Algoritmi optimizacije kolonijom mrava

Hršak, Iva

SVEUČILIŠTE U ZAGREBU
PRIRODOSLOVNO–MATEMATIČKI FAKULTET
MATEMATIČKI ODSJEK

Iva Hršak

ALGORITMI OPTIMIZACIJE KOLONIJOM MRAVA

Diplomski rad

Voditelj rada:
izv. prof. dr. sc. Saša Singer

Zagreb, studeni 2018.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Sadržaj

Sadržaj **iii**

Uvod **1**

1 Optimizacija kolonijom mrava **2**
1.1 Motivacija iz prirode 2
1.2 Definicija problema 5
1.3 Ponašanje mrava 6
1.4 ACO metaheuristika 8
1.5 Primjena ACO metaheuristike na TSP 9

2 Vrste ACO algoritama **11**
2.1 Mravlji sustav 11
2.2 MAX–MIN mravlji sustav 14
2.3 Sustav mravlje kolonije 17
2.4 Implementacija ACO algoritama 19

3 Eksperimentalni rezultati **27**
3.1 Rezultati algoritama nad kružno i slučajno raspoređenim gradovima . 27
3.2 Ovisnost o parametrima 32

4 Kod implementacije ACO algoritama u Pythonu **34**

Bibliografija **43**

Uvod

Algoritmi optimizacije kolonijom mrava (engl. Ant Colony Optimization, skraćeno ACO) su populacijski orijentirane metaheuristike koje se koriste za rješavanje teških optimizacijskih problema, ponajviše kombinatornih NP-teških problema.

Ovu vrstu algoritma prvi je predstavio Marco Dorigo u svojoj doktorskoj disertaciji. Prvi ACO algoritam korišten je za nalaženje optimalnog puta u grafu, a algoritam je oponašao ponašanje mrava u potrazi za hranom. Danas ACO algoritmi imaju svestranu primjenu, a najpoznatije su za rješavanje problema raspoređivanja, problema usmjeravanja vozila (engl. vehicle routing), detekciju rubova na slikama, itd.

U prvom poglavlju govorimo o tome otkuda dolazi motivacija za ACO algoritme. Nakon toga, objasnit ćemo kako problem kojeg promatramo, možemo poistovjetiti s prikazom kojeg umjetni mravi mogu koristiti kod konstrukcije rješenja. Objasnit ćemo koje su sve karakteristike umjetnih mrava. Zatim ćemo opisati ACO metaheuristiku i njezinu primjenu na problem trgovačkog putnika (engl. travelling salesman problem, skraćeno TSP).

U drugom poglavlju bavimo se vrstama ACO algoritama. Prvo ćemo opisati prvi ACO algoritam, koji se zove Mravlji sustav (engl. Ant System, skraćeno AS). Nakon toga, opisujemo dva poboljšanja navedenog algoritma, a to su algoritmi MAX–MIN mravlji sustav (engl. MAX–MIN Ant System, skraćeno MMAS) i Sustav mravlje kolonije (engl. Ant Colony System, skraćeno ACS). Za svaki od tih algoritama objasnit ćemo način na koji mravi konstruiraju rješenje, kako odlažu feromone i koji su dobri parametri algoritma. Također, u ovom poglavlju opisat ćemo sve korake koje je potrebno poduzeti da bismo mogli implementirati ACO algoritme.

U posljednjem poglavlju navodimo eksperimentalne rezultate koji su dobiveni primjenom ACO algoritama na dvije vrste problema trgovačkog putnika. U prvoj vrsti problema gradovi su kružno raspoređeni, a u drugoj vrsti gradovi su slučajno raspoređeni. Također, vidjet ćemo kako različite postavke parametara algoritma utječu na rezultate. Rezultati su dobiveni iz implementacije algoritama u Pythonu.

Rad se temelji na knjizi Marca Doriga i Thomasa Stützlea, Ant Colony Optimization [3].

