**Univerzitet u Banjoj Luci**

**Elektrotehnički fakultet**

**Katedra za automatiku**

**Metodi vještačke inteligencije**

Izvještaj o urađenom projektnom zadatku

**Genetički algoritmi**

Student:

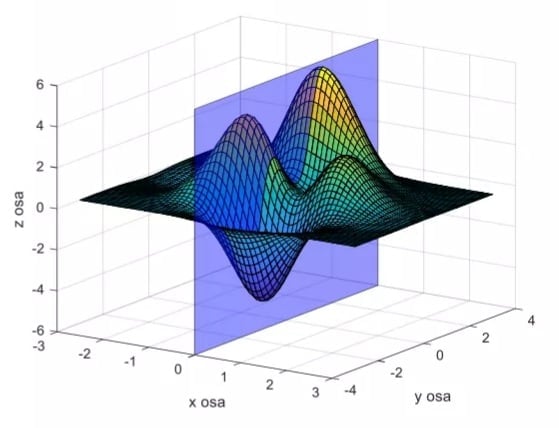
Ivana Marković, 11133/17

# Opis problema

Funkcija je data izrazom:

(1.1)

pri čemu vrijednosti promjenljivih *x* i *y* uzimaju vrijednosti iz intervala [-3, 3] i [-4, 4], respektivno.



Slika 1.1 *Grafik zadate funkcije*

Za datu funkciju je potrebno na datom intervalu odrediti globalni minimum i globalni maksimum krive koja se dobija kao presjek funkcije *z(x, y)* i ravni prikazane na slici 1.1*,* pri čemu je za rješavanje problema potrebno upotrijebiti genetički algoritam.

# Rješenje

Za pronalazak maksimuma i minimuma problem treba modelovati na odgovarajući način da bi se algoritam mogao efikasno sprovesti. Da bi se operacije ukrštanja i mutacije hromozoma izvele na pravi način, potrebno je dovesti ih u odgovarajuću formu. Najbolji način za realizaciju ovog problema je da svaki bit reprezentuje jedan gen. To radimo binarnim kodovanjem. U ovom slučaju je malo vjerovatno da će rješenje biti cjelobrojno, pa standardno binarno kodovanje cijelih brojeva ne dolazi u obzir. Kodovanje treba izvršiti tako da sve dekodovane vrijednosti pripadaju intervalu na kojem se traži rješenje. Proces se sastoji od toga da se dobijeni interval podijeli na manje intervale tako da širina tih intervala bude dovoljno mala. Ovo „dovoljno“ je određeno željenom preciznošću, tj. ako je željena preciznost na 𝑝 decimala, tada ta širina treba da bude manja od 10-p. Na osnovu preciznosti određuje se broj bita koji su potrebni za kodovanje:

(2.1)

(2.2)

Tabela 3.1*. Primjer kodovanja*

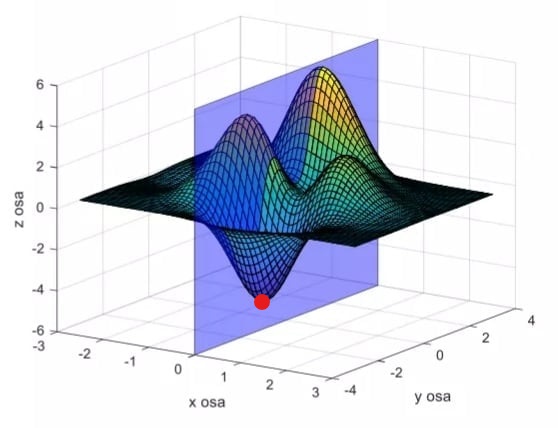
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **x** | **y** | **x\_BD** | **y\_BD** | **x\_KOD** | **y\_KOD** |
| 0.2031249 | -1.535156 | 328 | 150 | 00101001000 | 00010010110 |
| 0.4667968 | -2.6289062 | 710 | 76 | 00101100011 | 00000100110 |

