Algoritam kolonije mrava

Goran Brajdić

1. listopada 2016.

Sažetak

Algoritam kolonije mrava spada u kategoriju heurističkih algoritama te se pokazao vrlo uspješan u rješavanju problema trgovačkog putnika. Za problem trgovačkog putnika zna se da je *NP-težak*, te nam je cilj u ovom radu predstaviti upravo algoritam kolonije mrava tj. njegov matematički model, njegove primjene u praksi, te probati dati prijedloge za poboljšanje tog algoritma u svrhu povećanja performansi tj. konvergencije optimalnom rješenju.

Sadržaj

1 Uvod

Mravi su izrazito jednostavna bića usporedimo li ih primjerice sa čovjekom. No i tako jednostavna bića zahvaljujući svojim socijalnim interakcijama postižu izvanredne rezultate.

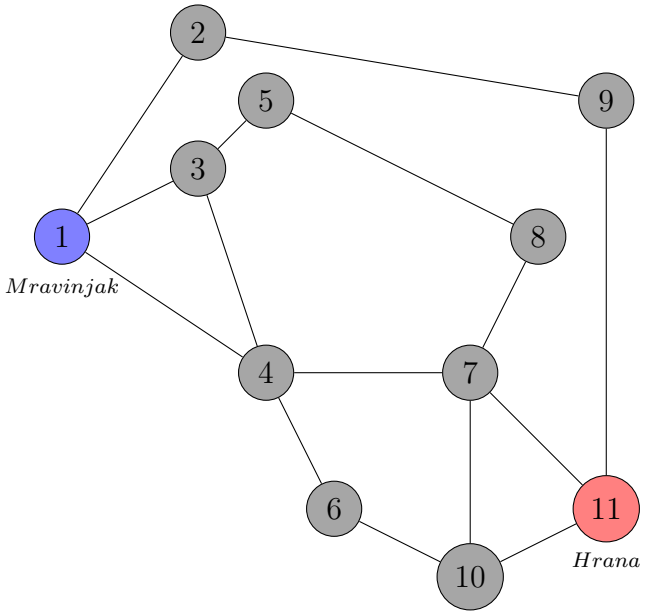
Mnoštvo različitih znanstvenih disciplina pokazalo je interes za mrave. Nama su mravi posebno zanimljivi iz vrlo praktičnog razloga – mravi uspješno rješavaju optimizacijske probleme. Naime, uočeno je da će mravi uvijek naći najkraći put od mravinjaka do izvora hrane, što im omogućuje da hranu dopremaju maksimalno brzo. Pri tome moramo imati na umu da mravlje vrste ili uopće nemaju razvijen vidni sustav ili je on ekstremno loš.

Mravi prilikom kretanja ne koriste osjet vida, već je njihovo kretanje određeno socijalnim interkacijama s drugim mravima. Na putu od mravinjaka do hrane, te na putu od hrane do mravinjaka svaki mrav ostavlja kemijski trag – feromone. Mravi imaju razvijen osjet feromona, te prilikom odlučivanja kojim putem krenuti tu odluku donose obzirom na jakost feromonskog traga koji osjećaju. Tipično, mrav će se kretati smijerom jačeg feromonskog traga. Svaki novi mrav koji se priključi pronalasku hrane pojačavati će trag feromona i tako privlačiti još više novih mrava. Riječ je o strategiji novačenja koji radi na principu povratne veze.

Kroz vrijeme trag feromona počinje slabiti, što je dobra stvar jer služi da bi se izbjegla neograničena akumulacija feromonskih tragova i omogučuje zaboravljanje prethodno poduzetih loših odluka tj. omogučuje zaboravljanje puteva sa velikim težinama.

Iako su njihove interakcije primitivne na nižoj razini, na onoj višoj – globalnoj razini često daju impresivne rezultate. Takvo kolektivno ponašanje koje izranja iz grupe naziva se inteligencija roja. Glavne prednosti su fleksibilnost (brza prilagodljivost promjeni okoline), robusnost (otpornost na manja odstupanja), te samo - organizacija (sposobnost funkcioniranja bez nadzora). Algoritam kolonije mrava upravo pripada klasi inteligencije roja.

U ovom radu osnovna ideja je krenuti od jednog kanonskog algoritma kolonije mrava, te ga raznim modifikacijama poboljšavati kako bi dobili precizniji i brži algoritam. Na kraju kako bi ustanovili koliko su naši algoritmi zapravo dobri, željeli bismo testirati dobivene algoritme i raznim metodama izmjeriti njihove performanse.



Slika 1: Primjer pronalaska hrane

2 Matematički model algoritma kolonije mrava

2.1 Jednostavni matematički model

Matematički model opisa kretanja mrava dan je u [?].
Mravi prilikom kretanja u oba smijera neprestano ostavljaju feromonski trag. Simulacije ovakvih sustava, posebice uzmemo li u obzir dinamiku isparavanja feromona izuzetno su kompleksne. Međutim, ideje i zakonitosti koje su ovdje uočene našle su primjenu u nizu mravljih algoritama – u donekle pojednostavljenom obliku.

