**Univerzitet u Banjoj Luci**

**Elektrotehnički fakultet**

**Katedra za automatiku**

**Metodi vještačke inteligencije**

Izvještaj o urađenom projektnom zadatku

**Genetički algoritmi**

Student:

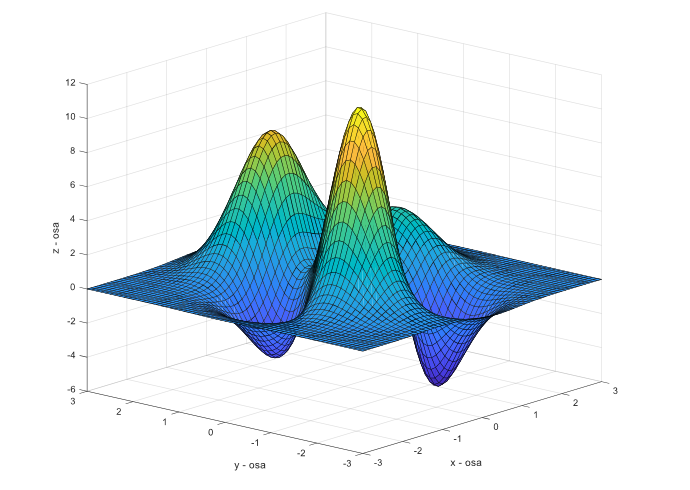
Stefan Milinčić, 1182/15

# Opis problema

Uz pomoć genetičkog algoritma za funkciju

(1.1)

na intervalu potrebno je odrediti globalni minimum i globalni maksimum.

Slika 1. *Grafik funkcije dvije promjenljive*

# Rješenje

Rješenje problema je realizovano kao konzolna aplikacija (napisana u JAVA programskom jeziku u Eclipse razvojnom okruženju) koja implementira genetički algoritam . Program omogućava unos željenih vrijednosti za odgovarajuće parametre. Određeni koraci algoritma za svaku iteraciju se ispisuju na konzoli, dok se krajnji rezultat predstavlja u obliku Message Box-a. U slučaju unosa parametara za koje program tj. algoritam ne može da pronađe rezultat biće ispisana odgovarajuća poruka na konzoli.

Kao rješenje je priložena datoteka u kojoj se nalazi Java kod sa jednim konfiguracionim fajlom config.properties i sledećim klasama:

* **config.properties** – Gd - donja granica

Gg - gornja granica

Broj\_iteracija

Velicina\_populacije

Vjerovatnoca\_ukrstanja

Vjerovatnoca\_mutiranja

Preciznost

Tip\_funkcije (min - minimum, maks - maksimum)