Genetički algoritam se sastoji iz sljedećih koraka:

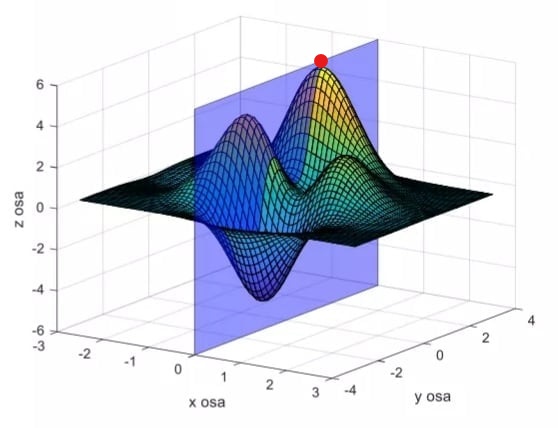
* Određivanje početne populacije (slučajan izbor)
* Procjena prilagođenosti
* Izbor najbolje prilagođenih jedinki
* Ukrštanje (rekombinacija)
* Mutacija

Neki od osnovnih parametara koji služe za određivanje toka i rezultata algoritma su: veličina populacije, broj generacija, vjerovatnoća ukrštanja, vjerovatnoća mutacije.

Algoritam započinje određivanjem početne populacije, a populaciju čini skup jedinki. U svojoj programskoj implementaciji izabrala sam veličinu populacije 100.



Slika 2.1 *Minimum zadate funkcije*



Slika 2.2 *Maksimum zadate funkcije*

Za svaku jedinku se pojedinačno računaju vrijednosti date funkcije i tako određujemo prilagođenost jedinke. Koristimo fitnes funkciju, koja je u obliku:

(2.3)

Traženje minimuma funkcije se svodi na to da nađemo jedinku koja ima najveću vrijednost za datu funkciju u jednoj populaciji. Za svaku jedinku prilagođenost predstavlja razliku između te vrijednosti i vrijednosti funkcije za tu jedinku. Na taj način će jedinka za koju funkcija ima najvišu vrijednost kao ocjenu dobiti 0, dok će najbolju ocjenu dobiti jedinka za koju funkcija ima najnižu vrijednost.

Takođe, moguće je dati i ocjenu cijele populacije. Nakon ocjenjivanja populacije, pristupa se izboru roditelja za sljedeću generaciju. To se može vršiti na više načina. U ovom primjeru rađena je ruletska selekcija.

Ruletska selekcija oponaša okretanje ruleta. Širina odjeljaka koji se nalazi na ruletu predstavljaju mjeru prilagođenosti jedinke računatu sledećom formulom:

(2.4)

gdje F predstavlja ocjenu cijele populacije. Biranjem slučajnog broja u intervalu [0, 1] simulira se okretanje točka. Obično se vrši onoliko puta koliko imamo jedinki u populaciji. Tako izračunate vjerovatnoće omogućavaju da jedinke koje su najbolje prilagođene imaju najviše šanse da budu roditelji.

*Kod korišten za ruletsku selekciju:*

public List<Individual> rouletteSelection()

{

calculateCumulativeProbability();

List<Individual> newPopulation = new ArrayList<>();

List<Individual>sortedList=population.stream().sorted(Comparator.

comparing(Individual::getQi)).collect(Collectors.toList());

double r;

for (int i = 0; i < population.size(); i++)

{

r = new Random().nextDouble();

if(r<population.get(0).getQi()) {

newPopulation.add(population.get(0).copy());

}

for(int j=1;j<population.size();j++)

if(population.get(j).qi>r && population.get(j-1).qi <= r)

{

newPopulation.add(population.get(j).copy());

break;

}

}

return newPopulation;

}

Nakon završene ruletske s

Nakon završene ruletske selekcije vrši se rekombinacija. U ovoj fazi neki od hromozoma će se ukrštati, a neki će proći nepromijenjeni, to zavisi od vjerovatnoće ukrštanja koja je zadata kao parametar.