Pogledajmo to na primjeru. Između mravinjaka i hrane nalazi se niz tunela (Slika 1, tuneli su modelirani bridovima grafa). Ideja algoritma je jednostavna. U fazi inicijalizacije, na sve se bridove postavi ista količina feromona. U prvom koraku mrav iz mravinjaka (čvor 1) mora odlučiti u koji će čvor krenuti. Ovu odluku donosi na temelju vjerojatnosti odabira p_{ij}^k:

$$p_{ij}^k = \begin{cases} \frac{\tau_{ij}^\alpha}{\sum_{l \in N_i^k} \tau_{il}^\alpha}, & \text{ako } j \in N_i^k \\ 0, & \text{ako } j \notin N_i^k \end{cases} \tag{1}$$

pri čemu τ_{ij} predstavlja vrijednost feromonskog traga na bridu između i i j, a α predstavlja konstantu koja određuje relativnu važnost vrijednosti feromona. Skup N_{ij}^k predstavlja skup svih indeksa svih čvorova u koje je u koraku k moguće prijeći iz čvora i. Konkretno u našem slučaju $N_1^1 = 2, 3, 4$. Ako iz čvora i nije moguće prijeći u čvor j, vjerojatnost će biti 0. Suma u nazivniku prethodnog izraza ide, dakle po svim bridovima koji vode do čvorova iz kojih se može stići iz čvora i.

Nakon što odabere sljedeći čvor, mrav ponavlja proceduru sve dok ne stigne do izvora hrane (čvor 11). Uočimo kako se rješenje problema ovom tehnikom gradi dio po dio – mravlji algoritmi pripadaju porodici konstrukcijskih algoritama.

Jednom kada stigne do izvora hrane, mrav zna koliki je put prešao. Naši umjetni mravi feromone tipično ostavljaju na povratku, i to na način da je količina feromona proporcionalna dobroti rješenja (odnosno obrnuto proporcionalna duljini puta). Ažuriranje radimo za sve bridove kojima je mrav prošao, i to prema izrazu:

$$\tau_{ij} \leftarrow \tau_{ij} + \Delta\tau^k$$

gdje je:

$$\Delta\tau^k = \frac{1}{L},$$

pri čemu je L duljina pronađenog puta. Isparavanje feromona modelirano je izrazom:

$$\tau_{ij} \leftarrow (1 - \rho) \cdot \tau_{ij}$$

gdje je $\rho \in [0, 1]$ brzina isparavanja. Isparavanje se primjenjuje na sve bridove grafa. Sada kad smo opisali prikazane ideje, pogledajmo u sljedećem poglavlju stvarne algoritme koji se danas koriste.

2.2 Algoritam Ant System

Algoritam *Ant System* predložili su Dorigo i suradnici [**?**, **?**]. Prilikom inicijalizacije grafa deponira se ista količina feromona. Koristi se izraz:

$$\tau_0 = \frac{m}{C^{nn}}$$

gdje je m broj mravi, a C_{nn} najkraća duljina puta pronađena nekim jednostavnim algoritmom (poput algoritma najbližeg susjedstva). Ideja je zapravo dobiti kakvu - takvu procjenu duljine puta od koje će mravi dalje tražiti bolja rješenja, te ponuditi optimalnu početnu točku za rad algoritma.

Rad algoritma započinje tako što m mrava stvara rješenje problema. U slučaju TSP-a, mravi se slučajno raspoređuju po gradovima, iz kojih tada počinju konstrukciju rješenja. Prilikom odlučivanja u koji grad krenuti umjesto pravila prijelaza (1) prikazanog u prethodnom poglavlju, mravi koriste pravilo koje uključuje dvije komponente: jakost prethodno deponiranog feromonskog traga te vrijednost heurističke informacije. Vjerojatnost prelaska iz grada i u grad j određena je izrazom:

$$p_{ij}^k = \begin{cases} \frac{\tau_{ij}^\alpha \cdot \eta_{ij}^\beta}{\sum_{l \in N_i^k} \tau_{il}^\alpha \cdot \eta_{il}^\beta}, & \text{ako } j \in N_i^k \\ 0, & \text{ako } j \notin N_i^k \end{cases} \tag{2}$$

η_{ij} je pri tome heuristička informacija koja govori koliko se čini da je dobro iz grada i otići u grad j. Ta informacije je tipično poznata unaprijed i u slučaju problema trgovačkog putnika računa se kao $\eta_{ij} = \frac{1}{d_{ij}}$, gdje je d_{ij} udaljenost grada i od grada j. Parametri α i β pri tome određuju ponašanje samog algoritma. Za:

- $\alpha = 0$, utjecaj feromonskog traga se poništava i pretraživanje se vodi samo heurističkom informacijom (imamo stohastički pohlepni algoritam).
- $\beta = 0$, utjecaj heurističke informacije se poništava, i ostaje utjecaj isključivo feromonskog traga, što često dovodi prebrzoj konvergenciji suboptimalnom rješenju (gdje mravi slijede jedan drugog po relativno lošoj stazi).

Pravilo (2) naziva se *slučajno proporcionalno pravilo (engl. random proportional rule)*. Nakon što su mravi napravili rješenja, rješenja se vrednuju. Potom se pristupa isparavanju feromona sa svih bridova prema izrazu:

$$\tau_{ij} \leftarrow (1 - \rho) \cdot \tau_{ij}.$$

Nakon isparavanja svi mravi deponiraju feromonski trag na bridove kojima su prošli, i to proporcionalno dobroti rješenja koje su pronašli:

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k} \tag{3}$$

pri čemu je:

$$\Delta\tau_{ij}^{k} = \begin{cases} \frac{1}{C^k}, & \text{ako je brid } i-j \text{ na stazi } k\text{-tog mrava} \\ 0, & \text{inače} \end{cases} \tag{4}$$

gdje je C^k duljina ture koju je kontruirao k-ti mrav. Iz relacije (4) zaključujemo da što je bolje rješenje mrava više se feromona deponira na lukove koji pripadaju toj turi.

