SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

PROJEKT

Evolucijske strategije

Monika Čeri

Iva Malović

Voditelj: *doc.dr.sc. Marin Golub*

Zagreb, studeni, 2007.

Sadržaj

1. Uvod

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. [2]

Evolucijske strategije su optimizacijske tehnike bazirane na prilagođavanju i evoluciji. Njihova središnja ideja je stvaranje jednog ili λ potomaka operacijom mutacije iz jednog ili μ roditelja. U sljedeću se generaciju prenosi ili μ roditelja i λ potomaka ili samo λ potomaka. Roditelji i potomci predstavljaju potencijalno rješenje optimizacijskog problema.

Potencijalna rješenja prikazuju se pomoću vektora realnih brojeva. Broj na određenom mjestu unutar vektora opisuje neku karakteristiku samog rješenja. Kako bi se došlo do rješenja određenog problema koristi se operator mutacije na samim vektorima, a rjeđe se upotrebljava i operacija rekombinacije. U sljedeću generaciju odlaze samo najbolje jedinke koje se odabiru procesom selekcije.

2. Evolucijske strategije

Pod sljedećim podnaslovima dan je opis selekcije, genetskih operatora koji se koriste u evolucijskim strategijama, klasifikacija evolucijskih strategija, te je detaljno iznesen algoritam i princip rada evolucijskih strategija.

2.1 Selekcija

Pretpostavimo da postoji neka početna populacija jedinki. 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 sljedećem 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. [4]

2.1.1 Funkcija dobrote

Funkcija dobrote (*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*. [6]

Što je dobrota jedinke veća, jedinka ima veću vjerojatnost preživljavanja, a time i veću mogućnost prijenosa svojih gena na sljedeću populaciju. Zbog toga je upravo funkcija dobrote ključ za proces selekcije. 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.

2.2 Genetski operatori

Druga važna karakteristika evolucijskih strategija je reprodukcija. Općenito, reprodukcija je proces razmnožavanja u kojem sudjeluju jedinke koje su preživjele proces selekcije u prethodnoj populaciji. Razmnožavanje se obavlja pomoću genetskih operatora koji djeluju na same jedinke. Kod evolucijskih strategija najviše se primjenjuje operator mutacije, a nešto manje operator rekombinacije.

2.2.1 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. Najjednostavniji primjer mutacije je vjerojatnost da se neki bit u genetskom kodu promijeni iz svog originalnog stanja u neko novo stanje. 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 $N(x_i, \sigma_i^2)$. [4]

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. [3]

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.

2.2.2 Rekombinacija

Rekombinacija ili križanje je operator koji se rjeđe upotrebljava od operatora mutacije. U samom procesu rekombinacije stvaraju se nove jedinke koje sadrže kombinirane informacije sadržane u dva roditelja. Dakle, križanje je binarni operator jer djeluje na dvije jedinke u populaciji istovremeno. To se postiže kombinacijom varijabli koje sadržavaju roditelji. Zbog toga je najvažnija karakteristika križanja da potomci nasljeđuju svojstva svojih roditelja. Ako su roditelji dobri, tada će najvjerojatnije i potomak koji nastaje njihovim križanjem biti dobar, ako ne i bolji od svojih roditelja.

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

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

2.3 Klasifikacija evolucijskih strategija

Evolucijske strategije dijele se s obzirom na različite tipove evolucijskih procesa koje oni koriste. Razlika je u samom izboru roditelja i upotrebi rekombinacije.

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. [1]

2.3.1 (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 ona jedinka koja je bolja, tj. koja ima bolji faktor dobrote.

2.3.2 (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.

2.3.3 (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.

2.3.4 (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.

2.3.5 (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.

2.3.6 (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.

2.3.7 $(\mu' , \lambda'(\mu , \lambda)^{\gamma})$-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.

2.4 Algoritam evolucijskih strategija

Kao što je ranije navedeno, evolucijska strategija ima kao svoj cilj stvoriti populaciju potomaka pomoću 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. [5]