* **Tacka** – predstavlja klasu koja opisuje jednu tačku koja sadrži ***dvije kordinate (x,y)*** tipa double, pomoćne liste ***listaFunkcijeZ, fitnesFunkcijaZadnjeGeneracije*** tipa double, pomoćne liste ***listaNovihTacaka, listaNoveGeneracijeTacaka, zadnjaGeneracija*** tipa Tacka kao i dva stringa ***tacka1, tacka2*** i jednu promjenljivu ***redniBrojGenaNaKomeSeVrsiUkrstanje*** tipa int. Svaka od ovih navedenih atributa klase Tacka ima smisleno ime tj. njihova imena govore za šta služe i šta predstavljaju, pa nema potrebe da se svaka posebno objašnjava jer će se sve ponovo spominjati i u ostatku algoritma. Klasa sadrži podrazumijevani konstruktor, konstruktor sa 2 promjenljive tipa double, odgovarajuće setere i getere za kordinate (x,y) kao i preklopljenu metodu toString koja služi za formatiran ispis tačke. Od metoda koje su vezane za sam algoritam sadrži metodu ***konverzijaTacke*** koja služi za pretvaranje koordinata tačke u string bitova tako što primi 2 argumenta tipa double tj. 2 koordinate i uz pomoć funkcije kodovanjeBDa (koja će biti kasnije objašnjena) vrši konverziju double-a u string. Zatim klasa sadrži i funkciju ***racunajFunkciju*** koja služi za računanje određene funkcije koja je definisana u projektnom zadatku za odgovarajuću tačku tako što primi 2 koordinate tačke i uz pomoć java ugrađene klase Math i osnovnih računskih operacija dođe do rezultata koji na kraju smješta u pomoćnu listu listaFunkcijeZ. Funkcija ***ukrstanje*** služi za ukrštanje 2 tačke koje su pretvorene u string bitova. Ukrštanje se vrši u jednoj tački tako što funkcija primi 2 stringa koji predstavljaju koordinate tačke pretvorene u stringove bita kao i cjelobrojnu vrijednost brojNaKomeSeUkrsta koja predstavlja bit na kome će se desiti ukrštanje. Prvo što je potrebno uraditi je da uz pomoć klase StringBuilder i njenih funkcija append i reverse izvršimo okretanje bita tako da težine bita idu sa desne u lijevu stranu kao što je i predstavljeno u algoritmu tj. da bit najmanje težine bude skroz desno (na kraju string), a bit najveće težine da bude na početku stringa. Nakon toga uz pomoć funkcije copyValueOf dijelimo stringove na 2 dijela, gdje prvi dio predstavljaju biti od početka stringa do bita na kom se dešava ukrštanja, a drugi dio predstavljaju biti od bita ukrštanja pa do kraja stringa. Poslije podjele stringova na 2 dijela vrši se njihovo unakrsno spajanje u pomoćne stringove tacka1 i tacka 2. Funkcija ***konverzijaTackeUDecimalniZapis*** je funkcija koja radi obrnut posao u odnosu na funkciju konverzijaTacke, tj. zadatak ove funkcije je da string bitova koji reprezentuju koordinate tačke pretvore u njen decimalni zapis tj. u double. Funkcija prima 2 argumenta tipa stringa koji ustvari predstavljaju tačke koje se uz pomoć funkcije copyValueOf dijele na 2 jednaka dijela. Nakon podjele stringa vrši se njegova konverzija u cjelobrojnu vrijednost uz pomoć funkcije parseInt da bi se zatim pomoću funkcije konverzijaBDuX (koja će kasnije biti objašnjena) kreirale nove tačke i smještale u pomoćnu listu listaNovihTacaka. Funkcija ***rekombinacijaTacke*** vrši rekombinaciju 2 susjedne tačke. Funkcija kao argumente prima 2 tačke tipa Tacka koje pomoću funkcije konverzijaTacke pretvara u stringove. Nakon toga uz pomoć java ugrađene klase Random generiše 2 slučajna broja, randomTackeRekombinacije koji predstavlja na kom bitu će doći do ukrštanja i randVjerovatnoce koji predstavlja koja će biti vjerovatnoća ukrštanja. Prvo je potrebno provjeriti da li je slučajno generisana vjerovatnoća ukrštanja manja od vjerovatnoće ukrštanja koja je postavljena u konfiguracionom fajlu, ako jeste onda će se vršiti ukrštanje tako što se generiše slučajan broj koji određuje na kom bitu će se vršiti ukrštanje i poziva se funkcija ukrstanje sa odgovarajućim parametrima i funkcija konverzijaTackeUDecimalniZapis, a ako nije ispunje taj uslov neće doći do ukrštanja i pozvaće se samo funkcija konverzijaTackeUDecimalniZapis. Funkcija ***mutacija*** vrši mutaciju tački u jednom bitu. Prvo se kao i