Ovakve jedinke dalje podliježu mutaciji. Analogno odlučivanju za rekombinaciju, za mutaciju se takođe generiše slučajan broj koji se poredi sa vjerovatnoćom mutiranja, te se na osnovu toga odlučuje da li će se vršiti mutacija.

Mutacija u ovom slučaju je modelovana kao „flipovanje“ slučajno odabranog bita, odnosno njegova zamjena sa logički inverznom vrijednošću.

Ukoliko uslov bude zadovoljen i dođe do odluke da jedinka treba da mutira, mutacija se vrši na sljedeći način. Prvo se određuje gen na kome će doći do mutacije, odnosno na slučajan način se bira jedan od nbita.

Listing 2.1 *Pseudo kod genetičkog algoritma*

|  |
| --- |
| START Generate the initial population Compute fitness REPEAT  Selection  Crossover  Mutation  Compute fitness UNTIL population has converged STOP |

Programska implementacija rješenja problema je odrađena u programskom jeziku *Java*.

U prilogu se nalaze tri fajla sa izvornim kodom: *Algorithm, Simulation i Individual*.

Sa zadatim parametrima, dobili smo sledeća rješenja:

Tabela 3.2*. Rezultati izvršavanja algoritma*

|  |  |  |
| --- | --- | --- |
|  | **Min** | **Max** |
| **1.** | -4.176134 | 5.677979 |
| **2.** | -4.119934 | 5.682054 |
| **3.** | -3.639786 | 5.556114 |
| **4.** | -3.541979 | 5.660798 |
| **5.** | -3.512551 | 5.583971 |
| **6.** | -3.876454 | 5.565323 |
| **7.** | -3.789889 | 5.579812 |
| **8.** | -3.543774 | 5.665499 |
| **9.** | -3.539799 | 5.672311 |
| **10.** | -3.822181 | 5.669181 |

Računanje maksimuma funkcije vrši se na sličan način, jedina promjena bi bila u određivanju fitnes funkcije, odnosno, koristi se drugačija funkcija *F(x)* u odnosu na proces računanja minimuma.

# Zaključak

Problem je bio prilično jednostavan, ali je svakako doveo do nekih korisnih zaključaka. Dosta je pomoglo što je sličan primjer rađen i detaljno analiziran na vježbama. Algoritam je zanimljiv i prilagodljiv velikom broju problema.

Prednost algoritma je baš to što je veoma prilagodljiv i jednostavan za razne probleme, međutim, kao glavni nedostatak se može uzeti to što ni za jedan skup parametara ne može se garantovati da će rješenje biti 100% tačno.

Algoritam najbolje rješenje daje za gore pomenute parametre, povećanjem vjerovatnoće mutacije dolazi do prevelikog „skakanja“ u rezultatima. Česta praksa pri implementaciji genetičkog algoritma jeste izbjegavanje fiksne vjerovatnoće za ukrštanje i mutaciju. Jedna od mogućnosti je da se pri svakoj iteraciji računa uticaj ukrštanja i mutacije na napredak u odnosu na prethodnu generaciju, te u skladu s tim povećati vjerovatnoću za događanje one operacije koja je unijela više napretka.

Takođe, algoritam dovoljno dobro radi i sa manjom populacijom, međutim, populacija od 100 jedinki nije usporila rad izvršavanja programa, a rezultati su precizniji.

Poboljšanje algoritma je možda moguće uz promjenu selekcije. U projektnom zadatku je rađena ruletska selekcija i zaključeno je da ruletska selekcija daje dosta dobre rezultate, ali možda postoji neka selekcija koja bi to uradila bolje.

Što se tiče praktične koristi analiziranog algoritma, svodi se na to da uspješno odrađen projektni zadatak nam predstavlja dobru osnovu za rješavanja nekih sličnih problema primjenom genetičkog algoritma, a ima ih dosta.