SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 319

Rješavanje problema trgovačkog putnika uz pomoć evolucijskih strategija

Iva Malović

Zagreb, lipanj 2008.

Sažetak

U ovom radu je opisana jedna vrsta evolucijskih algoritama pod nazivom evolucijske strategije, te je napravljena aplikacija za rješavanje problema trgovačkog putnika uz pomoć evolucijskih strategija.

Ključne riječi: problem trgovačkog putnika, evolucijska strategija, mutacija, križanje.

Abstract

Title: Solving the travelling salesman problem with evolution strategies

In this document is described a kind of evolution algorithm called evolution strategy, and it is developed an application for solving the Travelling salesman problem with evolution strategies.

Key words: travelling salesman problem, evolution strategy, mutation, crossover.

Sadržaj

1. Uvod

U ovom radu opisan je način rješavanja problema trgovačkog putnika uz pomoć evolucijskih strategija. U prvom dijelu rada opisane su evolucijske strategije, jedna vrsta evolucijskih algoritama. Zatim je opisan problem trgovačkog putnika, te zbog čega taj kombinatorni problem predstavlja toliki izazov. U zadnjem dijelu slijedi praktični rad u okviru kojeg je napravljena aplikacija koja pokušava naći najbolje rješenje za problem trgovačkog putnika pomoću evolucijskih strategija. Osim objašnjenja načina izrade aplikacije, prikazani su rezultati eksperimenata u rješavanju problema trgovačkog putnika.

2. Evolucijske strategije

Evolucijske strategije su jedna od tehnika optimizacije iz područja evolucijskih algoritama. Razvoj evolucijskih strategija započeli su Ingo Rechenberg i Hans Peter Schwefel, studenti berlinskog Tehničkog fakulteta 60-ih godina 20. stoljeća. Oni su razvili prve sheme za razvoj optimalnih oblika tijela s minimalnim trenjem u aerodinamici koristeći Darwinovu teoriju evolucije.

Princip rada evolucijskih strategija kao i evolucijskih algoritama općenito, temelji se na Darwinovoj teoriji evolucije. U populaciji jedinki, bolje jedinke opstaju i šire svoj genetski materijal u sljedeće generacije. Svaka jedinka predstavlja jedno potencijalno rješenje problema. Tijekom izmjena generacija, dolazi se do sve boljeg rješenja, korištenjem operatora križanja i mutacije.

2.1 Prikaz rješenja

Populacija se sastoji od određenog broja jedinki. U daljnjem tekstu oznaka VEL_POP označavat će veličinu populacije, odnosno broj jedinki u populaciji. Svi podaci koji obilježavaju jednu jedinku zapisani su u jednom kromosomu. Podaci u kromosomu mogu biti pohranjeni na razne načine. Kromosom se može sastojati od niza bitova (binarni prikaz kromosoma), ili se može sastojati od realnog broja ili vektora realnih brojeva.

Za evolucijske strategije karakteristično je prikazivanje rješenja pomoću vektora realnih brojeva. Broj na određenom mjestu unutar vektora opisuje neku karakteristiku samog rješenja.

2.2 Funkcija dobrote

Funkcija dobrote (eng. *fitness*) ili funkcija ocjene kvalitete jedinke, ekvivalent je funkciji *f* koju treba optimizirati:

$$dobrota(v) = f(x), \tag{2.1}$$

gdje binarni vektor *v* predstavlja neki realan broj *x*.

Što je dobrota jedinke veća, jedinka ima veću vjerojatnost preživljavanja, a time i veću mogućnost prijenosa svojih gena na sljedeću populaciju. Tijekom procesa evolucije ukupna dobrota bi trebala biti sve bolja i bolja, ukoliko je zadani evolucijski algoritam dobar. Time bi se na samom kraju, prilikom odabira jedinki s najboljim vrijednostima funkcije dobrote, dobilo optimalno rješenje za neki problem. [1]

2.3 Selekcija

Uloga selekcije je čuvanje i prenošenje dobrih svojstava jedinki iz trenutne populacije na sljedeću generaciju jedinki. Njom se odabiru one jedinke koje će sudjelovati u reprodukciji u slijedećem koraku i time prenijeti svoj genetski materijal na sljedeću populaciju. Dakle, selekcija čuva dobre, a odbacuje loše jedinke iz populacijskog rješenja. Kako bi se odredilo

koje su jedinke kvalitetnije tj. koje će jedinke prijeći u sljedeću populaciju, koristi se funkcija dobrote.

2.3.1 Jednostavna selekcija

Cilj jednostavne selekcije je odabir roditelja čija je vjerojatnost selekcije proporcionalna njihovoj dobroti. Postupak je slijedeći:

- Ako *f(x)* poprima negativne vrijednosti, tada se dodaje pozitivna konstanta *C* tako da vrijedi *dobrota(v) = f(x) + C >0*, za svaki *x* iz domene.
- Izračunaju se sve dobrote jedinki u populaciji i ukupna dobrota populacije.
- Izračunaju se kumulativne dobrote q_k za svaki kromosom prema formuli (2.2) tako da vjerojatnost selekcije za svaki kromosom iznosi p_k:

$$q_k = \sum_{i=1}^{k} dobrota(v_i) \text{ , gdje je } k=1,2,\ldots,VEL\_POP \tag{2.2}$$

$$p(v_k) = \frac{dobrota(v_k)}{D} \tag{2.3}$$

$$D = \sum_{i=1}^{VEL\_POP} dobrota(v_i) \tag{2.4}$$

- Generira se slučajni realni broj *r* u intervalu *(0,D)* i potraži se *i-ti* kromosom za koji vrijedi da je $r \in (q_{i-1}, q_i)$.

Ovim postupkom selekcije jedan kromosom se može pojaviti više puta u sljedećoj generaciji. To je najveći nedostatak ove vrste selekcije.

2.3.2 Turnirska selekcija

Turnirska selekcija odabire *VEL_POP* puta *k* jedinki iz populacije jednakom vjerojatnošću. Zatim se u svakom koraku od odabranih *k* jedinki uzima najbolja jedinka i stavlja u bazen za reprodukciju. U eliminacijskoj turnirskoj selekciji, odabire se *k* jedinki, najlošija jedinka se briše i zamjenjuje djetetom koje će nastati križanjem dviju jedinki od *k-1* preostalih jedinki.

2.3.3 Eliminacijska selekcija

Eliminacijska selekcija bira *loše* kromosome koje treba eliminirati i reprodukcijom ih zamijeniti novima. Dakle, loši kromosomi umiru, a njih zamjenjuju djeca nastala reprodukcijom roditelja, tj. preživjelih kromosoma.

Vjerojatnost selekcije jedinke je to veća što je dobrota manja. Umjesto funkcije dobrote treba definirati funkciju kazne:

$$kazna(v_k) = \max_i \{dobrota(v_i)\} - dobrota(v_k) \tag{2.5}$$

Najbolja jedinka će na ovaj način imati vjerojatnost selekcije jednaku nuli, čime se osigurava opstanak najbolje jedinke (elitizam).

Jednom odabrani kromosom se više ne može odabrati. U svakom koraku potrebno je izračunavati kumulativnu dobrotu i ukupnu dobrotu populacije, jer jednom odabrani kromosom za eliminaciju ne može više doprinositi svojom dobrotom kumulativnoj dobroti.

Ovaj način selekcije je veoma brz i daje dobre rezultate. [2]

2.4 Genetski operatori

Genetski operatori se primjenjuju u reprodukciji jedinki koje su preživjele proces selekcije.

2.4.1 Križanje

Križanje je proces kojim iz dvije jedinke (roditelji) nastaje nova jedinka (dijete). Najvažnija karakteristika križanja jest da djeca nasljeđuju svojstva svojih roditelja. Križanje nije česta pojava kod evolucijskih strategija.

Križanje se definira proizvoljnim brojem prekidnih točaka. Ovisno o izboru tih točaka postoji nekoliko vrsta križanja:

- križanje u jednoj točki (eng. *one-point crossover*),
- križanje u dvije točke (eng. *two-point crossover*),
- križanje rezanjem i spajanjem (eng. *cut and splice crossover*),
- uniformno križanje (eng. *uniform crossover*) i
- polu-uniformno križanje (eng. *half-uniform crossover*).

Metode križanja korištene u ovom radu opisane su u poglavlju *4.1.5 Operatori križanja*.

2.4.2 Mutacija

Mutacija je operator karakterističan za proces rekombinacije kod evolucijskih strategija. Ovaj operator je unarni operator jer djeluje samo nad jednom jedinkom. Mutacija je slučajna promjena jednog ili više gena kako bi se dobila genetska raznolikost sljedeće generacije rješenja. Mutacija sprječava da neka rješenja unutar populacije postanu slična drugima i na taj način uspore ili potpuno zaustave proces evolucije. Mutacija kod evolucijskih strategija najčešće mijenja element x_i vektora x u broj izabran iz normalne razdiobe $N(x_i, \sigma_i^2)$.

Kod evolucijskih strategija postoji važno pravilo koje je definirao Rechenberg prilikom svojih istraživanja. Pravilo se naziva pravilo 1/5 uspjeha. Samo pravilo predviđa optimalne performanse za vrijeme trajanja mutacija i to kada od svih mutacija koje se provedu 20% njih daje uspješne potomke. To znači da kvocijent broja uspješnih mutacija i ukupnog broja mutacija unutar neke populacije mora biti približno 1/5. Ukoliko je taj kvocijent manji od 1/5, vrijednost parametra σ u normalnoj razdiobi se mora smanjiti. Nasuprot tome, ukoliko je kvocijent veći od 1/5, vrijednost parametra σ se mora povećati.

Mutacijom se pretražuje prostor rješenja što daje mutaciji jednu od najvažnijih osobina. Naime, ovim postupkom se omogućava izbjegavanje lokalnih minimuma. Ako cijela populacija završi u nekom lokalnom minimumu, mutacija će slučajnim pretraživanjem prostora (izborom elementa pomoću normalne razdiobe) pronaći bolje rješenje.

2.5 Vrste evolucijskih strategija

Pretpostavimo da je broj roditelja u nekoj generaciji γ označen sa μ, a broj potomaka u generaciji γ označen sa λ. Postoji sedam različitih tipova evolucijskih procesa koje koriste evolucijske strategije.

2.5.1 (1+1)-ES

U populaciji postoje samo dvije jedinke. Jedna jedinka je roditelj iz kojeg nakon reprodukcije procesom mutacije nastaje potomak. Proces selekcije se obavlja između te dvije jedinke, a u sljedeću generaciju ide bolja jedinka, tj. koja ima bolji faktor dobrote.

2.5.2 (μ +1)-ES

Populacija se sastoji od μ jedinki roditelja. Mutacijom jedne jedinke nastaje jedan potomak koji se konstantno reproducira. Iz spojenog seta potomaka izabrane jedinke i trenutne populacije odbacuje se jedinka koja ima najmanji faktor dobrote.

2.5.3 (μ + λ)-ES

U ovom slučaju iz μ jedinki roditelja nastaje λ potomaka procesom mutacije. Uvjet za ovaj proces je da je nastalo više djece nego što je roditelja, dakle λ>μ. Svaki od λ potomaka ima svoj faktor dobrote, kao što imaju i svi roditelji. Najboljih μ jedinki iz skupa roditelja i potomaka zajedno prelazi u sljedeću generaciju.

2.5.4 (μ , λ)-ES

Populacija se sastoji od μ jedinki roditelja, koji procesom mutacije daju λ potomaka, s time da vrijedi λ>μ. Svaki od λ potomaka ima svoj faktor dobrote. Za razliku od prethodnog slučaja, ovdje se za prijelaz u novu generaciju promatraju samo potomci, dakle roditelji ne ulaze u izbor. Iz λ potomaka izabire se μ najboljih jedinki koje prelaze u sljedeću generaciju.

2.5.5 (μ/ρ , λ)-ES

Ova strategija je (μ, λ) strategija uz dodatak parametra ρ. Ovaj parametar označava broj jedinki roditelja koji se upotrebljava u procesu reprodukcije. Ukoliko je ρ=1, ova strategija je analogna strategiji (μ, λ)-ES, tj. prilikom reprodukcije koristi se samo jedna jedinka iz populacije roditelja, što znači da se upotrebljava operator mutacije. Ukoliko je ρ=2, prilikom reprodukcije se koriste dvije jedinke iz populacije roditelja, što znači da se upotrebljava operator rekombinacije. Selekcija je analogna selekciji kod (μ , λ)-ES.

2.5.6 (μ/ρ + λ)-ES

Ova strategija je (μ + λ) strategija uz ponovni dodatak parametra ρ. Za reprodukciju vrijede ista pravila kao i kod (μ/ρ , λ) strategije, a selekcija je analogna selekciji kod (μ + λ)-ES.

2.5.7 $(\mu', \lambda'(\mu, \lambda)^{\gamma})$-ES

Kod ove strategije se iz populacije roditelja veličine μ' kreira λ' potomaka i izolira na γ generacija. U svakoj od γ generacija stvara se λ potomaka od kojih samo μ najboljih prelazi u sljedeću generaciju. Nakon γ generacija izabiru se najbolje jedinke od γ izoliranih populacija i krug kreće ponovno sa λ' novih jedinki potomaka.

