**Univerzitet u Banjoj Luci**

**Elektrotehnički fakultet**

**Katedra za automatiku**

**Metodi vještačke inteligencije**

Izvještaj o urađenom projektnom zadatku

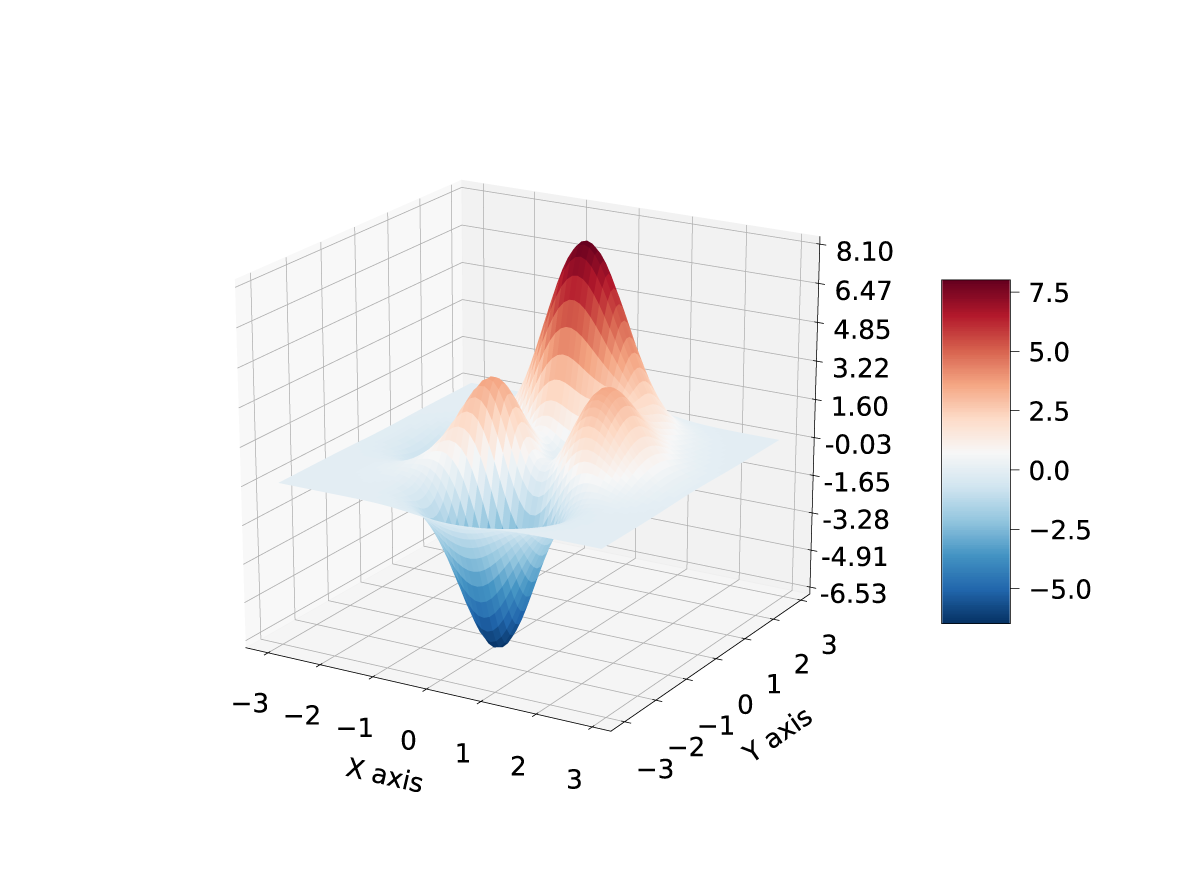
**Genetički algoritmi**

Student:

Nikola Blagojević, 11136-15

# Opis problema

Na slici je prikazan grafik funkcije dvije promjenljive.



Slika 1.1 - Grafik funkcije dvije promjenljive

Analitički izraz funkcije čiji je grafik prikazan na Slici1.1 dat je sa:

Korišćenjem genetičkog algoritma potrebno je odrediti globalni minimum i globalni maksimum posmatrane funkcije na intervalu na kome je funkcija grafički prikazana na Slici1.1.

# Rješenje

Da bi se odredilo rješenje ovog problema potrebno je odrediti opseg vrijednosti promjenljivih *x* i *y* u kojem data funkcija *z* ima minimum i maksimum. Ukoliko funkcija ima minimum (maksimum), to znači da postoji neki interval *I* = (Gd,Gg) takav da postoji i za koje važi:

Sa slike 1.1 zaključujemo da je taj interval isti za vrijednosti *x* i *y* i iznosi *I* = [-3,3].

Nakon utvrđivanja intervala za vrijednosti *x* i *y*, potrebno je izabrati početnu populaciju.

Početna populacija je birana pseudoslučajno. Za potrebe genetičkog algoritma moguća rješenja su kodovana binarno. Kodovanje je izvršeno tako da sve dekodovane vrijednosti pripadaju intervalu na kom se traži rješenje procesom binarnog kodovanja decimalnih brojeva.

Dobijeni interval je podijeljen na manje intervale sa širinom intervala od 10-p, pri čemu je *p* željena preciznost i pri izvođenju eksperimenata rađeno je sa preciznošću *p=2*. Svaki broj iz manjeg intervala zamijenjen je donjom granicom tog intervala i kodovan rednim brojem tog intervala. Binarno kodovanje decimalnog broja je tada binarno kodovanje tog rednog broja.

