SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1012

Optimizacija kolonijom mrava

Tomislav Bronić

Zagreb, siječanj 2010.

Sažetak

U ovom radu objašnjen je princip rada evolucijskog algoritma *optimizacije kolonijom mrava* nad dva NP teška optimizacijska problema. U okviru završnog rada osmišljen i ostvaren je program ACOTB, koji radi na principu optimizacije kolonijom mrava. Program je detaljno ispitan i opisan u ovom radu.

Abstract

This article describes the principles and ideas of the evolutionary algorithm ant colony optimization on two NP complex problems. Within the project, the program ACOTB was designed and made for demonstration purposes of the ant colony optimization algorithm. The program is well tested and described in this article.

Sadržaj

1 Uvod 1

2 Optimizacijski problem 2

2.1 Optimizacijski problemi 2

2.2 Problem trgovačkog putnika 2

2.3 Problem usmjeravanja vozila 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

5 Projektiranje ACO sustava 9

5.1 Ideja 9

5.2 Definiranje problema 9

5.3 Programsko ostvarenje 10

5.4 Struktura 10

5.5 Ulaz i izlaz 12

5.6 WPF korisničko sučelje 13

5.6.1 Lijevi izbornik 13

5.6.2 Desni izbornik 15

5.6.3 Graf najboljih ruta 16

5.6.4 Mapa 16

5.7 Serijsko pokretanje optimiranja 18

5.8 Pomoć 18

6 Mjerenja 20

6.1 Nasumičan problem Random400 20

6.2 Koncentrični krugovi sa 396 gradova 23

6.3 Interpretacija mjerenja i parametara 25

7 Zaključak 27

1 Uvod

U ovom radu obrađuje se optimizacija kolonijom mrava (engl. *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 algoritme zasnovane na inteligenciji roja (engl. *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 okviru rada osmišljen, napravljen, demonstriran i ispitan je program koji rješava optimizacijske probleme trgovačkog putnika i usmjeravanja vozila. Program se služi osnovnim ACO algoritmom i podržava nekoliko dodataka ACO algoritmu.

Program je sustavno ispitan na nekoliko problema te je načinjen grafički prikaz dobivenih rezultata.

2 Optimizacijski problem

2.1 Optimizacijski problemi

Postoji mnogo vrsta optimizacijskih problema, neki su jednostavni, a neki jako složeni za riješavanje. Za sve jednostavne zadatke postoje deterministički algoritmi koji brzo mogu pronaći optimalno rješenje. No za veće i složenije probleme deterministički pristup susreće se sa svojim granicama. Pod pojmom "složeni optimizacijski problemi" (*NP* složeni) mislimo na zadatke koji se ne mogu riješiti determinističkim metodama u prihvatljivom 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.

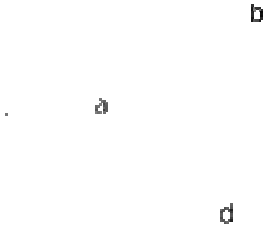
2.2 Problem trgovačkog putnika

Problem trgovačkog putnika (engl. *Travelling Salesman Problem*, TSP) je naziv za specifičnu vrstu optimizacijskog problema. Problem je potekao iz potrebe za minimiziranjem cijene putovanja. Jedan trgovac 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?

Matematički se ovaj problem zadaje kao potražnja najkraćeg Hammiltonovog ciklusa nad potpunim težinskim grafom. Prikazano izrazom (*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.

$$MIN(d_{1,2} + d_{2,3} + d_{3,4} + ...) = MIN\left[(\sum_{i=1}^{n-1} d_{i,i+1}) + d_{n,1} \right] \qquad (2.1)$$

U programu korištenom u ovom radu gradovi su zadani kao točke u ravnini, međusobno udaljene neki konstantan iznos (primjer *Slika 2.1*). Naravno, evolucijski 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 – Problem trgovačkog putnika 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 deterministčkih algoritama za velike brojeve gradova. Primjerice za 50 gradova bi broj mogućnosti koje deterministički stroj mora proći bio 3×10^{64}.

2.3 Problem usmjeravanja vozila

Problem usmjeravanja vozila (engl. *Vehicle Routing Problem*, VRP) sličnog je porijekla kao problem putujućeg trgovca. 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 *izrazom 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.

$$MIN\left[\sum_{v=1}^{V}\left[\left(\sum_{i=(v-1)\cdot(n/V)}^{v\cdot(n/V)-1}\left(d_{i,i+1}\right)\right)+d_{v\cdot(n/V)-1,1}\right]\right] \qquad (2.2)$$

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 riješavanje problema, primjerice određeni period u danu kada dostava mora biti obavljena na pojedinim lokacijama.

Tako primjerice postoji *Pickup and Delivery* verzija VRP-a. *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.

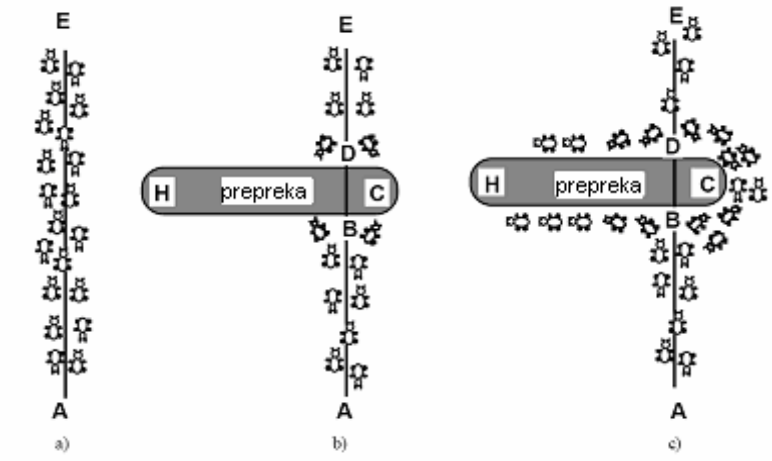
Za rješavanje problema usmjeravanja vozila uz mnoge dodatne uvjete potrebna je malo složenija programska potpora od one opisane u ovom radu.

3 Sustav Mrava

3.1 Mravi u prirodi

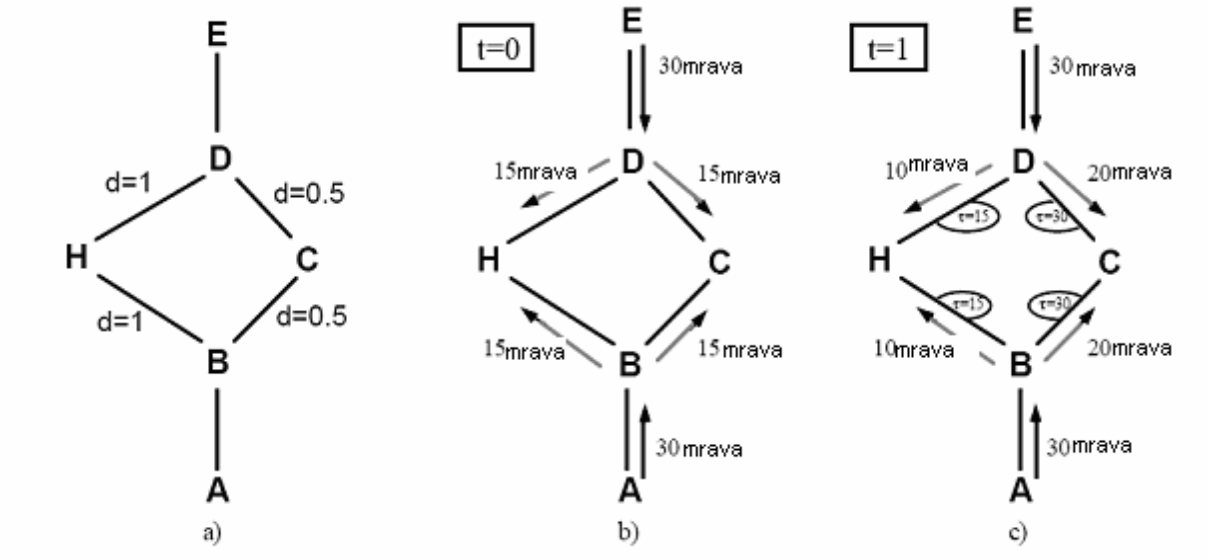
Način na koji mravi u prirodi pronalaze hranu je neko vrijeme bila nepoznanica, dok nije otkriveno da mravi kad pronađu hranu iza sebe ostavljaju trag feromona. Drugi mravi koji osjete feromone prate ih do hrane i vraćaju se do mravinjaka. Ukoliko negdje osjete jači trag feromona radije idu za njim nego za onim tragom koji je slabiji. To ima više razloga, jedan je isparavanje feromona. Naime što više vremena je prošlo od postavljanja feromona to je trag slabiji. To 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. Još jedan faktor koji utječe na snagu traga feromona je kvaliteta nalazišta hrane. Što više i bolje hrane mrav pronađe to jači trag ostavlja iza sebe.

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 biti 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 većina mrava odlučuje 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 problem trgovačkog putnika.



Slika 3.2 - Zaobilaženje prepreke kraćim putem

3.2 Umjetni mravi

Neka je zadan neki problem trgovačkog putnika. Broj gradova koje treba obići je *n*, *m* je broj mrava ukupno u sustavu.

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 kroz problem koje će svi mravi napraviti ograničen je zadanom vrijednosti *NC* (*Number of Cycles*), odnosno maksimalnim brojem ciklusa.

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. Zatim se svi iznosi feromona *τ* množe sa faktorom isparavanja *ρ* (*faktor isparavanja* je zbunjujući naziv, on naime govori upravo suprotno, koliki dio traga ne isparava), te mu se nadoda trag feromona svakog mrava koji je tuda prolazio (*izrazi 3.1* i *3.2*).

$$\tau_{ij}(t+n) = \rho \cdot \tau_{ij}(t) + \Delta\tau_{ij} \qquad (3.1)$$

$$\Delta\tau_{ij} = \sum_{k=1}^{m} \Delta\tau_{ij}^{k} \qquad (3.2)$$

Količina feromona koju mrav doda na neki put je 0 ako nije prošao tim putem. Ukoliko je mrav prošao tim putem, ostavlja neki iznos traga izračunat obrnuto proporcionalno sa duljinom puta koju je prošao (*3.3*).

$$\Delta\tau_{ij}^{k} = \frac{Q}{L_k} \qquad (3.3)$$

U izrazu *3.3 Q* je neka zadana konstanta (proporcionalna broju gradova *n* u danom problemu), a L_k ukupni put kojeg je *k*-ti mrav prešao za svoj obilazak svih gradova.

Pri odluci kamo ići, 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 *izrazom 3.4*.

$$p_{ij}^{k}(t) = \frac{\left[\tau_{ij}(t)\right]^{\alpha} \cdot \left[\eta_{ij}\right]^{\beta}}{\sum \left[\tau_{ik}(t)\right]^{\alpha} \cdot \left[\eta_{ik}\right]^{\beta}} \qquad (3.4)$$

Ovaj izraz (*3.4*) vrijedi u slučaju da mrav ima još lokacija za obići, odnosno njegova *tabu* lista ne obuhvaća sve postojeće gradove. η_{ij} u izrazu 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 resetiraju, isprazne im se tabu liste, te se postave u početni grad. Takav ciklus od *n* iteracija se ponavlja toliko puta dokle god nije ispunjen uvjet zaustavljanja optimizacije.

Pri pokretanju potrebno je postaviti (inicijalizirati) program:

```
    postaviti Trag[i,j] na početnu vrijednost c za svaki i i
j od 1 do n;
```

Zatim se obavlja optimizacija:

```
    Dok (nije(ispunjen_uvjet_zaustavljanja)){
        Odradi_Ciklus();
        Provjeri_Najbolju_Rutu();
        Ispari_Trag();
        Nadodaj_Novi_Trag();
        Resetiraj_Mrave();
    }
```

Funkcija `Odradi_Ciklus()` izgleda otprilike ovako:

```
Za svakog mrava k od 1 do m{
     k.Napravi_Ciklus(); }
```

Funkcija `Napravi_Ciklus()` pojedinog mrava radi sljedeće:

```
Za k od 1 do n{
     Napravi_Korak();    }
```

Funkcija `Napravi_Korak()` pojedinog mrava radi sljedeće:

```
Za sve gradove i od 1 do n {
     Izračunaj p(trenutni_položaj_mrava, i); }
Slijedeći_Grad = Roulette_Wheel_Selekcija(p);
Tabu_Lista.Dodaj(Slijedeći_Grad);
```

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 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 broj ponavljanja petlje koja zove funkciju `Odradi_Ciklus`, *m* je broj mrava čiji se ciklus poziva (broj poziva funkcije `Napravi_Ciklus`), *n* je broj pozivanja funkcije `Napravi_Korak` kojeg mrav obavi u jednom ciklusu, te se sve još jednom množi sa *n,* zbog računa vjerojatnosti prelaska u pojedini grad za svaki korak mrava. Time se dobije ukupna složenost algoritma $NC \times m \times n \times n$.

Ovaj okvir izrade ACO algoritma nije obavezan. Program smije odstupati od ovdje prikazanog algoritma, na primjer modeli algoritma *Ant density* (gustoća mrava) i *Ant quantity* (brojnost mrava). U ta dva modela se trag feromona ne obnavlja 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) na putu $\tau(i,j)$ pri svakom koraku, dok *Ant quantity* postavlja Q/d_{ij}, čime se postiže veća ovisnost o duljini puta.

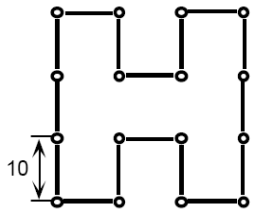
4 Sustavi mrava u pokusima

4.1 Broj mrava

Pokusi i mjerenja opisana u ovom odjeljku napravljeni su od skupine autora *[referenca 1]* kao usporedba podvrsta ACO sustava. Učinjena su mjerenja na mravljem sustavu, *ant density* i *ant quantity*, opisanim u poglavlju *3.3*.

