UDK: 004.42JAVA Stručni rad

IMPLEMENTACIJA REŠENJA PROBLEMA TRGOVAČKOG PUTNIKA GENETSKIM ALGORITMIMA U JAVA PROGRAMSKOM JEZIKU

SOLUTION IMPLEMENTATION GENETIC ALGORITHMS OF TRAVELING SALESMAN PROBLEM IN JAVA PROGRAMMING LANGUAGE

Selver Pepić[1], Zoran Lončarević[2], Goran Miodragović[1], Slobodan Aleksandrov[1]

[1]*Visoka tehnička mašinska škola strukovnih studija Trstenik*

[2]*Visoka škola strukovnih studija za IT Beograd*

[1]selver.pepic@vtmsts.edu.rs, [2]zoran.loncarevic@its.edu.rs

[1]goran.miodragovic@vtmsts.edu.rs

[1]slobodan.aleksandrov@vtmsts.edu.rs

Rezime: *U ovom radu prikazana je implementacija algoritma trgovačkog putnika, koja ima veliku primenu u adresiranju i slanju paketa kroz mrežnu infrastrukturu. U cilju unapređenja efikasnosti trgovačkog putnika prikazan je GA (genetski algoritam). Predstavljeni su i rezultati postignuti primenom ovog algoritma implementiranog u Java programskom jeziku.*

Ključne reči: *Problem trgovačkog putnika,genetski algoritmi, Java programiranje.*

Abstract: *This paper shows the implementation of the algorithm of a Salesman, which is widely used in addressing and sending packets through the network infrastructure. In order to improve the efficiency of a Salesman is shown GA (genetic algorithms). Presented are the results achieved by applying this algorithm implemented in the Java programming language.*

Key words: *Traveling salesman problem, genetic algorithms, Java programming.*

1. UVOD

Problem trgovačkog putnika (Traveling Salesman Problem, TSP) je jedan od najpoznatijih problema iz grupe NP - teških problema diskretne i kombinatorne optimizacije, čija je složenost O(n!). Leonhard Euler /Ojler/ (1707 - 1783) poznati švajcarski matematičar i fizičar prvi je proučavao probleme slične problemu trgovačkog putnika, on je proučavao problem kako bi skakač prešao sva polja na šahovskoj table i to isključivo svako polje samo jednom. Poznato je i to da su Matematičari William Rowan

Hamilton i Thomas Kirkman proučavali probleme koji se svode na problem trgovačkog putnika početkom XX veka, dok se opšta forma ovog problema pojavljuje negde 30-ih godina XX veka. Pojam "trgovački putnik" prvi put je upotrebljen 1932. godine.
Problem trgovačkog putnika je u osnovi relativno prost - trgovački putnik tačno zna koje gradove treba da poseti, takođe zna i kolika je njihova međusobna udaljenost, jedini problem je taj što je u obavezi da posetiti svaki grad samo jednom i vratiti se u grad polaska. Rešenje problema se ogleda u tome da trgovački putnik odredi redosled gradova i da pritom putuje najoptimalnijom mogućom rutom, što bi značilo najkraćim i najbržim putem. Pri čemu kretanje od grada do grada se odvija istom brzinom, odnosno kraći put je brži put. Na prvi pogled ovaj se problem i ne čini teškim, ali ako se uzme u obzir da ima faktorijelnu složenost, računanjem se dobija da već za 10 gradova postoji 3,628,800 mogućih kombinacija obilazaka gradova. U teoriji, ali i u praksi postoji mnogo problema koji su povezani sa problemom trgovačkog putnika, ili se prosto svode na njega.