kod rekombinacije uz pomoć java ugrađene klase Random generiše 2 slučajna broja, randomTackeMutacije koji predstavlja na kom bitu će doći do mutacije i randVjerovatnoce koji predstavlja koja će biti vjerovatnoća mutacije. Funkcija ako argument prima listu tačaka kroz koju prolazi i uzima tačku po tačku i vrši određene procese. Nakon što preuzme tačku iz liste vrši njenu konverziju u string pomoću funkcije konverzijaTacke. Zatim kao i kod ukrštanja uz pomoć klase StringBuilder i njenih funkcija append i reverse izvršimo okretanje bita tako da težine bita idu sa desne u lijevu stranu kao što je i predstavljeno u algoritmu tj. da bit najmanje težine bude skroz desno (na kraju string), a bit najveće težine da bude na početku stringa. Kada smo to uradili vršimo provjeru da li je slučajno generisana vjerovatnoća mutacije manja od vjerovatnoće mutacije koja je postavljena u konfiguracionom fajlu, ako jeste onda će se vršiti mutacija tako što se prvo generiše slučajan broj koji određuje na kom bitu će se vršiti mutacija, pretvara se string u niz bita i pronalazi se odgovarajući bit na kom je potrebno izvršiti mutaciju. U slučaju da je pronađeni bit 0 on se mijenja sa 1 i obrnuto, ako je pronađeni bit 1 on će se zamijeniti sa 0. Tako mutiranu tačku uz pomoć funkcije copyValueOf dijelimo na 2 jednaka dijela da bi dobili njene koordinate . Nakon podjele stringa vrši se njegova konverzija u cjelobrojnu vrijednost uz pomoć funkcije parseInt da bi se zatim pomoću funkcije konverzijaBDuX (koja će kasnije biti objašnjena) kreirale nove tačke i smještale u pomoćnu listu listaNoveGeneracijeTacaka. U slučaju da slučajno generisana vjerovatnoća mutacije nije manja od vjerovatnoće mutacije koja je postavljena u konfiguracionom fajlu, mutacija se neće ni dogoditi i tačka će samo biti dodana u listu listaNoveGeneracijeTacaka.
* **Pomocna** – predstavlja pomoćnu klasu koja služi za izvršavanje nekih koraka algoritma i čitanje parametara iz konfiguracionog fajla. Sadrži promjenljive ***gg, gd, preciznost, velicinaPopulacije, vjerovatnocaUkrstanja, vjerovatnocaMutacije i brojIteracija*** koje su tipa double i dobijene su iz konfiguracionog fajla uz pomoć funkcije citajIzPropertijaDouble. Svaka od ovih promjenljivih ima smisleno ime, pa samim tim nije potrebno njihovo dodatno pojašnjene. Liste ***rezultatFitnesa***, ***vjerovatnocaIzbora i listaKumulativnihVjerovatnoca*** koje su tipa double, kao i lista ***lstaZaSledecuGeneraciju*** tipa integer i lista ***listaSledeceGeneracije*** tipa Tacka. Ove liste takođe imaju smislena imena, pa se može zaključiti za šta će se koristiti, ali biće i kasnije objašnjene prilikom analize funkcija. Takođe, postoji i niz tacaka ***nizTacaka*** tipa Tacka u koji se smještaju sve tačke prilikom izvršavanja algoritma i jedna cjelobrojna vrijednost ***n*** koja se dobija pomoću funkcije brojBitaN i određuje broj bita sa kojim se predstavlja jedna tačka. Postoji i jedna promjenljiva ***tipFunkcije*** koja je tipa string i dobija se pomoću funkcije citajIzPropertijaString i određuje da li se traži globalni minimum ili globalni maksimum funkcije. Tu je i pomoćna promjenljiva ***ocjenaPopulacija*** koja je tipa double i predstavlja sumu svih fitnesa funkcije. Metode ***citajIzPropertijaDouble i citajIzPropertijaString*** su pomoćne metode koje služe da se pročita sadržaj iz konfiguracionog fajla config.properties. Sadrže jedan argument nazivPromjenljive tipa string koji predstavlja naziv promjenljive u konfiguracionom fajlu. Čitanje sadržaja promjenljivih se obavlja uz pomoć ugrađenih java klasa FileInputStream i Properties i njenih metoda load i getProperty. Rezultat se smješta u promjenljivu određenog tipa. Jedina razlika između metoda citajIzPropertijaDouble i citajIzPropertijaString je ta što prva metoda vraća tip double, a druga metoda vraća tip string. Metoda ***log2*** je pomoćna metoda koja služi za računanje logaritma po bazi 2. Metoda kao argument prima double vrijednost, a kao rezultat vraća vrijednost int koja je dobijena pomoću formule (2.1):

