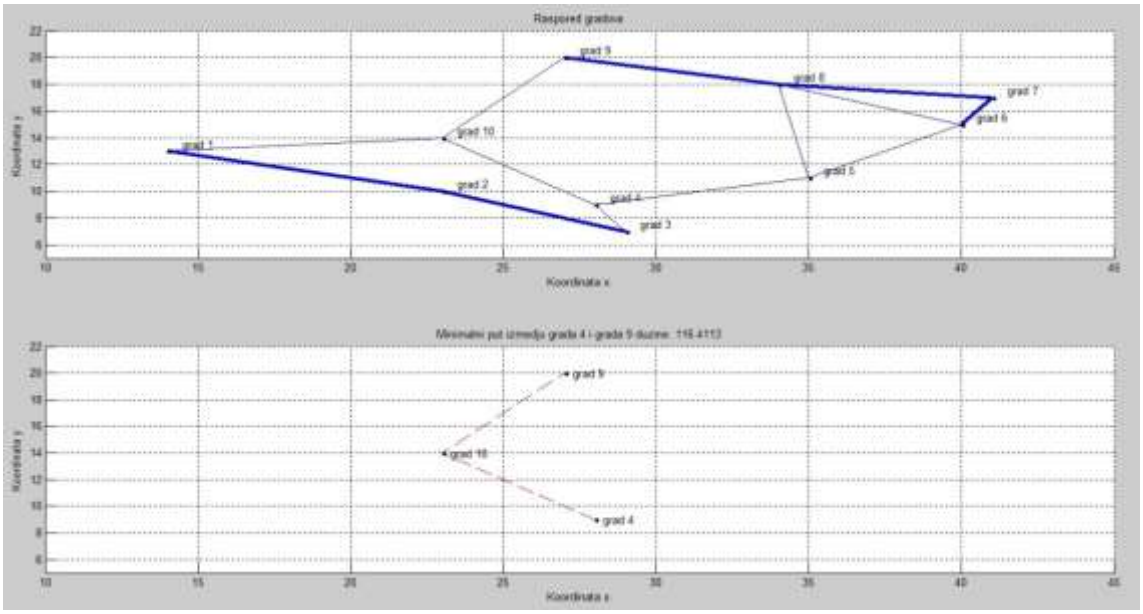
Algoritam kojim sam uveo reprodukciju i uvrštavanje potomaka i pripadajućeg fitnesa pozivanjem funkcije *dominacija* je sledeći:

```
% Uvodim reprodukciju (ponavljam postupak)
for i=1:3000
    [pop,fitnes]=dominacija(br_grd,fitnes,pop);
end
```

