SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 6490

RADNI OKVIR ZA RJEŠAVANJE PROBLEMA TRGOVAČKOG PUTNIKA PARALELNIM ALGORITMOM KOLONIJE MRAVA

Ivan Rissi

Zagreb, lipanj 2020.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 6490

RADNI OKVIR ZA RJEŠAVANJE PROBLEMA TRGOVAČKOG PUTNIKA PARALELNIM ALGORITMOM KOLONIJE MRAVA

Ivan Rissi

Zagreb, lipanj 2020.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 13. ožujka 2020.

ZAVRŠNI ZADATAK br. 6490

Pristupnik:	**Ivan Rissi (0036506108)**
Studij:	Računarstvo
Modul:	Računarska znanost
Mentor:	prof. dr. sc. Marin Golub

Zadatak: **Radni okvir za rješavanje problema trgovačkog putnika paralelnim algoritmom kolonije mrava**

Opis zadatka:

Navesti podjelu i opisati mravlje algoritme. Definirati i opisati problem trgovačkog putnika. Opisati mravlji algoritam prilagođen rješavanju općenitog problema trgovačkog putnika. Razmotriti mogućnost paralelnog rješavanja problema koristeći višedretvenost te navesti prednosti i nedostatke paralelnog rada u odnosu na sekvencijalno rješavanje problema. Ostvariti i opisati programski sustav za zadavanje i rješavanje problema trgovačkog putnika te za određivanje vrste mravljeg algoritma i njegovih parametara. Usporediti dobivene rezultate s poznatim rješenjima iz literature. Uz rad priložiti izvorne tekstove programa i citirati korištenu literaturu.

Rok za predaju rada: 12. lipnja 2020.

SADRŽAJ

1. Uvod

Od samog početka postojanja i korištenja računala, njihov razvoj rapidno napreduje. Količina tranzistora koja se može smjestiti u integrirani krug se prema Mooreovom zakonu udvostručava svake dvije godine. Unatoč tom eksponencijalnom rastu, postoje brojni problemi koji će za klasična računala uvijek biti računski prezahtjevni. Rješavanje nekih od tih problema deterministčkim bi algoritmima moglo potrajati stotinama, tisućama, milijunima godina, čak i redovima veličina mnogo većim od toga.

Iz tog je razloga čovjek počeo promatrati pojave u prirodi i modelirati ih na način upotrebljiv na računalu. Tako su otkriveni prirodom inspirirani algoritmi i jedna od grana računarstva — evolucijsko računarstvo, koje koristi nedeterminističke metode inspirirane prirodom za rješavanje problema čije je rješavanje deterministčkim algoritmima predugotrajno.

U ovom se radu promatra jedan od takvih problema — problem trgovačkog putnika. Promatra se i način na koji bi se mravi snašli u problemu trgovačkog putnika kad bi se skalirao na njihovu razinu te se model njihovog ponašanja koristi u dobivanju konačnog rješenja.

2. Evolucijsko računarstvo

Evolucijsko računarstvo (engl. *evolutionary computation*) je podgrana mekog računarstva (engl. *soft computing*) — velike podgrane umjetne inteligencije koja se bavi aproksimacijskim izračunom kako bi donijela neprecizna, ali prihvatljiva rješenja za vrlo kompleksne računske probleme, koji su često nepolinomijalne vremenske složenosti. Iz tog razloga se do optimalnih rješenja tih problema klasičnim deterministčkim metodama ne bi moglo doći u razumnom vremenu.

Riječ je o velikoj skupini nedeterminističkih algoritama koji svoju inspiraciju nalaze u biološkoj evoluciji i pojavama u prirodi. Zajedničko svojstvo svih algoritama evolucijskog računanja leži u tome što ne kreću od jedne početne točke, već od početne populacije jedinki, koja ovisno o podskupini algoritama evoluira (stvara nove i bolje jedinke te odbacuje stare i lošije jedinke) te tako teži prema konačnom rješenju ili međusobno surađuje i gradi put prema konačnom rješenju.

2.1. Evolucijski algoritmi

Evolucijski algoritmi su jedna od dvije najveće podgrane evolucijskog računanja. Riječ je o skupini nedeterminističkih metaheurističkih optimizacijskih algoritama koja se temelji na klasičnom pogledu na kromosom kao niz gena, gdje se na gene gleda kao na podatke[6]. Oblikovani su pomoću Darwinove teorije evolucije koja govori da se sve jedinke organizama razvijaju i podliježu prirodnoj selekciji — preživjet će i razmnožiti se one najbolje i najspremnije jedinke, dok se slabije jedinke odbacuju (engl. *surival of the fittest*), a spremnost (engl. *fitness*) — dobrota svake jedinke računa se pomoću funkcije dobrote (engl. *fitness function*).

Generički evolucijski algoritam izvodi se na sljedeći način:

1. Nasumično generiranje inicijalne (prve) populacije jedinki.
2. Ponavljanje sljedećih koraka do kraja izvođenja algoritma:

(a) izračunavanje vrijednosti dobrote svake od jedinki u populaciji;

(b) odabir najspremnijih jedinki (onih s najvećom vrijednošću dobrote) za reprodukciju;

(c) stvaranje novih jedinki križanjem postojećih odabranih jedinki i mutacijom njihovih gena;

(d) pridruživanje novonastalih jedinki postojećima i odbacivanje onih s najmanjom vrijednošću dobrote.

Algoritam završava ispunjenjem jednog od kriterija [14]:

- Ukoliko nije došlo do poboljšanja u prethodnih X generacija (iteracija)
- Kada se dostigne predefinirani broj generacija (iteracija)
- Kada vrijednost funkcije dobrote dosegne neku predefiniranu graničnu vrijednost

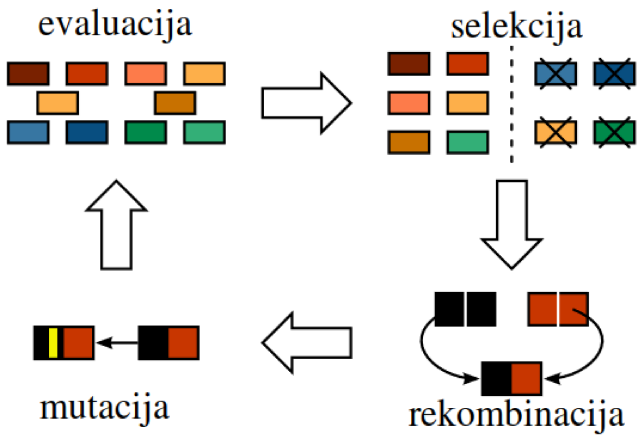
2.1.1. Genetski algoritam

Najpoznatiji predstavnik skupine evolucijskih algoritama je genetski algoritam. U genetskom algoritmu, svaka od jedinki je potencijalno rješenje te ima skup svojstava (gena) koji se mogu mijenjati i mutirati. Ta svojstva su najčešće zapisana u obliku binarnih nizova, ali moguće je koristiti i druge oblike zapisa, odnosno prikaza rješenja. Upravo zbog njihovog zapisa u obliku nizova moguće je jednostavno poravnavanje tih nizova, a time i jednostavno križanje (rekombinacija) jedinki.

Prvi korak genetskog algoritma je generiranje početne populacije koja se sastoji od predefiniranog broja jedinki (potencijalnih rješenja) koje su najčešće nasumične, a ponekad ih se može smjestiti u blizinu potencijalnih optimalnih rješenja.

Tijekom svake generacije jedinki (iteracije), svakoj jedinki se pomoću funkcije dobrote izračunava vrijednost dobrote po kojoj se populacija potom sortira. Funkcija dobrote na temelju podataka zapisanih u genetskom zapisu računa kvalitetu rješenja — što je vrijednost dobrote veća, to je rješenja bolje. Iz tako sortirane populacije, biraju se najbolje jedinke, čiji broj odgovara predefiniranom parametru.

Nakon selekcije najboljih jedinki, nad njima se provode dvije operacije — križanje (rekombinacija) i mutacija. Rekombinacija se provodi nad dvije jedinke odabrane iz gore navedenog skupa — roditeljske jedinke. Genetski zapis obiju jedinki se poravnava te se kombiniranjem oba zapisa stvara novi zapis — jedinka dijete. Kod svake



Slika 2.1: Slikovni prikaz genetskog algoritma [2]

rekombinacije se biraju novi roditelji te se ona provodi sve dok se ne generira potreban broj novih jedinki kako bi se popunio kapacitet ukupne populacije. Nakon rekombinacije se provodi mutacija, gdje svaki od podataka u nizu ima malu vjerojatnost za promjenu svoje vrijednosti.

U konačnici se računa vrijednost dobrote svake od novih jedinki te se stvara novi skup jedinki kao unija skupa starih i skupa novih jedinki, a dio jedinki s najmanjom vrijednošću dobrote u tom skupu se odbacuje kako veličina populacije ne bi premašila svoj kapacitet. Nakon svake od iteracija, ispituju se kriteriji završetka algoritma te se, ukoliko je neki od njih ispunjen, algoritam prekida. U suprotnom, algoritam se ponovo provodi s novogeneriranom populacijom u ulozi početne populacije.

2.1.2. Ostali evolucijski algoritmi

Među ostale evolucijske algoritme spadaju genetsko programiranje, evolucijska strategija, evolucijsko programiranje, diferencijalna evolucija i drugi.

Evolucijska strategija je algoritam sličan genetskom, uz nekoliko manjih razlika. Neke od tih razlika su reprezentacija genetskog zapisa realnim brojevima umjesto cijelih, kao i veći fokus na mutaciju nego rekombinaciju [15].

Genetsko programiranje je metodologija koja traži računalni program koji rješava zadatak ili problem, zadan od strane korisnika, pomoću evolucijskih principa.

Evolucijsko programiranje je metodologija slična genetskom programiranju u kojoj je struktura programa koji se optimira fiksna, dok su brojčani parametri promjenjivi i mogu evoluirati.

2.2. Algoritmi inteligencije roja

Druga od dvije najveće podgrane evolucijskog računanja su algoritmi inteligencije roja. Za razliku od evolucijskih algoritama, algoritmi inteligencije roja baziraju se na kolektivnoj inteligenciji kolonija životinja poput mrava, pčela, jata riba ...

Algoritmi prije svega modeliraju način samoorganizacije i zajedničkog djelovanja cijele kolonije te međusobne komunikacije između jedinki kako bi se riješio zadani problem.

Svaki će sustav inteligencije roja imati sljedeća svojstva [4]:

- sastoji se od velikog broja jedinki;
- jedinke su relativno homogene — sve su identične (Algoritam kolonije mrava) ili postoji malen broj tipologija (Algoritam kolonije pčela);
- interakcija među jedinkama se bazira na jednostavnim ponašajnim pravilima koja iskorištavaju isključivo lokalnu informaciju koju jedinke razmjenjuju direktno (ples pčela) ili preko okoline (trag feromona);
- sveukupno ponašanje sustava proizlazi iz interakcija između jedinki i okoline te jedinki međusobno — kolonija se samoorganizira;
- sposobnost koordinacije bez prisustva vođe.

Zahvaljujući tim svojstvima, moguće je oblikovati skalabilan, paralelan sustav koji je otporan na greške. Skalabilnost označava mogućnost proizvoljnog mijenjanja veličine sustava bez promjene načina međudjelovanja jedinki. Sustav je moguće paralelizirati zato što svaka jedinka djeluje zasebno te jedinke vrlo često djeluju na različitim mjestima. Otpornost na greške postoji upravo zbog decentraliziranosti sustava i neovisnosti jedinki — ukoliko dođe do greške s jedinkom, ona se uklanja i zamjenjuje je druga (nova) jedinka.

