SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

PROJEKT

Optimizacija kolonijom mrava

Tomislav Bronić

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

Zagreb, studeni, 2008.

Sadržaj

1. Uvod 1

2. Optimizacijski problem 2

2.1 Optimizacijski problemi 2

2.2 Traveling salesman problem 2

2.3 Vehicle routing problem 3

3. Sustav Mrava 4

3.1 Mravi u prirodi 4

3.2 Umjetni mravi 5

3.3 Algoritam mrava 6

4. Sustavi mrava u pokusima 8

4.1 Broj mrava 8

4.2 Applications of AI 9

4.2.1 Funkcijske tipke 9

4.2.2 Lista informacija 10

4.2.3 Skica gradova 10

4.2.4 Graf najbolje rute 11

4.3 Mjerenja 11

4.4 Usporedba sa Genetičkim algoritmom 14

5. Projektiranje ACO sustava 15

5.1 Ideja 15

5.2 Input & output 15

5.3 Realizacija 16

5.4 Korisničko sučelje 18

5.4.1 Lijevi meni 19

5.4.2 Desni meni 20

5.4.3 Graf najboljih ruta 20

5.4.4 Mapa 21

5.5 Mjerenja 24

5.5.1 Krugovi 24

5.5.2 Slučajni problem TSP 26

5.5.3 VRP Krugovi 27

6. Sažetak 29

7. Literatura 30

1. Uvod

U ovom radu obrađuje se optimizacija kolonijom mrava (*ant colony optimization* – ACO u daljnjem dijelu teksta). ACO je algoritam koji po svojem načinu rada spada u kategoriju evolucijskih algoritama. Uže ga možemo svrstati među *swarm intelligence*, jer ne koristi tipične rutine genetskih algoritama, već se zasniva na kolektivnom znanju i dijeljenju informacija među mnogim jedinkama.

Nakon objašnjenja optimizacijskih problema koji se rješavaju ACO algoritmima slijedi biološka osnova ACO algoritma. Način rada mrava pri pronalaženju hrane, kao dio velike kolonije, pokušava se imitirati računalnim algoritmom.

U sklopu projekta demonstrirani su jedan slobodno dostupan program koji koristi ACO algoritam, te jedan novi program napisan za ovaj projekt.

2. Optimizacijski problem

2.1 Optimizacijski problemi

Vrsta optimizacijskih problema ima mnogo. Za sve jednostavne zadatke postoje deterministički algoritmi koji brzo mogu pronaći savršeno rješenje, no za veće i složenije probleme deterministički pristup susreće se sa svojim granicama. Pod pojmom "složeni optimizacijski problemi" mislimo na zadatke koji se ne mogu riješiti determinističkim metodama u kratkom vremenu, iz jednostavnog razloga što postoji prevelik broj mogućih kombinacija. Vrijeme potrebno da računalo prođe kroz sve postojeće moguće kombinacije rješenja je često neprihvatljivo dugo, stoga se koriste evolucijski algoritmi koji ne daju uvijek optimalno rješenje, ali dođu prihvatljivo blizu optimalnom rješenju u prihvatljivo kratkom vremenu.

2.2 Travelling salesman problem

Problem putnika, odnosno putujućeg prodavača, je naziv za specifičnu vrstu optimizacijskog problema. Problem je zadan na sljedeći način:

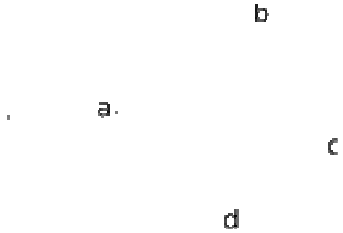
Jedan prodavač mora obići sve gradove koji su mu zadani, no u svaki grad smije ući samo jednom. Kojim putem, odnosno redoslijedom je najbolje obilaziti gradove, ako su udaljenosti među gradovima različite?

Prikazano formulom (*formula 2.1*) ovaj se zadatak može postaviti kao traženje minimuma ukupnog puta, pri čemu je *n* broj gradova u problemu, a *i* redni brojevi grada kojeg je putnik obišao.

$$MIN(d_{1,2} + d_{2,3} + d_{3,4} + ...) = MIN\left[(\sum_{i=1}^{n-1} d_{i,i+1}) + d_{n,1} \right]$$

Formula 2.1 – definicija TSP

Gradovi su zadani kao točke u ravnini, međusobno udaljene neki konstantan iznos (primjer *Slika 2.1*). Naravno genetski algoritmi se neće primjenjivati na ovako bizarno jednostavne zadatke kada su samo četiri grada u pitanju, već za probleme kada je u pitanju više stotina ili tisuća gradova.



Slika 2.1 - Travelling salesman problem sa 4 grada

Složenost ovog problema je $\frac{1}{2}(n-1)!$. To znači da će za ova četiri grada postojati svega tri moguća rješenja, od kojih je lako izabrati najbolje, no već za pet gradova tu je 12 mogućih rješenja. Broj mogućnosti raste strahovito brzo i tako nadilazi sposobnosti determinističkih

algoritama za velike brojeve gradova. Primjerice za 50 gradova bi broj mogućnosti koje deterministički stroj mora proći bio 3×10^{64}.

2.3 Vehicle routing problem

Problem usmjeravanja vozila sličnog je porijekla kao problem putujućeg prodavača. Radi se o skupu vozila koja moraju poslužiti klijente. Problem usmjeravanja vozila poznat je diljem svijeta i često se mora rješavati, primjerice službe isporuke i dostave kao što su pošta, picerije, ispunjavanje popisa stanovništva. Potrebno je organizirati isporuke više dostavljača kako bi se povećala ušteda goriva i vremena. Bez dodatnih uvjeta, i uz broj vozila 1, vehicle routing problem odgovara travelling salesman problemu. Postoje mnoge varijacije problema usmjeravanja vozila koje lagano mijenjaju problem, a većina ih potječe iz konkretnih primjera u svijetu. Osnovni VRP se može prikazati *formulom 2.2*, pri čemu *V* označava broj vozila, *v* redni broj vozila, *n* broj gradova, *i* redni broj grada obiđen, a *(n/V)* broj gradova koje obilazi svako vozilo.

$$MIN\left[\sum_{v=1}^{V}\left[\left(\sum_{i=(v-1)\cdot(n/V)}^{v\cdot(n/V)-1}\left(d_{i,i+1}\right)\right)+d_{v\cdot(n/V)-1,1}\right]\right]$$

Formula 2.2 – definicija VRP

Tako primjerice postoji *Pickup and Delivery* verzija *vehicle routing* problema. *Pickup and delivery* razlikuje se od osnovnog problema po tome što agenti (dostavljači) ne kreću iz baze s robom za isporuku, već moraju najprije pokupiti određenu robu s nekog mjesta te ju dostaviti na neko drugo mjesto. Ovaj problem može se lako još dodatno zakomplicirati dodajući u njega uvjet LIFO (*Last In First Out*). Pozadina ovog uvjeta je također simulirana iz svijeta, naime najlakše je dostaviti (izvaditi) robu koja je na vrhu, odnosno koja je zadnja ubačena u vozilo.

Postoje mnogi drugi uvjeti koji se mogu nadodati na bilo koju verziju problema usmjeravanja vozila, primjerice ograničenje kapaciteta vozila. Ukoliko ograničimo kapacitet vozila moramo i definirati veličinu pojedinih paketa koje je potrebno isporučiti. Uvijek se može nadodati nekakav uvjet koji će zakomplicirati rješenje problema, primjerice određeni period u danu kada dostava mora biti obavljena na pojedinim lokacijama.

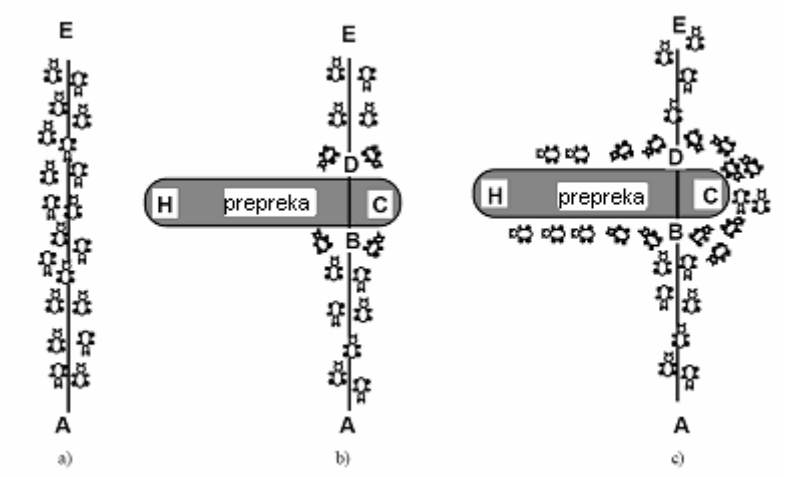
Za rješavanje *vehicle routing* problema uz mnoge dodatne uvjete potrebna je malo složenija programska potpora od one opisane u ovom radu.

3. Sustav Mrava

3.1 Mravi u prirodi

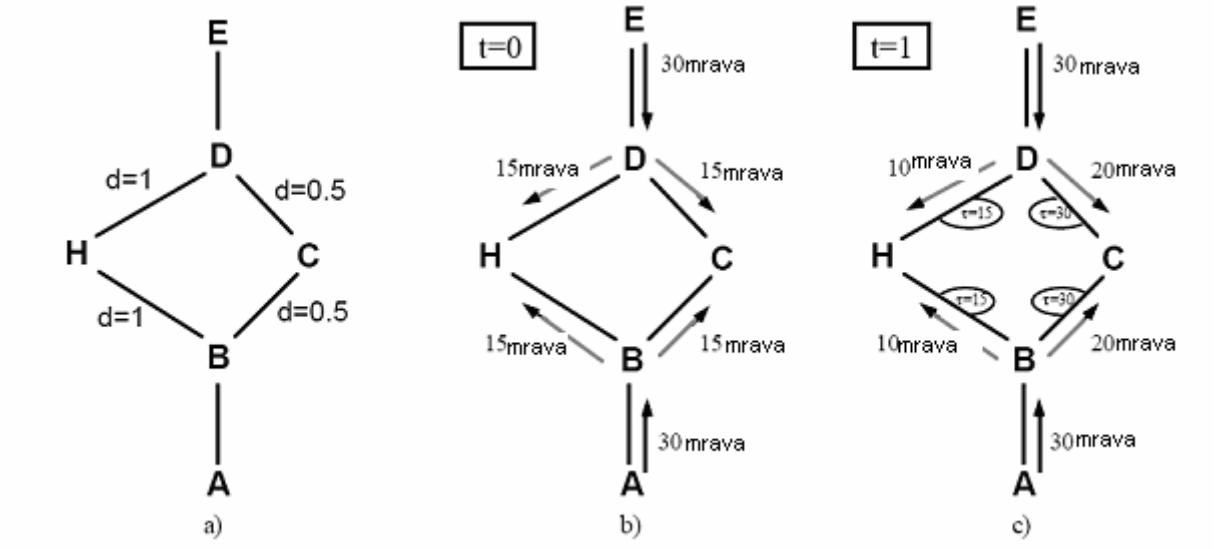
Način na koji mravi u prirodi pronalaze hranu je neko vrijeme bila nepoznanica, dok nije otkriveno da mravi kad pronađu hranu iza sebe ostavljaju trag feromona. Drugi mravi koji osjete feromone prate ih do hrane i vraćaju se do mravinjaka. Ukoliko negdje osjete jači trag feromona radije idu za njim nego za onim tragom koji je slabiji. To ima dva razloga, jedan je isparavanje feromona. Dakle što više vremena je prošlo od postavljanja feromona to je trag slabiji, što se dobro uklapa u logiku pronalaženja, jer ona hrana koja je više udaljena od mravinjaka na taj način ima manju šansu da bude praćena u odnosu na neku bližu. Drugi razlog razlike među tragovima feromona je broj mrava koji su ga postavili. Što više mrava se prethodno odlučilo za jedan put to je veća vjerojatnost da je put dobar te ga i ostali mravi prate.