Veza između preciznosti i broja bita potrebnih za kodovanje data je sljedećom formulom:

Pri čemu su:

* n – broj bita za kodovanje
* Gg – gornja granica intervala
* Gd – donja granica intervala
* p – tražena preciznost

Pri testiranju korišćene su sljedeće vrijednosti:

Pa se dobija da je:

U Tabeli 2.1 je dato nekoliko primjera kodovanja decimalnih brojeva iz intervala *I*.

Tabela 2.1 - Nekoliko kodova za brojeve iz intervala I

|  |  |  |
| --- | --- | --- |
| *x* | KOD | BINARNO |
| -0.356876 | 450 | 0111000010 |
| 0.248271 | 553 | 1000101001 |
| 1.093082 | 697 | 1010111001 |
| -1.506315 | 254 | 0011111110 |
| -0.912184 | 355 | 0101100011 |
| 0.882638 | 661 | 1010010101 |
| 0.804371 | 648 | 1010001000 |
| 2.168293 | 881 | 1101110001 |

Za kodovanje proizvoljnog broja iz intervala *I* je korišćena funkcija *encode* koja vraća cijeli broj po formuli:

Gdje je *x* vrijednost realnog broja koji se koduje.

Za dekodovanje dobijenih cijelih brojeva u realne brojeve koristi se funkcija *decode* koja vraća realni broj iz cijelog broja po sljedećoj formuli:

Gdje je *d* – vrijednost cijelog broja koji se dekoduje.

Nakon definisanja kodovanja potrebno je preći na izbor početne pretpostavke, tj. izbor početne populacije za genetički algoritam. Prilikom generisanja vrijednosti za početnu populaciju koristi se funkcija koja daje broj iz intervala [0, 1], te je neophodno preslikati brojeve iz ovog intervala u traženi interval *I*. Ovo se postiže po sljedećoj formuli:

Gdje je *r* – slučajno generisana vrijednost na intervalu [0, 1].

Budući da je tražena funkcija *z* funkcija od dvije promjenljive, potrebno je generisati populaciju za vrijednosti *x* i *y* koje su predstavljene sa dva niza: *population\_x, population\_y*.

Izgled početne populacije dat je u Tabeli 2.2.

Tabela 2.2 - Populacija za x i y generisane od slučajnih brojeva r\_x i r\_y sa kodovanim vrijednostima

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| R\_X | POP\_X | POP\_X\_C | R\_Y | POP\_Y | POP\_Y\_C |
| 0.426205 | -0.442771 | 436 | **0.468304** | -0.190179 | 479 |
| 0.324866 | -1.050807 | 332 | **0.695913** | 1.175475 | 711 |
| 0.060409 | -2.637545 | 61 | **0.304933** | -1.170404 | 311 |
| 0.824969 | 1.949811 | 843 | **0.045613** | -2.726324 | 46 |
| 0.546220 | 0.277319 | 558 | **0.409164** | -0.545016 | 418 |
| 0.903693 | 2.422155 | 924 | **0.756485** | 1.538908 | 773 |
| 0.438134 | -0.371194 | 448 | **0.488931** | -0.066412 | 500 |
| 0.996869 | 2.981213 | 1019 | **0.817998** | 1.907990 | 836 |

Ciklus genetičkog algoritma započinje ocjenom kvaliteta rješenja. Za ocjenu kvaliteta koristi se funkcija koja ima najmanju vrijednost za najlošije rješenje, a najveću vrijednost za najbolje najbolje rješenje. Ova funkcija se naziva *fitnes funkcija*.

Za potrebe traženja minimuma funkcije *z* koristi se sljedeća fitnes funkcija:

Dok se za potrebe traženja maksimuma funkcije *z* koristi sljedeća fitnes funkcija:

Za izbor fintes funkcije za minimum ili maksimum koristi se parametar *minOrMax.* Ukoliko je *minOrMax =* 1 biće izabrana fitnes funkcija za minimum i algoritam će tražiti minimum funkcije *z*, dok za bilo koju drugu vrijednost biće izabrana fitnes funkcija za maksimum i algoritam će tražiti maksimum funkcije *z*.

U Tabeli 2.3. dat je prikaz populacije sa vrijednostima funkcije *z* i odgovarajućim vrijednostima fitnes funkcije za traženje maksimuma.

Tabela 2.3 - Populacija sa vrijednosti funkcije i fitnes funkcijom pri traženju maksimuma

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| POP\_X | POP\_X\_C | POP\_Y | POP\_Y\_C | *z(x,y)* | *ff(x,y)* |
| -0.442771 | 436 | **-0.190179** | 479 | 2.440384 | 1.921009 |
| -0.372434 | 448 | **-0.190616** | 479 | 2.527509 | 2.008134 |
| -0.442815 | 436 | **-0.067449** | 500 | 1.922200 | 1.402825 |
| -0.524927 | 422 | **2.642229** | 962 | 0.907745 | 0.388370 |
| 0.307918 | 564 | **-0.378299** | 447 | 0.519375 | 0.000000 |
| -0.478006 | 430 | **-0.360704** | 450 | 3.104673 | 2.585298 |
| 0.307918 | 564 | **-0.378299** | 447 | 0.519375 | 0.000000 |
| -0.371194 | 448 | **-0.066412** | 500 | 2.032508 | 1.513133 |

Takođe je moguće dati ocjenu cijele populacije:

Potrebno je napomenuti da ocjena populacije ne znači nužno i bolju prilagođenost.