2.2.1. Algoritam optimizacije rojem čestica

Algoritam optimizacije rojem čestica je metoda koja iterativno pomiče jedinke — čestice — kroz prostor stanja problema prema najboljim rješenjima. Čestice se miču prema formulama oblikovanima pomoću pozicije konkretne čestice, njene brzine te njene vlastite najbolje pozicije, kao i najboljim pozicijama nađenim od strane cijelog roja.

Osnovna varijanta algoritma započinje generiranjem roja čestica — svaka od čestica predstavlja jedno potencijalno rješenje. Te se čestice pomiču kroz prostor stanja na temelju nekoliko jednostavnih formula.

Smjer kretanja jedne čestice određen je kombinacijom podatka o najboljoj poznatoj poziciji te čestice i podatka o najboljoj poznatoj poziciji cijelog roja. Kada se pronađe nova najbolja pozicija, ona će nadomjestiti staru te umjesto nje sudjelovati u navigiranju čestica kroz prostor stanja.

Proces se, u nadi za pronalaskom zadovoljavajućeg rješenja, ponavlja dok se ne zadovolji neki od kriterija završetka algoritma:

- dosegnut je predefiniran ukupan broj iteracija ili
- pronađeno je rješenje sa zadovoljavajućom objektivnom vrijednošću funkcije.

2.2.2. Algoritam optimizacije umjetnom kolonijom pčela

Algoritam optimizacije umjetnom kolonijom pčela imitira način na koji pčele medarice pronalaze hranu kako bi se riješio traženi računski problem. Algoritam je 2005. godine predstavio Derviş Karaboğa [8].

Umjetna kolonija pčela, koja se oblikuje algoritmom, sastoji se od tri vrste pčela:

- pčele radnice (engl. *employed bees*) — lete do svog izvora hrane te pri povratku u košnicu plešu intenzitetom proporcionalnim količini nektara na svom izvoru;
- pčele promatrači (engl. *onlooker bees*) — promatraju plesove te odlučuju o izvorima hrane u ovisnosti o plesovima;
- pčele izviđači (engl. *scout bees*) — pronalaze nove izvore hrane.

Algoritam započinje dodjeljivanjem izvora hrane svakoj od pčela radnica — pozicija svakog izvora hrane predstavlja potencijalno rješenje problema, dok je količina nektara na tom izvoru proporcionalna kvaliteti rješenja. Nakon što je izveden početni korak, sljedeći koraci izvode se u petlji:

1. Svaka pčela radnica odlazi do svog izvora hrane, traži novi najbliži i evaluira njegovu količinu nektara te potom pleše u košnici
2. Svaka pčela promatrač promatra plesove te bira jedan od izvora hrane na temelju plesova, odlazi do tog izvora te bira jedan od susjednih izvora i evaluira njegovu količinu nektara

3. Određuju se izvori hrane koji će se napustiti, njihove pčele radnice postaju izviđači, a napušteni izvori zamjenjuju se novima koji su pronašle pčele izviđači

4. Bilježi se podatak o trenutno najboljem izvoru hrane

Ovaj postupak se ponavlja sve dok se ne zadovolji neki od uvjeta završetka algoritma.

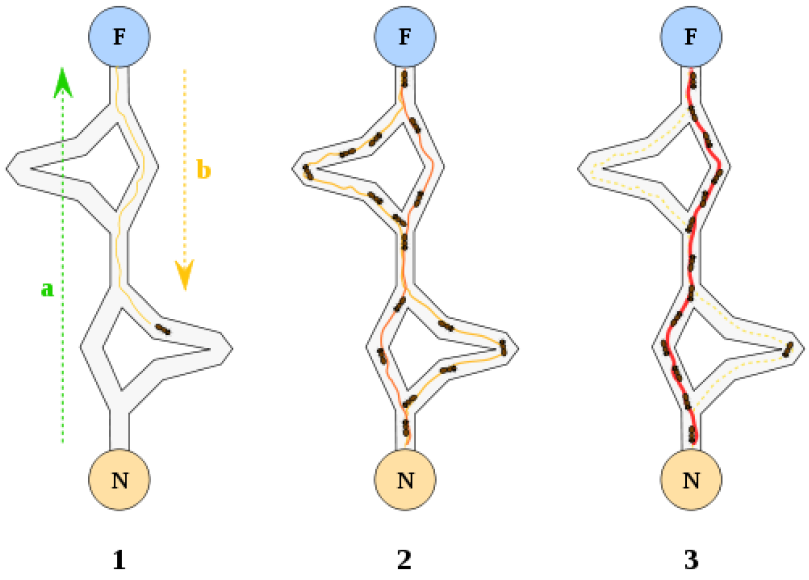
3. Algoritam optimizacije kolonijom mrava

Algoritam optimizacije kolonijom mrava (engl. *Ant colony optimization*, u daljnjem tekstu „mravlji algoritam") je danas najzastupljeniji skup algoritama iz skupine algoritama inteligencije roja koji se temelji na načinu na koji mravi u prirodi nalaze najkraći put između hranilišta i mravinjaka.

3.1. Povijest algoritma

Pokus koji su 1990. godine proveli belgijski biolog Jean-Louis Deneubourg i njegovi suradnici pokazao je da će kolonija argentinskih mrava, ukoliko su im ponuđena dva puta između hranilišta i mravinjaka, kolektivno težiti odabiru onog kraćeg. Razlog leži u tome što mravi hodajući luče *feromone*. Naime, mrav se ponaša prema probabilističkom modelu gdje će vjerojatnost da odabere neki put biti proporcionalna koncentraciji feromona na tom putu. Ukoliko je više mrava prošlo istim putem, trag feromona na tom će putu biti jači. Također, feromoni koji se nalaze na putu s vremenom isparavaju pa tako trag feromona na tom putu slabi.

Kada se mravima ponude dva puta koji se uvelike razlikuju u dužini, budući da trag feromona ne postoji niti na jednom od puteva, oni će pri prvom odabiru slučajno odabrati između dva puta te će njihova razdioba na ta dva puta težiti uniformnoj. Međutim, budući da je jedan od puteva mnogo kraći od drugog, mravi će tim putem prije doći od mravinjaka do hranilišta te će se, nakon prikupljanja hrane, vratiti do mjesta gdje moraju odabrati između dva puta prije nego što su mravi koji šeću dužim putem stigli do istog. Stoga će trag feromona na kraćem putu biti intenzivniji pa će mravi težiti odabiru kraćeg puta te time dodatno pojačati njegov trag feromona. Ostali će mravi također nastaviti odabirati put s jačim intenzitetom feromona čime će se on dodatno pojačavati, dok će trag feromona na dužem putu oslabiti. Nakon nekog vremena, svi mravi će se kretati kraćim putem.



Slika 3.1: Vizualizacija Deneubourgova pokusa [13]

Koristeći rezultate Deneubourgova pokusa, Marco Dorigo i njegovi suradnici su 1991. godine razvili i predstavili populacijsku metaheuristiku te je nazvali „Algoritam optimizacije kolonijom mrava".

3.2. Generički mravlji algoritam

U mravljem algoritmu, umjetni mravi traže rješenje optimizacijskog problema. To rješenje je rijetko optimalno, ali je zato zadovoljavajuće i dobiveno u prihvatljivom vremenu. Svaki optimizacijski problem mora biti preslikan u odgovarajući težinski graf. Rješenje traže šetajući grafom u kojem je odabir sljedećeg čvora oblikovan prema modificiranom modelu iz Deneubourgova pokusa — pojavljuje se n puteva umjesto njih 2, međutim, koncept ostaje isti. Umjetni mravi luče feromone te koriste postojeći trag feromona kako bi odabrali put kojim će proći — vjerojatnost odabira jednog puta (brida grafa) je proporcionalna intenzitetu feromona na istom.

Prije početka izvođenja algoritma, učitavaju se potrebni parametri (3.2.1) te se inicijalizira početni intenzitet feromona koji ovisi o podvrsti algoritma i implementaciji. Potom se ulazi u petlju u kojoj se izvode sljedeća tri koraka [3]:

1. Konstrukcija mravljih puteva (3.2.2)
2. *Daemon* akcije (opcionalno) (3.2.3)
3. Ažuriranje feromona (3.2.4)

sve do ispunjenja nekog od kriterija završetka algoritma.

3.2.1. Parametri mravljeg algoritma

Način rada mravljeg algoritma ovisi o nekoliko parametara. Parametri se mogu podijeliti na dvije skupine:

1. Cjelobrojni parametri koji označavaju strukturu samog algoritma te njegov način izvođenja:
   - Broj mrava m – povećanjem broja mrava, rješenje će biti bolje zato što će više feromona biti izlučeno diljem grafa. Međutim, povećanjem broja mrava usporavamo algoritam zato što je potrebno izvršiti više izračuna.
   - Broj iteracija n – slično kao i u slučaju s brojem mrava, rješenje će povećanjem broja iteracija biti bolje (osim u slučaju konvergencije u lokalni optimum), ali će izvođenje algoritma u konačnici biti sporije.
2. Realni koeficijenti koji se koriste u formulama za izračun vjerojatnosti i promjene koncentracije feromona:
   - Faktor prikupljanja informacija (engl. *information elicitation factor*) $\alpha \in \mathbb{R}^+$ – označava mjeru u kojoj se uvažava trag feromona pri odabiru puta. Povećanjem faktora α, mravi će više težiti odabiru puta na temelju koncentracije feromona te će time surađivati u većoj mjeri. Međutim, ukoliko je faktor α prevelik, povećava se mogućnost konvergencije u lokalni optimum, što nije željeno rješenje. Nasuprot tome, smanjenjem faktora α se smanjuje mjera u kojoj mravi surađuju, ali se povećava globalna moć pretraživanja.
   - Faktor očekivane heuristike (engl. *expected heuristic factor*) $\beta \in \mathbb{R}^+$ – označava mjeru u kojoj se uvažava vrijednost heurističke informacije pri odabiru puta. Povećanjem faktora β, mravi će više težiti odabirati put na temelju heurističke vrijednosti, no ukoliko je faktor β prevelik, algoritam teži pohlepnom algoritmu jer u prevelikoj mjeri uvažava vrijednost heurističke informacije. Ukoliko je, nasuprot tome, faktor β premalen, algoritam može stagnirati ili konvergirati u lokalni optimum.
   - Koeficijent isparavanja feromona (engl. *pheromone evaporation coefficient*) $\rho \in (0, 1]$ – označava stopu isparavanja feromona. Za veći koeficijent ρ, feromoni će isparavati brže te se na taj način povećava globalna moć pretraživanja algoritma, ali se algoritam usporava. Za malu

vrijednost koeficijenta ρ, algoritam će imati veću težnju konvergencije k lokalnom optimumu, ali će brže doći do rješenja.

3.2.2. Konstrukcija mravljih puteva

Izvođenje algoritma započinje generiranjem određenog broja umjetnih mrava te smještanjem svakog u nasumični vrh grafa. Mravi potom biraju sljedeći neposjećeni vrh grafa na temelju intenziteta feromona na svakom od bridova kao lokalne informacije te udaljenosti između trenutnog i potencijalnog sljedećeg čvora kao vrijednosti heuristike.

Svaki mrav potom prelazi u novi vrh te ponavlja postupak dok ne posjeti sve vrhove grafa i vrati se u početni. Mravi pamte put kojim su prošli, kao i duljinu puta (kvalitetu rješenja) koji su prošli na temelju koje ažuriraju vrijednost intenziteta feromona na putu kojim su prošetali. Na razini algoritma se pamti najbolji do sad nađeni put.

3.2.3. *Daemon* akcije

Nakon što je svaki od mrava konstruirao svoj put, a prije sljedećeg koraka, možda je za izvedbu algoritma potrebno izvesti neke pozadinske akcije nužne za izvedbu algoritma. Te akcije se izvode na razini cijele populacije te ih svaki od mrava ne može izvesti zasebno. Jedan od primjera je rangiranje mrava prema kvaliteti rješenja u nekim od mravljih algoritama. Kao što je već spomenuto, ovaj korak nije nužno izvesti u svakoj varijaciji mravljeg algoritma.