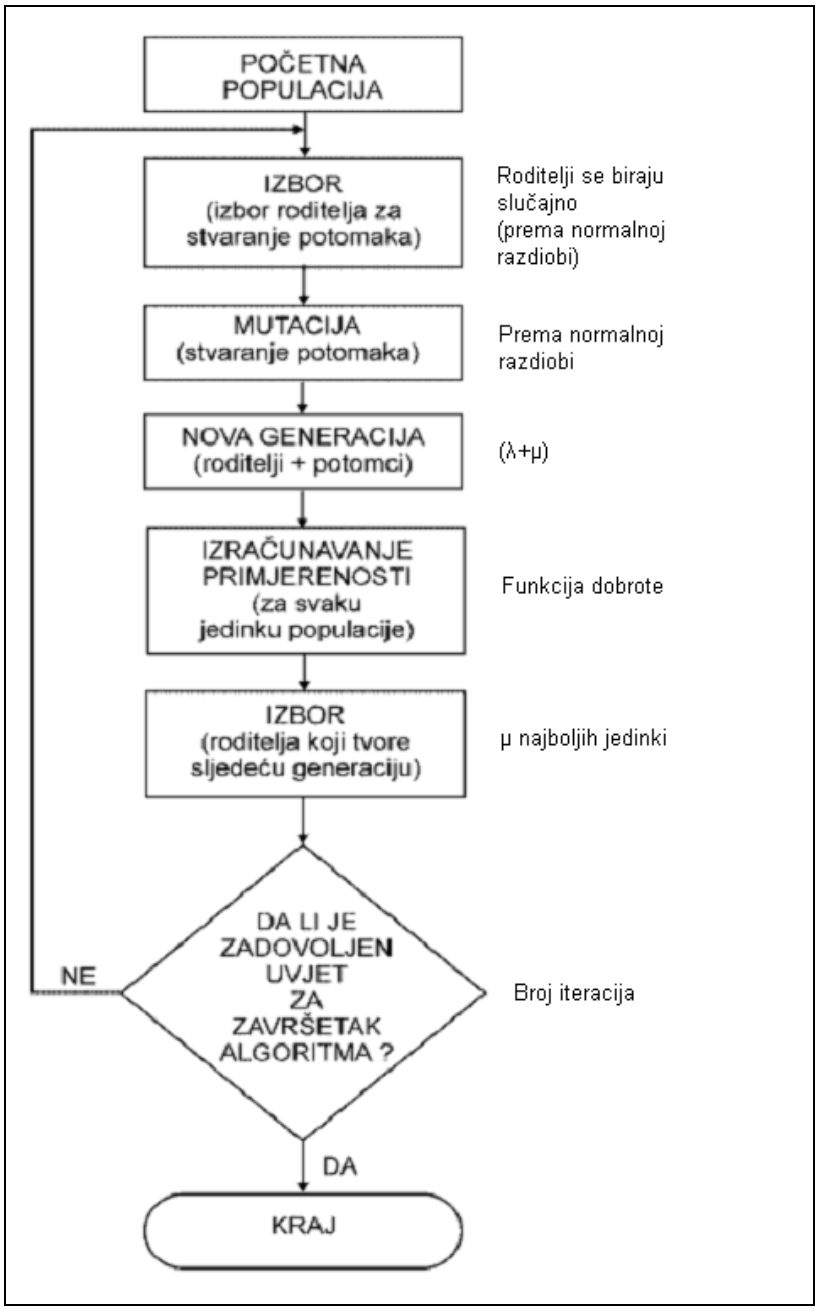
```
Evolucijska strategija{
      t = 0;
      generiraj početnu populaciju P(0);
      evaluiraj P(0); //provjeri dobrotu inicijalne populacije
      sve dok nije zadovoljen uvjet završetka evolucijskog procesa {
            izaberi najboljih μ roditelja iz P(t) i stavi ih u Pp(t);
            iz Pp(t) 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 sljedeće korake:

1. stvori λ novih jedinki:
    a) odaberi slučajnih ρ≤μ 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 (λ + μ)-ES.

Dijagram tijeka evolucijske strategije prikazan je na Slici 2.2. [7]



Slika 2.2. Dijagram toka evolucijske strategije

3. Primjena evolucijskih strategija

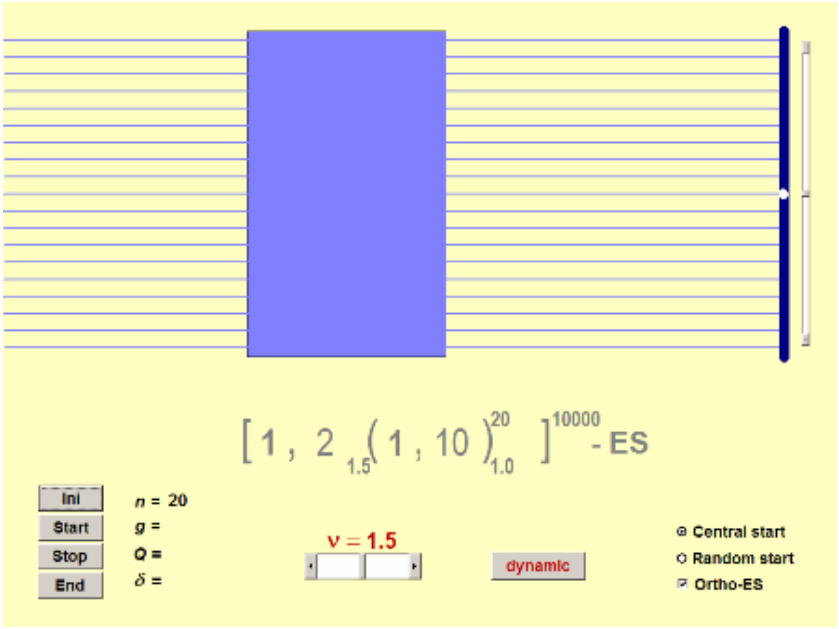
Evolucijske strategije imaju vrlo široku primjenu u poljima optimizacije, uključujući optimizacijske probleme u kontinuiranom i diskretnom vremenu, vektorsku optimizaciju i kombinatorne probleme sa i bez ograničenja. [2] Koriste se u *data miningu*, u izradi kompliciranih vremenskih rasporeda, u kemiji, 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][8]

4. Aplikacije

4.1 Izrada leće

Zadatak evolucijske strategije je inicijalni oblik leće promijeniti na takav način da se svaka paralelna zraka nakon prolaska kroz leću reflektira u zadanu točku P na zastoru.

Leća je u početku nakupina od n prizmi, te se stoga debljina leće može promijeniti na (n+1) mjesta. Parametri r2 i r1 su indeksi loma svjetlosti prizme i okolnog medija respektivno. Omjer r2/r1, u aplikaciji označen simbolom *v* može se podesiti između vrijednosti 0.2 i 3.0. Taj omjer zapravo određuje želimo li konkavnu (v<1) ili konveksnu leću (v>1). Na slici 4.1 prikazano je inicijalno stanje leće. Parametar n označava broj paralelnih zraka, g broj generacije, Q funkciju dobrote.



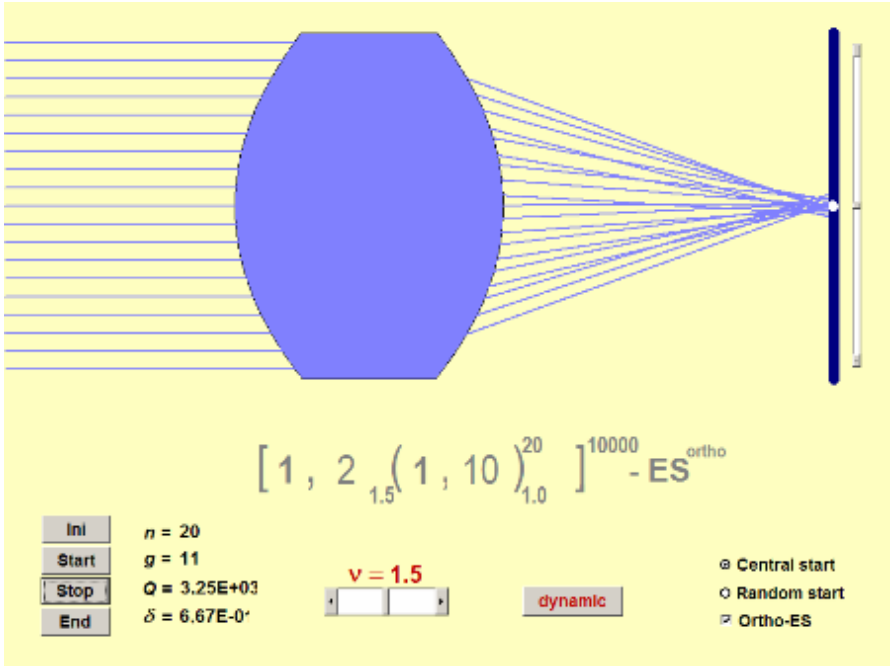
Slika 4.1 Inicijalno stanje leće

Funkcija dobrote je suma kvadrata udaljenosti između točke P i točke u koju se zraka zapravo reflektirala:

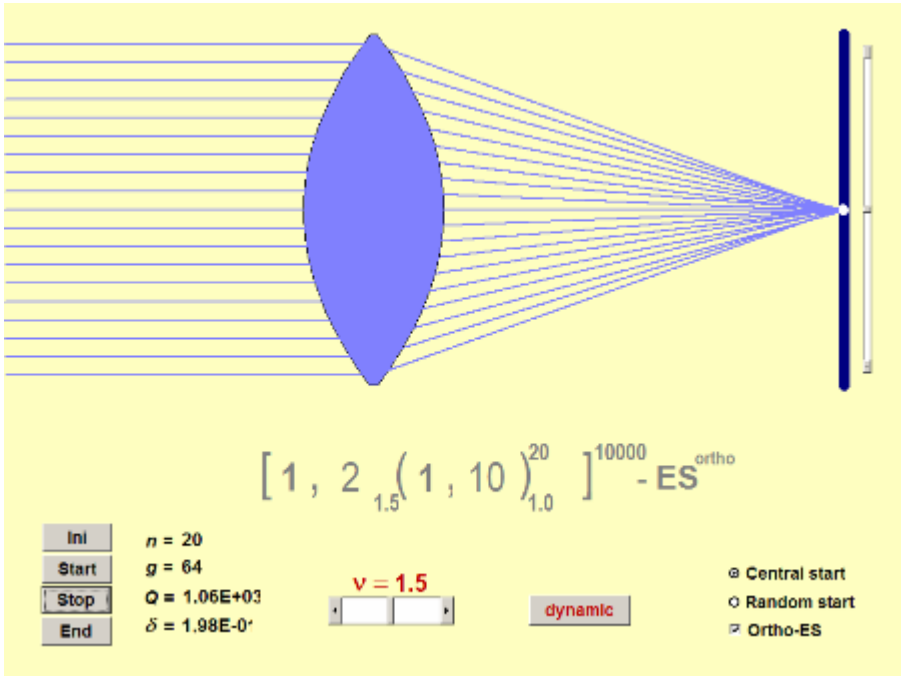
$$F = \sum_{i=1}^{n} d_i^2 \quad \text{min} \tag{3.1}$$

Osim ovog kriterija, optimizira se i debljina leće tako da leća bude što uža.

Na slikama 4.2 i 4.3 prikazan je rad aplikacije.



Slika 4.2. Izgled leće nakon 11 generacija



Slika 4.3. Izgled leće nakon 64 generacija

Opisana aplikacija i još drugih primjera evolucijskih strategija mogu se za vlastitu upotrebu skinuti na Internet adresi http://www.bionik.tu-berlin.de/institut/s2anima.html . [8]

4.2 Problem trgovačkog putnika

4.2.1 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:

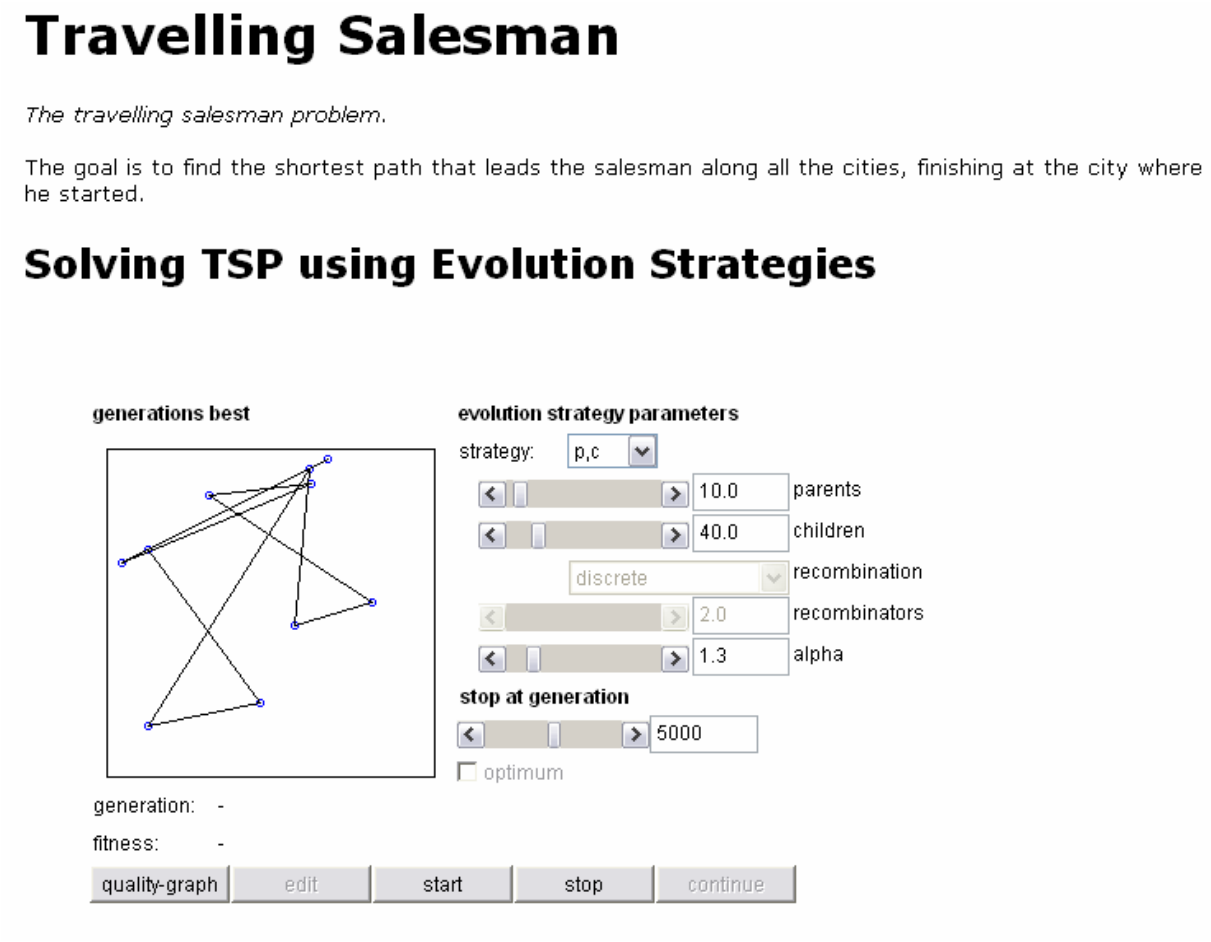
> Neka je zadan cijeli broj $n \geq 3$ i $n \times n$ matrica $C = (c_{ij})$, gdje je c_{ij} poprima nenegativne cjelobrojne vrijednosti. Traži se koja permutacija π cijelih brojeva od 1 do n minimizira sumu $\sum_{i=1}^{n} c_{i\pi(i)}$. [9]