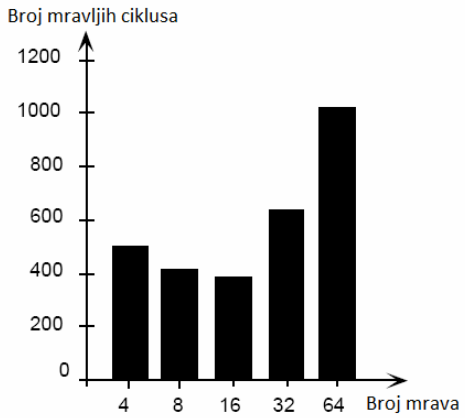
Broj mrava *m*, koje treba stvoriti u nekom programu varira o problemu kojeg treba riješiti. Veći broj mrava će zasigurno dati brže rješenje u prirodi, no ukoliko promatramo računalo kao stroj konačnih mogućnosti, tada veći broj mrava više opterećuje procesor i vrijeme potrebno da svi mravi obave jedan ciklus raste. U svrhu bržeg riješavanja problema, promatrati ćemo vrijeme potrebno da sustav pronađe optimalno rješenje (zadan je dovoljno jednostavan problem da se lako ručno može provrjeriti optimalnost rješenja). Kao jedinica za vrijeme poslužit će nam mravlji ciklus, kako račun nebi ovisio o snazi računala na kojem se algoritam pokreće.

Kao pokus zadan je TSP u obliku 4×4 simetrične geometrijske mreže (*slika 4.1*), čija su optimalna rješenja očita i dobro poznata.



Slika 4.1 – TSP u obliku 4×4 mreže od 16 gradova

Rezultat pokusa je prikazan na *slici 4.2.* 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 *TSP*-a jednak broju gradova u problemu.

5 Projektiranje ACO sustava

5.1 Ideja

Zadatak za rad je ostvariti program koji može riješavati osnovne verzije problema trgovačkog putnika i usmjeravanja vozila opisane u drugom poglavlju ovog rada. Program mora imati mogućnost mijenjanja parametara prema kojima optimira rješenje zadanog problema. Problem se programu zadaje na četiri moguća načina: čitanjem iz datoteke, preko slučajnog generatora, unaprijed zadan algoritamski problem, ili preko korisničkog sučelja stvoren problem. Rezultati optimizacije trebaju biti dostupni u prozoru programa, te mora biti omogućeno zapisivanje rezultata u datoteku.

Algoritam programa će raditi po principu prikazanome u odjeljku 3.3. Formula unutar mrava za dodavanje traga je $\tau = \tau + \frac{ph \times n}{m \times D}$, pri čemu je τ iznos traga na pojedinom putu, *n* broj gradova u aktivnom problemu, *ph* konstana feromona, *m* broj mrava, a *D* duljina ukupnog puta kojeg je mrav prešao.

5.2 Definiranje problema

Ulaz za program biti će omogućen na više načina. Za učitavanje određenog gotovog problema u program potrebno je gumbom *Load* izabrati željenu datoteku sa zapisanim problemom. 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, radi usporedbi uspješnosti optimizacije s različitim parametrima.

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)), 7 vozila u VRP-u, primjer navođenja komentara (#) i ulaznih parametara (!).

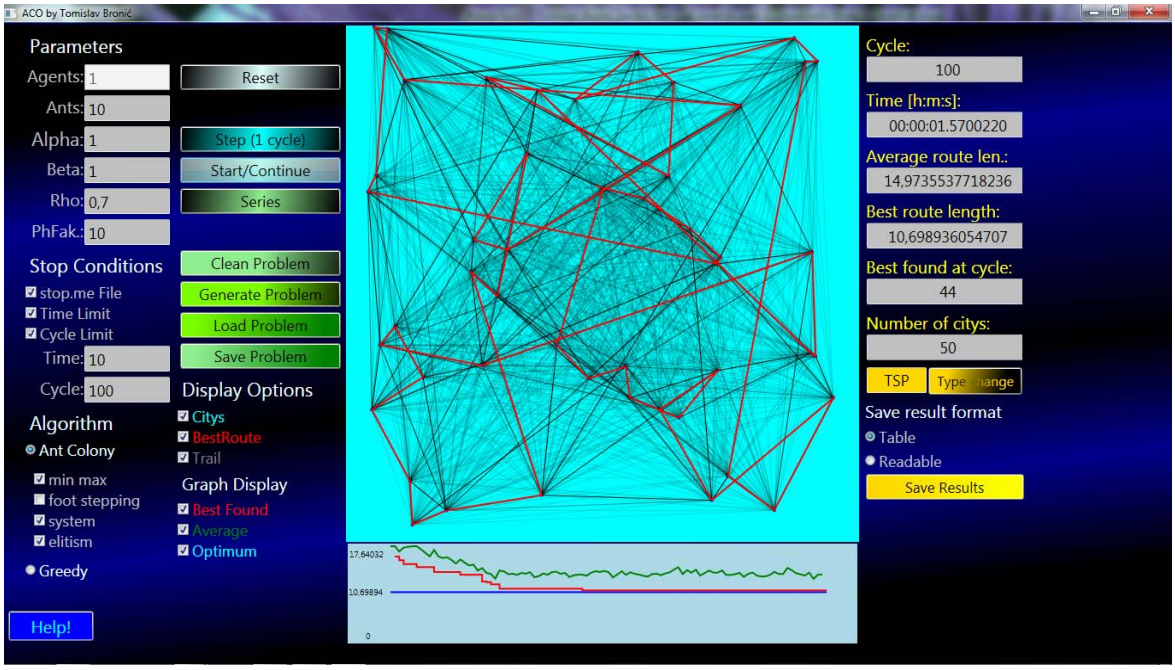
Tablica 5.1: Primjer ulaznih podataka

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, 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 Programsko ostvarenje

Program je nazvan *"ACOTB"*, prema ACO algoritmu, te autoru Tomislavu Broniću. Program je ostvaren u okruženju *Visual Studio 2008* programskim jezikom C#, uz korištenje WPF forme za prikaz korisničkog sučelja. Za pokretanje programa potrebno je imati instaliran Microsoft .NET Framework 3.5. Prilikom prevođenja i pokretanja programa otvara se korisničko sučelje u obliku prozora prikazanog na *slici 5.1*.



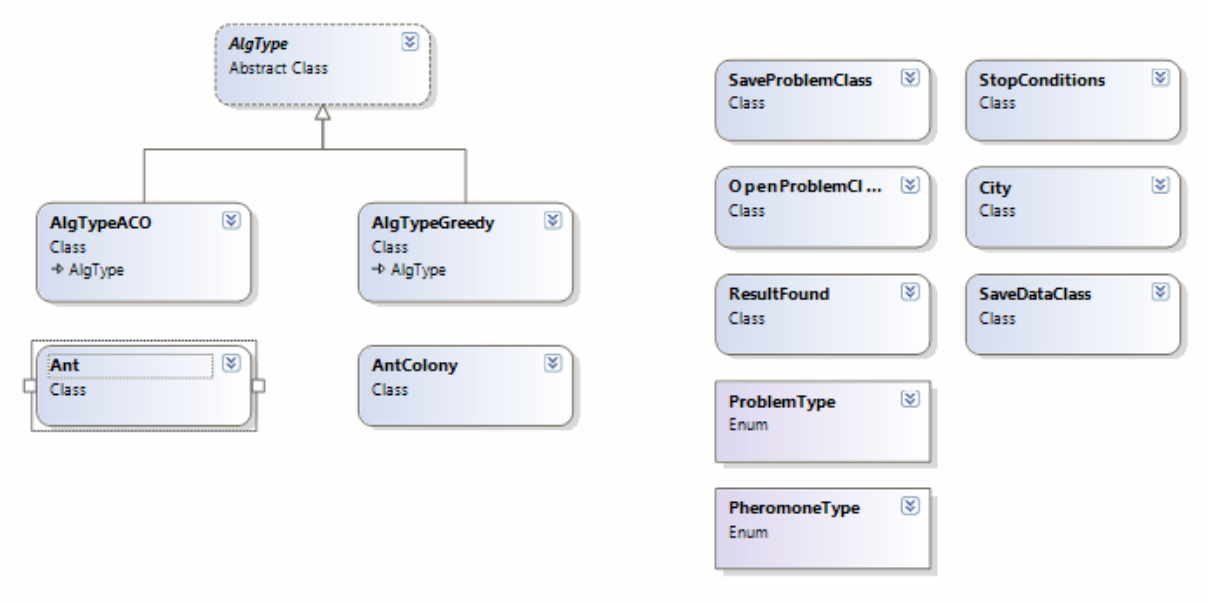
Slika 5.1. – Korisničko sučelje programa ACOTB

5.4 Struktura

Program je rastavljen na tri cjeline: `ACO Biblioteka`, `Console Runner`, te `WPF Window`. `WPF Window` je korisničko sučelje prikazano na slici 5.1. Primarni način pokretanja programa je preko WPF prozora, zbog veće kontrole nad programom i svim parametrima. Program se može pokrenuti pozivom iz konzole, pri čemu se može odrediti ulazni problem, ime izlazne datoteke, te vrijeme optimizacije. Primjerice poziv može izgledati ovako: ***"ConsoleRunner.exe problem.txt rezultat.txt 20"***, pri čemu je *problem.txt* potpuni put do datoteke u kojoj je spremljen željeni problem, *rezultat.txt* je ime datoteke u koju se spremaju rezultati, te broj 20 označava vrijeme optimizacije u sekundama. Ukoliko se programu ne zada vrijeme optimizacije moguće ga je zaustaviti stvaranjem "*stop.me"* datoteke u kazalu gdje se nalazi pozvani program.

U pozadini programa je `ACO Biblioteka`, koja je u potpunosti sposobna samostalno izvršavati sve funkcije programa. `Console Runner` i `WPF Window` služe samo za pozivanje funkcija i klasa iz `ACO biblioteke`. Osnovna klasa `ACO biblioteke` je `AntColony`, jer se sva komunikacija korisničkih sučelja sa `ACO bibliotekom` vezana uz optimizaciju odvija preko `AntColony` klase. Ona također šalje rezultate optimizacije korisničkom sučelju, koje ih zatim iscrtava na odgovarajuća mjesta. Ukoliko se prikaže dijagram klasa projekta, prepun je

metoda i atributa koji služe lakšem čitanju koda funkcija unutar klasa. Na *slici 5.2* prikazan je pojednostavljen dijagram klasa `ACO biblioteke`.



Slika 5.2 - Dijagram klasa ACO biblioteke

Klasa `AntColony` je zaštićena *singleton* oblikovnim obrascem preko atributa `Instance`, čime je program osiguran da se neće stvoriti više od jednog objekta kolonije. Kolonija sadrži listu mrava, listu gradova, te sve parametre algoritma i sve funkcijske varijable potrebne u procesu optimiranja.

Gradovi su objekti tipa `City`, no po svim svojim osobinama oni su zapravo točke (sadrže X i Y vrijednosti, te konstruktor koji prima 2 *double* vrijednosti (x i y)).

Mravi su objekti tipa `Ant` koji sadrže atribute važne za njihovo ispravno funkcioniranje. Najvažniji među njima je `Tabu` lista, u kojoj je zapamćeno koje gradove je mrav obišao, i kojim redoslijedom. Pri računanju duljine puta kojeg je mrav prešao, zapravo se samo zbroje udaljenosti među gradovima kako ih je mrav obilazio.

Za obilazak ciklusa i odluku kojim putem će ići, svaki mrav ima metodu `makeCycle` koja poziva privatnu metodu `makeStep` dok se svi gradovi problema ne nađu u tabu listi. Kad se tabu lista mrava popuni, mrav ne može pronaći nijedan grad u koji će se pomaknuti, te ga `makeStep` vrati u početni grad (samo ukoliko nije već tamo). Kada svi mravi završe cijeli ciklus kolonija zove metodu `CalculateRouteLenght` svakog mrava te rezultat uspoređuje sa dotad pronađenima. Mrav unutar te metode izračuna, vrati i privremeno upamti duljinu puta kojeg je prešao. Nakon svih proračuna i potrebnih kopiranja podataka, kolonija poziva dodatke algoritmu specifične za izabranu verziju algoritma. Budući da formula za računanje iznosa feromona ovisi i o verziji algoritma, se dodavanje feromona poziva direktno iz `AlgType` klase (klasa kojom se definiraju dodaci algoritmu) preko metode `DoVersionExtras` (*što koji dodatak točno radi piše u dodatku A na kraju ovog rada*). Ovisno o algoritmu, `AlgType` će pozvati metode `AddTrail` svih mrava, sa različitim ulaznim parametrima. Unutar metode `AddTrail` mravi nadodaju svoj feromon na odgovarajuće bridove. Kolonija na

kraju ciklusa očisti sve mrave (vrati ih u njihove početne gradove i resetira *Tabu* liste) pozivom metode `ResetAnt` svih mrava.

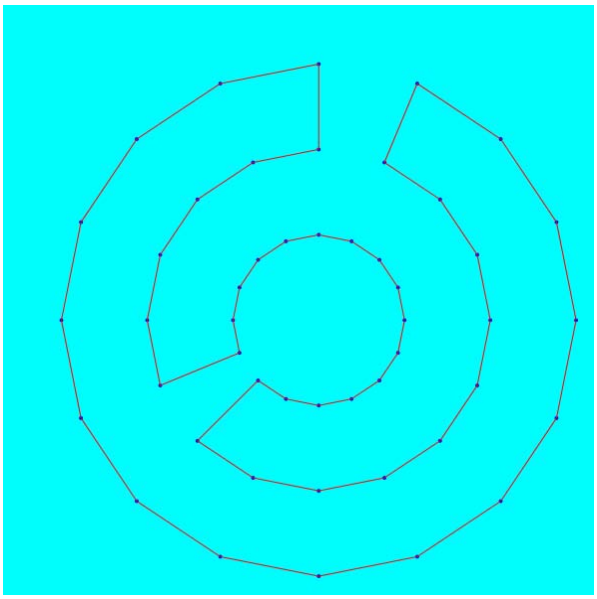
Radi jednostavnosti nadogradnje i budućih dodataka programu, `AlgType` je abstraktna klasa, čime se omogućava dodavanje novih dodataka i/ili verzija algoritma u program bez mijenjanja postojećeg koda.