(2.1)

Metoda ***brojBitnaN*** je metoda koja računa broj bita sa kojim će se predstaviti jedna tačka. Metoda nema argumenata, a kao rezultat vraća vrijednost Integer i računa se po formuli (2.2):

(2.2)

Metoda ***racunajBD*** je metoda koja računa redni broj intervala u kom se nalazi rješenje x tj. neka tačka. Metoda kao argument prima double vrijednost koja predstavlja neko x na osnovu koga se traži redni broj intervala, a ako rezultat vraća cjelobrojnu vrijednost int. Računa se po formuli (2.3):

(2.3)

Metoda ***kodovanjeBDa*** je pomoćna metoda koja pretvara odgovarajući bd u string bitova. Kao argument prima cjelobrojnu vrijednost int koja ustvari predstavlja određeni bd koji treba konvertovatni u string bitova, a kao reultat vraća string. Pretvaranje cjelobrojne vrijednosti u string se vrši pomoću java ugrađene klase Integer i njene funkcije toBinaryString. Metoda ***konverzijaBDuX*** je metoda koja računa realan broj x za određeni bd. Metoda kao argument prima cjelobrojna vrijednost int koja predstavlja bd, a kao rezultat se dobija double vrijednost i računa se po formuli (2.4):

(2.4)

Metoda ***funkcija*** je funkcija koja služi za računanje preslikavanja sa intervala na interval . Metoda kao rezultat vraća tip double, a kao argument prima vrijednost tipa double koje predstavlja neki broj iz intervala i računa se po formuli (2.5):

(2.5)

Metoda ***izborPocetnePopulacije*** je metoda koja služi za određivanje početne populacije tačaka tj. tačaka koje ulaze u prvu generaciju. Metoda sadrži for petlju uz pomoć koje kreiramo onoliki broj tačaka kolika je populacija koja je određena u konfiguracionom fajlu. Uz pomoć ugrađene java klase Random i prethodno opisane funcije funkcija vršimo generisanje tačaka uz pomoć konstruktora sa 2 argumenta koje kao rezultat smještamo u niz tačaka nizTacaka. Metoda ***fitnesFunkcijaMin***  je metoda koja služi za računanje funkcije prilagođenosti (dobrote) za traženje globalnog minimuma. Kao argument prima listu tipa double, dok kao rezultat vraća listu koja je takođe tipa double. Metoda sadrži pomoćnu promjenljivu maks tipa double koja služi za dohvatanje prvog elementa iz liste koja je proslijeđena kao argument funkcije i jednu for petlju koja služi za prolazak kroz sve elemente liste. Fitnes funkcija se računa pomoću formule (2.6) i rezultat se smješta u listu rezultatFitnesa, dok se ukupna ocjena populacije računa po formuli (2.7) i rezultat se smješta u listu ocjenaPopulacije. Ova funkcija je izabrana jer daje najveću vrijednost za najbolje prilagođenu jedinku, a nultu vrijednost za najlošije prilagođenu jedinku.

(2.6)

(2.7)

Metoda ***fitnesFunkcijaMaks*** je metoda koja služi za računanje funkcije prilagođenosti (dobrote) za traženje globalnog maksimuma. Sve je isto kao i kod metode fitnesFunkcijaMaks samo što se razlikuje formula (2.8) za računanje fitnes funkcije.