2.6 Algoritam evolucijskih strategija

Cilj evolucijske strategije je stvoriti populaciju potomaka uz pomoć procesa rekombinacije ili mutacije iz početne populacije roditelja. Postupak mutacije i rekombinacije se ponavlja i nastaju nove generacije potomaka, sve dok se ne dođe do optimuma nekog zadanog problema. Ovaj proces prikazan je pseudokodom na slici 2.1.

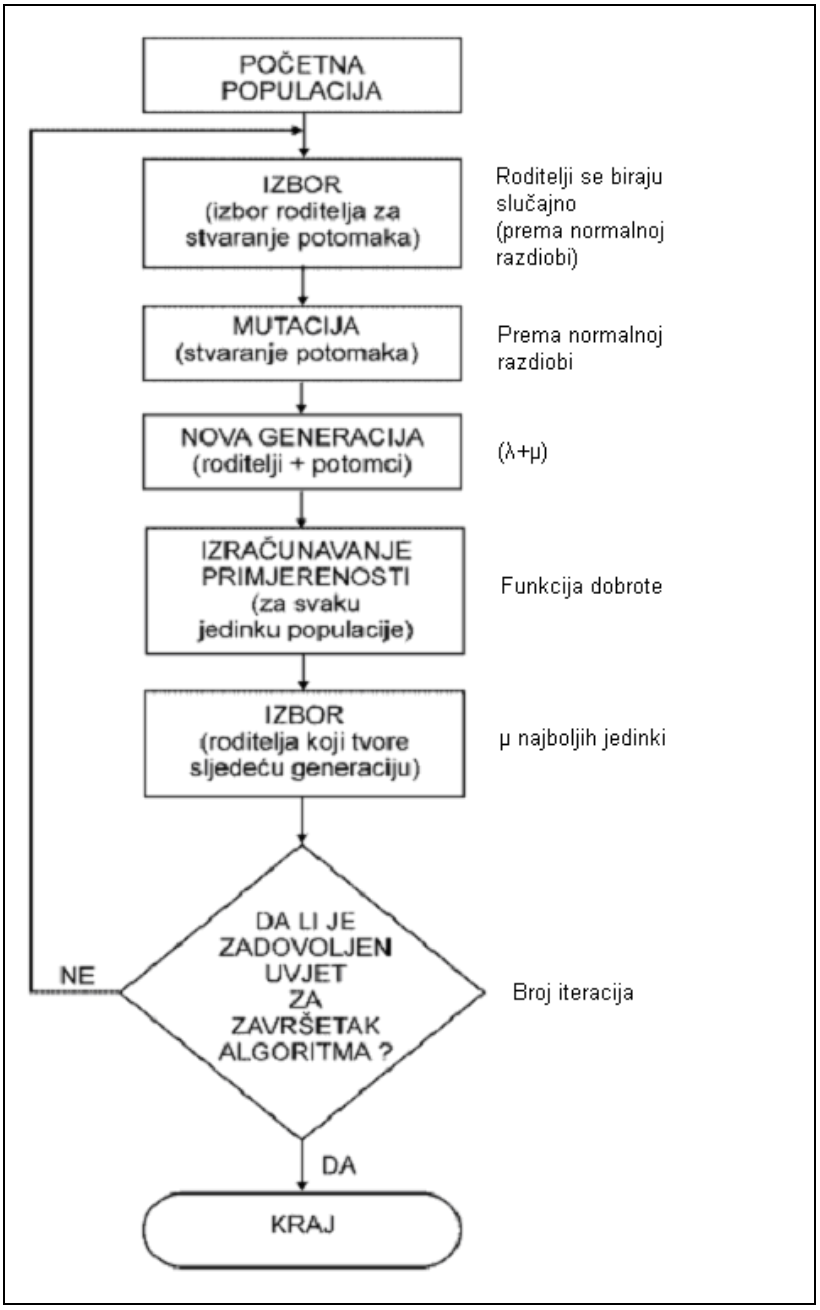
```
Evolucijska strategija{
      t = 0;
      generiraj početnu populaciju P(0);
      evaluiraj P(0); //provjeri dobrotu inicijalne populacije
      sve dok nije zadovoljen uvjet završetka evolucijskog procesa {
            izaberi najboljih μ roditelja iz P(t) i stavi ih u Pp(t);
            iz Pp(t) reproduciraj λ potomaka i stavi ih u Pc(t);
            mutiraj Pc(t);
            evaluiraj Pc(t);
            ako se koristi plus strategija P(t+1) = Pc(t) u P(t);
                  inače P(t+1) = Pc(t);
            t=t+1;
      }
}
```

Slika 2.1 *Pseudokod procesa evolucijske strategije*

Proces stvaranja nove generacije može se opisati kroz slijedeće korake:

1. stvori λ novih jedinki:
    a) odaberi slučajnih $\rho \leq \mu$ roditelja
    b) križaj ρ roditelja kako bi dobio potomka
    c) odredi novu vrijednost parametra p prema normalnoj razdiobi
    d) mutiraj dijete pomoću nove vrijednosti parametra p
2. odaberi novu populaciju na temelju funkcije dobrote:
    a) iz populacije djece ukoliko se radi o (λ, μ)-ES
    b) iz populacije djece i roditelja ukoliko se radi o $(\lambda + \mu)$-ES.

Dijagram tijeka evolucijske strategije $(\lambda + \mu)$ prikazan je na slici 2.2.



Slika 2.2 *Dijagram tijeka (μ+λ) evolucijske strategije*

2.7 Primjena evolucijskih strategija

Evolucijske strategije imaju vrlo široku primjenu u rješavanju raznih optimizacijskih problema, uključujući optimizacijske probleme u kontinuiranom i diskretnom vremenu, vektorsku optimizaciju i kombinatorne probleme sa i bez ograničenja. Koriste se u *data miningu*, u izradi kompliciranih vremenskih rasporeda, u kemiji, za nalaženje izvora emisije iz atmosferskih promatranja, za geometriju (npr. kod izrade mostova), rekonstrukciju površina, za rješavanje raznih verzija problema trgovačkog putnika, u optici i obradi slike, u aproksimaciji polinomnih funkcija, itd... [1]

3. Problem trgovačkog putnika

3.1 Opis problema trgovačkog putnika

Problem trgovačkog putnika očituje se u potrazi za najkraćim putem koji putnik mora prijeći tako da, krenuvši od početnog grada N1, obiđe sve zadane gradove točno jednom i ponovno se vrati u grad N1. Matematička definicija bi bila:

> Neka je zadan cijeli broj $n \geq 3$ i $n \times n$ matrica $C = (c_{ij})$, gdje je c_{ij} poprima nenegativne cjelobrojne vrijednosti. Traži se koja permutacija π cijelih brojeva od 1 do n minimizira sumu $\sum_{i=1}^{n} c_{i\pi(i)}$. [1]

Kroz godine ovaj problem je okupirao umove istraživača zbog mnogih razloga. Problem trgovačkog putnika je vrlo lako opisati, ali predstavlja veliki problem prilikom rješavanja. Problem trgovačkog putnika je teško riješiti jer je faktorjelne složenosti. Najkraći put između n gradova se nalazi negdje unutar prostora stanja koji je velik $\frac{(n-1)!}{2}$. Prostor stanja za problem od 10 gradova ima *181 440* mogućih rješenja. Za problem od 100 gradova broj potencijalnih rješenja je čak *4.66e155* .Nije poznat niti jedan vremenski polinomni algoritam kojim se on može riješiti. Osim što predstavlja veliki izazov za rješavanje, problem trgovačkog putnika je široko primjenjiv na različite probleme usmjeravanja i raspoređivanja.

4. Programsko ostvarenje

4.1 Evolucijske strategije za rješavanje problema trgovačkog putnika

Zbog faktorijelnog rasta prostora rješenja s porastom broja gradova, metoda evolucijskih strategija mogla bi dati dobre rezultate u rješavanju problema trgovačkog putnika. Uz pomoć evolucijskih strategija nije potrebno pretraživati cijeli prostor rješenja, nego se evolucijom iz generacije u generaciju algoritam sve više približava optimalnom rješenju.

4.1.1 Prikaz kromosoma

Zbog prirode problema, odabran je intuitivni prikaz kromosoma koji se sastoji od redoslijeda obilazaka gradova. Dakle, za problem od 9 gradova jedan kromosom bi mogao izgledati ovako:

(1, 4, 0, 7, 8, 5, 3, 6, 2)

Iz ovog prikaza može se vidjeti da je redoslijed obilazaka slijedeći: iz grada 1 ide se u grad 4, iz grada 4 u grad 0, zatim u grad 7, itd.

U programu je za prikaz kromosoma korištena klasa Kromosom.cs, čiji se kod može vidjeti u Dodatku B.

4.1.2 Funkcija dobrote

Funkcija dobrote jednaka je sumi udaljenosti između gradova u pojedinoj ruti, odnosno kromosomu. Što je duljina puta kraća, jedinka ima veću vjerojatnost preživljavanja.

4.1.3 Operator selekcije

U algoritmu se upotrebljava posebna inačica jednostavne selekcije, opisane ranije. Razlika je u tome što se vjerojatnosti selekcije pojedinih jedinki određuju prema funkciji normalne razdiobe $N(x_i, \sigma_i^2)$, gdje x_i predstavlja redni broj jedinke u populaciji. Osim toga, vrijednosti vjerojatnosti su skalirane tako da se slučajno odabire decimalni broj iz intervala [0,1].

4.1.4 Operatori mutacije

U algoritmu su korištene tri vrste mutacija.

Jednostavna mutacija

Slučajnim odabirom odabiru se dva različita grada u kromosomu nakon čega se obavlja njihova zamjena. Npr.:

*(2 **5** 1 3 **8** 4 6 7)* postaje *(2 **8** 1 3 **5** 4 6 7)*.

Mutacija gaussovom razdiobom

Ideja mutacije je slijedeća: slučajnim odabirom odabrati jedan grad, a drugi grad s kojim će se ovaj zamijeniti izabrati tako da veću vjerojatnost odabira ima grad koji se u kromosomu nalazi na poziciji bližoj prvom odabranom gradu.

Postupak je dan u točkama koje slijede:

1. Prije početka izvođenja algoritma radi se matrica vjerojatnosti *A* (formula (4.3))
2. Slučajnim odabirom izabire se jedan grad u kromosomu i pamti se njegova pozicija
3. Odredi se za koliko se maksimalno mjesta u kromosomu može pomaknuti od pozicije prvog odabranog grada, odnosno traži se:

$$Udaljenost = \max\{POG - 1, brojGradova - POG\}, \tag{4.1}$$

   gdje je POG pozicija prvog odabranog grada.

4. Računa se koji redak matrice će biti potreban za određivanje drugog grada prema slijedećoj formuli:

$$redak = Udaljenost - \left(\left\lceil \frac{brojGradova}{2} \right\rceil - 1\right) \tag{4.2}$$

5. Slučajnim odabirom generira se broj q iz intervala $\langle 0,1]$, i traži se za koji stupac, odnosno *j* vrijedi: $q \in \langle a_{redak,j-1}, a_{redak,j} \rangle$. Redni broj stupca, odnosno j označava za koliko mjesta u kromosomu se treba pomaknuti u lijevu ili desnu stranu kako bi se odabrao drugi grad. Ovim postupkom dobit će se dva moguća grada s kojima bi se prvi odabrani grad mogao zamijeniti. Napravit će se zamjene za oba slučaja, a na kraju će se uzeti rješenje s boljom dobrotom.

Kako bi se malo bolje objasnio postupak opisat ćemo način izrade matrice vjerojatnosti.

Recimo da neki problem ima 9 gradova. Nakon odabira prvog grada, drugi grad se može birati na poziciji udaljenoj za najviše 8 mjesta od prvog (to će vrijediti ako je prvi odabrani grad na jednoj od krajnjih pozicija). Prema maksimalnoj udaljenosti određuje se broj redaka matrice vjerojatnosti. Minimalna udaljenost bila bi kad bi prvi odabrani grad bio na poziciji točno u sredini kromosoma. Za slučaj od 9 gradova ta udaljenost iznosi 4. Dakle, maksimalne udaljenosti se ovisno o prvom odabranom gradu mogu kretati od iznosa 4 do 8. Za svaki iznos potreban je jedan redak u matrici, pa zaključujemo da za ovaj slučaj treba 5 redaka. Općenito, broj redaka u matrici će biti $\left\lceil \frac{brojGradova}{2} \right\rceil$.

Sad je poznato koliko redaka treba biti u matrici vjerojatnosti. Za prvi redak u matrici vrijedi da je maksimalna udaljenost između pozicija prvog i drugog grada najmanja. U ovom slučaju s 9 gradova ta udaljenost je 4. Za drugi redak će maksimalna udaljenost biti 5, za treći 6, za četvrti 7 i za peti redak 8. Kolika je maksimalna udaljenost određena za pojedini redak toliko će stupaca taj redak imati. Matrica će na kraju izgledati ovako:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & & & & \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & & & \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} & & \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} & a_{47} & \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} & a_{56} & a_{57} & a_{58} \end{bmatrix} \tag{4.3}$$

Na mjestu a_{ij} u matrici biti će kumulativna suma decimalnih brojeva dobivenih prema formuli za normalnu razdiobu (4.4):

$$N(x,\sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} \cdot \exp\left(\frac{-x^2}{2\sigma^2}\right), \tag{4.4}$$

gdje je varijabla x redni broj stupca matrice, odnosno *j*.