Broj Gradova	Broj Mogucih Koraka
1	
2	
3	
4	
5	1
6	7
7	50
8	403
9	3628
10	36288
20	2,4329E+
30	2,65253E+
50	3,04141E+
100	9,33262E+1

Slika 1. Složenost problema trgovačkog putnika

2. REŠAVANJE PROBLEMA TRGOVAČKOG PUTNIKA

Zbog svoje faktorijelne složenosti, problem trgovačkog putnika predstavlja pravi izazov za rešavanje. Postoji veliki broj algoritama za rešavanje, a pojavom računara uspešnost optimizacije i rešavanja ovog problema se znatno uvećala. Problemi projektovanja ruta saobraćajnih sredstava su po svojoj prirodi kombinatorni.[1] Od velikog broja mogućnosti neophodno je izabrati najbolje moguće rešenje. Optimalno rešenje predstavlja ona ruta kojoj odgovaraju najmanji transportni troškovi, vreme putovanja ili sama dužina rute u zavisnosti koji od ovih parametara ima najznačajniju ulogu u određenom problema, odnosno koja je cena rute minimalna.
Dakle, za dati skup gradova i cenu putovanja između svakog para gradova, problem trgovačkog putnika mora pronaći najjeftiniji put koji će obići sve gradove tačno jednom i na kraju se vratiti u početni grad. Iako je definicija vrlo jednostavna i jasna problem je jako težak. Kao što je već rečeno, problem je faktorijalne složenosti što u prevodu znači da za 10 gradova broj mogućih rešenja iznosi 10! = 3,628,800. Ako se ima veći broj

gradova, što su u današnje vreme realne ture, 100-tinak gradova, broj mogućih tura se povećava na približno 9.3e157, što se uz današnju snagu računara ne može rešiti 3e144 godina uz pretpostavku da je današnji racunar može obraditi 1.000.000 ruta po sekundi.
Problem se može proširiti dodavanjem različitih ograničenja. Tako je nastao i vremenski zavisan problem trgovačkog putnika, koji uz minimalnu daljinu puta u obzir uzima i vreme koje je potrebno da se put obavi kao i eventualne vremenski periodi u kojima se jedan deo posla mora napraviti. Uz ovu defniciju postoji još i asimetrični problem trgovačkog putnika u kojem put između grada A i grada B nije isti kao i put između grada B i grada A.
Problem trgovačkog putnika (TSP) je jedan od najstarijih i najpoznatijih optimizacijskih problema. Njegova popularnost ogleda se u jednostavnoj formulaciji, težak je za rešavanje i ima veliki broj praktičnih primera.
Neka je dat povezan težinski graf *G* sa *n* čvorova. Potrebno je pronaći najkraći put, koji prolazi kroz sve čvorove tačno jedanput i završava se u početnom čvoru (*Hamiltonova kontura*). Hamiltonova kontura je put koji kroz sve čvorove grafa prolazi jednom i samo jednom i završava se u početnom čvoru:

$$hk = (i1, i2, i3, \ldots, in-1, in, i1), ip \neq ik \; \forall \, p,k \in \{1,\ldots,n\}, p \neq k. \qquad (1)$$

Zadatak nalaženja najkraće Hamiltonove konture mreže naziva se problem trgovačkog putnika. Polazeći od prvog grada potrebno je obići *n* novih gradova kada su poznata njegova međusobna rastojanja. Zadatak putnika je da obiđe sve planirane gradove i da izabere takav raspored njihovog obilaženja koji bi garantovao da pređeni put trgovačkog putnika bude najkraći. Drugim rečima, problem koji se postavlja sastoji se u izboru redosleda obilaženja pojedinih gradova. Neki grafovi imaju jednu ili više Hamiltonovih kontura, a neki nijednu. Graf koji sadrži Hamiltonovu konturu, naziva se Hamiltonov graf. Formulacija odgovarajućeg matematičkog modela bi mogla izgledati ovako:
C_{ij} - rastojanje između gradova (*i-j*),
$x_{ij} = 1$, ukoliko trgovački putnik putuje iz grada *i* u grad *j*,
$x_{ij} = 0$, u protivnom slučaju,
n - broj gradova koje treba obići,
u_i - proizvoljne realne vrednosti.
Odgovarajući matematički model sastoji se u nalaženju minimuma funkcije cilja