Implementacija ovog algoritma dana je pseudokodom:

Algorithm 2.1: Pseudokod algoritma Ant System

Input: $\alpha \geq 0$, $\beta \geq 0$, broj mravi m, brzina isparavanja ρ, n gradova
Output: **Najbolje pronađeno rješenje**
Inicijaliziraj feromonske tragove;
while (!*zadovoljen kriterij*) **do**
  for $ant \leftarrow 1$ ***to*** m **do**
    $S = \{1, 2, \ldots, n\}$;
    Slučajno izaberi početni grad i;
    while $(!S \neq 0)$ **do**
      Izaberi sljedeći grad j sa vjerojatnnošću $p_{ij} = \frac{\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_{k \in S} \tau_{ik}^{\alpha} \cdot \eta_{ik}^{\beta}}$;
      $S = S - j$;
  $\tau_{ij} \leftarrow (1 - \rho) \cdot \tau_{ij}, \forall i, j \in \{1, \ldots, n\}$;
  $\Delta\tau_{ij}^{k} = \frac{1}{C^{k}}$;
  $\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}$;

2.3 Algoritam Elitist Ant System

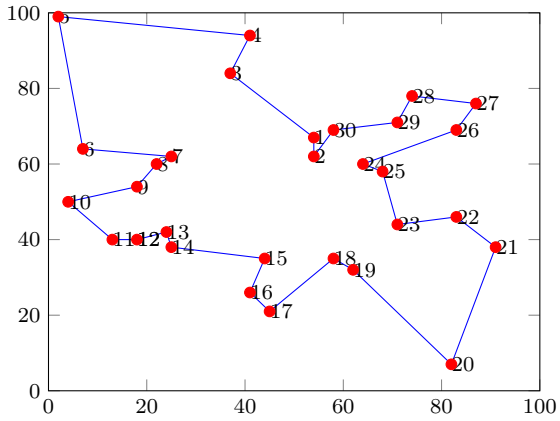
Elitistička verzija algoritma (*engl. Elitist Ant System*) predložena je kao poboljšanje algoritma u [**?**]. U ovoj verziji algoritma dodatno se pamti globalno najbolje pronađeno rješenje. U svakoj iteraciji to se rješenje također koristi za ažuriranje feromonskih tragova s određenom težinom e. Ovo zamišljamo kao da postoje još jedan mrav (označimo ga s **bs**) koji u svakoj iteraciji prođe upravo najboljim zapamćenim putem. Pravilo ažuriranja feromona (3) tada dobiva još jedan član pa glasi:

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k} + e \cdot \Delta\tau_{ij}^{bs}$$

pri čemu je:

$$\Delta\tau_{ij}^{bs} = \begin{cases} \frac{1}{C^{bs}}, & \text{ako je brid } i - j \text{ na stazi } bs\text{-tog mrava} \\ 0, & \text{inače} \end{cases}$$

gdje je C^{bs} duljina globalno najbolje ture.



Slika 2: Rješenje TSP-a dobiveno algoritmom Elitist Ant System

2.4 Algoritam Rank-based Ant System

Kod ovog algoritma nakon što svi mravi kreiraju svoja rješenja, sortiraju se obzirom na kvalitetu tih rješenja. Ažuriranje feromona obavlja samo $w-1$ najboljih mrava proporcionalno svom rangu, te mrav koji čuva globalno najbolje rješenje (koji ulazi s najvećim utjecajem). U tom slučaju pravilo ažuriranja feromona će nam glasiti:

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{w-1}(w-k)\Delta\tau_{ij}^{k} + w \cdot \Delta\tau_{ij}^{bs}$$

Svaki mrav pri tome ažurira samo bridove koji su dio njegovog puta.

2.5 Max-Min Ant System

Max-Min Ant System (MMAS) još je jedno unaprijeđenje algoritma Ant System koje uvodi 4 modifikacije.

1. Samo najbolji mrav može obavljati ažuriranje feromona. Postoje varijante algoritma koje ažuriranje dopuštaju samo najboljem mravu u trenutnoj iteraciji, te varijante algoritama koje ažuriranje dopuštaju i globalno najboljem mravu i najboljem mravu u iteraciji i to različitim intenzitetom.

2. Kako bi se spriječila prerana stagnacija algoritma zbog prethodne modifikacije uvodi se donja i gornja granica na jakost feromonskog traga $\tau_{ij} \in [\tau_{min}, \tau_{max}]$.

3. Inicijalizacija algoritma feromonski trag na bridovima postavlja na njihovi maksimalnu vrijednost. Ovo zajedno s niskom stopom isparavanja osigurava da mravi na početku obavljaju široko pretraživanje prostora rješenja.

4. Svaki puta kada algoritam dođe u stagnaciju, odnosno kada nema poboljšanja u kvaliteti rješenja unutar zadanog broja iteracija, obavlja se reinicijalizacija feromonskih tragova na njihove maksimalne vrijednosti.

Gornja granica se tada postavlja na vrijednost:

$$\tau_{max} = \frac{1}{\rho \cdot C^{bs}}$$

i ažurira kada se pronađe novo najbolje rješenje C^{bs}. Donja granica određena je parametrom a prema izrazu:

$$\tau_{min} = \frac{\tau_{min}}{a},$$

gdje je a paramatar koji traba izabrati.

