SVEUČILIŠTE U ZAGREBU

**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

**SEMINAR**

# GENETSKI ALGORITMI

*Dorija Humski*

Voditelj: *dr. sc. Marko Čupić*

Zagreb, ožujak, 2012.

# Sadržaj

# 1. Uvod

Prilagodba u živom svijetu nije ništa novo. Živi svijet reagira na okolinu, prilagođava se i opstaje. Možemo li napraviti program koji simulira živi svijet i bira samo najjače?

Razradu te ideje putem genetskih algoritama, John Holland[2] iznosi u svojoj knjizi „Adaptation in Natural and Artificial System“[1]. Genetski algoritam imitira prirodni evolucijski proces, selekciju, križanje i mutaciju. Primjenjuje se na populaciju u kojoj svaka jedinka predstavlja potencijalno rješenje problema koji se obrađuje. Selekcijom se odabiru najsposobnije jedinke koje postanu *roditelji* sljedećoj populaciju, dok se križanjem *roditelja* stvaraju nove inačice jedinki koje čine sljedeću populaciju. Novonastale jedinke mogu biti podvrgnute mutaciji. Evolucijski proces se ponavlja sve dok se ne zadovolji uvjet završetka. Na kraju dobivamo najbolje rješenje problema koje je mogao naći.

Detaljniji rad genetskih algoritama opisan je u sljedećem poglavlju, dok je primjena istog prikazana u trećem poglavlju.

1- Prilagodba u prirodnim i umjetnim sustavima (1975. god)
2- John Hollan (1929)-profesor računarskih znanosti poznat kao otac genetskih algoritama

## 2. Genetski algoritam

Genetski algoritam sastoji se od sljedećih koraka [1].

1. Generira se početna generacija $n$ potencijalnih rješenja.
2. Izračuna se dobrota svake jedinke u populaciji.
3. Izabire se određeni broj jedinki koji će biti *roditelji* sljedećoj generaciji.
4. Uzimamo parove izabranih *roditelja.* Svaki par križanjem stvara novu jedniku, s određenom vjerojatnošću mutacija. Novonastala jednika ulazi u novu populaciju.

   Proces se nastavlja sve dok se u novoj populaciji ne stvori $n$ jedinki.
5. Provjerava se uvjet završetka, ako nije ispunjen ponovljaju se koraci 2-5.

Kada se uvjet završetka ispuni i program izlazi iz petlje, iz populacije se uzima jedinka s najvećom dobrotom te ona predstavlja rješenje problema.

### 2.1. Početna populacija

Genetski algoritam počinje generiranjem početne populacije na kojoj se primjenjuje. Početna populacija može biti generirana slučajnim odabirom vrijednosti iz domene ili unosom rješenja nekog drugog zadanog algoritma. Svaka jedinka unutar populacije prikazuje se jednakom podatkovnom strukturom (niz bitova, broj, niz simbola, stablo…).

### 2.2. Selekcija

Selekcija se vrši vjerojatnosno, ali prema vrijednosti dobrote. Svakoj jedinki pridružena je odgovarajuća dobrota. Jedinke s većom dobrotom imaju veću mogućnost opstanka. Dobrotu određuje funkcija dobrote koja ovisi o problemu koji rješavamo.

Postoje različite vrste selekcija. Neke od često korištenih su: jednostavna, turnirska i eliminacijska [5]. Svaka od njih je u nastavku samo ukratko opisana.

- Kod *jednostavne selekcije* vjerojatnost odabira neke jedinke proporcionalna je njenoj dobroti.
- *Turnirska selekcija* simulira turnire u kojima se pobjednici stavljaju u skup *roditelja*.
- *Eliminacijska selekcija* svakoj jedinki pridružuje kaznu koja je jednaka razlici maksimalne dobrote i dobrote jedinke. Izbacuju se one s najvećom kaznom. Izbacivanjem jedne jedinke, dolazi druga koja je nastala genetskim operatorima dvije nasumično odabrane jedinke iz populacije.

Često korišten mehanizam prilikom selekcija je elitizam. Elitizmom se štite najbolje jedinke od izmjena ili eliminacija. Najbolje jedinke prenose se u sljedeću generaciju, dok se ostale jedinke nove populacije stvaraju križanjem između odabranih roditelja i mutacijom.

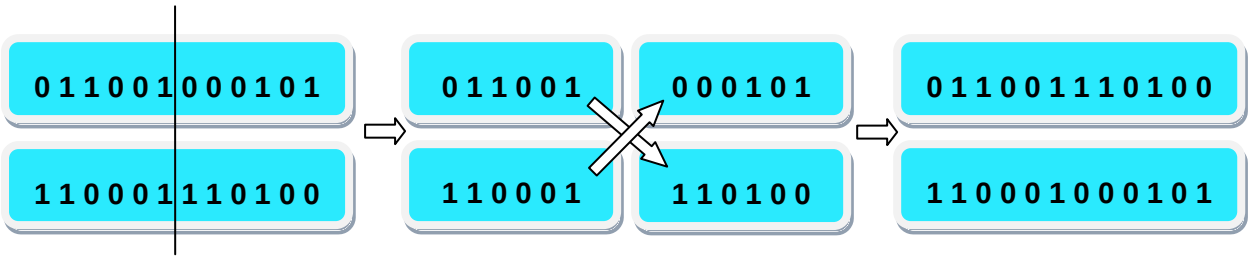
## 2.3. Genetski operatori

Genetski operatori su križanje i mutacija. U koraku 3. genetskog algoritma na parovima *roditelja* primjenjujemo križanje. Na novonastaloj jedinki vršimo mutaciju. Svrha genetskih operatora jest stvaranje novih jedinki visoke dobrote i time osigurati napredak populacije.