Nakon ocjenjivanja populacije, potrebno je odrediti roditelje za sljedeću generaciju genetičkog algoritma. Ovaj proces se naziva **selekcija**. Najbolje prilagođena jedinka ima najveću šansu da postane roditelj ili jedinka u sljedećoj generaciji. Za izbor jedinke korišćena je ruletska selekcija. Ruletska selekcija realizovana je na sljedeći način:

* svaka jedinka posjeduje vjerovatnoću izbora jedinke koja se računa po formuli:

,

* svaka jedinka „zauzima“ određeni interval na ruletu koji definiše kumulativna vjerovatnoća data sa:

Okretanje točka ruleta se simulira generisanjem pseudoslučajnog broja r iz intervala [0,1]. Tada važi ili ili . U prvom slučaju bira se hromozom rednog broja 0, dok se u drugom slučaju bira hromozom rednog broja *i*. Proces selekcije se završava kada se izabere populacija odgovarajuće veličine.

Potrebno je napomenuti da je proces selekcije realizovan u dva načina rada. Prvi način je neelitistička selekcija što podrazumijeva da će se sve jedinke sljedeće generacije birati na slučajan način. Drugi način je elitistička selekcija, tj. dvije najbolje jedinke iz prethodne generacije direktno prelaze u sljedeću, dok se ostale biraju na slučajan način. Izbor procesa selekcije vrši se pomoću parametra *elit*. Ukoliko je on postavljen na nulu (*elit = 0*) tada će biti vršena neelitistička selekcija. Ukoliko je parametar *elit* postavljen na 1, biće vršena elitistička selekcija na način da se bira 2\**elit* najbolje prilagođenih jedinki za sljedeću generaciju.

Prilikom testiranja algoritma uočeno je da se u slučaju elitističke selekcije dobijaju značajno bolji rezultati i algoritam u manje generacija daje traženo rješenje. Ovo je posljedica toga da najbolje jedinke direktno prelaze u sljedeću generaciju, ali isto tako imaju i veću vjerovatnoću prelaska, te se u narednoj generaciji dobija veći broj bolje prilagodjenih jedinki nego u slučaju neelitističke selekcije.

Sljedeća faza algoritma je **rekombinacija**. U ovoj fazi neki od hromozoma će ostati nepromijenjeni, dok će se neki ukrštati. Parovi hromozoma za ukrštanje se biraju tako što se redom uzimaju po dva hromozoma. Generiše se pseudoslučajan broj i ukoliko je on manji od vjerovatnoće rekombinacije dolazi do ukrštanja. Zatim se generiše slučajan broj *r\_x* i u 50% slučajeva se generiše slučajan broj *r\_y*, dok se u drugih 50% slučajeva *r\_y* izjednačava sa *r\_x* (*r\_y=r\_x*). Ova dva slučajna broja služe da bi se odredilo u kojoj tački hromozoma *x* i *y* će doći do rekombinacije. Rekombinacija je prikazana na sljedećem primjeru:

X Y

Prije ukrštanja:

Hromozom 1 100011010 : 1 001101 : 0111

Hromozom 2 100010000 : 0 001101 : 1101

Nakon ukrštanja:

Hromozom 1 100011010 : 0 001101 : 1101

Hromozom 2 100010000 : 1 001101 : 0111

Dijele se oba hromozoma (za *x* i  *y*) i potom im se zamijene desne strane. Ovdje su posmatrani kodovani hromozomi. Na ovaj način je od kodova 565 i 544 za *x* dobijeni kodovi 564 i 545, dok se od kodova 215 i 221 za *y* dobijaju kodovi 221 i 215. Dobijenim kodovima odgovaraju decimalne vrijednosti 0.307917 i 0.19648 za *x*, dok za *y* imamo -1.70381 i -1.739 redom.

Nakon rekombinacije, sljedeća faza algoritma je **mutacija**. Svrha mutacije je da promjenom jednog bita hromozoma unese veću varijabilnost u populaciji. Ukoliko se populacija koncentrisala u okolini lokalnog ekstrema, na ovaj način je moguće generisati hromozom koji će biti bliže globalnom ekstremu. Mutacija ne bi trebala da bude česta, pa bi njena vjerovatnoća trebala biti relativno mala. Generisanjem slučajnog broja *r* i poređenjem sa vjerovatnoćom mutacije *pm* odlučuje se da li će doći do mutacije ili ne. Ukoliko je *r* < *pm* dolazi do mutacije. U programskom kodu, *pm*  se određuje pomoću parametra *prob\_mut*.

Eksperimentalno je utvrđeno da za vrijednosti parametra *prob\_mut = 0.2* algoritam daje zadovoljavajuće rezultate po pitanju brzine pronalaska traženog rješenja i vjerovatnoće izlaska iz lokalnog ekstrema. Ovo je tačno za slučaj kada se koristi elitistička selekcija. Za slučaj neelitističke selekcije potrebno je smanjiti vjerovatnoću mutacije na *prob\_mut = 0.05*, jer u suprotnom algoritam nema optimalne performanse.