Ukoliko se na putu pojavi prepreka koja postojeći put dijeli na dva nejednaka puta (*slika 3.1*), mravi na početku ne znaju koja strana je kraća. Stoga je vjerojatnost da će mravi krenuti na dulju i kraću stranu jednaka. Pošto je jedna strana kraća, na toj strani će trag feromona ubrzo postati jači, jer na kraćoj strani ne stigne ispariti koliko na duljoj strani. Pošto osjećaju da je trag feromona na jednoj strani jači više mrava se odlučuje za njega, te se ubrzo svi mravi odlučuju za kraći put.



Slika 3.1 – nastanak prepreke na mravljem putu

Primjer odabira puta među dva različita puta do istog cilja može se brojčano prikazati u vremenu na drugačiji način (*Slika 3.2: a) postojeće veze među "gradovima" i njihove duljine, geometrijska interpretacija prepreke sa slike 3.1; b) prosječno polovica dolazećih mrava ide duljim putem, a druga polovica kraćim; c) nakon što prvi mravi prođu kraćim putem i ostave trag feromona iza sebe, slijedeći mravi će vjerojatnije ići kraćim putem.*). Ukoliko se ključni dijelovi puta zamijene točkama brzo se dođe do slike koja podsjeća na *travelling salesman problem*.



Slika 3.2 - zaobilaženje prepreke kraćim putem

3.2 Umjetni mravi

Neka je zadani problem TSP (*travelling salesman problem*). Broj gradova koje treba obići je *n*, *m* je broj mrava ukupno u sustavu, a $b_i(t)$ je broj mrava u gradu *i* u trenutku *t*.

Mrav se odlučuje u koji grad *j* će sljedeće ići po funkciji koja kao parametre ima udaljenost gradova i trag feromona na njihovoj poveznici. Kako bi se izbjeglo obilaženje istih gradova više puta svaki mrav ima svoju tabu listu u koju stavi svaki grad koji je obišao. Kada tabu lista mrava obuhvati svih *n* gradova, mrav postavi feromone na svaki put kojeg je koristio za obilazak gradova te resetira svoju tabu listu.

Broj obilazaka koje će svaki mrav napraviti ograničen je zadanom vrijednosti.

Globalno gledajući svaki mrav se u trenutku *t* odlučuje gdje će biti u trenutku *t+1*, te se pri idućoj iteraciji koraka pomiče tamo. Na taj način svi će mravi nakon *n* iteracija proći kroz sve gradove (trenutak *t+n*) i postaviti trag. Pri tome se učitaju novi podatci za feromone na način da se stara vrijednost pomnoži faktorom isparavanja ρ (*faktor isparavanja* je zbunjujući naziv, on naime govori upravo suprotno, koliki dio traga NE isparava), te mu se nadoda trag svakog mrava koji je tuda prolazio (*formule 3.1* i *3.2*).

$$\tau_{ij}(t+n) = \rho \cdot \tau_{ij}(t) + \Delta\tau_{ij}$$

Formula 3.1 - obnavljanje (update) traga feromona na putu (i,j)

$$\Delta\tau_{ij} = \sum_{k=1}^{m} \Delta\tau_{ij}^{k}$$

Formula 3.2 – dodana vrijednost traga

Količina feromona koju mrav doda na neki put je 0 ako nije prošao tim putem. Ukoliko je mrav prošao tim putem, ostavlja neki određeni iznos traga ili obrnuto proporcionalan sa duljinom puta koju je prošao kao što je prikazano na *formuli 3.3*.

$$\Delta \tau_{ij}^{k} = \frac{Q}{L_k}$$

Formula 3.3 – trag pojedinog mrava

Na *formuli 3.3 Q* je zadana konstanta, a L_k ukupni put kojeg je *k*-ti mrav prešao za svoj obilazak svih gradova.

Pri odluci kamo da ide, osim traga feromona, mrav koristi i vidljivost. Vidljivost je parametar obrnut od udaljenosti. Što je grad udaljeniji to je vidljivost manja (vidljivost = 1/udaljenost).

Ukupni račun za odluku mrava prikazan je *formulom 3.4*.

$$p_{ij}^{k}(t) = \frac{\left[\tau_{ij}(t)\right]^{\alpha} \cdot \left[\eta_{ij}\right]^{\beta}}{\sum \left[\tau_{ik}(t)\right]^{\alpha} \cdot \left[\eta_{ik}\right]^{\beta}}$$

Formula 3.4 – vjerojatnost sljedećeg koraka

Ova formula (*3.4*) vrijedi u slučaju da mrav ima kamo za otići, odnosno njegova tabu lista ne obuhvaća sve postojeće gradove. η_{ij} predstavlja vidljivost između gradova *i* i *j*.

α i β su parametri koji upravljaju odnos važnosti traga u odnosu na vidljivost. Što je α veći to će mrav svoje odluke više bazirati na tragu feromona. Što je β veći, analogno tragu, mrav će pri donošenju odluke veću važnost pridijeliti vidljivosti gradova.

Rezultat $p_{ij}^{k}(t)$ predstavlja vjerojatnost da će mrav *k* krenuti iz grada *i* u grad *j* u trenutku *t*.

3.3 Algoritam mrava

Pri pokretanju algoritma najprije se stvore i postave gradovi, te tragovi (*trail*) feromona na ceste koje ih povezuju. Zatim se stvore i postave svi mravi. Pri postavljanju mrava stvore se tabu liste, mravi se postave u početni grad (svaki mrav ima svoj početni grad, može biti i više mrava ili nijedan mrav u nekom gradu) te se taj početni grad stavi kao prvi član tabu liste.

Zatim počinju iteracije kretanja mrava. Nakon n iteracija svi mravi imaju pune tabu liste te se postavlja trag feromona svakog mrava. Mravi se nanovo postave u neki početni grad i tabu liste im se isprazne (odnosno ostane samo novi početni grad). Takav ciklus od n iteracija se ponavlja toliko puta dok korisnik ne prekine rad programa ili program dostigne maksimum dozvoljenih ciklusa.

Pri pokretanju potrebno je postaviti (inicijalizirati) program:

```
brojac_ciklusa = 0;
Najbolja_Ruta = [maximalna vrijednost]
postaviti n gradova (učitati iz datoteke ili radne memorije)
postaviti Trag[i,j] na početnu vrijednost c za svaki i i j od 1 do n;
stvori m mrava
```

Zatim se vrti određeni broj ciklusa:

```
Dok (brojac_ciklusa < MAX_broj_ciklusa){
    brojac_ciklusa++;
```

```
            Odradi_Ciklus();
            Provjeri_Najbolju_Rutu();
            Ispari_Trag();
            Nadodaj_Trag();
            Resetiraj_Mrave();
        }
```

Funkcija Odradi_Ciklus() izgleda ovako:

```
        za svaki i od 0 do n{
            Za svakog mrava k od 1 do m{
                k.Sljedeći_Korak();
            }
        }
```

Funkcija Provjeri_Najbolju_Rutu() radi sljedeće:

```
        Za svakog mrava k od 1 do m{
            Ako je (Najbolja_Ruta > k.Duljina_Rute){
                Najbolja_Ruta = k.Duljina_Rute;
            }
        }
```

Funkcija Ispari_Trag samo množi sve elemente matrice Trail sa faktorom isparavanja, a funkcija Nadodaj_Trag prolazi kroz sve mrave, te za svaki njihov korak dodaje određen iznos traga u matricu Trail na odgovarajuće mjesto.

Funkcija Resetiraj_Mrave briše Tabu liste svih mrava i vraća ih u svoje početne gradove.

Složenost pojedinih koraka se razlikuje, no uz malo promatranja i uspoređivanja moguće je odrediti složenost najsloženijeg dijela algoritma (funkcija Odradi_Ciklus). Ako broj maksimalno dozvoljenih ponavljanja ciklusa označimo sa *NC*, broj mrava sa *m*, a broj gradova sa *n*, tada je složenost algoritma $O(NC \times m \times n^2)$. NC je petlja koliko puta se sve unutra odvije, n je broj prolaza *for* petlje u funkciji Odradi_Ciklus, m je broj mrava čiji se korak poziva, te još jednom ×n zbog računa vjerojatnosti (sume) unutar funkcije Sljedeći_Korak svakog mrava; čime se dobije $NC \times n \times m \times n$.

Ovaj okvir izrade mrava nije obavezan. Algoritam programa smije odstupati od ovdje prikazanog, na primjer modeli algoritma *Ant density* (gustoća mrava) i *Ant quantity* (brojnost mrava). U ta dva modela se trag feromona ne obnavlja u koraku 5 (na kraju ciklusa), već pri svakom koraku iz grada *i* u *j*. Time se postiže da već u prvom ciklusu mravi međusobno profitiraju od traga feromona. Razlika između ta dva algoritma je formula po kojoj se računa količina feromona koju mrav postavlja. *Ant density* postavlja puni trag Q (zadana konstanta, iznos feromona kojeg mrav postavlja na put) na putu (i,j) pri svakom koraku, dok *Ant quantity* postavlja Q/d_{ij}, čime se postiže veća ovisnost o duljini samog puta.

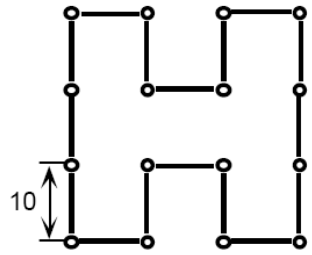
4. Sustavi mrava u pokusima

4.1 Broj mrava

Pokusi i mjerenja opisana u ovom odjeljku napravljeni su od skupine autora kao usporedba podvrsta ACO sustava. Učinjena su mjerenja na mravljem sustavu ciklusa, *ant density* i *ant quantity*, opisanim u poglavlju 3.3.

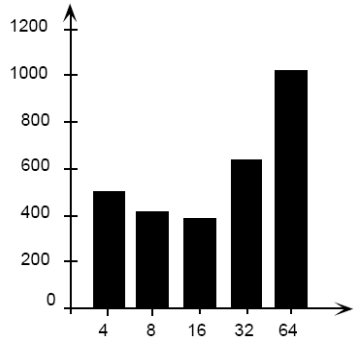
Broj mrava m, koje treba stvoriti u nekom programu varira o problemu kojeg treba riješiti. Veći broj mrava će zasigurno dati brže rješenje u prirodi, no ukoliko promatramo računalo kao stroj konačnih mogućnosti, tada veći broj mrava više opterećuje procesor i vrijeme potrebno da svi mravi obave jedan ciklus raste. Stoga je pri računu koliko je mrava idealno imati potrebno promatrati zbroj ciklusa koje pojedini mrav mora proći da ukupni algoritam dobije ispravno (najtočnije) rješenje.

Kao pokus zadana je 4×4 simetrična geometrijska mreža (*slika 4.1*) čija su optimalna rješenja dobro poznata.



Slika 4.1 - 4×4 mreža od 16 gradova

Rezultat pokusa je prikazan na *slici 4.2*. Ukoliko promatramo računalnu snagu kao nešto konačno u vremenu, tada je vrijeme izvođenja algoritma jednaka vremenu potrebnom za jedan mravlji korak pomnoženom s brojem mravljih koraka. Dok prvi faktor ovisi o računalnoj snazi *hardware*-a drugi faktor je algoritamski. Sa *slike 4.2* lako je očitati da je najmanji broj mravljih ciklusa bio potreban kod broja mrava 16 (najniži stupac -> najmanje koraka mrava).