### 2.3.1. Operator križanja

U križanju sudjeluju dvije izabrane jedinke, *roditelji.* Križanjem mogu nastati jedna ili dvije jedinke.

Križanje može biti definirano s proizvoljnim brojem prekidnih točaka [2]. Najjednostavnije je križanje sa samo jednom prekidnom točkom (slika 2.1).

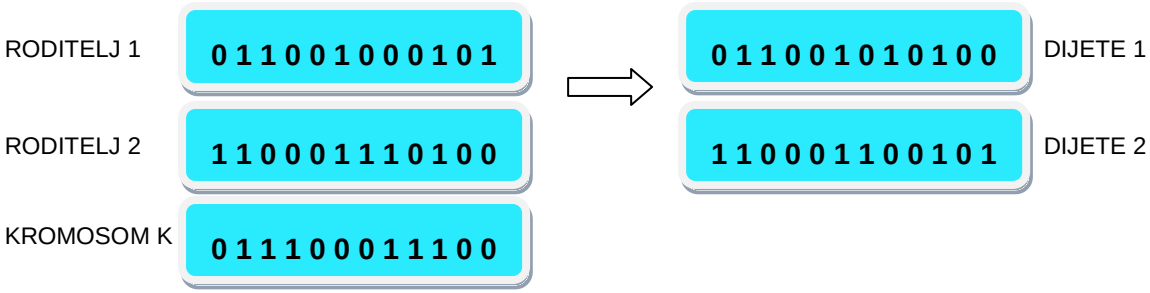



Slika 2.1 Križanje sa jednom točkom prekida

Križanje s $b$-1 točaka prekida nazivamo **uniformno križanje**. Vjerojatnost da će dijete naslijediti svojstvo roditelja je 50%. Uniformno križanje može se realizirati pomoću formula 2.1 i 2.2 u kojima $A$ i $B$ predstavljaju roditelje, a $K$ slučajno odabrani kromosom. Uniformno križanje prikazano je na slici 2.2.

$$DIJETE1 = AB + KA + KB \quad (2.1)$$

$$DIJETE2 = AB + \neg KA + \neg KB \quad (2.2)$$




Slika 2.2 Uniformno križanje

### 2.3.2. Operator mutacije

Nakon križanja nastaje jedinka koja se podvrgava mutaciji. Mutacija je slučajna promjena jednog ili više gena. Vjerojatnost da će neki kromosom mutirati definira se u algoritmu.

Različite vrste mutacija navedene su u nastavku [2].

1. Kod *jednostavne mutacije* vjerojatnost mutacije za sve kromosome je jednaka.
2. *Miješajuća mutacija* odabire kromosom za mutaciju, prvu i drugu granicu (uzorak) unutar koje ili izmiješa gene ili ih slučajno generira.
3. *Invertirajuća mutacija* odabire prvu i drugu granicu i unutar nje invertira sve gene.

## 2.4. Uvjet završetka

Ispunjenjem uvjeta završetka, genetski algoritam završava s radom i dobivamo rješenje.

Mogući uvjeti završetka su:

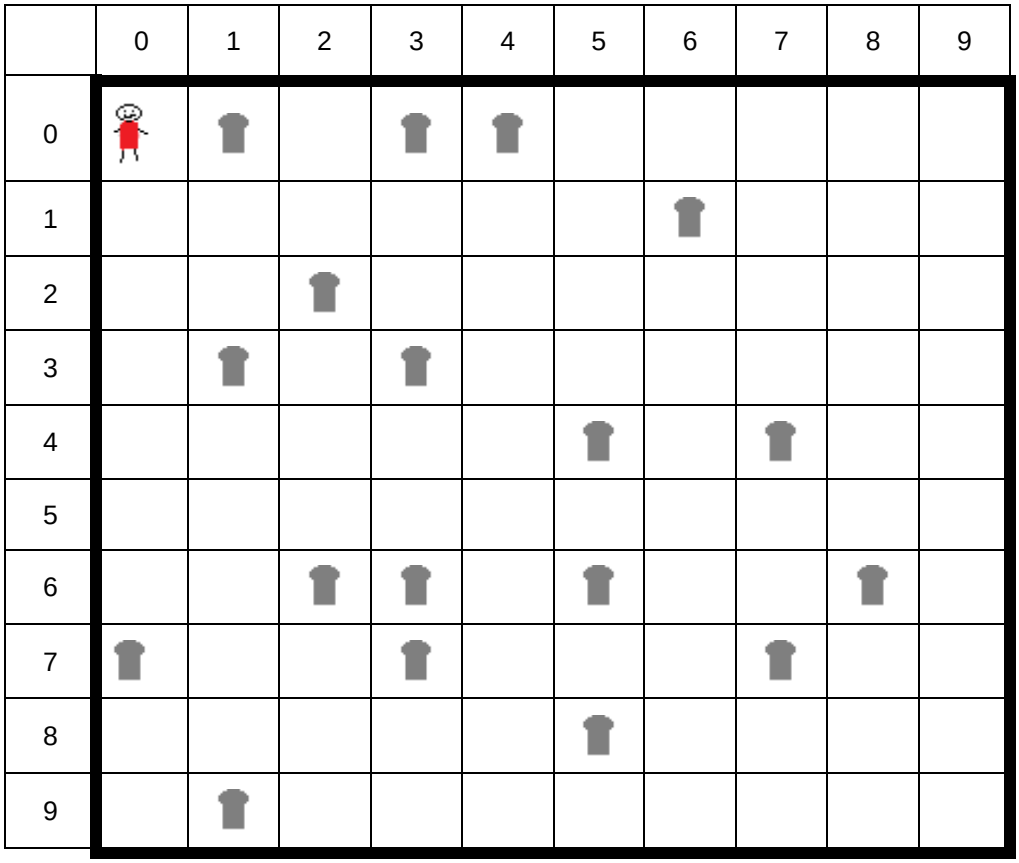
- vremensko ograničenje,
- broj iteracija,
- broj iteracija bez poboljšanja ,
- dostignuta vrijednost funkcije dobrote te drugi.