(2.8)

Metoda ***kumulativnaVjerovatnocaMin*** je metoda koja računa kumulativnu vjerovatnoću za traženje globalnog minimuma funkcije. Kao argument prima listu tipa double, a kao rezultat takođe vraća listu tipa double. Metoda sadrži jednu pomoćnu promjenljivu maks tipa double koja služi za dohvatanje prvog elementa liste koja je proslijeđena kao argument metode kao i još jednu pomoćnu promjenljivu suma tipa double koja služi za računanje kumulativnih vjerovatnoća na osnovu prethodno izračunatih vjerovatnoća. Kumulativna vjerovatnoća se računa tako što postoji for petlja koja prolazi kroz sve element proslijeđene liste i uz pomoć formule (2.9) koja je implementirana u funkciji vjerovatnocaIzboraJedinke računa vjerovatnoću izbora za svaku jedinku i smješta je kao rezultat u listu vjerovatnocaIzbora tipa double, dok se suma tj. kumulativna vjerovatnoaća računa pomoću formule (2.10) ili (2.11) i dodaje u listu listaKumulativnihVjerovatnoca tipa double.

(2.9)

(2.10)

(2.11)

Metoda ***kumulativnaVjerovatnocaMaks*** je metoda koja računa kumulativnu vjerovatnoću za traženje globalnog maksimuma funkcije. Metoda je implementirana isto kao i metoda kumulativnaVjerovatnocaMin samo što je prilagođena da traži globalni maksimum, a ne globalni minimum. Metoda ***vjerovatnocaIzboraJedinke*** je metoda koja služi za računanje vjerovatnoće izbora za određenu jedinku. Povratni tip metode je double, dok kao argument prima tip double koji predstavlja fitnes funkciju. Vjerovatnoća izbora se računa uz pomoć formule (2.9) tj. kao količnik fitnes funkcije i ukupne ocjene populacije. Metoda ***izborSelekcije*** je metoda koja određuje koja će jedinka tj. u našem slučaju tačka ići u sledeću generaciju. Metoda vraća listu tipa integer koja sadrži indekse tačaka koje su prošle u sledeću generaciju, a kao argument prihvata listu koja je tipa double. Metoda sadrži 2 for petlje od kojih prva služi za prolazak kroz sve jedinke u populaciji, a druga za prolazak kroz listu kumulativnih vjerovatnoća gdje se uz pomoć java ugrađene klase Random i njene ugrađene funkcije nextDouble generiše slučajan broj iz opsega koji se zatim provjerava da li je manji od neke kumulativne vjerovatnoće iz liste i u slučaju da jeste indeks te tačke se dodaje u listu listaZaSledecuGeneraciju koja je tipa integer i tu se prekida ta druga for petlja, a u slučaju da nije ispunje taj uslov for petlja se izvršava dok se ne pronađe odgovarajuća kumulativna vjerovatnoća koja ispunjava ovaj prethodni uslov. Metoda ***praviSledecuGeneraciju*** je metoda koja služi za pravljenje nove liste tačaka za sledeću generaciju na osnovu prethodno dobijenih indeksa. Metoda ne vraća ništa, tj. povratni tip joj je void, a kao argument prima listu tipa integer. Metoda sadrži for petlju koja se koristi za prolazak kroz sve elemente populacije i u njoj se nalazi pomoćna promjenljiva indeks tipa int koja služi za prikupljanje indeksa iz liste koja je proslijeđena kroz argument metode. Na osnovu dobijenog indeksa tačka se pronalazi u nizu tačaka nizTacaka i dodaje se u listu listaSledeceGeneracije koja je tipa Tacka. Metoda ***mijesanjeTacaka*** je metoda koja na slučajan način vrši miješanje tačaka tj. vrši zamjenu njihovih indeksa. Metoda kao argument prihvata listu koja je tipa Tacka, a povratni tip joj je void. Sadrži while petlju koja će izvršiti broj promjena koji je jedna polovini ukupne veličine populacije. U while petlji ima 2 pomoćne promjenljive indeks1 i indeks2 koje su tipa int i čije se vrijednosti određuju na slučajan način uz pomoć java ugrađene klase Random i njene funkcije nextInt, a služe za pamćenje pozicija dvije tačke. Nakon generisanja indeksa dvije tačke koje će izvršiti zamjenu mjesta kreira se jedna pomoćna tačka t u koju se smješta tačka koja odgovara indeksu1 i uz pomoću java ugrađene funkcije set za klasu ArrayList na njenu poziciju se postavlja tačka koja odgovara indeksu2, a na poziciju indeks2 se postavlja tačka t gdje smo na taj način izvršili njihovu zamjenu.