3.2.4. Ažuriranje feromona

Kako bi algoritam mogao funkcionirati prema Deneubourgovu modelu, nakon svake je iteracije potrebno ažurirati intenzitet feromona na bridovima. Intenzitet feromona na bridovima koji pripadaju dobrim rješenjima je potrebno pojačati, dok je intenzitet feromona na bridovima koji pripadaju lošim rješenjima potrebno smanjiti. To se najčešće radi na način da se smanji intenzitet feromona na svim bridovima u ovisnosti o veličini parametra ρ te da se potom poveća u ovisnosti o broju i kvaliteti rješenja koja prolaze tim bridom.

3.3. Taksonomija algoritma kolonije mrava

Kako je mravlji algoritam jedan od najpopularnijih, najistraživanijih i najkorištenijih algoritama inteligencije roja, činjenica da postoje njegove brojne podvarijante koje se na razne načine blago odmiču od generičkog mravljeg algoritma nije začuđujuća. Tijekom godina, razvijene su mnoge nadogradnje na osnovni algoritam, a neke od najbitnijih su [16]:

- Sustav mrava (3.3.1)
- Sustav mrava s ugrađenim elitizmom (3.3.2)
- Sustav mrava baziran na rangu (3.3.3)
- Max-min sustav mrava (3.3.4)
- Sustav mravlje kolonije (3.3.5)

3.3.1. Sustav mrava

Sustav mrava (engl. *Ant system* — u daljnjem tekstu „AS") prvi je mravlji algoritam koji je objavljen u literaturi. Prvu verziju algoritma objavili su Dorigo i suradnici 1991. godine, a nadograđivana je 1992. i 1996. godine. Njegova glavna karakteristika leži u tome što vrijednosti intenziteta feromona ažuriraju svi mravi u jednakoj mjeri.

Prilikom konstrukcije rješenja, mravi odabiru vrhove grafa na temelju formule:

$$p(e_{i,j}, s_v) = \frac{\tau_{i,j}^{\alpha} \cdot \nu_{i,j}^{\beta}}{\sum_{k \notin s_v} \tau_{i,k}^{\alpha} \cdot \nu_{i,k}^{\beta}}, \forall j \notin s_v \tag{3.1}$$

gdje je i indeks trenutnog vrha, j indeks promatranog vrha, $e_{i,j}$ promatrani brid, s_v skup posjećenih vrhova, $\tau_{i,j}$ intenzitet feromona na bridu $b_{i,j}$, a $\nu_{i,j}$ vrijednost heurističke informacije na bridu $b_{i,j}$.
Nakon što su svi mravi pronašli puteve, odmah se prelazi na korak u kojem se ažuriraju vrijednosti intenziteta feromona (u algoritmu AS ne izvode se *daemon* akcije). Na svakom se bridu $e_{i,j}$ ažurira vrijednost intenziteta feromona prema formuli:

$$\tau_{i,j} \leftarrow (1 - \rho) \cdot \tau_{i,j} + \rho \cdot \sum_{k=1}^{m} \Delta\tau_{i,j}^{k}, \tag{3.2}$$

gdje je m broj mrava, a $\Delta\tau_{i,j}$ je funkcija dobrote (kvaliteta rješenja) koja najčešće odgovara:

$$\Delta\tau_{i,j} = \begin{cases} \frac{1}{L_k}, & \text{ako je mrav } k \text{ prošao bridom } e_{i,j} \\ 0, & \text{inače} \end{cases} \tag{3.3}$$

gdje je L_k duljina puta koji je prošao mrav k.

3.3.2. Sustav mrava s ugrađenim elitizmom

Sustav mrava s ugrađenim elitizmom (engl. *Elitist ant system*, u daljnjem tekstu „EAS") nastao je kao prvo proširenje AS, a razvijali su se gotovo paralelno. Jedina razlika EAS u odnosu na AS leži u tome što se u svakoj iteraciji dodatno pojačava intenzitet feromona na najboljem nađenom rješenju, čak i ako se u toj iteraciji tim putem nije prošlo.

To se izvodi na način da se pamti trenutno najbolje rješenje i mrav koji ga je pronašao te da se prilikom globalnog ažuriranja feromona dodatno ažurira intenzitet feromona na bridovima najboljeg nađenog rješenja prema formuli (3.2).

3.3.3. Sustav mrava baziran na rangu

Sustav mrava baziran na rangu (engl. *Rank-based ant system*, u daljnjem tekstu „AS_{Rank}") još je jedna od nadogradnji AS koju su 1999. objavili Bullnheimer i suradnici. U AS_{Rank}-u se mravlji putevi konstruiraju na jednak način kao i kod AS, ali AS_{Rank} uvodi *dameon* akciju — nakon što su svi mravi konstruirali svoja rješenja, sortira ih se po kvaliteti pronađenog rješenja. Potom se bira $(w-1)$ najboljih mrava zajedno s mravom koji je pronašao najbolje rješenje kao najbolje rangiranim.

Svaki odabrani mrav će pojačati intenzitet feromona za $\Delta\tau_{i,j}$ iz izraza (3.3) pomnožen s $(w-r)$ pri čemu je r rang mrava, dok ostali mravi u toj iteraciji ne pojačavaju intenzitet feromona.

3.3.4. Max-min sustav mrava

Max-min sustav mrava (engl. *Max-min ant system*, u daljnjem tekstu „$MMAS$") je nadogradnja na AS koju su 2000. objavili Stützle i Hoos. $MMAS$ se od AS razlikuje po tome što se minimalne ($\tau_{\min}$) i maksimalne ($\tau_{\max}$) vrijednosti intenziteta feromona zadaju eksplicitno. Drugim riječima, ako bi se tijekom promjene intenziteta feromona došlo na vrijednost manju od $\tau_{\min}$ ili veću od $\tau_{\max}$, intenzitet feromona na određenom bridu će se postaviti upravo na tu graničnu vrijednost.

Druga razlika u odnosu na AS leži u tome što će vrijednost intenziteta feromona na putu koji je prošao povećati isključivo mrav koji je u toj iteraciji pronašao najbolje rješenje.

3.3.5. Sustav mravlje kolonije

Sustav mravlje kolonije (engl. *Ant colony system*, u daljnjem tekstu „ACS") nastao je 1997. godine, a predstavili su ga Dorigo i Gambardella. Iako nije najnovija nadogradnja AS algoritma, svakako je najunikatniji. Razlog leži u tome što se u ovom algoritmu, za razliku od ostalih, ažuriranje intenziteta feromona provodi nakon svakog koraka (prelaska iz jednog u drugi čvor).

Druga velika razlika je način na koji mravi odlučuju o sljedećem putu koji će odabrati iz svakog vrha koji posjete. Uvode se dva nova parametra na razini algoritma:

- $q_0 \in [0, 1]$, koji će se koristiti za uspoređivanje
- $\phi \in (0, 1]$, koji označava lokalnu stopu isparavanja feromona

Prije svakog koraka, svaki mrav nasumično generira vrijednost varijable q, koja se potom uspoređuje s q_0. Na temelju rezultata usporedbe određuje se način odabira sljedećeg vrha koji će mrav posjetiti — ukoliko vrijedi $q \leq q_0$, tada se kao sljedeći vrh k bira onaj koji maksimizira produkt $\tau_{i,k}\nu_{i,k}^{\beta}$. Ukoliko pak vrijedi suprotno ($q > q_0$), novi vrh se bira na isti način kao i u AS — jednadžbom (3.1).

Nakon što mrav stigne do novog vrha j, ažurira vrijednost intenziteta feromona na bridu koji je prošao jednadžbom:

$$\tau_{i,j} \leftarrow (1 - \phi) \cdot \tau_{i,j} + \phi \cdot \tau_0 \tag{3.4}$$

pri čemu je τ_0 inicijalna vrijednost intenziteta feromona. Vidljivo je da duljina prijeđenog puta ni na koji način ne utječe na ažuriranje intenziteta feromona koje se provodi nakon svakog koraka.

Nakon što su svi mravi konstruirali svoje rješenje, provodi se i takozvano *offline* ažuriranje intenziteta feromona na razini cijelog algoritma gdje se koristi metoda slična onoj u AS. Razlika je u tome što se u ACS intenzitet feromona povećava samo na putu koji odgovara najboljem rješenju. Ovisno o načinu implementacije algoritma, kao najbolje rješenje može se promatrati ukupno najbolje rješenje ili najbolje rješenje u trenutnoj iteraciji.

3.4. Paralelizacija

Mravlji algoritam je algoritam inteligencije roja koji se bazira na kolektivnoj inteligenciji kolonije. Iz toga se može zaključiti da bi se učinkovitost algoritma mogla pospješiti paralelizacijom.

Budući da svaki mrav ne izravno ovisno o drugim mravima obilazi graf, njihovo se obilaženje ne mora izvoditi sekvencijalno — naime, bolje bi bilo kada bi svi mravi istovremeno obilazili graf.

3.4.1. Višedretveni rad

Instrukcijska dretva je slijed instrukcija u izvođenju („povezanih vremenom izvođenja") [7]. Kako bi se dretva mogla izvoditi, mora imati svoj slijed instrukcija, podatke i stog, koji su sačuvani u njenom spremniku.

Na jednojezgrenim procesorima, stvaran paralelni rad nije ostvaren. Međutim, dretve se u jednoprocesorskom sustavu učestalo izmjenjuju, što korisniku stvara dojam paralelnog rada. Prilikom izmjene, dretva koja se prestaje izvoditi trenutni sadržaj registara mora spremiti u svoj opisnik kako bi kasnije mogla nastaviti s radom [1]. Nova dretva koja ju zamjenjuje obnavlja kontekst spremljen u svom opisniku te nastavlja s radom.

Međutim, u višeprocesorskim sustavima, na svakoj se procesorskoj jezgri istovremeno može izvoditi zasebna dretva, čime je postignuto i fizičko paralelno izvođenje više dretvi. Na taj se način, ukoliko se svaki podzadatak izvodi na zasebnoj dretvi, fizički može paralelizirati izvođenje više međusobno nezavisnih podzadataka.

3.4.2. Sinkronizacija dretvi

Iako višedretveni rad u višeprocesorskim sustavima donosi brojne prednosti, on također može dovesti do mnogih neželjenih situacija i komplikacija. Primjerice, ukoliko dvije dretve istovremeno pristupaju varijabli u kojoj se nalazi brojač koji trebaju uvećati za 1, može se dogoditi da obje dretve približno istovremeno pročitaju vrijednost varijable, svaka od njih uveća pročitanu vrijednost za 1 te obje dretve potom spreme izračunatu vrijednost. Dobiveni rezultat bit će za 1 veći od početnog, iako su dvije dretve posjetile brojač te bi njegova vrijednost trebala biti za 2 veća od početne.

Takve kritične odsječke u programu potrebno je osigurati na način da im istovremeno može pristupiti samo jedna dretva, dok ostale „čekaju u redu". Taj se postupak naziva sinkronizacija dretvi. Iako postoje mnogi algoritmi međusobnog isključivanja poput Dekkerovog, Petersonovog i Lamportovog algoritma, oni nisu najbolje rješenje zbog jedne velike mane — radnog čekanja.

Semafori

Za razliku od algoritama međusobnog isključivanja, semafori za sinkronizaciju dretvi koriste jezgrine funkcije. Ukoliko neka dretva treba čekati na kritični odsječak, semafor ju zaustavlja u svom redu blokiranih dretvi. Postoje *binarni* i *opći* semafor.