Ukoliko treba da dođe do mutacije, generiše se slučajan broj koji određuje u kom bitu hromozoma *x* će doći do mutacije i u 50% slučajeva se generiše drugi slučajan broj za *y.* U ostalih 50% slučajeva u hromozomu *y* će se vršiti mutacija na istom bitu kao i na hromozomu *x*. Nakon toga se vrijednost odgovarajućih bita komplementira

X Y

Prije mutacije:

Hromozom 1 100011010 : 1 001101 : 0 : 111

Nakon mutacije:

Hromozom 1 100011010 : 0 001101 : 1 : 101.

## Eksperimenti

Maksimalan broj iteracija za algoritam je podešen na 500 iako je algoritam u manje iteracija dolazio do rješenja. Veličina populacije je uvijek bila preko 20 jedinki jer za manje vrijednosti postoji veća vjerovatnoća da algoritam neće uspjeti da izađe iz lokalnog ekstrema. Takođe je poželjno da broj jedinki u populaciji bude paran zbog lakšeg procesa rekombinacije.

Početni eksperimenti su vršeni sa neelitističkom selekcijom. Pri neelitističkoj selekciji važno je napomenuti da vjerovatnoća mutacije ne smije biti velika jer u tom slučaju postoji previše varijabilnosti u generaciji i algoritam neće u najmanjem broju iteracija dati željeno rješenje. Parametri pomoću kojih se dobijaju zadovoljavaju rezultati su dati u tabeli 2.4.

Tabela 2.4 - Parametri neelitističke selekcije

|  |  |  |
| --- | --- | --- |
| Naziv parametra | Vrijednosti za minimum | Vrijednosti za maksimum |
| Veličina populacije | 30 | 30 |
| Maksimalan broj ponavljanja najboljeg | 20 | 20 |
| Vjerovatnoća ukrštanja | 95 % | 95 % |
| Vjerovatnoća mutacije | 5 % | 5 % |

Potrebno je napomenuti da uz date parametre algoritam daje dovoljno dobro rješenje u relativno malom broju generacija. Veličina populacije je nešto manja zbog toga što pri velikoj populaciji, zbog neelitističke selekcije, ne dođe do ponovnog izbora najboljeg elementa 20 puta za redom, što znači da će algoritam završiti sa maksimalnim brojem iteracija.

Vjerovatnoća ukrštanja je nešto veća što daje mogućnost opstanka bolje prilagođenih jedinki u populaciji, tj. njihovim potomcima. Vjerovatnoća mutacije je relativno mala zbog toga što mutacija unosi dosta nepravilnosti u populaciju i u slučaju veće mutacije algoritam neće raditi optimalno.

Nakon neelitističke selekcije vršeni su eksperimenti sa elitističkom selekcijom. Pri elitističkoj selekciji bitan je parametar *elit*, pomoću kog se određuje koliko najbolje prilagođenih jedinki će preći u sljedeću generaciju. Najbolja jedinka se ne rekombinuju, već prelazi u sljedeću generaciju bez izmjena. Eksperimenti su vršeni sa vrijednošću *elit=1*, pri čemu dvije najbolje jedinke prelaze u sljedeću generaciju. Najbolje jedinke su one za koje je vrijednost fitnes funkcije najveća. Ostali parametri su dati u tabeli 2.5.

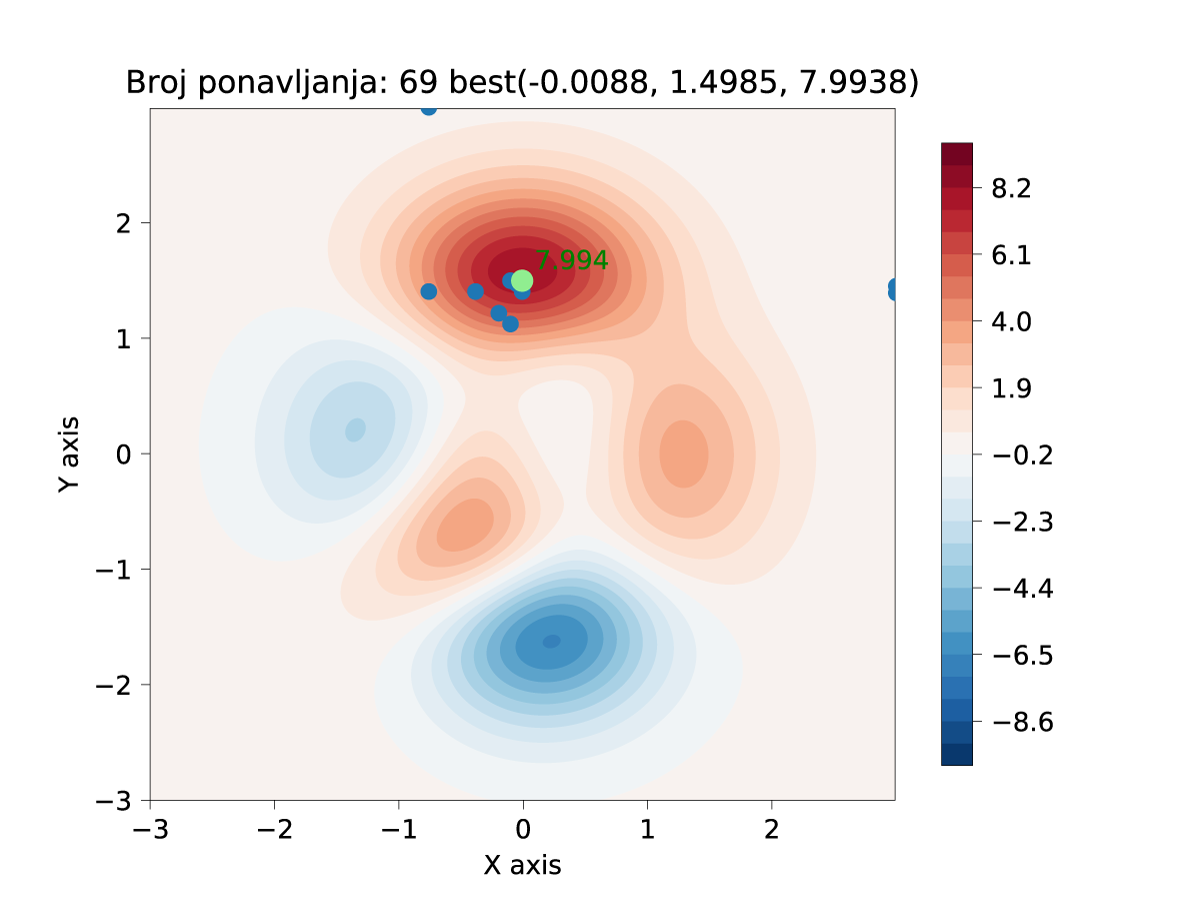
Tabela 2.5 - Parametri elitističke selekcije