Slika 4.2 - Ovisnost broja mravljih koraka o broju mrava

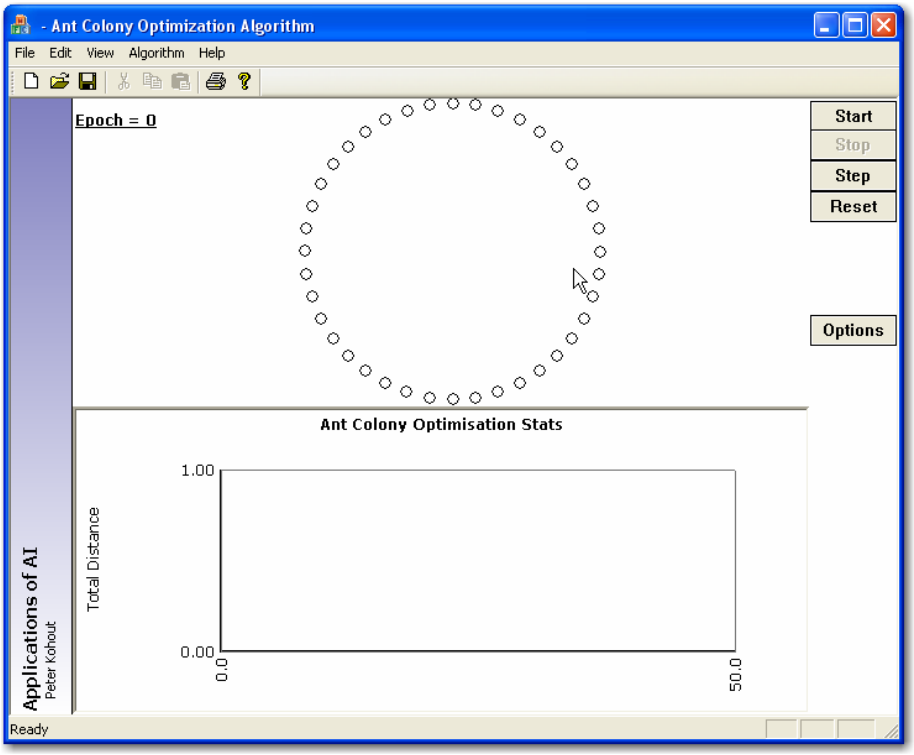
Gotovo identičan pokus izveden je još nekoliko puta, uz promjenu broja gradova (rešetke 5×5, 6×6 i 7×7). Svaki pokus dao je različit rezultat, naime optimalan broj mrava kod rešetke 5×5 bio je 25, kod 6×6 bio je 36, a kod rešetke 7×7 gradova optimalan broj mrava je bio 49. Time je donesen zaključak da je optimalan broj mrava za rješavanje *Travelling salesman problem*-a jednak broju gradova u problemu.

4.2 *Applications of AI*

Na internetu ima mnogo besplatno dostupnih programa koji demonstriraju upotrebu ACO sustava. Jedan od takvih programa je *"Applications of AI"*. Autor tog programa je Peter Kohout, a program je namijenjen demonstraciji optimizacije problema putujućeg putnika preko genetičkog algoritma i ACO algoritma, te njihovoj usporedbi.

Program je dostupan u *source code* obliku kao Visual studio projekt, napisan u C++ jeziku.

Radno sučelje programa prikazano je na *slici 4.3*.



Slika 4.3 Radno sučelje programa Applications of AI

U izborniku *"Algorithm"* moguća su dva izbora, naime GA i ACO. Preostale funkcije izbornika nisu potrebne, odnosno prazne su (ne čine ništa, vjerojatno samo ostatak šablone windows-framework iz Visual studia). Radna je površina podijeljena u okvirno 4 dijela:

1. Funkcijske tipke na desnoj strani radne površine

2. Lista informacija o trenutnom pokusu optimizacije (lijevo, bez pokrenute animacije to je samo jedan red)

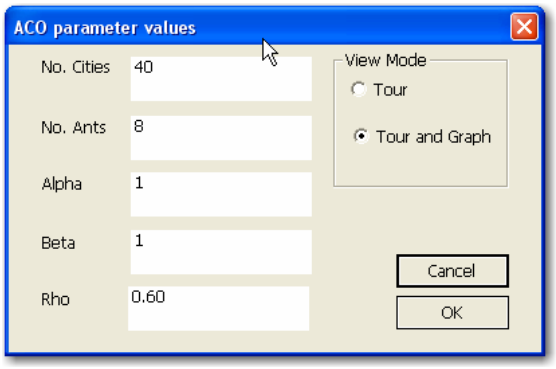
3. Skica kretanja mrava po gradovima (gore sredina). Gradovi su posloženi u krug, stoga je očito optimalno rješenje kružnica

4. Graf koji uspoređuje najbolju rutu dotad pronađenu s poznatom optimalnom (dolje)

4.2.1 Funkcijske tipke

Ponuđeno je 5 tipki na radnoj površini, 4 za pokretanje pokusa i jedan za kontrolu nad parametrima. Odabirom funkcije "Options" otvara se prozor prikazan na slici 4.4 kojim se upravlja svim slobodnim parametrima algoritma. Moguće je odrediti broj gradova zadanih u

TSP-u kao i broj mrava. Parametri alpha i beta odnose se na važnost traga (α) i vidljivosti (β) u mravljoj formuli odluke za idući korak. Rho je faktor isparavanja traga feromona (ρ). U view mode dijelu može se isključiti graf (4 dio *slike 4.3*)



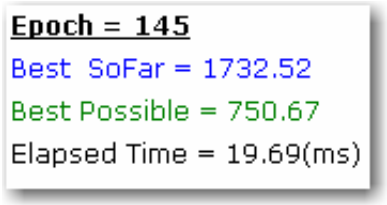
Slika 4.4. - Kontrola parametara izborom tipke Options

Vrijednosti postavljene pri pokretanju programa definirane su u pojedinoj datoteci projekta i lako se mogu promijeniti. Za ACO algoritam početne vrijednosti su: 40 gradova, 8 mrava, α i β jednaki 1, te faktor isparavanja postavljen na 0.6.

Pritiskom na tipku *start* pokreće se simulacija prema postavljenim parametrima. Simulacija se vrti dok algoritam ne pronađe optimalno rješenje ili dok korisnik ne zaustavi simulaciju. Zaustavljanje je moguće pritiskom tipke *stop* ili *reset*. *Stop* tipka zapravo samo pauzira simulaciju, dok ju *reset* zaustavi i vrati u početno stanje. Tipka *Step* vrti jedan korak mrava (radi samo dok je simulacija zaustavljena) te opet stane u izvođenju.

4.2.2 Lista informacija

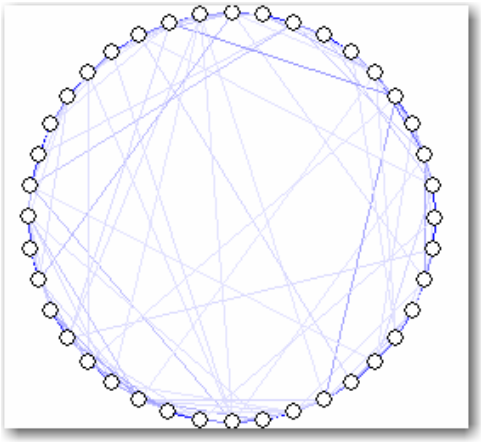
Prije pokretanja simulacije lista je prazna (prikazuje tekući broj ciklusa, koji je 0), čim se simulacija pokrene otvore se 3 dodatna reda prikazana na *slici 4.5*. Epoch je brojač vremena, "Best SoFar" je najbolja ruta koju je algoritam dosad našao, "Best Possible" je poznati optimum rješenja ovog problema, a "Elapsed Time" je pokušaj da se prikazuje realno vrijeme koliko dugo algoritam radi, no ta funkcija prikazuje vrlo krive vrijednosti. Funkcija *step* mijenja *Epoch* (brojač vremena) svakom primjenom za 1, sukladno time ažurira i ostale vrijednosti koje se mijenjaju. Bojama su naznačene najbolje rute, plavom izmjerena algoritmom, a zelenom konstanta koja predstavlja poznato optimalno rješenje.



Slika 4.5 – prikaz rezultata

4.2.3 Skica gradova

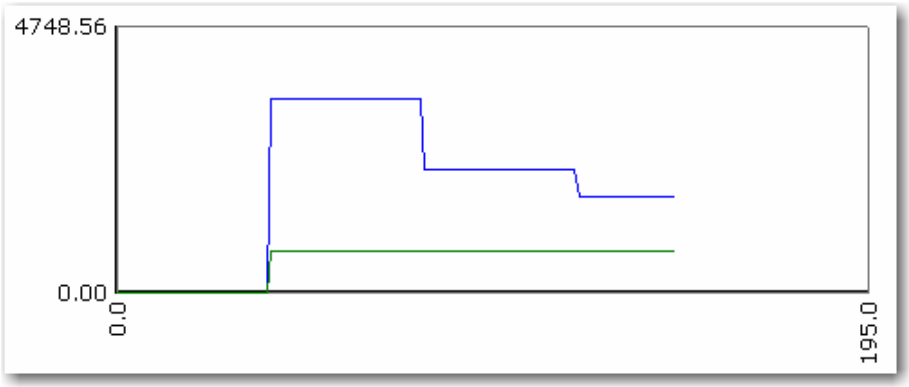
Slikom n gradova predočen je problem zadan mravima da ga riješe (*slika 4.6*). Plavim crtama među gradovima predstavljene su rute koje su mravi izabrali u trenutnom ciklusu. Što je crta deblja to je više mrava prošlo njome.



Slika 4.6 - Skica 40 gradova i mravljih ruta među njima

4.2.4 Graf najbolje rute

Vrijednosti iz liste informacija prate se i bilježe u radnu memoriju programa. Iste vrijednosti koriste se za iscrtavanje grafa (*slika 4.7*) dok traje simulacija. Na horizontalnoj osi je vrijeme, a na vertikalnoj je duljina (put prijeđen) najbolje rute pronađene do tada. Na svakom grafikonu su slike 2 funkcije, plava je najbolja nađena ruta dosad, a zelena je horizontalna linija koja je konstanta najbolje moguće rute za zadani problem.



Slika 4.7 Graf najbolje pronađene rute u vremenu

4.3 Mjerenja

Kako bi se dobili i usporedili statistički podatci optimizacije potrebno je provesti mnogo pokusa uz sustavno mijenjanje parametara.

Parametri na raspolaganju su: n (broj gradova), m (broj mrava), α (važnost traga), β (važnost vidljivosti) i ρ (faktor isparavanja).

U *tablici 4.1.* prikazani su rezultati mjerenja prema unaprijed postavljenim parametrima za ACO algoritam. Poznati optimum za taj zadatak je 634.15. Pokusi su prekinuti najkasnije nakon 1500 koraka (jer se u većini slučajeva nakon 1000-1200 koraka (300 ciklusa) rute više ne mijenjaju, tako da mravi ne uspijevaju pronaći bolji put), ukoliko algoritam pronađe optimalno rješenje prije toga automatski se sam zaustavi. Prema izmjerenim vrijednostima

vidimo da je od 5 pokušaja algoritam u samo jednom uspio naći optimum, a u ostalima je zapeo na neoptimalnim rješenjima.

Tablica 4.1: Mjerenja s inicijalnim parametrima programa

n	m	α	β	ρ	Best SoFar	Best Possible	Koraka
40	8	1	1	0.6	667.27	634.15	1500
40	8	1	1	0.6	726.57	634.15	1500
40	8	1	1	0.6	696.26	634.15	1500
40	8	1	1	0.6	788.51	634.15	1500
40	8	1	1	0.6	**634.15**	634.15	360

Zbog prirode zadatka kakav je zadan algoritmu, sasvim je očito da će program prije naći rješenje ako veću važnost pridijeli udaljenosti među gradovima nego postojećim tragovima. U *tablici 4.2* prikazani su rezultati pokusa uz postepeno mijenjanje omjera α i β sa 1:1 na 2:3, 3:5, 1:2, 1:3, 1:4, 1:5.