Kroz godine ovaj problem je okupirao umove mnogih istražitelja zbog mnogih razloga. Prvo, problem trgovačkog putnika je vrlo lako za opisati, ali predstavlja veliki problem prilikom rješavanja. Nije poznat niti jedan vremenski polinomni algoritam kojim se on može riješiti. Nedostatak tih polinomnih algoritama je karakteristika klase NP – teških algoritama. Drugo, problem je široko primjenjiv na različite probleme usmjeravanja i raspoređivanja. Na kraju, kako je poznato mnogo informacija o ovom problemu, postao je neka vrsta testnog problema za sve nove kombinatoričke metode optimizacije.

4.2.2 Opis aplikacije

Aplikacija prikazuje problem riješen pomoću evolucijskih strategija. [10]

Traži se optimalna ruta između deset unaprijed zadanih gradova. Korisnik može sam izabrati jednu od četiri vrste ES. U ovom slučaju se umjesto oznaka μ i λ koriste p i c. Tako je moguće pokrenuti aplikaciju korištenjem (p, c) – ES, (p+c) – ES, (p/r+c) – ES, te (p/r, c) – ES. Nadalje, korisnik može izabrati broj roditelja i djece, a ako koristi jednu od zadnje dvije spomenute strategije, u kojima se koristi i rekombinacija, onda može izabrati vrstu rekombinacije i broj rekombinata. Na samom kraju bira broj generacija. Na lijevoj strani može se vidjeti izabrana ruta u određenom koraku, a ispod slike se ispisuje vrijednost funkcije dobrote. (Slika 4.2)



Slika 4.4 Prikaz aplikacije koja rješava TSP

5. Programsko rješenje problema trgovačkog putnika

5.1 TSP – korištenje (μ , λ) – ES (autor: Monika Čeri)

5.1.1 Opis ostvarenja

Razvijen je program koji rješava problem trgovačkog putnika korištenjem evolucijskih strategija. Problem se očituje u potrazi za najkraćim putem kojeg putnik mora prijeći počevši od nekog početnog grada N1, tako da prođe kroz sve gradove točno jednom i ponovno se vrati u početni grad N1. Mreža gradova i udaljenosti između svakoga grada moraju biti unaprijed poznati. Program barata sa simetričnim udaljenostima, tj. udaljenost od grada N1 do N2 jednaka je udaljenosti od grada N2 do N1. Program je napravljen tako da korisnik na početku može izabrati da računalo generira te udaljenosti za njega ili da sam unese neki vlastiti problem koji želi optimizirati.

Pri razvoju programa korištena je strategija (μ, λ)-ES. Jedna jedinka je predstavljena kao jedna od mogućih ruta koju putnik može prijeći. Cilj strategije je pronaći minimum funkcije dobrote, koja je u ovom slučaju definirana kao suma svih udaljenosti između gradova u nekoj ruti. Nakon generiranja početne populacije rješenja (jedinki roditelja), svaka jedinka roditelja procesom mutacije daje s/p potomaka. Mutacija se temelji na zamjeni dva slučajno izabrana grada, a vjerojatnost izbora svih gradova je jednaka (uniformna razdioba). Dobiveni potomci se sortiraju po funkciji dobrote i najboljih p prelazi u sljedeću generaciju roditelja procesom selekcije. Program završava nakon unaprijed zadanog broja generacija i ispisuje najbolje dobiveno rješenje.