Binarni semafori imaju jednostavnu strukturu podataka potrebnu za rad:

- `.v` – označava vrijednost semafora, koja može biti `0` kad je semafor neprolazan, odnosno `1` kad je semafor prolazan
- `.red` – označava red blokiranih dretvi nad semaforom

Jezgrine funkcije koje se koriste za rad s binarnim semaforom su:

- `ISPITAJ_BSEM(i)` – ukoliko je vrijednost semafora jednaka `1`, dretva ju postavlja na `0` i ulazi u kritični odsječak, a inače se blokira u redu semafora te ju u ulozi aktivne dretve zamjenjuje sljedeća dretva iz reda pripravnih dretvi.
- `POSTAVI_BSEM(i)` – ukoliko postoje dretve u redu semafora, izvlači se ona prva te se zajedno s aktivnom stavlja u prioritetni red pripravnih dretvi. Na taj se način osigurava izvođenje prioritetnije dretve i oslobađanje sljedeće dretve iz reda semafora. Nasuprot tome, ukoliko u redu semafora nema blokiranih dretvi, vrijednost `.v` se postavlja na `1`.

Primjer pseudokoda u kojem se binarnim semaforom osigurava kritični odsječak glasio bi:

```
dretva(){
    nekritični_dio_1;
    ISPITAJ_BSEM(i);
    kritični_odsječak;
    POSTAVI_BSEM(i);
    nekritični_dio_2;
}
```

Za razliku od binarnog semafora, opći semafor ima jedno neprolazno i više prolaznih stanja (svaka vrijednost `.v>0`). Jezgrine funkcije koje se koriste za rad s općim semaforom su:

- `ISPITAJ_OSEM(i)` – vrlo slična `ISPITAJ_BSEM(i)`, uz razliku da se vrijednost semaforske varijable `.v` smanjuje za `1` za bilo koju njenu prolaznu vrijednost

– `POSTAVI_OSEM(i)` – vrlo slična `POSTAVI_OSEM(i)`, samo što se u slučaju praznog reda blokiranih dretvi vrijednost `.v` povećava za 1 umjesto postavljanja na 1

Monitori

Korištenje semafora u određenim tipovima sustava može dovesti do mnogih problema, poput potpunog zastoja (engl. *deadlock*). Stoga su 1970–ih britanski znanstvenik Charles Anthony Richard Hoare i danski znanstvenik Per Brinch Hansen predložili novi način sinkronizacije — monitore.

Sinkronizacija pomoću monitora funkcionira na način da se sve kritične operacije izvode u kontroliranom okruženju pomoću monitorskih funkcija. Te su funkcije vrlo slične jezgrinim funkcijama, no nisu jezgrine, već korisničke. Osim funkcija „ulaska u monitor" te „izlaska iz monitora", koje predstavljaju zaštitu kritičnih odsječaka, monitor ima i dodatne funkcionalnosti. Jedna od takvih je i rad s takozvanim redovima uvjeta.

Red uvjeta (engl. *condition variable*) je svojevrsni red dretvi koji je vezan uz konkretan monitor i sadržava blokirane dretve. Dretve koje čekaju u redu uvjeta ne zauzimaju monitor i iz tog reda neće biti oslobođene sve dok nisu primile signal o ispunjenju uvjeta čije ispunjenje čekaju (Slika 3.2).

Monitorske funkcije su sljedeće:

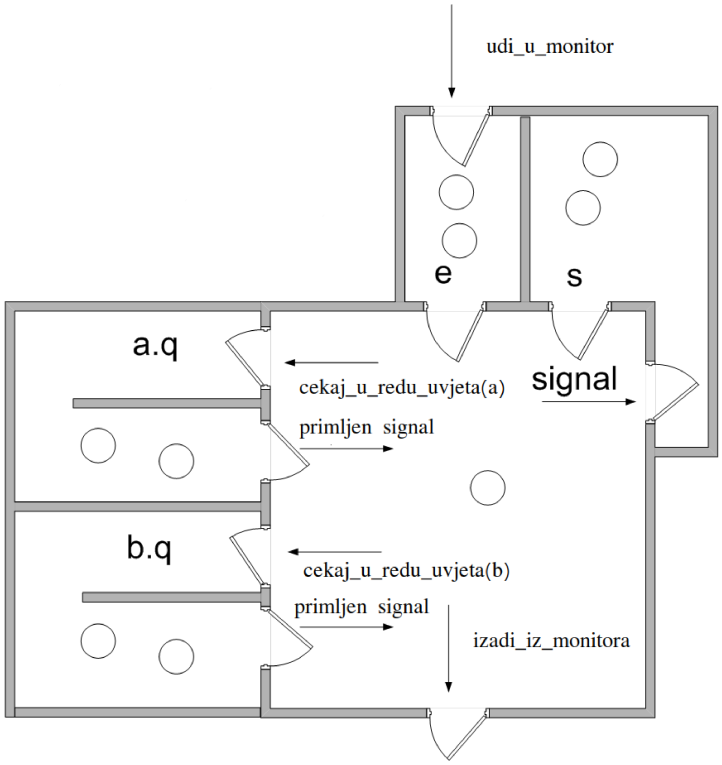
– `uđi_u_monitor(m)` (engl. *`mutex_lock(m)`*) – ekvivalent funkciji `ISPITAJ_BSEM(i)` za rad s binarnim semaforima

– `izađi_iz_monitora(m)` (engl. *`mutex_unlock(m)`*) – ekvivalent funkciji `POSTAVI_BSEM(i)` za rad s binarnim semaforima

– `čekaj_u_redu_uvjeta(r, m)` (engl. *`cond_wait(r, m)`*) – dretva se uvrštava u red uvjeta `r` monitora te po ispunjenju uvjeta ulazi u monitor (ili čeka ulazak)

– `oslobodi_iz_reda_uvjeta(r, m)` (engl. *`cond_signal(r, m)`*) – šalje se signal za potencijalno ispunjenje uvjeta prvoj dretvi u redu uvjeta `r` koja potom provjerava uvjet i izlazi ukoliko je ispunjen

– `oslobodi_sve_iz_reda_uvjeta(r, m)` (engl. *`cond_broadcast(r, m)`*) – šalje se signal za potencijalno ispunjenje uvjeta svim dretvama u redu uvjeta `r` te one, ukoliko je uvjet ispunjen, izlaze iz reda uvjeta



Slika 3.2: Vizualizacija monitora [10]

3.4.3. Paralelizacija mravljeg algoritma

Kada bi se svaki mrav u mravljem algoritmu oblikovao na način da se njegovo ponašanje izvodi na zasebnoj dretvi, takav bi se sustav mogao jednostavno sinkronizirati pomoću monitora. Sustav bi se sastojao od `n` mravljih dretvi te jedne glavne dretve koja ih je lansirala i koja čeka njihovo ponovno spajanje (engl. *join*). Kako se u mravljem algoritmu ažuriranje feromona treba izvesti tek nakon što je svaki mrav pronašao put u toj iteraciji te mravi ne smiju krenuti u novu šetnju prije no što je ažuriranje feromona završilo, mrave i glavnu dretvu potrebno je sinkronizirati pomoću dva reda uvjeta:

- `cv1` – red uvjeta koji blokira glavnu dretvu toliko dugo dok svaki od mrava nije pronašao rješenje
- `cv2` – red uvjeta koji blokira sve mravlje dretve toliko dugo dok nije završilo ažuriranje feromona

Pseudokod navedenog sustava glasio bi kako je prikazano u algoritmu 1 i algoritmu 2.

Prednost paralelne izvedbe leži u tome što se više mravljih dretvi može istovremeno izvoditi, što vodi velikom ubrzanju izvođenja algoritma. Međutim, budući da

Algorithm 1 Glavna dretva

```
i := 0
repeat
  uđi_u_monitor(m)
  while (broj != n) do
    čekaj_u_redu_uvjeta(cv1, m)
  end while
  ažuriraj_feromone()
  broj := 0
  oslobodi_sve_iz_reda_uvjeta(cv2, m)
  izađi_iz_monitora()
  i := i + 1
until (i < n)
```

Algorithm 2 Mravlja dretva

```
i := 0
repeat
  pronađi_put()
  uđi_u_monitor(m)
  broj := broj + 1
  oslobodi_iz_reda_uvjeta(cv1, m)
  while (broj != 0) do
    čekaj_u_redu_uvjeta(cv2, m)
  end while
  i := i + 1
until (i < n)
```

će u paralelnoj izvedbi više dretvi raditi istovremeno, potrebno je zaštititi kritične odsječke kako izvođenje algoritma ne bi dovelo do pogrešnih vrijednosti. Također je potrebno ispravno sinkronizirati dretve redovima uvjeta, jer polazak tog dijela po zlu može dovesti do potpunog zastoja, što je najgori mogući scenarij do kojeg se može doći radeći u višedretvenom okruženju.

Ovi uvjeti se možda čine složenima, što i jest mana višedretvenih okruženja, no ukoliko osoba koja provodi sinkronizaciju dretvi zna što radi te istu provede dobro, neće biti nikakvih komplikacija.

4. Problem trgovačkog putnika

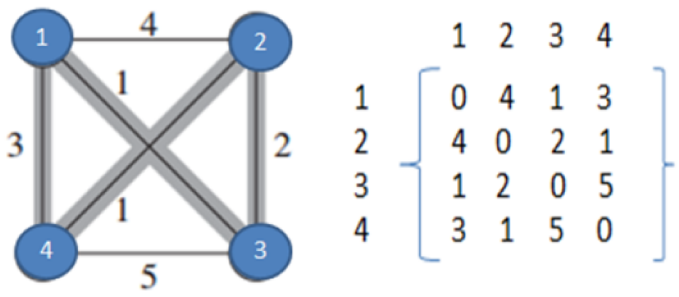
4.1. Opis problema trgovačkog putnika

Problem trgovačkog putnika jedan je od najpopularnijih i najpoznatijih optimizacijskih problema današnjice. Iako su njegovi točni korijeni i vrijeme nastanka nepoznati, problem su službeno matematički formulirali i definirali irski matematičar W. R. Hamilton i britanski matematičar T. Kirkman u 19. stoljeću.