* **Main –** klasa je klasa u kojoj se vrši izvršavanje cijelog programa i samog algoritma. Sadrži pomoćnu metodu ***prikaziRezultat*** koja služi samo za prikazivanje rezultata u obliku informacionog Message Box-a. Kao argument prima string koji predstavlja poruku koja će biti ispisana. Izvršavanje algoritma započinje tako što se prvo pozove funkcija ***izborPocetnePopulacije*** koja generiše početnu populaciju tačaka. Nakon toga dolazi for petlja koja služi za prolazak kroz određeni broj iteracije koji je definisan u konfiguracionom fajlu. U toj for petlji se izvršava i cijeli algoritam tako što se prvo prolazi pomoćnom for petljom kroz sve tačke i računa odgovarajuća funkcija pomoću funkcije ***računajFunkcijuZ*** koja kao argument prima koordinate tačke. Posle toga se provjerava da li se traži globalni minimum ili globalni maksimum. U slučaju da korisnik u konfiguracionom fajlu izabere da se traži globalni minimum prvo što će se desiti je sortiranje liste opadajuće, a zatim se pozivaju dvije funkcije ***fitnesFunkcijaMin*** koja kao argument prima listu listaFunkcijeZi  ***kumulativnaVjerovatnocaMin*** koja kao argument takođe prima listu listaFunkcijeZ, respektivno. Ako korinsnik u konfiguracionom fajlu izabere da se traži globalni maksimum lista će se sortirati rastuće, a onda će se pozvati dvije funkcije ***fitnesFunkcijaMaks*** koja kao argument prima listu listaFunkcijeZi  ***kumulativnaVjerovatnocaMaks*** koja kao argument takođe prima listu listaFunkcijeZ, respektivno. Nakon toga imam jednu provjeru koja nam služi da odredimo fitnes funkciju poslednje generacije koja će nam biti od koristi prilikom određivanja krajnjeg rješenja. Ako je u pitanju poslednja generacija onda se ona dodaje u listu fitnesFunkcijaZadnjeGeneracije. Nakon odrađenog jednog dijela algoritma potrebno je izvršiti čišćenje određenih listi koje se koriste u svakoj od iteracija. Liste listaFunkcijeZ i rezultatFitnesa se prve prazne tj. čiste i ukupna ocjena populacije se restartuje na 0. Zatim se određuju indeksi tačaka koje prolaze u sledeću generaciju uz pomoć funkcije izborSelekcije koja kao argument prima listu listaKumulativnihVjerovatnoca. Nakon izabranih indeksa prelazi se na korak kreiranja tačaka na osnovu tih indeksa. To se vrši pomoću funkcije praviSledecuGeneraciju kojoj se prosleđuje lista listaZaSledecuGeneraciju. Posle ovog koraka se čiste liste listaZaSledecuGeneraciju, listaKumulativnihVjerovatnoca i vjerovatnocaIzbora. Sledeće što se radi je miješanje tih tačaka uz pomoć funkcije mijesanjeTacaka koja prima listu listaSledeceGeneracije koja se odmah potom i čisti. Za tako izmiješane tačke vrši se njihova rekombinacija korištenjem funkcije rekombinacijaTacke koja kao argumet prima 2 susjedne tačke iz izmiješane populacije i vrši njihovu rekombinaciju. U slučaju da je veličina populacije neparna tj. da ima neparan broj tačaka u populaciji potrebno je dodati poslednju tačku koja nije ušla u rekombinaciju, jer nije imala svoj par, u sledeću generaciju kako ta tačka ne bih ostala neiskorištena. Nakon izvršene rekombinacije tačaka vrši se i njihova mutacija uz pomoć funkcije mutacija kojoj se prosleđuje lista listaNovihTacaka koja ustvari predstavlja listu tačaka kod kojih je izvršena rekombinacija. Nakon izvršenja i ovog koraka algoritma potrebno je još samo dodati u početni niz tačaka nizTacaka novonastale tačke koje predstavljaju jednu iteraciju algoritma i izvršiti čišćenje preostalih listi koji nisu očišćene, a to su listaNovihTacaka i listaNoveGeneracijeTacaka. Sve ovo će se izvršavati u for petlji određeni broj iteracija koji nam je definisan u konfiguracionom fajlu. Prilikom zadnje iteracije algoritma moram zapamtiti te tačke u listu zadnjaGeneracija kako bi nam pomogle da dođemo do krajnjeg rezultata. Kada smo izvršili sve iteracije algoritma tj. dobili tačke zadnje generacije potrebno je još samo odrediti krajnji rezultat. Ono što uradimo je da za tu zadnju generaciju tačaka izračunamo vrijednost funkcije za sve te tačke pomoću funkcije racunjaFunkcijuZ i u zavisnosti od toga da li se traži globalni minimum ili globalni maksimum