# 3. Primjer implementacije genetskog algoritma

Genetski algoritam ćemo ilustrirati na problemu robota Robby.

## 3.1. Opis problema

Robot Robby živi u dvodimenzionalnom svijetu 10x10 u kojem su razbacane limenke (slika 3.1). Robbyjev glavni zadatak je pokupiti sve limenke i tako počistit svijet.




Slika 3.1 Robbyjev svijet

Njegov svijet ograđen je zidovima koji su na slici 3.1 označeni debelom linijom. Robby s trenutne pozicije vidi sadržaj te pozicije i sadržaj mjesta koja graniče s njim na sjeveru, jugu, istoku i zapadu. Mjesto može biti prazno, na njemu može stajati limenka ili može biti zid. Robby nema mogućnost pamćenja.

U jednoj iteraciji Robby može napraviti 200 poteza. Za svaki potez ima sedam mogućnosti: pomaknuti se prema sjeveru, jugu, istoku, zapadu, pomaknuti se na slučajno odabrano mjesto, pokupiti limenku ili ostati na mjestu. Ako pokupi limenku dobiva deset bodova, ako želi pokupiti limenku tamo gdje je nema, oduzima mu se jedan bod. Ako se zabije u zid, kažnjava se s pet bodova i vraća na prijašnju poziciju.

## 3.2. Primjena genetskog algoritma

Robby gleda na pet različitih mjesta koja mogu biti u tri stanja (prazna, s limenkom, zid). Prema tome može se naći u 243 različite situacije. Za razvoj strategije potrebno je svakoj situaciji pridružiti potez koji će Robby napraviti. Primjer strategije naveden je u tablici 3.1. Strategije su jedinke u populaciji genetskog algoritma.

Tablica 3.1 Strategije

| **Situacija** | | | | | **Potez** |
|---|---|---|---|---|---|
| Sjever | Jug | Istok | Zapad | Trenutna pozicija | |
| Prazno | Prazno | Prazno | Prazno | Prazno | Sjeverno |
| Prazno | Prazno | Prazno | Prazno | Limenka | Istočno |
| Prazno | Prazno | Prazno | Prazno | Zid | Slučajan odabira |
| Prazno | Prazno | Prazno | Limenka | Prazno | Pokupi limenku |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| Zid | Zid | Zid | Zid | Zid | Ostani na mjestu |

Početna populacija sastoji se od 200 slučajno izabranih strategija. Jedinke su prikazane pomoću niza od 243 broja. Brojevi mogu poprimati vrijednosti od 0 do 6 i predstavljaju:

0=pomak sjeverno,

1=pomak južno,

2=pomak istočno,

3=pomak zapadno,

4=ostani na mjestu,

5=pokupi limenku te

6=izaberi 0-5.

Funkcija dobrote za pojedino stanje računa se prema 3.1, gdje $L$ predstavlja broj pokupljenih limenka, $Z$ broj zabijanja u zid, a $C$ pokušaj uzimanja limenke tamo gdje je nema. Dobrota je prosječna vrijednost funkcija dobrote za 100 različitih stanja svijeta. Svaki put Robby počne na poziciji (0,0) i u 200 pomaka pokuplja limenke koje su u različitim stanjima svijeta na različitim mjestima.