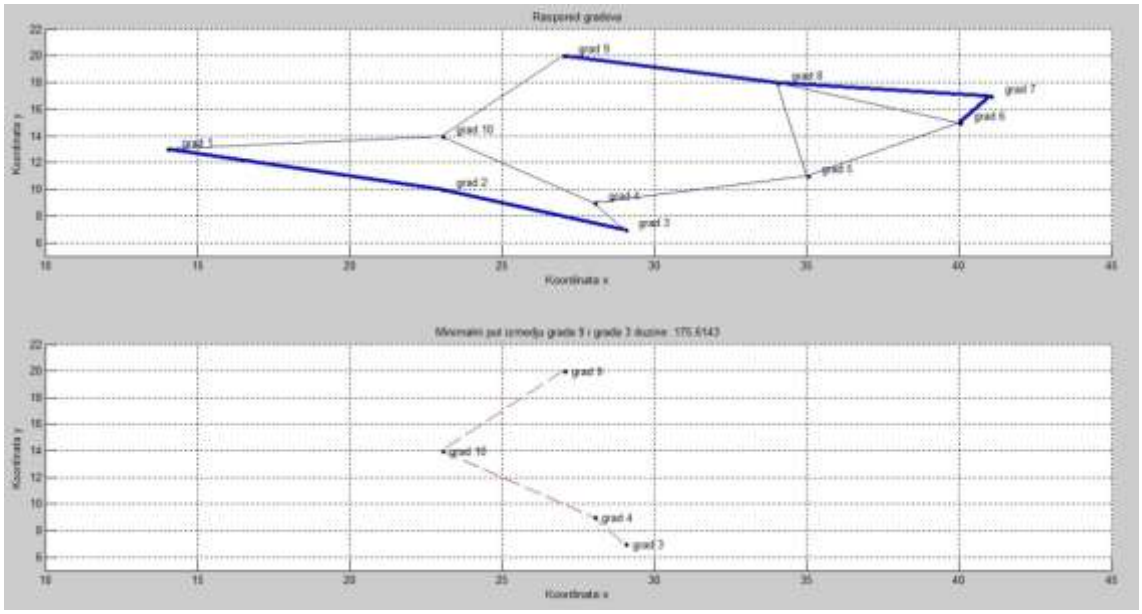
8. Upotrebna vrednost rešenja problema

Kao i rešavanje problema trgovačkog putnika i rešavanje ovog problema nije ostalo na striktno teorijskom nivou. Očigledan primer upotrebe ovakvog algoritma je u GPS uređajima, međutim upotreba se može proširiti i na drugi, veći skup problema vezanih za logistiku i optimizaciju.

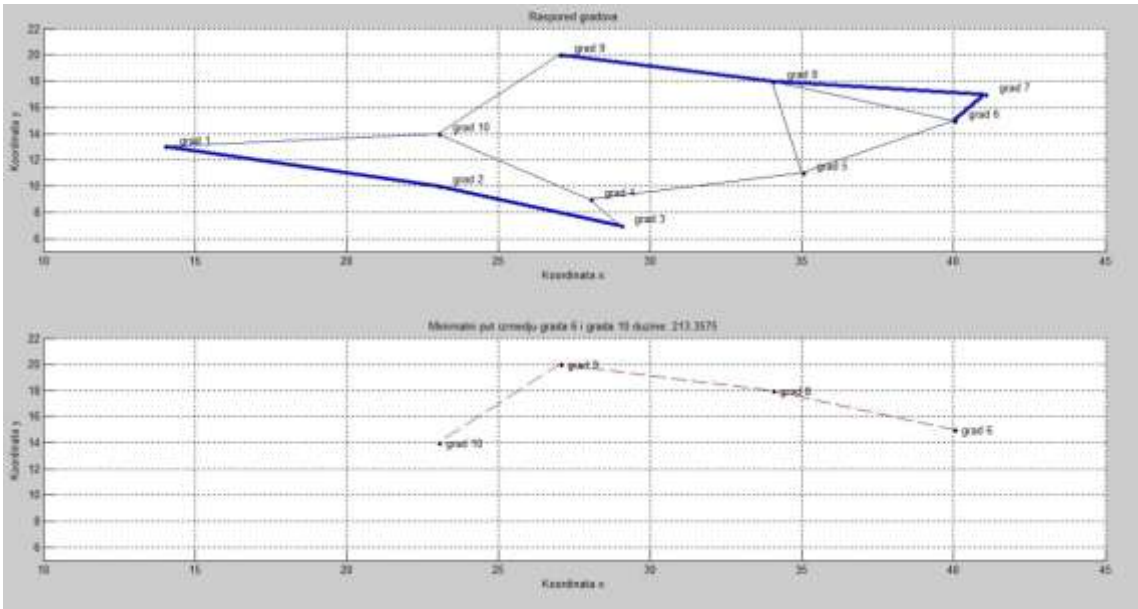
Čest je slučaj da je potrebno optimizovati neku proizvodnju ili, na primer, transtport dobara u čemu bi ovaj algoritam mogao naći svoju upotrebu. U svakom slučaju, problem je rešen a neka od varijanti rešenja posle nasumičnog generisanja početnog i krajnjeg grada su date na sledećim slikama.