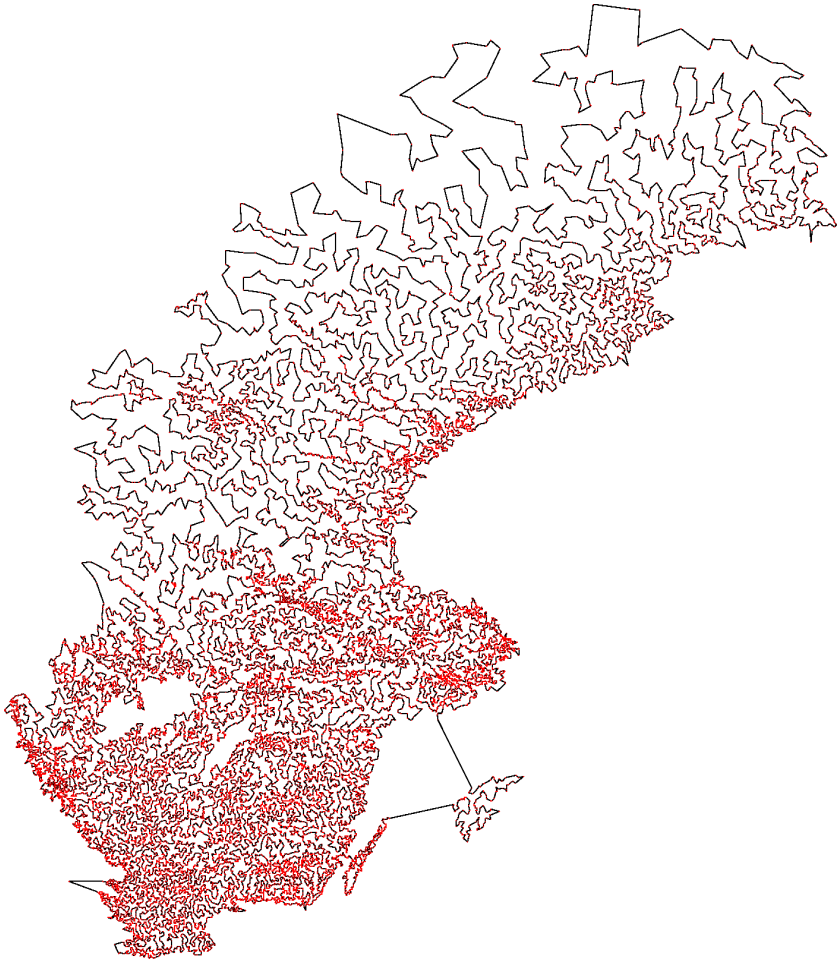
Problem u originalu glasi: *Trgovači putnik treba posjetiti n gradova. Ako su poznate udaljenosti između svakog para gradova, koji je najkraći put kojim trgovački putnik može obići svih n gradova i vratiti se u početni, pod uvjetom da svaki grad posjeti točno jednom?*

Prevedeno na jezik teorije grafova, ekvivalentna formula problema trgovačkog putnika glasila bi [12]: *U potpunom težinskom grafu nađi hamiltonovski ciklus minimalne duljine.*

Ovaj problem spada u kategoriju NP-teških problema, ali je zbog svoje jednostavne formulacije vrlo pogodan kao optimizacijski problem. Kako je u problemu trgovačkog putnika veličine n moguće naći ukupno $(n-1)!/2$ hamiltonovska grafa, očigledno je da bi naivni *brute-force* algoritam imao faktorijelnu vremensku složenost. Do današnjeg dana još uvijek nije pronađen deterministički algoritam koji bi rješavao problem u polinomskoj vremenskoj složenosti. Međutim, mnoge metaheuristike mogu s jed-



Slika 4.1: Primjer problema trgovačkog putnika i njegov matrični zapis [11]



Slika 4.2: Optimalni put kroz 24978 naseljenih mjesta u Švedskoj [5]

nostavnošću naći rješenje koje se nalazi među 1% najboljih. Među njima je, naravno, i mravlji algoritam.

Iako problem trgovačkog putnika ima brojne varijante, razmatrat će se samo klasični problemi trgovačkog putnika, dakle oni koji zadovoljavaju sljedeća svojstva:

- **Svi vrhovi (gradovi) povezani su u potpuni težinski graf** — ovime se uvjetuje da su svi gradovi međusobno direktno povezani te da je poznata cijena puta između svakog para gradova.
- **Graf mora biti simetričan** — drugim riječima, za proizvoljni par vrhova iz grafa koji opisuje problem trgovačkog putnika moguće je prijeći brid koji povezuje ta dva vrha u oba smjera za istu cijenu.
- **Svaka trojka vrhova mora zadovoljavati nejednadžbu trokuta**, dakle vrijedi:

$$d_{i,j} \leq d_{j,k} + d_{i,k}, \forall i, j, k \in V(G) \tag{4.1}$$

Ukoliko se gradovi zadaju pomoću koordinata dvodimenzionalnog Kartezijevog koordinatnog sustava, zadovoljit će se svi gore navedeni uvjeti. Naime, budući da su nam poznate koordinate svih gradova, možemo odrediti udaljenost između svakog para gradova. Također, nejednadžba trokuta u Kartezijevom koordinatom sustavu uvijek vrijedi za bilo koje tri točke.

4.2. Mravlji algoritam prilagođen rješavanju problema trgovačkog putnika

Budući da struktura problema trgovačkog putnika (pronalaženje najkraćeg hamiltonovskog ciklusa na potpunom težinskom grafu) savršeno odgovara strukturi mravljeg algoritma, vrlo je jednostavno konstruirati mravlji algoritam prilagođen rješavanju problema trgovačkog putnika.

Svaki od gradova (vrhova) u problemu trgovačkog putnika potrebno je pretočiti u vrh grafa mravljeg algoritma. Udaljenost između svaka dva grada odgovara vrijednosti heurističke funkcije između njih. Mravi šeću tim vrhovima i luče feromone na putevima između njih.

Na početku se svaki od mrava smješta u nasumično odabrani vrh te započinje svoju šetnju. Nakon što je mrav završio šetnju i vratio se u početni vrh, to odgovara jednom hamiltonovskom ciklusu na potpunom težinskom grafu. Što je pronađeno rješenje za svakog mrava na kraju svake iteracije bolje, to je kraća duljina hamiltonovskog ciklusa u problemu trgovačkog putnika. U konačnici će najbolje nađeno rješenje odgovarati jednom od rješenja problema trgovačkog putnika koje nije nužno optimalno (niti će u većini slučajeva biti), ali je zadovoljavajuće.

5. Programska implementacija

Radni okvir za rješavanje problema trgovačkog putnika napisan je u programskom jeziku C++17. Za rad s dretvama i monitorima se koriste C++ biblioteke `<thread>`, `<mutex>` i `<condition_variable>`, dok se za pseudo-nasumično generiranje brojeva koristi generator iz biblioteke `<random>`.

5.1. Učitavanje podataka

Primjerak razreda `TravellingSalesmanProblem` instancira se naredbom prikazanom u isječku koda 5.1.

```
1 TravellingSalesmanProblem tsp("../Data/dj38.tsp");
```

Isječak koda 5.1: Instanciranje primjerka problema trgovačkog putnika

gdje se unutar zagrada navodi put do datoteke[1] u kojoj je zapisan konkretan problem trgovačkog putnika. Unutar konstruktora se poziva parser ulazne datoteke koji učitava koordinate gradova.

Struktura ulazne datoteke je:

```
NAME : wi29
COMMENT : 29 locations in Western Sahara
COMMENT : Derived from National Imagery and Mapping Agency data
TYPE : TSP
DIMENSION : 29
EDGE_WEIGHT_TYPE : EUC_2D
NODE_COORD_SECTION
1 20833.3333 17100.0000
2 20900.0000 17066.6667
```

[1] Sve ulazne datoteke su preuzete s `http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsp/index.html` i `http://www.math.uwaterloo.ca/tsp/world/countries.html`

```
3 21300.0000 13016.6667
4 21600.0000 14150.0000
...
EOF
```

U prvih nekoliko linija naveden je opis konkretnog problema te se te linije ignoriraju. Nakon opisnih linija slijede linije koje se sastoje od tri broja: prvi broj predstavlja indeks grada, dok druge dvije predstavljaju x i y koordinatu grada, redom.

Nakon što su učitani podaci o gradovima, na temelju koordinata se računa dijagonalna udaljenost između svakog para gradova. Izračunate udaljenosti potom se spremaju u kvadratnu matricu. Također, postavlja se i inicijalna vrijednost intenziteta feromona postavljanjem intenziteta feromona na svakom bridu na recipročnu vrijednost najvećoj pronađenoj udaljenosti između dva grada.

5.2. Instanciranje konkretnog primjerka mravljeg algoritma

Nakon što je instanciran primjerak razreda `TravellingSalesmanProblem`, konkretan primjerak mravljeg algoritma možemo instancirati naredbom (Isječak koda 5.2).

```
1 AntColonySystem antColonySystem(tsp);
```

Isječak koda 5.2: Instanciranje primjerka mravljeg algoritma

gdje je tsp primjerak razreda `TravellingSalesmanProblem`. Konkretne varijante mravljeg algoritma koje se mogu stvoriti su:

- `AntSystem`
- `ElitistAntSystem`
- `RankBasedAntSystem`
- `MaxMinAntSystem`
- `AntColonySystem`

Stvaranjem instanci bilo kojeg od tih razreda, njihovi se parametri, uključujući i one specifične za konkretni algoritam, postavljaju na pretpostavljene vrijednosti, ali se mogu i ručno podešavati (Isječak koda 5.3).

```
1 antColonySystem.setNumberOfAnts(30);
2 antColonySystem.setNumberOfIterations(100);
3 antColonySystem.setAlpha(1.0);
```

```
4 antColonySystem.setBeta(2.0);
5 antColonySystem.setRho(0.1);
6 antColonySystem.setQ0(0.9);
7 antColonySystem.setPhi(0.1);
```

Isječak koda 5.3: Način podešavanja parametara za primjerak razreda `AntColonySystem`

5.3. Funkcionalnost mravljeg algoritma

Rad algoritma započinje instanciranjem potrebnog broja umjetnih mrava i lansiranjem njihovih dretvi. Umjetnim mravima se u konstruktoru šalju svi potrebni podaci:

- indeks mrava
- pseudo-slučajno generiran indeks početnog grada
- pokazivači na monitor, redove uvjeta i varijable koji se koriste za sinkronizaciju
- dimenzije problema trgovačkog putnika
- pokazivači na kvadratne matrice udaljenosti i intenziteta feromona među gradovima
- parametri algoritma($\alpha, \beta, \rho, \ldots$)

Stvaranjem primjerka umjetnog mrava i vezanjem funkcije na njegovu dretvu, dretva se automatski pokreće i započinje s izvođenjem te funkcije.

Nakon što su stvoreni umjetni mravi i lansirane njihove dretve, na glavnoj se dretvi nastavlja izvađati kod algoritma. Glavna dretva ulazi u monitor `m` i ulazi u red uvjeta `cvMain` te čeka dok svi mravi nisu dovršili svoju šetnju. Završetak svoje šetnje mravi glavnoj dretvi dojavljuju putem varijable `sync`. Glavna dretva će u redu uvjeta `cvMain` čekati toliko dugo dok iznos sinkronizacijske varijable `sync` ne dosegne ukupan broj mrava (Isječak koda 5.4).

```
1 {
2     unique_lock<mutex> lock(m);
3     cvMain.wait(lock, [this]{return *this->sync == this->numberOfAnts;});
4     lock.unlock();
5 }
```

Isječak koda 5.4: Ulazak glavne dretve u red uvjeta `cvMain`

Nakon što je glavna dretva izašla iz reda uvjeta, ona ponovo ulazi u monitor gdje najprije globalno smanjuje vrijednost intenziteta feromona u ovisnosti o parametru ρ, provjerava kvalitetu rješenja svakog mrava, ažurira vrijednost intenziteta feromona na

njegovom putu te po potrebi ažurira trenutno najbolje nađeno rješenje. U konačnici se postavljaju potrebne vrijednosti sinkronizacijskih varijabli, oslobađaju se sve dretve iz reda uvjeta `cvAnts` te se kreće u sljedeću iteraciju. Ovisno o varijanti mravljeg algoritma, mravi se mogu sortirati prema kvaliteti rješenja ili se može ažurirati intenzitet feromona samo na putu mrava s najboljim rješenjem. Navedeni isječak koda za algoritam AS prikazan je u isječku koda 5.5:

```
1  {
2      lock_guard <mutex> lg(m);
3      reducePheromone();
4      for(int i = 0; i < numberOfAnts; i++){
5          vector<int> path = this->ants[i]->getPath();
6          double length = this->ants[i]->getLength();
7          updatePheromone(path, length);
8          if(length < this->best || (iteration == 0 && i == 0)){
9              this->best = length;
10             this->bestPath = vector(path);
11             this->bestAnt = ants[i]->getIndex();
12         }
13     }
14
15     *sync = 0;
16     *syncWait = 0;
17     cvAnt.notify_all();
18 }
```

Isječak koda 5.5: Ažuriranje feromona i rangiranje mrava u glavnoj dretvi algoritma AS

U navedenom isječku koda također je moguće ispisivati podatke o najboljem ukupnom rješenju iteracije.