$$\min F(x) = \sum_{i=0}^{n} \sum_{j=0}^{n} C_{ij} \cdot X_{ij} \qquad (2)$$

pri ograničenjima:

$$\sum_{i=0}^{n} x_{ij} = 1 \;\; (j = 1, \ldots, n) \qquad (3)$$

$$\sum_{j=1}^{n} x_{ij} = 1 \;\; (j = 1, \ldots, n) \qquad (4)$$

$$u_i - u_j + n \cdot x_{ij} \leq n - 1 \;\; (i, j - 1, \ldots, n; i \neq j) \qquad (5)$$

$$x_{ij} \in \{0,1\} \;\; (i, j = 1, \ldots, n) \qquad (6)$$

Prvo ograničenje ima značenje da trgovački putnik odlazi iz svakog grada samo jedanput, a drugo ograničenje da trgovački putnik dolazi u svaki grad samo jedanput. Naredno ograničenje obezbeđuju da je put trgovačkog putnika obavezno cikličan. U protivnom, pređeni put bi mogao da se raspadne na nekoliko podciklusa koji nisu

međusobno povezani, tj. postojao bi neki podciklus sa brojem temena manjim od broja *n*, koji ne prolazi kroz grad 0. Promenljive $x_{ij} = 1 \ (j = 1, \dots, n)$ odnosno x_{ij}, $i, j \in N$ (skup čvorova – broj gradova), su binarne promenljive. Kada je $x_{ij} = 0$, grana *(i, j)* ne pripada *hk*, a kada je $x_{ij} = 1$, grana *(i, j)* pripada *hk*. Promenljive u_i, $(i = 1,\dots,n)$ su pomoćne promenljive koje nemaju značaja kada se rešenje dobije. U slučaju kada graf ne sadrži Hamiltonovu konturu, ovaj problem neće imati dopustivo rešenje[2].
Uvodimo matricu rastojanja *W*, gde *w(i j)* predstavlja rastojanje između čvorova *i* i *j*, ukoliko postoji grana koja ih spaja, u suprotnom se uzima vrednost ∞. Dakle, potrebno je pronaći permutaciju *p* skupa čvorova *V ={1, 2, ... , n}*, tako da minimizujemo funkciju:

$$f(p) = min\,(w(p1,p2) + w(p2,p3) + \dots + w(pn-1;\ pn) + w(pn,p1)) \quad (7)$$

Prostor pretraživanja svih permutacija je veliki. Za 30 čvorova, postoji 30*!* permutacija. Zbog toga je od posebnog značaja razvoj heurističkih metoda, koje daju rešenje blisko optimalnom, uz relativno kratko vreme izvršavanja. Ovde dolazi do žrtvovanja optimalnosti, u korist vremena izvršenja programa. Algoritme za rešavanje problema iz ove oblasti možemo podeliti na:

1. Egzaktne čija primena garantuje nalaženje optimalnog rešenja:
   - Metod vraćanja po tragu
   - Eksplicitna enumeracija ili potpuno pretažiavanjeimplicitna enumeracija ili posredno pretraživanje
   - Dinamičko programiranje
2. Heurističke koji daju dovoljno dobro rešenje, tj. rešenje koje je blizu optimalnog, a često se dešava da je upravo optimalno.Mogu se svratati u dve grupe:
   - Nalaženje početnog rešenja
   - Metod za poboljšanje rešenja