Sve vrijednosti u matrici *A*, skaliraju se tako da zadnji broj u svakom retku bude jednak 1. Time je dobivena matrica kumulativnih vjerojatnosti za odabir drugog grada. Na slici 4.1 prikazan je pseudokod opisanog postupka mutacije, a čitav kod može se vidjeti u dodatku B.

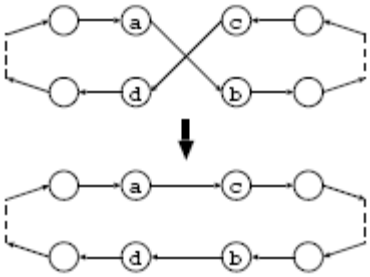
```
Kromosom GaussovaMutacija(Kromosom zaMutirati){
    int gen1 = slučajno odabrana pozicija u kromosomu;
    int d =Max(_brojGradova - gen1 – 1, gen1); //maksimalna udaljenost
    int gen2 = izbor_grada(d) + 1;
    ako ((gen2 + gen1) < _brojGradova){
        stvori dijete1 zamjenom gena na pozicijama gen1 i gen1+gen2;
        ako ((gen1 - gen2) >= 0){
            stvori dijete2 zamjenom gena na pozicijama gen1 i gen1-gen2;
            ako (dijete1.DuljinaPuta < dijete2.DuljinaPuta)
                vrati dijete1;
            inače vrati dijete2;
        }
        inače vrati dijete1;
    }
    inače{
        stvori dijete zamjenom gena na pozicijama gen1 i gen1-gen2;
        vrati dijete;
    }
}
private int izbor_grada(int grad1){
    broj=slučajni broj iz intervala [0,1];
    int m = veće cijelo (_brojGradova/2);
    int k=0,
    int i = m - (_brojGradova - grad1); //redni broj retka matrice
    dok vrijedi (broj > vjerMutacije[i][k])
        k++;
    vrati k;
}
```

Slika 4.1 *Pseudokod mutacije normalnom razdiobom*

2opt mutacija

Ova metoda je jedna od najpoznatijih metoda pretraživanja u problemu trgovačkog putnika. Algoritam je prikazan na slici 4.2, a može se opisati jednostavnim primjerom. Uzme se put od grada A do grada B, i put od grada C do grada D. Ako je nejednakost AB + CD > AC + BD istinita, dolazi do zamjene kako je prikazano na slici 4.2. Ovaj postupak se ponavlja za sve bridove dok god je moguće skratiti rutu. Velik problem 2opt metode je zapinjanje u lokalnom optimumu iz kojeg se na može izvući zbog načina usporedbe. Usprkos ovom nedostatku, ova metoda je najbolji operator mutacije. [3]

U dodatku B, prikazan je kod 2opt mutacije.



Slika 4.2 *2opt metoda*

4.1.5 Operatori križanja [3]

U strategijama (μ/ρ , λ)-ES i (μ/ρ + λ)-ES koristi se operator križanja. Vrste križanja koje su implementirane u algoritmu su slijedeće:

- PMX križanje (eng. *Partially Matched Crossover*)
- GX križanje (eng. *Greedy Crossover*)
- GSX križanje (eng. *Greedy Subtour Crossover*)

PMX križanje

U oba roditelja označe se dvije točke prekida. U primjeru na slici 4.3, točke prekida označene su znakom ' | '. Geni između tih točaka se zamijene između roditelja. Ostatak kromosoma se popunjava tako da se gradovi izvan točaka prekida vraćaju na svoje mjesto ukoliko već ne postoje kao rezultat zamjene. Ako grad već postoji na nekom mjestu između točaka prekida, umjesto njega se upisuje onaj grad kojeg je zamijenio taj novi grad.

2 5 1 | 3 8 4 | 7 6
8 6 7 | 2 4 1 | 3 5 → 3 5 8 | 2 4 1 | 7 6
1 6 7 | 3 8 4 | 2 5

Slika 4.3 *Primjer PMX križanja*

U primjeru na slici 4.3 geni između točaka prekida zamijene mjesta. Dakle, iz prvog roditelja 3 se zamijeni s 2, dok se u drugom roditelju 2 zamijeni s 3, i tako za ostala dva gena. Sada se puni ostatak prvog kromosoma. Na prvoj poziciji više ne može biti broj 2 jer se on već nalazi na poziciji 4, te se stoga, umjesto broja 2 upisuje broj koji je prije bio na poziciji 4, a to je broj 3. Na sljedećoj poziciji bio je broj 5. Taj broj nije iskorišten u zamjeni, pa ga se slobodno upiše natrag na poziciju dva. Za broj 1 je isti slučaj kao na poziciji jedan. Umjesto tog broja bi trebalo upisati broj 4, no i on je već upisan, te se umjesto njega upisuje 8. Nadalje, brojevi 7 i 6, nisu korišteni u zamjeni, te ih se upisuje na njihovo staro mjesto u kromosomu. Isti postupak se izvodi za drugi kromosom.

Kod PMX križanja može se također vidjeti u dodatku B.

GX križanje

Definicija *greedy crossover* križanja je: *Greedy crossover* uzima prvi grad iz jednog roditelja, uspoređuje gradove u koje se dolazi iz tog grada u oba roditelja i uzima onog čiji je put kraći. Ako se jedan grad u djetetu već pojavio onda se uzima drugi. Ako su se oba grada već pojavila, tada se slučajnim odabirom odabire jedan neodabrani grad. Postupak je vrlo jednostavan, a kod se nalazi u dodatku B.

GSX križanje

Ovo križanje radi tako da iz oba roditelja uzima što je moguće dulji dio genetskog materijala, tj. što dulji podskup gradova. Pseudokod je prikazan na slici 4.4.

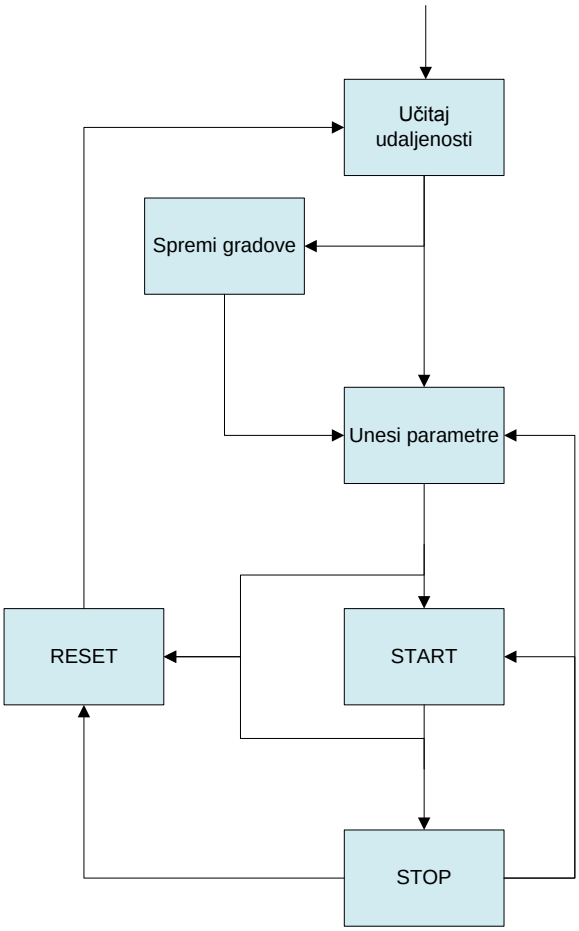
```
Ulaz: kromosom k = (a0,a1,..., an-1) i k = (b0,b1,..., bn-1)
        izlaz: dijete k
        procedura krizaj(ka, kb){
                fa <-true
                fb <- true
                izaberi jedan slucajan grad g
                izaberi x gdje je ax = t
                izaberi y gdje je by = t
                k <- t
                čini{
                      x = (x – 1) mod n
                      y = (y - 1) mod n
                      ako fa = true onda{
                            ako ax nije u k onda
                                  k <- ax * k
                            inače
                                  fa <- false
                      }
                      ako fb = true onda{
                            ako by nije u k onda
                                  k <- k * by
                            inače
                                  fb <- false
                      }
                } dok fa = true ili fa = true
                ako staza nije potpuna onda
                      dodaj ostale gradove slucajnim redoslijedom u k
                vrati k
            }
```

Slika 4.4 *Pseudokod GSX križanja*

4.2 Rad s aplikacijom

Aplikacija je izrađena u okruženju Microsoft Visual Studio 2005, u programskom jeziku C#.

Tijek rada aplikacije, prikazan je na slici 4.5.



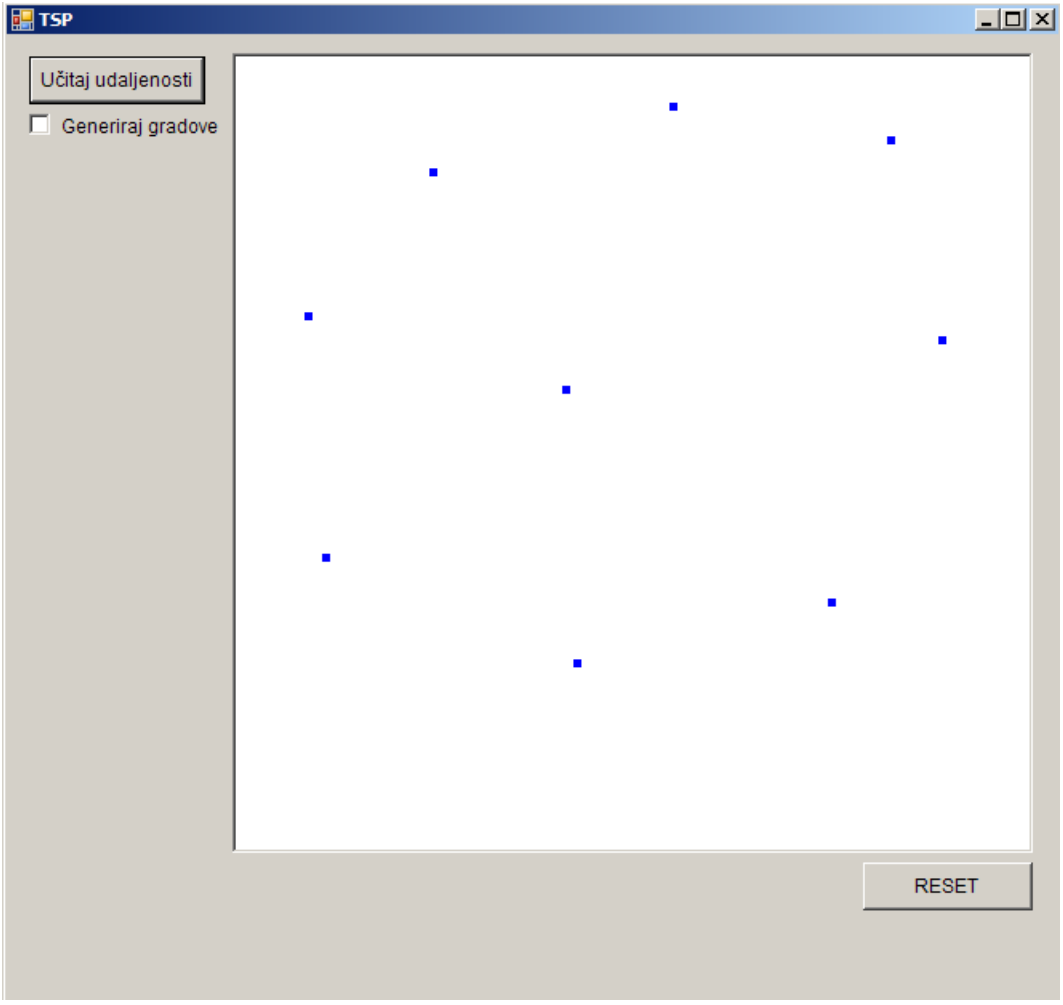
Slika 4.5 *Tijek izvođenja aplikacije*

Učitavanje udaljenosti moguće je na dva načina. Korisnik može učitati koordinate točaka iz datoteke. Pritiskom tipke *Učitaj udaljenosti*, otvorit će se dijalog za otvaranje datoteke. Zapis u datoteci mora biti formata prikazanog na slici 4.6. X_i i Y_i predstavljaju x, odnosno y koordinatu točke i. Koordinate x i y moraju biti cjelobrojne nenegativne vrijednosti, i u zapisu moraju biti razmaknute tabulatorom.