Od klasa ključnih za funkcioniranje optimizacije potrebno je navesti još dvije (uz `AlgType`, `Ant`, `AntColony` i `City`), `ResultFound` i `StopConditions`. `ResultFound` je klasa u koju se zapakira novo pronađeno rješenje, odnosno vrijeme kada je ruta pronađena, njena duljina i redni broj ciklusa u kojem je pronađena. `StopConditions` je klasa kojom se definiraju uvjeti zaustavljanja optimizacije. Kolonija na početku svakog ciklusa šalje upit objektu tipa `StopConditions`, te ukoliko je rezultat pozitivan (ispunjen neki od uvjeta zaustavljanja) prekida sa optimizacijom.

Napomena: kod nekih funkcija ključnih za rad algoritma dodan je u na kraju rada u Dodatku A.

5.5 Ulaz i izlaz

Ostale klase `ACO biblioteke` su klase za ulaz i izlaz, kako bi se nebitni dio koda (nebitni za funkcioniranje optimizacije) odvojio od kolonije. Za učitavanje i generiranje problema postoji klasa `OpenProblemClass`. `OpenProblemClass` posjeduje 5 metoda, svaka od njih ima istu svrhu, naime da upiše optimizacijski problem u koloniju. Metodom `ReadProblemFromFile` moguće je učitati problem iz dane datoteke (ukoliko je problem ispravno zapisan, u formatu opisanom u poglavlju 5.2.). Ostale 4 metode algoritamski generiraju neki oblik pogodan za demonstraciju programa (pogodan jer je rješenje nad grafičkim prikazom tih problema očito ljudima, stoga je lako provjeriti uspješnost algoritma). Primjerice problem sa koncentričnim krugovima, gdje je očito najbolje rješenje direktno opisivanje svakog od tri kruga, te jedna "rupa" gdje agent skače iz jednog kruga u drugi (slika 5.3.).



Slika 5.3 - Jedno od optimalnih rješenja problema 3 koncentrična kruga

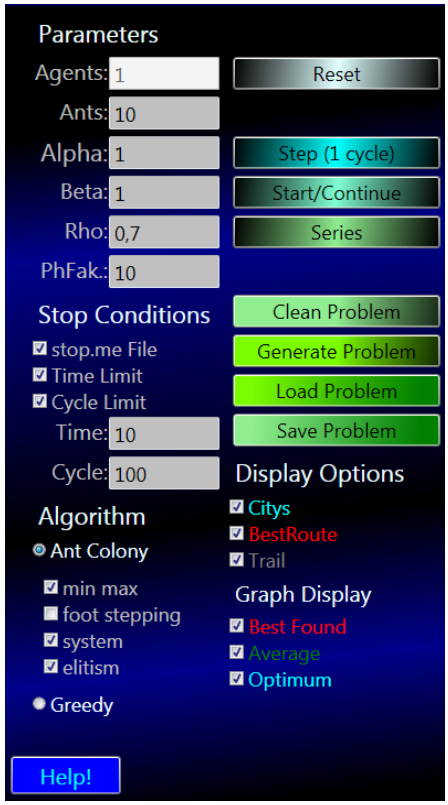
Osim klase za učitavanje problema, `ACO biblioteka` sadrži i klasu za zapisivanje problema, `SaveProblemClass`. Svrha ove klase je spremanje problema stvorenih u *ACOTB* programu u adekvatnom formatu za kasnije korištenje. Primjerice problemi stvoreni preko korisničkog sučelja koje korisnik želi spremiti, ili nasumični problem nad kojim se želi pokrenuti više optimizacija.

Treća od ulazno izlaznih klasa je `SaveDataClass`. Radi jednostavnosti komunikacije korisničkih dijelova programa sa `ACO bibliotekom`, ova klasa je *public* vidljivosti, kao i kolonija. `SaveDataClass` služi za pamćenje rezultata optimizacije, spremanje rezultata u adekvatan format (kao tablica pogodna za učitavanje u Microsoft Excel, ili format lakše čitljiv bez dodatne obrade), te spremanje tog zapisa u danu datoteku.

5.6 WPF korisničko sučelje

Sučelje za korištenje programa može se podijeliti u nekoliko dijelova. Lijevi izbornik upravlja većinom funkcija i parametara programa. Desni dio sučelja 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.

5.6.1 Lijevi izbornik



Slika 5.4 - Lijevi korisnički izbornik

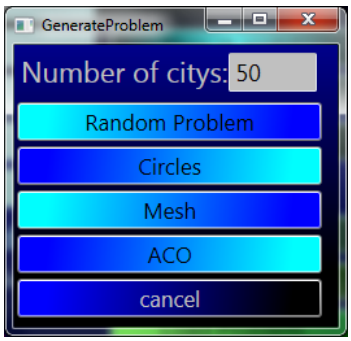
Lijevi izbornik (*slika 5.4*) sadrži kontrole za učitavanje, stvaranje i spremanje problema, te reguliranje svih parametara optimizacije. U trećem poglavlju ovog rada opisane su uloge pojedinih parametara. Na raspolaganju su parametri: broj vozila za VRP (0<*brojVozila*<300), broj mrava *m* (0<*m*<1000), važnost Traga

Alpha (0<=*α*<100), važnost vidljivosti *Beta* (0<=β<100), faktor isparavanja *Rho* (0<=*ρ*<=1) i konstanta feromona *PhFak* (1<=*PhFak*<100000). Osim parametara moguće je definirati i uvjete zaustavljanja optimizacije. Postoje tri moguća uvjeta zaustavljanja: vremensko ograničenje, maksimalni broj ciklusa i *"stop.me"* datoteka. Optimizacija se zaustavlja ukoliko je ispunjen bilo koji od uvjeta zaustavljanja. Osim ACO algoritma i 4 različita dodatka algoritmu, moguće je pokrenuti optimizaciju pohlepnim algoritmom (engl. *Greedy*). Pohlepni algoritam ne koristi *Trail* komponentu, niti vjerojatnost (α= 0, β= ∞), već uvijek ide u najbliži neposjećeni grad. Pohlepni algoritam jako brzo nađe svoje najbolje rješenje, koje je gotovo uvijek lošije od ACO rješenja, no razlika u brzini algoritama je velika.

Gumb `Reset` briše dosadašnje rezultate optimizacije i vraća brojač ciklusa na 0. Gumb `Step (1 cycle)` pokreće optimizaciju iz trenutnog stanja kolonije sa uvjetom zaustavljanja 1 ciklus. Gumb `Start/Continue` pokreće optimizaciju iz trenutnog stanja kolonije sa uvjetima zaustavljanja zadanim u korisničkom sučelju, pri čemu se broj ciklusa ponaša kao uvjet na novo dodane cikluse.

Primjer: kolonija je odradila 50 ciklusa optimizacije i našla nekakvo rješenje. Novo pokretanje optimizacije pritiskom gumba `Start/Continue` sa uvjetima zaustavljanja kakvi su na slici 5.4 će rezultirati optimiranjem koje će trajati narednih 10 sekundi, ili dok kolonija ne dođe do 150-og ciklusa, ili dok korisnik ne kreira stop.me datoteku u kazalu programa. Pritom će se kao početno stanje kolonije koristiti stanje u kojem se kolonija nalazila na kraju prošlog pokretanja (iznosi feromona i najbolja ruta).

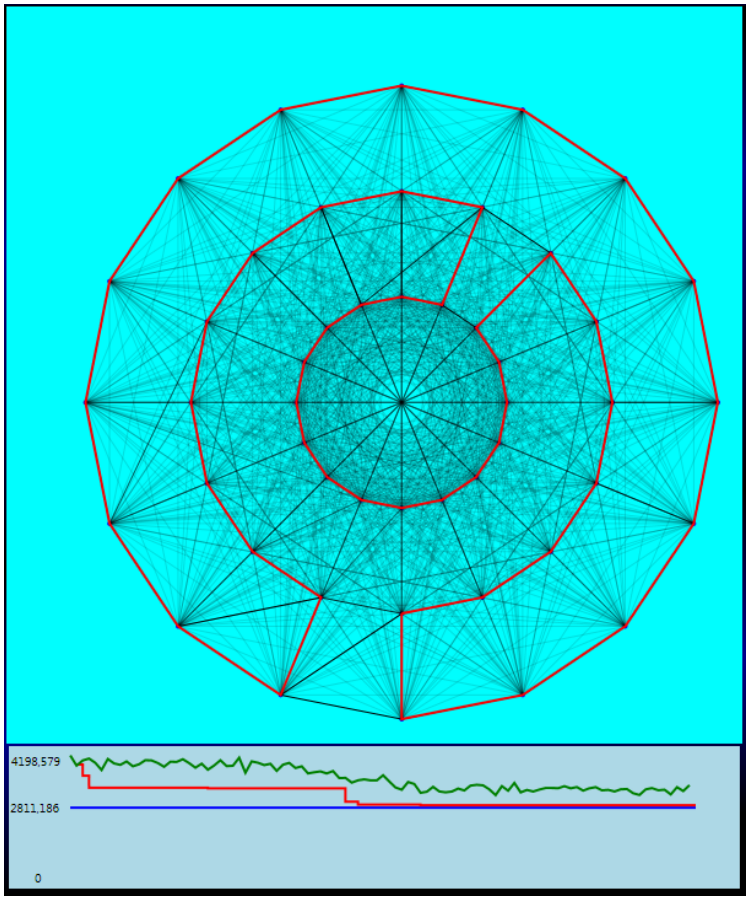
Gumb `Series` otvara prozor za pokretanje serije optimizacija, koji će biti dodatno objašnjen u kasnijem poglavlju. Za otvaranje, spremanje i kreiranje problema postoje 4 dodatna gumba, čije je korištenje intuitivno i jednostavno. Gumbom `Clean Problem` se briše učitani problem, omogućavajući crtanje vlastitog problema (klikanjem mišem po mapi gradova se dodaju gradovi u problem). Gumbom `Load Problem` se otvara *open file dialog* kojim se bira .txt datoteka za učitavanje spremljenog problema. `Save Problem` gumb sukladno tome otvara *save file dialog* za spremanje trenutno učitanog problema u datoteku. Za kreiranje slučajnog ili algoritamskog problema postoji gumb `Generate Problem` kojim se otvara prozor prikazan na slici 5.5. Pri generiranju problema moguće je izabrati i željeni broj gradova. Pri stvaranju algoritamskog problema broj gradova može odstupati od željenog, jer je zadržavanje generiranog oblika većeg prioriteta nego zadovoljavanje točnog broja gradova.



Slika 5.5 – Prozor za definiranje i generiranje problema trgovačkog putnika

Ispod liste gumba na sučelju se nalazi i nekoliko opcija za određivanje grafičkih prikaza rezultata. Na mapi gradova je moguće birati prikaz 3 elementa: gradovi

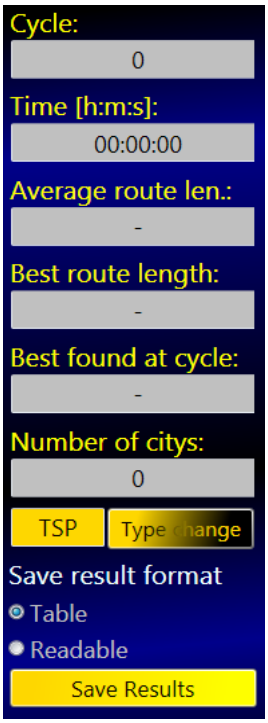
(plave točke), najbolja ruta (crvena linija) i trag feromona (crne crte različitih debljina; pri velikom broju gradova crtanje traga feromona može trajati više sekundi). A za graf najboljih ruta moguće je birati najbolju rutu u vremenu (crvena linija), prosječnu duljinu ruta u pojedinom ciklusu (zelena linija), te minimum (plava linija na najnižoj točki koju je crvena linija dostigla). Primjer prikaza sa svim opcijama uključenim prikazan je na slici 5.6.



Slika 5.6 - Grafički prikaz rezultata: gore mapa gradova, dolje graf ruta

5.6.2 Desni izbornik

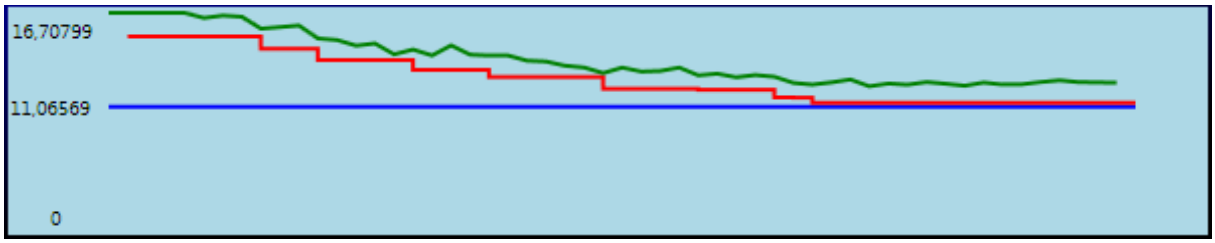
U desnom dijelu sučelja *(slika 5.7)* nalaze se polja koja prikazuju rezultate mjerenja ili parametre problema koje korisnik ne može mijenjati unosom novih vrijednosti u prozor. Prikazuju se broj ciklusa koji je korišten do sad, vrijeme prošlo od početka optimiranja do kraja, prosječna duljina svih ruta nađenih tokom optimizacije, 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 tipa problema (TSP ili VRP), te opciju spremanja rezultata u datoteku. Spremanje može biti u formatu čitkom za ljude iz txt datoteke, ili pogodnijem za učitavanje u Excel tablice.



Slika 5.7 - Desni izbornik za prikaz rezultata optimiranja

5.6.3 Graf najboljih ruta

Na dnu korisničkog sučelja *(slika 5.8)* ugrađen je graf koji prikazuje duljinu najbolje pronađene rute u vremenu.



