SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 4372

Vizualizacija ponašanja populacije jedinki prilikom procesa optimizacije genetskim algoritmom

Tvrtko Zadro

Zagreb, lipanj 2016.

Umjesto ove stranice ide izvornik rada.

SADRŽAJ

1. Uvod

Genetski algoritmi heuristička su metoda pretraživanja prostora rješenja u potrazi za optimalnim rješenjem. Podskup su evolucijskih algoritama i inspirirani su procesom evolucije. Iz skupa genetskih algoritama odabran je jedan algoritam te je ostvaren u programskom jeziku Java. Programsko ostvarenje je ispitano problemom funkcije s dvije varijable. Izabrani algoritam ima troturnirsku selekciju.

U drugom poglavlju navedena je podjela genetskih algoritama, dok je u trećem detaljno prikazan algoritam troturnirske selekcije. U četvrtom poglavlju opisano je programsko ostvarenje navedenog algoritma, a u petom poglavlju rezultati testiranja programski ostvarenog rješenja.

2. Uvod u genetske algoritme

Genetski algoritmi (eng. *Genetic algorithms*) su, kao dio **evolucijskih algoritama** (eng. *Evolutionary algorithms*), **metaheuristički optimizacijski algoritmi** (eng. *Metaheuristic optimization algorithms*).

2.1. Evolucijski algoritmi

Genetski algoritmi spadaju u veću skupinu evolucijskih algoritama i kao takvi nasljeđuju neke osnovne principe na kojima funkcioniraju evolucijski algoritmi.

Simuliranje evolucije znači da algoritmi uključuju primjenu bioloških metoda **reprodukcije**, **mutacije**, **rekombinacije**, **prirodne selekcije** i **preživljavanje najboljih**. U populaciji jedinku predstavlja jedno **moguće rješenje**. Kako se radi o problemu koje rješava računalo, svaka jedinka (moguće rješenje) ima svoju **dobrotu** (eng. *fitness*), koja se računa **funkcijom dobrote** (eng. *fitness function*).

Kao evolucijski algoritmi, genetski algoritmi oponašaju proces prirodne selekcije. Ovaj princip heurističkog pretraživanja koristi se kako bi se brzo pronašla rješenja kod problema u kojima klasične metode daju loše ili nikakvo rješenje. To se postiže žrtvovanjem optimalnosti, točnosti i preciznosti kako bi se postigla brzina. Ovaj način usmjerenog slučajnog pretraživanja prostora rješenja (eng. *guided random search technique*) u potrazi za optimumom zbog navedenog može u više pokretanja vratiti drugačije rješenje. *Slika 2.1* prikazuje primjer primjene genetskih algoritama.

Slika 2.1 NASA-ina antena iz 2006. godine čiji je oblik nađen genetskim algoritmom dizajniranim za pronalazak oblika antene koji će omogućiti najbolje radio valove

2.2. Prednosti

Prednost genetskih algoritama je to što nemaju pretpostavke o izgledu domene pretrage nad kojim se provode, pa im to daje široku primjenu u raznovrsnim područjima, npr. ekonomiji, umjetnosti, marketingu, sociologiji, robotici, fizici, kemiji, itd.

2.3. Ograničenja

U odnosu na druge optimizacijske algoritme, genetski algoritmi imaju neka ograničenja.

- Stalni izračun funkcije dobrote može biti jako vremenski zahtjevan, pa se u nekim slučajevima moraju koristiti aproksimacije dobrote.
- Bolje rješenje je samo relativno bolje u odnosu na druga rješenja, pa u nekim problemima gornja granica dobrote, tj. granica dobrog rješenja nije jasna.
    - Genetski algoritmi ne mogu rješavati probleme u kojima funkcija dobrote vraća samo dobro/loše ili true/false, jer u takvim slučajevima nema "brda na koje se može penjati" (doduše, u slučajevima u kojima se taj proces ponavlja, vraćajući više takvih vrijednosti, dobrotu može predstavljati omjer uspjeha i neuspjeha).
- U nekim problemima genetski algoritmi konvergiraju prema nekim lokalnim optimumima, umjesto globalnim, što se teoretski može riješiti drugačijom funkcijom dobrote, većom vjerojatnošću za mutacijom i metodom selekcije koja će zadržati veću raznolikost u populaciji.
- Za neke optimizacije, brže rezulate mogu dati neki drugi algoritmi navedeni u ovom radu.

2.4. Podjela

Mijenjanjem nekih od aspekata genetskih algoritama možemo utjecati na način na koji radi čime ga možemo bolje prekrojiti našim potrebama.

Elitizam unutar genetskog algoritma dopušta najboljim jedinkama unutar populacije da budu imune na bilokakve promjene ili eliminacije. Ovo garantira da rješenju dobivenom

genetskim algoritmom neće padati kvaliteta kroz iteracije.

Turnirksa selekcija je način odabira jedinki iz populacije. Uključuje nekoliko turnira između nasumično odabranih jedinki. Pobjednici svakog turnira su odabrani za križanje. Veličina turnira utječe na utisak koji će ostaviti selekcija. U većim turnirima jedinke s malom dobrotom imaju manje šanse proći u sljedeću iteraciju.

3. Troturnirska selekcija

Za ovaj rad odabran je genetski algoritam sa tročlanom turnirskom selekcijom.

3.1. Princip rada

Kao što je navedeno u uvodu u genetskim algoritme, **jedinka populacije** predstavlja jedno moguće rješenje. U **genu** jedinke nalazi se spremljen zapis rješenja, koje se mora moći mijenjati i kombinirati.

Evolucija počinje od nasumično generirane populacije zadane veličine. Populacija u svakoj iteraciji postupka zove se **generacija**. U svakoj generaciji izračunava se **dobrota** svake jedinke pomoću **funkcije dobrote**. Nakon određenog broja operacija postupak se ponavlja, iz generacije u generaciju. Zaustavljanje algoritma može biti izazvano nekim od uvjeta koji će biti definirani u nastavku.

Da bi se određeni problem riješio genetskim algoritmom potrebno je osmisliti dvije bitne komponente:

1. Način da se rješenje zapiše u "obliku gena".
2. Funkcija dobrote kojom bi se izračunala dobrota svake jedinke populacije.

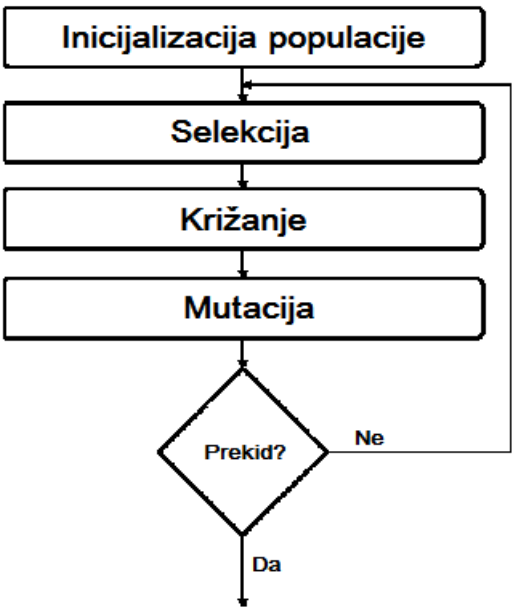
Standardna reprezentacija rješenja (gena) je niz bitova (jedinica i nula), ali se može prikazati i bilokojim drugim nizom zadanih tipova ili struktura. Glavno svojstvo gena mora biti mogućnost da se dijelovi mijenjaju i miješaju. Tako je ostvaren gen i kod živih bića. DNK je linearan niz fiksne duljine elemenata koji se mogu premiještati. Neki algoritmi, kao što su algoritmi evolucijskog programiranja, isprobavaju mogućnosti gena koji su oblika stabla, a ne linearni.

Nakon što su osmišljene te dvije potrebne komponente karakteristične za svaki problem, algoritam nastavlja svojim zadanom procedurom opisanom sljedećim postupkom ili ukratko slikom 3.1.

Koraci genetskog algoritma:

1. Inicijalizira se populacija
2. Računa se dobrota jedinki pomoću funkcije dobrote
3. Nasumično se odabiru 3 jedinke koje ulaze u turnirsku selekciju
    a. Jedinke se rangiraju po dobroti
    b. Geni najlošije jedinke se mijenjaju križanjem druge dvije jedinke
4. Provodi se mutacija
5. Prekida se algoritam ili se nastavlja od koraka 2

Osim inicijalne populacije, računanja dobrote u svakoj iteraciji postupka i uvjeta zaustavljanja algoritma još valja razjasniti neke operatore koji se ovdje koriste. Operatori će biti objašnjeni u nastavku.