X_0	Y_0
X_1	Y_1
X_2	Y_2
X_3	Y_3
X_4	Y_4

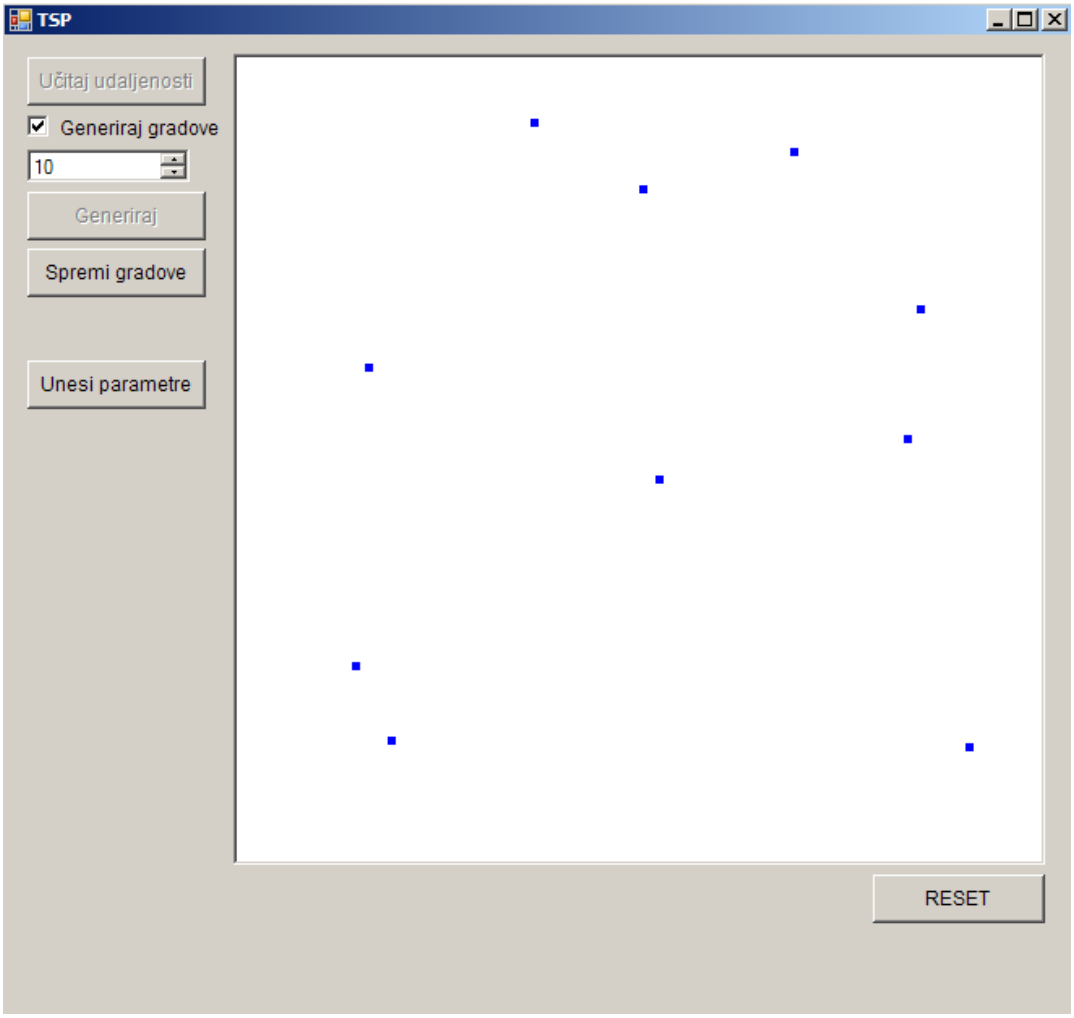
Slika 4.6 *Format zapisa točaka u datoteci*

Drugi način učitavanja točaka je postavljanje točaka na panelu za crtanje putem miša. Pritiskom tipke *Učitaj udaljenosti*, program učita koordinate postavljenih točaka.



Slika 4.7 *Izgled sučelja - unos točaka na panelu za crtanje*

Osim ovakvog načina zadavanja problema, korisnik može odabrati opciju da program samostalno generira problem zadan brojem gradova. Stavljanjem kvačice na *Generiraj gradove*, dobiva se opcija za odabir broja gradova. Pritiskom tipke *Generiraj* stvara se zadani broj točaka. Program će slučajnim odabirom koordinata unutar dimenzija panela za crtanje odabrati koordinate točaka i prikazati ih na panelu (slika 4.8).



Slika 4.8 *Sučelje nakon generiranja 10 točaka*

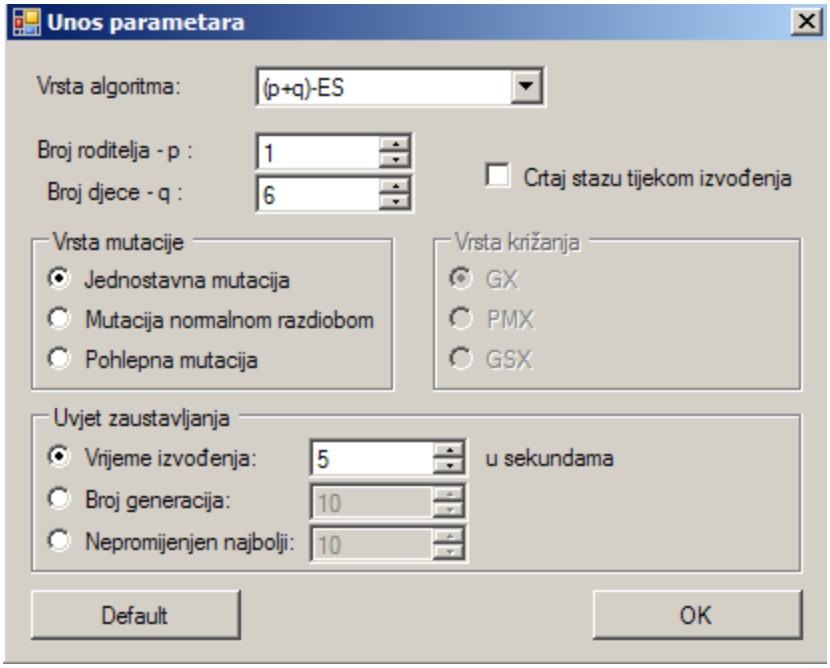
Za slučajeve učitavanja udaljenosti putem miša (preko panela) ili slučajnim generiranjem postoji mogućnost spremanja koordinata nastalih točaka, pritiskom tipke *Spremi gradove*.

Nakon učitavanja udaljenosti slijedi zadavanje parametara. Pritiskom tipke *Unesi parametre* otvara se dijalog za izbor parametara (slika 4.9). Moguće je odabrati postavke za slijedeće parametre:

1. Vrsta algoritma:
   - (1+1) – ES,
   - (μ, λ) – ES,
   - (μ + λ) – ES,
   - (μ + 1) – ES,
   - (μ/2, λ) – ES,
   - (μ/2 + λ) – ES.
2. Broj roditelja i djece – mogućnost unosa broja jedinki roditelja i djece ovisi o odabranom algoritmu
3. Vrsta mutacije:
   - Jednostavna mutacija,

- Mutacija s normalnom razdiobom,
- Pohlepna mutacija (*2opt mutacija*).

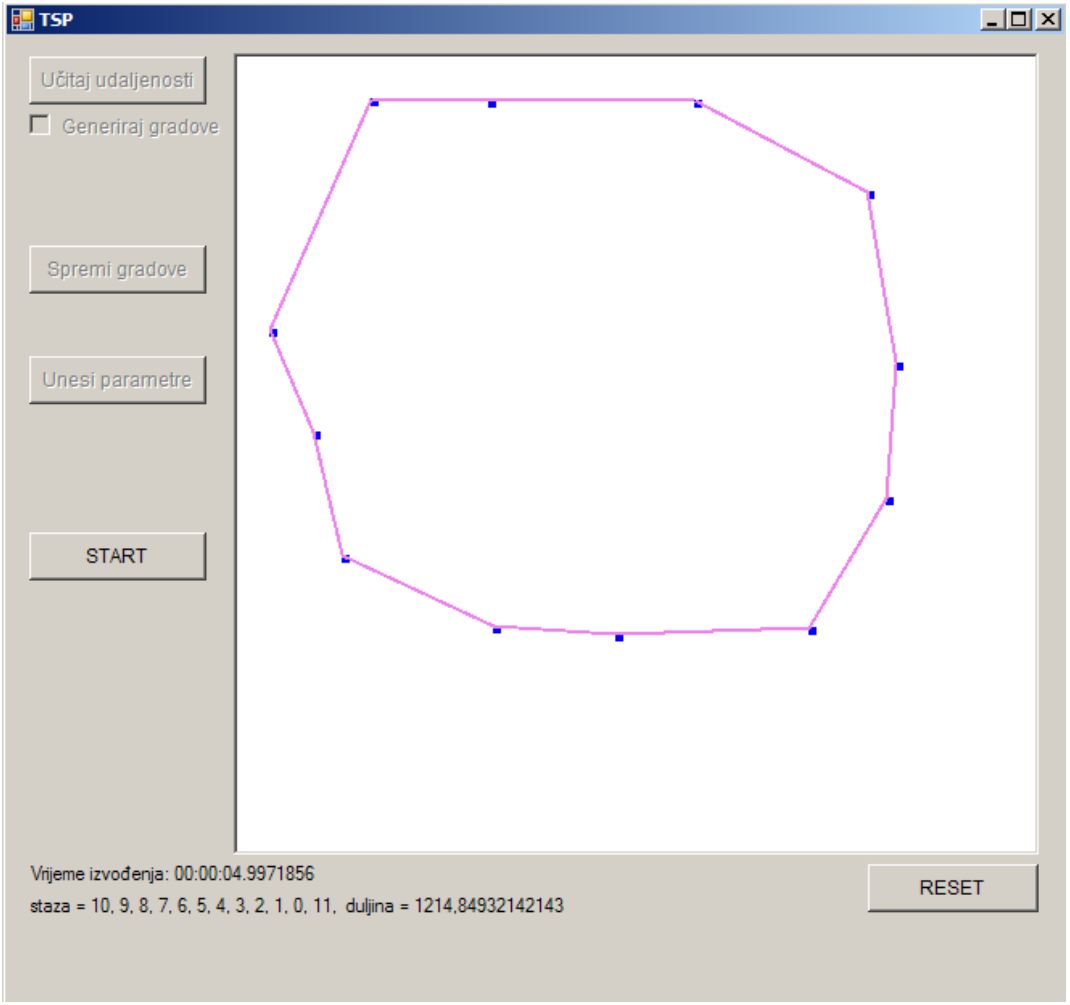
4. Vrsta križanja – za strategije (μ/2, λ) – ES, (μ/2 + λ) – ES:
   - *Greedy crossover* (GX),
   - *Partially matched crossover* (PMX),
   - *Greedy subtour crossover* (GSX).
5. Uvjet zaustavljanja:
   - Vremensko ograničenje,
   - Ukupni broj generacija,
   - Broj generacija s nepromijenjenom najboljom jedinkom.
6. Mogućnost crtanja ruta tijekom izvođenja programa – ne preporuča se za broj gradova veći od 50 (usporava program)



Slika 4.9 *Dijalog za unos parametara*

Pritiskom tipke *Default* postavljaju se predviđeni parametri. Nakon pritiska tipke *OK*, dijalog se zatvara i fokus se vraća na glavni prozor. Sada je moguće pokrenuti program pritiskom tipke *START*.

Tijekom izvođenja programa, u lijevom dijelu prozora ispod panela za crtanje ispisuje se broj generacije i dobrota najbolje jedinke u svakoj iteraciji. Nakon završetka izvođenja, ispisuje se vrijeme izvođenja, te najbolja ruta i njen put (slika 4.10). Najbolja ruta se zbog preglednosti ispisuje samo ako je broj gradova manji ili jednak 100.



Slika 4.10 *Izgled sučelja nakon završetka izvođenja*

Program se zaustavlja nakon što je zadovoljen uvjet zaustavljanja zadanog u parametrima, ili pritiskom tipke *STOP*. Program se nakon zaustavljanja može ponovno pokrenuti s istim parametrima ponovnim pritiskom tipke *START*. Program se može ponovo pokrenuti s istim problemom, ali različitim parametrima ako se ponovo pritisne tipka *Unesi parametre* i zatim tipka *START*. Pritiskom tipke *RESET*, parametri i točke na panelu se brišu, te je moguće zadati novi problem i unijeti nove parametre za ponovno pokretanje programa.

4.3 Rezultati eksperimentiranja

Računalo na kojem je testirana aplikacija je Intel Pentium IV 1.8 GHz s 512 MB RAMa.