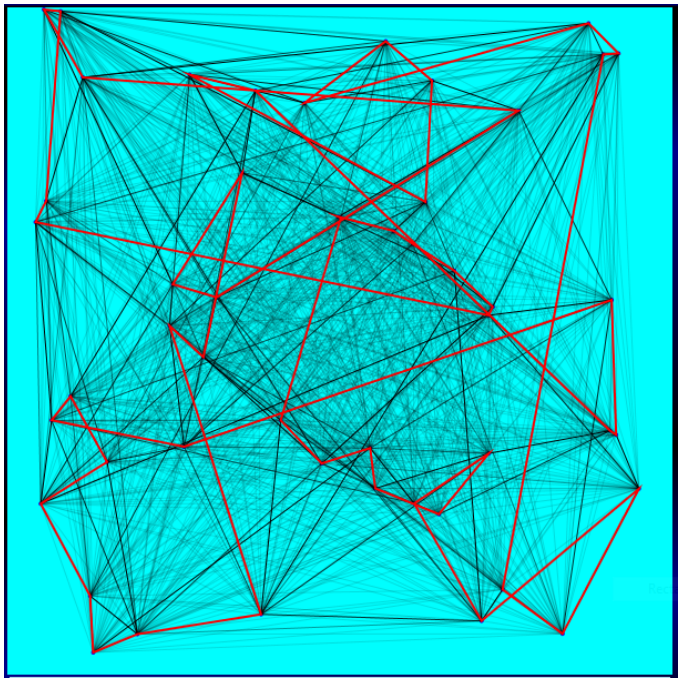
Slika 5.8 – Graf duljina najboljih ruta u vremenu

Vertikalna os grafa prikazuje duljinu rute, a horizontalna vrijeme, odnosno broj ciklusa u kojem je pojedina ruta pronađena. U sustavu su prikazane 3 linije, crvenom bojom je naznačen graf najbolje nađene rute u vremenu, zelenom prosječna duljina svih ruta u tom ciklusu, a plavom bojom je naznačena vodoravna linija kao najbolja ruta pronađena 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.

5.6.4 Mapa

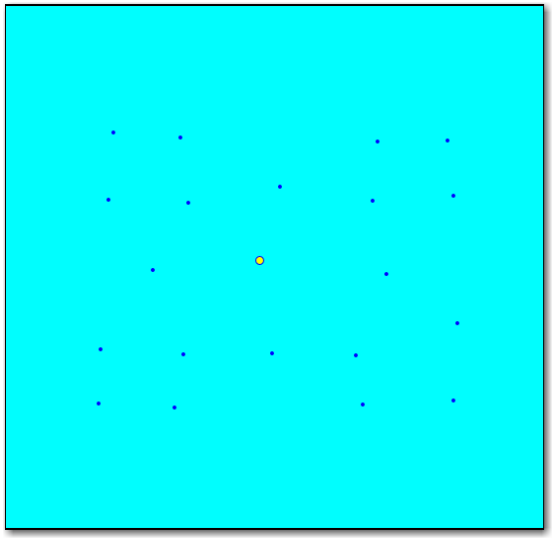
Većina površine korisničkog sučelja je preostali element: mapa gradova. Na mapi se mogu prikazivati tri stvari: gradovi trenutno učitanog problema, iznos feromona na pojedinim bridovima, te najbolja ruta pronađena tokom optimizacije. Pojedini

prikazi se mogu uključivati i isključivati pored mape u lijevom meniju. Slika mape sa svim mogućim prikazima prikazana je na *slici 5.9*.

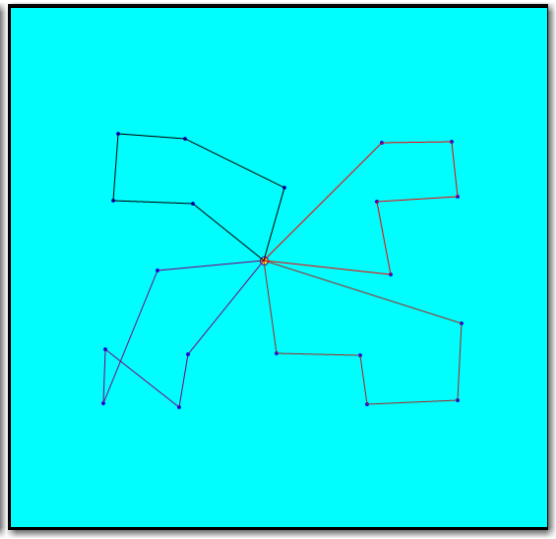


Slika 5.9 – Slučajni problem prikazan na mapi sa svim opcionalnim prikazima

Ukoliko se radi o problemu usmjeravanja vozila, prikaz na mapi se malo razlikuje od onog u slučaju TSP-a. Početni grad (prvi po redu) se naime iscrtava većim kružićem od drugih (slika 5.10), a najbolja ruta je razložena na više dijelova, od kojih je svaki pobojan u neku drugu boju (ima 8 boja, nekon toga se počinju ponavljati) (*slika 5.11*).



Slika 5.10 – VRP gradovi

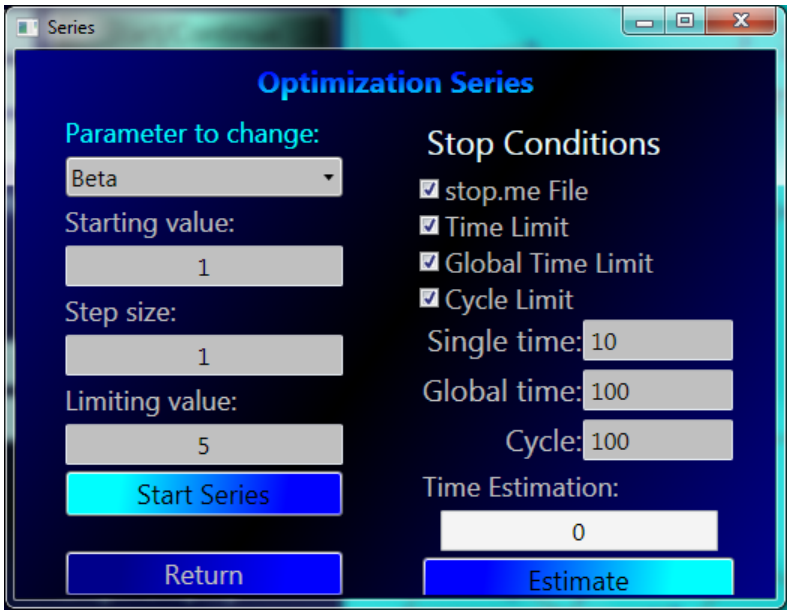


Slika 5.11 – VRP rješenje

Osim prikazivanja problema i rješenja, mapa ima i još jednu funkciju. Naime, unutar programa ACOTB 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).

5.7 Serijsko pokretanje optimiranja

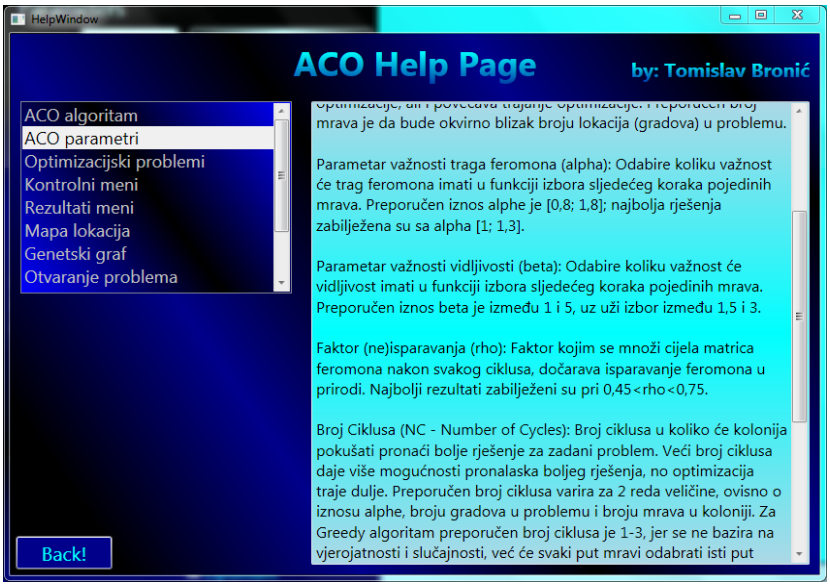
Pritiskom na gumb `Series` otvara se prozor prikazan na *slici 5.12*, kojim se može optimiranje pokrenuti u seriji više puta za redom. Omogućene opcije za pokretanje serije optimiranja su: parametar koji će se mijenjati, početna vrijednost, krajnja vrijednost, veličina koraka, te uvjeti zaustavljanja pojedinih optimizacija. Dodatno je omogućena opcija globalnog vremenskog ograničenja, koja zaustavlja seriju optimizacija prije nego odabrani parametar dođe do granične vrijednosti (ukoliko dano vrijeme istekne prije toga). Uz globalno vremensko ograničenje dana je i funkcija za procjenu trajanja optimizacija. Ukoliko korisnik ne želi postavljati globalni vremenski uvjet za zaustavljanje rada programa, funkcijom `Estimate` može saznati koliko će (otprilike) trajati izvođenje serije. Pri pokretanju serije optimiranja odmah se otvara i dijalog za spremanje datoteke sa rezultatima (jer pokretanje serije nema nikakve svrhe ukoliko se rezultati ne zapišu u neku datoteku). Ostali parametri se uzimaju kakvi su bili podešeni u osnovnom prozoru programa.



Slika 5.12 – Prozor za pokretanje serije optimiranja

5.8 Pomoć

Unutar programa *ACOTB* ugrađen je i jednostavan sustav za pomoć. Na mnogim gumbima i poljima ugrađeni su tooltip dodaci (kratko objašnjenje elementa), te je dodan gumb "Help" u donji lijevi kut prozora. Help otvara prozor prikazan na slici 5.13.



Slika 5.13 – ACO Help Page

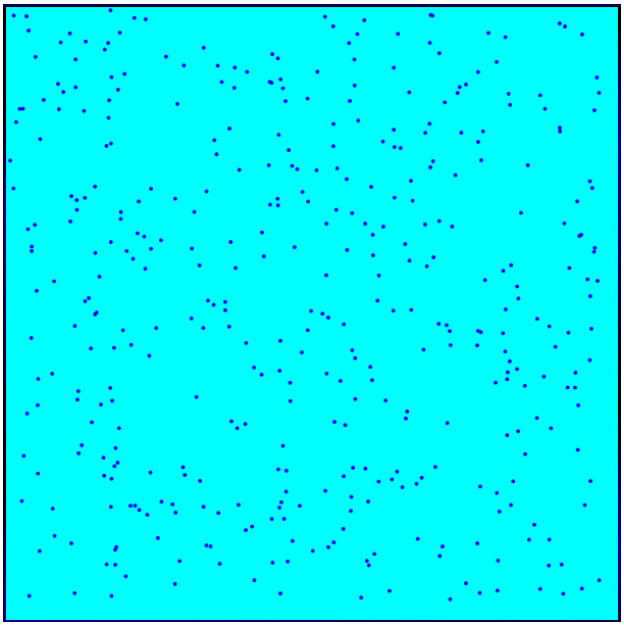
Na prozoru moguće je odabrati neku od tema za koju je potrebna pomoć, te će se tekst vezan uz temu ispisati desno na za to predviđen prostor. Prozor pomoći sadrži svu teorijsku podlogu potrebnu za razumijevanje problema i programa, te upute za korištenje i opis svih drugih dijelova programa.

6 Mjerenja

U svrhu demonstracije rada programa, isprobano je optimiranje nad mnogim različitim TSP i VRP problemima. Problemi sa malim brojem gradova nam nisu zanimljivi, jer bi se mogli riješiti deterministčkim algoritmom, čime bi se osiguralo optimalno rješenje. Složenost problema TSP je (kao prije napomenuto) ½(n-1)! što je za male brojeve još uvijek u razumno brzom vremenu. Osim toga postoji verzija algoritma kojim se može TSP riješiti deterministički u vremenu $n^2 \times 2^n$, što je puno brže od faktorijalne složenosti, no još uvijek pre sporo za velike brojeve gradova.

6.1 Nasumičan problem Random400

Za demonstraciju optimizacije i pretragu mape parametara korišten je slučajni problem sa 400 gradova (prikazan na slici 6.1.).



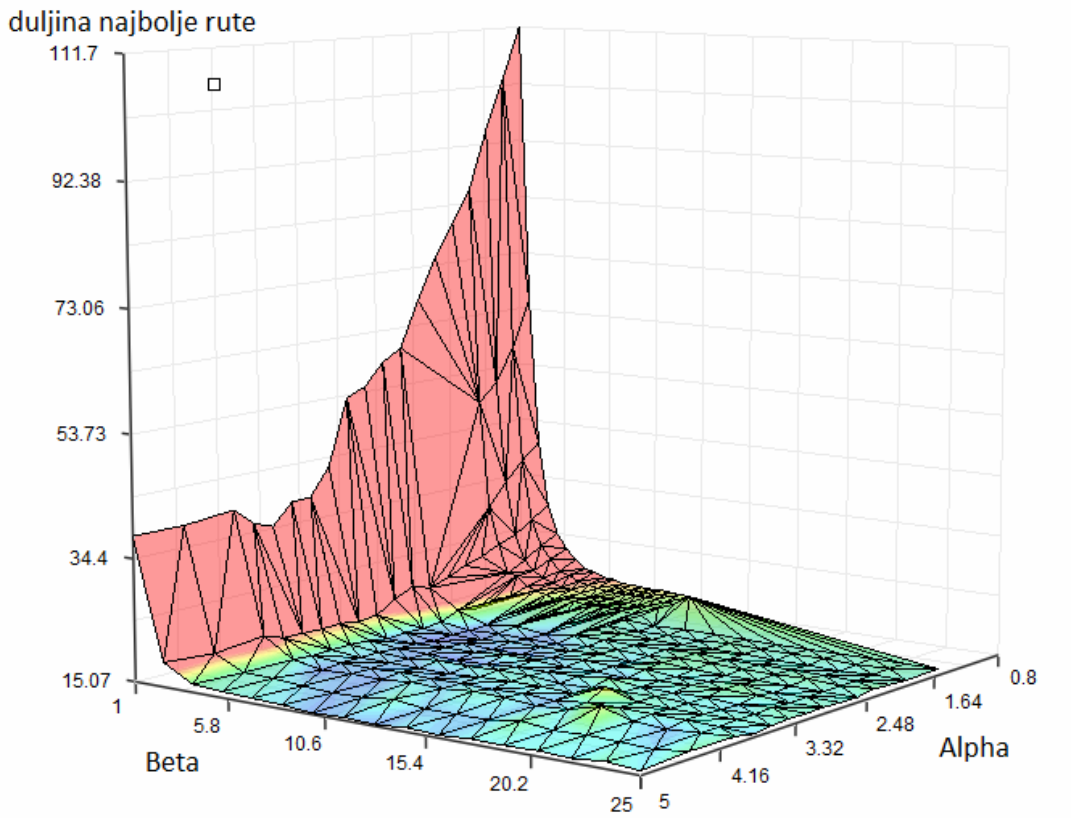
Slika 6.1 – Slučajni problem 400 gradova korišten za mjerenja

Koristeći problem sa 400 gradova, program *ACOTB* treba otprilike 0.212 sekunde za jedan ciklus jednog mrava (na određenom računalu). Ukoliko se broj mrava postavi na 100 (red veličine jednak redu veličine broja gradova), te broj ciklusa na 100 (dovoljno za pronalaženje nekog lokalnog ili općeg minimuma), programu treba otprilike 2120 sekundi (35 minuta) da završi sa radom. Deterministčkom algoritmu sa faktorijelnom složenosti bi trebalo 8×10^{865} iteracija (odnosno 2×10^{853} godina rada istog računala). Algoritam eksponencijalne složenosti bi na istom računalu trajao otprilike 1×10^{113} godina. Deterministički algoritmi imaju ipak jednu prednost nad evolucijskim algoritmima, a ta je da mogu osigurati da je nađeno rješenje optimalno. ACO algoritam nikada neće moći samostalno osigurati uspješnost, odnosno kvalitetu, rezultata. No brzina pronalaska rješenja je kod ovakvih problema ipak ključna točka koju promatramo. Iako rješenje evolucijskog algoritma neće biti savršeno, biti će dovoljno blizu optimalnog da se razlika isplati žrtvovati u svrhu bržeg rješavanja problema.