Slika 3.1. Dijagram toka jedne petlje algoritma

3.3. Operatori

Kako bi se izvršila iteracija algoritma moraju se provesti sljedeće operacije nad jedinkama.

Troturnirska selekcija (*Slika 3.2.*) – U svakoj iteraciji postupka jedna jedinka populacije ustupa mjesto novoj jedinki. U algoritmu će jedinke s boljom dobrotom imati veće šanse biti izabrane. Funkcija dobrote zato mora odrediti kvalitetu jedinke. U navedenoj selekciji nasumično se biraju tri jedinke te se potom rangiraju po dobroti. Najlošija jedinka ustupa mjesto novoj jedinki čiji su roditelji bolje dvije jedinke.

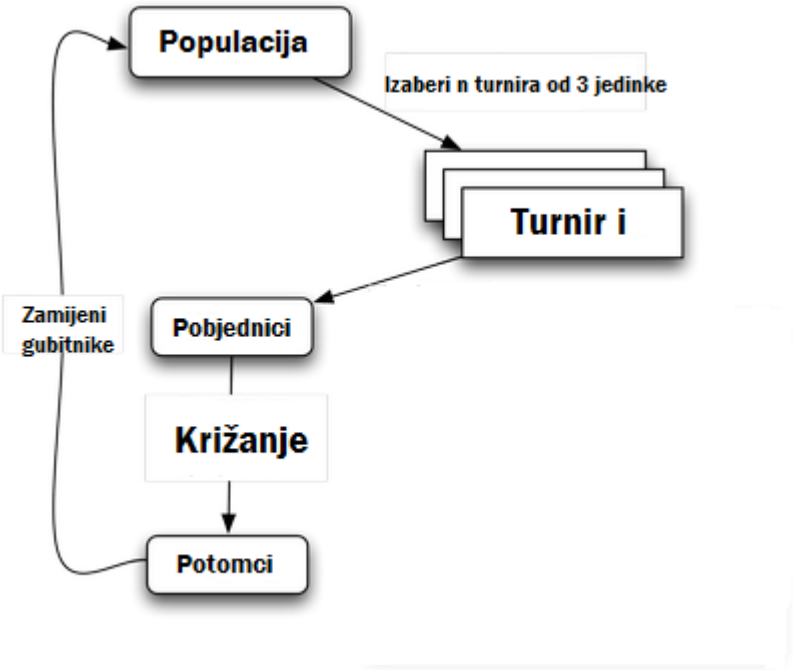
Na kraju selekcije vrši se križanje kojim se stvara genetski materijal nove jedinke koja zamjenjuje najlošiju, treću, jedinku.

Križanje (*Slika 3.3.*) – Operator kojim dvije (ili više) jedinki stvara novu jedinku kombinacijom svojih gena. Ovaj proces je analogan reprodukciji u prirodi. Time nastaje dijete koje se dodaje u populaciju. Ovaj postupak ponavlja se dok se ne dostigne zadana veličina nove populacije.

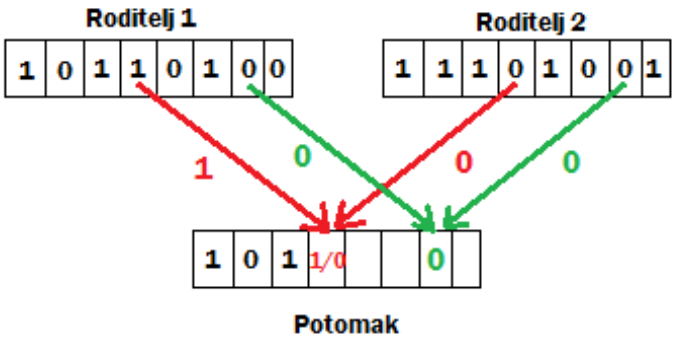
Mutacija (*Slika 3.4.*) – Operator koji potiče evoluciju tako što nasumično mijenja jedno ili više svojstava jedinki. Ovaj proces se javlja i u prirodi. Kao i u biologiji, vjerojatnost mutacije mora biti mala, zato što je njen učinak nepredvidljiv i može bitno promijeniti jedinku. Ako je vjerojatnost mutacije velika pretraživanje se pretvori u obično nasumično pretraživanje. Svrha mutacije je očuvanje i poticanje raznolikosti.

Ovi postupci obično će poboljšati prosječnu dobrotu populacije, zato što se iz prijašnje populacije uzimaju uglavnom bolje jedinke, a one lošije koje prežive zadržavaju raznolikost.

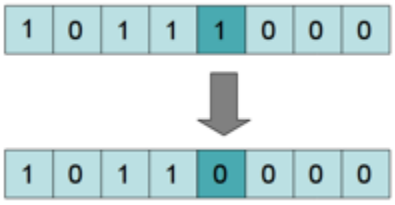
Valja još istaknuti da u programskom ostvarenju algoritma moraju biti zadani parametri: **vjerojatnost mutacije**, **vjerojatnost križanja** i **veličina populacije**.



Slika 3.2. Turnirska selekcija.



Slika 3.3. Križanje



Slika 3.4. Mutacija

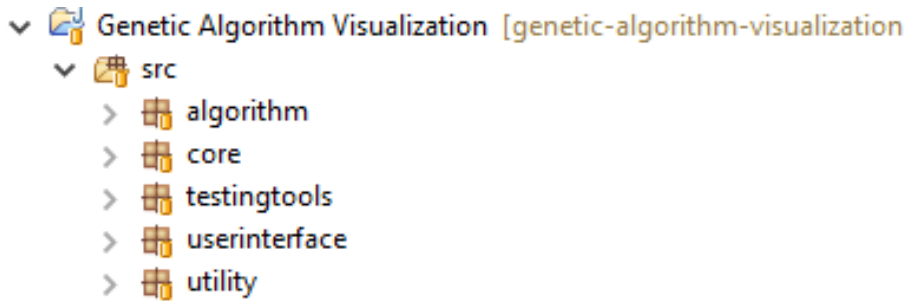
3.4. Uvjeti zaustavljanja

Proces se ponavlja dok nije zadovoljen jedan od sljedećih zadanih uvjeta:

- nađeno je rješenje koje zadovoljava minimalne uvjete
- dosegnut je određen broj generacija
- potrošen je "budžet" (vrijeme izvođenja / novac)
- dosegnuta je maksimalna moguća dobrota rješenja ili razinu koja više ne daje bolje rezultate
- ručno zaustavljanje
- kombinacija gore navedenih uvjeta

4. Programsko ostvarenje

Struktura projekta podijeljena je u nekoliko paketa kako bi se dobila veća preglednost projekta i njegovih komponenti (*Slika 4.1.)*.

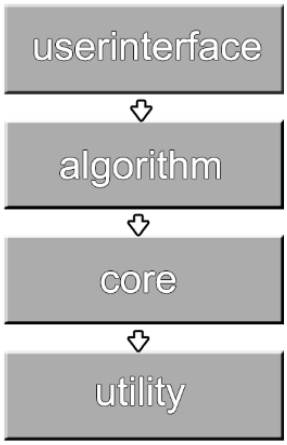


Slika 4.1. Paketi unutar projekta

Osnovni paketi su:

- algoritam (*algorithm*) – Paket unutar kojega se nalaze sva ostvarenja zadanih sučelja.
- jezgra (*core*) – Paket unutar kojeg se nalaze osnovni modeli razreda, to jest sučelja koja definiraju zahtjeve.
- alati za testiranje (*testingtools*) – Paket u kojem se nalaze razredi koji su bili potrebni za testiranje ispravnog rada aplikacije. Njegovi elementi nisu dio zadnje verzije aplikacije.
- korisničko sučelje (*userinterface*) – Elementi ovog paketa su razredi koji definiraju izgled korisničkog sučelja i sadrže glavni razred koji pokreće aplikaciju.
- uslužne komponente (*utility*) – Ovdje se nalaze razredi koje je potrebno definirati za ispravan rad programa, ali nisu dio samog algoritma. Na primjer definicija koordinatnog sustava, itd.

Dijagram (*Slika 4.2.*) prikazuje međuovisnosti paketa, to jest o kojim komponentama ovisi pojedini paket. Vidimo da svaka komponenta ovisi o onima ispod nje, zbog čega se razvoj ovako strukturiranog programa može početi odozdo i onda penjati prema gore kako završimo pojedinu komponentu i testiramo njenu ispravnost. Zbog toga će se ostatak poglavlja nastaviti u istom smjeru. Programsko ostvarenje će biti opisano u četiri koraka: uslužne komponente, jezgra, komponente algoritma i korisničko sučelje.