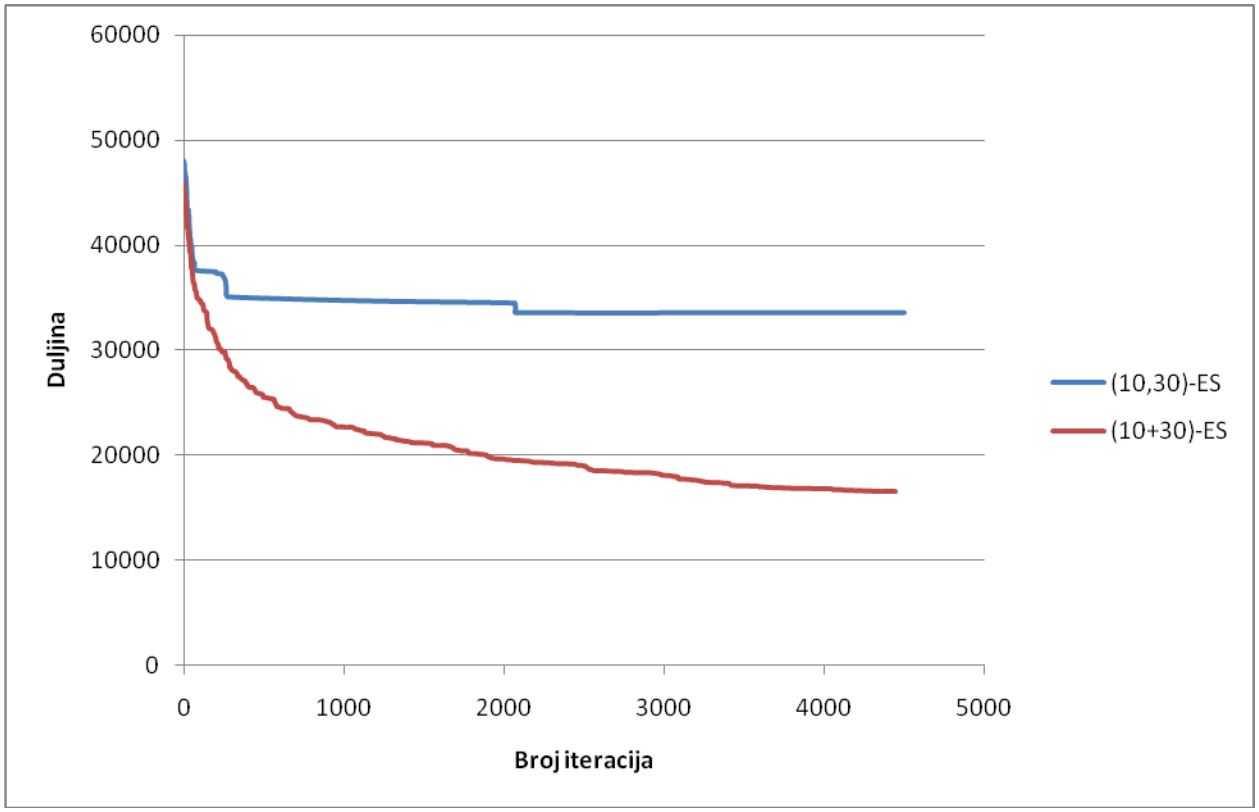
Eksperimenti su izvođeni na problemu od 200 gradova koji se nalazi na cd-u s izvornim kodom aplikacije. Za problem od 200 gradova postoji 1.97e+372 mogućih rješenja. Pretpostavimo da za stvaranje jedne staze od 200 gradova treba stotinka milisekunde. Tada bi za pretraživanje cjelokupnog prostora rješenja trebalo 6.25e+359 godina!

U daljnjem tekstu prikazani su rezultati kombinacija pojedinih strategija i genetskih operatora.

Sve vrste strategija su pokazale izvrsne rezultate ako se koristi 2opt mutacija, no prvo ćemo opisati kakve rezultate daju ostale vrste mutacija. Uz pomoć ostalih mutacija može se lakše uočiti razlika u radu pojedinih vrsta strategija.

4.3.1 Rezultati i usporedbe strategija koje od genetskih operatora koriste samo mutaciju

Od testiranih strategija koje koriste isključivo mutaciju, najbolja se pokazala (μ + λ) – ES, dok je najlošija bila (μ, λ) – ES. Uočena je velika razlika u brzini konvergencije k optimumu. Kod (μ, λ) strategije konvergencija je nakon vrlo kratkog vremena linearna i gotovo potpuno paralelna s x koordinatnom osi, dok je kod plus strategije konvergencija bliskija eksponencijalnoj funkciji, kako se i očekuje od evolucijskog algoritma (slika 4.11).



Slika 4.11 *Rad evolucijskih strategija (10,30)-ES i (10+30)-ES za problem od 200 gradova uz korištenje jednostavne mutacije*

(μ, λ) – ES

Kod (μ, λ) strategije se prilikom povećavanja broja jedinki uz konstantan omjer jedinki roditelja i djece poboljšava rješenje. Za sve veličine broja jedinki i vrste mutacija napravljeno je barem 10 eksperimenata i izračunata je srednja vrijednost rješenja. Uz korištenje jednostavne mutacije rješenje se poboljšavalo s porastom broja jedinki, dok se kod mutacije s normalnom razdiobom nisu dogodile veće promjene rješenja (Tablica 4.1).

Tablica 4.1 *Vrijednosti najkraćeg puta (μ,λ) strategije za TSP od 200 gradova i izvođenje od 5 minuta*

(μ,λ)–ES, uz μ:λ=1:2	Jednostavna mutacija (duljina / broj generacija)	Mutacija normalnom razdiobom (duljina / broj generacija)
(1,2)-ES	44 832 / 50 000	28 722 / 41 400
(4,8)-ES	41 376 / 17 700	27 272 / 13 800
(10,20)-ES	37 482 / 7900	27 871 / 6000
(20,40)-ES	33 983 / 4200	27 745 / 3100
(50,100)-ES	31 621 / 1700	27 706 / 1300

Najbolje rješenje za (μ, λ) strategiju i jednostavnu mutaciju je, dakle, 31 621, dok je stvarno optimalno rješenje 3184! Nešto bolji rezultati kod (μ, λ) strategije mogu se postići smanjenjem omjera μ : λ, odnosno povećanjem broja jedinki djece. Već uz omjer μ : λ = 1 : 3, jednostavna mutacija je davala bolje rezultate od mutacije normalnom razdiobom. Kako god mijenjali parametre (μ, λ) strategije, uz korištenje mutacije normalnom razdiobom, dobivene vrijednosti su uvijek bile između 26,000 i 29,000.

(μ + λ) – ES

Strategija (μ + λ) najbolje rezultate daje uz što manji broj jedinki. Promjena omjera jedinki roditelja i djece nije pokazala nikakve promjene u rezultatima. Za isti problem od 200 gradova i izvođenje od 5 minuta rješenja su se kretala između 11,000 i 14,000. Povećanje ukupnog broja jedinki rezultiralo je lošijim rezultatima, obično između 15,000 i 17,000. Eksperimenti su izvođeni do 50 jedinki roditelja i 200 jedinki djece za (μ + λ) strategiju, te do 100 jedinki roditelja i 400 jedinki djece kod (μ, λ) strategije.

(μ + 1) – ES

(μ+1) strategija se pokazala lošijom od (μ+λ) strategije. Vrijednosti dobivenih rješenja se pogoršavaju s povećanjem broja roditelja, odnosno μ (Tablica 4.2).

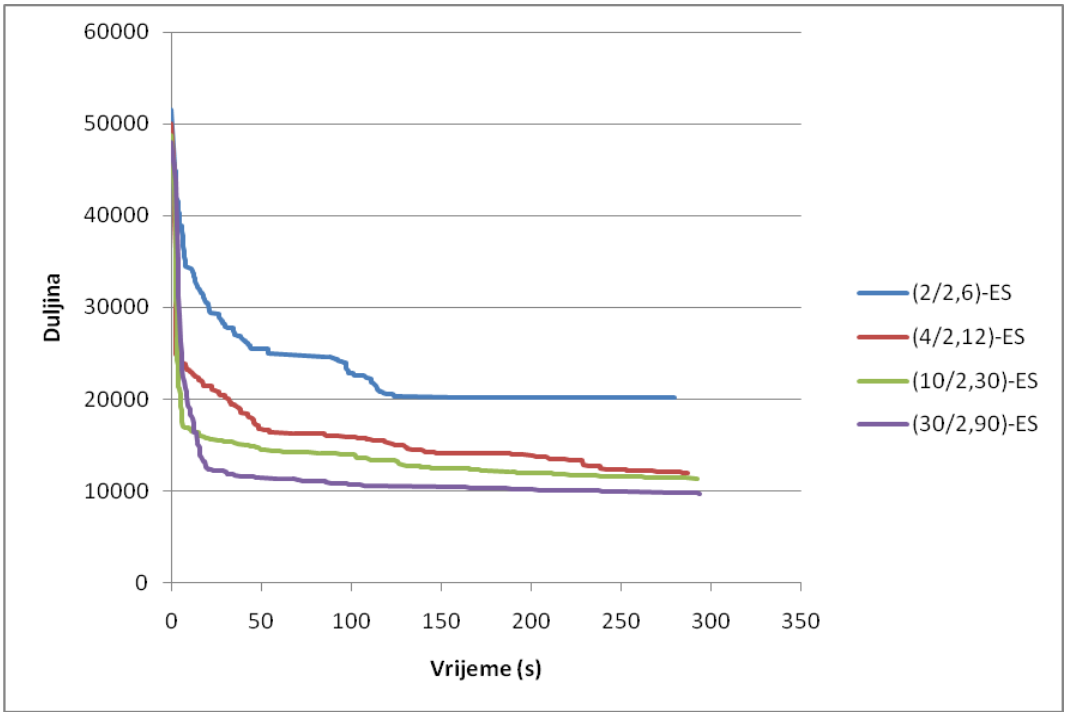
Tablica 4.2 *Rezultati (μ+1)-ES za TSP od 200 gradova uz korištenje jednostavne mutacije*

(μ+1)–ES	Duljina puta	Broj generacija
(1+1)–ES	13 211	57 000
(10+1)–ES	17 617	19 000
(20+1)–ES	22 543	11 000
(50+1)–ES	32 903	4900

4.3.2 Rezultati strategija koje koriste križanje

Od strategija koje koriste križanje najboljom se pokazala (μ/2, λ) strategija. Za problem od 200 gradova (30/2, 90) – ES, uz korištenje jednostavne mutacije i GX križanja, dobivao je rezultate između 8 000 i 10 000, što je najbolje od svih dosad opisanih strategija.

Konvergencija k optimalnom rješenju i postignuto rješenje nakon 5 minuta rada se poboljšavalo kod korištenja GX križanja i jednostavne mutacije, kako se povećavao broj jedinki uz jednak omjer broja jedinki roditelja i djece. Na slici 4.12, prikazano je poboljšanje učinkovitosti (μ/2, λ) strategije uz korištenje GX križanja i jednostavne mutacije, s omjerom μ:λ=1:3. Na apscisi je vrijeme u sekundama, a na ordinati udaljenost u tom trenutku najbolje rute.



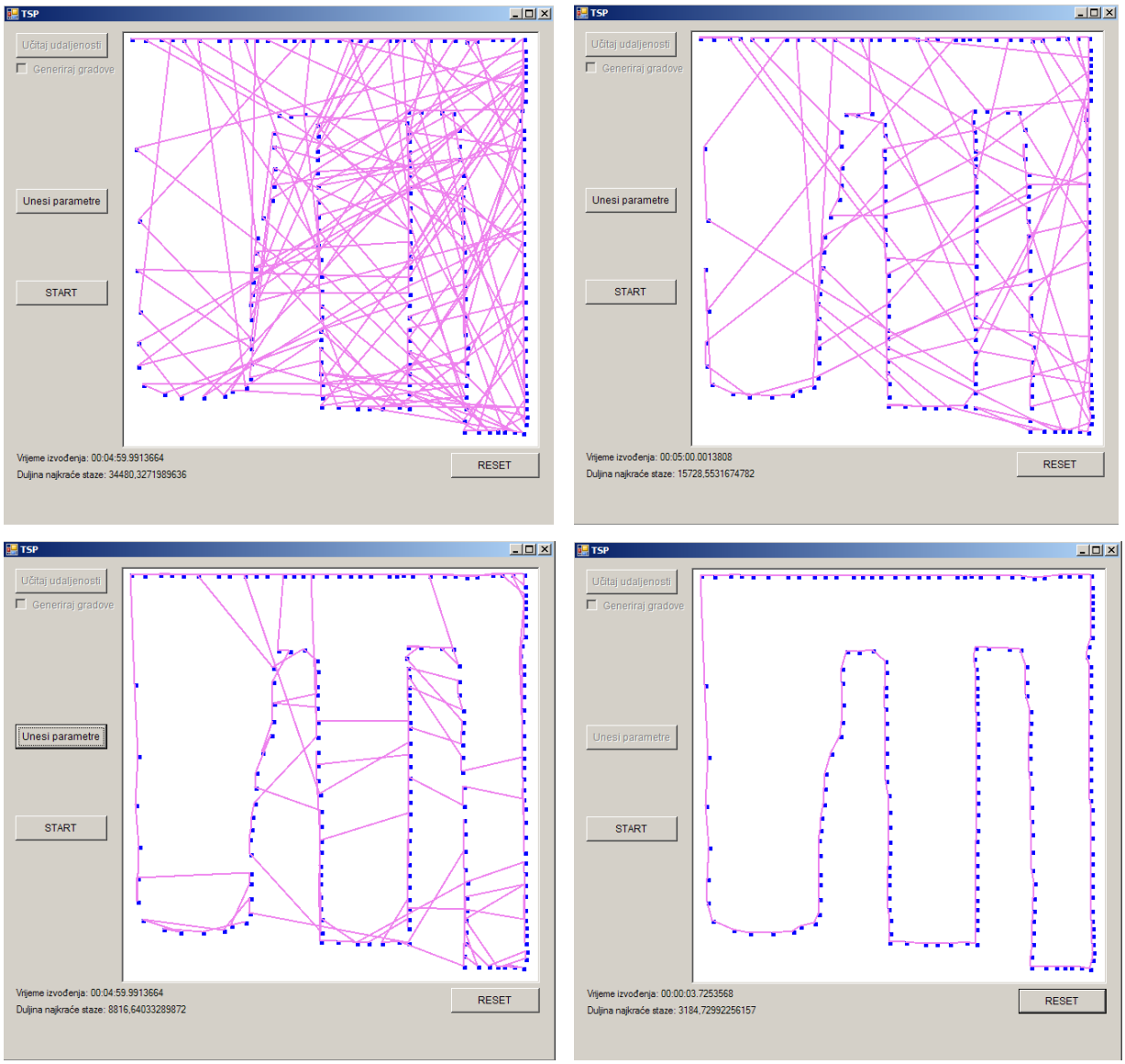
Slika 4.12 *Izvođenje (μ/2, λ) strategije s konstantnim omjerom μ:λ=1:3 s različitim brojem jedinki*

Za GX križanje i jednostavnu mutaciju rješenje je ovisilo o ukupnom broju jedinki kako kod zarez tako i kod plus strategije, dok kod svih ostalih kombinacija križanja i mutacija takva ovisnost nije bila izražena. Prosječni rezultati kombinacija križanja i mutacija dani su u tablici 4.3.