Tablica 4.2: Mjerenja uz veću važnost vidljivosti

n	m	α	β	ρ	Best SoFar	Best Possible	Koraka
40	8	2	3	0.6	**634.15**	634.15	80
40	8	2	3	0.6	**634.15**	634.15	80
40	8	2	3	0.6	**634.15**	634.15	80
40	8	3	5	0.6	**634.15**	634.15	40
40	8	3	5	0.6	**634.15**	634.15	40
40	8	3	5	0.6	**634.15**	634.15	40
40	8	1	2	0.6	**634.15**	634.15	80
40	8	1	2	0.6	**634.15**	634.15	80
40	8	1	2	0.6	**634.15**	634.15	80
40	8	1	3	0.6	**634.15**	634.15	40
40	8	1	3	0.6	**634.15**	634.15	80
40	8	1	3	0.6	**634.15**	634.15	40
40	8	1	4	0.6	**634.15**	634.15	40
40	8	1	4	0.6	**634.15**	634.15	40
40	8	1	4	0.6	**634.15**	634.15	40
40	8	1	5	0.6	**634.15**	634.15	40
40	8	1	5	0.6	**634.15**	634.15	40
40	8	1	5	0.6	**634.15**	634.15	40

Kao što se i očekivalo, primjećuje se drastično poboljšanje uspjeha algoritma ukoliko mu povećamo ovisnost o vidljivosti (udaljenosti gradova). Naime, uzimajući u obzir da 40 koraka predstavljaju jedan ciklus, prema vrijednostima u *tablici 4.2* vidljivo je da je većina pokušaja završila pronalaskom idealnog rješenja već u prvom ciklusu, a preostali u drugom.

Ukoliko bi gradovi bili drugačije postavljeni mravi bi eventualno imali više koristi od traga feromona nego vidljivosti. Pokusima prikazanima u *tablici 4.3* provjereno je da li je za ovakav problem rješenje moguće pronaći samim tragom.

Tablica 4.3: Mjerenja uz veću važnost traga

n	m	α	β	ρ	Best SoFar	Best Possible	Koraka
40	8	3	1	0.6	1279.73	634.15	2500
40	8	3	1	0.6	1190.00	634.15	2500
40	8	5	1	0.6	1160.25	634.15	2500
40	8	5	1	0.6	1538.43	634.15	2500
40	8	1	0	0.6	3677.03	634.15	3000

40	8	1	0	0.6	4046.34	634.15	3000

Očito je da naglaskom na trag mravi gube važnost najbližeg grada koji je u ovom zadatku optimalno rješenje. U pokusima bez osjetljivosti na udaljenost (beta = 0) ACO sustav nije dao niti približan rezultat optimalnom, jer mravi koji su postavili prvi najkraći put, postavili su i najjači trag feromona, stoga pri odlučivanju mrav nema gotovo nikakvu samostalnost, već samo prati prošloga. Na taj način ne otkrivaju nova rješenja već samo gaze po istim putevima.

Pokušaj da se malo poveća uloga traga u algoritmu bez mijenjanja koeficijenta alpha i beta može se postići povećanjem faktora isparavanja. Takvi pokusi opisani su u *tablici 4.4*. Smanjenjem ρ bi trebao porasti utjecaj vidljivosti u odnosu na trag.

Tablica 4.4: Mjerenja važnosti faktora isparavanja

n	m	α	β	ρ	Best SoFar	Best Possible	Koraka
40	8	1	1	0.4	694.27	634.15	1500
40	8	1	1	0.4	695.09	634.15	1500
40	8	1	1	0.4	**634.15**	634.15	160
40	8	1	1	0.5	821.13	634.15	1500
40	8	1	1	0.5	697.18	634.15	1500
40	8	1	1	0.5	665.14	634.15	1500
40	8	1	1	0.7	759.13	634.15	1500
40	8	1	1	0.7	666.27	634.15	1500
40	8	1	1	0.7	696.22	634.15	1500
40	8	1	1	0.8	665.21	634.15	1500
40	8	1	1	0.8	667.27	634.15	1500
40	8	1	1	0.8	821.46	634.15	1500

Iz rezultata pokusa iz tablice 4.4 vidljivo je da faktor isparavanja ne utječe na rezultat tako jako kao sami α i β, no ipak utječe, jer zadatak teži rješenju pohlepnog algoritma (gdje se gleda samo blizina, a ne trag).

Tablicom 4.5 provjerene su tvrdnje iz poglavlja 4.1. Usporedba rezultata ovisno o broju mrava (*m*), gradova (*n*), uz konstantne ostale parametre (α = β = 1, ρ = 0.6).

Tablica 4.5: Ovisnost broja mravljih ciklusa o broju mrava

n	m	Best SoFar	Best Possible	Koraka	Ciklusa×mrava
30	10	**634.32**	634.32	120	40
30	10	759.43	634.32	1500	-
30	20	**634.32**	634.32	120	**80**
30	20	**634.32**	634.32	120	**80**
30	30	**634.32**	634.32	120	**120**
30	30	**634.32**	634.32	90	**90**
30	40	**634.32**	634.32	90	120
30	40	**634.32**	634.32	120	160
10	5	**625.27**	625.27	70	35
10	5	**625.27**	625.27	40	20
10	10	**625.27**	625.27	30	**30**
10	10	**625.27**	625.27	20	**20**
10	15	**625.27**	625.27	30	45

10	15	**625.27**	625.27	30	45
10	20	**625.27**	625.27	30	60
10	20	**625.27**	625.27	40	80

Prema podatcima iz *tablice 4.5* potvrđena je tvrdnja da je za računalo kao konačan stroj najisplativije koristiti broj mrava otprilike jednak broju gradova. Jedino mjerenje koje odstupa od ove tvrdnje je simulacija sa 30 gradova i 10 mrava. Naime u prvom pokušaju ove verzije algoritma, "posrećilo" se programu i pronašao je optimum mnogo brže nego kod ostalih pokušaja za isti problem (30 gradova). No drugi pokušaj iste verzije nije uopće pronašao optimum rješenja, stoga nikako ne dolazi u obzir da je to najisplativije za računalo.

4.4 Usporedba sa Genetičkim algoritmom

Applications of AI nudi i opciju rješavanja optimizacijskog problema genetičkim algoritmom (GA). U *tablici 4.6* prikazani su rezultati optimizacije genetičkim algoritmom u usporedbi s ACO rješenjima. Za mjerenja dano je otprilike 2000 koraka ACO algoritmu, i jednako toliko GA algoritmu. Za parametre ACO sustava uzeti su $\alpha=1$, $\beta=1$, $\rho=0.6$, $m=n$.

Tablica 4.6: Usporedba GA sa ACO

n gradova	Best possible	GA best	GA time	ACO best	ACO time
20	632.88	**632.88**	471ms	**632.88**	10.24ms
20	632.88	1079.64	1411ms	**632.88**	13.31ms
20	632.88	1046.68	1476ms	**632.88**	10.47ms
40	634.15	1352.67	1631ms	**634.15**	103.13ms
40	634.15	1442.30	1661ms	**634.15**	102.73ms
40	634.15	1393.61	1773ms	**634.15**	103.01ms
50	634.74	2093.10	1803ms	**634.74**	200.05ms
50	634.74	2610.51	1776ms	**634.74**	200.55ms
50	634.74	2037.20	1860ms	**634.74**	197.30ms

Uzimajući u obzir da ACO algoritam nije radio po očito najboljem principu ($\beta>\alpha$), uspoređivanjem rješenja iz tablice 4.6 možemo doći do dva moguća zaključka:

1: GA algoritam programa je mnogo lošije napravljen od ACO algoritma

2: ACO algoritam je mnogo povoljniji za rješavanje TSP nego mnogi drugi načini rješavanja.

Ukoliko se postavi broj mrava jednak broju gradova, ACO algoritam daje optimalno rješenje u vrlo kratkom vremenu. GA algoritam, kao i ACO i mnogi drugi evolucijski algoritmi, ima svoje jake i slabe strane. Neki optimizacijski problemi bili bi brže i bolje riješeni GA algoritmom nego ACO, no na primjeru TSP sa kružno postavljenim gradovima zlatna medalja pripada mravima.

5. Projektiranje ACO sustava

5.1 Ideja

Najprije je potrebno definirati što program uopće treba raditi. Za ovaj projektni program zadatak je da može riješiti osnovne verzije *travelling salesman* i *vehicle routing* problema opisane u drugom poglavlju ovog rada. Program bi trebao imati mogućnost mijenjanja parametara prema kojima će optimizirati zadani problem. Problem će programu biti zadan na četiri moguća načina. Preko ulazne klase se odvija svako učitavanje gotovog problema, postoje verzije: čitanje iz datoteke, preko slučajnog generatora ili unaprijed zadan i upisan problem. Četvrti način je stvaranje problema za vrijeme izvođenja programa kroz korisničko sučelje. Izlaz programa treba biti u prozoru programa.

Algoritam programa će raditi po principu prikazanome u odjeljku 3.3. Nepromjenjivi parametri optimizacije su početna vrijednost traga na svim putovima i formula za dodavanje traga. Početna vrijednost traga (Q) je 10. Formula unutar mrava za dodavanje traga je $\tau = \tau + \frac{10 \cdot n}{D}$, pri čemu je τ iznos traga na pojedinom putu, n broj gradova u aktivnom problemu, a D duljina ukupnog puta kojeg je mrav prešao.

5.2 Ulaz i izlaz

Ulaz za program biti će omogućen na više načina. Za učitavanje određenog gotovog problema u program potrebno je učitati željenu datoteku preko opcije "*Load*". Za dobro definiran problem potrebno je najprije navesti tip problema (TSP ili VRP) te navesti sve gradove. Za gradove potrebno je navesti 2 pozitivna broja, x i y koordinatu u koordinatnom sustavu. U slučaju da se želi riješiti VRP potrebno je dodatno definirati i koliko je vozila (dostavljača) na raspolaganju za rješenje.

Poželjna je i mogućnost da se parametri mogu mijenjati unutar programa kao što je to moguće u programu *Applications of AI* opisanom u poglavlju 4.

U *tablici 5.1* prikazani su oblici zapisa podataka u ulaznoj datoteci (3 grada sa koordinatama (0.1, 0.2), (0.256, 200) i (250, 500), te 7 vozila u VRP problemu), te primjer navođenja komentara (#) i ulaznih parametara (!).

Tablica 5.1: Primjer ulaznih podataka

TSP	VRP
0,1 0,2	!vozila 7
0,256 200	# & ovo %!/“(je $% sve „#% komentar
#ovo je komentar	0,1 0,2
250 500	0,256 200
!mrav 10	!beta 2,1
!alpha 1,2	250 500

Izlaz bi trebao sadržavati broj ciklusa obavljen u pokusu, realno vrijeme utrošeno na račun, najbolju rutu pronađenu, te zapis rednog broja ciklusa u kojem je pronađena najbolja ruta. Grafičko sučelje bi trebalo prikazivati zadani problem, najbolju rutu pronađenu među njima, te odvojeno od toga graf duljine najbolje rute u vremenu.

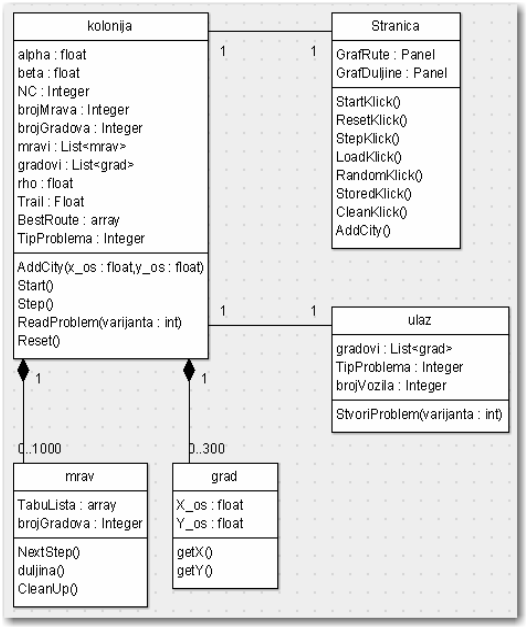
5.3 Realizacija

Program je nazvan *"SL ACO"*, prema Silverlight-u koji je korišten za izradu korisničkog sučelja. Program je realiziran pomoću okruženja *Visual Studio 2008* u jeziku C#, uz korištenje dodatka *Silverlight 2 toolkit*. *Silverlight* je dodan radi lakšeg grafičkog prikaza podataka i radnih opcija bez smetanja samim C# klasama koje provode optimizaciju. Još jedna prednost *Silverlight*-a je mogućnost da se program prikaže kao web stranica a optimizacija se provodi na računalu klijenta koji gleda stranicu (u suprotnom bi značilo veliko zauzeće procesora servera). Za pokretanje programa potrebno je imati instaliran Silverlight 2 dodatak za Internet pretraživače i windowse (Silverlight 2 toolkit je potreban za otvaranje source kod-a u Visual Studiju). Pri kompilaciji i pokretanju projekta program se otvara u obliku Internet stranice (*slika 5.1*).