Slika 4.2. Međuovisnosti paketa

4.1. Uslužne komponente

Za lakšu impementaciju bilo je potrebno prvo definirati neki kostur, to jest sučelja koja bi postavila zahtjeve za svaku klasu.

4.1.1. Koordinate

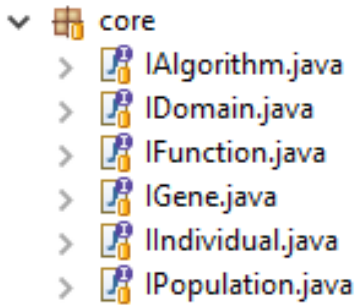
Razred Coordinate predstavlja koordinatu u dvodimenzionalnom koordinatnom sustavu. Ovo je potrebno za rad algoritma jer će jedinka populacije preko genetskog materijala biti smještena u domenu. Genetski materijal će biti zapis varijable *x* i *y, to jest pozicije na x osi i y osi.*

4.1.2. Funkcije

Sučelje IFunction služit će domeni kao zapis izgleda domene, to jest domena će prilikom inicijalizacije primiti objekt koji nasljeđuje sučelje. Objekt funkcije će predstavljati funkciju sa dvije varijable i mora ostvariti metode getValueAt koja će za dane varijable *x* i *y* vraćati vrijednost funkcije i metodu toString koja će vraćati zapis funkcije kao niz znakova kako bi je mogli ispisati na grafičko sučelje.

4.2. Jezgra

Kako bi se olakšalo ostvarenje algoritma u core paketu definirani su sučelja sa osnovnim zahtjevima koje razred mora ostvariti kako bi algoritam ispravno funkcionirao. Nazivima razreda prethodi slovo I koje označava da se radi o sučelju (*Slika 4.3.*).



Slika 4.3. Sadržaj paketa

4.2.1. Algoritam

Algoritam u sučelju IAlgorithm osim metoda za pokretanje i prisilno zaustavljanje, mora imati metodu preko koje će u korisničko sučelje dohvaćati informacije o jedinkama populacije za prikaz na zaslonu.

Metode sučelja IAlgorithm (*Slika 4.4.*):

- getLabels() (hrv. *dohvati oznake*) – Vraća korisničkom sučelju listu oznaka koje predstavljaju genetske materijale jedinki populacije.
- start – Pokreće algoritam
- stop – Prisilno zaustavlja algoritam prije ispunjenja zadanih uvjeta zaustavljanja algoritma.

```
7  public interface IAlgorithm extends Runnable {
8      public List<Label> getLabels();
9
10     public void start();
11
12     public void stop();
13 }
```

Slika 4.4. IAlgorithm sučelje

4.3.2. Domena

Za pravilno funkcioniranje algoritma trebalo je ostvariti diskretnu domenu funkcije sa dvije varijable. Također je trebalo biti moguće domenu iscrtati na zaslon. Budući da svaka jedinka sa svojim genetskim materijalom predstavlja poziciju u domeni treba biti moguće i dohvatiti vrijednost svake pojedine pozicije.

Metode sučelja IDomain (*Slika 4.5.*):

- getValueAt (hrv. *vrati vrijednost na*) – Vraća vrijednost određene točke u domeni.
- drawPopulation (hrv. *iscrtaj populaciju*) – Kao argument prima grafiku na koju iscrtava i populaciju koju iscrtava na grafiku.
- drawDomain (hrv. i*scrtaj domenu*) – Prima grafiku na koju se iscrtava domena

```
6  public interface IDomain {
7      public double getValueAt(Coordinate c);
8
9      public void drawPopulation(GraphicsContext gc, IPopulation p);
10
11     public void drawDomain(GraphicsContext gc);
12 }
13
```

Slika 4.5. IDomain

4.3.3. Gen

IGene je sučelje za razred genetskog materijala kojeg ima svaka jedinka. Budući da je genetski materijal niz bitova koji predstavlja x i y koordinate jedinke u domeni, razred će morati moći vratiti koordinatu koju predstavlja gen. Osim toga, kako bi se jedinke mogle prikazati na zaslonu genetski materijal mora se moći dohvatiti u obliku niza znakova. Radi lakše inicijalizacije gen bi se trebao moći sam postaviti na neku nasumičnu vrijednost, također treba moći vršiti nad sobom operacije genetskog algoritma kao što su mutacija i križanje.

Metode sučelja IGene (*Slika 4.6.*):

- getPosition (hrv. *dohvati poziciju*) – Vraća poziciju u obliku koordinate.
- getGeneticMaterial (hrv. *dohvati genetski materijal*) – Vraća genetski materijal u obliku niza znakova.
- setRandom (hrv. *postavi na nasumičnu vrijednost*) – Postavlja genetski materijal na nasumičnu vrijednost.
- Mutate (hrv. *mutacija*) – Mutira vrijednost genetskog materijala, tj. mijenja vrijednost nasumičnih bitova.

- crossover (hrv. *križanje*) – Prima informacije o genetskim materijalima oba roditelja, križa ih i postavlja se na dobivenu vrijednost.

```
5  public interface IGene {
6      public Coordinate getPosition();
7
8      public String getGeneticMaterial();
9
10     public void setRandom();
11
12     public void mutate();
13
14     public void crossover(IGene parent1, IGene parent2);
15 }
```

Slika 4.6. IGene

4.3.4. Jedinka

Razred koji predstavlja jedinku u populaciji je dio populacije i posjeduje svoj genetski materijal. Računa svoju dobrotu i upravlja svojom oznakom na zaslonu, to jest mijenja boje svog zapisa sukladno tome koja operacija se vrši nad njim. Sučelje IIndividual sadrži zahtjeve za sve potrebne operatore.

4.3.4. Populacija

Sučelje IPopulation definira zahtjeve razreda populacije koje ima informacije o svim jedinkama koje su dio te populacije. Za razred algoritma dohvaća nasumične pojedince kako bi algoritam mogao odabrati jedinke za turnirsku selekciju i mutaciju.

4.2. Algoritam

U ovom paketu nalaze se ostvarenja svih sučelja i glavna logika algoritma.

4.2.1. Algoritam

Razred algoritma (*Algorithm*) ostvarenje je sučelja IAlgorithm. Ovo je glavni razred ostvarenja algoritma koji upravlja izvršavanjem i sadrži informacije o populaciji, domeni i grafici na koju se iscrtavaju domena i informacije o populaciji.

Glavna komponenta ovog razreda je samo iteriranje algoritma, i budući da se razred može pokrenuti kao zasebna dretva, logika o iteriranju algoritma nalazi se u metodi *run()* (*Slika 4.7.*) , u nastavku će biti objašnjen kod prikazan na slici.

Genetski algoritam, koji je prikazan na dijagramu (*Slika 3.1*), prolazi kroz iteracije dok se ne ispuni uvjet zaustavljanja, zbog toga je kod unutar while petlje i vrti se dok se globalna zastavica stop ne postavi na true. Tu zastavu mogu postaviti ili korisnik preko sučelja, ili drugi dijelovi programa koji su zaduženi za provjeravanje je li ispunjen neki od uvjeta zaustavljanja.

Budući da je naglasak na vizualizaciji algoritma nakon svake operacije zove se metoda pause(), koja zaustavlja rad algoritma na neko kratko vrijeme ili dok korisnik ne odluči preći u sljedeći korak. Ovo se radi kako bi bilo lakše pratiti tok algoritma, vidjeti koje jedinke su prošle kroz mutaciju, i slično.

Budući da boja oznake jedinke označava koja operacija se nad njom izvršila, na početku svake iteracije te boje se resetiraju na zadanu boju, u ovom slučaju crnu. Nakon toga slijedi iscrtavanje populacije na zaslon.

Sljedeći korak je računanje dobrote jedinki u trenutnoj iteraciji, kako bi se mogle evaluirati u sljedećim koracima.

Algoritam preko razreda populacije dobiva tri nasumično odabrane jedinke koje zatim silazno sortira po njihovoj dobroti. Budući da se po ovakvom sortiranju najlošija jedinka nalazi na zadnjem mjestu, nad njom se vrši križanje, gdje su roditelji prve dvije jedinke u toj troturnirskoj selekciji. Genetski materijal zadnje jedinke se u potpunosti briše i mijenja križanjem genetskog materijala preostale dvije jedinke. Na ovaj način jedna jedinka u populaciji ispada i njeno mjesto popunjava nova.

Nakon troturnirske selekcije slijedi mutacija. Odabire se nasumična jedinka u populaciji nad kojom se vrši mutacija.