5.4. Funkcionalnost umjetnih mrava

Prilikom konstrukcije primjerka umjetnog mrava, lansira se dretva koja izvršava odgovarajuću metodu koja oblikuje njegovo ponašanje.

Prema načinu na koji se mravi ponašaju u izvođenju algoritma, razlikujemo dvije vrste umjetnih mrava:

- Mravi kod kojih se vrijednost intenziteta feromona ažurira tek nakon dovršene šetnje („ciklusni mravi”)— algoritmi AS, EAS, AS_{Rank} te $MMAS$
- Mravi kod kojih se vrijednost intenziteta feromona ažurira nakon svakog prijelaza između gradova („korački mravi”) — algoritam ACS

Zbog sporosti oslobađanja dretvi iz reda uvjeta `cvAnts` te kao prevencija blokiranja dretvi u istom, u početku svake iteracije se mravlje dretve sinkroniziraju pomoću reda uvjeta `cvWait` i sinkronizacijske varijable `syncWait`, a u izvođenje algoritma kreću tek nakon što su sve spremne (Isječak koda 5.9).

```
1 {
2     unique_lock<mutex> lock(*m);
3     *syncWait = *syncWait + 1;
4     if(*syncWait != noAnts)
5         cvWait->wait(lock, [syncWait, noAnts]{return *syncWait == noAnts;});
6     else
7     cvWait->notify_all();
8 }
```

Isječak koda 5.6: Sinkronizacija mravljih dretvi na početku iteracije

Svaka od mravljih dretvi uvećava vrijednost sinkronizacijske varijable `syncWait` za 1 te ulazi u red uvjeta `cvWait`, osim posljednje koja oslobađa sve ostale iz reda uvjeta.

Nakon što se mravlja dretva oslobodila iz reda uvjeta `cvWait`, ona kreće u izvođenje iteracije. Prvi korak u tome je resetiranje struktura podataka u kojima se pohranjuju informacije o putu i prijeđenoj udaljenosti.

5.4.1. Ciklusni mravi

Nakon resetiranja, ciklusni mravi ulaze u petlju koja se izvodi sve dok nisu obišli sve gradove. Na početku svake iteracije, mrav za sve neposjećene vrhove određuje težinsku vrijednost na temelju koje se, dijeljenjem sa sumom svih težinskih vrijednosti, dobiva vjerojatnost da mrav odabere taj vrh kao sljedeći. Nakon toga se u ovisnosti o vjerojatnostima bira novi vrh koji se potom dodaje u skup posjećenih vrhova te se put do njega pribrojava ukupnom putu (Isječak koda 5.7).

```
1  double totalWeight = 0;
2  vector<double> weights;
3  int index = 0;
4
5  for(int node : this->notVisited){
6      while(index < node){
7          weights.push_back(0);
8          index++;
9      }
10     double weight = pow (*pheromones[currentNode][node], alpha)
11     * pow((100 / *distances[currentNode][node]), beta);
```

```
12      weights.push_back(weight);
13      totalWeight += weight;
14      index++;
15  }
16
17  double randomNumber = dist(engine);
18  double currentProbability = 0;
19  int nextNode = 0;
20
21  for(double weight: weights){
22      double probability = weight / totalWeight;
23      currentProbability += probability;
24      if(randomNumber <= currentProbability)
25      break;
26      nextNode++;
27  }
28  path.push_back(nextNode);
29  this->length += *distances[currentNode][nextNode];
30  this->fillSet(dim, path);
31
32  currentNode = nextNode;
33
34  this->path.push_back(this->startingNode);
35  this->length += *distances[currentNode][this->startingNode];
```

Isječak koda 5.7: Način biranja sljedećeg vrha kod ciklusnih mrava

Nakon što je mrav obišao cijeli graf, njegova dretva ulazi u monitor i inkrementira vrijednost sinkronizacijske varijable `sync`, a potom obavještava dretvu koja se nalazi u redu uvjeta `cvMain` o mogućem ispunjenju uvjeta te nakon toga i sama ulazi u red uvjeta `cvAnts`, kao što je prikazano u isječku koda 5.8.

```
1  {
2      unique_lock<mutex> lock(*m);
3      *sync = *sync + 1;
4      cvMain->notify_one();
5      cvAnts->wait(lock, [sync]{return *sync == 0;});
6  }
```

Isječak koda 5.8: Sinkronizacija mravlje dretve s glavnom dretvom

Po potrebi se može ispisati i statistika o trenutnoj iteraciji mrava pozivom metode `printAnt()`.

5.4.2. Koračni mravi

Veliki dio funkcije koračnog mrava odgovara funkciji ciklusnog mrava, uz jedan dodatak — ažuriranje feromona na bridu koji je mrav prošao nakon svakog prijeđenog grada. Budući da je više mrava moglo prošetati istim bridom, potrebno je zaštititi odsječak koda u kojem mravi ažuriraju vrijednost intenziteta feromona na bridovima kako više mrava istovremeno ne bi pokušalo ažurirati vrijednost feromona na istom bridu. To se može jednostavno riješiti monitorom — mravlja dretva zaključa monitor pomoću primjerka razreda `lock_guard<mutex>` te potom ažurira vrijednost intenziteta feromona na potrebnom vrhu, kao što je prikazano u isječku koda 5.9. Primjerci razreda `lock_guard<mutex>` prilikom izlaska iz bloka u kojem su instancirani otključavaju monitor, stoga to nije potrebno raditi ručno.

```
1 {
2     lock_guard<mutex> lg(*m);
3     double newValue = (1 - this->phi) *
4     *pheromones[currentNode][nextNode] + this->phi * this->tau0;
5     *pheromones[currentNode][nextNode] = newValue;
6     *pheromones[nextNode][currentNode] = newValue;
7 }
```

Isječak koda 5.9: Sinkronizacija koračnih mrava pri ažuriranju feromona

5.5. Pokretanje algoritma

Sveukupan izgled funkcije `main()` za pokretanje mravljeg algoritma nad željenim problemom trgovačkog putnika prikazan je na isječku koda 5.10

```
1  int main() {
2      TravellingSalesmanProblem tsp("../Data/dj38.tsp");
3      AntColonySystem antColonySystem(tsp);
4
5      antColonySystem.setNumberOfAnts(30);
6      antColonySystem.setNumberOfIterations(100);
7      antColonySystem.setAlpha(1.0);
8      antColonySystem.setBeta(2.0);
9      antColonySystem.setRho(0.1);
10     antColonySystem.setQ0(0.9);
11     antColonySystem.setPhi(0.1);
12
13     antColonySystem.run();
14     antColonySystem.printStats();
```

```
15
16      return 0;
17  }
```

Isječak koda 5.10: Funkcija main()

Algoritam se pokreće pozivom metode `run()`, dok se informacije o najboljem pronađenom rješenju i duljini izvođenja algoritma mogu ispisati pozivom metode `printStats()`. Primjer ispisa metode `printStats()` prikazan je u nastavku:

```
    Results:
============================================================
Number of cities: 38
Best value: 6665.57
Best path: [8 7 6 5 4 2 3 1 0 9 13 20 28 29 31 34 36 37 32
  33 35 30 26 27 23 21 24 25 22 19 14 12 17 18 16 15 11 10]
Execution time: 1.56058 seconds.
============================================================
```

6. Rezultati ispitivanja djelotvornosti programskog ostvarenja

U nastavku slijedi statistika izvođenja programske implementacije nad 4 problema trgovačkog putnika:

- `ulysses16.tsp` — Problem trgovačkog putnika sa 16 gradova, „Uliksova odiseja”
- `dj38.tsp` — Problem trgovačkog putnika s 38 gradova u državi Džibuti
- `kroA100.tsp` — Varijanta Krolak/Felts/Nelson problema trgovačkog putnika sa 100 gradova
- `gr202.tsp` — Varijanta Groetschel problema trgovačkog putnika s 202 grada

Za svaki od navedenih problema, pokrenuto je svih 5 algoritama s fiksnim vrijednostima parametara $n = 100$, $\alpha = 1.0$, $\beta = 2.0$, $\rho = 0.1$. Vrijednosti parametara specifičnih za konkretne algoritme fiksirani su na:

- AS_{Rank}:

$$w = \begin{cases} \text{m}, & \text{ako je } m < 6 \\ 6, & \text{inače} \end{cases} \tag{6.1}$$

- $MMAS$:

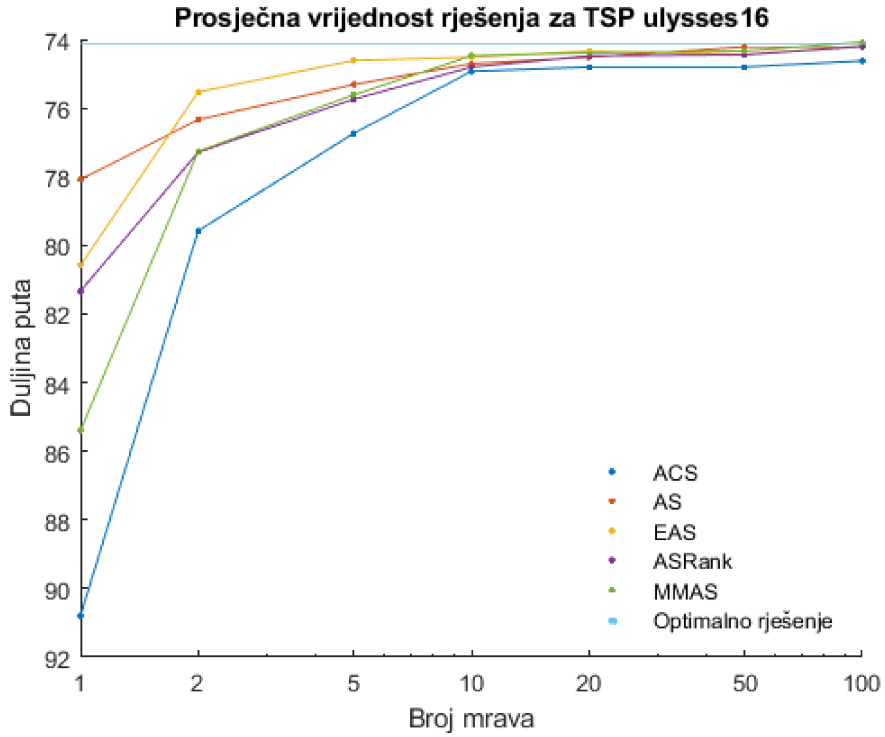
$$\tau_{\min} = 10^{-10}, \tau_{\max} = 10^{-4} \tag{6.2}$$

- ACS:

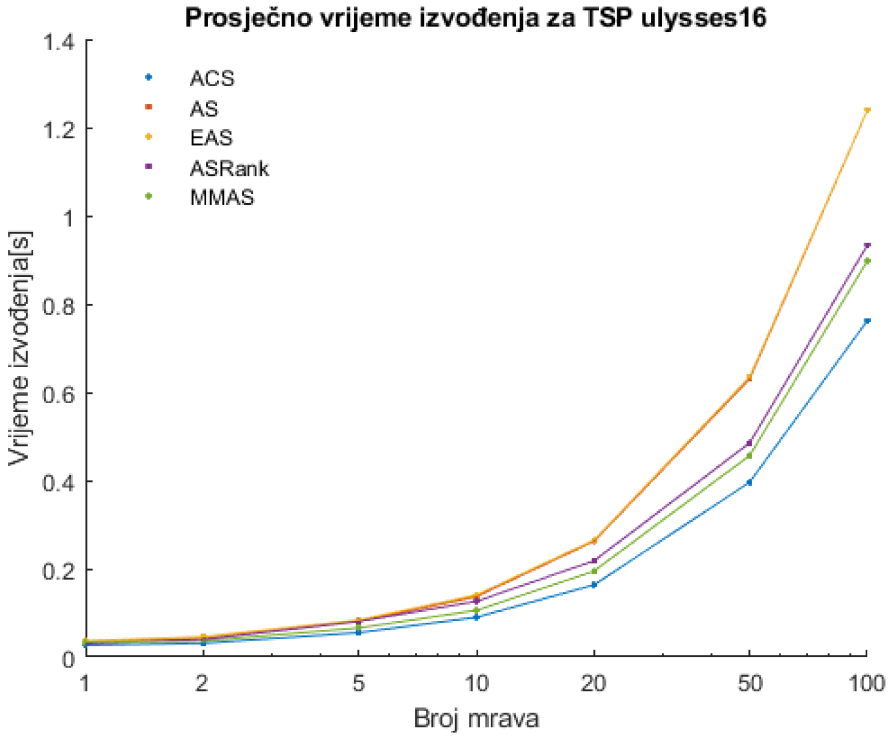
$$q_0 = 0.9, \phi = 0.1 \tag{6.3}$$

Za svaku od (na grafovima) navedenih vrijednosti parametra n, algoritam je pokrenut 5 puta te su u graf unesene prosječne dobivene udaljenosti te vremena izvođenja algoritama.