2.6 Ant Colony System

Kod ove verzije algoritma mrav k koji se nalazi u gradu i prijeći će u grad j prema sljedećem *pseudoslučajnom proporcionalnom pravilu (engl. pseudorandom proportional rule)*:

$$j = \begin{cases} argmax_{l \in N_i^k}\{\tau_{il} \cdot \eta_{il}^{\beta}\}, & \text{ako } q \leq q_0 \\ J, & \text{inače} \end{cases} \tag{5}$$

gdje je q slučajna varijabla uniformno distribuirana na intervalu $[0, 1]$, $q_0\,(0 \leq q_0 \leq 1)$ je parametar, a J je slučajna varijabla dobivena prema distribuciji vjerojatnosti danoj jednadžbom (2) (za $\alpha = 1$). Preciznije, svaki puta kada mrav k mora izabrati grad j u koji će otići on na slučajan način izabire broj $0 \leq q \leq 1$. Ako je $q \leq q0$, tada će prema (5) biti izabran najbolji rub (eksploracija), u protivnom najbolji rub će biti izabran prema *slučajnom proporcionalnom pravilu* (2). U *Ant colony optimization* algoritmu obnavljanje feromona se vrši na dve razine:

Globalna razina obnavljanja feromona Kod ovog pravila obnavljanje feromona dopušta se samo globalno najboljem mravu (tj. mravu koji je konstruirao najkraće rješenje od početka izvođenja algoritma). Globalno obnajvljanje feromona primjenjuje se nekon što svi mravi konstruiraju svoja rješenja u danoj iteraciji. Pravilo je dano sljedećim izrazom:

$$\tau_{ij} \leftarrow (1-\rho) \cdot \tau_{ij} + \rho \cdot \Delta\tau_{ij}^{bs}$$

pri čemu je:

$$\Delta\tau_{ij}^{bs} = \begin{cases} \frac{1}{C^{bs}}, & \text{ako je brid } i-j \text{ na stazi } bs\text{-tog mrava} \\ 0, & \text{inače} \end{cases}$$

gdje je C^{bs} duljina globalno najbolje ture.

Lokalna razina obnavljanja feromona Prilikom konstruiranja svojih tura mravi posjećuju bridove te odmah na njima primjenjuju *lokalno pravilo obnavljanja feromona* (6):

$$\tau_{ij} \leftarrow (1-\delta) \cdot \tau_{ij} + \delta \cdot \tau_0 \tag{6}$$

gdje je τ_0 početni feromonski trag koji se obično postavlja kao $\tau_0 = n \cdot (C_{nn})^{-1}$, δ je parametar brzine isparavanja na lokalnoj razini (obično se postavlja na 0.1), n broj gradova TSP problema, a C_{nn} najkraća duljina puta pronađena nekim jednostavnim algoritmom (poput algoritma najbližeg susjedstva).

3 Primjena i performanse algoritama na problemima trgovačkog putnika

U ovom dijelu napravljena je analiza performansi za implementacije prethodno opisanih pet verzija algoritama kolonije mrava (*Ant System, Elitist Ant System, Rank-based Ant Aystem, Max-Min Ant System te Ant Colony System*).

Svaki od algoritama pokrenut je 25 puta za 100, 500 i 1000 iteracija, na tri različita simetrična problema trgovačkog putnika (benchmark) problema. Na taj način za svaki algoritam dobivena su tri uzorka (za svaku od 100, 500 i 1000 iteracija) na temalju kojih smo mjerili performanse. Sve simulacije u svrhu analize performansi izvođene su na istom računalu.

Algoritmi su pozivani sa sljedećim parametrima: faktor relativne važnosti feromonskog traga i heurističke informacije $\alpha = 1$ i $\beta = 5$, u svaki grad stavili smo jednog mrava, tj broj mravi jednak je broju gradova pojedinog TSP problema $m = n$ (n je broj gradova), faktor isparavanja $\rho = 0.1$, osim kod MMAS algoritma gdje je ρ postavljen na 0.02 (jer želimo osigurati da mravi na početku obavljaju široko pretraživanje prostora), težina feromonskog traga u EAS algoritmu $e = n$, parametar Rank-based algoritma $w = 8$, te je parametar ACS algoritma $q_0 = 0.7$.

Za svaki od tri TSP problema sastavljena je tablica sa rezultatima performansi koja uključuje: najkraću dobivenu turu uzorka duljine 25 *(Best)*, srednju vrijednost uzorka te standardnu devijaciju, duljinu najlošije dobivene ture u danom uzorku *(Max)*, postotak pogotka optimuma u uzorku *(Hit.opt)*, te prosječno vrijeme izvršavanja algoritma u sekundama *(Avg. t(s))*.

U tablici 1 prikazani su rezultati performansi za TSP problem sa 30 gradova (*Oliver30*). Optimalno rješenje tog problema (obzirom na euklidsku udaljenost između gradova) iznosi 423.74. Prvo što možemo primjetiti su vrlo loši rezultati algoritma Ant System koji nije niti jednom u 25 poziva pogodio optimum za 100, 500 i 1000 iteracija, stoga potreba za modifikacijom tog algoritma u vidu dobivene četiri verzije Ant Systema (EAS, Rank, Max-Min AS, ACS) je opravdana. Ostali algoritmi pokazuju relativno slične rezultate pogotka optimuma, ono što valja primjetiti je loš rezultat pogotka optimuma Max-Min algoritma sa 100 iteracija. No, to smo i očekivali jer Max-Min algoritam ima sve bridove inicijalno postavljene na njihove maksimalne vrijednosti, što sa vrlo malenim faktorom isparavnja (0.02) omogućuje mravima na početku široko pretraživanje prostora. Zato je bilo za očekivati vrlo sporu konvergenciju ka optimumu u prvih 100 iteracija. No već za 500 iteracija

vidimo da je algoritam sustigao i ostale svojim performansama. Prosječno vrijeme prva četiri algoritma je relativno isto, dok se ACS algoritam izvršava nešto brže.