$$F(L,Z,C)=10L-5Z-1C \qquad (3.1)$$

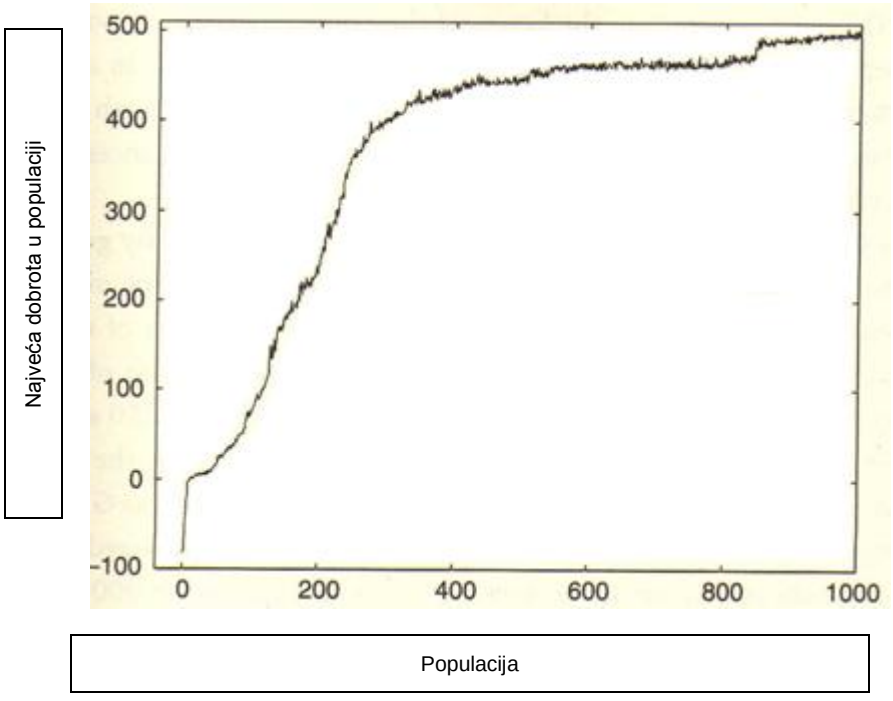
Odabir roditelja sljedećoj populaciji bira se pomoću jednostavne selekcije uz elitizam. Provodi se križanje s jednom točkom prekida i stvaraju se dvije nove jedinke. S malom vjerojatnošću, nad jedinkama provodi se jednostavna mutacija. Jednostavna selekcija, elitizam, križanje s jednom točkom prekida i jednostavna mutacija objašnjeni su u drugom poglavlju.

Uvjet završetka je provedenih 1000 iteracija.

Genetski algoritam s gore navedenim karakteristikama u programskom jeziku Java nalazi se u prilogu [3].

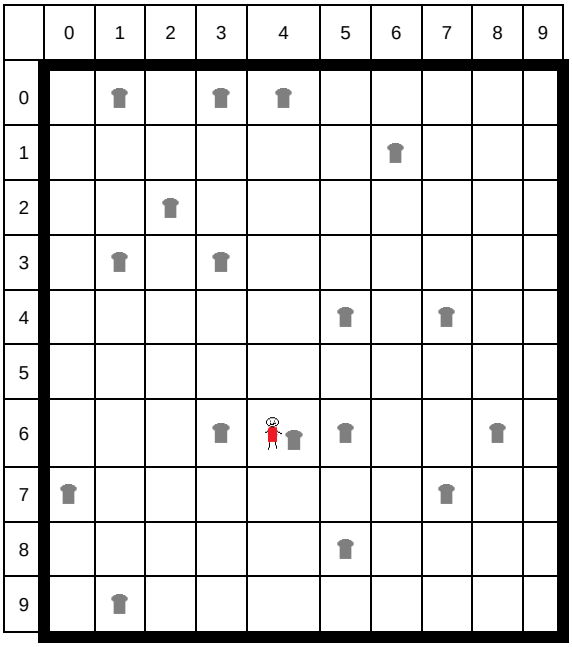
## 3.3. Analiza rezultata

Analizom rada programa jasno je uočljiv napredak populacije. Na slici 3.2 os apscisa predstavlja broj populacije (iteracija), a os ordinata faktor dobrote najbolje jedinke u generaciji. Iako se koristi elitizam, funkcija nije diferencijabilna, jer Robi u nekim situacijama slučajnim odabirom bira akciju koju će obavit i na tako mijenja dobrotu.