Tablica 4.3 *Prosječne vrijednosti duljine puta za problem od 200 gradova nakon 5 minuta izvođenja*

Strategija	GX, jednostavna mutacija	GX, mutacija normalnom razdiobom	PMX, jednostavna mutacija	PMX, mutacija normalnom razdiobom	GSX, jednostavna mutacija	GSX, mutacija normalnom razdiobom
(2/2,6)	19 755	30 035	15 549	33 255	14 974	24 697
(4/2,12)	14 503	23 394	15 680	33 413	14 513	-
(10/2,30)	10 360	-	15 981	-	15 157	-
(2/2+6)	14 001	27 288	15 436	34 603	14 685	23 949
(5/2+5)	12 482	24 079	15 576	-	-	-

Dosad navedene kombinacije parametara i vrsti strategija nisu davale dobre rezultate. Najbolji rezultat za problem od 200 gradova je dosad bio 8000, a najkraći put je zapravo duljine 3184! Na slici 4.12 prikazano je kako izgledaju rješenja nekih strategija, te kako izgleda stvarno točno rješenje.



Slika 4.13 *Izgled rješenja: lijevo gore – (10,30)-ES, jednostavna mutacija; desno gore – (10+30)-ES, jednostavna mutacija; lijevo dolje – (30/2,90)-ES, GX, jednostavna mutacija; desno dolje – (2+6)-ES, 2opt mutacija*

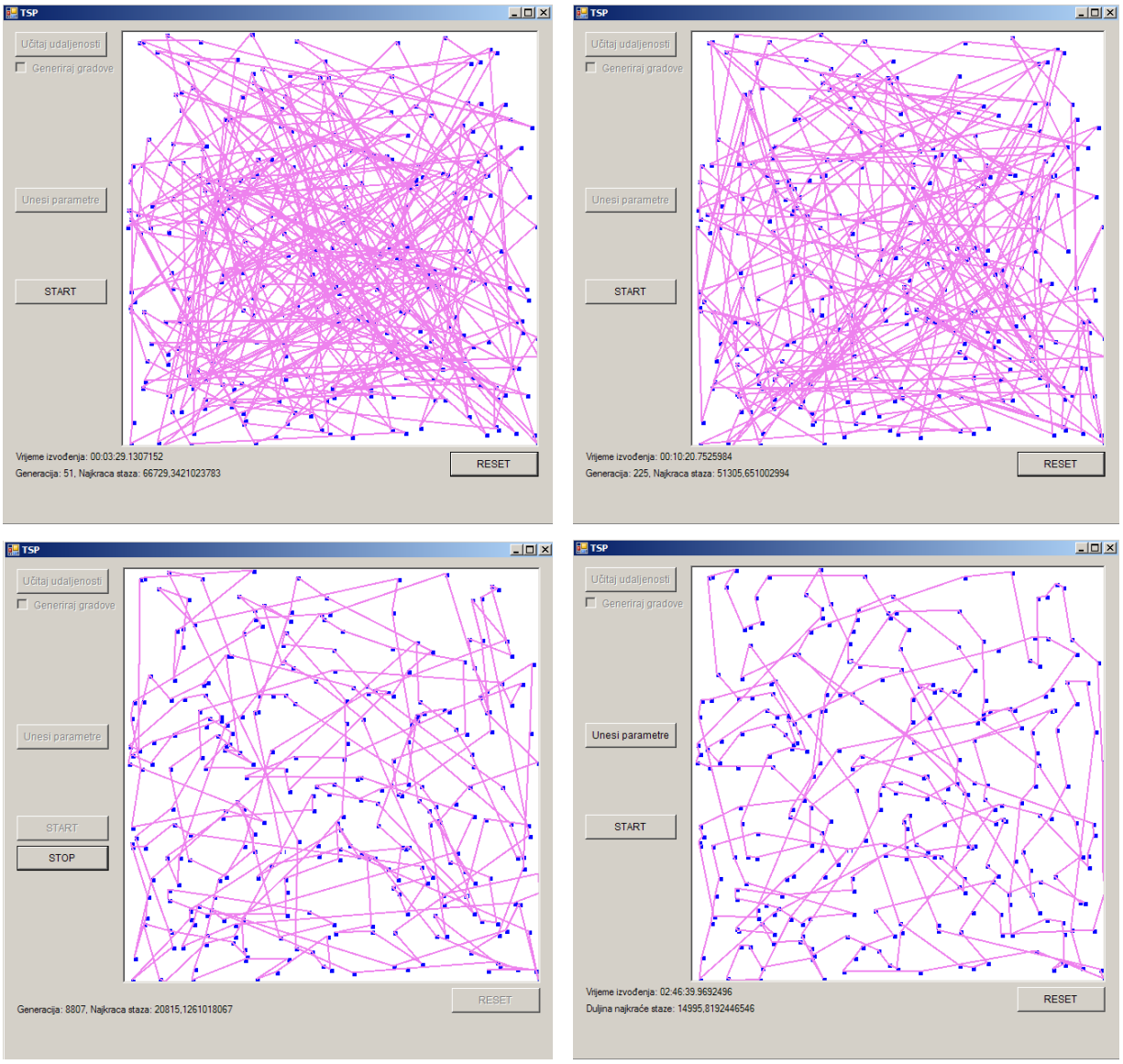
Najbolja metoda koja je gotovo uvijek došla do optimalnog rješenja je 2opt mutacija. Do točnog rješenja se uspjelo doći korištenjem mutacije 2opt i bilo koje vrste strategije, dok je god ukupni broj jedinki u populaciji veći od 2. Budući da 2opt metoda može zapeti u lokalnom minimumu, korištenje većeg broja jedinki umanjuje taj nedostatak. 2opt mutacija nalazi točno rješenje nakon samo 2 sekunde izvođenja, za problem od 200 gradova prikazan na slici 4.13.

Dakle, najbolje rezultate daje (μ+λ) strategija uz korištenje 2opt mutacije, uz uvjet da je ukupni broj jedinki u populaciji veći od 2, odnosno (μ+λ>2).

4.3.3 Rješavanje problema veličine 300 gradova

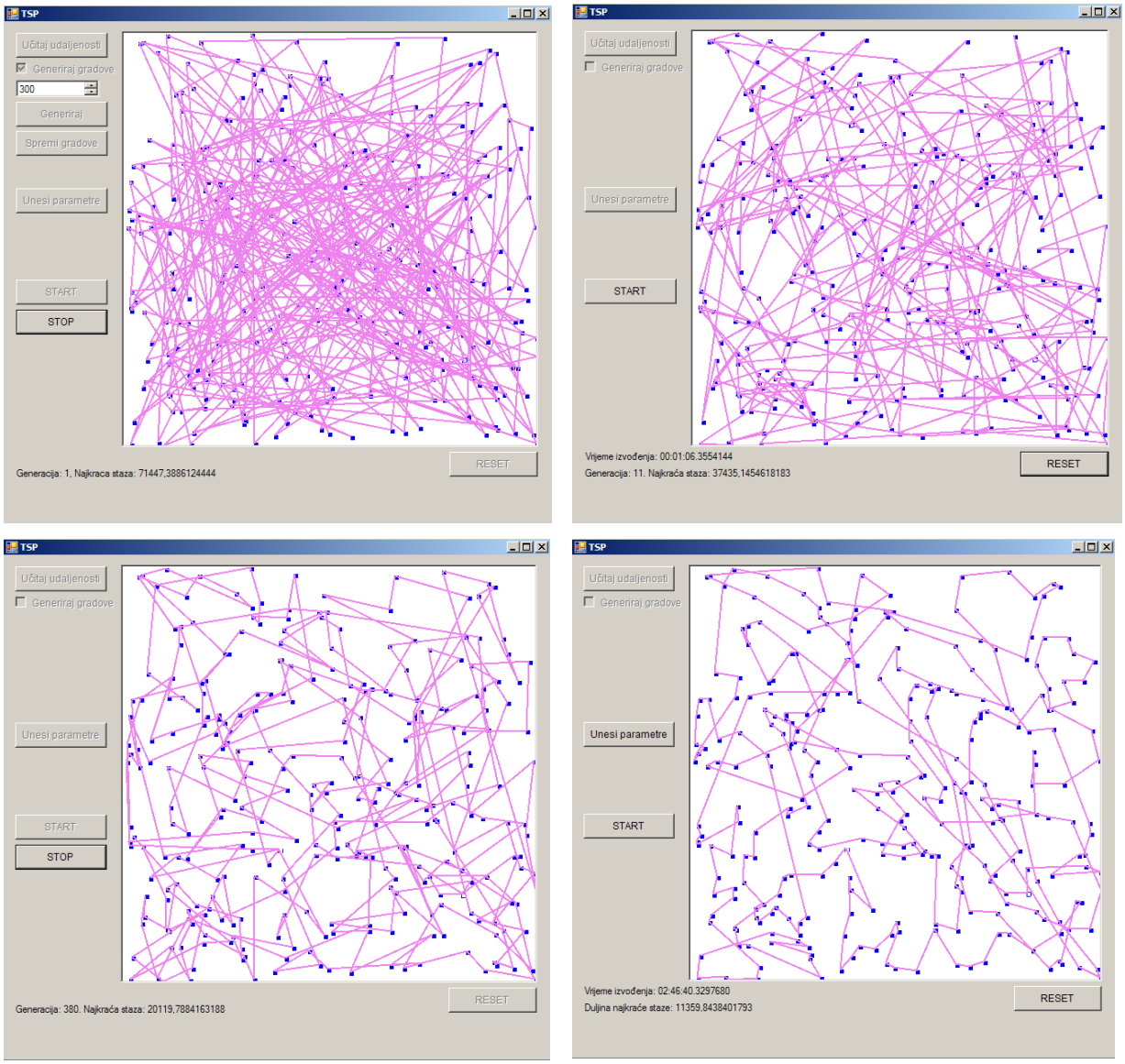
Osim eksperimenata s problemom od 200 gradova za koji je bilo poznato najbolje rješenje, napravljeno je nekoliko testova izvođenja sa slučajno generiranim problemom od 300 gradova.

Program je pokrenut s tri različite strategije i pušten je da radi 2 sata, 46 minuta i 40 sekundi. Na slici 4.14 prikazani su rezultati u pojedinim iteracijama za strategiju (5+15) i jednostavnu mutaciju.



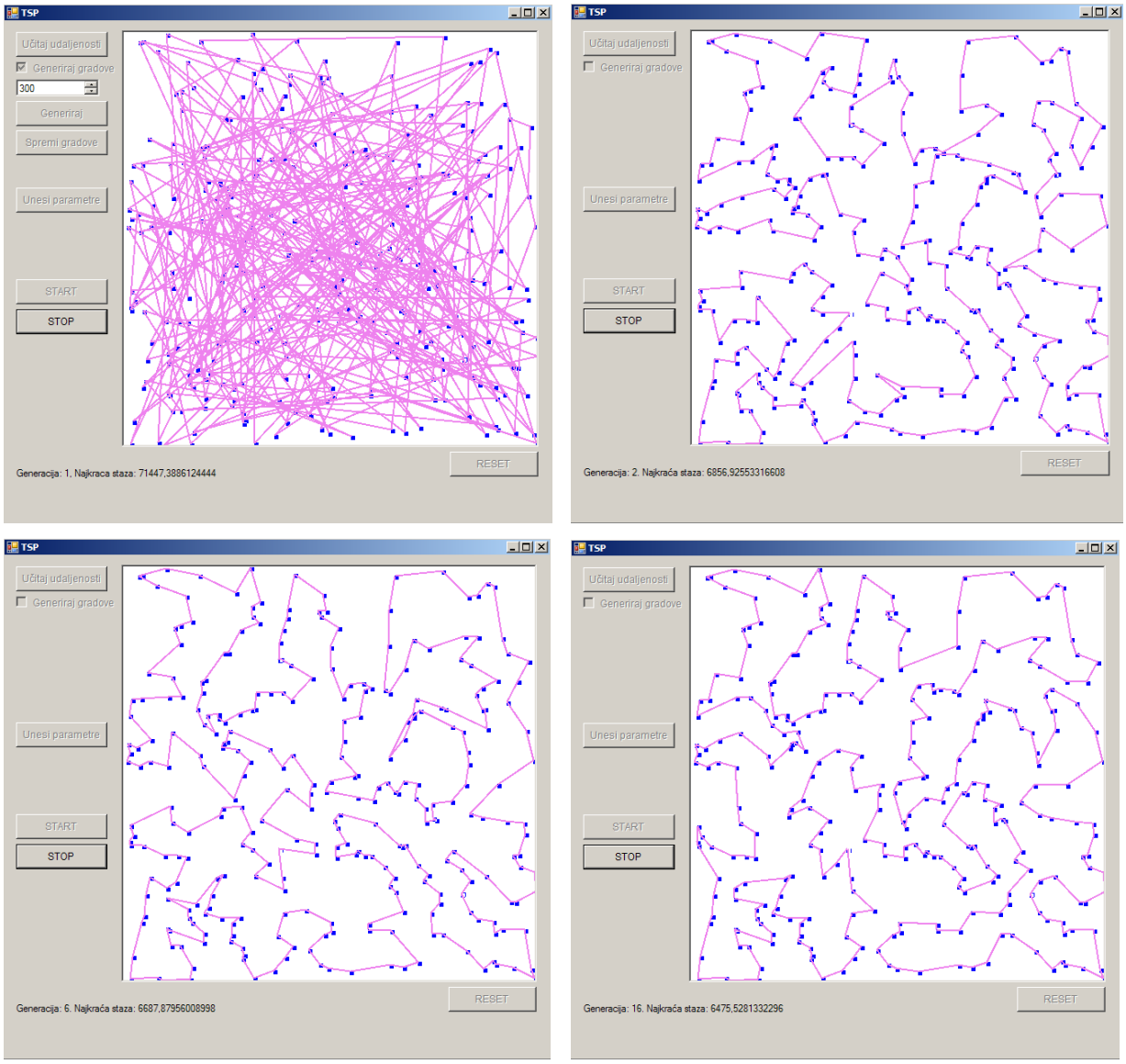
Slika 4.14 *Izvođenje (5+15)-ES uz jednostavnu mutaciju: gore lijevo – generacija 51, duljina 66729; gore desno – generacija 225, duljina 51305; dolje lijevo – generacija 8807, duljina 20815; dolje desno – generacija 11000, duljina 14995.*

Na slici 4.15 prikazani su rezultati istog problema uz korištenje (30/2,90) strategije, GX križanja i jednostavne mutacije.