Iteriranje se nastavlja.

```
42    @Override
43    public void run() {
44        while (!stop) {
45            // draw
46            population.resetColors();
47            domain.drawPopulation(gc, population);
48            pause();
49
50            // calculate fitness
51            population.caluclatePopulationFitness(domain);
52
53            // tournament selection
54            List<IIndividual> randomIndividuals = population.getRandomIndividuals(3);
55            Collections.sort(randomIndividuals);
56
57            IIndividual individual1 = randomIndividuals.get(0);
58            IIndividual individual2 = randomIndividuals.get(1);
59            IIndividual individual3 = randomIndividuals.get(2);
60            individual3.crossover(individual1, individual2);
61            pause();
62
63            // mutation
64            for (IIndividual i : population.getRandomIndividuals(1))
65                i.mutate();
66            pause();
67        }
68        stop = false;
69    }
```

Slika 4.7. Iteracije algoritma

4.2.2. Domena

Razred domene (*Domain*) ostvaruje sučelje IDomain i sadrži podatke o veličini domene. Veličina domene će ovisiti o duljini genetskog materijala jedinke.

Kako bi algoritam lako mogao vizualizirati više različitih domena razred nema informacije o funkciji domene. Prilikom kreiranja predaje mu se razred koji nasljeđuje sučelje IFunction, preko kojeg razred domene može dohvaćati informacije o vrijednosti funkcije na pojedinom polju.

Domena je diskretna i koordinate su parovi prirodnih brojeva koji se dobiju prijevodom genetskog materijala.

Prije nego se počne vrtiti algoritam potrebno je na zaslon iscrtati izgled domene unutar zadanih granica. Za to se koristi metoda drawDomain (*Slika 4.8.*) koja kao argument prima grafiku na koju iscrtava domenu. Program se koristi kao prikaz rada genetskog algoritma, tako da je bitno znati izgled domene kako bi utvrdili kako algoritam radi. Na početku iscrtavanja domene pronalaze se minimalna i maksimalna vrijednost domene na području koji se koristi za vizualizaciju algoritma (linija 43 do 53). Minimalna i maksimalna vrijednost potrebne su nam kako bi lakše iscrtali domenu na zaslon.

Nakon utvrđivanja minimalne i maksimalne vrijednosti, iterira se kroz cijelu domenu i svaka točka domene poprima boju nijanse sive. Koju boju će točka poprimiti ovisi o njenoj vrijednosti. Točke domene koje su maksimalne vrijednost na zaslonu će biti najsvjetlije točke (vrijednost 255), dok će minimalna vrijednost biti najtamnija točka (vrijednost 0). Ostale će poprimiti nijanse sive po formuli u retku 57:

$$v = \frac{f(x,y) - min}{\max - min} * 255$$

gdje je vrijednost (*v*) prirodni broj između 0 i 255, i označava nijansu sive koja će se pojaviti na ekranu. *f(x, y)* je vrijednost dane funkcije za točku koju razmatramo, kojoj su koordinate x i y. *min* i *max* su minimalna i maksimalna vrijednost domene na danom rasponu.

```
41    @Override
42    public void drawDomain(GraphicsContext gc) {
43        double z = function.getValue(0, 0);
44        max = z; min = z;
45
46        for (int i = 1; i < domainSize; i++) {
47            for (int j = 1; j < domainSize; j++) {
48                z = function.getValue(i, j);
49
50                if (z > max) max = z;
51                if (z < min) min = z;
52            }
53        }
54
55        for (int i = 0; i < domainSize; i++) {
56            for (int j = 0; j < domainSize; j++) {
57                int value = Math.abs((int) Math.round((function.getValue(i, j) - min) / (max - min) * 255));
58                Color c = Color.rgb(value, value, value);
59                gc.setFill(c);
60                gc.fillRect(i * blockSize, j * blockSize, blockSize, blockSize);
61            }
62        }
63    }
```

Slika 4.8. Metoda iscrtavanja domene na zaslon

Domena također iscrtava danu populaciju na danu grafiku. Ova grafika bit će drugi sloj grafike na kojoj je iscrtana domena kako bi se lako kroz iteracije mogla brisati i ponovo na nju iscrtavati cijela domena bez da moramo ponovo iscrtavati cijelu domenu.

Metoda drawPopulation (*Slika 4.9.*) prima kao argument grafiku i populaciju, dohvaća svaku jedinku iz populacije i iscrtava ju na zaslon. Svaka jedinka bit će iste boje.

```
29    @Override
30    public void drawPopulation(GraphicsContext gc, IPopulation p) {
31        gc.clearRect(0, 0, domainSize * blockSize, domainSize * blockSize);
32
33        for (IIndividual i : p.getIndividuals()) {
34            Coordinate c = i.getPosition();
35
36            gc.setFill(Color.PALEVIOLETRED);
37            gc.fillRect(c.y * blockSize, c.x * blockSize, blockSize, blockSize);
38        }
39    }
```

Slika 4.9. Metoda za iscrtavanje populacije na domenu

Budući da algoritam rješava problem funkcije sa dvije varijable dobrotu jedinke najbolje je računati po toj funkciji, koju sprema razred domene. Tako da će domena imati metodu preko koje će algoritam računati dobrotu jedinke. Metoda getValueAt (*Slika 4.10.*) kao argument prima poziciju jedinke i vraća vrijednost funkcije na toj poziciji, što algoritmu ujedno i služi kao dobrota jedinke.

```
24    @Override
25    public double getValueAt(Coordinate c) {
26        return function.getValue(c.x, c.y);
27    }
```

Slika 4.10. Metoda koja vraća vrijednost funkcije na danoj točki

4.2.3. Gen

Razred gena (*Gene*) ostvaruje sučelje IGene i sadrži informacije o genetskom materijalu u obliku niza znakova. Genetski materijal je niz bitova, to jest nula i jedinica. Prilikom inicijalizacije populacije i jedinki, jedinka dobiva nasumično dobiven gen.

Metoda getPosition (*Slika 4.11.*) prevodi genetski materijal u poziciju u domeni i vraća koordinate. Genetski materijal sastoji se od *n* nula i jedinica. Broj bitova mora biti paran,

jer se genetski materijal prepolavlja i prva polovica označava varijablu *x*, a druga varijablu *y* koordinatnog susava. Te dvije varijable su vrijednosti dvije varijable u funkciji sa dvije varijable i smještaju gen, to jest jedinku koja nosi taj gen, na točku u domeni s nekom vrijednošću. Nizovi bitova varijable *x* i *y* su binarni zapis prirodnog broja. Tako metoda prepolavlja genetski materijal i prevodi binarni zapis u decimalni, te vraća koordinate s dobivenim vrijednostima.

```
16    @Override
17    public Coordinate getPosition() {
18        String xbin = (String) geneticMaterial.subSequence(0, bits / 2);
19        String ybin = (String) geneticMaterial.subSequence(bits / 2, bits);
20        int x = Integer.parseInt(xbin, 2);
21        int y = Integer.parseInt(ybin, 2);
22        return new Coordinate(x, y);
23    }
```

Slika 4.11. Metoda koja prevodi genetski materijal i vraća koordinate

Genetski materijal se prilikom inicijalizacije postavlja na nasumičnu vrijednost, što obavlja metoda setRandom (*Slika 4.12.*). Metoda za dužinu gentskog materijala na svakoj poziciji zapisuje nasumično 1 ili 0 sa vjerojatnošću od 50% za bilokoji bit.

```
30    @Override
31    public void setRandom() {
32        Random r = new Random();
33
34        geneticMaterial = "";
35        for (int i = 0; i < bits; i++) {
36            geneticMaterial += (Math.abs(r.nextInt()) % 2 == 1) ? "1" : "0";
37        }
38    }
```

Slika 4.12. Postavljanje genetskog materijala na nasumičnu vrijednost

Kako bi se nad pojedincem mogle vršiti operacije mutacije i križanja genetski materijal ostvaruje te metode.