3. GENETSKI ALGORITMI

Genetski algoritmi (*Genetic Algorithms, GA*) su najpoznatija vrsta evolucijskih algoritama, a predstavljaju model optimizacije čije ponašanje oponaša prirodni proces evolucije. Osnovni cilj genetskih algoritama je pronalaženje rešenja nekog problema koje tradicionalne determinističke metode ne mogu rešiti. Za razliku od većine determinističkih algoritama, genetski algoritmi pretragu ne započinju od jedne tačke nego od niza potencijalnih rešenja. Ta potencijalna rešenja su najčešće kreirana slučajnim odabirom, a predstavljaju početnu populaciju genetskog algoritma. Početnoj populaciji određuje se sposobnost (fitness). Sposobnost je mera kvaliteta i govori koliko je neko rešenje dobro - bolja rešenja će imati veću sposobnost i obrnuto. Sposobnost se određuje funkcijom cilja, koja zavisi od problema koji se rešava. Kada se odredi sposobnost, može započeti genetski proces. Početna populacija rešenja se iterira kroz generacije, a primenjuje se princip preživljavanja najsposobnijeg (*survival of the fittest*). U svakoj od generaciji biraju se bolja rešenja i odbacuju ona lošija odnosno manje podobna. Odabrana rešenja su podvrgnuta genetskim operatorima [3]. Prvo se ukršta, a zatim se sa određenom verovatnošću i mutiraju. Tako nastaju nova potencijalna rešenja koja predstavljaju novu generaciju genetskog algoritma. Ta nova rešenja su po pravilu

bolja nego stara rešenja od kojih su nastala. Ceo proces se zasniva na prirodnom procesu evolucije. Stvaranje novih rešenja se nastavlja dok se ne zadovolji kriterijum završetka genetskog procesa koji je definirao. Kada se kriterijum zadovolji, genetski algoritam je završio sa radom. Međutim, to ne znači da je pronađeno optimalno rešenje. Završetak rada genetskog algoritma znači samo da je nađeno dovoljno dobro rešenje, koje može a i ne mora biti najbolje.

```
Genetski_algoritam(Velicina_Populacije, t, p_m, p_c, M)
{
   t =  0;
   generiši početnu populaciju potencijalnih rešenja P(0);
   sve dok nije zadovoljen uslov završetka evolucijskog procesa
   {
      t = t + 1;
      selektuj P'(t) iz P(t-1);
      ukrštaj jedinke iz P'(t) i decu sačuvaj P(t);
      mutiraj jedinke P(t);
   }
   ispiši rešenje;
}
```

Slika 2. Pseudo kod Genetskog algoritma

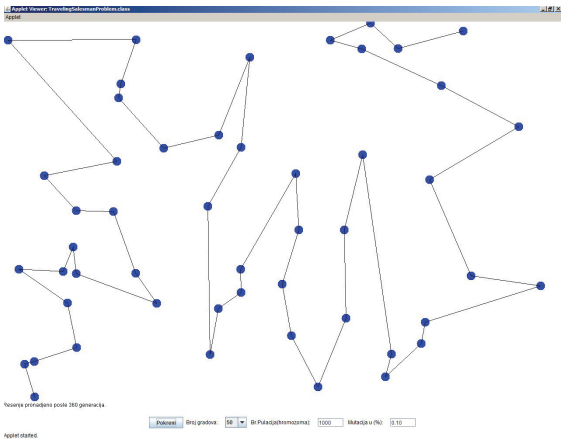
4. REŠAVANJE PROBLEMA TRGOVAČKOG PUTNIKA UZ POMOĆ GENETSKOG ALGORITMA

U implementaciji genetskog algoritma koristi se permutacijska reprezentacija čvorova grafa, tj. svaka jedinka se sastoji od niza rednih brojeva čvorova u permutaciji p i njihovih međusobnih udaljenosti izračunavanjem rastojanja primenom pitagorine teoreme. U implementaciji je primenjena inicijalizacija, selekcija, ukrštanje i mutaciju. Algoritam počinje sa skupom inicijalizacijom gradova, odnosno rešenja (predstavljenih hromozomima) koji se naziva populacija. Inicijalizacija je proces koji se odvija jedanput, na početku rada algoritma. Ovaj deo je zadužen da načini polaznu populaciju koja će daljim procesima prirodne selekcije evoluirati u bolje populacije. Jedan od pristupa je sačiniti nasumičnu populaciju. Kod problema TSP, nasumična populacija predstavlja skup nasumičnih permutacija skupa {1,2,…,n}, gde je n broj gradova. Selekcija je proces koji se odvija između svake dve uzastopne generacije. U ovom delu se iz cele populacije izdvajaju jedinke koje su dovoljno podobne da pređu u sledeću generaciju, ili ekvivalentno, one jedinke koje nisu dovoljno jake, te stoga neće pripadati sledećoj generaciji odnosno potomstvu. Selekcija u ovom slučaju je funkcija koja određuje podobnos neke jedinke računajući dužinu rastojanja između dva grada, odnosno dužina puta koju bi trgovački putnik morao da pređe ako bi gradove posećivao u poretku zadatom permutacijom te jedinke. Konstantan procenat p_e jedinki biće eliminisan. Postoje različiti pristupi za izbor ovog procenta.[4] Zamka u koju se često upada je odbaciti p_e procenat jedinki sa najnižim odnosom podobnosti odnosno rastojanja. Ovo u većini slučajeva nije najoptimalnije rešenje, sa obzirom da u selekciji treba obratiti pažnju ne samo na podobnost jedinki (međusobno najkraće rastojanje) već i na raznovrsnost budućih generacija.[5] Ukrštanje je ekvivalent biološkom ukrštanju