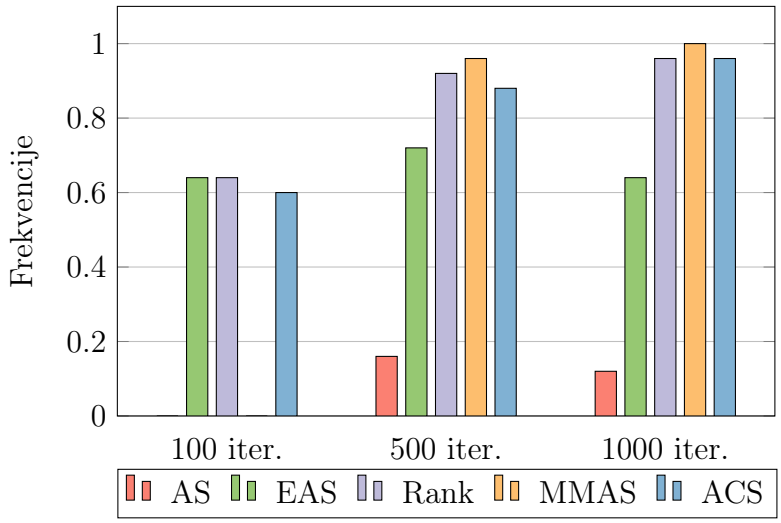
Tablica 1: Usporedba algoritama na problemu Oliver30

Alg.	**Br.iter**	**Best**	**Avg.(st.dev)**	**Max**	**Hit.opt(%)**	**Avg. t(s)**
AS	100	426.37	433.23(3.8)	440.39	0%	3.9
	500	424.46	427.85(2.7)	432.27	0%	17.64
	1000	424.67	427.57(2.5)	434.83	0%	33.58
EAS	100	423.74	424.83(0.8)	426.60	16%	3.9
	500	423.74	424.56(0.9)	427.18	32%	17.15
	1000	423.74	424.69(1.0)	426.60	40%	33.7
Rank	100	423.74	424.85(1.4)	430.70	20%	3.8
	500	423.47	424.17(0.4)	424.90	36%	16.40
	1000	423.74	424.07(0.4)	424.84	44%	33.9
Max-Min	100	425.27	430.64(1.4)	439.54	0%	3.8
	500	423.74	424.04(0.5)	425.65	36%	16.5
	1000	423.74	423.96(0.3)	424.70	48%	31.9
ACS	100	423.74	426.24(5.2)	443.64	24%	2.2
	500	423.75	424.37(1.2)	429.38	36%	9.8
	1000	423.74	424.03(0.4)	424.90	48%	18.6

Do sada smo se kod mjerenja performansi naših verzija algoritama pretežno fokusirali na postotku pogotka optimuma. Kako su naši algoritmi zapravo heuristički algoritmi postizanje optimuma u određenom broju iteracija nije garantirano. Stoga još jedan način na koji možemo izmjeriti performanse je da gledamo koliko vrijednosti (od onog našeg uzorka veličine 25) je upalo u određeni interval sa duljinom optimalne ture u lijevom rubu. Preciznije, koliko puta je duljina ture odstupala od optimuma za manje od 0.25%.

Slika 3 prikazuje stupčasti dijagram rezultata takvog postupka mjerenja performansi. Ono što odmah možemo uočiti su vrlo loši rezultati algoritama Ant System i Max-Min ant system za 100 iteracija. Kod njih niti jedna vrijedost nije odstupala za manje od 0.25% od optimuma. Ostali algoritmi zbog svoje brze konvergencije daju puno bolje rezultate. Povećanjem broja iteracija algoritam Ant System i dalje pokazuje relativno loše rezultate, dok

kod Max-Min Ant System algoritma vidimo značajno poboljšanje, štoviše za veće iteracije on pokazuje najbolje rezultate, te za broj iteracija 1000 vidimo da su sve njegove vrijednosti duljina tura odstupale od optimuma za manje od 0.25%.



Slika 3: Stupčasti dijagram performansi algoritama na problemu Oliver30

Još jedan način da izmjerimo performanse naših algoritama je korištenje statističkih metoda. Kako bi bili sigurni da postoji (ili ne postoji) statistički značajna razlika između rezultata dobivenih od naših pet vezrija algoritama primjenili smo odrađene statističke testove. Prvo što smo htjeli napraviti jest usporediti svih pet algoritama zajedno nekim statističkim testom. Test analize varijance (ANOVA), kao i post-hoc analiza t-testovima nisu bili prikladni jer nam uzorci (25 rezultata dobivenih od svakog od pet verzija algoritama sa 1000 iteracija) nisu bili normalno distribuirani sa jednakim varijancama. Stoga su napravljeni neparametarski testovi.

Kako su svi uzorci naših pet algoritama dobiveni na istom problemu (Oliver30) tj. ulazni parametar za svih pet algoritama su bile koordinate 30 gradova TSP problema Oliver30, morali smo pretpostaviti zavisnost među uzorcima naših pet algoritama. U tu svrhu napravljen je prvo *Friedmanov test* kako bi usporedili svih pet uzoraka odjednom, te je za post-hoc analizu (kako bi mogli ustanoviti koji od pet algoritama pravi razliku, ukoliko bi Friedmanov test odbacio nultu hipotezu da nema stat. značajne razlike između rezultata dobivenih našim algoritmima) korišten *Wilcoxon signed rank*

test uz Bonferroni korekciju p-vrijednosti.

Analiza je napravljena pomoću statističkog alata *R*. Rezultati analize su prikazani sljedećim outputom 1.