5.1.2 Tehničke značajke

Za izradu programa korišten je programski alat Dev-C++. Program je napisan u programskom jeziku C, a može se pokretati u Windows okruženju. Sastoji se od 376 linija koda. Izvršni program je 32-bitna aplikacija veličine 36,0 KB.

Na početku programa se uključuju standardne potrebne biblioteke. Također, definirane su dvije globalne veličine: maksimalan broj gradova (`MAXGRAD`) i pomoćna varijabla (`POM`) potrebne za definiciju globalnih varijabli. Nakon toga definirane su četiri cjelobrojne globalne varijable, matrice, da bi se izbjegao prijenos dvodimenzionalnog polja između funkcija:

1.) `int mat_udalj[MAXGRAD][MAXGRAD];` – simetrična matrica u kojoj se nalaze udaljenosti između gradova,

2.) `int populacija[MAXGRAD][MAXGRAD+1];` – matrica u kojoj se nalaze jedinke roditelja u određenoj generaciji, svaki red predstavlja jednu jedinku, tj. jednu moguću rutu obilaska gradova,

3.) `int potomci[POM][MAXGRAD+1];` – matrica u kojoj se nalaze svi potomci koji se dobiju mutacijom iz roditelja, svaki red predstavlja jednu jedinku, a zadnje mjesto je rezervirano za funkciju dobrote te jedinke,

4.) `int temprjesenje[POM][MAXGRAD+1];` – matrica u kojoj se nalazi najbolje rješenje iz svake generacije koje služi za izračun konačnog rješenja, svaki red predstavlja najbolju rutu obilaska gradova u toj generaciji, a na zadnjem mjestu se nalazi izračunata funkcija dobrote te jedinke.

Program se sastoji od glavne (`main`) funkcije i 14 jednostavnih funkcija. Unutar glavne funkcije implementiran je izbornik koji poziva jednu od prve četiri niže navedene funkcije. Slijedi popis i kratak opis ostalih funkcija.

1.) `void opis_TSP();` – Služi za opis problema trgovačkog putnika.

2.) `void opis_ES();` – Služi za opis korištene evolucijske strategije i parametara s kojima se susreće korisnik.

3.) `void TSP_random();` – Služi za unos broja gradova i generiranje matrice udaljenosti ukoliko korisnik želi da samo računalo zada neki problem.

4.) `void TSP_vlastiti();` – Služi za unos broja gradova i elemenata u matricu udaljenosti ukoliko korisnik želi riješiti neki svoj problem.

Zadnje dvije funkcije nakon generiranja matrice udaljenosti pozivaju funkciju za izbor postavki.

5.) `void TSP_postavke (int brgrad);` – Funkcija prima parametar brgrad, koji predstavlja odabrani broj gradova . Služi za unos početnog grada (int pocgrad), broja početne populacije roditelja (int brpop), broja potomaka (int brpot) i izbora ukupnog broja generacija (int brgeneracija). Na kraju poziva funkciju kojoj šalje sve ove parametre.