|  |  |  |
| --- | --- | --- |
| Naziv parametra | Vrijednosti za minimum | Vrijednosti za maksimum |
| Veličina populacije | 60 | 60 |
| Maksimalan broj ponavljanja najboljeg | 20 | 20 |
| Vjerovatnoća ukrštanja | 95 % | 95 % |
| Vjerovatnoća mutacije | 20 % | 20 % |

Sa ovim parametrima algoritam daje najbolja rješenja po pitanju tačnosti, broja iteracija (generacija) za koje dolazi do rješenja i performansi izvršavanja. U odnosu na neelitističku selekciju, može se uočiti par razlika. Veličina populacije je veća. Ovo omogućava brže pronalaženje okoline globalnog ekstrema. Zbog toga što najbolje prilagođene jedinke direktno prolaze u narednu generaciju, algoritam nema problema sa većim brojem generacija. Vjerovatnoća mutacije je veća što osigurava brže i bolje pronalaženje preciznije vrijednosti traženog ekstrema. Mutaciju je moguće postaviti i na veću vrijednost, ali za ovo nema potrebe. Teoretski, moguće je postaviti je i na 100%, jer uvijek najbolja jedinka prelazi u sljedeću generaciju i ona nikada ne mutira. Ovo ne daje značajno poboljšanje performansi.

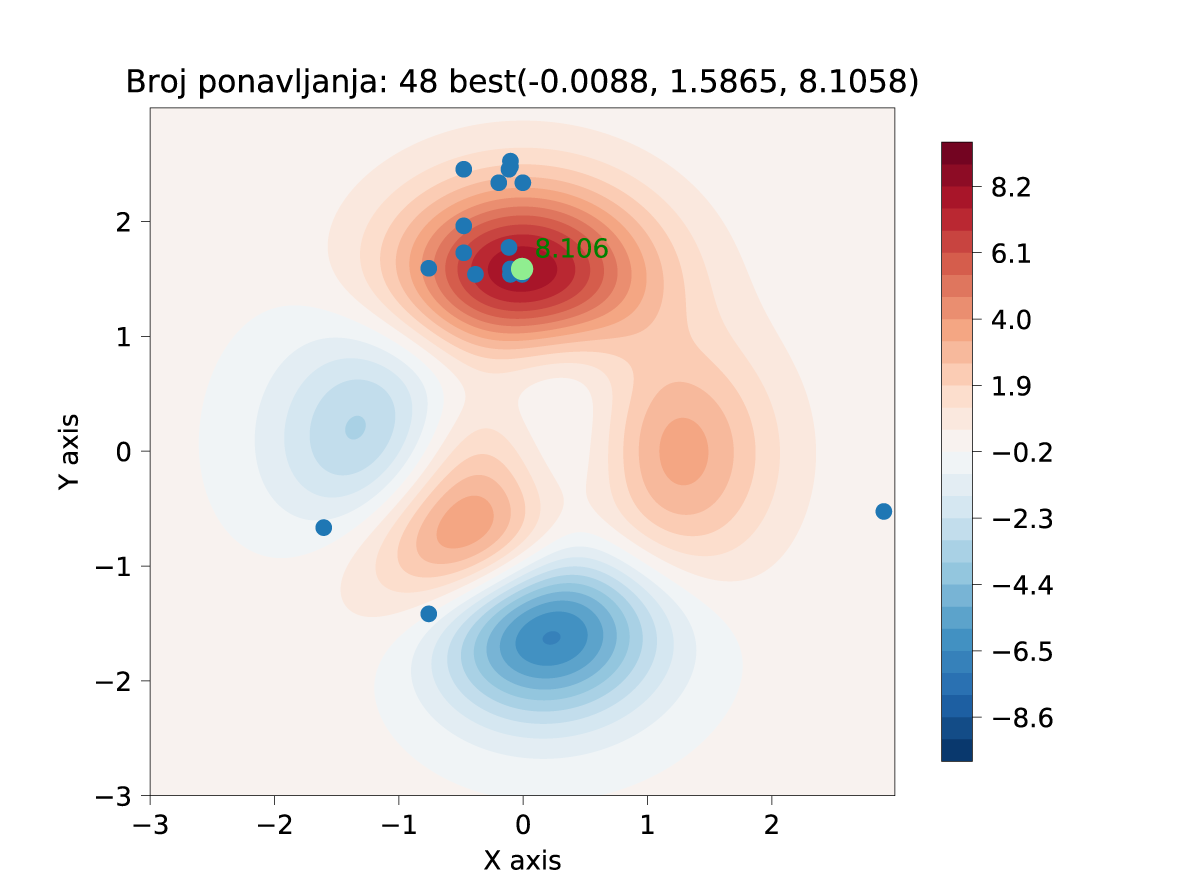
U nastavku slijedi par primjera izvršavanja algoritma.