(parenju). Biramo po dve jedinke i od njih formiramo druge dve - nalik detetu dve roditeljske jedinke. Ovim procesom stvaramo onoliko jedinki koliko nam je potrebno da bi generacija ponovo sadržala *M* jedinki, tj. onoliko koliko smo eliminisali prethodnim korakom. Za izbor roditelja koriščen je metod „turnirske" selekcije kao što je navedeno u [6] Odabira se slučajani broj jedinki iz populacije (sa ili bez zamena), a najbolje od njih postaju roditelji sledeće generacije. Mutacija je poslednji korak u jednom ciklusu. U ovom koraku se bira procenat *pi* jedinki nasumično, i one se na neki način promene, opet simulirajući mutaciju u biološkoj evoluciji.[6] Značaj ovog koraka je opet povećanje raznovrsnosti populacije, jer se mutacijom dobijaju nove jedinke sa manje ili nimalo korelacije u odnosu na ostale jedinke iste populacije.

4. IMPLEMENTACIJA GA ALGORITMA U JAVI

Cilj je kreirati aplikacjiju u javi koja će pronaći najkraći put od unapred deifisanog broja gradova, broja populacija i procenta mutacije. Nakon početnog setovanja parametara aktivira nit i iscrtava rešenje na ekranu slično primeru u [7]. Sledi opis implementiranog rešenja.



Slika 3. Grafički prikaz rešenja

Važno je razumeti odnos između pojedinih klasa koje čine program trgovačkog putnika. Implementirano rešenje se sa stoji od tri klase (class Ggrad, class Chromosome i class TravelingSalesmanProblem).

class Ggrad - Ova klasa skladišti kordinate gradova. Ova klasa takođe sadrži metode koje se koriste da se izračuna udaljenost između gradova.

class Chromosome - Ova klasa implementira hromozom. Ovo je najsloženiji klasa programa, koja sprovodi većinu funkcionalnosti genetskog algoritma. Ona u sustini predstavlja implementaciju GA.TravelingSalesman - Ova klasa implementira korisnički interfejs i vrši opštu inicijalizaciju, kao iscrtavanje grafičkog rešenja.

Ovom klasom implementiran je, kao što je već spomenuto, veći deo GA. U ovoj klasi vrši se parenje hromozoma, mutiranje i sortiranje liste hromozoma po njihovoj podobnosti. Najvažniji metod u ovoj klasi je *mate* metod. Ovaj metod će izazvati otac

koji se nalazi kao parametar u ovom metodu, koji će se ukrštati sa matičnom klasom. Ovaj metod koristi algoritam koji kombinuje genetski materijal dva hromozoma za stvaranje nova dva potomka.[7] Slučajnim odabirom se utvrđuje koliki je procenat gena koji će se uzeti od svakog roditelja u kreiranju puta od grada do grada.
U tekstu koji sledi su dati fragmenti koda.

```
int tacka1 = (int) (0.999999*Math.random()*(double)(GgradList.length-duzina_gena));
int tacka2 = tacka1 + duzina_gena;
```