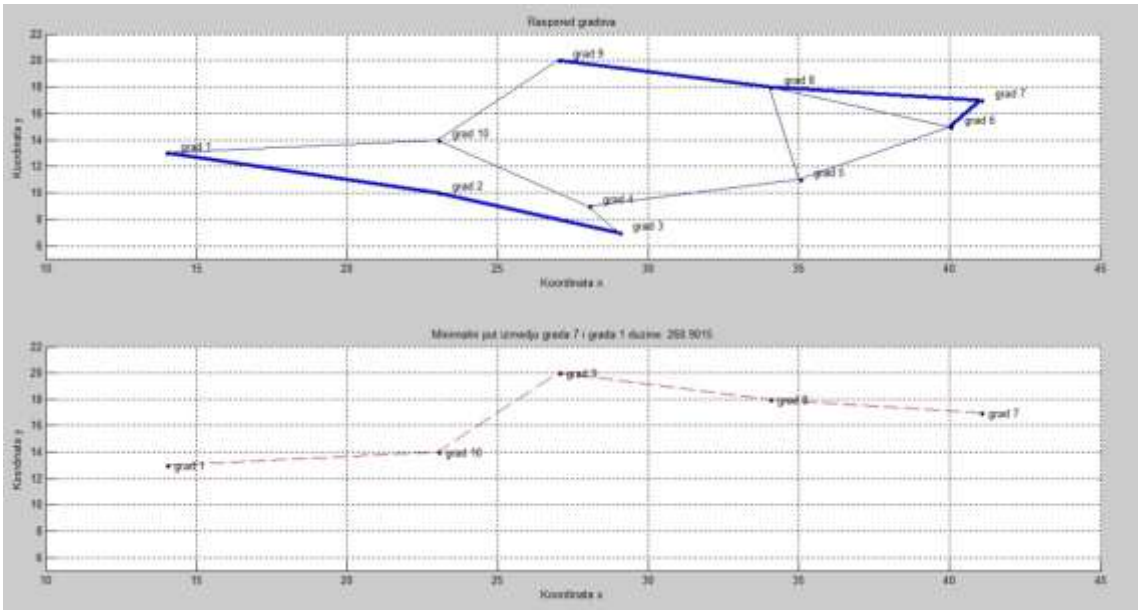
Slika 7: Primer 1.



Slika 8: Primer 2.



Slika 9: Primer 3.



Slika 10: Primer 4.

Dodatak A

Glavni program.

```
%% Domaci zadatak 1 - 08.09.2010. Autor: Ognjen Zelenbabic
% Problem: Trazenje optimalne putanje izmedju dva grada pomocu genetskog algoritma
clc
clear all
close all

% Inicijalizujem pocetne podatke
[x,y,br_grd,pp,kp,d,f,poc_grd,krj_grd]=pocetne_info;
% Formiram populaciju za date gradove
pop=zeros(br_grd*br_grd,br_grd);
for i=1:br_grd*br_grd
    [privr_pop]=formiram_pop(poc_grd,krj_grd,br_grd,pp);
    pop(i,:)=privr_pop(1,:);
end
% Racunam fitnes za sve clanove date populacije
[fitnes]=fitnes_novo(br_grd,pop,f,pp,kp);
% Iscrtavam pocetni fitnes
t=1:1:br_grd*br_grd;
figure(1)
subplot(211)
plot(t,fitnes);
title('Pocetni fitnes');
xlabel('Hromozomi');
ylabel('Vrednost fitnesa');
% Uvodim reprodukciju (ponavljam postupak)
for i=1:3000
    [pop,fitnes]=dominacija(br_grd,fitnes,pop);
end
subplot(212)
plot(t,fitnes);
title('Krajnji fitnes');
xlabel('Hromozomi');
ylabel('Vrednost fitnesa');

%% Iscrtavam resenje
figure(2)
subplot(211)
hold on
grid on
% Menjam dimenzije matrice resenja tako da nema nula
k=0;
pom_pop_0=zeros(1,br_grd); % Pomocna matrica
for i=1:br_grd
    if (pop(1,i)~=0)
        k=k+1;
        pom_pop_0(1,k)=pop(1,i);
    end
end
pom_pop=zeros(1,k);
for i=1:k
    pom_pop(1,i)=pom_pop_0(1,i);
end
% Racunam duzinu predjenog puta
suma=0;
for i=1:k-1
    tr_grd=pop(1,i);
    nr_grd=pop(1,i+1);
    suma=suma+d(tr_grd,nr_grd);
end
duzina_puta=suma;

for i=1:br_grd
```

```
    for j=1:br_grd
        if (pp(i,j)==1 && i==1 && j==2)
            X=[x(1,i) x(1,j)];
            Y=[y(1,i) y(1,j)];
            line(X,Y,'Color','b','LineWidth',3);
        elseif (pp(i,j)==1 && i==2 && j==3)
            X=[x(1,i) x(1,j)];
            Y=[y(1,i) y(1,j)];
            line(X,Y,'Color','b','LineWidth',3);
        elseif (pp(i,j)==1 && i==2 && j==3)
            X=[x(1,i) x(1,j)];
            Y=[y(1,i) y(1,j)];
            line(X,Y,'Color','b','LineWidth',3);
        elseif (pp(i,j)==1 && i==6 && j==7)
            X=[x(1,i) x(1,j)];
            Y=[y(1,i) y(1,j)];
            line(X,Y,'Color','b','LineWidth',3);
        elseif (pp(i,j)==1 && i==7 && j==8)
            X=[x(1,i) x(1,j)];
            Y=[y(1,i) y(1,j)];
            line(X,Y,'Color','b','LineWidth',3);
        elseif (pp(i,j)==1 && i==8 && j==9)
            X=[x(1,i) x(1,j)];
            Y=[y(1,i) y(1,j)];
            line(X,Y,'Color','b','LineWidth',3);
        elseif (pp(i,j)==1)
            X=[x(1,i) x(1,j)];
            Y=[y(1,i) y(1,j)];
            line(X,Y,'Color','b','LineWidth',1);
        end
    end
end

for i=1:br_grd
    text(x(1,i),y(1,i),['\bullet']);
    text(x(1,i)+0.5,y(1,i)+0.5,['grad ',num2str(i)]);
end
axis([10 45 5 22]);

title('Raspored gradova');
xlabel('Koordinata x');
ylabel('Koordinata y');

subplot(212)
hold on
grid on
for i=1:k
    xd(1,i)=x(1,pom_pop(1,i));
    yd(1,i)=y(1,pom_pop(1,i));
end
plot(xd,yd,'--r');
for i=1:k
    n=pom_pop(1,i);
    text(xd(i),yd(i),['\bullet grad ',num2str(n)]);
end
axis([10 45 5 22]);
title(['Minimalni put izmedju grada ',num2str(poc_grd),' i grada
',num2str(krj_grd),' duzine: ',num2str(duzina_puta)]);
xlabel('Koordinata x');
ylabel('Koordinata y');
```