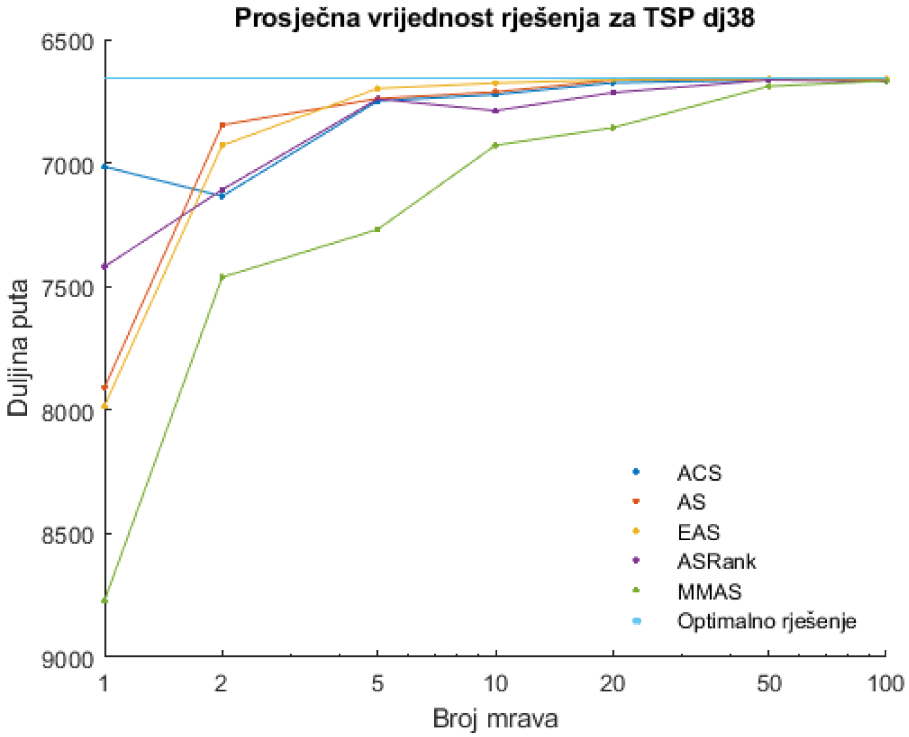
Svi testovi provedeni su na istom procesoru — $Intel^{®}Core^{™}i3-6006U$ s dvije fizičke jezgre takta 2.00 GHz.



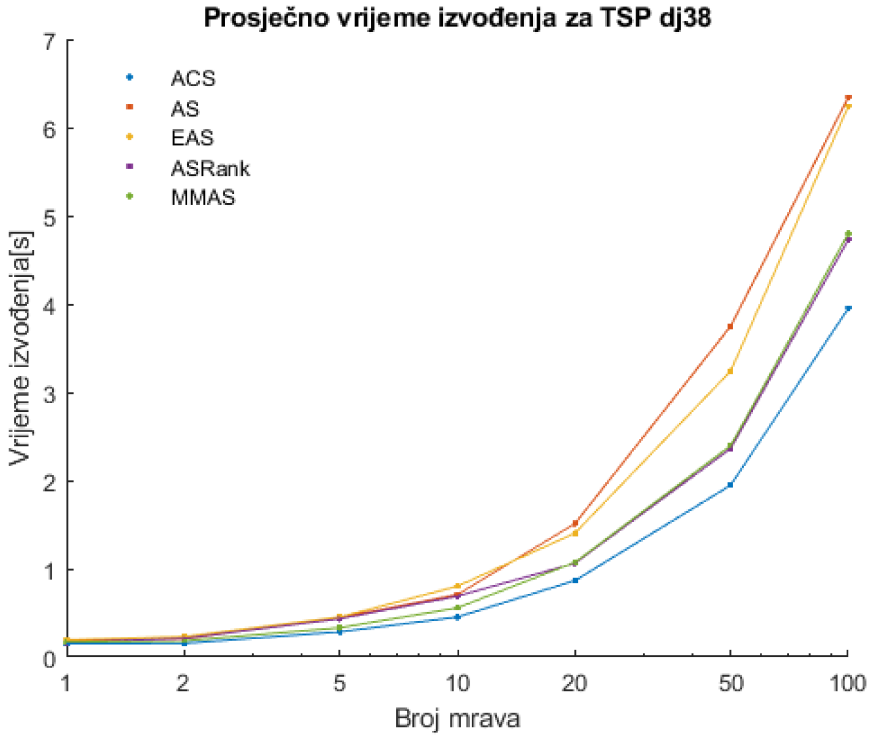
Graf 6.1: Prosječne vrijednosti udaljenosti za TSP ulysses16 u ovisnosti o broju mrava



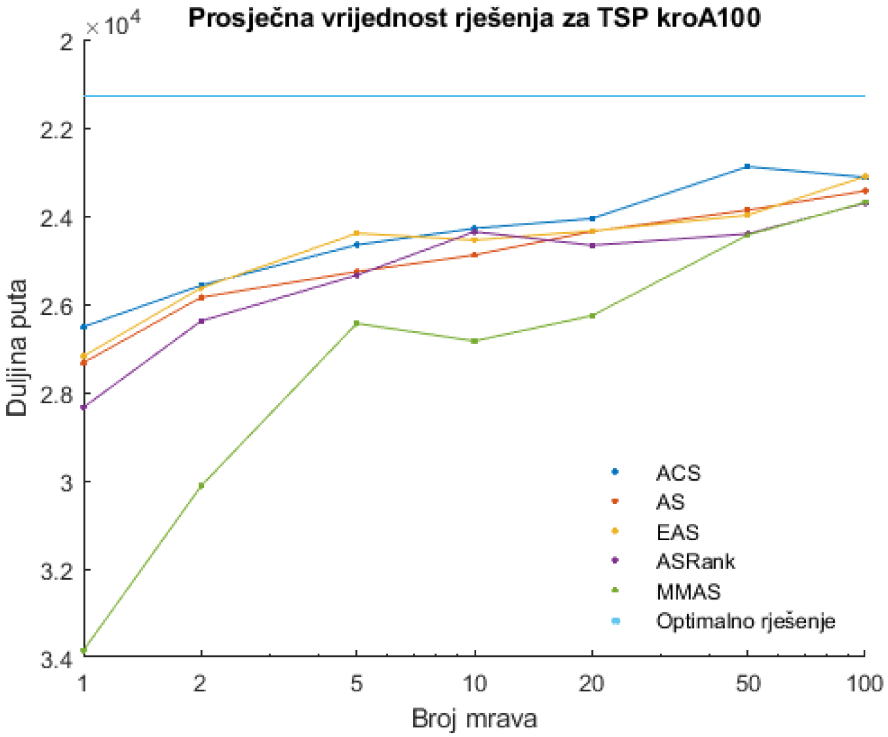
Graf 6.2: Prosječno trajanje izvođenja algoritama za TSP ulysses16 u ovisnosti o broju mrava



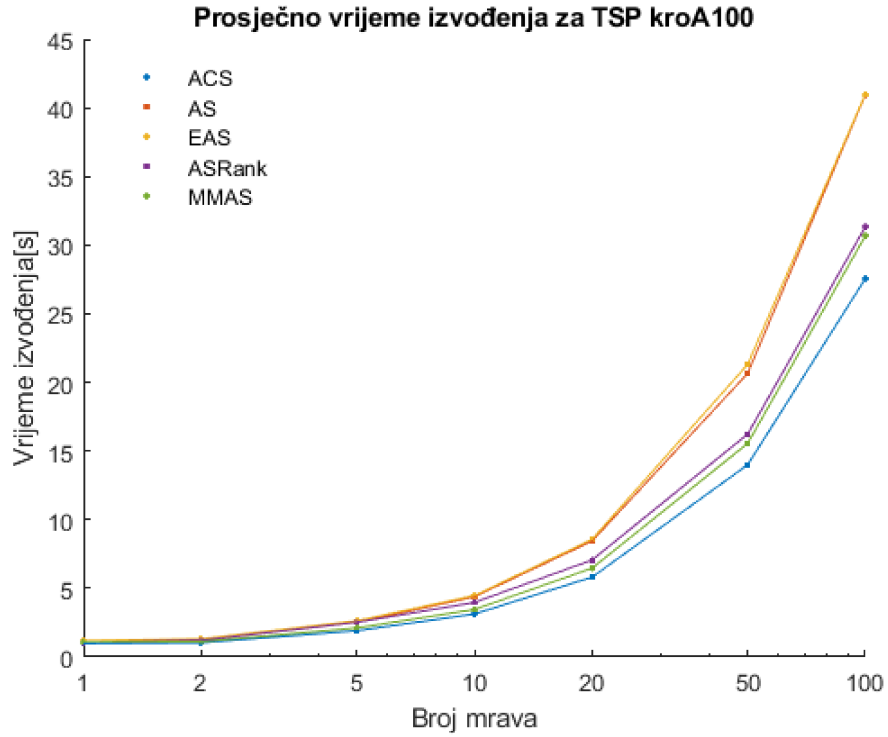
Graf 6.3: Prosječne vrijednosti udaljenosti za TSP dj38 u ovisnosti o broju mrava



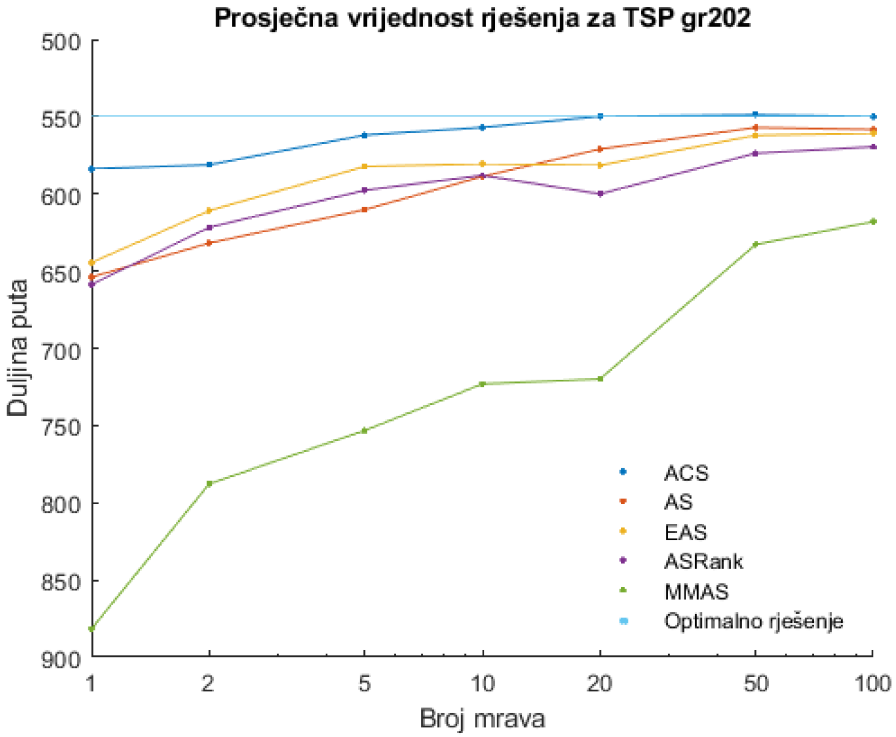
Graf 6.4: Prosječno trajanje izvođenja algoritama za TSP dj38 u ovisnosti o broju mrava