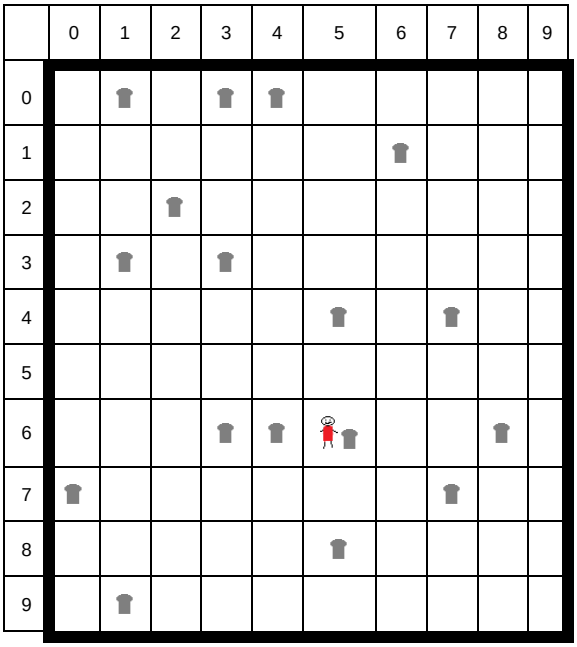



Slika 3.2 Ovisnost dobrote o populaciji

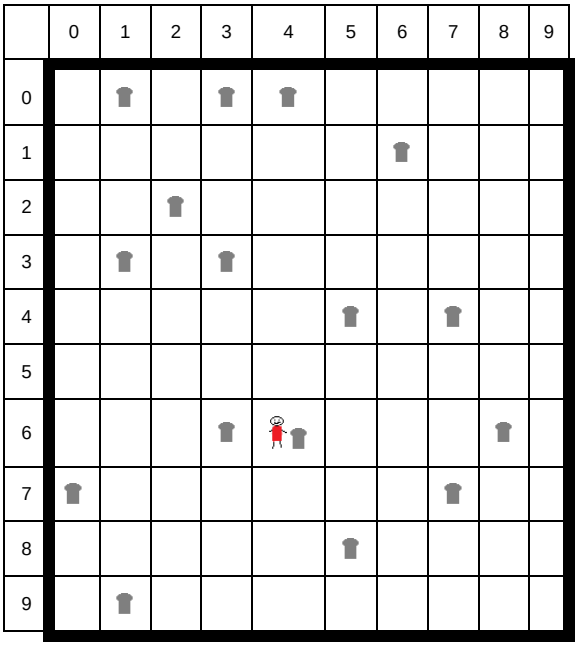
Zanimljivo je što rješenje problema nije strategija prema kojoj Robby uvijek uzme limenku kad stoji na njenom mjestu. Genetskim algoritmom dobivena je strategija prema kojoj u određenim situacijama Robi neće uzeti limenku. Situacija je prikazana na slici 3.3.



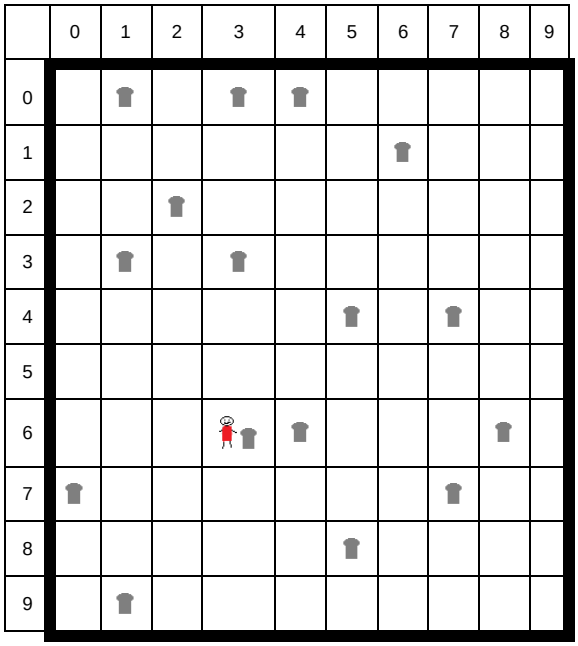
a) ostavlja limenku i odlazi na istok



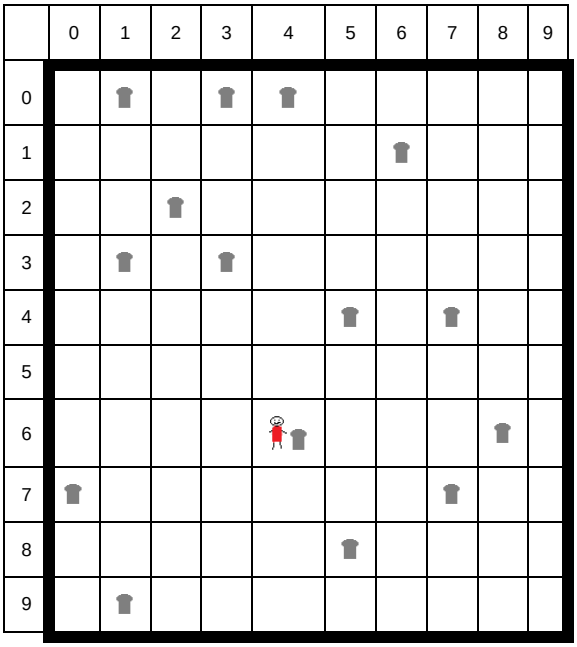
b)uzima limenku i vraća se



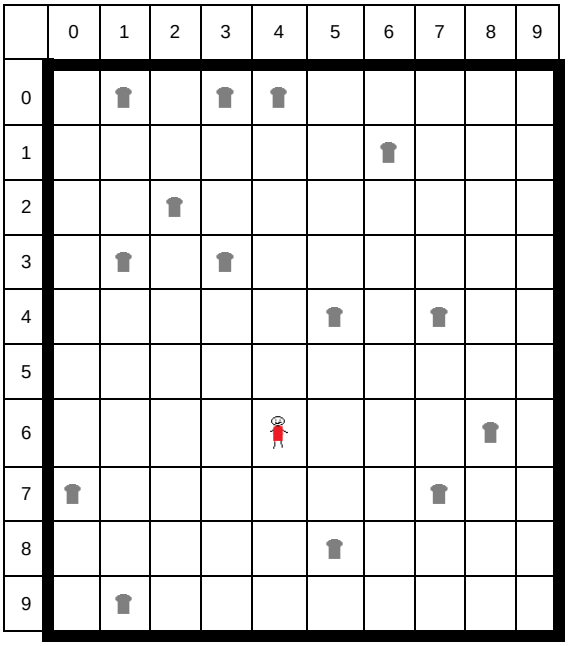
c) ostavlja limenku i odlazi na zapad



d) uzima limenku i vraća se



e) uzima limenku



f)