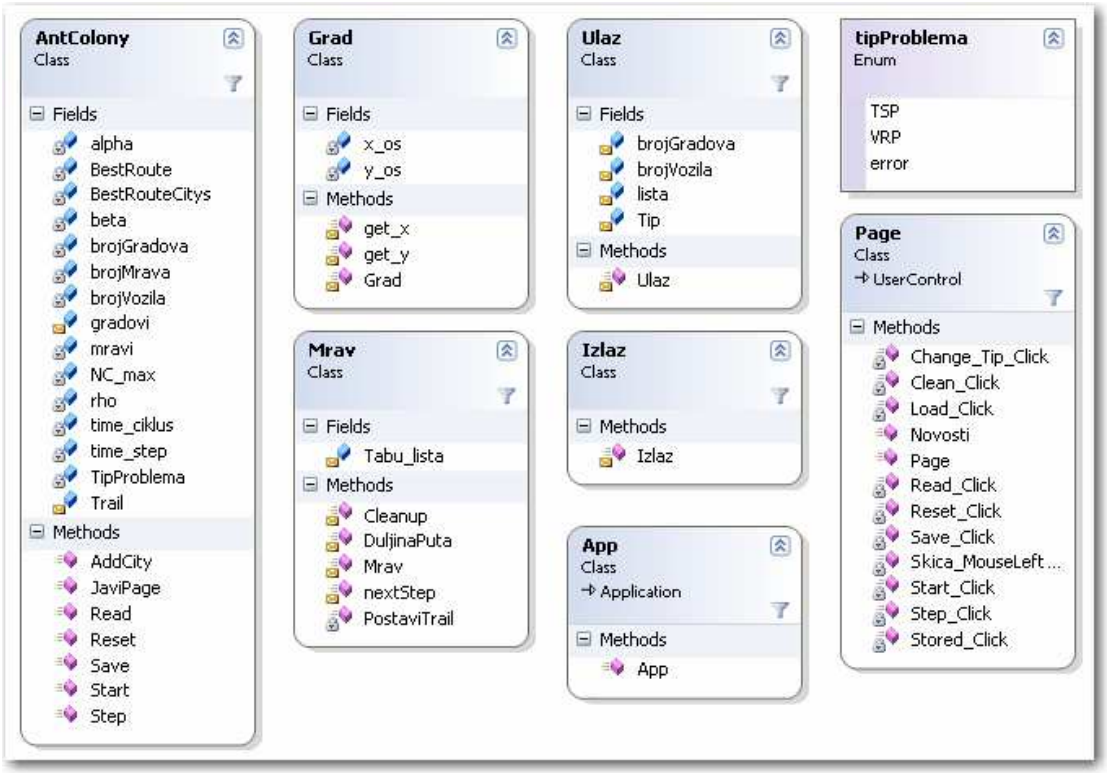
Slika 5.1. - Program SL_ACO

U pozadini programa je objektno realizirana kolonija mrava kojoj stranica (korisničko sučelje) šalje podatke od korisnika. Kolonija šalje povratne informacije stranici i ucrtava graf i mapu gradova na stranicu. *Slika 5.2 a)* prikazuje UML dijagram klasa napravljen prije početka programiranja. Ukoliko se prikaže dijagram klasa projekta, prepun je metoda i atributa koji služe lakšem čitanju koda funkcija unutar klasa. Na *slici 5.2 b)* prikazan je dijagram klasa sa kojeg su izbačene mnoge metode i atributi nebitni za shvaćanje principa rada programa (primjerice mnoge Get i Set funkcije).

Na *slici 5.2.* postoji razlika među dva dijagrama, koja je nastala pri implementaciji klasa u kodu. Primjerice nije bilo zamišljeno program ima opciju spremiti aktivni problem (klasa Izlaz).



Slika 5.2 a) UML dijagram klasa



Slika 5.2 b) dijagrami klasa bez nebitnih metoda i atributa

Klasa *AntColony* je srce cijelog programa. Ona se brine za komunikaciju optimizacijskog algoritma sa korisničkim sučeljem (Page), vrši optimizaciju, posjeduje mrave, sadrži zapis gradova optimizacijskog problema, crta problem i rješenja na stranicu, te je posrednik svakog komuniciranja među ostalim objektima u programu. Klasa AntColony je zaštićena *singleton* obrascem preko atributa instanca, čime je program osiguran da se neće stvoriti više od jednog objekta kolonije. Kolonija sadrži listu mrava, listu gradova (do 300 gradova), te sve parametre optimizacije i sve funkcijske varijable potrebne u algoritmu. Radi jednostavnosti

komunikacije među mravima i kolonijom, dio lista i nizova za optimizaciju su "internal" tipa zaštite, a preostale su "private" tipa, dostupni preko skupa metoda get i set.

Gradovi su objekti tipa "grad", no po svim svojim osobinama oni su zapravo točke (sadrže X i Y vrijednosti, te konstruktor koji prima 2 *double* vrijednosti (x i y)), osim što im se koordinate ne mogu mijenjati nakon što su stvoreni, već se samo mogu čitati preko internih metoda "get_x" i "get_y".

Mravi su objekti tipa "mrav" koji sadrže atribute važne za njihovo ispravno funkcioniranje. Najvažniji među njima je Tabu lista (internal zaštićenosti kako bi ju kolonija lakše kopirala u slučaju najbolje rute), preostale vrijednosti su samo implementirane radi lakšeg čitanja i razumijevanja koda, program bi mogao naime raditi i bez njih.

Osnovna metoda mrava je konstruktor koji prima parametre *n* i *poc*, *n* je broj gradova u aktivnom problemu, a *poc* je početni grad u kojem se mrav nalazi (za TSP se dodjeljuje slučajni početni grad, a za VRP je to prvi grad u listi). Za pojedine korake i odluku u koji će se grad premjestiti, svaki mrav ima metodu "NextStep". Kad se tabu lista mrava popuni, mrav ne može pronaći nijedan grad u koji će se pomaknuti, te ga NextStep vrati u početni grad (samo ukoliko nije već tamo). Kada završi jedan ciklus kolonija zove metodu DuljinaPuta svakog mrava te rezultat uspoređuje sa dotad pronađenima. Mrav unutar te metode automatski poziva privatnu metodu "PostaviTrail" kojom prema svojoj tabu listi postavlja trag u internu matricu Trail kolonije. Nakon svih proračuna i potrebnih kopiranja podataka, kolonija očisti sve mrave (vrati ih u njihove početne gradove i resetira tabu liste) pozivom metode "Cleanup".

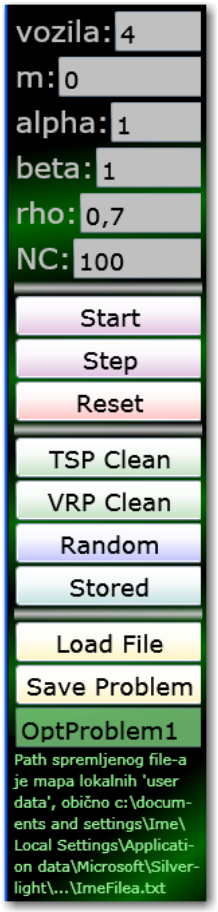
Klasa Ulaz se koristi pri učitavanju gotovog problema. To može biti iz određene tekstualne datoteke, generiranju slučajnog problema, ili stvaranju jednog od zapisanih problema. Ulaz ima konstruktor koji prima jedan parametar. Prema parametru stvori slučajan problem (kada je parametar 0), pokuša učitati problem (uz parametar 1) iz ulazne datoteke, a za parametar >1 Ulaz stvara jedan od zapisanih problema. Parametar 2 označuje 3 koncentrična kruga, 3 mrežu gradova, a za parametar 4 Ulaz stvara 30 gradova raspoređenih u slova ACO. Optimizacijski problem se iz objekta klase Ulaz čita preko njegovih internal vrijednosti i polja.

Klasa Izlaz služi za zapisivanje aktivnog problema u tekstualnu datoteku. Ima izlazne datoteke se upisuje u korisničko sučelje, a mapa u koju se datoteka pohranjuje se ne može mijenjati unutar programa. Ova mala neugodnost posljedica je Windows i Internet protokola i pravila za sigurnost. Naime kako bi se izbjegli internetski zločini i smanjila štetnost Internet stranica, omogućeno je samo pohranjivanje podataka u mapu dodijeljenu od Windows-a. Uobičajeno (u slučaju Windows XP) ta dodijeljena mapa nalazi se na lokaciji: c:\ Documents and Settings\ ImeKorisnika\ Local Settings (hidden/system mapa)\ Application Dana\ Microsoft\ Silverlight\ is\ pdimhioj\ Ofchesze\ 1\ s\ (neki code)\ ImeFile-a.txt, pri čemu imena nekih podmapa mogu varirati ovisno o računalu na kojem se program izvodi.

5.4 Korisničko sučelje

Sučelje za korištenje programa može se podijeliti u 4 dijela. Lijevi meni upravlja većinom funkcija i parametara programa. Desni meni služi za prikazivanje dobivenih rezultata i mijenjanje vrste problema (TSP/VRP). U sredini je skica gradova i najbolje nađene rute, a pri dnu ekrana je graf duljine najbolje rute u vremenu.

5.4.1 Lijevi meni



Slika 5.3 - Lijevi korisnički meni

Lijevi meni (*slika 5.3*) sadrži kontrole za učitavanje i stvaranje problema, te reguliranje parametara optimizacije. U trećem poglavlju ovog rada opisane su uloge pojedinih parametara. Na raspolaganju su parametri: broj vozila za VRP (0<brojVozila<300), broj mrava *m* (0<*m*<1000), važnost Traga *alpha* (0<=α<100), važnost vidljivosti *beta* (0<=β<100), faktor isparavanja *rho* (0<=ρ<=1) i broj ciklusa *NC* (1<*NC*<20000). Pri pokretanju programa vrijednosti se postave na početne vrijednosti brojVozila=4, m=10, α=1, β=1, ρ=0,7, NC=100. Pri učitavanju problema broj mrava se automatski promijeni na broj gradova(razlog tome je opisan u poglavlju 4.1).

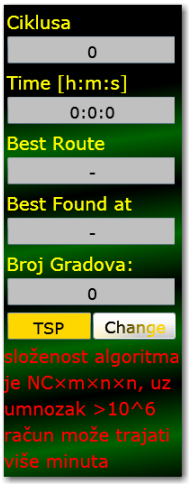
Gumb "Stored" nudi naizmjence jedan od tri problema: napisati ACO gradovima (30 gradova), pravilna mreža n×n ili tri koncentrična kruga. Broj gradova u mreži i krugovima varira među vrijednostima preporučljivim za demonstraciju algoritma. Pritiskom gumba *"Load"* otvara se prozor za biranje datoteka, te se učita problem iz označene datoteke. "Save" opcija pohranjuje aktivan problem na prije opisan način, uzimajući ime datoteke iz kutije teksta ispod gumba. Gumb *"Random"* daje slučajni problem za optimiziranje (slučajni su broj gradova, tip problema i lokacije gradova). Tipke *"VRP Clean"* i *" TSP Clean"* brišu listu gradova u koloniji, uz razliku da *VRP Clean* postavi tip problema na VRP, dok *TSP Clean* postavi TSP.

Gumb *"Start"* pokreće optimizaciju trenutno zadanog problema. Gumb *"Reset"* vraća koloniju u stanje prije pokrenute optimizacije (učitani problem ostaje). Opcija *"Step"* pokreće

optimizaciju do pronalaska sljedeće najbolje rute. Kada pronađe rutu bolju od trenutno najbolje, ili broj ciklusa pređe broj dozvoljenih ciklusa, optimizacija se zaustavlja. Zaustavljenu optimizaciju moguće je nastaviti gumbima *Start* ili *Step*, gdje *Start* dovrši optimizaciju do NC, a *Step* opet radi samo do iduće najbolje rute.