Sledeća dva logička niza su podešavanja koja ukazuju na gene koji su uzeti od svakog roditelja. Ovaj niz je inicijalizovan tako da bude neistinit (*false*).

```
boolean preuzmi1 [] = new boolean[GgradList.length];
boolean preuzmi2 [] = new boolean[GgradList.length];
int prvi [] = new int[GgradList.length];
int drugi [] = new int[GgradList.length];
for ( int i=0;i<GgradList.length;i++ ){
        preuzmi1[i] = false;
        preuzmi2[i] = false;
}
```

Proces ukrštanja je relativno jednostavan proces. Majka i otac su hromozomi, izrezani na dve tačke, i svaki od njih se deli na tri dela. Ta tri dela se zatim ponovo selektivno spajaju, pa se formiraju dva potomka. Prvo se obrađuje srednju komponentu, što je prikazano sledećim segmentom koda.

```
for ( int i=0;i<GgradList.length;i++ ){
        if ( i<tacka1 || i>= tacka2 ) {
                prvi[i] = -1;
                drugi[i] = -1;
         } else {
                int imajka = GgradList[i];
                int iotac = otac.getGgrad(i);
                prvi[i] = iotac;
                drugi[i] = imajka;
                preuzmi1[iotac] = true;
                preuzmi2[imajka] = true;
        }
}
```

Srednja komponenta uzima se od oca za dete jedan, a od majke za dete dva. Svaki od gena koji se koriste za ovu sekciju, označiće se kao preuzeti. Kroz sledeću petlju pristupamo levom delu hromozoma. Za levu stranu hromozoma, prvo dete će uzeti dostupne gene od majke, a drugo dete dostupne gene od oca. Na kraju, program mora obraditi i desnu stranu hromozoma. Desni deo hromozoma radi slično što i levi. Glavna razlika se ogleda u tome što će prvo dete uzeti dostupne gene od oca, a drugo od majke.
Proces mutacije je potpuno slučajan. Dva slučajno odabrana gena će biti zamenjena kao deo procesa mutacije. Slučajna mutacija je korisna, kako bi se suprotstavio veoma promišljen proces razmnožavanja koji se koristi. Drugi najbitniji metod ove klase je *sortChromosomes.* Ovaj metod se koristi za sortiranje hromozoma prema sposobnosti, odnosno prilagodjenosti problema, što omogućava da manje podobni hromozomi budu odbačen, odnosno postavljeni na dnu liste dok oni podobniji se nalaze na vrhu liste koja

služi za ukrštanje. Dakle, ovim metodom se vrši selekcija jedinki koje će ostvariti svoje potomstvo.

```
public static void sortChromosomes(Chromosome chromosomes[],int num) {
        Chromosome ctemp;
        boolean swapped = true;
        while (swapped ) {
                swapped = false;
                for ( int i=0;i<num-1;i++ ) {
                        if ( chromosomes[i].getduzina() > chromosomes[i+1].getduzina() )
{                               ctemp = chromosomes[i];
                                chromosomes[i] = chromosomes[i+1];
                                chromosomes[i+1] = ctemp;
                                swapped = true;
                        }
                    }
                }
```

5. ANALIZA REZULTATA

Prikazano rešenje je testirano na računaru pod OS Windows Ultimate SP1 i 32bit-nom platformom, na core 2 quad procesoru q6600 sa 4GB Ram memorije. Testiranjem je trebalo utvrditi sledeće: da li program može pronaći optimalnu rutu do nivoa težine od sto gradova, u kojoj generaciji je pronađeno rešenje, kao i uticaj broja početne populacije, procenta mutacije,veličine prozora i vremena izvršavanja.