Funkcija *pocetne_info*.

```
function [x,y,br_grd,pp,kp,d,f,poc_grd,krj_grd]=pocetne_info
%% Inicijalizujem koordinate gradova i pocetne podatke

x=[14 23 29 28 35 40 41 34 27 23]; %koordinata x
y=[13 10  7  9 11 15 17 18 20 14]; %koordinata y
br_grd=length(x); %daje mi broj gradova

% Matrica postojanja puteva izmedju gradova - pp
pp=[...
    0 1 0 0 0 0 0 0 0 1;
    1 0 1 0 0 0 0 0 0 0;
    0 1 0 1 0 0 0 0 0 0;
    0 0 1 0 1 0 0 0 0 1;
    0 0 0 1 0 1 0 1 0 0;
    0 0 0 0 1 0 1 1 0 0;
    0 0 0 0 0 1 0 1 0 0;
    0 0 0 0 1 1 1 0 1 0;
    0 0 0 0 0 0 0 1 0 1;
    1 0 0 1 0 0 0 0 1 0];

% Matrica kvaliteta puteva izmedju gradova - kp
kp=[...
    0 2 0 0 0 0 0 0 0 1;
    2 0 2 0 0 0 0 0 0 0;
    0 2 0 1 0 0 0 0 0 0;
    0 0 1 0 1 0 0 0 0 1;
    0 0 0 1 0 1 0 1 0 0;
    0 0 0 0 1 0 2 1 0 0;
    0 0 0 0 0 2 0 2 0 0;
    0 0 0 0 1 1 2 0 2 0;
    0 0 0 0 0 0 0 2 0 2;
    1 0 0 1 0 0 0 0 2 0];

% Racunam daljine izmedju svih gradova
d=zeros(br_grd,br_grd);
f=zeros(br_grd,br_grd);
for i=1:br_grd
    for j=1:br_grd
        d(i,j)=sqrt((x(1,i)+x(1,j))^2+(y(1,i)+y(1,j))^2);
        f(i,j)=1/d(i,j); %reciprocna vrednost daljine
    end
end

%% Biram nasumicno pocetni i krajnji grad
poc_grd=fix(1+br_grd*rand(1));
krj_grd=fix(1+br_grd*rand(1));
% Uveravam se da nisam izabrao dva ista grada
while (poc_grd==krj_grd)
    krj_grd=fix(1+br_grd*rand(1));
end
end
```