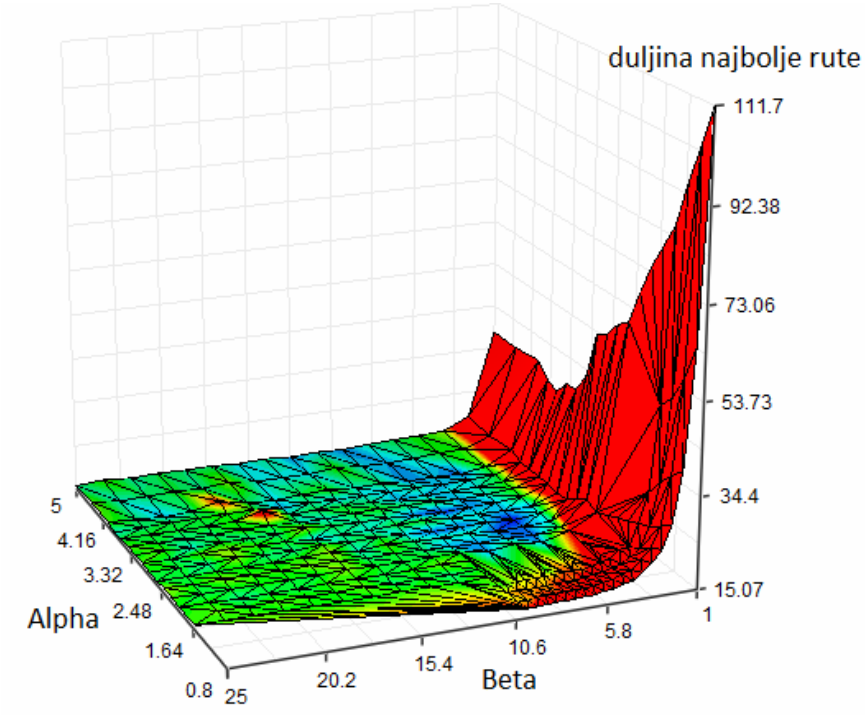
Za generiranje tlocrta parametara (*parameter landscape*) će se proučavati dva najbitnija parametra algoritma: *α* i *β*. Cilj je pronaći set parametara (*α* i *β*) za koje program pronalazi najbolje rezultate. Za to se ostali parametri moraju fiksirati. Problem nad kojim će se vršiti optimizacija je prikazan na slici 6.1, iznos feromonskog faktora *PhFak* je 10, broj ciklusa maksimalno dozvoljen optimizaciji *NC* je 100, te je pojedina optimizacija ograničena vremenski na 40 minuta.

Kroz nekoliko jednostavnih pokusnih optimiranja donesen je zaključak o vrijednosti preostalih parametara, kao što je faktor isparavanja *ρ*=0.7. Tip Algoritma korišten je *ACO* uz dodatke *Elitism, MinMax* i *System*.

Parametar *α* je mijenjan koracima veličine 0.2, počevši od 0.8 (x os). a parametar *β* od 1 do 25 koracima 1.5 (y os). Na *slici 6.2* prikazan je tlocrt parametara dobiven takvim mjerenjima, plavom bojom prikazana su najbolja rješenja, te kako boje postaju toplije, to je rješenje lošije. Sa slike je odmah vidljivo da za vrijednosti β<4 program daje jako slaba rješenja, slično vrijedi i za α<1.5. Pri boljem proučavanju grafa uočavamo da su najbolja rješenja pronađena za α između 2 i 4, te β između 5 i 10. Isti rezultati prikazani iz druge perspektive prikazani su na *slici 6.3*.



Slika 6.2 – Parameter landscape A β=[1, 25], α=[0.8, 5.0]



Slika 6.3 - Parameter landscape B β=[1, 25], α=[0.8, 5.0]

Kako su najbolji rezultati ispali za jedno (relativno) usko polje parametara, optimiranje je ponovljeno još nekoliko puta sa vrijednostima iz tog područja, te je za prikaz uzeta samo prosječna vrijednost najbolje rute na pojedinom skupu parametara. Tako dobiven tlocrt parametara prikazan je na *slici 6.4*.

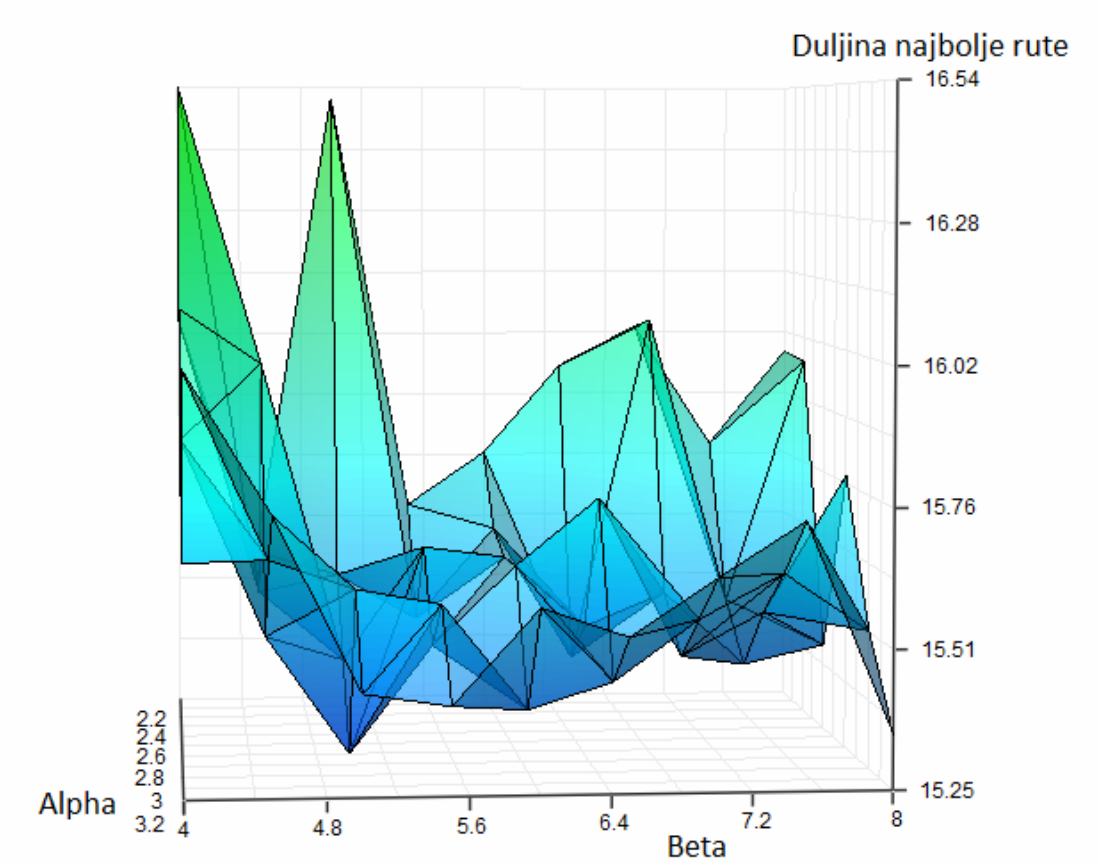
Napomena: U dodatku na kraju rada nalaze se tablice sa mjerenjima i rezultatima iz kojih su izvađene vrijednosti za prikazane grafove. Prvih nekoliko redova mogu se vidjeti u tablici 6.1. Ukoliko se izbace stupci tablice koji se ne mijenjaju dobivamo tablice iz Dodatka B, prikazane na tablici 6.2.

Tablica 6.1: Mjerenja za problem Random400

najbolja ruta	nadjena	nadjena u trenutku	broj gradova	broj vozila	broj mrava	alpha	beta	rho	broj ciklusa	konstanta	Tip algoritma	Tip feromona
16,05167917	98	00:38:32.6471322	400	1	100	2,2	4	0,7	0	10	Elitism, MinMax, System	Simple
15,5593373	102	00:38:45.7043861	400	1	100	2,2	4,5	0,7	0	10	Elitism, MinMax, System	Simple
16,51248388	78	00:30:38.0619761	400	1	100	2,2	5	0,7	0	10	Elitism, MinMax, System	Simple
15,65723177	94	00:37:22.8836447	400	1	100	2,2	5,5	0,7	0	10	Elitism, MinMax, System	Simple
15,76686536	93	00:37:21.9788409	400	1	100	2,2	6	0,7	0	10	Elitism, MinMax, System	Simple

Tablica 6.2: Mjerenja za problem Random 400, samo varijabilne vrijednosti

najbolja ruta	nadjena u ciklusu	nadjena u trenutku	alpha	beta
111,713388	23	00:08:51.9218750	0,8	1
64,98306795	21	00:08:05.0625000	0,8	1,5
40,33393895	21	00:08:17.3437500	0,8	2
29,6122689	21	00:07:52.3437500	0,8	2,5
24,71663414	20	00:07:25.6562500	0,8	3
22,28430972	36	00:13:16.9218750	0,8	3,5
20,61181079	34	00:12:34.9843750	0,8	4
19,22458719	25	00:09:15.0312500	0,8	4,5
18,94465139	16	00:05:55.2500000	0,8	5
18,2005641	17	00:06:18.0312500	0,8	5,5

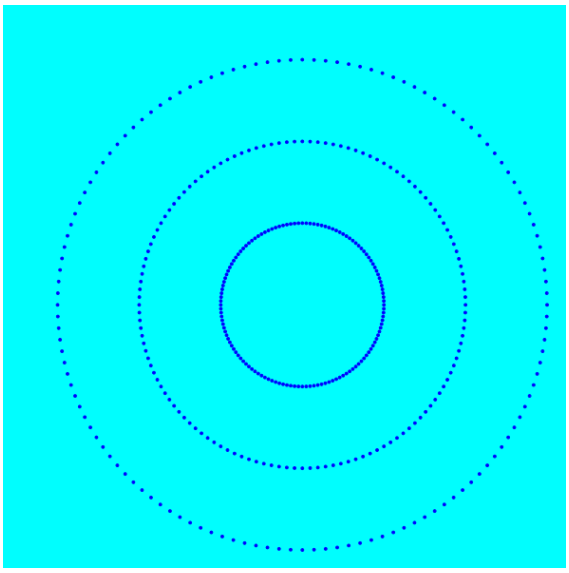


Slika 6.4 – Parameter landscape C β=[4, 8], α=[2.2, 3.2]

Sa slika uočavamo da su najbolja rješenja potekla iz optimizacija gdje je beta bio ~ 5, a alpha ~2.7. Kako bi utvrdili da rješenja ne vrijede samo za ovaj problem, već općenitije, sa dobivenim parametrima pokrenuta su optimiranja drugih problema sa okvirno jednakim brojem gradova.

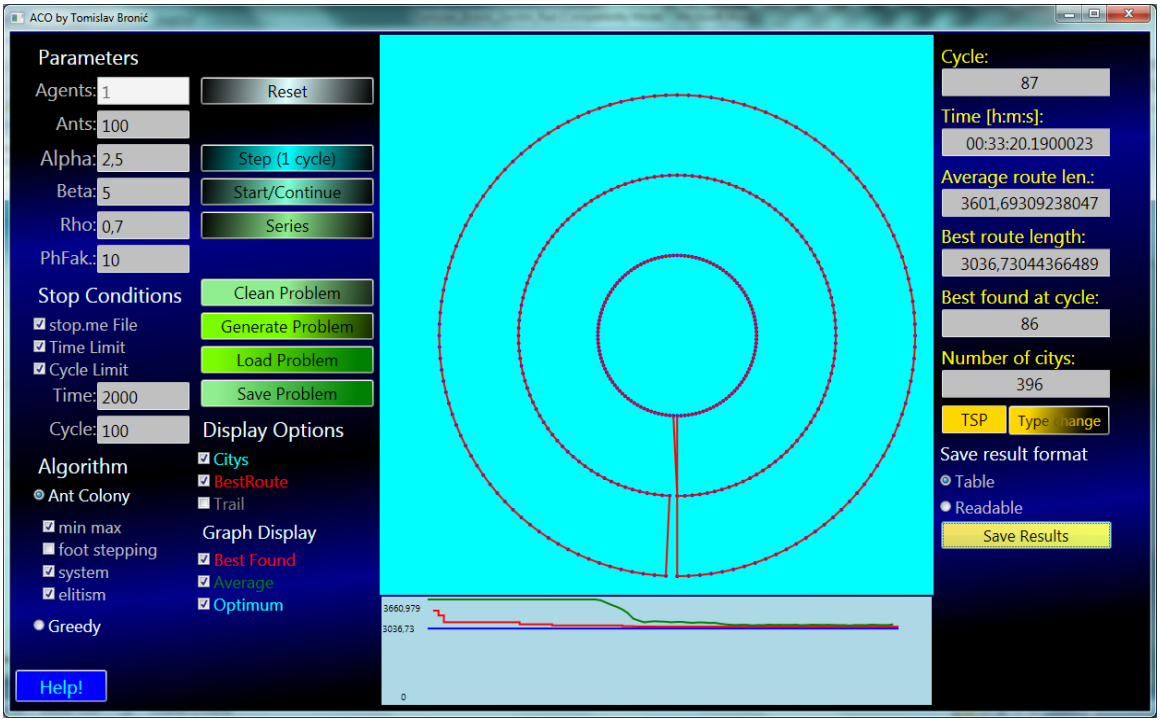
6.2 Koncentrični krugovi sa 396 gradova

Na *slici 6.5* prikazan je problem sa tri koncentrična kruga koji sadrži 396 gradova, te je nad njim provedena optimizacija sa parametrima kakvi su dobiveni kao optimalni na slučajnom problemu.