Slika 4.15 *Izvođenje (30/2,90)-ES uz GX križanje i jednostavnu mutaciju: gore lijevo – inicijalno stanje, duljina 77000; gore desno – generacija 11, duljina 37435; dolje lijevo – generacija 380, duljina 20119; dolje desno – generacija 9000, duljina 11359.*

Strategija (30/2,90) je u kratkom vremenu prepolovila duljinu najkraće staze, dok je strategiji (5+15) za isti učinak trebalo puno više vremena i generacija.



Slika 4.16 *Izvođenje (30/2,90)-ES uz korištenje GX križanja i 2opt mutacije: gore lijevo – inicijalno stanje; gore desno – 2 generacija, duljina 6856; dolje lijevo – generacija 6, duljina 6687; dolje desno – generacija 16, duljina 6475.*

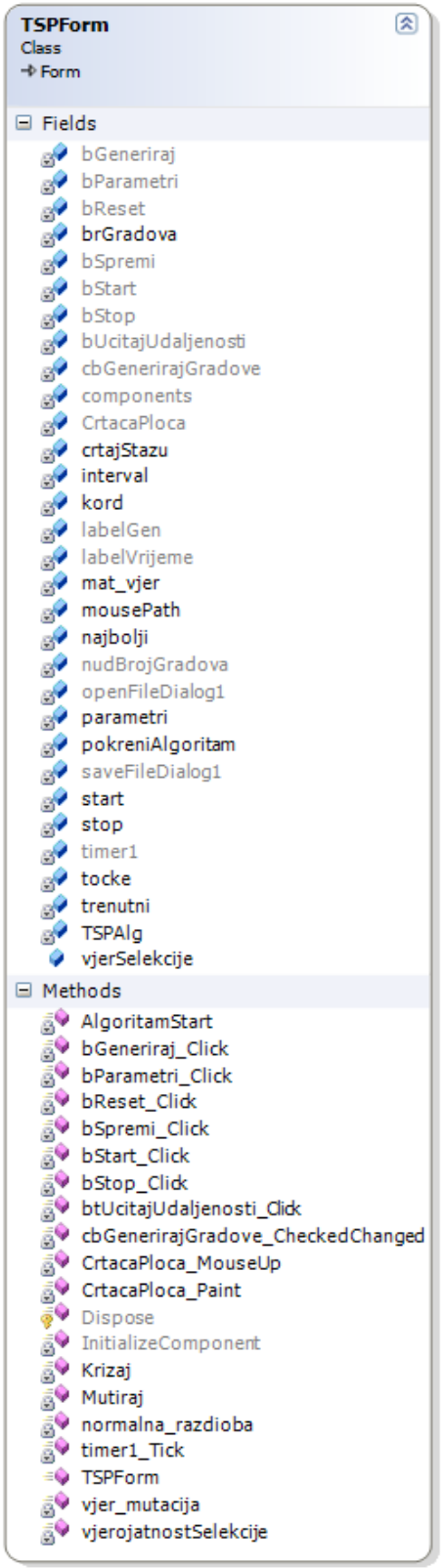
Slika 4.16 prikazuje rješavanje istog problema strategijom (30/2,90) uz korištenje GX križanja i 2opt mutacije. Kako se iz slika vidi, 2opt mutacija se opet pokazala najboljom. Već u drugoj iteraciji našla je rješenje koje je zadovoljavajuće blizu optimalnom rješenju. U prvih 16 iteracija na koje je utrošeno dvije minute vremena dale su rezultat duljine 6485. U preostalom izvođenju, algoritam nije našao bolje rješenje. Budući da je problem zadan slučajno, ne zna se je li nađeno rješenje uistinu optimalno ili je algoritam zapeo u lokalnom optimumu. Kako bilo, rezultat 6485 je daleko bolji od rezultata ostalih strategija.

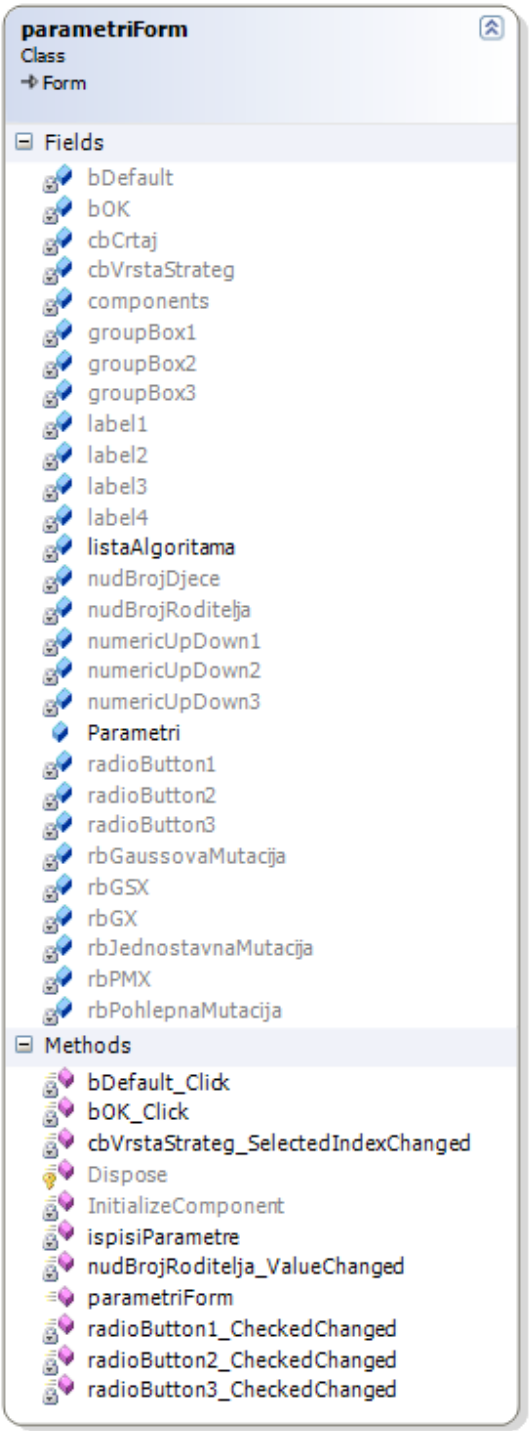
5. Zaključak

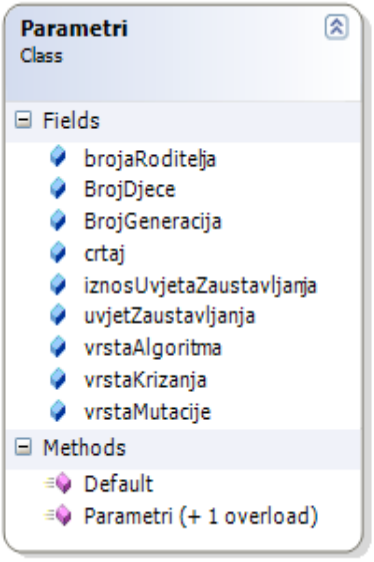
U radu su prikazane evolucijske strategije kao jedna od tehnika optimiranja. Dobiveni rezultati kod uporabe 2opt mutacije su više nego zadovoljavajući. Donekle zadovoljavajuće rezultate davala je (μ/2, λ) strategija uz korištenje GX križanja i jednostavne mutacije, te nešto većeg broja jedinki roditelja i djece.

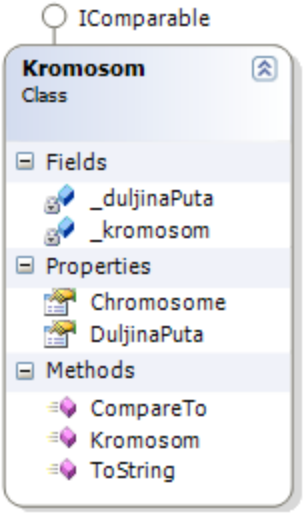
Eksperimenti su pokazali od kolike je važnosti ispravan odabir genetskih operatora i broja jedinki u populaciji. Pri odabiru broja jedinki u populaciji treba uzeti u obzir da što je više jedinki to će algoritam napraviti manje iteracija u konstantnom vremenu. No, veći broj jedinki proširuje prostor pretraživanja i tako omogućava brže nalaženje optimuma. Ako se ovi parametri ispravno odrede, algoritam daje izvrsne rezultate.

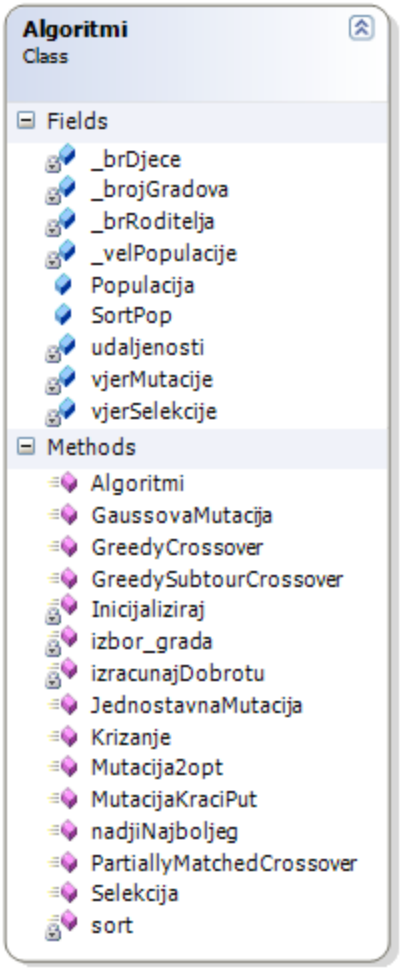
6. Dodatak A – dijagrami klasa






Parametri
Class
Fields
brojaRoditelja
BrojDjece
BrojGeneracija
crtaj
iznosUvjetaZaustavljanja
uvjetZaustavljanja
vrstaAlgoritma
vrstaKrizanja
vrstaMutacije
Methods
Default
Parametri (+ 1 overload)


IComparable
Kromosom
Class
Fields
_duljinaPuta
_kromosom
Properties
Chromosome
DuljinaPuta
Methods
CompareTo
Kromosom
ToString


Algoritmi
Class
Fields
_brDjece
_brojGradova
_brRoditelja
_velPopulacije
Populacija
SortPop
udaljenosti
vjerMutacije
vjerSelekcije
Methods
Algoritmi
GaussovaMutacija
GreedyCrossover
GreedySubtourCrossover
Inicijaliziraj
izbor_grada
izracunajDobrotu
JednostavnaMutacija
Krizanje
Mutacija2opt
MutacijaKraciPut
nadjiNajboljeg
PartiallyMatchedCrossover
Selekcija
sort

7. Dodatak B – dijelovi programskog koda

7.1 Klasa Kromosom

```
    // Klasa Kromosom definira jedan kromosom, stazu i duljinu puta
    public class Kromosom:IComparable
    {
        private ArrayList _kromosom = new ArrayList();
        private double _duljinaPuta;
        public Kromosom(ArrayList jedinka, double duljina)
        {
            foreach (int i in jedinka)
            {
                _kromosom.Add(i);
            }
            _duljinaPuta = duljina;
        }
        // Svojstvo koje vraca stazu
        public ArrayList Chromosome
        {
            get
            {
                return _kromosom;
            }
        }
        // Svojstvo koje vraca ili postavlja duljinu puta kromosoma
        public double DuljinaPuta
        {
            get
            {
                return _duljinaPuta;
            }
            set
            {
                _duljinaPuta = value;
            }
        }
        public override string ToString()
        {
            StringBuilder staza = new StringBuilder();
            foreach (int i in _kromosom)
            {
                staza.AppendFormat("{0}, ", i);
            }
            return "staza = " + staza.ToString() + " " + "duljina = " +
this._duljinaPuta.ToString();
        }
        #region IComparable Members
        public int CompareTo(object obj)
        {
            double d1 = this._duljinaPuta;
            double d2 = ((Kromosom)obj)._duljinaPuta;
            return d2.CompareTo(d1);
        }
        #endregion
    }
```