Funkcija *formiram_pop*.

```
function [privr_pop]=formiram_pop(poc_grd,krj_grd,br_grd,pp)
%% Formiram populaciju na osnovu odabranog pocetnog i krajnjeg grada
pom_pop=zeros(1,br_grd); % Pomocna matrica populacije

% Postavljam odabrani grad na prvo mesto u pomocnoj matrici populacije
pom_pop(1,1)=poc_grd;

% Pravim matricu gradova koji preostaju da se posete
pom_lok_1=zeros(1,br_grd); % Pomocna matrica neposecenih lokacija
k_1=0;
for i=1:br_grd
    if (i~=poc_grd)
        k_1=k_1+1;
        pom_lok_1(1,k_1)=i; % Pomocna matrica preostalih gradova
    end
end
pom_lok_2=zeros(1,k_1); % Matrica lokacija
for i=1:k_1
    pom_lok_2(1,i)=pom_lok_1(1,i); % Matrica preostalih gradova (bez nula)
end

% Pravim matricu neposecenih gradova do kojih postoji put
pom_lok_3=zeros(1,k_1);
k_2=0;
for i=1:k_1
    if (pp(poc_grd,pom_lok_2(1,i))==1)
        k_2=k_2+1;
        pom_lok_3(1,k_2)=pom_lok_2(1,i); % Neposeceni gradovi do kojih postoji put
    end
end

% Pravim matricu neposecenih gradova do kojih postoji put bez nula
lok=zeros(1,k_2); % Matrica neposecenih gradova do kojih postoji put
for i=1:k_2
    lok(1,i)=pom_lok_3(1,i);
end

% Ponavljam postupak dok god ne dodjem do krajnjeg grada
br=1; % Pomocni brojac
while (poc_grd~=krj_grd)
    % Nasumicno biram jedan od neposecenih gradova do kojih postoji put
    sl_grd=lok(1,fix(1+k_2*rand(1))); % Sledeci grad
    % Usnimavam u informaciju hromozoma ovaj sledeci grad
    pom_pop(1,br+1)=sl_grd;
    % Pocetni grad mi je sada sledeci grad
    poc_grd=sl_grd;
    % Sada trazim preostale neposecene gradove za ovaj grad
    pom_lok_1=zeros(1,br_grd); % Pomocna matrica preostalih gradova
    k_1=0;
    for i=1:br_grd
        k=0;
        for j=1:br_grd
            if (pom_pop(1,j)~=i)
                k=k+1; % Ako broj nije jednak inkrementiram k
            end
        end
        if (k==br_grd) % Ako je k=10 znaci da taj grad nije posecen
            k_1=k_1+1;
            pom_lok_1(1,k_1)=i;
        end
    end

    % Formiram matricu neposecenih gradova bez nula
    pom_lok_2=zeros(1,k_1); % Pomocna matrica
```

```
    for i=1:k_1
        pom_lok_2(1,i)=pom_lok_1(1,i);
    end

    % Formiram matricu neposecenih gradova do kojih postoji put
    pom_lok_3=zeros(1,k_1); % Pomocna matrica
    k_2=0;
    for i=1:k_1
        if (pp(poc_grd,pom_lok_2(1,i))==1)
            k_2=k_2+1;
            pom_lok_3(1,k_2)=pom_lok_2(1,i); % Neposeceni gradovi do kojih postoji
put
        end
    end

    % Pravim matricu neposecenih gradova do kojih postoji put bez nula
    lok=zeros(1,k_2); % Matrica neposecenih gradova do kojih postoji put
    for i=1:k_2
        lok(1,i)=pom_lok_3(1,i);
    end

    % Ako vise nema neposecenih gradova do koji postoji put setujem
    % preostale clanove na 11 i izlazim iz petlje
    if (k_2==0)
        for i=1:br_grd
            if (pom_pop(1,i)==0)
                pom_pop(1,i)=11;
            end
        end
        break
    end
    % Inkrementiram brojac
    br=br+1;
end

% Kada sam zavrsio odabir gradova upisujem dobijeno resenje
privr_pop=pom_pop;
end
```