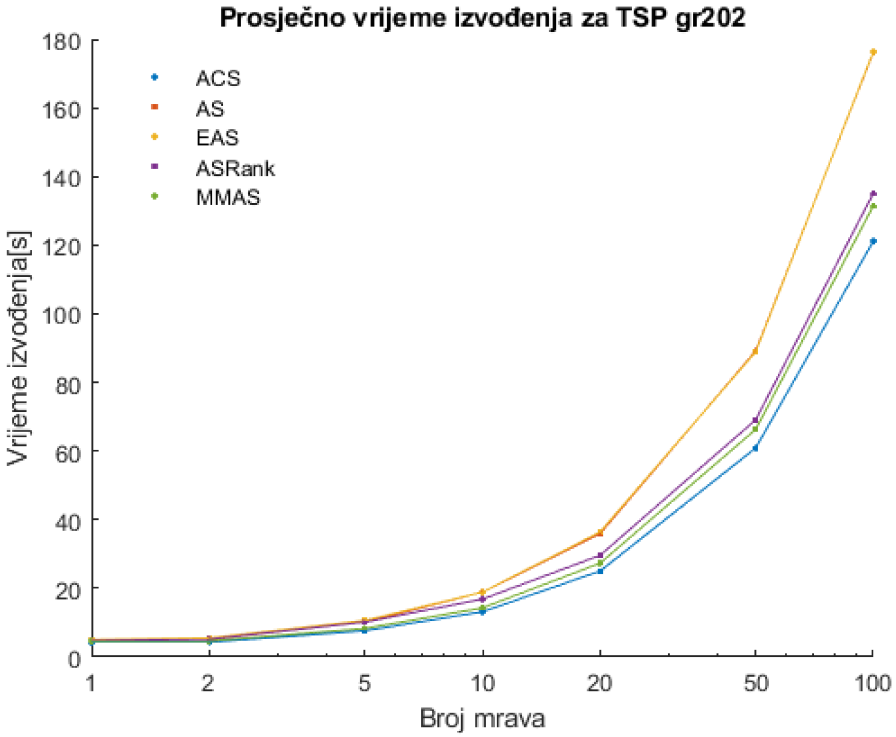
Graf 6.5: Prosječne vrijednosti udaljenosti za TSP kroA100 u ovisnosti o broju mrava



Graf 6.6: Prosječno trajanje izvođenja algoritama za TSP kroA100 u ovisnosti o broju mrava



Graf 6.7: Prosječne vrijednosti udaljenosti za TSP gr202 u ovisnosti o broju mrava



Graf 6.8: Prosječno trajanje izvođenja algoritama za TSP gr202 u ovisnosti o broju mrava

Na grafovima na kojima se prikazuje prosječna vrijednost udaljenosti prikazana je i najbolja poznata vrijednost za taj problem. Budući da su na grafovima prikazane samo vrijednosti dobivene kao prosječna vrijednost 5 testiranja provedenih nad istim parametrima, najbolje dobivene vrijednosti nisu prikazane na grafu. Ovim su testiranjima na nekim od problema pronađena bolja rješenja od trenutnih najboljih poznatih, što se može vidjeti i na kod nekih prosječnih vrijednosti (Graf 6.7).

Što se tiče dobivenih rezultata, vidljivo je da se algoritmi ponašaju različito na različitim konkretnim problemima trgovačkog putnika. Na problemima trgovačkog putnika manjih dimenzija, bolji rezultati se dobivaju jednostavnijim algoritmima poput AS i EAS, posebice pri manjem broju mrava. S druge strane, na većim problemima trgovačkog putnika, algoritam ACS vidljivo dominira nad ostalim algoritmima. S grafova je vidljivo i da algoritam $MMAS$ daje to lošija rješenja što je problem trgovačkog putnika složeniji. Razlog leži u tome što su parametri $\tau_{\min}$ i $\tau_{\max}$ fiksirani, a njihove vrijednosti uvelike utječu na ponašanje algoritma — kod određenih dimenzija problema trgovačkog putnika fiksirani parametri će rezultirati dobrim rješenjem (`ulysses16`), dok će kod ostalih problema rezultirati znatno lošijim rješenjima od ostalih algoritama.

Nasuprot tome, na grafovima koji prikazuju vremena izvođenja algoritama, kroz sva 4 grafa može se raspoznati uzorak:

- algoritmi AS i EAS imaju jako slična vremena izvođenja te su vidljivo najsporiji;
- algoritmi AS_{Rank} i $MMAS$ također imaju slična vremena izvođenja te su ponešto brži od prethodna dva;
- algoritam ACS je u svim slučajevima vidljivo najbrži.

Dobiveni rezultati ne iznenađuju zbog sličnosti među algoritmima sa sličnim vremenima izvođenja. Algoritmi AS i EAS ažuriraju vrijednost feromona na *svim* posjećenim bridovima i reduciraju vrijednost feromona na *svim* bridovima grafa. Kod algoritama $MMAS$ i AS_{Rank}, vrijednost feromona ažurira samo određen broj mrava. Algoritam ACS je najbrži zbog putpuno drugačijeg pristupa načinu obilaska grafa, kao i ažuriranju feromona.

Dobiveni rezultati bi kod nekih problema mogli biti i bolji kad bi se parametri prilagodili konkretnom problemu, međutim, to je dug i mukotrpan posao koji se često povjerava drugim metaheuristikama.

7. Zaključak

Algoritmi evolucijskog računanja, a među njima i mravlji algoritam, pokazuju se kao vrlo dobri algoritmi pri aproksimaciji rješenja optimizacijskih problema. Iako ta rješenja u većini slučajeva nisu optimalna, ona su često dovoljno dobra. Vrijeme izvođenja algoritama čini „dovoljno dobra" rješenja još boljima.

Izvođenje deterministčkih algoritama kojima bi dobiveno rješenje bilo optimalno moglo bi, za dovoljno velik problem, potrajati i tisućama godina. Nasuprot tome, korištenje algoritama evolucijskog računarstva će vrlo često rezultirati rješenjem koje se nalazi čak i među 1% najboljih, a dobiveno je za svega nekoliko minuta.

Ova tvrdnja je potvrđena i programskom implementacijom koja je nastala u sklopu ovog rada. Pokazalo se da su mravlji algoritmi vrlo moćno sredstvo za rješavanje optimizacijskih problema koje daje kvalitetne rezultate u jako kratkom vremenu. Brzina izvođenja algoritama je dodatno pospješena paralelizacijom, što je ponajviše vidljivo u slučaju algoritma ACS.

Zahvaljujući upravo svojoj brzini i kvaliteti rješenja te primjenjivosti na širokom spektru problema, algoritmi evolucijskog računanja danas su vrlo popularni i često korišteni u brojnim područjima.

LITERATURA

[1] L. Budin, M. Golub, D. Jakobović, i L. Jelenković. *Operacijski sustavi*. Element, Zagreb, 2010.

[2] J. Cheng. Numerical optimization. `http://www.jade-cheng.com/au/coalhmm/optimization/`, nepoznata godina. *slika preuzeta i uređena*: 18. 5. 2020.

[3] M. Dorigo. Ant colony optimization. *Scholarpedia*, 2(3):1461, 2007. doi: 10.4249/scholarpedia.1461. revision #90969, *datum pristupa*: 10. 5. 2020.

[4] M. Dorigo i M. Birattari. Swarm intelligence. *Scholarpedia*, 2(9):1462, 2007. doi: 10.4249/scholarpedia.1462. revision #138640, *datum pristupa*: 9. 5. 2020.

[5] K. Helsgaun. Sw24978 - Sweden. `http://www.math.uwaterloo.ca/tsp/world/swtour.html`, nepoznata godina. *slika preuzeta*: 18. 5. 2020.

[6] J. H. Holland. Genetic algorithms. *Scholarpedia*, 7(12):1482, 2012. doi: 10.4249/scholarpedia.1482. revision #128222, *datum pristupa*: 9. 5. 2020.

[7] L. Jelenković. Interni materijali za predavanja iz predmeta Operacijski sustavi, 2020.

[8] D. Karaboga. Artificial bee colony algorithm. *Scholarpedia*, 5(3):6915, 2010. doi: 10.4249/scholarpedia.6915. revision #91003, *datum pristupa*: 9. 5. 2020.

[9] korisnici Wikipedie. Travelling salesman problem. `https://en.wikipedia.org/w/index.php?title=Travelling_salesman_problem&action=info`, 2001. *datum pristupa*: 10. 5. 2020.

[10] Korisnici Wikipedije. Monitor (synchronization). `https://en.wikipedia.org/wiki/Monitor_(synchronization)`, 2006. *slika preuzeta i uređena*: 18. 5. 2020.

[11] N. Mishra. Travelling salesman problem in C and C++. https://www.thecrazyprogrammer.com/2017/05/travelling-salesman-problem.html, 2017. *slika preuzeta*: 18. 5. 2020.

[12] A. Nakić i M.O. Pavčević. *Uvod u teoriju grafova*. Element, Zagreb, 2015.

[13] Nepoznati autor. Ant colony optimizaiton. https://notyetsecure.com/ant-colony-optimization.html, 2011. *slika preuzeta*: 18. 5. 2020.

[14] Morteza Rahmanpour. How can i decide the stopping criteria in genetic algorithm? https://www.researchgate.net/post/How_can_I_decide_the_stopping_criteria_in_Genetic_Algorithm, 05 2019. *datum pristupa*: 9. 5. 2020.

[15] Stephan. What are the differences between genetic algorithms and evolution strategies? https://stackoverflow.com/a/7833415, 10 2011. *datum pristupa*: 9. 5. 2020.

[16] M. Yaghini. Ant colony optimization, part 4: Algorithms. http://webpages.iust.ac.ir/yaghini/Courses/AOR_881/Ant%20Colony%20Optimization_04.pdf, 2009. *datum pristupa*: 10. 5. 2020.

Radni okvir za rješavanje problema trgovačkog putnika paralelnim algoritmom kolonije mrava

Sažetak

U ovom je radu prikazan način rada mravljih algoritama i njihova primjena na rješavanju problema trgovačkog putnika. Prikazan je i način upotrebe radnog okvira za rješavanje problema trgovačkog putnika paralelnim algoritmima kolonije mrava. Navedene su prednosti i mane implementirane paralelizacije višedretvenošću. Grafički je prikazana i statistika dobivenih rješenja, kao i vremena izvođenja algoritama.

Ključne riječi: radni okvir, mravlji algoritam, problem trgovačkog putnika, paralelizacija, višedretvenost, evolucijsko računarstvo

Parallel ant colony optimization framework for the travelling salesman problem

Abstract

This thesis shows the principles of the ant colony optimization algorithms, as well as their usage in solving the travelling salesman problem. The way of using the framework for solving the travelling salesman problem using the parallel ant colony optimization algorithms is also shown. The benefits, as well as the shortcomings of the multi-thread parallelization, are stated in the thesis. The results and execution times statistics of the algorithms are shown in the graphs.

Keywords: framework, ant colony optimization, travelling salesman problem, parallelization, multi-threading, evolutionary computation