Slika 6.5 – Koncentrični krugovi sa 396 gradova

Program je pokrenut sa parametrima alpha 2.5, beta 5, 100 mrava, rho 0.7, te feromonski faktor 10. Kao uvjet zaustavljanja korišteno je vremensko ograničenje od 2000 sekundi (33 minute), a rezultati optimizacije prikazani su na *slici 6.6*. Na slici se vidi (desni meni) da optimizacija trajala 33 minute, najbolja ruta je pronađena u predzadnjem ciklusu (86-om, ukupno je program stigao odvrtit 87 ciklusa). Također se sa slike može očitati i kvaliteta rješenja, naime sva tri kruga su iscrtana ravnom linijom (nema preskakivanja gradova), što je dio optimalnog rješenja. No skokovi među krugovima nisu savršeni, naime postoji skok iz trećeg u prvi krug, preko malo nakošene veze. Program u ovom pokušaju nije uspio pronaći optimalno rješenje, no došao je do drugo najboljeg rješenja za taj problem, koje je zanemarivo slabije od najboljeg.

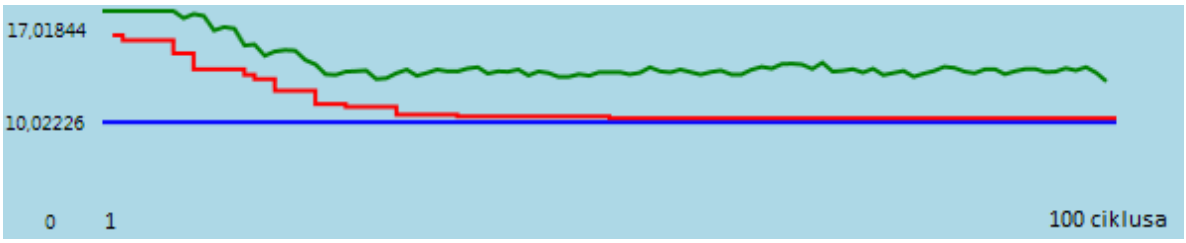


Slika 6.6 – Rezultati optimiranja problema 3 kruga sa 396 gradova

Osim interpretacije rješenja, sa *slike 6.6* moguće je pročitati i prikazati mnogo više. Na dnu slike prikazano je poboljšanje najboljeg rješenja u vremenu, te prosječnu duljinu ruta koje mravi pronalaze u određenim ciklusima.

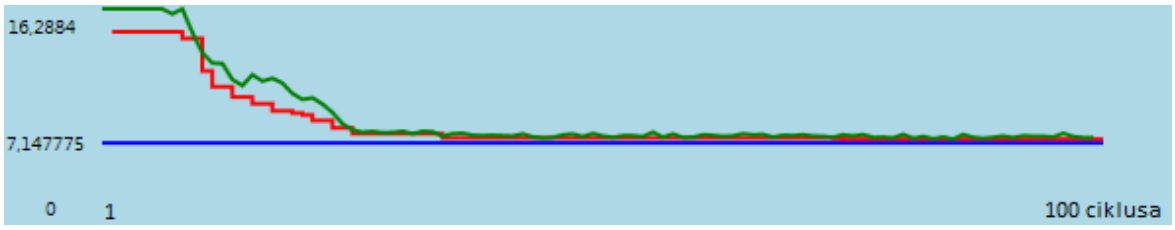
6.3 Interpretacija mjerenja i parametara

Mnogim mjerenjima doneseni su neki zaključci o korelaciji parametara i rezultata optimiranja. Uspoređujući ispis na grafu najboljih ruta sa parametrima, može se doći do zaključka da iznos parametra beta direktno utječe na crvenu liniju, naime veći beta znači vodoravniju crvenu liniju, dok veći alpha znači nižu zelenu liniju (polegnuta na crvenu). Na slijedećim slikama prikazani su rezultati optimiranja TSP-a sa 50 nasumičnih gradova. Horizontalna os je vrijeme, odnosno redni brojevi ciklusa, a vertikalna je duljina najbolje rute. *Slika 6.7* prikazuje graf dobiven parametrima α = 1, β = 1. Na slici se vidi da se vrijednost crvene (najbolja ruta) linije gotovo prepolovila tokom optimizacije, a zelena linija (prosječna duljina rute) drži velik razmak od crvene.



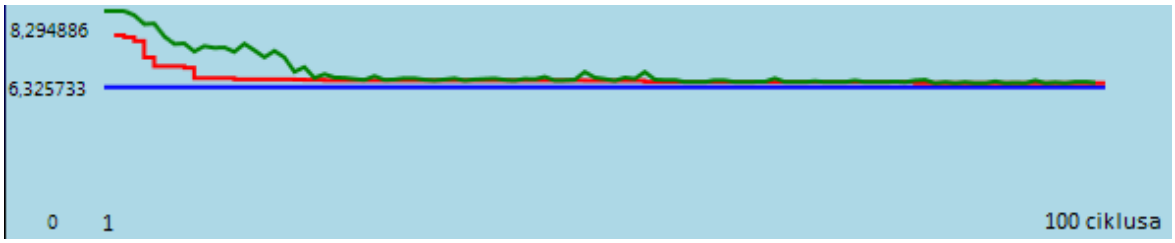
Slika 6.7 – graf najboljih ruta za α=1, β=1

Na slici 6.8 prikazan je graf sa parametrima α=3, β=1, na kojem se vidi da je zelena linija polegnuta na crvenu. Slika 6.9 prikazuje α=1, β=3, pri čemu možemo uočiti "plitku" crvenu liniju, a visoku zelenu. Slika 6.10 prikazuje optimizaciju identičnu onima sa slika 6.7, 6.8 i 6.9, samo sa parametrima α=3, β=3.



Slika 6.8 – graf najboljih ruta za α=3, β=1

Slika 6.9 – graf najboljih ruta za α=1, β=3



Slika 6.10 – graf najboljih ruta za α=3, β=3

Kako bi se dobio najbolji rezultat, potrebno je da zelena linija bude dovoljno visoko iznad crvene, kako program nebi zapeo u lokalnom minimumu (što više gradova u problemu, to je veći alpha dopušten/poželjan), no ipak dovoljno nisko da neki od mrava pronađe neki lokalni (ili opći) minimum. U problemu sa 50 gradova program daje najbolja rješenja za $1.2<\alpha<1.5$, dok u problemu sa 400 gradova najbolja rješenja dolaze uz $2.4<\alpha<3$.

Analogno vrijednostima za alphu, beta ima svoj pojas optimalnih rezultata. Za crvenu liniju je dobro da bude polegnuta, ukoliko želimo što prije pronaći neko "zadovoljavajuće" rješenje, no program se sa velikim beta bliži pohlepnom algoritmu, što rijetko kad znači pronalazak optimalnog rješenja. Za probleme sa 50 gradova optimalnim vrijednostima su se iskazale brojke $3<\beta<4$, dok za probleme sa 400 gradova preporučene vrijednosti bete se nalaze u intervalu $4.5<\beta<6$.

7 Zaključak

U okviru završnog rada osmišljen i izrađen je program *ACOTB*, koji služi za optimiranje rješenja problema trgovačkog putnika na principu ACO algoritma. Prikazana je struktura izvornog teksta programa (dijagrami klasa) i način na koji je program izveden.

Najveći utjecaj na kvalitetu rješenja imaju parametri α i β. Optimalni iznos tih parametara ovisi o problemu čije se rješenje optimira. Točnije, najviše utječu broj gradova i "grupiranost" gradova u problemu. Ukoliko su gradovi grupirani u gomile, te je velika udaljenost među gomilama, parametar beta dobiva na važnosti (veći beta daje bolja rješenja). Kako bi korisnik lakše uočio o kakvom se problemu radi, program *ACOTB* nudi grafički prikaz učitanog problema. Primjer jednog grupiranog problema je automatski generiran problem sa 3 koncentrična kruga, pri čemu svaki koncentrični krug predstavlja jednu gomilu gradova. U suprotnom slučaju, kada su gradovi jednoliko raspoređeni u ravnini, parametar alpha dobiva na važnosti (veći alpha daje bolja rješenja). Primjer problema sa jednolikim rasporedom gradova je automatski generirani *mash* problem (pravilna pravokutna mreža gradova).

Literatura

Radovi:

[1] M. Dorigo, V. Maniezzo, A. Colorni, *The Ant System: Optimization by a colony of cooperating agents*, IEEE Transactions on Systems, Man, and Cybernetics-Part B, Vol. 26, 1996

[2] V. Maniezzo, L. M. Gambardella, F. de Luigi, *Ant Colony Optimization*

[3] M. Dorigo, M. Birattari, T. Stützle, *Artificial Ants as a Computational Intelligence Technique,* IRIDIA – Technical Report Series, September 2006

Internet stranice:

[4] wikipedia; grupa autora; dostupno na Internet adresi: http://en.wikipedia.org/wiki/Ant_colony_optimization

[5] P. Balaprakash, M. A. Montes de Oca, Official ACO metaheuristic site, dostupno na Internet adresi: http://www.aco-metaheuristic.org/about.html

[6] P. Kohut, Genetic and Ant Colony Optimization Algorithms, dostupno na Internet adresi: http://www.codeproject.com/KB/recipes/GeneticandAntAlgorithms.aspx

Dodatak A: Izvorni tekst nekih ključnih funkcija

AntColony -> Start

Osnovna metoda za pokretanje rada kolonije ima dvije inačice, jedna je overload-ana sa nekoliko parametara, radi lakšeg pozivanja rada kolonije. Svi parametri koji se šalju preko metode Start mogu se definirati i pojedinačno, izvan metode Start.

```
public void Start(AlgType type, PheromoneType phtype, StopConditions stopMe)
        {
            this.algType = type;
            this.PheromoneTyp = phtype;
            this.StopCondition = stopMe;
            this.Start();

        }

public void Start()
        {
            if (this.cityList.Count < 2)
            { return; }

            this.startedAt = DateTime.Now;
            if (this.ants.Count != this.antNumber)
            {
                this.ants.Clear();
                for (int i = 0; i < this.antNumber; i++)
                {
                    ants.Add(new Ant());
                }
            }
            while (!this.stopCondition.IsStopConditionTrue())
            {
                this.MakeCyclus();
                this.cyclusCounter++;
                this.CheckBestRoutes();
                this.Vaporize();
                this.algType.DoVersionExtras();
                this.ResetAnts();
            }
            this.endAt = DateTime.Now;

        }
```

Funkcija Start poziva mnoge druge funkcije, neke od njih su prikazane niže.

```
private void MakeCyclus()
        {
            foreach (Ant a in this.ants)
            {
                a.makeCycle();
            }
            return;

        }

private void Vaporize()
        {
            for (int i = 0; i < this.cityList.Count ; i++)
            {
                for (int j = 0; j < this.cityList.Count; j++)
                {
                    this.trail[i, j] = this.rho * this.trail[i, j];
                };
            };
            return;

        }

private void ResetAnts()
        {
            foreach (Ant a in this.ants)
            {
                a.ResetAnt();
            }
```

```
        }
```

Ant -> makeStep

Metoda makeStep je uistinu kompleksna i višeslojna, stoga je ovdje prikazana samo srž te funkcije, dio koda koji vrši njenu osnovnu svrhu (da prebaci mrava u slijedeći grad). Dio koda zaslužan za račun vjerojatnosti (prema formuli opisanoj u poglavlju 3.2) prikazan je ovdje, te niže "roulette wheel" selekcija sljedećeg koraka te pomicanje u izabrani grad.

```
            List<double> probability = new List<double>();
            double sum = 0;
            for (int i = 0; i < AntColony.Instance.cityList.Count; i++)
            {
               if (this.tabuList.Contains(i))
                   {
                       probability.Add(0);
                   }
                   else
                   {

probability.Add(Math.Pow(AntColony.Instance.trail[this.tabuList[tabuList.Count - 1], i],
AntColony.Instance.Alpha) *

Math.Pow(AntColony.Instance.visibility[this.tabuList[tabuList.Count - 1], i],
AntColony.Instance.Beta));
                        sum += probability[probability.Count - 1];
                   }
            }
            for (int i = 0; i < AntColony.Instance.cityList.Count; i++)
            {
                  probability[i] = probability[i] / sum;

            }
```

Roulette wheel selekcija, te odlazak u izabrani grad

```
                        double random = AntColony.Instance.randomizer.NextDouble();
                        if (random == 0)
                        {
                            random = 1;
                        }
                        int placeToGo = -1;
                        while (random > 0)
                        {
                            placeToGo++;
                            random -= probability[placeToGo];
                        }
                        this.tabuList.Add(placeToGo);

                        return;
```