Listing 1: R output (Oliver 30)

```
*----------------------------------*
*        Friedman rank sum test     *
*----------------------------------*
data:  data
Friedman chi-squared = 47.57, df = 4, p-value = 1.163e-09
-------------------------------------------------------
 Pairwise comparisons using Wilcoxon signed rank test
-------------------------------------------------------
data:  Value and group

       ACS     AS      EAS     MMAS
AS   0.00013 -       -       -
EAS  0.18445 0.00067 -       -
MMAS 1.00000 0.00019 0.10511 -
Rank 1.00000 0.00015 0.29187 1.00000

P value adjustment method: bonferroni
```

P-vrijednost *Friedmanovog testa* je vrlo malena ($1.163e-09$), tj manja od razine značajnosti $\alpha = 0.05$, zbog čega odbacujemo nultu hipotezu o jednakosti rezultata dobivenih našim algoritmima. Dakle razlike ima. Post-hoc analiza pokazuje nam koji od algoritama prave tu razliku.

Ono što vidimo u usporednoj tablici rezultata outputa je da postoji statistički značajna razlika između algoritma AS i ostala četiri algoritma (EAS, Rank, MMAS i ACS).

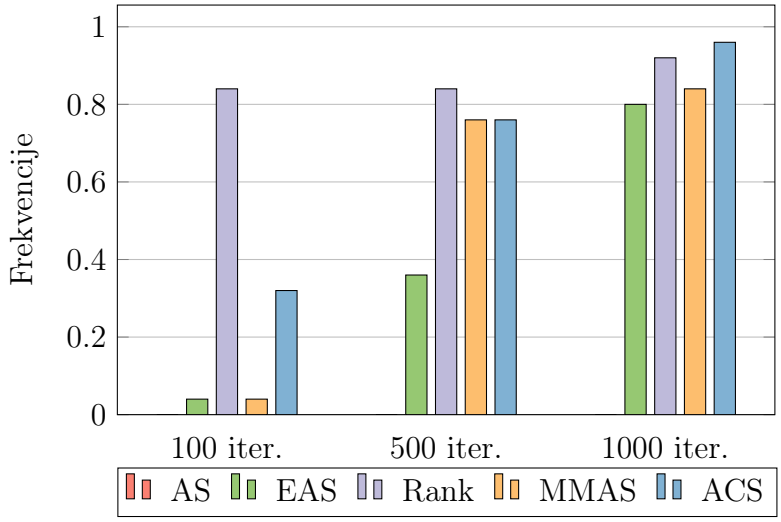
Dakle statistički gledano na temelju uzorka dobivenog iz svakog od pet implementiranih algoritama (u 1000 itaracija) zaključujemo da četiri implementirana algoritma (EAS, Rank, MMAS i ACS) daju bolje rezultate od našeg prvotnog algoritma Ant System na razini značajnosti $\alpha = 0.05$.

U tablici 2 prikazani su rezultati performansi za TSP problem sa 52 grada (*Berlin52*). Optimalno rješenje tog problema (obzirom na euklidsku udaljenost) iznosi 7544.4. Ponovno vidimo vrlo loše rezultate algoritma Ant System. Ono što valja primjetiti su vrlo dobri rezultati algoritma Ant Colony System koji je za 1000 iteracija u 84% slučaja našao optimalno rješenje. Algoritam Max-Min Ant System ponovno pokazuje dosta loše rezultate za manje brojeve iteracija (100 i 500), dok za 1000 iteracija pokazuje znatno poboljšanje. Što se tiće vremena izvođenja ACS algoritam se izvodi najbrže.

Tablica 2: Usporedba algoritama na problemu Berlin52

Alg.	**Br.iter**	**Best**	**Avg.(st.dev)**	**Max**	**Hit.opt(%)**	**Avg. t(s)**
AS	100	7915.3	8095.3(96.0)	8312.4	0%	14.1
	500	7819.2	7977.4(76.7)	8117.9	0%	70.8
	1000	7611.0	7895.5(100.4)	8050.8	0%	140.6
EAS	100	7549.0	7711.6(69.3)	7873.0	0%	14.4
	500	7544.4	7611.2(59.2)	7725.0	8%	72.5
	1000	7544.4	7558.0(27.9)	7664.5	40%	144.7
Rank	100	7544.4	7564.0(42.2)	7712.6	32%	14.6
	500	7544.4	7558.5(33.2)	7663.6	44%	56.5
	1000	7544.4	7555.6(32.8)	7676.8	56%	103.5
Max-Min	100	7555.4	7980.0(162.7)	8235.2	0%	16.2
	500	7544.4	7580.6(58.6)	7724.5	8%	68.8
	1000	7544.4	7563.1(44.6)	7721.3	40%	136.2
ACS	100	7544.4	7659.8(93.3)	7849.6	8%	7.2
	500	7544.4	7570.7(48.1)	7693.4	32%	36.1
	1000	7544.4	7546.7(9.8)	7593.4	84%	71.1

Na slici 4 prikazan je stupčasti dijagram vrijednosti koja su odstupala za manje od 0.25% od optimalne vrijednosti na problemu (*Berlin52*). Očekivano, i kod ovog problema možemo primjetiti vrlo loše rezultate Ant System algoritma kroz sve iteracije (100, 500 i 1000). Također vidimo da je i vrlo malo vrijednosti (tj. duljina najkraćih tura) koje su dali algoritmi Elitist Ant System i Max-Min Ant System odsupalo od optimuma za 0.25%. Ono što još možemo uočiti je da algoritam Rank-based Ant System daje vrlo dobre rezultate u svim iteracijama. Zbog svoje brze konvergencije vidimo da daje daleko najbolje rezultate usporedno sa ostalim algoritmima već za 100 iteracija.