Poglavlje 1

Optimizacija kolonijom mrava

Optimizacija kolonijom mrava (engl. Ant Colony Optimization, u nastavku koristimo skraćenicu **ACO**) je metaheuristika koja se koristi za rješavanje teških optimizacijskih problema. Kod optimizacije kolonijom mrava, skup umjetnih mrava traži dobro rješenje za dani optimizacijski problem. Surađivanje je osnovna značajka ACO algoritama. Dobro rješenje proizlazi iz međusobnog surađivanja mrava. Za primjenu ovakve optimizacije, optimizacijski problem opisuje se problemom nalaženja najboljeg puta u težinskom grafu. Umjetni mravi postupno grade rješenje, krećući se kroz graf. Proces konstrukcije rješenja je stohastičan i pod utjecajem feromon modela, a to je skup parametara povezanih s komponentama grafa (ili čvorovima, ili bridovima), čije vrijednosti su modificirane prolazima mrava. ACO algoritmi mogu se koristiti za rješavanje statičkih i dinamičkih kombinatornih problema. Karakteristike statičnog problema su definirane samo jednom i ne mijenjaju se za vrijeme njegovog rješavanja. Primjer statičkog problema je TSP. S druge strane, dinamički kombinatorni problemi su definirani kao funkcija nekih veličina, čije vrijednosti mijenja dinamika ishodišnog sistema. Definicija problema mijenja se za vrijeme rješavanja i optimizacijski algoritam mora se moći mijenjati, ovisno promjenjama definicije. Primjer dinamičkog problema je problem rutiranja podataka kroz mrežu.

U ovom poglavlju objasnit ćemo zašto su mravi inspiracija za ovu vrstu optimizacije, dajemo formalnu karakterizaciju klase problema na koje možemo primjeniti ACO metaheuristiku, ponašanja umjetnih mrava i opće strukture ACO metaheuristike.

1.1 Motivacija iz prirode

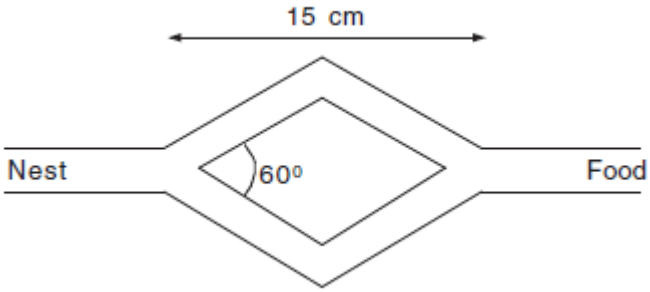
Kolonija mrava predstavlja kompleksnu, moćnu organizaciju i to upravo zbog međusobne suradnje mrava. Zahvaljujući toj kompleksnosti, kolonija može riješiti zadatke koji nadmašuju sposobnosti pojedinog mrava. Sposobnost vida kod određenih

vrsta mrava je djelomično razvijena, a neke vrste su i potpuno slijepe. Dokazano je da se većina komunikacije između mrava ostvaruje ispuštanjem feromona. Kretanjem od mravinjaka prema hrani i natrag, mravi iza sebe ostavljaju trag feromona. Količina feromona koju ostavljaju može ovisiti o kvaliteti i količini hrane. Ostali mravi mogu namirisati feromone i slijediti njihov trag do hrane. Mravi, na putu prema hrani, s većom vjerojatnošću biraju put s većom koncentracijom feromona. Upravo takva komunikacija omogućuje mravima da nađu najkraći put između hrane i mravinjaka. Ovakvo ponašanje mrava istraživao je Deneubourg u eksperimentu dvokrakog mosta.

Eksperiment dvokrakog mosta

U eksperimentu dvokrakog mosta, mravinjak argentinskih mrava povezan je s hranom s dva mosta. Mravi mogu doći do hrane i vratiti se koristeći bilo koji od ta dva mosta. Cilj eksperimenta je bio proučiti ponašanje kolonije. Zaključeno je da, ako oba mosta imaju istu duljinu, mravi konvergiraju prema korištenju jednog od ta dva mosta. Kad je eksperiment ponovljen više puta, pokazalo se da je svaki od dva mosta korišten jednako mnogo puta. Rezultat objašnjava činjenica da mravi, dok se kreću, ispuštaju feromone na tlo, a kad trebaju odlučiti koji put će slijediti, njihov izbor temelji se na feromonima. Ako je veća količina feromona nađenih na tlu, onda je i veća vjerojatnost da će slijediti taj put.