## Primjeri izvršavanja



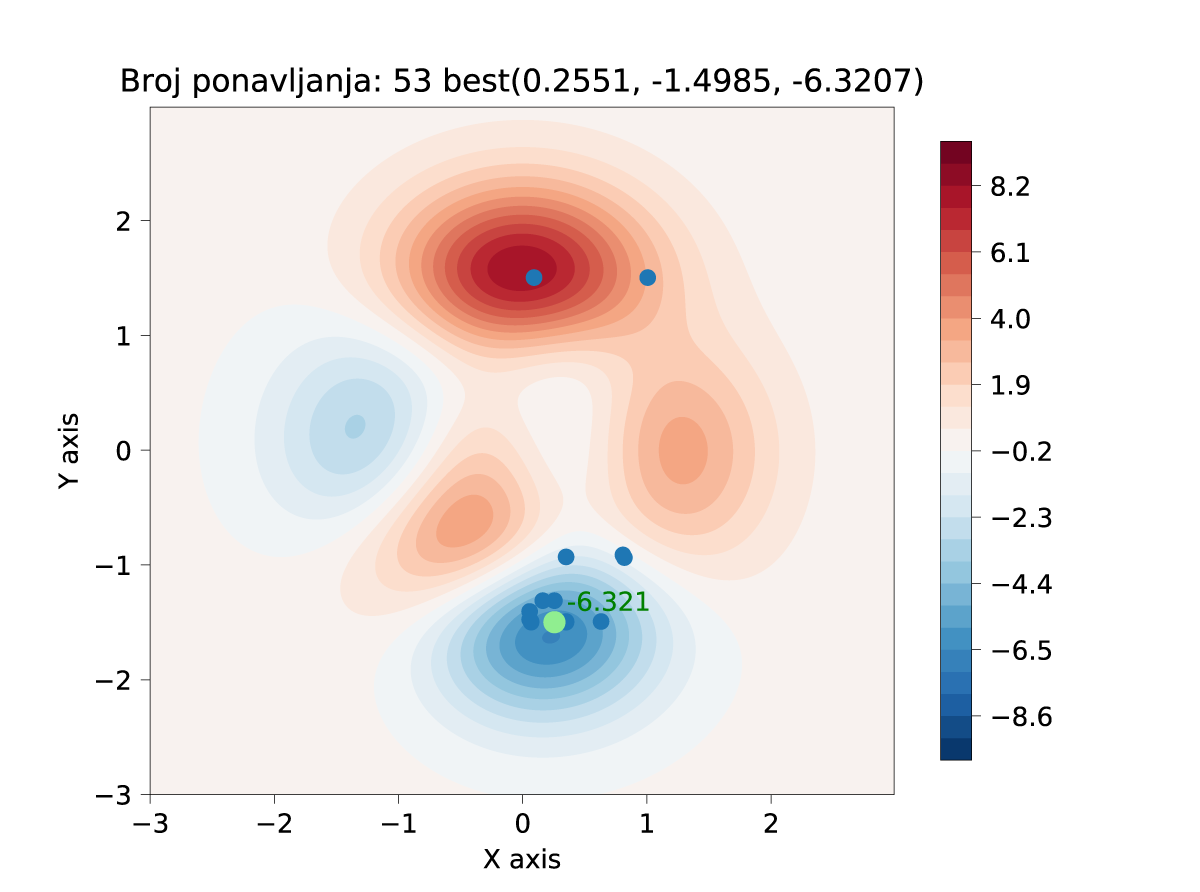
Slika 2.1 - Elitistička selekcija (maksimum) – loše rješenje

Sa slike 2.1 može se uočiti da i pri elitističkoj selekciji algoritam nekad ne daje tačno rješenje, ali zato dovoljno dobro. Najbolje rješenje je dato na slici 2.2.