Slika 4: Stupčasti dijagram performansi algoritama na problemu Berlin52

Na kraju za problem *Berlin 52* u 1000 iteracija, isto kao i prije napravljena je analiza usporedbi algoritama statističkim testovima. Analiza napravljena u statističkom alatu R dala nam je sljedeći output:

Listing 2: R output (Berlin 52)

```
*-----------------------------------*
*        Friedman rank sum test     *
*-----------------------------------*
data:  data
Friedman chi-squared = 66.4652, df = 4, p-value = 1.264e-13
-------------------------------------------------------------
 Pairwise comparisons using Wilcoxon signed rank test
-------------------------------------------------------------
data:  Value and group

     ACS     AS      EAS     MMAS
AS   0.00013 -       -       -
EAS  0.07822 0.00013 -       -
MMAS 0.03109 0.00013 1.00000 -
Rank 0.62105 0.00013 1.00000 0.75437

P value adjustment method: bonferroni
```

P-vrijednost *Friedmanovog testa* je vrlo malena (1.2864e-13), tj. manja je od razine značajnosti $\alpha = 0.05$. Pa odbacujemo nultu hipotezu o jednakosti rezultata dobivenih našim algoritmima. Stoga zaključujemo da razlika postoji.

Post-hoc analiza dala nam je tablicu usporednih rezultata gdje vidimo da postoji statistički značajna razlika između algoritma Ant System i ostala četiri algoritma.

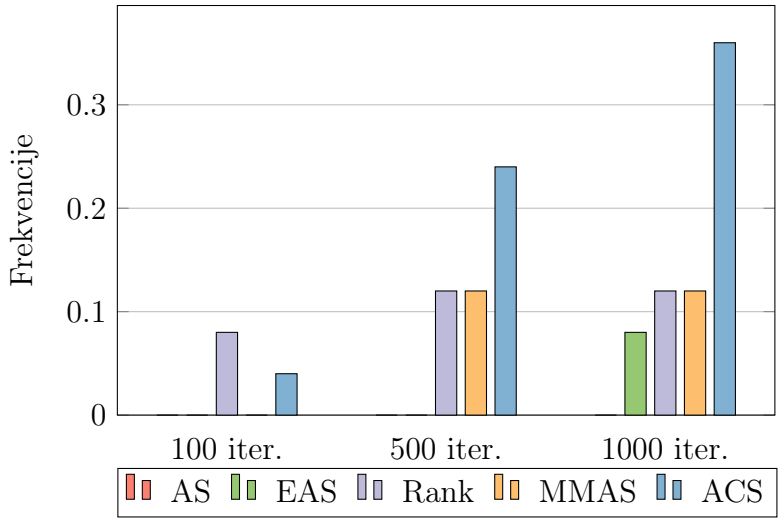
Zadnja simulacija napravljena je na benchmark problemu trgovačkog putnika sa 48 gradova (*Att 48*). Rezultati su prikazani u tablici 3. Ovog puta rezultati performansi svih algoritama su vrlo loši, s time da algoritam Ant Colony System ima najbolji rezultat pogotka optimuma, kao i najbrže vrijeme izvršavanja. Loši rezultati nad pojedinim benchmark problemima trgovačkog putnika nisu u potpunosti neočekivani jer su sve simulacije pokratane sa istim parametrima (faktori rel. važnosti feromonskog traga i heurističke informacije, kao i faktor brzine isparavanja). Stoga je za pretpostaviti da isti parametri neće odgovarati svim TSP problemima, tj. nekim drugim izborom parametra vjerojatno je moguće značajno povećati performanse algoritama.

Tablica 3: Usporedba algoritama na problemu Att 48

Alg.	**Br.iter**	**Best**	**Avg.(st.dev)**	**Max**	**Hit.opt(%)**	**Avg. t(s)**
AS	100	35203	36088(476.5)	36912	0%	12.1
	500	34419	35289(362.1)	35917	0%	62.6
	1000	34495	35107(353.1)	35620	0%	123.0
EAS	100	33903	34424(259.7)	34888	0%	12.0
	500	33613	33972(164.9)	34247	0%	62.0
	1000	33549	33831(172.6)	34198	0%	122.7
Rank	100	33601	34144(294.1)	34686	0%	11.9
	500	33549	33826(182.9)	34174	0%	49.9
	1000	33549	33809(166.1)	34161	0%	87.9
Max-Min	100	34753	35642(453.5)	36435	0%	11.6
	500	33549	34038(236.8)	34424	0%	58.6
	1000	33524	34007(229.1)	34319	4%	115.2
ACS	100	33608	34648(557.1)	35828	0%	6.0
	500	33524	33778(209.3)	34110	12%	31.2
	1000	33524	33689(153.2)	34131	16%	59.6

Slika 5 prikazuje stupčasti dijagram vrijednosti koja su odstupala za manje od 0.25% od optimalne vrijednosti na problemu (*Att 48*). Vidimo nešto lošije

performanse svih algoritama, ali još uvijek bolje od klasičnog Ant System algoritma.