Promotrimo slučaj kada dva mosta imaju jednaku duljinu. Kako mravi konvergiraju prema upotrebi samo jednog mosta možemo bolje razumjeti uz pomoć slike 1.1.

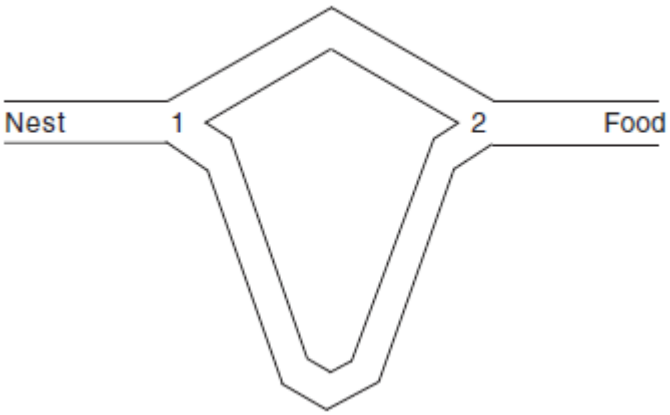


Slika 1.1: Mostovi jednake duljine.

U početku mravi istražuju okolinu mravinjaka. Kada dođu do trenutka odluke koji most će odabrati, biraju s vjerojatnošću temeljenoj na količini feromona koju osjećaju na dva mosta. Prvotno, svaki mrav bira jedan od dva mosta s jednakom vjerojatnošću, zato što još nema feromona. Ipak, nakon nekog vremena, zbog slučajnih varijacija,

jedan od dva mosta sadrži veću količinu feromona pa zbog toga privlači više mrava. Zbog takvog mehanizma, nakon nekog vremena, cijela kolonija konvergira prema samo jednom mostu.

U drugom slučaju, jedan most je bio značajno kraći od drugog (v. sliku 1.2). Jako bitno je za primijetiti da su mravi, koji su odabrali kraći most, bili prvi koji su došli do hrane. Kada su se ti mravi vraćali natrag u mravinjak, došli su do točke odluke 2 na slici 1.2.



Slika 1.2: Mostovi različitih duljina.

U toj točki osjete više feromona na kraćoj grani pa odabiru tu granu s većom vjerojatnošću i opet ta grana dobiva dodatne feromone. Ova činjenica povećava vjerojatnost da sljedeći mravi prije biraju kraću granu od duže.

Također, interesantno je promotriti što se dogodi kad je koloniji, nakon što konvergira na jedan most, ponuđen još jedan most, kraći od prvotnog. Prvo je koloniji ponuđen samo duži most i, nakon 30 minuta, kraći most je dodan. U tom slučaju, kraći most je odabran povremeno i kolonija je zapela na dužem mostu. Ovo se može objasniti većom koncentracijom feromona na dužem mostu. Isparavanje feromona, koje bi moglo pomoći otkrivanju novih puteva, je presporo.

Goss je razvio model opisanog ponašanja: uz pretpostavku da je, u danom trenutku, m_1 mrava odabralo prvi most i m_2 mrava odabralo drugi most, vjerojatnost p_1 da mrav odabere prvi most je

$$p_1 = \frac{(m_1 + k)^h}{(m_1 + k)^h + (m_2 + k)^h},$$

gdje su parametri k i h dobiveni eksperimentalno (očito je $p_2 = 1 - p_1$). Monte Carlo simulacija je pokazala da je dobar odabir $k \approx 20$ i $h \approx 2$. Ova jednadžba je poslužila kao inspiracija za Mravlji sistem, prvi ACO algoritam [2].

1.2 Definicija problema

Umjetni mravi u ACO su stohastički konstrukcijski postupci koji inkrementalno grade rješenje, dodavajući odgovarajuće komponente rješenja na parcijalno rješenje koje se gradi. Iz tog razloga, ACO metaheuristika se može primijeniti na sve kombinatorne optimizacijske probleme za koje se može definirati konstrukcijska heuristika.

Iako ovo znači da se ACO metaheuristika može primijeniti na bilo koji zanimljiv kombinatorni optimizacijski problem, pravi problem je kako preslikati problem kojeg promatramo, u prikaz kojeg umjetni mravi mogu koristiti kod konstrukcije rješenja. Nadalje dajemo formalnu karakterizaciju prikaza kojeg koriste umjetni mravi i strategije koju implementiraju. Definirajmo model problema kombinatorne optimizacije.