pronađemo najmanju vrijednost odnosno najveću vrijednost i tačku u kojoj je ta vrijednost dobijena i to proslijedimo funkciji prikaziRezultat koja nam ispiše krajnji rezultat.

Nakon što je završeno objašnjavanje algoritma i programskog koda kojim je isti realizovan izvršićemo par eksperimenata kako bi utvrdili za koje parametre algoritam daje optimalna rješenja. Eksperimente ćemo predstaviti tabelarno. U tabeli nećemo predstavljati sve parametre tj. neće se predstavljati oni parametri koji se neće mijenajti kao što su gornja (3) i donja granica (-3) intervala, preciznost (2) i broj iteracija algoritma (500). Za svaku vrijednosti parametara program će se izvršavati po 5 puta i gledaće se srednja vrijednost tih rezultata.

Tabela 2.1 *Prikaz izvršenog eksperimenta za globalni minimum*

|  |  |  |  |
| --- | --- | --- | --- |
| **Vjerovatnoća ukrštanja** | **Vjerovatnoća mutacije** | **Veličina populacije** | **Srednja vrijednost rezultata** |
| 0.50 | 0.10 | 8 | -2.3988771250 |
| 0.85 | 0.10 | 8 | -2.3444966491 |
| 0.30 | 0.70 | 8 | -3.0729971600 |
| 0.85 | 0.70 | 8 | -2.7175853241 |
| 0.50 | 0.10 | 32 | -3.3567254302 |
| 0.85 | 0.10 | 32 | -3.4078037047 |
| 0.30 | 0.70 | 32 | -2.8819000804 |
| 0.85 | 0.70 | 32 | -3.2510460920 |
| 0.50 | 0.10 | 200 | -6.3912799520 |
| 0.85 | 0.10 | 200 | -4.8409718427 |
| 0.30 | 0.70 | 200 | -5.1220844773 |
| 0.85 | 0.70 | 200 | -5.3712279159 |

Tabela 2.2 *Prikaz izvršenog eksperimenta za globalni maksimum*

|  |  |  |  |
| --- | --- | --- | --- |
| **Vjerovatnoća ukrštanja** | **Vjerovatnoća mutacije** | **Veličina populacije** | **Srednja vrijednost rezultata** |
| 0.50 | 0.10 | 8 | 4.0459538645 |
| 0.85 | 0.10 | 8 | 4.4328261549 |
| 0.30 | 0.70 | 8 | 4.8314391458 |
| 0.85 | 0.70 | 8 | 4.4662217410 |
| 0.50 | 0.10 | 32 | 5.3184320705 |
| 0.85 | 0.10 | 32 | 4.7921760986 |
| 0.30 | 0.70 | 32 | 3.3962463593 |
| 0.85 | 0.70 | 32 | 7.0517237418 |
| 0.50 | 0.10 | 200 | 7.8669451760 |
| 0.85 | 0.10 | 200 | 5.9619572579 |
| 0.30 | 0.70 | 200 | 7.0878400078 |
| 0.85 | 0.70 | 200 | 7.2755376997 |