Slika 5: Stupčasti dijagram performansi algoritama na problemu Att 48

Za problem *Att 48* u 1000 iteracija, isto kao i prije napravljena je analiza usporedbi algoritama statističkim testovima. Analiza napravljena u statističkom alatu *R* dala nam je sljedeći output:

Listing 3: R output (Att 48)

```
*-----------------------------------*
*        Friedman rank sum test     *
*-----------------------------------*
data:  data
Friedman chi-squared = 63.3414, df = 4, p-value = 5.751e-13
-----------------------------------------------------------
 Pairwise comparisons using Wilcoxon signed rank test
-----------------------------------------------------------
data:  Value and group

     ACS     AS      EAS     MMAS
AS   0.00013 -       -       -
EAS  0.09786 0.00013 -       -
MMAS 0.00165 0.00013 0.11449 -
Rank 0.20639 0.00013 1.00000 0.02578

P value adjustment method: bonferroni
```

P-vrijednost *Friedmanovog testa* je vrlo malena (5.751e-13), tj. manja je od razine značajnosti $\alpha = 0.05$. Pa odbacujemo nultu hipotezu o jednakosti rezultata dobivenih našim algoritmima. Stoga zaključujemo da razlika postoji. Post-hoc analiza dala nam je tablicu usporednih rezultata gdje vidimo da postoji statistički značajna razlika između algoritma Ant System i ostala četiri algoritma. Također vidimo i statistički značajnu razliku između algoritama MMAS i ACS, te između MMAS i Rank algoritama. Tu razliku možemo objasniti vrlo lošom performansom MMAS algoritma na problemu *Att 48*,jednim dijelom zbog spore konvergencije ka optimumu, a drugim dijelom zbog lošeg izbora parametara.

Na kraju za zaključak ove sekcije o performansama naših algoritama na tri različita (benchmark) problema trgovačkog putnika možemo reći da za manji broj iteracija algoritam Rank-based Ant System pokazuje najbolje rezultate zbog svoje brze konvergencije kao optimalnom rješenju, te bez obzira što algoritam možda ne pogodi optimalno rješenje u malom broju iteracija (tj često "zaglavi" u lokalnom optimumu), on mu se najviše približi. Stoga za maleni broj iteracija preporuka je koristiti Rank-based algoritam.

Za veći broj iteracija preporučujemo koristiti algoritme Ant Colony System i Max-Min Ant System, koji imaju sporiju konvergenciju ka optimumu, no njihova eksploracija puteva je veća, te je samim time i veća vjerojatnost pronalaska optimalnog rješenja. Malenu prednost dali bi ipak Ant Colony System algoritmu zbog kraćeg vremena izvršavanja.

4 Dodatak – programska dokumentacija

U nastavku slijedi programska dokumantacija koda u Matlab-u. Algoritmi su implementirani kao funkcije koji primaju parametare, te vraćaju dva parametra – duljinu najkraće ture i broj iteracije u kojoj je najkraća tura pronađena. Algoritmi također crtaju gradove spojene najkraćom pronađenom turom, te crtaju konvergenciju ka optimalnom rješenju po iteracijma. Na kraju ispisuju redosljed gradova kojim bi se trgovački putnik trebao kretati u cilju postizanja najkraće ture.

Kako imamo implementiranih pet verzija algoritama, a željeli bismo korisniku omogučiti što jednostavnije korištenje istih za različite probleme trgovačkog putnika, napravili smo input korisničko sučelje što više "user friendly". U glavnom programu (ACO_EXE()) implementiran je određen broj problema trgovačkog putnika koje korisnik može izabrati za rješavanje, nakon što se odluči koji problem će rješavati, odlučuje kojim algoritmom ga želi riješavati, te nakon izbora algoritma bira odgovarajuće parametre. Sve te odluke (tj. upiti) napravljeni su preko Matlab-ovog "dialog box-a". Svaki algoritam implementiran je kao jedna funkcija koju glavni program poziva preko dialog boxa, ovisno o želji korisnika.

Funkcije su:

- TSP_AS_ACO() – funkcija u kojoj je implementiran Ant System algoritam
- TSP_EAS_ACO() – funkcija u kojoj je implementiran Elitisti Ant System algoritam
- TSP_Rank_ACO() – funkcija u kojoj je implementiran Rank-based Ant System algoritam
- TSP_MMAS_ACO() – funkcija u kojoj je implementiran Max-Min Ant System algoritam
- TSP_ACS_ACO() – funkcija u kojoj je implementiran Ant Colony System algoritam
- nn_tsp() – funkcija u kojoj je implementiran algoritam najbližeg susjeda, koja služi kao pomoćna funkcija za izračun početnih stanja feromona u gore navedenim algoritama.

Samo kod napisan u Matlabu je komentiran, te osnovne strukture koje smo koristili su vektori i matrice za čiji rad je Matlab vrlo pogodan. Struktura samih funkcija u kojima su implementirani algoritmi podjeljena je u 3 osnovna djela. Prvi dio obuhvaća inicijalizaciju i računanje (euklidske) udaljenosti među gradovima čije koordinate dobivamo preko parametra (cities). Drugi dio je sam algoritam, koji prolazi kroz iteracije u kojima mravi kreiraju svoja rješenja (tj. dio u kojem svaki mrav kreira svoju turu nakon čega nastupa obnavljanje i isparavnje feromona). Te zadnji dio koji se odnosi na output rezultata (tj. crtanje grafa najkraće pronađene ture i grafa koji prikazuje konvergenciju ka optimumu po iteracijama, te sam ispis te ture i njene udaljenosti).

Na samom kraju prikazati ćemo ulazne parametre koje algoritmi primaju. Ulazni paramtri su:

- cities – koordinate gradova
- Nants – broj mravi
- a (α) i b (β) – paramtri koji kontroliraju relativnu važnost vrijednosti feromona tj heurističke informacije.
- rr (ρ) – parametar isparavanja
- e – težina elitističkog puta (za elitističku verziju algoritma)
- k – broj mravi koji će ažurirati feromone (za rank-based algoritam)
- q_0 – parametar eksploracije (za ant colony system verziju)
- Q – proizvoljna konstanta
- maxit – maksimalan broj iteracija kroz koji će algoritam proći