6.) `void TSP_main (int pocgrad, int brpop, int brpot, int razdioba, int brgrad, int brgeneracija);` - Funkcija prima parametre: početni grad, broj roditelja, broj potomaka, koja se razdioba koristi (u našem slučaju implementirana je samo uniformna razdioba), broj gradova i broj generacija. Na početku se izračuna koliko svaki roditelj mora proizvesti djece procesom mutacije (`int omjer`), a nakon toga je prikazan osnovni algoritam evolucijske strategije. Na početku se generira početna populacija i ispiše po želji. Sve dok nije zadovoljen uvjet zaustavljanja (dok se ne izvrti `brgeneracija` iteracija) prvo se mutiraju roditelji. Dobivenim potomcima se izračuna funkcija dobrote i sortira ih se po toj vrijednosti. Nakon toga se u novu generaciju premjesti `brpop` broj najboljih potomaka i oni postaju novi roditelji. Na samom kraju slijedi ispis najboljih rješenja kroz generacije, te konačan ispis najboljeg dobivenog rješenja.

7.) `void generiraj_pocetnu (int pocgrad, int brpop, int brgrad);` – Funkcija prima parametre: početni grad, broj populacije i gradova. Korištenjem `random` funkcije generira različite moguće rute obilaska gradova (jedinke roditelja) i puni matricu `populacija[][]`.

8.) `void ispisi_pocetnu (int brpop, int brgrad);` – Funkcija prima parametre: broj populacije i broj gradova. Služi po želji za kontrolu generiranja matrice udaljenosti i početne populacije. Ukoliko se želi ovaj ispis potrebno je maknuti oznake komentara iz funkcije.

9.) `void mutiraj (int razdioba, int brpop, int brpot, int brgrad, int omjer);` – Funkcija prima parametre: vrstu razdiobe, broj populacije, broj potomaka, broj gradova i omjer. Prije svake iteracije matrica potomaka se postavlja na nulu. Izabiru se slučajno dva različita grada koji moraju biti različiti od početnog grada i zamijene im se mjesta te tako nastaje mutant koji se sprema u polje `mutant[]`. Nakon toga se provjerava postoji li u matrici `potomci[][]` identična jedinka dobivena od tog roditelja. Ukoliko postoji postupak se ponavlja, inače se jedinka kopira u matricu `potomci[][]`. Gore

navedeni postupak ponavlja se za svaku jedinku iz matrice `populacija[][]` omjer puta.

10.) `void fnk_dobrote (int brpop, int brgrad, int omjer);` – Funkcija prima parametre: broj populacije, broj gradova i omjer. Služi za izračun funkcije dobrote (sume svih udaljenosti između gradova unutar jedne zadane rute obilaska gradova). Funkcija dobrote se računa za svaku jedinku unutar matrice `potomci[][]`, a sprema se na zadnje mjesto u svakom retku.

11.) `void sortiraj (int brpop, int brgrad, int omjer);` – Funkcija prima parametre: broj populacije, broj gradova i omjer. Služi za sortiranje jedinki u matrici `potomci[][]` po izračunatoj funkciji dobrote.