AlgType -> versionExtras

Dodaci algoritmu prikazani su u slijedećem kodu. Najprije se riješava System dodatak, budući da on utječe samo na trag feromona prije dodavanja feromona u prvom ciklusu. Zatim se obrađuje Elitism, koji određuje da će mravi sa boljim rezultatom postavljati mnogo više feromona od lošijih (razlika je mnogo veća nego inače). Na kraju se odrađuju dodaci FootStepping algoritma (selektivno pojačanje feromona na nekim bridovima) i MinMax ograničenja (ograničava iznos feromona na neku minimalnu i maksimalnu vrijednost, kako bi se spriječilo zapinjanje u lokalnim minimumima).

```
internal override void DoVersionExtras()
        {
            if (this.System)
            {
                if (AntColony.Instance.cyclusCounter==1)
                {

AntColony.Instance.SetTrail(AntColony.Instance.ants[0].foundRouteLength);
                }
```

```
            }
            if (this.Elitism)
            {
                AntColony.Instance.ants.Sort(CompareAnts);
                int i = 1;
                foreach (Ant a in AntColony.Instance.ants)
                {
                    a.AddTrail(1 / i);
                    i++;
                    if (i > 10)
                    {
                        break;
                    }
                }
            }
            else
            {
                AntColony.Instance.AddTrail();
            }
            if (this.FootStepping)
            {
                for (int i = 0; i < AntColony.Instance.cityList.Count / 4.0; i++)
                {
                    int city =
AntColony.Instance.randomizer.Next(AntColony.Instance.cityList.Count);
                    int maxTrailIndex = -1;
                    double maxTrail=0;
                    for (int j = 0; j < AntColony.Instance.cityList.Count; j++)
                    {
                        if (AntColony.Instance.trail[city, j] > maxTrail)
                        {
                            maxTrail = AntColony.Instance.trail[city, j];
                            maxTrailIndex = j;
                        }
                    }
                    AntColony.Instance.trail[city, maxTrailIndex] *= 1.05;
                    AntColony.Instance.trail[maxTrailIndex, city] *= 1.05;
                }
            }
            if (this.MinMax)
            {
                double sum=0;
                for (int i = 0; i<AntColony.Instance.cityList.Count;i++)
                {
                    for (int j = 0; j<AntColony.Instance.cityList.Count;j++)
                    {
                        sum+= AntColony.Instance.trail[i,j];
                    }
                }
                double min =(1.0/5.0) * (sum /
(AntColony.Instance.cityList.Count*AntColony.Instance.cityList.Count));
                double max = 15*min;
                for (int i = 0; i<AntColony.Instance.cityList.Count;i++)
                {
                    for (int j = 0; j<AntColony.Instance.cityList.Count;j++)
                    {
                        if (AntColony.Instance.trail[i,j]<min)
                        {
                            AntColony.Instance.trail[i,j]=min;
                        }
                        else
                        {
                            if (AntColony.Instance.trail[i,j]>max)
                            {
                                AntColony.Instance.trail[i,j]=max;
                            }
                        }
                    }
                }
            }
            return;

        }
```

7.1 Dodatak B: Mjerenja

Tablica B.1: α =[0.8, 5], β =[1, 25]

najbolja ruta	nadjena u ciklusu	nadjena u trenutku	α	β
111,713388	23	00:08:51.9218750	0,8	1
64,98306795	21	00:08:05.0625000	0,8	1,5
40,33393895	21	00:08:17.3437500	0,8	2
29,6122689	21	00:07:52.3437500	0,8	2,5
24,71663414	20	00:07:25.6562500	0,8	3
22,28430972	36	00:13:16.9218750	0,8	3,5
20,61181079	34	00:12:34.9843750	0,8	4
19,22458719	25	00:09:15.0312500	0,8	4,5
18,94465139	16	00:05:55.2500000	0,8	5
18,2005641	17	00:06:18.0312500	0,8	5,5
18,14229135	19	00:07:00.7500000	0,8	6
17,60686704	30	00:11:07.9218750	0,8	6,5
17,62370314	21	00:07:47.3750000	0,8	7
17,47241711	21	00:07:46.7343750	0,8	7,5
17,36478437	29	00:10:44.1093750	0,8	8
17,46111661	33	00:12:13.4375000	0,8	8,5
17,29185148	31	00:11:28.7031250	0,8	9
17,20636246	13	00:04:48.5156250	0,8	9,5
17,09155947	32	00:11:51.3750000	0,8	10
102,9945791	66	00:25:40.5937500	1	1
57,1377896	21	00:08:07.6093750	1	1,5
37,71513394	31	00:12:06.5000000	1	2
27,56858055	19	00:07:26.9062500	1	2,5
23,34934294	21	00:08:16.8906250	1	3
21,18245488	17	00:06:14.2500000	1	3,5
19,33567906	38	00:13:58.0937500	1	4
18,85197453	17	00:06:13.8906250	1	4,5
18,02060991	40	00:14:42.4062500	1	5
17,65859997	48	00:17:43.4843750	1	5,5
17,33248073	43	00:15:50.2187500	1	6
17,10004963	68	00:25:02.9531250	1	6,5
17,02762324	96	00:37:29.8437500	1	7
17,05847912	62	00:26:30.9531250	1	7,5
16,99527662	46	00:17:54.1250000	1	8
16,99324713	43	00:16:39.9062500	1	8,5
16,91272399	83	00:41:57.7656250	1	9
16,58949691	57	00:26:18.7656250	1	9,5
16,58229426	31	00:12:01.1562500	1	10
94,7283161	24	00:09:34.5781250	1,2	1
51,71526901	42	00:16:34.8593750	1,2	1,5
34,64100262	79	00:31:32.2031250	1,2	2
25,70607053	32	00:12:22.5000000	1,2	2,5
21,46670004	89	00:32:52.0625000	1,2	3
19,92845959	52	00:19:14.6406250	1,2	3,5

18,85044477	75	00:27:47.7500000	1,2	4
18,20918318	36	00:13:17.6406250	1,2	4,5
17,73872081	80	00:29:39.3437500	1,2	5
17,59529131	91	0:33:42	1,2	5,5
17,22180496	44	00:16:16.4843750	1,2	6
16,8153047	99	00:36:45.2812500	1,2	6,5
17,04244033	88	00:32:35.7656250	1,2	7
16,98980606	79	00:29:29.7031250	1,2	7,5
16,63965445	58	00:22:20.2812500	1,2	8
16,81110476	80	00:30:46.9687500	1,2	8,5
16,56854275	67	00:25:45.4218750	1,2	9
16,8506787	59	00:22:41.8750000	1,2	9,5
16,63400686	41	00:15:48.7500000	1,2	10
84,88700977	22	00:08:23.5468750	1,4	1
48,75488675	21	00:07:53.9062500	1,4	1,5
30,91538033	87	00:32:57.1562500	1,4	2
24,69347327	52	00:19:39.3750000	1,4	2,5
19,97928594	53	00:20:01.8750000	1,4	3
19,73349377	53	00:20:01.9843750	1,4	3,5
18,14857656	74	00:27:59.8437500	1,4	4
17,66285847	99	00:37:27.7343750	1,4	4,5
17,21000432	56	00:21:09.3750000	1,4	5
16,69873981	36	00:17:46.0937500	1,4	5,5
16,91924759	58	00:24:50.8125000	1,4	6
16,88092304	94	00:45:42.6406250	1,4	6,5
16,60391895	39	00:15:43.1250000	1,4	7
16,63097259	75	00:30:01.2187500	1,4	7,5
16,62467761	70	00:27:59.8125000	1,4	8
16,51466037	71	00:28:19.2031250	1,4	8,5
16,52692173	55	00:21:19.0781250	1,4	9
16,16761738	97	00:37:39.0625000	1,4	9,5
16,19777946	24	00:09:15.7812500	1,4	10
79,58957838	73	00:29:26.7187500	1,6	1
23,35884653	76	00:30:43.2031250	1,6	2,5
17,16246292	66	00:26:37.1562500	1,6	4
16,61672079	45	00:17:36.0312500	1,6	5,5
15,90146359	30	00:11:44.5156250	1,6	7
16,2336608	87	00:44:49.7656250	1,6	8,5
16,2500096	85	00:33:38.3750000	1,6	10
15,94918561	60	00:23:45.5937500	1,6	11,5
16,1469848	37	00:16:57.4531250	1,6	13
16,14245478	99	00:53:19.5937500	1,6	14,5
16,04366298	47	00:22:06.7343750	1,6	16
15,996947	95	00:38:58.9062500	1,6	17,5
16,05655508	91	00:37:11.7656250	1,6	19
16,07743301	39	00:17:19.9843750	1,6	20,5
16,18706153	34	00:17:50.2968750	1,6	22
16,13457065	60	00:24:56.2656250	1,6	23,5
16,16511327	96	00:38:05.6562500	1,6	25
74,25574958	23	00:09:19.5937500	1,8	1
22,46758275	26	0:10:16	1,8	2,5

17,34622289	61	00:24:08.2031250	1,8	4
16,38652141	47	00:18:38.4687500	1,8	5,5
16,35526734	43	00:17:05.5312500	1,8	7
15,73624623	85	00:33:38.0625000	1,8	8,5
16,00401005	37	00:14:38.4843750	1,8	10
15,8268546	84	00:33:17.3750000	1,8	11,5
16,01397901	46	0:18:11	1,8	13
15,85612488	71	00:28:10.8593750	1,8	14,5
16,27158595	50	00:19:48.6406250	1,8	16
16,08037481	71	00:26:19.7031250	1,8	17,5
16,12922122	44	00:16:18.0937500	1,8	19
16,17511682	79	00:29:15.2500000	1,8	20,5
16,03384515	25	00:09:15.5468750	1,8	22
16,23465082	56	00:20:47.4375000	1,8	23,5
67,45834481	94	00:36:46.4531250	2	1
21,1325303	30	00:12:12.2500000	2	2,5
16,58950795	42	00:17:17.1250000	2	4
16,1368556	92	00:35:20.2968750	2	5,5
16,14160513	100	00:37:00.4687500	2	7
16,12722238	90	00:33:24.9375000	2	8,5
16,10750702	95	00:35:07.9687500	2	10
16,29012146	67	00:24:51.7968750	2	11,5
16,04442809	61	00:22:36.5468750	2	13
15,98413651	79	00:29:11.9375000	2	14,5
16,09492638	91	00:33:44.8125000	2	16
16,12460518	87	00:32:16.9218750	2	17,5
15,90883862	96	00:35:25.9218750	2	19
15,93447639	77	00:28:28.5000000	2	20,5
15,79350484	79	00:29:13.1250000	2	22
16,06572897	59	00:21:47.4375000	2	23,5
16,02712352	91	00:33:40.6250000	2	25
59,97774305	66	00:25:58.5538656	2,2	1
20,14425018	78	00:29:50.2457593	2,2	2,5
17,08654447	83	00:32:01.4889522	2,2	4
15,42704329	95	00:35:55.4896526	2,2	5,5
15,99715314	75	00:31:04.8763827	2,2	7
15,3939679	93	00:38:23.3936954	2,2	8,5
16,02805384	43	00:16:39.4169919	2,2	10
15,9986912	36	00:14:52.9466731	2,2	11,5
15,95983792	69	00:28:40.7007511	2,2	13
16,02768406	67	00:28:25.9743070	2,2	14,5
16,31346841	12	00:05:05.0901133	2,2	16
15,97909234	99	00:38:43.7049562	2,2	17,5
16,03128047	93	00:36:12.8569079	2,2	19
16,04697564	88	00:36:25.2545418	2,2	20,5
15,98059013	70	00:27:05.3844657	2,2	22
15,93730813	68	00:26:53.4660301	2,2	23,5
16,19595721	76	00:30:12.5538260	2,2	25
58,05875539	66	00:26:30.1783540	2,4	1
21,12543868	68	00:30:36.0095007	2,4	2,5
15,94342474	81	00:38:22.3444697	2,4	4

15,24656924	86	00:39:00.1629537	2,4	5,5
15,48874614	84	00:38:58.9809386	2,4	7
15,35734445	84	00:38:50.0838247	2,4	8,5
15,4231055	83	00:38:20.5464466	2,4	10
15,79271005	68	00:31:17.0900265	2,4	11,5
16,00697284	53	00:23:47.2662688	2,4	13
15,83583277	52	00:23:37.9771499	2,4	14,5
16,25232717	66	00:35:20.0361361	2,4	16
16,29599266	34	00:17:47.6646660	2,4	17,5
16,13090902	36	00:19:19.4178403	2,4	19
16,12777307	37	00:19:49.1992216	2,4	20,5
16,32159716	15	00:07:40.1518899	2,4	22
16,02649698	45	00:25:22.1904838	2,4	23,5
16,40353804	77	00:36:16.0388530	2,4	25
54,5927591	55	00:22:05.8769710	2,6	1
19,71426168	93	00:38:50.2638270	2,6	2,5
15,57805946	98	00:39:00.0722030	2,6	4
15,33806561	90	00:39:00.2909554	2,6	5,5
15,26519212	98	00:38:40.7557053	2,6	7
15,71788513	100	00:39:00.1069530	2,6	8,5
15,69321857	53	00:21:30.3485162	2,6	10
15,839722	67	00:25:53.4648841	2,6	11,5
16,21866605	83	00:32:57.9023168	2,6	13
15,65353583	85	00:34:13.5302848	2,6	14,5
16,37472468	86	00:35:00.4518854	2,6	16
16,11859487	81	00:31:32.5502244	2,6	17,5
15,97900581	97	00:38:56.2181665	2,6	19
16,02758849	56	00:22:06.0949738	2,6	20,5
15,97042758	96	00:37:56.0551332	2,6	22
15,9295122	100	-00:18:43.4022999	2,6	23,5
15,91599585	71	00:27:56.0384530	2,6	25
53,25053137	84	00:33:54.2060375	2,8	1
17,86618964	93	00:39:00.0489522	2,8	2,5
15,70083829	99	00:39:00.1829539	2,8	4
15,06869639	98	00:39:00.1539536	2,8	5,5
15,51457074	97	00:38:28.9585543	2,8	7
15,30993034	93	00:39:00.2489548	2,8	8,5
15,27398225	92	00:38:33.6366141	2,8	10
15,54223008	83	00:34:42.4506551	2,8	11,5
15,96830472	84	00:34:43.8986736	2,8	13
15,80789953	89	00:36:44.9392229	2,8	14,5
16,17221	29	00:12:13.5423892	2,8	16
15,7811574	60	00:23:34.0540996	2,8	17,5
15,86592395	100	00:39:00.2099543	2,8	19
16,34458112	97	00:38:57.3319175	2,8	20,5
15,75817615	78	00:31:11.5099550	2,8	22
16,00762317	73	00:29:17.8234998	2,8	23,5
16,00625844	50	00:20:39.0988603	2,8	25
42,51380816	93	00:36:01.5000000	3	1
17,54378706	98	00:37:48.1406250	3	2,5
15,457019	100	00:38:26.4218750	3	4