5.4.2 Desni meni

U desnom dijelu sučelja *(slika 5.4)* nalaze se prozori koji prikazuju rezultate mjerenja ili parametre problema koje korisnik ne može mijenjati unosom novih vrijednosti u prozor.

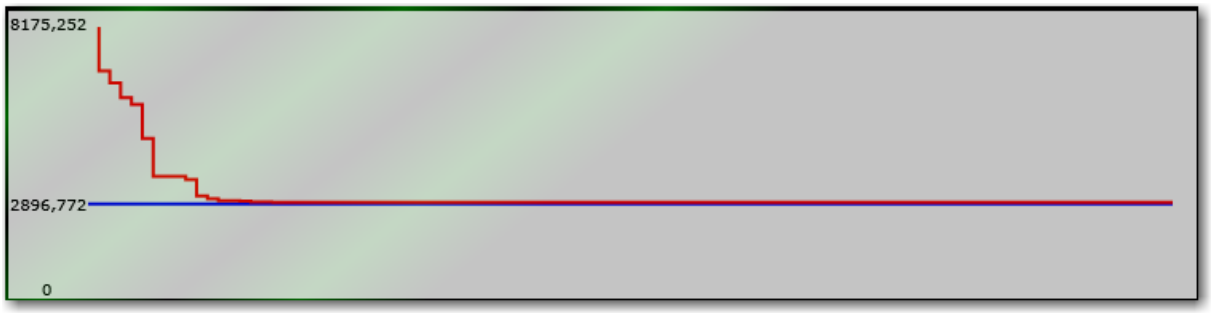


Slika 5.4 – Desni meni za prikaz rezultata optimizacije

Desni meni sadrži broj ciklusa koji je korišten do sad, vrijeme prošlo od početka optimizacije do kraja, duljina najbolje pronađene rute, redni broj ciklusa u kojem je najbolja ruta pronađena, broj gradova prisutnih u trenutno aktivnom problemu, tip aktivnog problema, gumb za promjenu problema (VRP/TSP) te upozorenje za korištenje programa s velikim brojkama. Upozorenje na sučelju programa služi kao podsjetnik na drugo i treće poglavlje ovog rada. Složenost ACO algoritma je naime $NC \times m \times n^2$, a računalo je ipak konačno brz stroj, te pri optimizaciji 100 gradova sa 100 mrava u 100 ciklusa modernom računalu treba više od minute da provede optimizaciju.

5.4.3 Graf najboljih ruta

Na dnu korisničkog sučelja *(slika 5.5)* programa *SL ACO* ugrađen je graf koji prikazuje duljinu najbolje pronađene rute u vremenu.

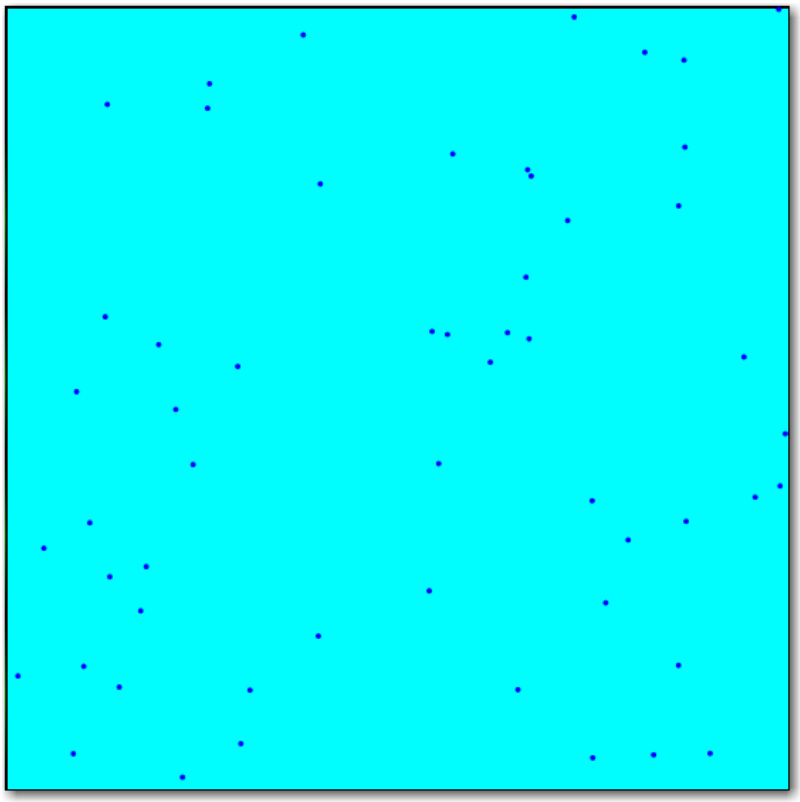


Slika 5.5 – graf duljina najboljih ruta u vremenu

Vertikalna os grafa prikazuje duljinu rute, a horizontalna vrijeme, odnosno broj ciklusa u kojem je pojedina ruta pronađena. U sustavu su prikazane 2 linije, crvenom bojom je naznačen graf najbolje nađene rute u vremenu, a plavom bojom je naznačena vodoravna linija najbolje rute pronađene u dosadašnjem tijeku optimizacije. Na lijevom kraju grafa nalaze se tri broja. Na dnu stoji uvijek 0, kao ishodište koordinatnog sustava na kojem se graf iscrta. Gornji broj prikazuje duljinu najbolje rute pronađene u prvom (nultom) ciklusu, što predstavlja najveću vrijednost ucrtanu na graf. Srednji broj prati plavu liniju, odnosno najbolju pronađenu rutu.

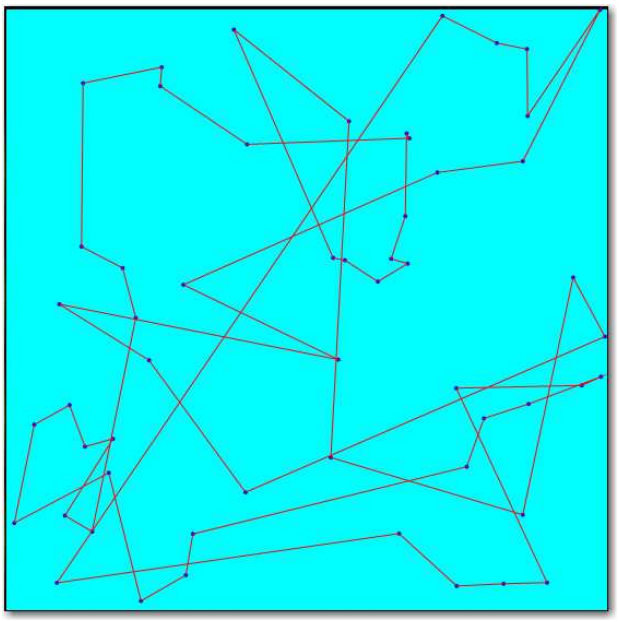
5.4.4 Mapa

Većina površine korisničkog sučelja je preostali element: mapa gradova i najbolje rute. Na polju se crtaju svi gradovi učitani u problem (*slika 5.6*). Pri bilo kakvoj promjeni u listi gradova cijeli graf se briše i nanovo crta. *Na slici 5.6* prikazan je izgled jednog slučajno stvorenog (TSP) problema koji sadrži 55 gradova.



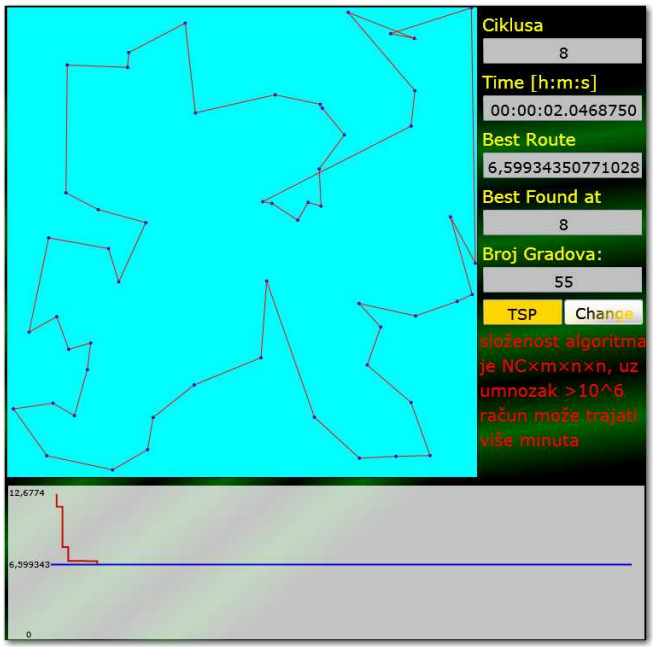
Slika 5.6 – 50 slučajnih gradova TSP-a

Nakon što je pokrenuta optimizacija (gumbom Start ili Step), na sučelju nije moguće ništa mijenjati dok se optimizacija ne završi. Pri završetku optimiranja kolonija stranici dojavi rezultate i ucrta najbolju pronađenu rutu na graf. Problem prikazan na *slici 5.6* pokrenut je tipkom Step, a dobiveni rezultat je prikazan na *slici 5.7*.



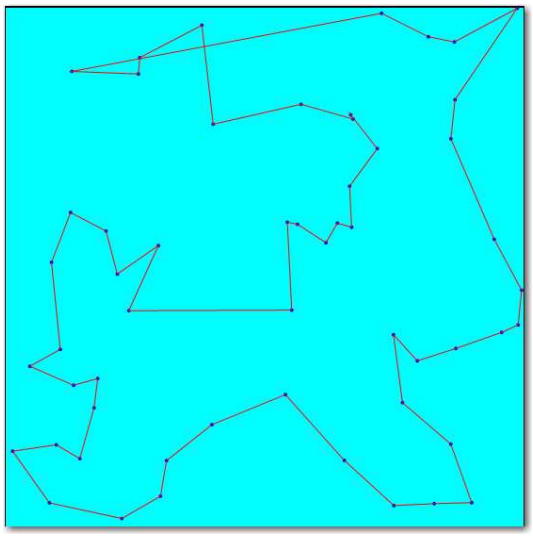
Slika 5.7 - Prva najbolja ruta pronađena za problem prikazan na slici 5.6

Ukoliko optimizaciju sa *slike 5.7.* nastavimo pojedinim koracima, nakon 4 koraka dobijemo rezultat prikazan na *slici 5.8*.



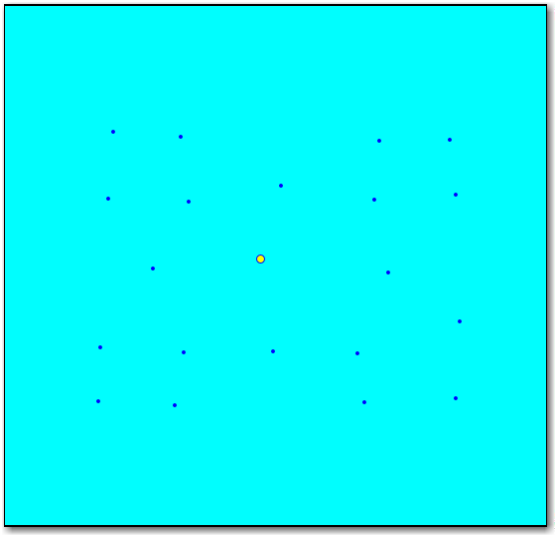
Slika 5.8 - Četvrta najbolja ruta pronađena za problem sa slike 5.6

Ukoliko u situaciji prikazanoj na *slici 5.8* pritisnemo gumb start, optimizacija nastavlja rad do zadanog broja ciklusa, te je konačan rezultat prikazan na *slici 5.9*.