12.) `void premjesti_u_novu (int brpop, int brgrad, int pom);` – Funkcija prima parametre: broj populacije, broj gradova i pom (varijabla sa oznakom trenutne generacije). Iz matrice `potomci[][]` premješta se `brpop` najboljih jedinki u matricu `populacija[][]`. Također se u matricu `temprjesenje[][]` sprema najbolje rješenje te generacije.

13.) `void ispis_rjesenja (int brpop, int brgrad, int pom);` – Funkcija prima parametre: broj populacije, broj gradova i pom. Na kraju se u matrici `temprjesenje[][]` nalaze sva najbolja rješenja tokom `brgeneracija` generacija. Ispiše se rješenje prve generacije, a kasnije se ispisuju samo ona rješenja koja su bolja od zadnjeg ispisanog. Ispisuje se broj generacije, funkcija dobrote (udaljenost između gradova) i ruta.

14.) `void najbolje (int pom, int brgrad);` – Funkcija prima parametre: pom i broj gradova. Služi za konačan ispis najboljeg nađenog rješenja i prikladne rute.

5.1.3 Upute za korištenje

Kako se proizvod sastoji od izvornog koda i izvršne (exe) datoteke, moguće ga je pokrenuti izravnim pokretanjem izvršne datoteke ili ponovnim *kompajliranjem* i pokretanjem samog koda. Nakon pokretanja programa otvara se izbornik, pomoću kojeg korisnik odabire jednu od pet mogućih opcije:

1.) opis problema trgovačkog putnika,

2.) opis korištene evolucijske strategije,

3.) TSP – random generiranje matrice udaljenosti,

4.) TSP – riješite vlastiti problem,

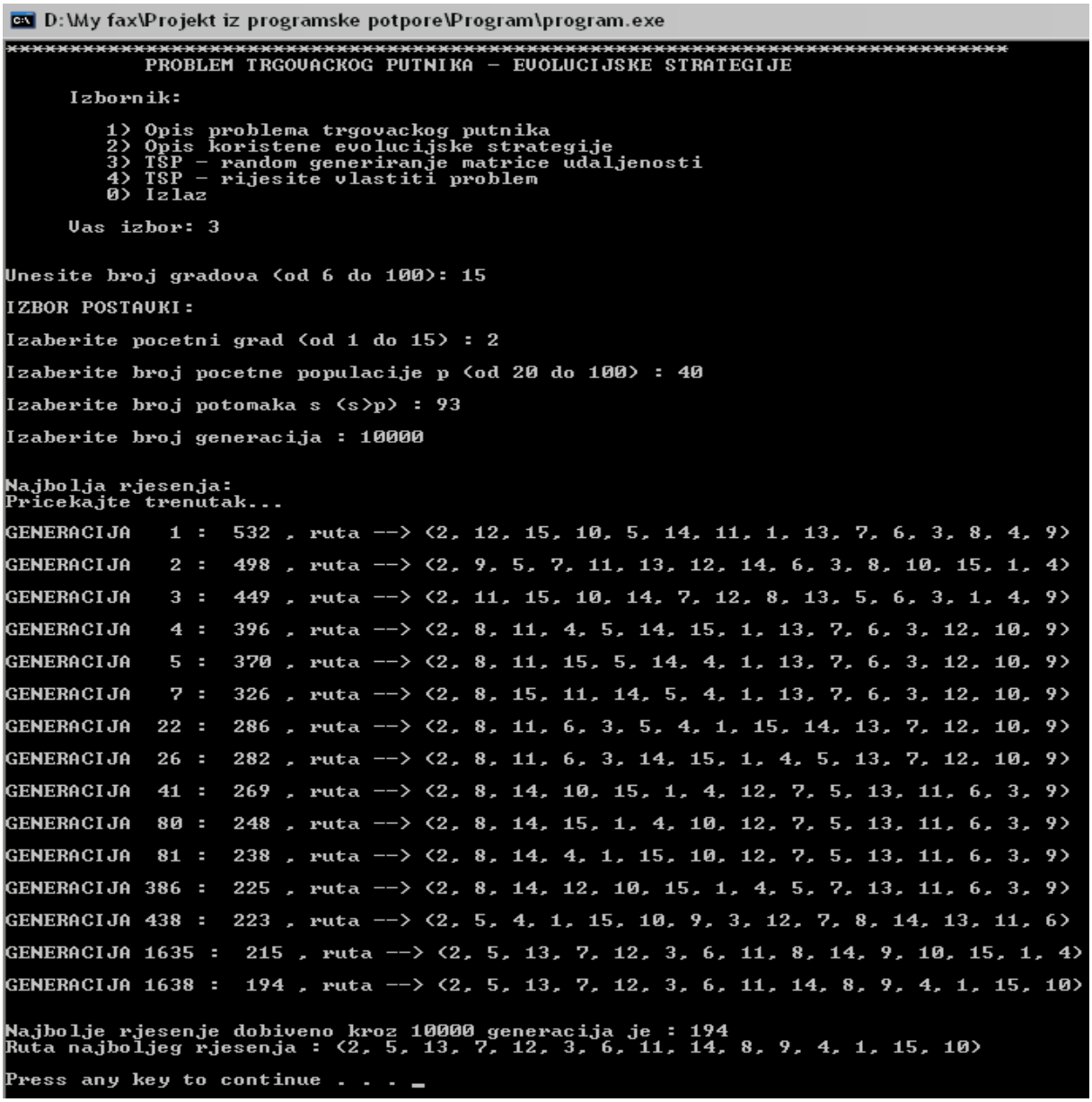
5.) izlaz.

Prva i druga opcija služe za upoznavanje korisnika sa problemom i kratkim opisom korištenog algoritma. Najvažnija svrha ove dvije opcije je da se korisnik upozna sa varijablama čije će vrijednosti sam morati kasnije unositi. Treća i četvrta opcija pokreću glavni program. Razlikuju se u tome što računalo kod opcije tri samo generira udaljenosti između gradova, a kod opcije četiri korisnik sam unosi vrijednosti.

Na početku kod obje opcije korisnik unosi broj gradova, koji je u ovom slučaju ograničen na interval [6, 100]. Nakon toga kod opcije četiri unose se udaljenosti između pojedinih gradova. Slijedi kod obje opcije izbor postavki. U postavkama se na početku bira početni grad. Nakon toga biraju se parametri evolucijske strategije p i s. Prvo se bira broj početne populacije, koji

predstavlja početnu populaciju roditelja, a ograničen je na interval [20, 100]. Slijedi izbor broja potomaka koji mora biti veći od početne populacije roditelja. Na kraju se bira broj generacija, a taj broj predstavlja uvjet zaustavljanja samog programa.

Nakon izbora postavki slijedi ispis rješenja. Ispisuje sa samo ono rješenje koje je bolje od trenutnog najboljeg rješenja. Na kraju se ispiše najbolje rješenje kroz zadan broj generacija i njegova ruta. (Slika 5.1)



Slika 5.1 Primjer jednog ispisa programa

5.2 TSP – korištenje (μ + λ) – ES (autor: Iva Malović)

5.2.1 Opis ostvarenja

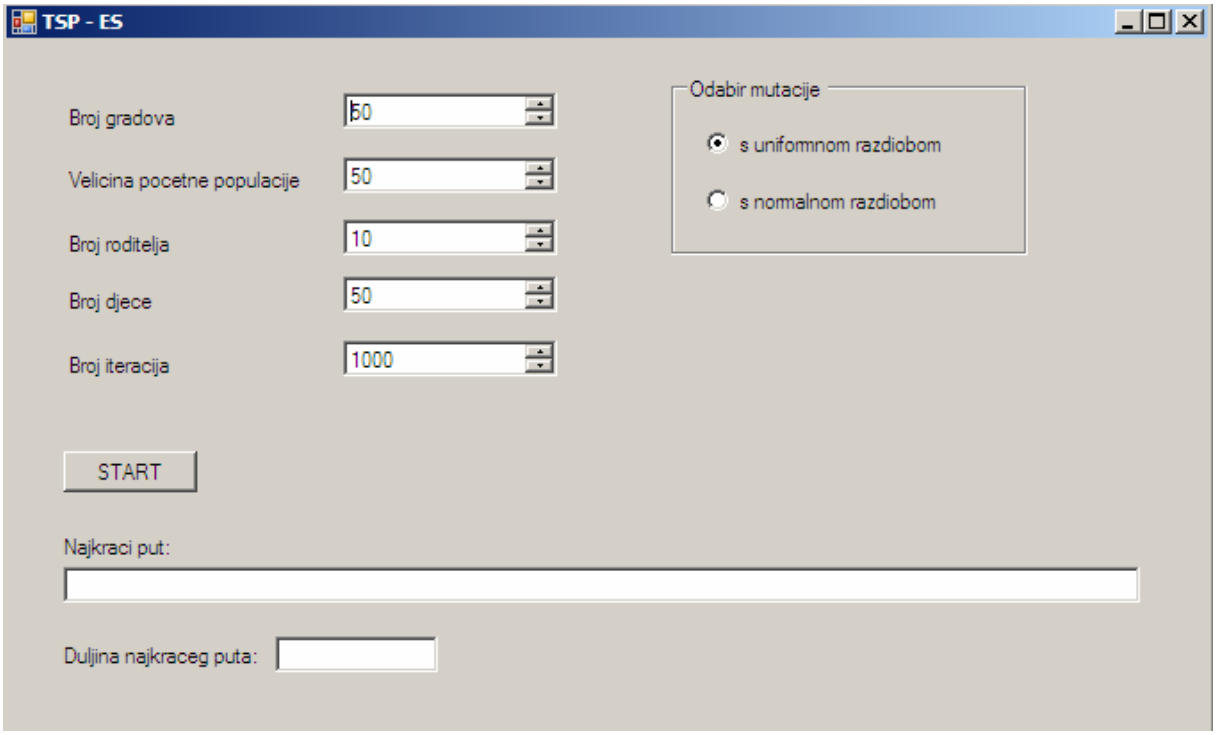