Tabela 1. Prosečni rezultati evaluacije GA algoritma

Broj gradova	Vrednost srednje dužine puta	Vrednost srednje dužine minimalnih puteva	Broj generacija pronalaska rešenja	Vreme pronalaska rešenja
5	1270	781	1 - 2	29.3 – 65.1 *ms*
10	1540.06	1223.5	3 - 13	38 - 4707 *ms*
20	2346.2	2018.1	38 - 101	257 - 27840 *ms*
30	3018.4	2679.2	65 - 234	409 - 58019 *ms*
50	4129.35	3648.85	120 - 416	967 - 47551 *ms*
100	6952.5	6433.3	416 - 9019	5529 - 581936 *ms*

Rezultati testiranja ukazuju na to da do nivoa težine od pet gradova sa postavkom parametara rešenje se pronalazi već u drugoj generaciji što je donekle i očekivano. Rešenje je uvek 100% tačno, što znači da je put uvek optimalan. Probleme sa 10 gradova program pronalazi gotovo trenutno već u prvih dvadeset generacija. U tabeli iznad prikazani su zbirni rezultati testiranja u okvirima radnog prozora rezolucije 1024x768 i 800x600. Iz ove tabele može se zaključiti nekoliko stvari bitnih za genetske algoritme. Sa povećanjem broja gradova, na primeru koji je testiran, povećava se ukupna udaljenost između gradova, isto se dešava kada se poveća i rezolucija ekrana, takođe dolazi do drastičnog povećanja vremena izvršavanja genetskog algoritma. Algoritam gotovo uvek pronalazi optimalno rešenje do nivoa od 30 gradova. Bitno je napomenuti da je vreme

izvršavanja algoritma obrnuto proporcionalno broju populacija. U slučaju kada su loše definisani parametri, često se dešava da program puca, zbog čega je potrebno dodatno unaprediti aplikaciju. Za postavku sa pet gradova algoritam daje uvek idealno rešenje za period od svega nekoliko ms. U primeru koji je testiran na rezolucijama ekrana od 1280x1024 i 1024x768, daje prosečno izvršavanje algoritma za 52,1 *ms*, odnosno za 29,3 *ms*, kroz niti [8] za navedene rezolucije respektivno.
Postupci testiranja sprovedeni su na deset, dvadeset, trideset, pedeset i sto gradova. Aplikacija je testirana na isti način za svaki od nivoa gradova od 10 - 100, pod kojim se podrazumeva deset nasumičnih testiranja programa sa uvek istom postavkom parametara. Postavke parametara se odnose na postavke populacije od 1000, 5000 i 10000 populacija, kao i primenom tri vrednosti mutacije za svaku selekciju populacije od 10%, 50% i 100%, postupak testiranja izvodi se u dve rezolucije ekrana od 800x600 i 1024x768 piksela.

LITERATURA

[1] Marko Pielić, *Rješavanje problema trgovačkog putnika uz pomoć genetskih algoritama*, završni rad br. 213, Sveučilište u Zagrebu - Fakultet Elektrotehnike i računarstva, Zagreb, jun 2008.
[2] P.Stanimirović, G. Milovanović, I. Jovanović (2008). *Primene linearnog i celobrojnog programiranja*, Niš.
[3] P. Stanimirović, A. Ilić (2008). *Analiza algoritama za probleme TSP i VRP*, Niš.
[4] *Matematičko Programiranje i Optimizacija*, Matematički fakultet Univerziteta u Beogradu, 2012.
[5] I.Salajic, J.Nikolic, M. Žoljom (2001), *TSP – Problem trgovalkog putnika na potpunom grafu, primjenom genetskog algoritma (GA) i algoritma simuliranog kaljenja (SA)*.
[6] D. Saiko (2005), *Traveling Salesman Problem , Java Genetic Algorithm Solution*
[7] G. Gutin, A. Punnen (2002), *The traveling salesman problem and its variations*, Kluwer, Dordrecht.
[8] http://www.saiko.cz/ai/tsp/
[9] http://www.or.deis.unibo.it/research_pages/tspsoft.html
[10] http://www.tsp.gatech.edu/
[11] http://rti.etf.bg.ac.rs/rti/ir2oo2/predavanja/Java/09%20Niti%20-%20Java.pdf