Slika 3.3 Situacija

Kada se Robby nađe na mjestu na kojem je limenka, a i s njegove dvije ili više strana nalaze se limenke, Robby limenku ne pokuplja. Kada bi Robby pokupio limenku, odabrao bi stranu na koju bi krenuo po sljedeću limenku. Kako na staroj poziciji više nema limenke, a Robby ne pamti, ne bi se vratio i limenke koje su bile s druge strane bi ostale.

# 4. Zaključak

„*Evolutionary algorithms are great tool for exploring the dark corners of design space.*“(Jason Lohn)[3]

Iz primjera opisanog u trećem poglavlju jasno je da su genetski algoritmi laki za primjenu. Iako smo očekivali rezultat u kojem Robby uvijek pokupi limenku, genetski algoritam našao je drugo, bolje rješenje. Genetski algoritmi često nalaze rješenja do kojeg čovjek sam ne bi došao, koja su neobjašnjiva i neshvatljiva, ali dosta bolja od drugih.

3- „Evolucijski algoritmi su izvrstan alat za istraživanje tamnim kutova oblikovanja prostora“

## 5. Literatura


[1] Mitchell M., *Complexity: A Guided Tour,*Oxford University Press, 2009.

[2] Golub M., *Genetski algoritam: prvi dio*, 2004.

[3] Čupić M., *Prirodom inspirirani optimizacijski algoritmi,*verzija:1.0.8, 2010.

[4] Eckel B., *Thinking in Java*, 4th edition, 2008.

[5] Mišljenčević N. i Spasojević B., *Genetski algoritmi*, http://www.zemris.fer.hr/~golub/ga/studenti/projekt2007/ga.html,
*Posjećeno*: 26.ožujak 2012.

# 6. Sažetak

Genetski algoritmi su heuristička metoda optimiranja koja imitira evolucijske procese. Razvoj algoritama počinje oko 1970. godine idejom Johna Hollanda da se računalni program ponaša kao živa jedinka, nastaje, mutira, bori se i opstaje.

Za rad algoritma potrebno je definirati početnu populaciju, funkciju dobrote, vrstu selekcije, odabrati genetske operatore te odrediti uvjet završetka. Genetski algoritam u svakoj iteraciji selektira one jedinke koje su prema faktoru dobrote proglašene boljim, od njih križanjem stvara nove jedinke koje poslije mutacije ulaze u sljedeću populaciju. Algoritam stvara nove populacije sve do ispunjenja uvjeta završetka. Tada odabire najbolju jedinku u populaciji i proglašava je konačnim rješenjem.

U poglavlju 3 opisana je primjena genetskog algoritma i prikazani su rezultati izvođenja.

## 7. Prilog

Genetski algoritam implementiran u programskom jeziku Java [4]:

### 1. Razred GeneticAlgorithm:

```
import java.util.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import java.io.*;
import javax.imageio.*;
import javax.swing.*;

public class GeneticAlgorithm extends Component {
  static BufferedImage Robi,Limenka,Svijet;
  static Random r=new Random();
  static Kromosom najboljaStrategija=new Kromosom(r);
  static Svijet pomocniSvijet=new Svijet(0);
  static int x=0,y=0;

  public static void crtanje() {
       for(int i=0;i<200;i++) {
         SwingUtilities.invokeAndWait (
              new Runnable() {
                 public void run() {
                      JFrame f = new JFrame("Robi");
                      f.addWindowListener(new WindowAdapter() {
                           public void windowClosing(WindowEvent e) {
                                        System.exit(0);
                           }
                      });

                      f.add(new GeneticAlgorithm());
                      f.pack();
                      f.setVisible(true);

                      int situacija=0;
                      //odredimo situaciju i pomak
                      //sjeverno
                      if(x==0) situacija+=2*27*3;
                      else if(pomocniSvijet.polje[x-1][y]==1)
                                        situacija+=27*3;

                      //trenutno
                      if(pomocniSvijet.polje[x][y]==1) {
                                        situacija+=1;
                      }
                      //juzno
                      if(x==9) situacija+=2*27;
                      else if(pomocniSvijet.polje[x+1][y]==1)
                                                        situacija+=27;
```

```
                    //istocno
                    if(y==9) situacija+=2*9;
                    else if(pomocniSvijet.polje[x][y+1]==1)
                                situacija+=9;

                    //zapadno
                    if(y==0) situacija+=2*3;
                    else if(pomocniSvijet.polje[x][y-1]==1)
                                        situacija+=3;

                    int pomak=najboljaStrategija.vrijednost[situacija];

                    while(true){
                          switch(pomak) {
                                        case 0: {
                                              if(x!=0) x--;
                                              break;
                                        }
                                        case 1: {
                                              if(x!=9) x++;
                                              break;
                                        }
                                        case 2:{
                                            if(y!=9)  y++;
                                            break;
                                        }
                                        case 3:{
                                              if(y!=0)  y--;
                                              break;
                                        }
                                        case 4: break;
                                        case 5:{
                                              pomocniSvijet.polje[x][y]=0;
                                        }
                              }
                              if(pomak!=6) break;
                              pomak=r.nextInt(6);
                        }
                     }
          });
          try {
               Thread.sleep(500);
          } catch (InterruptedException ie) {}
       }
}

public void paint(Graphics g) {
   int x=0,y=0;
   Graphics2D g2=(Graphics2D) g;

   g2.drawImage(Svijet,0,0,500,500,this);
   for(int i=0;i<10;i++) {
     for(int j=0;j<10;j++) {
           if(pomocniSvijet.polje[i][j]==1)
                 g2.drawImage(Limenka,i*50+2,j*50+2,40,45,this);
     }
   }
```

```
   g2.drawImage(Robi, x*50, y*50,50,50, this);
}

public GeneticAlgorithm() {
     try {
        Robi = ImageIO.read(new File("Robi.png"));
        Limenka=ImageIO.read(new File("Limenka.jpg"));
        Svijet=ImageIO.read(new File("Svijet.png"));
     } catch (IOException e) {}
}

public static void main(String[] args) {
           int vel_pop=200;
           double vjer_mut=0.005;
           Kromosom najbolji=null;

           Random rand=new Random();

           //stvaranje svijetova
           final Svijet[] svijetovi=new Svijet[100];
           for(int i=0;i<100;i++) {
                 svijetovi[i]=new Svijet(rand.nextInt(101));
                 svijetovi[i].stvaranje();
           }

           kopirajSvijet(svijetovi[0],pomocniSvijet);

           //stvaranje populacije
           Kromosom[] populacija=stvoriPopulaciju(vel_pop,rand);
           Kromosom[] novaGeneracija=stvoriPopulaciju(vel_pop,rand);

           //definiramo funkciju koju optimiramo
           IFunkcija funkcija=new IFunkcija() {
             public int izracunaj(int[] varijable) {
                  Random rand1=new Random();
                  Svijet pomocni=new Svijet(0);
                  int vrijednost=0;
                  for(int i=0;i<100;i++) {
                    kopirajSvijet(svijetovi[i],pomocni);

                    //pocetna pozicija
                    int x=0,y=0;
                   int brojac=0;
                    //u 200 koraka
                    while(brojac<200) {
                         int trenutno=0;
                         int situacija=0;
                         //odredimo situaciju i pomak
                         //sjeverno
                         if(x==0) situacija+=2*27*3;
                          else if(pomocni.polje[x-1][y]==1)
                                situacija+=27*3;

                         //trenutno
                         if(pomocni.polje[x][y]==1) {
                                situacija+=1;
                                trenutno=1;
```

```
                }

                //juzno
                if(x==9) situacija+=2*27;
                else if(pomocni.polje[x+1][y]==1)situacija+=27;

                //istocno
                if(y==9) situacija+=2*9;
                else if(pomocni.polje[x][y+1]==1) situacija+=9;

                //zapadno
                if(y==0) situacija+=2*3;
                else if(pomocni.polje[x][y-1]==1) situacija+=3;

                int pomak=varijable[situacija];

                while(true) {
                      switch(pomak) {
                                case 0: {
                                      if(x==0) vrijednost-=5;
                                      else x--;
                                      break;
                                }
                                case 1: {
                                      if(x==9) vrijednost-=5;
                                      else x++;
                                      break;
                                }
                                case 2: {
                                       if(y==9) vrijednost-=5;
                                       else y++;
                                       break;
                                }
                                case 3: {
                                       if(y==0) vrijednost-=5;
                                       else y--;
                                       break;
                                }
                                case 4: break;
                                case 5: {
                                       if(trenutno==1)
                                          vrijednost+=10;
                                       else vrijednost-=1;
                                       pomocni.polje[x][y]=0;
                                }
                      }
                      if(pomak!=6) break;
                      pomak=rand1.nextInt(6);
                }
                            brojac++;
          }

    }
    return vrijednost;
  }
};
```

```
            //pocetna evaluacija populacije
            evaluirajPopulaciju(populacija,funkcija);

            //ponovi kroz 1000 generacija
            for(int generacija=0;generacija<1000;generacija++) {
              Arrays.sort(populacija);

              //stvaranje jedinki nove generacije
              kopirajKromosom(populacija[0],novaGeneracija[0]);
              kopirajKromosom(populacija[1],novaGeneracija[0]);

              for(int i=1;i<vel_pop/2;i++) {
                 Kromosom roditelj1=odaberiRoditelja(populacija,rand);
                 Kromosom roditelj2=odaberiRoditelja(populacija,rand);
                 Kromosom dijete1=novaGeneracija[2*i];
                 Kromosom dijete2=novaGeneracija[2*i+1];

                 krizaj1TockaPrijeloma(roditelj1,roditelj2,dijete1,dijete2,rand);

                 mutiraj(vjer_mut,dijete1,rand);
                 mutiraj(vjer_mut,dijete2,rand);
              }

              //mijenjamo staru i novu populaciju
              Kromosom[] pomocni=populacija;
              populacija=novaGeneracija;
              novaGeneracija=pomocni;

              evaluirajPopulaciju(populacija,funkcija);

              //pronadi najbolje rjesenje

              for(int i=0;i<populacija.length;i++) {
                   if(i==0 || najbolji.fitnes<populacija[i].fitnes){
                                najbolji=populacija[i];
                         }
              }

              System.out.println("trenutno rjesenje:f("
                           +Arrays.toString(najbolji.vrijednost)+")="
                           +funkcija.izracunaj(najbolji.vrijednost));
            }

            kopirajKromosom(najbolji,najboljaStrategija);

            new Thread() {
                  public void run() {
                        crtanje();
                  }
            }.start();
      }

      private static void kopirajSvijet(Svijet original,Svijet kopija) {
            for(int i=0;i<10;i++) {
                  for(int j=0;j<10;j++) {
                        kopija.polje[i][j]=original.polje[i][j];
                  }
```

```
            }
    }

    private static void kopirajKromosom(Kromosom original,Kromosom kopija) {
                for(int i=0;i<original.vrijednost.length;i++) {
                        kopija.vrijednost[i]=original.vrijednost[i];
                }
    }

    public static Kromosom[] stvoriPopulaciju(int brojJedinki,Random rand) {
                Kromosom[] populacija=new Kromosom[brojJedinki];
                for(int i=0;i<populacija.length;i++) {
                        populacija[i]=new Kromosom(rand);
                }
                return populacija;
    }

    public static void evaluirajPopulaciju(Kromosom[] populacija, IFunkcija
funkcija) {
          for(int i=0;i<populacija.length;i++) {
             populacija[i].fitnes=funkcija.izracunaj(populacija[i].vrijednost);
          }
    }

    private static Kromosom odaberiRoditelja(Kromosom[] populacija, Random rand) {
             double sumaDobrota=0;
             int najvecaVrijednost=populacija[0].fitnes;
             for(int i=0;i<populacija.length;i++) {
                sumaDobrota+=najvecaVrijednost-populacija[i].fitnes;

             }

                double slucajniBroj=rand.nextDouble();
                double broj=0;

             for(int i=0;i<populacija.length;i++) {
                 broj+=1-(najvecaVrijednost-populacija[i].fitnes)/sumaDobrota;
                if(slucajniBroj<broj) return populacija[i];
           }
                return populacija[populacija.length-1];
    }

    private static void krizaj1TockaPrijeloma(Kromosom roditelj1,Kromosom
                        roditelj2,Kromosom dijete1,Kromosom dijete2,Random rand) {
                int tockaPrijeloma=rand.nextInt(243);
                for(int i=0;i<tockaPrijeloma;i++) {
                                dijete1.vrijednost[i]=roditelj1.vrijednost[i];
                                dijete2.vrijednost[i]=roditelj2.vrijednost[i];
                }
                for(int i=tockaPrijeloma; i<243;i++) {
                                dijete1.vrijednost[i]=roditelj2.vrijednost[i];
                                dijete2.vrijednost[i]=roditelj1.vrijednost[i];
                }
    }

    private static void mutiraj(double vjerMut, Kromosom dijete, Random rand) {
                for(int i=0;i<243;i++) {
```

```
                    if(rand.nextFloat()<=vjerMut) {
                          int pomocni=rand.nextInt(7);
                          dijete.vrijednost[i]=pomocni;
                    }
              }
  }

}
```