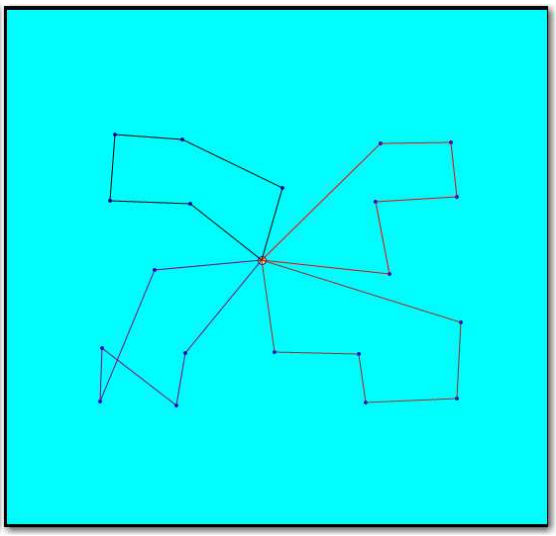


Slika 5.9 - Konačan rezultat pronađen za problem sa slike 5.6

Ukoliko se radi o VRP-u graf gradova i ruta, mapa izgleda malo drugačije. Naime početni grad je označen drugačije od ostalih. Vozila su obojena u 8 različitih boja; ukoliko ih ima više, boje se počinju ponavljati. Primjer izgleda VRP-a sa 4 vozila i njegovo rješenje može se vidjeti na *slikama 5.10* i *5.11*.



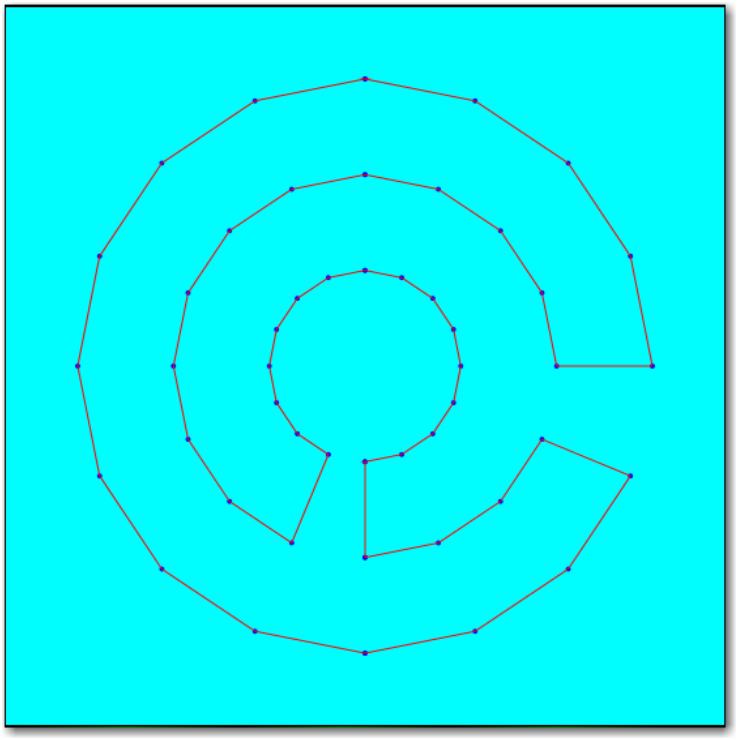
Slika 5.10 – VRP gradovi



Slika 5.11 – VRP rješenje

Osim prikazivanja gradova i ruta, mapa ima i još jednu funkciju. Naime, unutar programa SL ACO moguće je nacrtati vlastiti problem klik-anjem po grafu. Svaki klik dodaje jedan grad. Prvi grad koji je dodan se u slučaju VRP-a prima kao početni grad za sva vozila (mrave). Na taj način može se lako ispitati program željenim problemima (*slike 5.10* i *5.11* su tako napravljene).

Jedan od najpoznatijih TSP zadataka su gradovi postavljeni kao koncentrični krugovi, a poznato optimalno rješenje su kružnice prekinute na jednom mjestu gdje se ruta spušta ili podiže na idući koncentrični krug. Takav problem i njegovo rješenje prikazani su na *slici 5.12* (SL ACO je u 33-em ciklusu sa 10 mrava u roku od 4 sekunde pronašao prikazano rješenje). Problem je dobiven korištenjem *Stored* funkcije u programu SL ACO.



Slika 5.12 – TSP koncentrični krugovi

5.5 Mjerenja

Za demonstraciju programa koristi će se jedan unaprijed zadan problem (koncentrični krugovi) i jedan slučajni problem.

Za zapis rezultata koristiti će se slika najbolje rute i grafa duljine, te izmjerene vrijednosti iz desnog korisničkog menija.

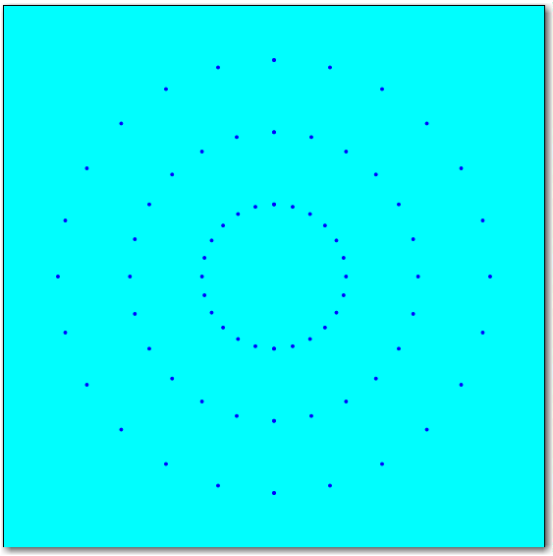
5.5.1 Krugovi

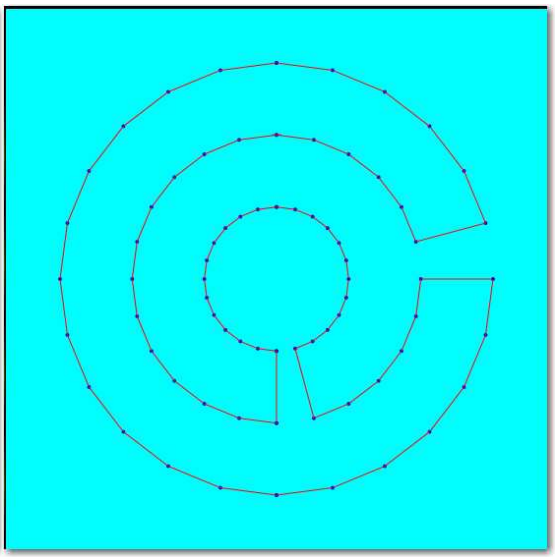
U *tablici 5.2* prikazani su rezultati mjerenja nad 3 koncentrična kruga sa 78 gradova. Izgled problema i njegovog optimalnog rješenja prikazani su na *slikama 5.13* i *5.14*. U tablici su parametri mjerenja (*m*, α, β, ρ, broj *ciklusa*) i rezultati mjerenja (*vrijeme* koliko dugo je optimizacija trajala h:m:s, *duljina* najbolje pronađene rute, redni broj ciklusa u kojem je *nađena* najbolja ruta).

Tablica 5.2: TSP koncentrični krugovi

m	α	β	ρ	Ciklusa	Vrijeme	Duljina	Nađena
78	1	1	0.7	100	0:1:0	**2897**	61
78	1	1	0.7	100	0:0:53	2907	46
78	1	1	0.7	100	0:1:03	2907	58
50	1	1	0.7	100	0:0:40	**2897**	66
50	1	1	0.7	100	0:0:40	2907	65
50	1	1	0.7	100	0:0:39	**2897**	50
25	1	1	0.7	100	0:0:20	**2897**	63
25	1	1	0.7	100	0:0:20	2907	58

25	1	1	0.7	100	0:0:20	2914	61
50	**3**	1	0.7	100	0:0:43	2907	9
50	**3**	1	0.7	100	0:0:39	2914	9
50	**3**	1	0.7	100	0:0:40	**2897**	10
50	**5**	1	0.7	100	0:0:38	2939	6
50	**5**	1	0.7	100	0:0:44	2940	13
50	**5**	1	0.7	100	0:0:44	2949	14
50	1	**3**	0.7	100	0:0:45	**2897**	18
50	1	**3**	0.7	100	0:0:43	**2897**	42
50	1	**3**	0.7	100	0:0:43	**2897**	17
50	1	**5**	0.7	100	0:0:53	**2897**	53
50	1	**5**	0.7	100	0:0:47	**2897**	20
50	1	**5**	0.7	100	0:0:43	**2897**	33
50	1	1	**0.5**	100	0:0:45	2907	42
50	1	1	**0.5**	100	0:0:43	**2897**	46
50	1	1	**0.3**	100	0:0:38	**2897**	83
50	1	1	**0.3**	100	0:0:48	**2897**	34
50	1	1	**0.9**	100	0:0:44	2996	85
50	1	1	**0.9**	100	0:0:42	3002	68





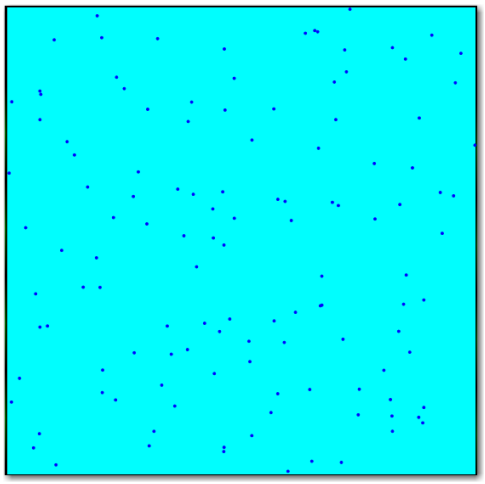
Slika 5.13 – 78 gradova u 3 koncentrična kruga *slika 5.14 – idealno rješenje slike 5.13*

Na *slici 5.14* je prikazano jedno od poznatih idealnih rješenja za zadani problem, čija je duljina 2897 (svejedno je gdje se prelazi iz jednog prstena u drugi, isti je rezultat za više različitih ruta). U *tablici 5.2* su podebljani parametri čiji se utjecaj isprobava. Dobivene vrijednosti pokazuju da α ubrzava okončavanje algoritma ("najbolja" ruta je pronađena već u prvih 10 ciklusa). Povećanje važnosti traga poveća broj mrava koji idu najboljim putem, čime raste i trag na tom putu, a time i vjerojatnost da će mravi idući ciklus ići istim putem. β je najvažniji parametar pri rješavanju ovog problema, jer se njegovim povećanjem ubrza pronalazak najbolje rute i poboljša vjerojatnost da će optimalna ruta biti pronađena. ρ bi smio biti i manji od 0.7, no ne i veći. Na bilo koju stranu (<0.3 ili >0.8) ekstreman ρ kvari rezultate mjerenja.

5.5.2 Slučajni problem TSP

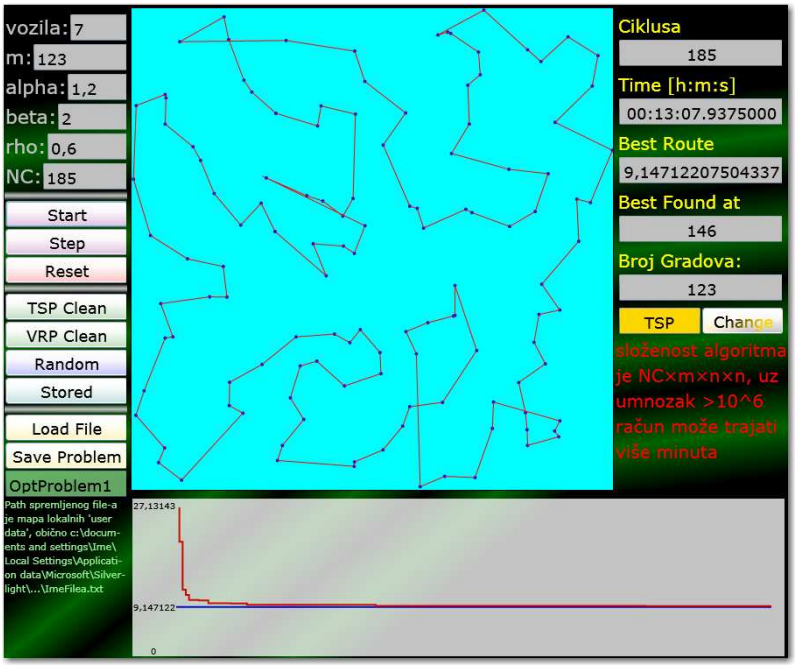
Nakon mnogih testova zaključeno je da SL ACO daje najbolje rezultate (najkraće rute za najmanje vremena) uz parametre ρ=0.5-0.7, α=0.5-1.5, β=1.5-4, m=n, NC=1.5×n. Ukoliko bilo koji od navedenih parametara izlazi iz svojeg područja (iznad ili ispod) rezultati optimizacije se kvare. Iz navedenih razloga su parametri pri sljedećoj optimizaciji iznosili: ρ=0.6, α=1.2, β=2. *Slika 5.15* prikazuje jedan slučajan problem dobiven gumbom *Random*.