Za ranije opisan problem trgovačkog putnika, ova verzija programa koristi (μ+λ) – ES. Problem generira računalo s obzirom na odabrani broj gradova.

Jedinke su zapisane kao niz koji čine redoslijedi obilazaka gradova. Nakon generiranja slučajne početne populacije bira se μ najboljih jedinki procesom selekcije od kojih svaka procesom mutacije (uniformnom ili normalnom) daje λ/μ potomaka. Mutacija s uniformnom razdiobom se temelji na zamjeni dva slučajno izabrana grada. Mutacija s normalnom razdiobom bira slučajno jedan grad, a zatim drugi grad bira s vjerojatnošću uzetu iz normalne razdiobe prema udaljenosti između polja u jedinki, te zamjenjuje ta dva odabrana grada. Iz skupa jedinki roditelja i djece (μ+λ) bira se opet najboljih μ jedinki za roditelje i iteracije se nastavljaju dalje. Program završava nakon unaprijed zadanog broja generacija i ispisuje najbolje dobiveno rješenje.

5.2.2 Tehničke značajke i upute za korištenje

Za izradu programa korišten je programski jezik C# u alatu Microsoft Visual Studio 2005. Izvršni program je Win32 aplikacija, veličine 28 KB.

Program se sastoji od forme TSP – ES (slika 5.2) i forme *wait_window*, koja će služiti za ispis trenutne najbolje jedinke.



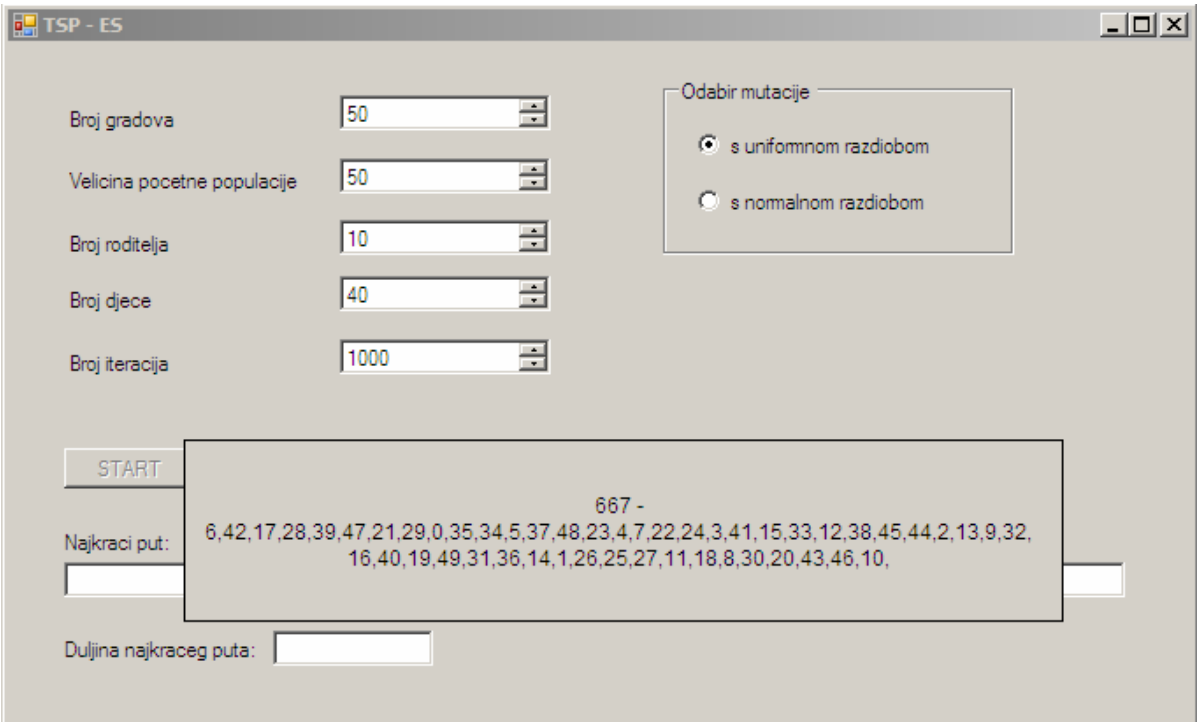
Slika 5.2 Izgled aplikacije prije pokretanja

U tablici 5.1 prikazani su opisi pojedinih funkcija:

Tablica 5.1 Funkcije i kratki opisi

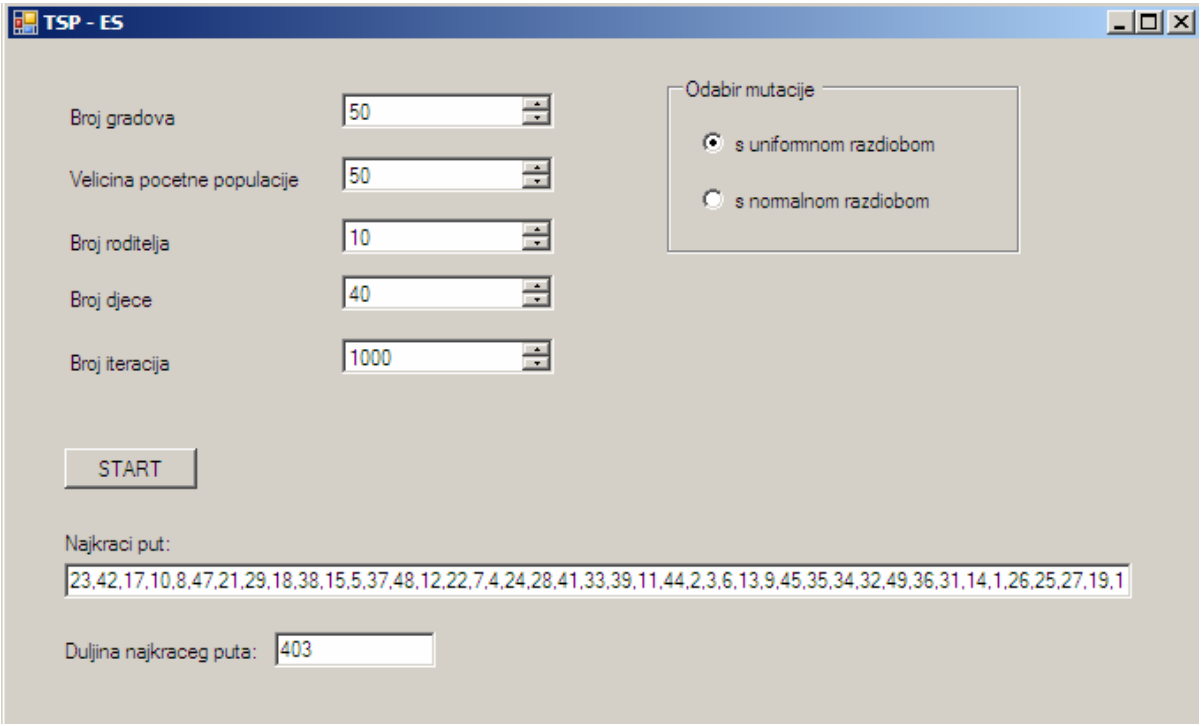
Funkcija	Opis
`void matrica_udaljenosti()`	Puni matricu `matrica[,]` brojevima iz intervala [1, 100], koji predstavljaju udaljenosti između gradova
`void pocetna_populacija()`	Puni retke matrice `populacija[,]` jedinkama. Jedinke se generiraju permutacijom rednih brojeva gradova
`void put_lenght(int[,] populacija, int br)`	Izračunava ukupni put za jednu rutu (jedinku) i upisuje ga na zadnje polje u pojedinom retku matrice `populacija[,]`
`void sort(int[,] populacija, int br)`	Funkcija sortira retke u matrici `populacija[,]` prema dobroti tj. duljini puta
`void selekcija()`	Radi selekciju `broj_roditelja` najboljih jedinki iz matrice `populacija[,]` i upisuje ih u matricu `roditelji[,]`
`void normalna_mutacija (double[,] matrica_vjer)`	Slučajno bira jedan grad (gen) iz jedinke, poziva funkciju `izbor_grada` koja prema normalnoj razdiobi izabire drugi grad, te nakon toga zamjenjuje ta dva odabrana grada
`int izbor_grada(int d, double[,] matrica_vjer)`	Izabire grad na temelju zadane vjerojatnosti koju čita iz matrice `matrica_vjer[,]`
`void vjer_mutacija (double[,] matrica_vjer)`	Generira matricu `matrica_vjer[,]` uz pomoć funkcije `normalna_razdioba`. Svaki redak matrice sadrži kumulativne vjerojatnosti odabira grada s obzirom na maksimalnu moguću udaljenost u jedinki
`double normalna_razdioba (int x, int q)`	Vraća vrijednost iz normalne razdiobe.