Funkcija *fitnes_novo*.

```
function [fitnes]=fitnes_novo(br_grd,pop,d,pp,kp)
%% Racunam fitnes za svakog clana populacije

suma_d=zeros(br_grd*br_grd,1); % Pomocna suma 1
suma_kp=zeros(br_grd*br_grd,1); % Pomocna suma 2
suma_pp=zeros(br_grd*br_grd,1); % Pomocna suma 3

% Racunam predjeni put svakog clana populacije
for i=1:br_grd*br_grd
    suma=0; % Pomocna suma
    for j=1:br_grd-1
        tr_grd=pop(i,j); % Trenutni grad
        nr_grd=pop(i,j+1); % Naredni grad
        if (nr_grd~=0 && nr_grd~=11)
            suma=suma+d(tr_grd,nr_grd);
        elseif (nr_grd~=0 && nr_grd==11)
            suma=suma+10000;
        end
    end
    suma_d(i,1)=1/suma; % Prva potrebna suma za fitnes (reciprocna duzina puta)
end

% Racunam kvalitet predjenog puta svakog clana populacije
for i=1:br_grd*br_grd
    suma=0;
    for j=1:br_grd-1
        tr_grd=pop(i,j); % Trenutni grad
        nr_grd=pop(i,j+1); % Naredni grad
        if (nr_grd~=0 && nr_grd~=11)
            suma=suma+kp(tr_grd,nr_grd);
        end
    end
    suma_kp(i,1)=suma; % Druga potrebna suma za fitnes (kvalitet puta)
end

% Racunam postojanje puta za odabranu putanju
for i=1:br_grd*br_grd
    suma=0;
    for j=1:br_grd-1
        tr_grd=pop(i,j); % Trenutni grad
        nr_grd=pop(i,j+1); % Naredni grad
        if (nr_grd~=0 && nr_grd~=11)
            suma=suma+pp(tr_grd,nr_grd);
        end
    end
    suma_pp(i,1)=suma; % Treca potrebna suma za fitnes (postojanje puta)
end

% Dobijeni fitnes je jednak
fitnes=zeros(br_grd*br_grd,1);
for i=1:br_grd*br_grd
    fitnes(i,1)=100*suma_d(i,1)+suma_kp(i,1)*suma_pp(i,1);
end
end
```

Funkcija *dominacija*.

```
function [pop,fitnes]=dominacija(br_grd,fitnes,pop)
%% Reprodukcija - izvedena pomocu "Tocka ruleta"

% Trazim zbir svih fitnesa (total fitnes)
tot_fitnes=0; % Inicijalizujem total fitnes
for i=1:br_grd*br_grd
    tot_fitnes=tot_fitnes+fitnes(i,1);
end

% Trazim prvog clana populacije ciji je fitnes sabran sa svim prethodnim
% fitnesima veci ili jednak od nasumicnog broja n za oba roditelja

% Prvi roditelj
% Trazim jedan nasumican broj od 0 do total fitnesa
n=tot_fitnes*rand(1);
suma=0; % Pomocna suma
br=0; % Pomocni brojac
while (suma<n)
    br=br+1;
    suma=suma+fitnes(br,1);
end
roditelj_1=pop(br,:); % Prvi roditelj
fit_roditelja_1=fitnes(br,1); % Fitnes prvog roditelja

% Drugi roditelj
% Trazim jedan nasumican broj od 0 do total fitnesa
n=tot_fitnes*rand(1);
suma=0; % Pomocna suma
br=0; % Pomocni brojac
while (suma<n)
    br=br+1;
    suma=suma+fitnes(br,1);
end
roditelj_2=pop(br,:); % Drugi roditelj
fit_roditelja_2=fitnes(br,1); % Fitnes drugog roditelja

%% Ubacujem izabrane roditelje u populaciju mesto inferiornih jedinki

% Trazim clana populacije sa minimalnim fitnesom (ubacujem prvog roditelja)
min_fitnes=min(fitnes);
for i=1:br_grd*br_grd
    if (min_fitnes==fitnes(i,1))
        pop(i,:)=roditelj_1; % Ubacujem prvog roditelja na njegovo mesto
        fitnes(i,1)=fit_roditelja_1; % Ubacujem i fitnes roditelja 1
    end
end

% Trazim clana populacije sa minimalnim fitnesom (ubacujem drugog roditelja)
min_fitnes=min(fitnes);
for i=1:br_grd*br_grd
    if (min_fitnes==fitnes(i,1))
        pop(i,:)=roditelj_2; % Ubacujem drugog roditelja na njegovo mesto
        fitnes(i,1)=fit_roditelja_2; % Ubacujem i fitnes roditelja 2
    end
end
end
```