Mutacija (*Slika 4.13.*) obavlja se tako što se prolazi svaki bit genetskog materijala i svaki bit se može invertirati u suprotnu vrijednost sa vjerojatnošću *p*. Vjerojatnost je određena kao $\frac{1}{l}$ gdje je *l* duljina genetskog materijala. Metoda prolazi cijeli genetski materijal i stvara novi. Generira nasumičan broj između 0 i *l* - *1*. Ako je generirani broj jednak 0, na novi genetski materijal se dodaje suprotan bit od trenutnog, u suprotnom se dodaje isti bit kao kod originalnog genetskog materijala na toj poziciji.

```java
    @Override
    public void mutate() {
        Random r = new Random();
        int l = geneticMaterial.length();
        String newGeneticMaterial = "";

        for (int i = 0; i < l; i++) {
            int leftover = r.nextInt() % l;
            String currentBit = "" + geneticMaterial.charAt(i);
            String oppositeBit = currentBit.equals("1") ? "0" : "1";

            if (leftover == 0)
                newGeneticMaterial += oppositeBit;
            else
                newGeneticMaterial += currentBit;
        }
        geneticMaterial = newGeneticMaterial;
    }
```

Slika 4.13. Programsko ostvarenje mutacije

Operacija križanja (*Slika 4.14.*) radi slično, samo se za bit koji se dodaje uzima ili bit na trenutnoj poziciji od jednog ili drugog roditelja. Za svaku poziciju unutar genetskog materijala generira se nasumičan broj. Ako je broj paran uzima se bit od jednog roditelja, ako je neparan uzima se od drugog roditelja. Ovime je vjerojatnost da će se uzeti bit od bilokojeg roditelja 50%.

```java
    @Override
    public void crossover(IGene parent1, IGene parent2) {
        Random r = new Random();
        String g1 = parent1.getGeneticMaterial();
        String g2 = parent2.getGeneticMaterial();

        geneticMaterial = "";
        for (int i = 0; i < bits; i++) {
            geneticMaterial += (Math.abs(r.nextInt()) % 2 == 1) ? g1.charAt(i) : g2.charAt(i);
        }
    }
```

Slika 4.14. Programsko ostvarenje križanja

4.2.4. Jedinka

Razred jedinke (*Individual*) ostvaruje sučenje IIndividual i sprema referencu na svoj gen (razred Gene, poglavlje 4.2.3.) i referencu na vizualnu oznaku koja se prikazuje na grafičkom sučelju. Oznaka se koristi kako bi u nju jedinka mogla zapisati svoj genetski materijal i

mijenjati boje oznake ovisno o operacijama koje su izvršene na njoj.

Jedinka većinu operacija koje se zovu nad njom delegira na svoj gen, na primjer križanje, mutaciju i dohvat koordinate jedinke. Kod računanja svoje dobrote jedinka primi domenu u kojoj se nalazi kao argument i preko nje dobiva svoju dobrotu (*Slika 4.15.*).

```
42      @Override
43      public void calculateFitness(IDomain domain) {
44          fitness = domain.getValueAt(gene.getPosition());
45      }
```

Slika 4.15. Računanje dobrote jedinke

Jedinka također ostvaruje sučelje *Comparable* kako bi se liste jedinki mogle sortirati po njihovim dobrotama (*Slika 4.16.*). Metoda dobiva jedinku s kojom uspoređuje trenutnu. Ako dobivena jedinka ima veću dobrotu od trenutne, metoda vraća pozitivan broj, u suprotnom negativan. Ako im je dobrota jednaka, funkcija vraća 0.

```
85      @Override
86      public int compareTo(IIndividual arg0) {
87          if (fitness < arg0.getFitness())
88              return 1;
89          if (fitness == arg0.getFitness())
90              return 0;
91          else
92              return -1;
93      }
```

Slika 4.16. Metoda usporedbe jedinki

4.2.5. Populacija

Razred populacije (*Population*) ostvaruje sučelje IPopulation i sprema referencu na sve jedinke populacije. Razred koristi razred Algoritma kako bi lako mogao vršiti operacije nad populacijom. Prilikom inicijalizacije populacija prima broj jedinki i zadanu duljinu genetskog materijala nakon čega se te jedinke generiraju sa nasumičnim genetskim materijalom (*Slika 4.17.*).

```
18    public Population(int size, int geneticMaterialSize) {
19        individuals = new ArrayList<IIndividual>();
20        this.size = size;
21
22        for (int i = 0; i < size; i++)
23            individuals.add(new Individual(geneticMaterialSize));
24    }
```

Slika 4.17. Inicijalizacija populacije

Populacija također može algoritmu vratiti listu nasumično odabranih jedinki (*Slika 4.18.*). Kao argument metode se predaje broj jedinki za nasumični odabir, tako da algoritam može zatražiti na primjer tri jedinke za troturnirsku selekciju, ili jednu za mutaciju. Metoda koristi pomoćnu listu u koju sprema jedinke koje će vratiti na kraju metode i listu indexa kako bi znala koje jedinke je već izabrala. Nakon toga generira slučajan broj u rasponu veličine populacije i stavlja ga u listu izabranih jedinki ako već nije izabrana.

```
50    @Override
51    public List<IIndividual> getRandomIndividuals(int n) {
52        Random r = new Random();
53        List<IIndividual> res = new ArrayList<IIndividual>();
54        Set<Integer> used = new HashSet<Integer>();
55
56        while (used.size() < n) {
57            int index = Math.abs(r.nextInt()) % size;
58
59            if (used.contains(index))
60                continue;
61
62            used.add(index);
63            res.add(individuals.get(index));
64        }
65        return res;
66    }
```

Slika 4.18. Metoda za nasumični odabir n jedinki populacije

4.3. Korisničko sučelje

Paket korisničkog sučelja sadrži klase koje pokreću program, nude korisniku da definira postavke algoritma i služe kao podloga na koju algoritam iscrtava domenu i populaciju tokom iteracija.

4.3.1. Glavni prozor

Razred MainFrame je razred koji se prvi pokreće. Instancira plohu na koju će se iscrtavati algoritam, zauzima mjesto na kojem će se ispisivati genetski materijali jedinki i nudi korisniku da definira postavke algoritma prije nego se algoritam pokrene.

4.3.2. Ploha za iscrtavanje

Razred SimulationPane nasljeđuje razred Pane i generira grafiku na koju algoritam iscrtava svoje komponente. Kako bi iscrtavanje bilo lakse, ploha generira dvije grafike, jedna na koju će se iscrtati domena prilikom pokretanja algoritma i jednu na koju će se iscrtavati jedinke kroz iteracije. Ovaj razred instancira domenu s postavkama koje je korisnik definirao, iscrtava domenu te kreira razred algoritma i predaje mu referencu na domenu. Nakon toga pokreće algoritam.

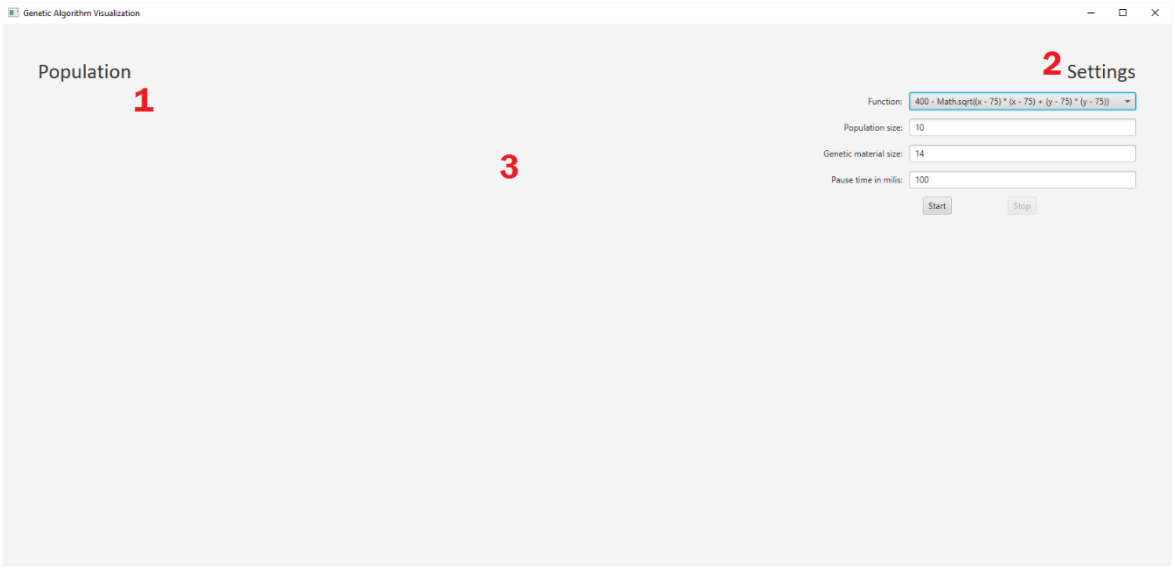
5. Ispitivanje programskog sustava

Ovo poglavlje proći će izgled korisničkog sučelja i demonstrirati rad algoritma.