Nakon izvršenih eksperimenata za globalni minimum i maksimu koji su prikazani u tabelama

2.1 i 2.2, reskeptivno, možemo zaključiti da određeni parametri koje smo mijenjali tokom eksperimentna različito utiču na krajnji rezultat. Što se tiče rezultata koje smo dobili, iz tabele se može zaključiti da se za manje veličine populacije dobijaju bolja rješenja kad je vjerovatnoća mutacije velika (oko 70%) jer jedinke početne populacije se rijetko kad nalaze blizu globalnog minimuma/maksimuma, pa je zbog toga potrebno da se mutacija češće izvršava tj. da njena vjerovatnoća bude velika kako bi se populacija izbacila iz ravnotežnog stanja i da se jedinke udalje od ostalih jedinki kako bi se izbjegla kocentracija oko nekog lokalnog minimuma odnosno lokalnog maksimuma. Na veoma maloj veličini populacije moramo imati dosta sreće da se nađe globalni minimum jer sve zavisi od generisanja slučajnih brojeva tj. da li će biti sreće da se tačke baš generišu blizu globalnog minimuma. Na velikim veličinama populacije bolja rješenja se dobijaju kada je vjerovatnoća mutacije manja (oko 10%) jer je velika vjerovatnoća da početna populacija sadrži jedinke u blizini globalnog minimuma odnosno globalnog maksimuma i nema potrebe da se jedinke udaljavaju jedna od druge. Što se tiče vjerovatnoće ukrštanja, iz eksperimenta se vidjelo da se najbolja rješenja i za globalni minimum i za globalni maksimum dobijaju kada je vjerovatnoća ukrštanja ni prevelika, a ni premala tj. najbolji rezultati su se dobijali kad je bila oko 50%. Ovo nam govori da sa velikom vjerovatnoćom ukrštanja možemo pretjerati jer dobro prilagođene jedinke nema potrebe ukrštati, dok sa malom vjerovatnoćom ukrštanja možemo rizikovati da nećemo dobiti dobro rješenje pogotovo ako je veličina populacije mala.

# Zaključak

Genetički algoritam predstavlja evolutivni algoritam koji ima široku primjenu, pogotovo u

matematici i biologiji. Pomoću genetičkih algoritma se rješavaju kompleksni kombinatorni i organizacioni problemi koristeći analogiju evolucije u prirodi. Ostvareni rezultati, iako pokazuju zadovoljavajuća rješenja, ne osiguravaju uvijek nalaženje optimalnog rješenja jer rješenje zavisi od izabranih parametara i ono daje skup rješenja čija je veličina jednaka veličini populacije. Prednosti genetičkog algoritma su veliki stepen slobode koji se odnosi na to da je moguće povećati efikasnost algoritma promjenama određenih parametara, ponavljanjem algoritma se može dobijati bolje rješenje, programska podrška gdje postoji dosta programa za optimizaciju funkcija uz pomoć genetičkog algoritma, mogućnost ugrađivanja bilo kojih informacija u hromozom koja nam je od velike pomoći u slučajevima kada ne možemo rješenje dobiti analitičkim postupcima, mogućnost optimizacije svake funkcije prilagođenosti bez obzira na njenu neprekidnost i diferencijabilnost. Neki od nedostataka genetičkog algoritma su spora konvergencija, nema garancije da je dobijeno rješenje optimalno ili blizu optimalnog, nikada nam ne nude 100% tačno rješenje, težina određivanja funkcije prilagođenosti. Prilkom izrade ovog algoritma i izvršenih eksperimenata može se zaključiti da algoritam daje prilično dobra rješenja za veće populacije, vjerovatnoću ukrštanja oko 50% i vjerovatnoću mutacije oko 10%. Za globalni minimum je dobijeno rješenje oko -6.39, a za globalni maksimum oko 7.86, a kao što je rečeno da algoritam nikada ne nudi 100% tačno rješenje tako i ovo nisu najbolja rješenja jer se mogu dobiti i bolja rješenja, ali ovo su srednje vrijednosti dobijene prilikom 5 mjerenja.