15,43924506	100	00:38:24.4531250	3	5,5
15,61013761	97	0:37:16	3	7
15,63225077	96	00:36:52.3125000	3	8,5
16,08040876	78	00:30:04.6875000	3	10
15,99726688	94	00:36:05.9375000	3	11,5
15,65767054	99	00:38:01.0625000	3	13
15,77625255	66	00:25:20.9375000	3	14,5
15,51714385	88	00:33:52.1093750	3	16
15,96899787	86	00:33:01.8125000	3	17,5
16,13120385	76	00:34:02.3437500	3	19
16,32819569	47	00:31:19.2656250	3	20,5
16,25934046	88	00:36:16.2031250	3	22
15,90894023	58	00:26:30.4218750	3	23,5
16,15477713	80	00:36:28.2187500	3	25
38,12615146	71	00:28:01.4191717	3,2	1
17,99925319	88	00:38:24.6488184	3,2	2,5
15,96841847	90	00:39:00.0628439	3,2	4
15,49248316	90	00:38:33.1983073	3,2	5,5
15,37830993	90	00:39:00.1708501	3,2	7
15,13938458	95	00:38:56.2676269	3,2	8,5
15,30407778	86	00:37:02.2541057	3,2	10
15,7219665	78	00:33:35.8573004	3,2	11,5
15,74577793	84	00:36:48.9333437	3,2	13
15,91405486	82	00:35:59.5465190	3,2	14,5
16,44095991	64	00:28:40.8819754	3,2	16
15,84205916	74	00:32:40.6591433	3,2	17,5
15,80813972	89	00:39:00.2508547	3,2	19
15,96603594	56	00:24:10.6819743	3,2	20,5
15,64157258	75	00:32:33.2487194	3,2	22
16,15009702	88	00:38:27.5209827	3,2	23,5
16,0573054	80	00:34:37.6658357	3,2	25
37,9536659	99	00:39:00.1108467	3,4	1
17,87182668	95	00:39:00.1058464	3,4	2,5
16,11447164	88	00:38:50.6683066	3,4	4
15,671199	78	00:38:38.9491381	3,4	5,5
15,42844673	88	00:38:57.8557177	3,4	7
15,95617446	81	00:38:58.4377510	3,4	8,5
15,86328043	88	00:39:00.1928514	3,4	10
16,04957332	40	00:19:11.2848497	3,4	11,5
15,99627353	87	00:39:00.0758447	3,4	13
15,59643998	82	00:38:10.6970204	3,4	14,5
15,7359034	86	00:36:40.6638708	3,4	16
15,97248231	39	00:18:18.7788465	3,4	17,5
16,10448865	81	00:38:13.8011980	3,4	19
15,89193674	66	00:30:53.5520172	3,4	20,5
15,81593783	85	00:36:37.9327146	3,4	22
15,96525511	68	00:31:12.3790940	3,4	23,5
16,160197	40	00:17:49.7241847	3,4	25
34,65771765	100	00:39:00.1158470	3,6	1
17,96124302	74	00:39:00.1238474	3,6	2,5
16,05870813	82	00:38:50.0632719	3,6	4

15,63317298	89	00:39:00.3018576	3,6	5,5
15,59151837	82	00:37:37.7361351	3,6	7
15,50595335	84	00:39:00.1408484	3,6	8,5
15,44723205	81	00:38:27.9540074	3,6	10
15,76963633	71	00:35:51.5290604	3,6	11,5
15,92260187	75	00:34:43.5641730	3,6	13
15,95427536	66	00:30:50.2348274	3,6	14,5
15,77141648	72	00:36:23.3508805	3,6	16
17,58907989	1	00:00:24.2943896	3,6	17,5
16,10023981	79	00:31:47.0030744	3,6	19
15,93288604	86	00:37:42.2843952	3,6	20,5
15,96342666	73	00:31:36.3084626	3,6	22
16,03444147	83	00:37:52.2429648	3,6	23,5
15,98162616	82	00:39:00.1058464	3,6	25
35,83440349	100	00:37:26.0894690	3,8	1
18,03451242	98	00:37:50.7928819	3,8	2,5
15,925209	88	00:35:44.3406493	3,8	4
15,5871513	93	00:37:48.6087570	3,8	5,5
15,48534244	97	00:39:00.1598495	3,8	7
15,73654535	88	00:35:42.0605188	3,8	8,5
15,52270217	97	00:38:44.9729809	3,8	10
15,64135682	93	00:36:59.5579515	3,8	11,5
15,52522866	99	00:37:56.9702352	3,8	13
15,74626546	94	00:37:04.4672141	3,8	14,5
15,44889447	94	00:36:26.7840769	3,8	16
16,0499196	73	00:27:59.1080395	3,8	17,5
16,09364771	94	00:36:31.8093643	3,8	19
16,02452709	39	00:15:09.7180329	3,8	20,5
15,98938754	78	00:30:52.2299415	3,8	22
15,83903503	82	00:33:15.9581623	3,8	23,5
15,62616563	94	00:37:37.1861037	3,8	25
38,66278165	93	00:39:00.0048225	4	1
19,12330494	89	00:38:20.0055528	4	2,5
16,13837086	85	00:38:37.1295322	4	4
15,25943737	85	00:36:42.9960042	4	5,5
15,47897424	73	00:37:37.5151224	4	7
15,38515634	76	00:36:03.6757551	4	8,5
15,82969562	82	00:39:00.0368424	4	10
15,635716	79	00:37:23.2333056	4	11,5
16,09931213	54	00:26:07.6666654	4	13
15,94051001	86	00:38:38.2105940	4	14,5
15,54694638	79	00:37:51.4589200	4	16
15,86461286	55	00:27:12.7073674	4	17,5
17,17604298	1	01:37:41.5523307	4	19
15,80514395	77	00:32:14.4896465	4	20,5
15,47101378	81	00:38:18.4784655	4	22
15,8234649	87	00:39:00.1228474	4	23,5
16,17959367	83	00:37:45.1925616	4	25
37,87618142	100	00:38:07.9198433	4,5	1
18,34415887	83	00:33:00.5232795	4,5	2,5
16,12346218	90	00:36:14.3243642	4,5	4

16,0455725	94	00:38:37.2485390	4,5	5,5
15,67419575	94	00:38:43.1568770	4,5	7
15,43579892	95	00:39:00.0508433	4,5	8,5
15,57917943	96	00:39:00.0708444	4,5	10
16,14955996	71	00:28:54.6292151	4,5	11,5
15,5493625	95	00:38:35.7264520	4,5	13
15,58311117	79	00:32:29.3214948	4,5	14,5
15,59696681	96	00:37:28.4866061	4,5	16
16,33806316	56	00:22:49.0432867	4,5	17,5
15,77356332	91	00:35:07.6505507	4,5	19
15,97130228	84	00:33:00.3142675	4,5	20,5
15,84897435	63	00:25:10.9944240	4,5	22
15,64498038	98	00:39:00.2278533	4,5	23,5
16,27828526	38	00:15:06.9428742	4,5	25
37,95353986	95	00:38:53.2814561	5	1
18,94893969	90	00:37:35.1289860	5	2,5
16,37473568	58	00:24:29.9380757	5	4
16,06751568	57	00:25:28.9754524	5	5,5
15,92934379	81	00:36:12.3062488	5	7
15,7207789	79	00:35:12.6158347	5	8,5
15,341538	88	00:38:48.3791756	5	10
15,59560016	84	00:37:23.3433119	5	11,5
15,35789892	94	00:39:00.2298535	5	13
15,77382635	82	00:36:02.8687090	5	14,5
15,54185088	84	00:39:00.1288295	5	16
15,85959033	75	00:34:46.8283597	5	17,5
15,81127732	78	00:36:36.0336059	5	19
15,94941146	75	00:32:51.7397771	5	20,5
15,73614373	84	00:37:54.8441137	5	22
16,20634226	68	00:29:14.4083464	5	23,5
15,7884927	88	00:38:23.6257599	5	25

Nakon otkrivenog potencijalnog minimuma u širokom tlocrtu parametara, napravljena su daljnja mjerenja uz manje izmjene parametara (rho = 0.7, umjesto 0.5 kao prije, te alpha i beta manje variraju).

Tablica B.2: α =[2.2, 3.2], β =[4, 8]

najbolja ruta	nadjena u ciklusu	nadjena u trenutku	alpha	beta
16,05167917	98	00:38:32.6471322	2,2	4
15,5593373	102	00:38:45.7043861	2,2	4,5
16,51248388	78	00:30:38.0619761	2,2	5
15,65723177	94	00:37:22.8836447	2,2	5,5
15,76686536	93	00:37:21.9788409	2,2	6
15,94996776	93	00:36:57.1435386	2,2	6,5
16,03263	49	00:19:28.2888154	2,2	7
15,78288485	78	00:31:12.3977177	2,2	7,5
15,97821957	87	00:35:00.3926574	2,2	8
16,535707	95	00:37:41.3026578	2,4	4
15,50355185	100	00:38:55.6986993	2,4	4,5
15,53787003	97	00:39:00.0137597	2,4	5

15,44646922	96	00:39:00.1927622	2,4	5,5
15,63081549	95	00:38:59.2237487	2,4	6
15,36235049	95	00:39:00.0717606	2,4	6,5
16,05726387	95	00:39:00.2707634	2,4	7
15,47867327	93	00:37:45.5157168	2,4	7,5
15,97084601	86	00:34:58.4413777	2,4	8
16,09522464	91	00:38:57.7747284	2,6	4
15,98650132	94	00:38:33.9613950	2,6	4,5
15,39640754	93	00:39:00.2687633	2,6	5
15,61959605	92	00:36:59.4600720	2,6	5,5
15,59852392	95	00:38:23.0032416	2,6	6
15,45771643	101	00:38:57.5167248	2,6	6,5
15,52407929	95	00:39:00.1657619	2,6	7
15,37829972	94	00:38:16.8521555	2,6	7,5
15,41584653	94	00:37:11.4502398	2,6	8
15,85594349	93	00:39:00.0247598	2,8	4
15,47656737	97	00:39:00.1547617	2,8	4,5
15,24774557	98	00:37:42.4486738	2,8	5
15,45627434	97	00:39:00.1257613	2,8	5,5
15,60880546	86	00:34:01.2705773	2,8	6
15,73857738	100	00:38:31.2843576	2,8	6,5
15,4296388	93	00:39:00.2337628	2,8	7
15,51386968	88	00:37:08.9062042	2,8	7,5
15,77813485	90	00:38:38.1254533	2,8	8
15,53860767	96	00:39:00.1487616	3	4
15,79229847	100	00:39:00.0857607	3	4,5
15,45682745	98	00:38:21.1212152	3	5
15,16799342	99	00:38:55.7236997	3	5,5
15,54577997	100	00:38:29.3123299	3	6
15,28650137	99	00:38:39.5044726	3	6,5
15,73934434	96	00:38:14.4941224	3	7
15,31829193	98	00:38:28.2343148	3	7,5
15,81505564	99	00:39:00.1207613	3	8
15,71951117	103	00:39:00.0097597	3	4
15,60980487	104	00:39:00.2147625	3	4,5
15,45538957	105	00:39:00.0927608	3	5
15,86523571	102	00:38:17.0891588	3	5,5
15,13377729	104	00:39:00.1947622	3	6
15,33335085	105	00:38:55.1716919	3	6,5
15,45296393	103	00:38:45.7835605	3	7
15,78262799	103	00:38:17.9261706	3	7,5
15,49106738	104	00:39:00.1157612	3	8
15,63879036	99	00:38:46.9765772	3	4
15,40168091	95	00:38:49.8536175	3	4,5
15,58299859	96	00:39:00.2257627	3	5
15,27144444	92	00:37:28.9914855	3	5,5
15,36767319	96	00:38:58.8297431	3	6
15,53990228	93	00:37:27.2324608	3	6,5
15,44154592	96	00:39:00.2637632	3	7
15,86839195	97	00:39:00.1287614	3	7,5
15,41850013	94	00:38:06.5510113	3	8

15,68948722	92	00:39:00.1217612	3	4
15,80220879	89	00:38:55.9297026	3	4,5
15,88770874	89	00:38:40.3724848	3	5
15,95969053	91	00:39:00.1717619	3	5,5
15,41755924	90	00:39:00.1497616	3	6
15,5096463	91	00:39:00.2547631	3	6,5
15,49612659	91	00:38:58.7267417	3	7
15,50123454	89	00:38:40.9904934	3	7,5
15,30623613	95	00:38:46.2995677	3	8
15,94557602	89	00:38:23.1692440	3,2	4
15,3515948	93	00:38:12.7050974	3,2	4,5
15,31094878	93	00:37:11.0482342	3,2	5
15,46901859	91	00:37:03.4391277	3,2	5,5
15,51732955	95	00:38:34.9074083	3,2	6
15,49921494	92	00:37:25.4844363	3,2	6,5
15,57448795	94	00:38:09.5410531	3,2	7
15,9617668	87	00:35:41.2629773	3,2	7,5
15,39294194	95	00:39:00.0427601	3,2	8
16,03581229	91	00:39:00.0477602	3,2	4
16,01822222	93	00:37:20.0773606	3,2	4,5
15,37167402	96	00:38:05.1969923	3,2	5
15,28538186	100	00:38:45.2305528	3,2	5,5
15,46352867	92	00:38:19.3881910	3,2	6
15,27363536	91	00:39:00.1307614	3,2	6,5
15,34427977	97	00:38:24.6012640	3,2	7
15,7530402	97	00:38:51.7526440	3,2	7,5
15,43261864	88	00:35:10.3645447	3,2	8
16,08424797	91	00:39:00.1577618	3,2	4
15,90563473	91	00:39:00.2627632	3,2	4,5
15,62520554	91	00:39:00.0677605	3,2	5
15,48152753	88	00:38:17.6541668	3,2	5,5
15,50758185	87	00:37:46.8987361	3,2	6
15,8269519	91	00:39:00.0377601	3,2	6,5
15,99911357	91	00:39:00.0237599	3,2	7
15,50824435	93	00:39:00.0237598	3,2	7,5
15,22767879	96	00:39:00.0087596	3,2	8