5.1. Pokretanje programskog rješenja

Na Slici 5.1. prikazan je izgled prozora prilikom pokretanja programa. Brojevima su označeni bitniji dijelovi:

1. Stupac u kojemu će biti ispisani genetski materijali i dobrota jedinki.
2. Postavke algoritma koje korisnik može definirati.
3. Prostor



Slika 5.1. Pokretanje programa

Postavke na koje korisnik može utjecati su:

- Funkcija – Pritiskom na padajući izbornik korisnik može odabratu neku od prethodno definiranih funkcija s dvije varijable za koju želi isprobati rad algoritma.
- Populacija – Broj jedinki koju će populacija imati, za ovaj algoritam broj jedinki u populaciji je konstantan kroz sve iteracije.
- Duljina genetskog materijala – Postavka kojom korisnik definira dužinu genetskog materijala jedinke. Što je genetski materijal duži to će više informacija moći spremiti čime će varijable koje predstavlja moći poprimiti veću vrijednost. To će rezultirati većim rasponom domene, što će algoritam uzeti u obzir.

- Vrijeme pauze između operacija – Između svake operacije unutar iteracije iscrtavanje se pauzira na dano vrijeme. Korisnik može produžiti pauzu kako bi povećao preglednost algoritma ili je smanjiti kako bi brže vidio rezultat algoritma.

Osim definiranja postavki korisnik pokreće algoritam pritiskom na gumb Start.

5.2. Pokretanje algoritma

Nakon što se algoritam pokrene iscrta se domena. Domena predstavlja zadanu funkciju sa dvije varijable. Nijansa sive opisuje vrijednost funkcije na danoj točki.

Mijenjanje postavki algoritma onemogućeno je dok algoritam radi. Algoritam se može samo zaustaviti pritiskom na gumb Stop.

Tekstualna reprezentacija populacije (Slika *5.2*., točka 1) je niz genetskih materijala jedinki i njihove dobrote. Niz bitova predstavlja genetski materijal kako je zapisan u genu, a broj koji ga slijedi je njegova dobrota.

Osim pretpostavljene, crne boje, oznake mogu poprimiti i zelenu, crvenu i plavu boju.

Po iteraciji se izvodi jedna troturnirska selekcija koja izabire tri jedinke. Dvije jedinke koje su bolje od treće poprimaju zelenu boju, a zadnja, treća, jedinka poprima crvenu boju.

Plavu boju oznake poprima jedinka koja je tu iteraciju izabrana za mutaciju.

Osim što im oznake promjene boju, jedinki koja je mutirala i najlošijoj jedinki troturnirske selekcije mijenja se boja i na domeni (Slika *5.*2., točka 2)p. Mutirana jedinka postaje plava, a najlošija jedinka troturnirske selekcije postaje crvena. Razlog tomu je što će u sljedećoj iteraciji te dvije jedinke promijeniti lokaciju na domeni, zato što su im se promijenili genetski materijali, povlačeći time i premještaj na novu lokaciju u domeni i novu dobrotu.

Slika 5.2. Izvođenje algoritma

5.3. Ponašanje algoritma

Ponašanje algoritma za funkciju:

$$f(x,y) = 400 - \sqrt{(x-75)^2 + (y-75)^2}$$

U prvoj iteraciji (*Slika 5.3.*) jedinke su nasumično raspoređene po domeni.

Slika 5.3. Inicijalna raspodjela

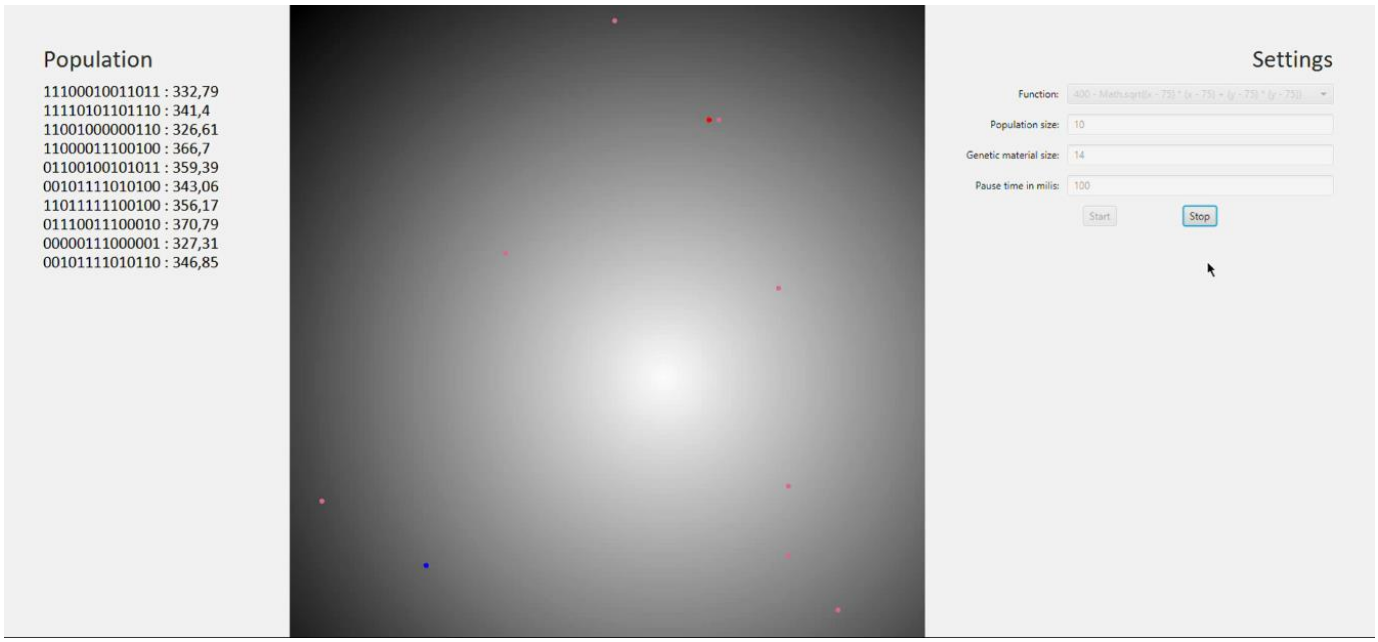
Odabiru se tri jedinke za troturnirsku selekciju (*Slika 5.4.*) i crveno je označena najlošija jedinka koja je sada poprimila novi genetski materijal.

Slika 5.4. Troturnirska selekcija prve iteracije

Nasumično se odabire i jedinka za mutaciju te je plavo označena (*Slika 5.5.*).

Slika 5.5. Mutacija prve iteracije

Ažurira se stanje na domeni i plavo i crveno se označavaju jedinke koje su promijenjene operacijama mutacije i križanja (*Slika 5.6.*).

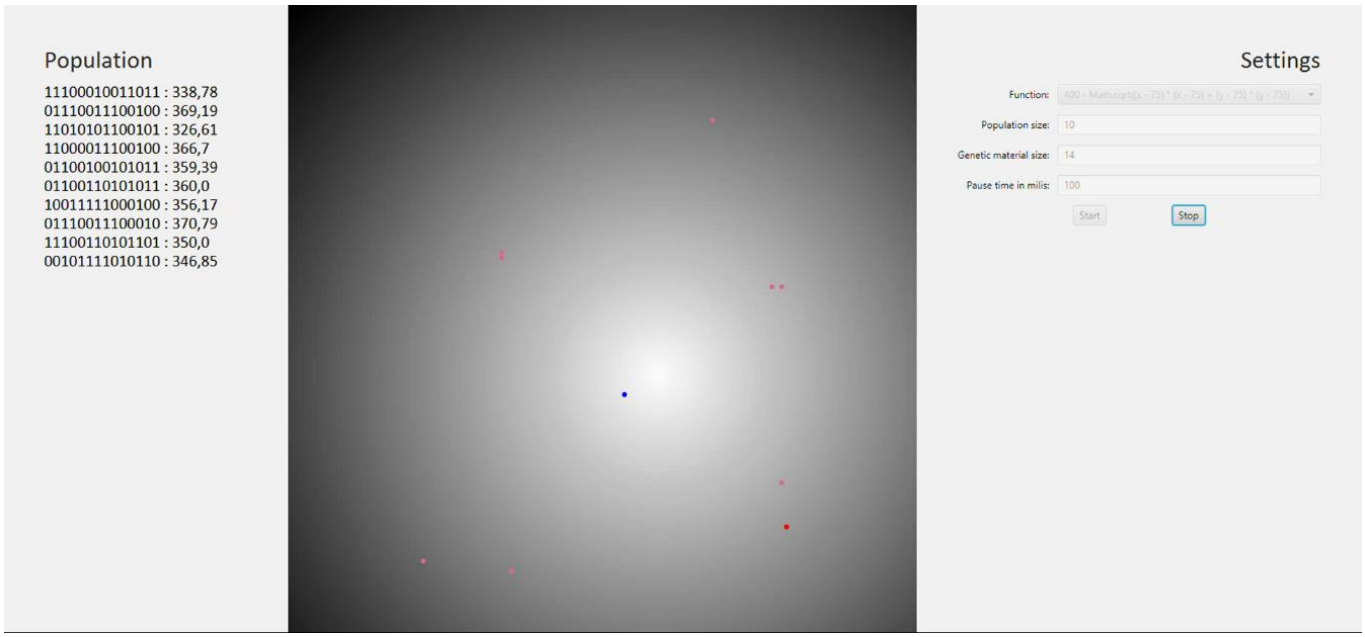


Slika 5.6. Ažuriranje domene prije ulaska u drugu iteraciju

Ulaskom u drugu iteraciju ažuriraju se dobrote jedinki i izabiru se nove tri jedinke za troturnirsku selekciju (*Slika 5.7.*).