7.2 Mutacija 2opt

```
public Kromosom Mutacija2opt(Kromosom zaMutirati)
{
    double staraDuljina;
    int brojac = 0;
    do
    {
        brojac++;
        staraDuljina = zaMutirati.DuljinaPuta;
    //za grad i provrti po svim ostalim gradovima je li moguce skratiti put
        for(int i=0; i<_brojGradova; i++)
        {
            int j=i+2;
            while(j<_brojGradova)
            {
                int indA = i;
                int indB = (i+1)%_brojGradova;
                int indC = j;
                int indD = (j+1)%_brojGradova;
             //provjera je li zatovoljen uvjet poboljsanja rute
if((udaljenosti[(int)zaMutirati.Chromosome[indA]][(int)zaMutirati.Chromosom
e[indB]]+
udaljenosti[(int)zaMutirati.Chromosome[indC]][(int)zaMutirati.Chromosome[in
dD]]) >
(udaljenosti[(int)zaMutirati.Chromosome[indA]][(int)zaMutirati.Chromosome[i
ndC]]+
udaljenosti[(int)zaMutirati.Chromosome[indB]][(int)zaMutirati.Chromosome[in
dD]]))
                {
                    int duljina = indC-indB;
                    if(duljina<0)
                       duljina+=zaMutirati.Chromosome.Count;
                    duljina=duljina/2 +1;
                    //okretanje indeksa podrute
                    for(int l=0; l<duljina; l++)
                    {
                        object temp = zaMutirati.Chromosome[indB];
                   zaMutirati.Chromosome[indB]=zaMutirati.Chromosome[indC];
                        zaMutirati.Chromosome[indC] = temp;
                        if(++indB>=zaMutirati.Chromosome.Count)
                            indB = 0;
                        if(--indC<0)
                            indC = zaMutirati.Chromosome.Count-1;
                    }
                     zaMutirati.DuljinaPuta =
izracunajDobrotu(zaMutirati.Chromosome);
                }
                j++;
            }
        }
    }while(staraDuljina>zaMutirati.DuljinaPuta);
    if (brojac > 1)
        return zaMutirati;
    else
        return this.JednostavnaMutacija(zaMutirati);
}
```

7.3 Mutacija normalnom razdiobom i potrebne funkcije

```
public Kromosom GaussovaMutacija(Kromosom zaMutirati)
{
    ArrayList elementi1 = new ArrayList();
    ArrayList elementi2 = new ArrayList();
    foreach (int i in zaMutirati.Chromosome)
    {
        elementi1.Add(i);
        elementi2.Add(i);
    }
    Random rnd = new Random();
    int gen1 = rnd.Next(1,this._brojGradova); // izabran prvi grad
    int x = _brojGradova - gen1 - 1;
    int d = Math.Max(x, gen1);
    int gen2 = izbor_grada(d) + 1; // udaljenost pozicije drugog grada

    if ((gen2 + gen1) < _brojGradova)
    {
      // zamjena gena na poziciji gen1 i gena na poziciji gen1+gen2
        int tmp = (int)elementi1[gen1];
        elementi1[gen1] = elementi1[gen2 + gen1];
        elementi1[gen2 + gen1] = tmp;
        Kromosom dijete1 = new Kromosom(elementi1,
this.izracunajDobrotu(elementi1));

        if ((gen1 - gen2) >= 0)
        {
            // zamjena gena na poziciji gen1 i gena na poziciji gen1-gen2
            tmp = (int)elementi2[gen1];
            elementi2[gen1] = elementi2[gen1 - gen2];
            elementi2[gen1 - gen2] = tmp;
            Kromosom dijete2 = new Kromosom(elementi2,
izracunajDobrotu(elementi2));
            // vraćanje boljeg djeteta
            if (dijete1.DuljinaPuta < dijete2.DuljinaPuta)
            {
                return dijete1;
            }
            return dijete2;
        }
        return dijete1;
    }
    else if ((gen1 - gen2) >= 0)
    {
        int tmp = (int)elementi2[gen1];
        elementi2[gen1] = elementi2[gen1 - gen2];
        elementi2[gen1 - gen2] = tmp;
        Kromosom dijete = new Kromosom(elementi2,
izracunajDobrotu(elementi2));
        return dijete;
    }
    else
        return zaMutirati;
}
```

```
private int izbor_grada(int grad1)
{
    double broj;
    decimal q = Convert.ToDecimal(this._brojGradova) / 2;
    int m = Convert.ToInt32(Math.Ceiling(q));
    int i = m - (this._brojGradova - grad1);
    int k=0;
    Random autoRand = new Random();
    broj = autoRand.NextDouble();
    while (broj > vjerMutacije[i][k])
    {
        k++;
    }
    return k;
}

// izrada matrice vjerojatnosti za mutaciju normalnom razdiobom
private void vjer_mutacija()
{
    mat_vjer = new double[brGradova][];
    for (int i = 0; i < brGradova; i++)
    {
        mat_vjer[i] = new double[brGradova];
    }
    decimal q = Convert.ToDecimal(brGradova) / 2;
// mora biti vece cijelo od broj_gradova/2 redaka u matrici
    int m = Convert.ToInt32(Math.Ceiling(q));
    for (int k = 0; k < m; k++)
    {
        for (int i = 0; i < k + m; i++)
        {
            mat_vjer[k][i] = normalna_razdioba(i + 1, 5);
        }
    }
    for (int k = 0; k < m; k++)
    {
        double suma = 0;
        for (int i = 0; i < k + m; i++)
        {
            suma = suma + mat_vjer[k][i];
        }
// koeficijent za skaliranje sume vjerojatnosti na 1
        double koef = Math.Pow(suma, -1);
        suma = 0;
        for (int i = 0; i < k + m; i++)
        {
            mat_vjer[k][i] = koef * mat_vjer[k][i] + suma;
            suma = mat_vjer[k][i];
        }
    }
}
private double normalna_razdioba(int x, int q)
{
    double y = -(Math.Pow(x, 2)) / (2 * Math.Pow(q, 2));
    y = 2 * (Math.Exp(y));
    double z = q * (Math.Pow(2 * Math.PI, 0.5));
    y = y / z;
    return y;
}
```

7.4 GX križanje

```
public Kromosom GreedyCrossover(Kromosom rod1, Kromosom rod2)
{
    ArrayList elementi = new ArrayList();
    ArrayList stazaRod1 = new ArrayList();
    ArrayList stazaRod2 = new ArrayList();
    Random rnd = new Random();

    stazaRod1 = rod1.Chromosome;
    stazaRod2 = rod2.Chromosome;
    elementi.Add(stazaRod1[0]);

    for (int i = 1; i < stazaRod1.Count; i++)
    {
        int ind1 = (stazaRod1.IndexOf(elementi[i - 1]) + 1) % _brojGradova;
        int ind2 = (stazaRod2.IndexOf(elementi[i - 1]) + 1) % _brojGradova;
        int ind0 = (int)elementi[i - 1];

        if ((elementi.IndexOf(stazaRod1[ind1]) != -1) &&
(elementi.IndexOf(stazaRod2[ind2]) != -1))
        {
            int novi;
            do
            {
                novi = rnd.Next(0,_brojGradova);
            } while (elementi.IndexOf(novi) != -1);
            elementi.Add(novi);
            continue;
        }
        if (elementi.IndexOf(stazaRod1[ind1]) != -1)
        {
            elementi.Add(stazaRod2[ind2]);
            continue;
        }
        if (elementi.IndexOf(stazaRod2[ind2]) != -1)
        {
            elementi.Add(stazaRod1[ind1]);
            continue;
        }
        if (udaljenosti[ind0][(int)stazaRod1[ind1]] <=
udaljenosti[ind0][(int)stazaRod2[ind2]])
        {
            elementi.Add(stazaRod1[ind1]);
            continue;
        }
        if (udaljenosti[ind0][(int)stazaRod1[ind1]] >
udaljenosti[ind0][(int)stazaRod2[ind2]])
        {
            elementi.Add(stazaRod2[ind2]);
            continue;
        }
    }
    Kromosom dijete = new Kromosom(elementi, izracunajDobrotu(elementi));
    return dijete;
}
```

7.5 PMX križanje

```
public Kromosom PartiallyMatchedCrossover(Kromosom rod1, Kromosom rod2, int
tockaP1, int tockaP2)
{
    ArrayList elementi1 = new ArrayList();
    ArrayList elementi2 = new ArrayList();
    Random rnd = new Random();

    for (int i = 0; i < _brojGradova; i++)
    {
        elementi1.Add(null);
        elementi2.Add(null);
    }
    //dodavanje dijelova izmedu tocaka prekida
    for (int i = tockaP1 + 1; i <= tockaP2; i++)
    {
        elementi2[i] = rod1.Chromosome[i];
        elementi1[i] = rod2.Chromosome[i];
    }
    //generiranje ostatka kromosoma prema definiciji
    for (int i = 0; i < _brojGradova; i++)
    {
        if ((i > tockaP1) && (i <= tockaP2))
            continue;
        if (elementi1.Contains(rod1.Chromosome[i]))
        {
            int index = rod2.Chromosome.IndexOf(rod1.Chromosome[i]);
            if (elementi1.Contains(rod1.Chromosome[index]))
            {
                index = rod2.Chromosome.IndexOf(rod1.Chromosome[i]);
                if (elementi1.Contains(rod1.Chromosome[index]))
                {
                    do
                    {
                        index = rnd.Next(0,_brojGradova);
                    } while (elementi1.Contains(rod1.Chromosome[index]));
                    elementi1[i] = rod1.Chromosome[index];
                }
                else
                    elementi1[i] = rod1.Chromosome[index];
            }
            else
            {
                elementi1[i] = rod1.Chromosome[index];
            }
        }
        else
        {
            elementi1[i] = (int)rod1.Chromosome[i];
        }
        if (elementi2.Contains(rod2.Chromosome[i]))
        {
            int index = rod1.Chromosome.IndexOf(rod2.Chromosome[i]);
            if (elementi2.Contains(rod2.Chromosome[index]))
            {
                index = rod1.Chromosome.IndexOf(rod2.Chromosome[i]);
                if (elementi2.Contains(rod2.Chromosome[index]))
```

```
                {
                    do
                    {
                        index = rnd.Next(0, _brojGradova - 1);
                    } while (elementi2.Contains(rod2.Chromosome[index]));
                    elementi2[i] = rod2.Chromosome[index];
                }
                else
                {
                    elementi2[i] = rod2.Chromosome[index];
                }
            }
            else
            {
                elementi2[i] = rod2.Chromosome[index];
            }
        }
        else
        {
            elementi2[i] = (int)rod2.Chromosome[i];
        }
    }
    Kromosom dijete1 = new Kromosom(elementi1,
izracunajDobrotu(elementi1));
    Kromosom dijete2 = new Kromosom(elementi2,
izracunajDobrotu(elementi2));
    if (dijete1.DuljinaPuta < dijete2.DuljinaPuta)
        return dijete1;
    return dijete2;
}
```

7.6 GSX križanje

```
public Kromosom GreedySubtourCrossover(Kromosom rod1, Kromosom rod2)
{
    ArrayList elementi = new ArrayList();
    Random rnd = new Random();
    bool fa = true, fb = true;
    int sluc = rnd.Next(0,_brojGradova);
    int indR1 = rod1.Chromosome.IndexOf(sluc);
    int indR2 = rod2.Chromosome.IndexOf(sluc);
    elementi.Add(sluc);

    do
    {
        if (indR1 == 0)
            indR1 = 2;

        indR1 = (indR1 - 1) % _brojGradova;
        indR2 = (indR2 + 1) % _brojGradova;

        if (fa)
        {
            if (elementi.Contains(rod1.Chromosome[indR1]))
                fa = false;
            else
            {
                elementi.Add(null);
                for (int i = elementi.Count - 1; i > 0; i--)
                {
                    elementi[i] = elementi[i - 1];
                }
                elementi[0] = rod1.Chromosome[indR1];
            }
        }
        if (fb)
        {
            if (elementi.Contains(rod2.Chromosome[indR2]))
                fb = false;
            else
                elementi.Add(rod2.Chromosome[indR2]);
        }
    } while ((fa) || (fb));

    while (elementi.Count < _brojGradova)
    {
        sluc = rnd.Next(0, _brojGradova);
        if (elementi.Contains(sluc))
            continue;
        else
            elementi.Add(sluc);
    }
    Kromosom dijete = new Kromosom(elementi, izracunajDobrotu(elementi));
    return dijete;
}
```

8. Literatura

[1] Čeri M., Malović I. : Projekt – Evolucijske strategije, http://www.autobrodogradnja.110mb.com/

[2] Golub M., Genetski algoritam: http://www.zemris.fer.hr/~golub/ga/ga_skripta1.pdf

[3] Lovrenčić V.: Genetski algoritam u primjeni, http://www.zemris.fer.hr/~golub/ga/studenti/lovrecic/Genetski%20algoritam%20u%20primjeni.htm