## 2. Razred Kromosom

```
package hr.fer.zemris.java.tecaj_1;

import java.util.*;

public class Kromosom implements Comparable<Kromosom> {

  int fitnes;
  int[] vrijednost;

  //bolja jedinka ima veci fitnes
  public int compareTo(Kromosom o) {
     if(this.fitnes<o.fitnes) {
          return 1;
     }
     if(this.fitnes>o.fitnes) {
          return -1;
     }
     return 0;
  }

  public Kromosom(Random rand) {
     vrijednost=new int[243];
     if(rand!=null) {
          for(int i=0;i<243;i++) {
            vrijednost[i]=rand.nextInt(7);
          }
     }

    fitnes=0;
  }
}
```

### 3. Razred svijet

```
package hr.fer.zemris.java.tecaj_1;
import java.util.Random;

public class Svijet {
	int[][] polje;
	Random generator=new Random();
	int stupac;
	int redak;
	int brLimenki;


	public void stvaranje() {
		while(brLimenki!=0) {
			stupac=generator.nextInt(10);
			redak=generator.nextInt(10);
			if(polje[redak][stupac] != 1)
				polje[redak][stupac]=1;
			else brLimenki++;
			brLimenki--;
		}

	}

	public Svijet(int i) {
		polje=new int[10][10];
		brLimenki=i;
	}
}
```

### 4. IFunkcija

```
package hr.fer.zemris.java.tecaj_1.numeric;

public interface IFunkcija {
    public int izracunaj(int[] varijable);
}
```