`void uniformna_mutacija()`	Vrši mutaciju prema uniformnoj razdiobi. Svi gradovi imaju jednaku vjerojatnost mutacije.
`void prijepis(int[] naj_jedinka, StringBuilder niz, long gen, long uvj)`	Služi za stvaranje *stringa* u obliku povoljnom za ispis u *wait_windowu* iz varijable `int[]naj_jedinka` koja predstavlja trenutnu najbolju jedinku

Prilikom pokretanja algoritma (pritiskom tipke START) učitavaju se parametri sa forme: broj gradova, veličina početne populacije, broj roditelja, broj djece, broj iteracija i vrsta mutacije. Nakon toga se provjeravaju nejednakosti parametara. Evolucijska strategija zahtijeva sljedeću nejednakost: (*veličina početne populacije)* ≥ *(broj roditelja)* ≤ *(broj djece*). Ako za neki od parametara ne vrijedi zadana nejednakost, algoritam se zaustavlja, a na formi iskače prozorčić s adekvatnom porukom o pogrešci. U aplikaciji su prikazane predviđene (*defaultne*) vrijednosti parametara i odabir mutacije, koje korisnik može promijeniti u dozvoljenim okvirima. Broj gradova se bira iz intervala [4,99], veličina početne populacije, broj djece i roditelja se biraju iz intervala [2,100], a broj iteracija iz intervala [2,100000]. Nakon pokretanja programa (tipkom START) pojavit će se prozorčić u kojem će se ispisivati trenutna najkraća ruta (slika 5.3).



Slika 5.3. Rad aplikacije

Nakon završetka programa, najbolja ruta i udaljenost će se ispisati u odgovarajućim *textboxovima* (slika 5.4).



Slika 5.4 Izgled aplikacije nakon završetka izvođenja programa

6. Zaključak

7.

8. Literatura

[1] Weise T., Global Optimization Algorithms; dostupno na:
http://www.it-weise.de/projects/book.pdf , 24.10.2007.

[2] http://en.wikipedia.org/wiki/Evolution_strategy, 24.10.2007.

[3] http://www.techfak.uni-bielefeld.de/bcd/Curric/ProtEn/112.html, 24.10.2007.

[4] http://ls11-www.cs.uni-dortmund.de/people/kursawe/Demos/EvoNet/EStheory.html, 24.10.2007.

[5] Pielić, M.: Pregled evolucijskih algoritama http://www.zemris.fer.hr/~golub/ga/studenti/seminari/2007_pielic/Pregled%20evolucijskih%20algoritama%5B2007%5DPielic_Marko.htm, 24.10.2007.

[6] Golub M., Genetski algoritam: http://www.zemris.fer.hr/~golub/ga/ga_skripta1.pdf, 24.10.2007.

[7] Petrović J., Evolucijski algoritmi http://www.zemris.fer.hr/~yeti/studenti/radovi/Petrovic_Juraj_Seminar.pdf, 24.10.2007.

[8] Evolution Strategy in Action, 10 ES-Demonstrations
http://www.bionik.tu-berlin.de/user/giani/esdemos/evo.html, 27.10.2007.

[9] Larranaga P., *Tackling the Travelling Salesman problem with Evolutionary Algorithms: Representations and Operators:* http://citeseer.ist.psu.edu/cache/papers/cs/9133/http:zSzzSzwww.sc.ehu.eszSzccwbayeszSzpostscriptzSzehu-kzaa-ik-2-94.pdf/tackling-the-travelling-salesman.pdf, 7.11.2007.

[10] Craenen B., *Travelling Salesman – Solving TSP using Evolution Strategies* http://evonet.lri.fr/CIRCUS2/node.php?node=81, 29.10.2007.