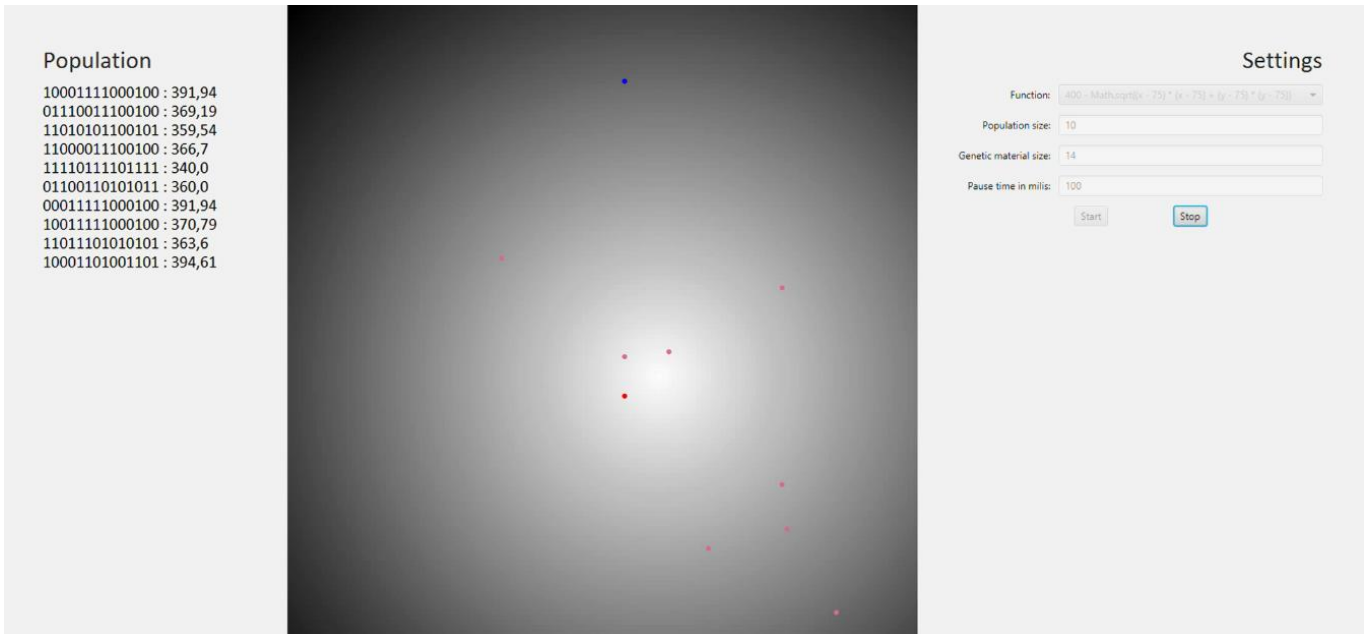
Slika 5.7. Ulazak u drugu iteraciju

Postupak iteriranja se ponavlja i nakon pet iteracija stanje domene izgleda kao na Slici *5.*8.



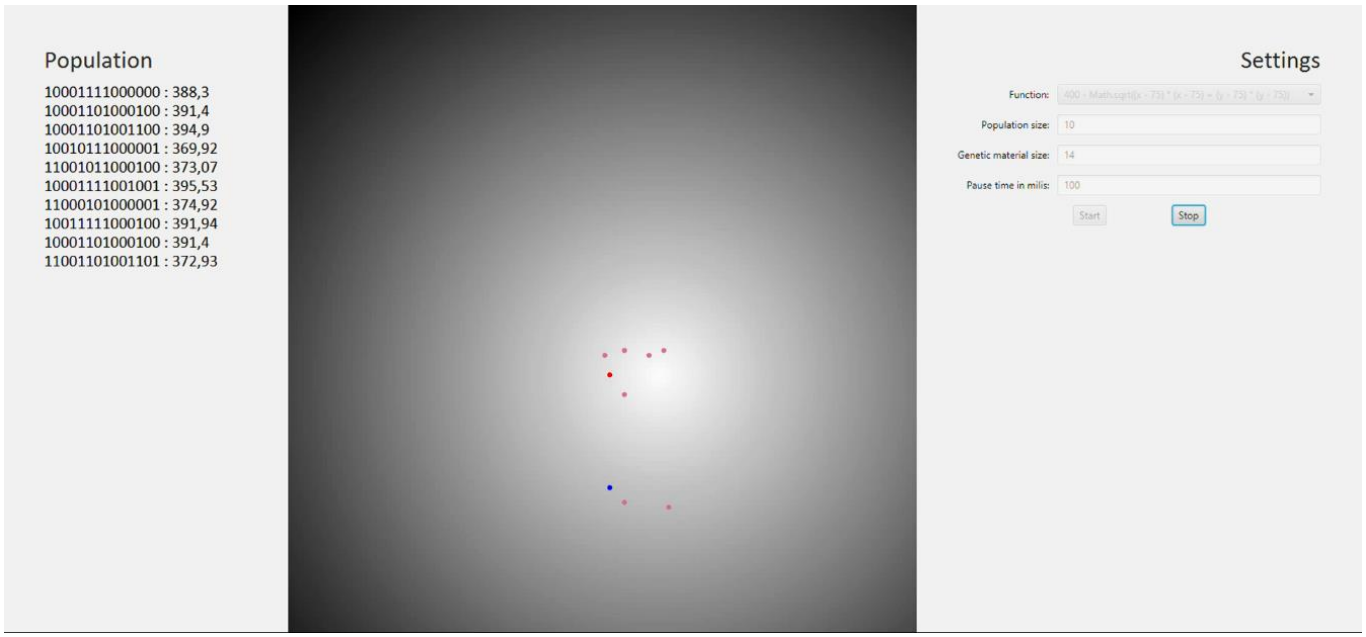
Slika 5.8. Domena nakon pet iteracija

Nakon deset iteracija (*Slika 5.9.*) jedinke počinju konvergirati prema najvrijednijoj točki domene (najsvjetlijoj).