Definicija 1.2.1. *Model $\mathcal{P} = (S, f, \Omega)$ problema kombinatorne optimizacije sastoji se od tri dijela:*

- *S je skup kandidata za rješenje,*
- *f je funkcija cilja koja svakom kandidatu $s \in S$ pridružuje vrijednost (cijenu) $f(s,t)$,*
- *$\Omega(t)$ je skup ograničenja.*

Parametar t označava da funkcija cilja i ograničenja mogu ovisiti o vremenu, kao što je primjer u dinamičkim kombinatornim problemima. Cilj je pronaći globalno optimalno dopustivo rješenje s^* minimizacijskog problema, to jest, rješenje najmanje cijene koje zadovoljava zadana ograničenja (dopustivo je).

Kombinatorni optimizacijski problem (S, f, Ω) preslikava se u problem kojeg karakteriziraju sljedeći dijelovi:

- Konačan skup *komponenata* $C = \{c_1, c_2, ..., c_{N_C}\}$, gdje je N_C broj komponenti.
- *Stanja* problema definirana su u terminima nizova $x = \langle c_i, c_j, \ldots, c_h, \ldots \rangle$ konačne duljine, s elementima iz C. Skup svih mogućih stanja označavamo s $\mathcal{X}$. Duljina niza x je broj komponenti u nizu i označavamo ju s $|x|$. Maksimalna duljina niza ograničena je pozitivnom konstantom $n < +\infty$.
- Skup mogućih rješenja $\mathcal{S}$ je podskup od $\mathcal{X}$ ($\mathcal{S} \subseteq \mathcal{X}$).
- Skup dopustivih stanja $\tilde{\mathcal{X}}$, gdje je $\tilde{\mathcal{X}} \subseteq \mathcal{X}$, definiran preko odgovarajućeg testa za pripadni problem, koji provjerava da nije nemoguće nadograditi (dopuniti) niz $x \in \tilde{\mathcal{X}}$ do rješenja koje zadovoljava uvjete Ω. Prema definiciji, dopustivost stanja $x \in \tilde{X}$ trebamo interpretirati u slabom smislu, jer nema garancije da postoji nadogradnja s od x, takva da je $s \in \tilde{\mathcal{X}}$.

- Neprazan skup $\mathcal{S}^*$ optimalnih rješenja, $\mathcal{S}^* \subseteq \tilde{\mathcal{X}}$ i $\mathcal{S}^* \subseteq \mathcal{S}$.
- Cijena $g(s,t)$ je definirana za svakog kandidata za rješenje $s \in \mathcal{S}$. U većini slučajeva $g(s,t) \equiv f(s,t), \forall s \in \tilde{\mathcal{S}}$, gdje je $\tilde{\mathcal{S}} \subseteq \mathcal{S}$ skup dopustivih rješenja, dobivenih iz skupa $\mathcal{S}$ tako da zadovoljavaju skup ograničenja $\Omega(t)$.
- U nekim slučajevima cijena, ili procjena cijene, $J(x,t)$ može biti povezana sa stanjem x različitim od kandidata za rješenje. Ako se stanje x_j može dobiti dodavanjem komponenti rješenja na stanje x_i, tada je $J(x_i,t) \leq J(x_j,t)$. Primijetimo da je $J(s,t) \equiv g(s,t)$.

Prema ovoj formulaciji problema, umjetni mravi grade rješenje izvodeći slučajne šetnje potpuno povezanim grafom $G_C = (C, L)$, čiji čvorovi su komponente C, a skup bridova L potpuno povezuje skup C. Graf G_C zovemo konstrukcijski graf i elemente od L nazivamo veze.

Ograničenja problema $\Omega(t)$ su implementirana u strategiji koju prate umjetni mravi, kao što ćemo objasniti u sljedećoj sekciji. Izbor da se ograničenja implementiraju u konstrukcijskoj strategiji umjetnih mrava omogućuje određen stupanj fleksibilnosti. Ovisno o kombinatornom problemu, ponekad ima više smisla strogo implementirati uvjete, dopuštajući mravima da grade samo dopustiva rješenja. S druge strane, možemo dopustiti da mravi grade i nedopustiva rješenja (iz $\mathcal{S} \setminus \tilde{\mathcal{S}}$), uz penalizaciju prema stupnju nedopustivosti.