Problem sadrži 123 grada, broj mrava je također 123, a tip problema je TSP.



Slika 5.15 - Slučajni TSP sa 123 grada

Na *slici 5.16* prikazani su rezultati pokretanja optimizacije problema prikazanog na *slici 5.15*. Uz spomenutu složenost algoritma, optimizacija je trajala 13 minuta (složenost algoritma je ovdje 185×123^3, što znači da jedan mravlji korak traje 2.265μs (10^{-6}s)) na AMD Athlon 64bit 3500+MHz procesoru.

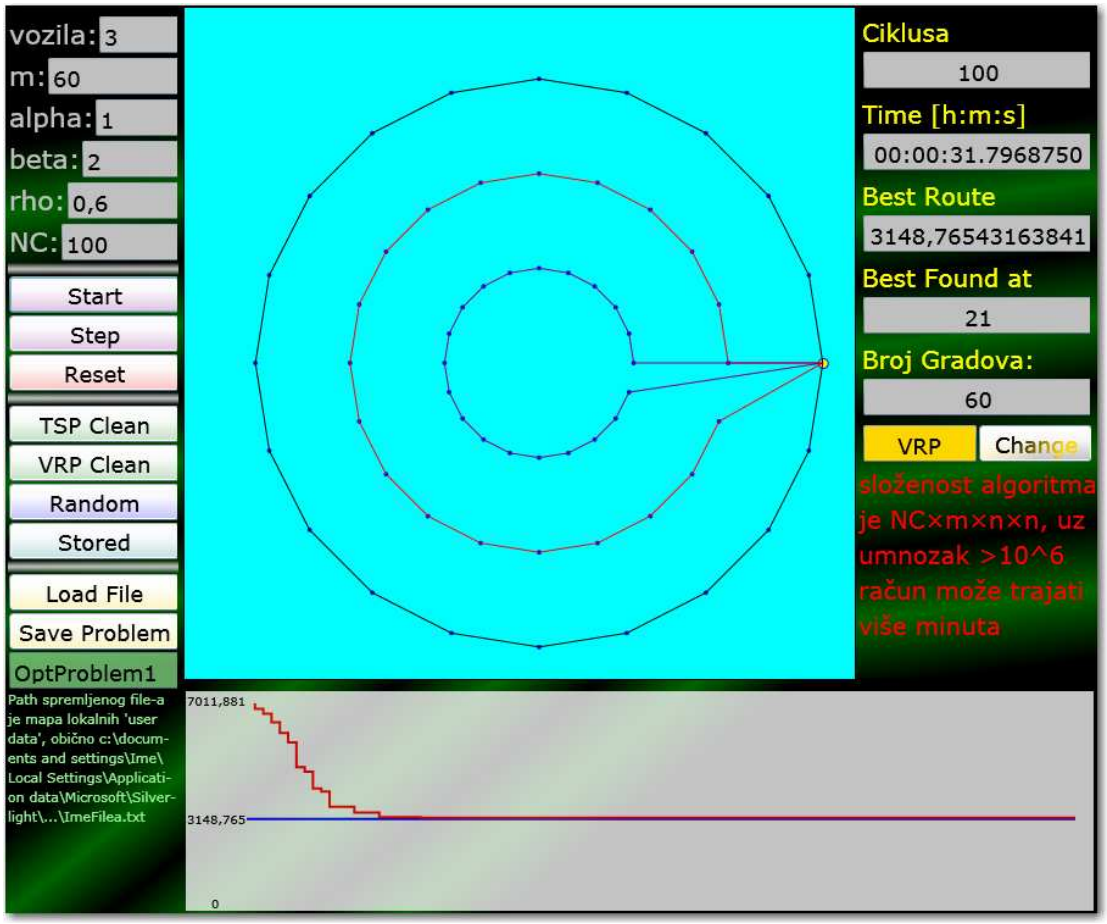


Slika 5.16 - Pronađena najbolja ruta za problem sa slike 5.15

Napomena: pronađena najbolja ruta nije optimum. To se na ovakvom grafičkom prikazu za TSP lako može prepoznati po sjecištima rute. Ukoliko se na bilo kojem mjestu ruta siječe, sigurno postoji bolja. Tek ukoliko se ruta ne siječe nigdje na grafu, je moguće da je pronađen optimum. Za sigurnu potvrdu bio bi potreban deterministički algoritam.

5.5.3 VRP Krugovi

Prije već prikazan problem sa 3 koncentrična kruga ima također i dobro poznatu varijantu u problemu usmjeravanja vozila. Naime ukoliko krugovi imaju isti broj točaka (gradova), a broj vozila se postavi na broj krugova, tada je optimalno rješenje da svako vozilo obiđe jedan od tih krugova. Takav problem i njegovo rješenje prikazan je na *slici 5.17*.

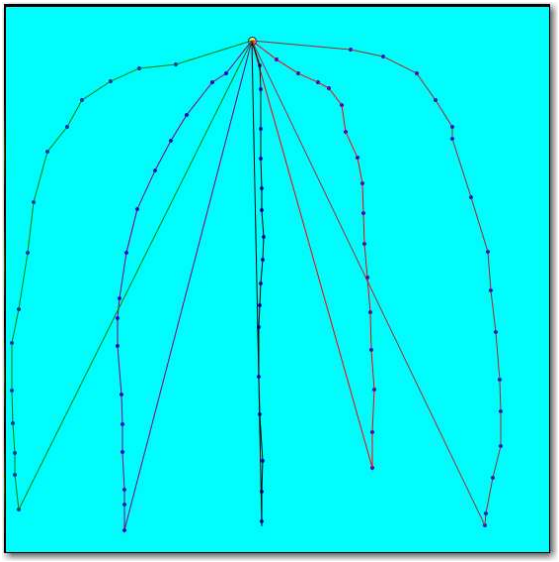


Slika 5.17 – VRP sa 3 vozila i 3 koncentrična kruga

Napomena: za VRP ne vrijedi uvijek pravilo sa sječenjem ruta. Dvije rute različitih vozila nad istim problemom smiju se presijecati bez sigurnosti da to nije optimalno rješenje. Za sigurnu potvrdu bio bi potreban deterministički algoritam.

Korištenjem opcije Mape da se nadodaju gradovi u prazan problem moguće je napraviti problem za kojeg smatramo da znamo optimalno rješenje. SL ACO optimizirao je jedan takav problem, te nađeno rješenje vidljivo je na *slici 5.18*. Prikazano na ovakvoj skici čovjeku je rješenje očito (koje je ponekad krivo!), dok ga algoritam mora tek pronaći, što je odlična alternativa provjere da li program uistinu optimizira svoja rješenja. Jedini stvarni način provjere optimalne rute bio bi deterministički algoritam. Ukoliko bi jedan korak mrava trajao jednako dugo kao i provjera jedne kombinacije gradova, odnosno jedne rute u

determinističkom algoritmu, za sigurni izračun problema sa *slike 5.18* bilo bi potrebno 2×10^{111} sekundi, odnosno **6×10^{103} godina** (SL ACO je trebao **15 sekundi**).



Slika 5.18 - VRP problem sa 5 vozila

5.6 Interpretacija rezultata

Program SL ACO uspješno rješava probleme putujućeg prodavača i usmjeravanja vozila. Prikaz rezultata uspješno je ostvaren i prikazuje rezultate kakvi su bili očekivani. Nad složenijim problemima program daje brze rezultate, koji su ili optimalno rješenje, ili rješenje slično optimalnom.

Parametri korišteni u optimizaciji koji su se pokazali najkorisniji i najprikladniji za demonstraciju rada optimizacije (da algoritam završi u prihvatljivo kratkom roku sa prihvatljivo dobrim rješenjima) su:

Broj gradova n: $30 < n < 90$

Broj mrava m: $m = n$.

Broj vozila v: $1 < v < 6$

Važnost traga α: $\alpha = 1$

Važnost vidljivosti β: $1.5 < \beta < 3.5$

Faktor isparavanja ρ: $0.3 < \rho < 0.6$

Broj ciklusa NC: $50 < NC < 100$

Ukoliko neki parametar za pojedini problem bolje odgovara izvan ovih područja, moguće ga je navesti u samoj datoteci problema. Primjerice, želimo li povećati početni broj ciklusa na 400, dovoljno je u ulaznoj datoteci dodati red "!NC 400" bilo gdje ispod prvog reda (koji označuje tip problema).

6. Sažetak

U ovom projektu opisana su dva NP teška optimizacijska problema (Travelling salesman problem, vehicle routing problem), te su predstavljena moderna rješenja na takve probleme. Moderna računala su moćne mašine velikih mogućnosti, no ipak ograničenih. Za rješavanje složenih optimizacijskih problema u svijetu se koriste mnogi različiti algoritmi, jedan od njih je optimizacija kolonijom mrava.

Optimizacija kolonijom mrava (*Ant Colony Optimization* - ACO) je jedan od popularnijih načina rješavanja problema putujućeg putnika jer daje jako dobre i brze rezultate. Imena dobivenog po mravima, ovaj algoritam oponaša roj mrava u potrazi za hranom. Sustav optimizacije mravljom kolonijom koristi se na mnoge različite načine u svijetu.

Načini rada mrava opisani i demonstrirani su u ovom radu, te su rezultati uspoređeni s rezultatima genetičkog algoritma (još jedan od nedeterminističkih optimizacijskih algoritama).

Izrađen je i vlastiti program na principu ACO sustava nazvan *SL ACO*. Prikazana je struktura koda (dijagrami klasa) i način na koji je program izveden. Program je u stanju učitavati, stvarati i zapisivati optimizacijske probleme tipa TSP i VRP, uz to, SL ACO daje dobre rezultate optimiziranja tih problema.

Rezultati i mjerenja učinjena vlastitom i pronađenom programskom podrškom mogu se pronaći na kraju rada, kao i interpretacija izmjerenih vrijednosti.

7. Literatura

Radovi:

(1) Autori: Marco Dorigo, Vittorio Maniezzo, Alberto Colorni
Ime rada: The Ant System: Optimization by a colony of cooperating agents
Objavljeno: IEEE Transactions on Systems, Man, and Cybernetics-Part B, Vol. 26, 1996

(2) Autori: Vittorio Maniezzo, Luca Maria Gambardella, Fabio de Luigi
Ime rada: Ant Colony Optimization

(3) Autori: Marco Dorigo, Mauro Birattari, Thomas Stützle
Ime rada: Artificial Ants as a Computational Intelligence Technique
Objavljeno: IRIDIA – Technical Report Series, September 2006

Internet stranice:

(4) wikipedia; grupa autora;
stranica: http://en.wikipedia.org/wiki/Ant_colony_optimization

(5) Official ACO metaheuristic site; **autori:** Prasanna Balaprakash, Marco A. Montes de Oca;
stranica: http://www.aco-metaheuristic.org/about.html

(6) Genetic and Ant Colony Optimization Algorithms; **autor:** Peter Kohut;
stranica: http://www.codeproject.com/KB/recipes/GeneticandAntAlgorithms.aspx