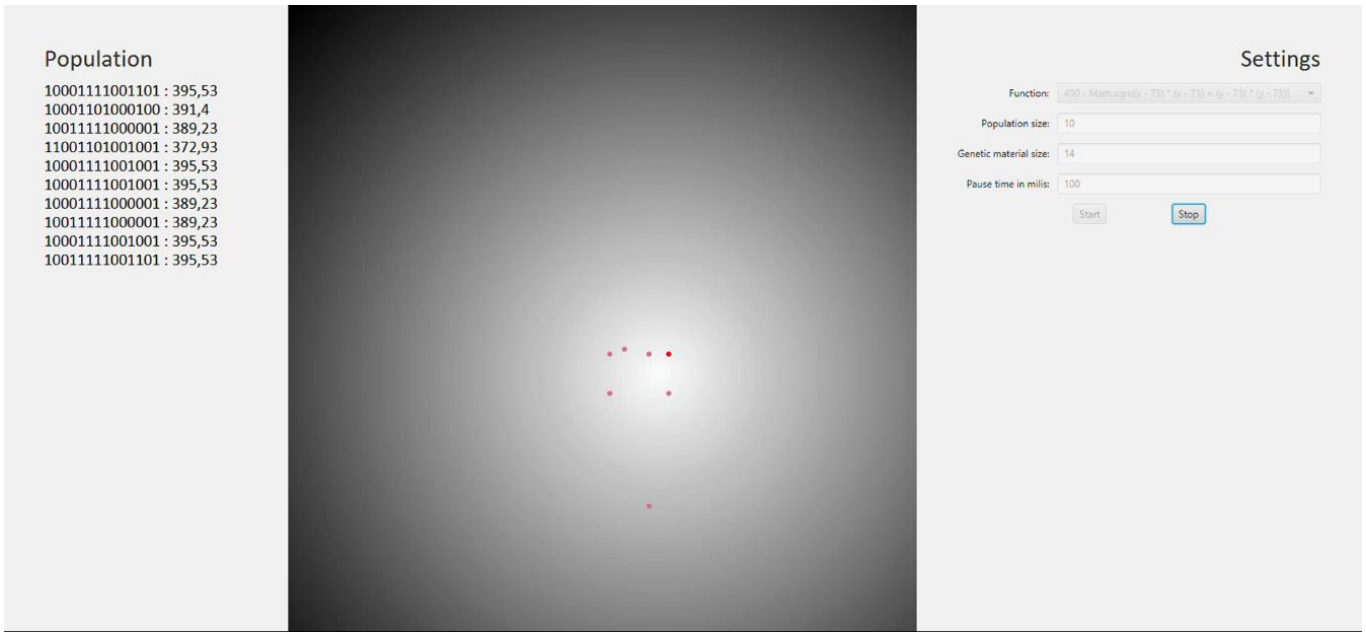
Slika 5.9. Domena nakon deset iteracija

Nakon dvadeset iteracija (*Slika 5.10.*) cijela populacija se nalazi u najboljih 20% domene.



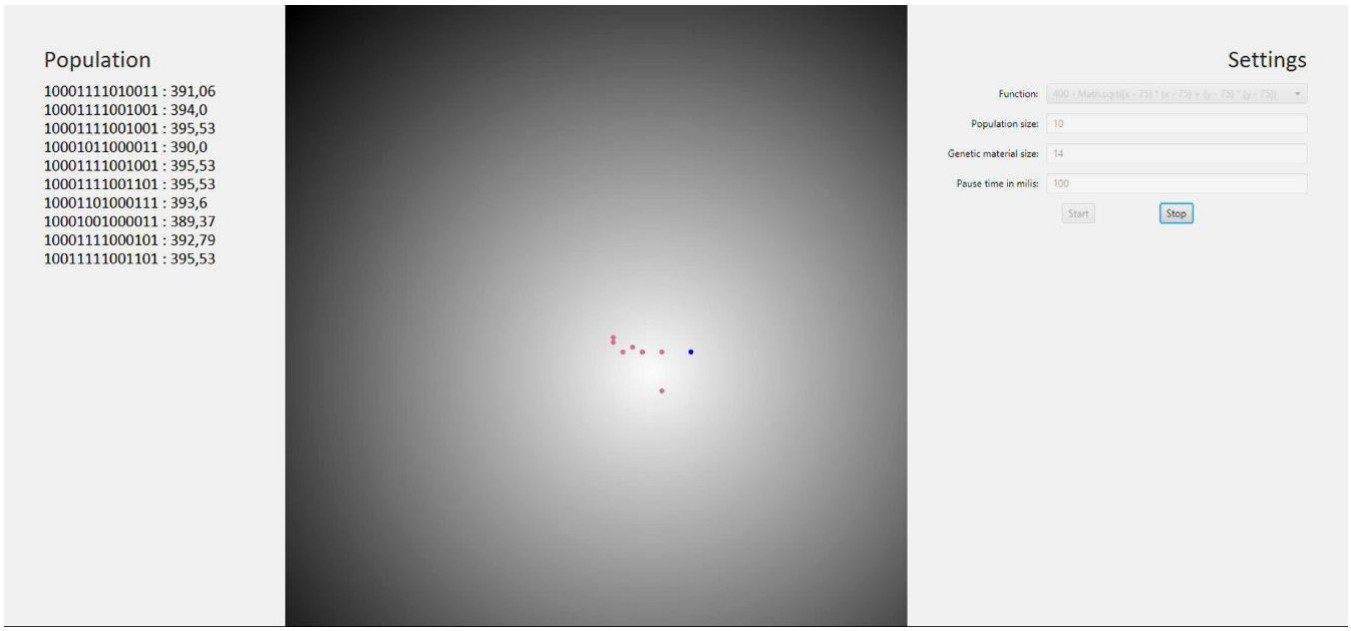
Slika 5.10. Domena nakon dvadeset iteracija

Nakon 40 iteracija (*Slika 5.11.*) napredak je sve sporije vidljiv jer jedinke zbog mutacije imaju veće šanse udaljiti se od rješenja nego mu prići bliže.



Slika 5.11. Domena nakon 40 iteracija

Nakon 100 iteracija (*Slika 5.12.*) najbolja jedinka u populaciji postiže rezultat 395,53 od maksimalnih 400.



Slika 5.12. Domena nakon 100 iteracija

6. Zaključak

Genetski algoritmi dobar su alat za pronalazak dobrih rješenja u prostoru koji je prevelik da bi ga cijelog iterirali. Time što su inspirirani funkcioniranjem stvarnog svijeta, zbog složenosti tog sustava, imamo puno verzija i preinaka koje možemo raditi na algoritmu. Analiziranje podataka rada algoritma može biti lakše ako ga vizualiziramo, to nam jer lakše možemo potvrditi koje promjene na algoritmu bi na nekoj vrsti problema mogle dati bolje rješenje. Na primjer, često skupljanje populacije u nekom lokalnom optimumu na domeni će biti prikazano sa puno agenata u točki domene koja je svjetlija od okolnih u nekom malom radijusu.

Programsko ostvarenje troturnirske selekcije prikazuje tok izvođenja algoritma na dva načina. S lijeve strane prikazuje se popis jedinki, a na sredini je slika domene funkcije s dvije varijable na kojoj su označene pozicije jedinki definirane njihovim genetskim materijalima. Boje oznaka jedinki i njihovih pozicija na domeni mijenja se ovisno o operacijama kroz koje prolaze. Na desnoj strani programskog rješenja nalaze se postavke kojima korisnik može utjecati na brzinu izvođenja algoritma, broj jedinki i dužinu genetskog materijala, čime povećava raspon domene koji se koristi za pronalazak rješenja.

LITERATURA

[1] Eisuke Kita. *Evolutionary Algorithms*. InTech, 2011.

[2] Witold Kosinski. *Advances in Evolutionary Algorithms*. InTech, 2008.

[3] Marin Golub. *Genetski algoritam, skripta – 1. dio*, 1997. URL http://www.zemris.fer.hr/~golub/ga/ga_skripta1.pdf

[4] Marin Golub. *Genetski algoritam, skripta – 2. dio*, 2004. URL http://www.zemris.fer.hr/~golub/ga/ga_skripta2.pdf

[5] Marko Čupić. *Prirodom inspirirani optimizacijski algoritmi. Metaheuristike.*, 2013

[6] Jason Brownlee. *Clever Algorithms*, 2012

[7] Domagoj Kusalić. *Evolucijski algoritmi insiprirani ljudskim psihosocijalnim ponašanjem*, 2010.

[8] Wikipedia. *Tournament Selection*. URL https://en.wikipedia.org/wiki/Tournament_selection

[9] Wikipedia. *Genetic algorithm*. URL http://en.wikipedia.org/wiki/Genetic_algori

Vizualizacija ponašanja populacije jedinki prilikom procesa optimizacije genetskim algoritmom

Sažetak

Genetski algoritmi su heuristički algoritmi za pretraživanje prostora rješenja. Dobri su kod problema u kojima nije poznat izgled domene. Troturnirska selekcija je mtoda selekcije koja prolazi proces odabira jedinki koje će biti križane i jedinke koja će ustupiti svoje mjesto novodobivenoj jedinki. Za ovaj rad razvijeno je programsko rješenje koje ostvaruje dani algoritam i prikazuje ponašanje jedinki kroz iteracije. Osim prikaza operacija jedinki prikazuje se i njihova pozicija u domeni rješenja. Rješava se problem funkcije sa dvije varijable, na čijoj domeni se smještaju jedinke. Programsko rješenje ostvareno je u jeziku Java.

Ključne riječi: genetski algoritam, troturnirska selekcija, vizualizacija, funkcija dvije varijable

Visualization of individuals behavior during optimization process by using genetic algorithm

Abstract

Genetic algorithms are heuristic algorithms for best solution searching. They exceed at problems at which shape of domain is not known. Tournament selection is a method of selecting individuals. Through a tournament it finds parents for new individual and weak individual whose place in the population will be replaced by new individual generated by crossover. This aglorithm was implemneted in programming language Java for purposes of this paper. Problem solved is function of two variables. Population individuals are set on domain of given function.

Keywords: genetic algorithm, tournament selection, vizualization, function of two variables