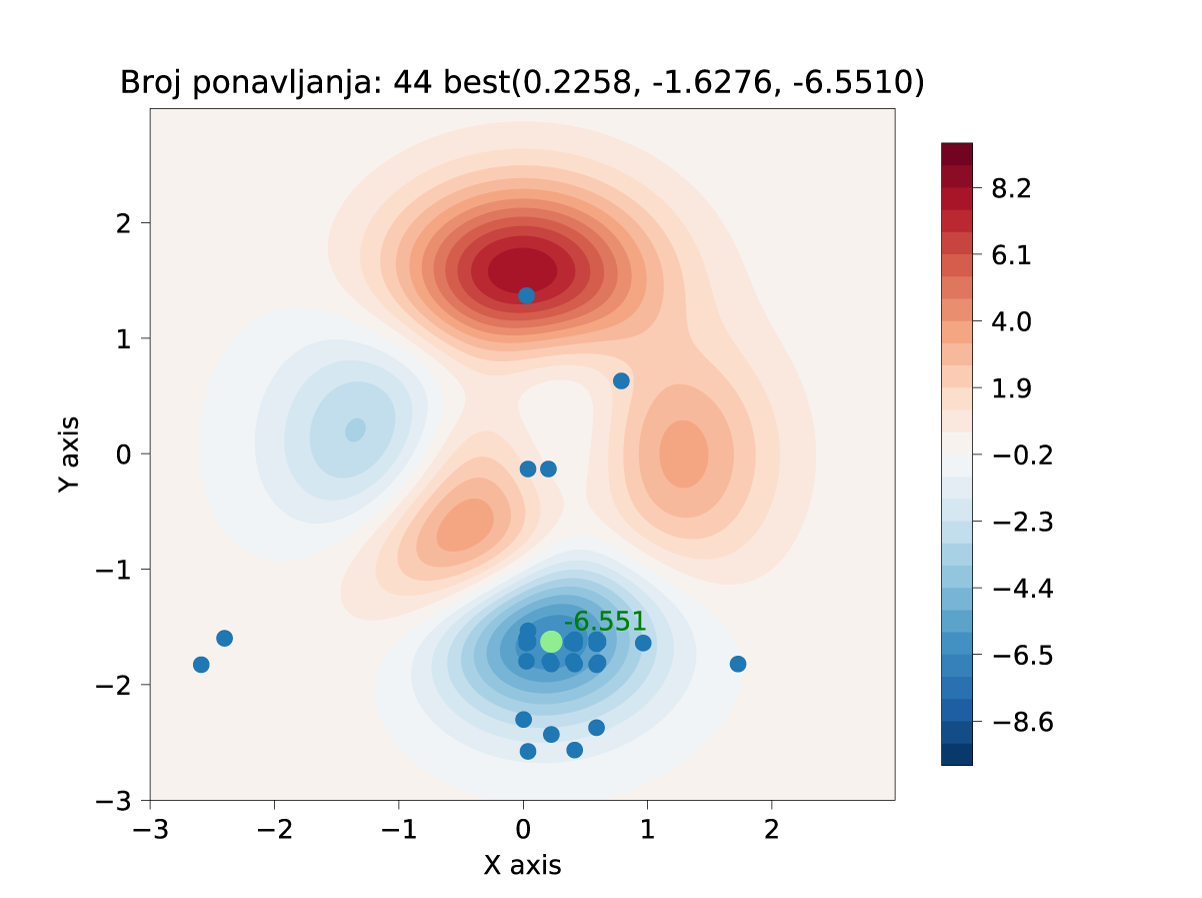
Slika 2.2 - Elitistička selekcija (maksimum) – dobro rješenje

U većini slučajeva algoritam daje najbolje rješenje u relativno malom broju generacija. Najbolje rješenje koje je dobijeno pri eksperimentima, za maksimum iznosi *8.106*.



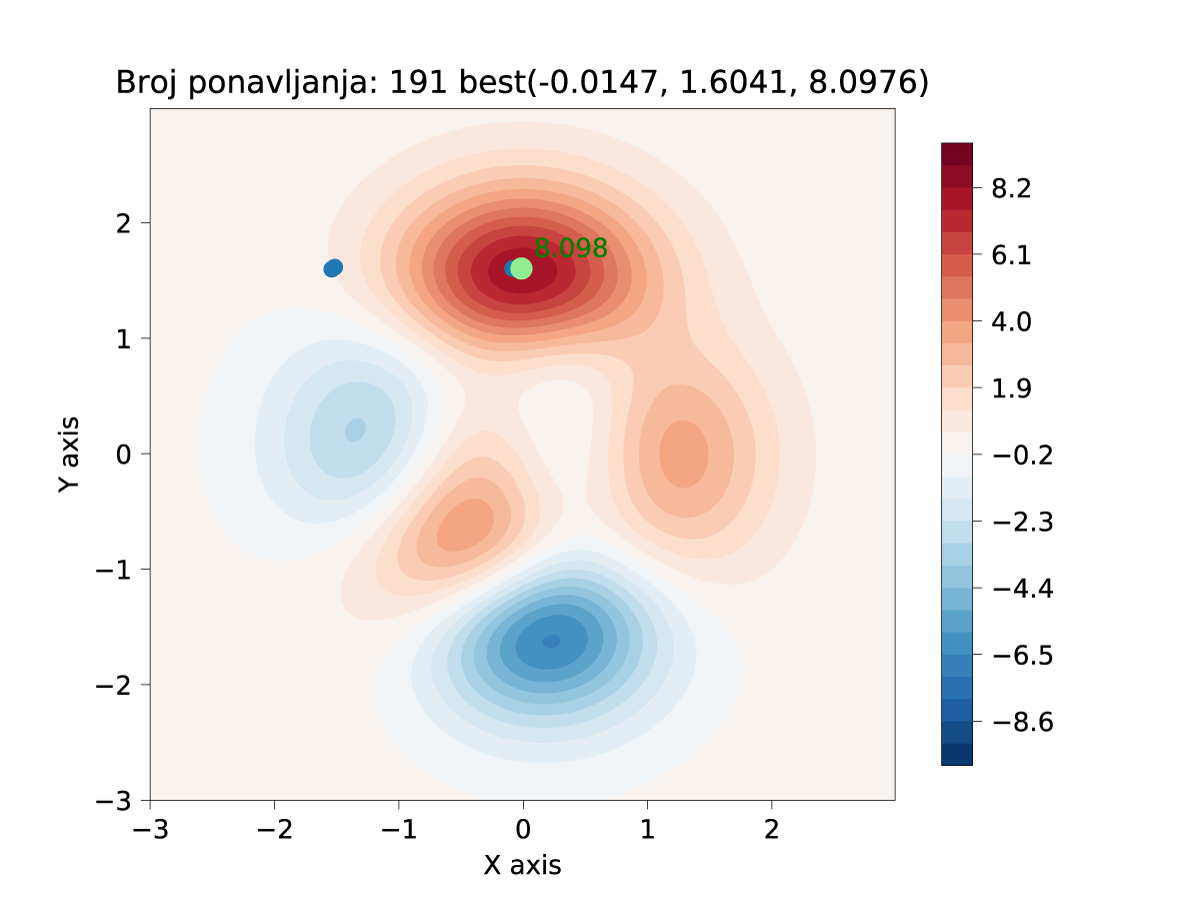
Slika 2.3 - Elitistička selekcija (minimum) – loše rješenje

Pri traženju minimuma funkcije algoritam takođe nekada ne daje tačno, ali ipak dovoljno dobro rješenje. Tačno rješenje za minimum funkcije dato je na slici 2.5.



Slika 2.4 - Elitistička selekcija (minimum) - dobro rješenje

Najbolje rješenje koje je algoritam dao za minimum tražene funkcije iznosi -6.551. Pri elitističkoj selekciji, algoritam relativno brzo dolazi do tražene vrijednosti i to u većini slučajeva.



Slika 2.5 - Neelitistička selekcija (maksimum)

Pri neelitističkoj selekciji algoritam daje zadovoljavajuće rezultate ali u znatno više iteracija.

# Zaključak

Prilikom izrade projektnog zadatka došao sam do sljedećeg zaključka. Za korektno izvršavanje algoritma najvažnije je dobro odrediti parametre i to: vjerovatnoću rekombinacije i vjerovatnoću mutacije. Ove dvije vrijednosti utiču na to da li će algoritam biti u stanju da pronađe tražene globalne ekstreme ili ne. Pomoću mutacije algoritam dobija mogućnost da u slučaju pogotka lokalnog ekstrema iz njega izađe u traženi globalni ekstrem. Ukoliko je vrijednost mutacije relativno mala, algoritam ima manje šanse da izađe iz lokalnog ekstrema. Pomoću rekombinacije se algoritam potencijalno približava traženoj vrijednosti. Što je vjerovatnoća rekombinacije veća, veće su šanse da se algoritam više približi tačnoj vrijednosti.

Što se tiče veličine populacije, uočio sam da za populacije manje od 30 algoritam ne radi dobro, tj. u dosta slučajeva pronalazi lokalni ekstrem iz koga ne može da “izađe” do globalnog ekstrema. Ovo je posebno izražajno pri traženju minimuma. Da bi se ovo izbjeglo, potrebno je povećati populaciju. Uočio sam da za populacije od 30 i više jedinki, algoritam daje zadovoljavajuća rješenja. Međutim, ukoliko je broj jedinki prevelik, performanse algoritma opadaju, tj. potrebno je više vremena za izračunavanja svih rekombinacija i mutacija. Ovo je jednim dijelom uzrokovano i generisanjem velikog broja slučajnih brojeva za potrebe određivanja tačaka mutacije i rekombinacije, ali svakako i svim izračunavanjima funkcije za populaciju i fitnes funkcije za svaki hromozom. Broj ponavljanja najbolje jedinke u populaciji sam uglavnom podešavao na 20. Na taj način sam osiguravao dovoljno dobro rješenje, ali isto tako i ne toliko pretjerano generisanje generacija. Moguće je smanjiti broj ponavljanja, ali to ne garantuje izbor najboljeg rješenja.

Ukoliko nam je bitna apsolutna preciznost algoritma, a ne i vrijeme izvršavanja, moj prijedlog bi bio povećati populaciju na nekoliko stotina. Mogao bi se smanjiti broj ponavljanja najbolje jedinke na 10 ili 15 puta, što bi algoritmu dalo šansu da pronađe najbolju jedinku, ali isto tako i da se u manjem broju generacija dođe do tačnog rješenja. Na ovaj način se povećava mogućnost da algoritam “pogodi” traženu vrijednost.

Ukoliko nam je pak bitnije vrijeme izvršavanja, a ne toliko i preciznost traženog rješenja, parametri prikazani u tabelama 2.4 i 2.5 daju dosta dobre rezultate.

Algoritam daje najbolje rezultate ukoliko se koristi elitistička selekcija.

Potencijalno unapređenje algoritma je realizacija rekombinacije u dvije tačke, ili neka modifikacija mutacije. Na ovaj način bi se moglo testirati kako različite mutacije i rekombinacije utiču na performanse algoritma.

Nedostatak algoritma je vezan za izbor parametara. Ukoliko ne poznajemo približnu vrijednost koju tražimo, teže će nam biti odrediti prave parametre. Međutim, kada su pravi parametri izabrani, algoritam u većini slučajeva daje zadovoljavajuće rješenje.