1.3 Ponašanje mrava

U ACO algoritmima umjetni mravi su stohastički konstrukcijski postupci, koji grade rješenje krećući se kroz konstrukcijski graf $G_C = (C, L)$, gdje skup L potpuno povezuje komponente C. Ograničenja problema $\Omega(t)$ su ugrađena u mravlju konstrukcijsku heuristiku.

Komponente $c_i \in C$ i veze $l_{ij} \in L$ mogu imati pripadajući trag feromona τ (τ_i ako pripada komponenti, a τ_{ij} ako pripada vezi) i pripadnu heurističku vrijednost η (η_i ako pripada komponenti, a η_{ij} ako pripada vezi). Trag feromona šifrira dugotrajno pamćenje cijelog postupka pretraživanja mrava i ažuriraju ga mravi samostalno.

Suprotno tome, heuristička vrijednost, često nazivana heurističkom informacijom, predstavlja raniju informaciju o samom problemu ili informaciju dobivenu za vrijeme trajanja potrage od nekog drugog izvora, a ne od mrava. U puno slučajeva, η je cijena ili procjena cijene dodavanja nove komponente ili veze u rješenje koje se trenutno gradi. Mrav koristi trag feromona i heurističku informaciju da donese vjerojatnosnu odluku kako će se kretati po grafu.

Svaki mrav k ima sljedeća svojstva:

- Koristi konstrukcijski graf $G_C = (C, L)$ da nađe optimalno rješenje $s^* \in \mathcal{S}^*$.
- Ima memoriju $\mathcal{M}^k$ u koju sprema informacije o putu koji je dosad prošao. Memorija se može koristiti za:
  1. gradnju dopustivih rješenja (tj., za implementaciju ograničenja Ω),
  2. izračunavanje heurističke vrijednosti η,
  3. ocjenjivanje pronađenog rješenja,
  4. vraćenje unatrag po putu.
- Ima početno stanje x_s^k i jedan ili više uvjeta zaustavljanja e^k. Obično je početno stanje prazan niz ili niz jedinične duljine, tj. niz s jednom komponentom.
- U stanju $x_r = \langle x_{r-1}, i \rangle$, ako nije zadovoljen uvjet zaustavljanja, mrav se pomiče u susjedni čvor j iz skupa susjeda kojeg označavamo s $\mathcal{N}^k(x_r)$, to jest, u stanje $\langle x_r, j \rangle \in \mathcal{X}$. Ako je barem jedan od uvjeta zaustavljanja ispunjen, onda mrav staje. Kada mrav gradi kandidata za rješenje, pokreti u nedopustiva stanja su najčešće zabranjeni, ili kroz memoriju mrava, ili kroz odgovarajuće definirane heurističke vrijednosti η.
- Mrav odabire pokret primjenjujući vjerojatnosno pravilo odlučivanja. Vjerojatnosno pravilo odlučivanja je funkcija (1) lokalnih tragova feromona i heurističkih vrijednosti (tj., tragova feromona i heurističkih vrijednosti povezanih s komponentama i vezama u susjedstvu pozicije u kojoj se trenutno mrav nalazi na grafu G_C), (2) memorije mrava koja pohranjuje njegovo trenutno stanje i (3) ograničenja problema.
- Dodavajući komponentu c_j trenutnom stanju, mrav može ažurirati trag feromona τ vezanog za pripadnu komponentu ili pripadnu vezu.
- Jednom kad je mrav sagradio rješenje, on može slijediti isti put unatrag i ažurirati feromone korištenih komponenti.

Važno je primjetiti da se mravi kreću istovremeno i neovisno i, iako je svaki mrav dovoljno kompleksan da nađe (vjerojatno slabo) rješenje problema, dobra rješenja mogu proizaći samo iz kolektivne interakcije među mravima. To se ostaruje indirektnom komunikacijom, određenom informacijama koje mrav čita i zapisuje u trag feromona. Na neki način, ovo je distribuirani proces učenja u kojem mravi nisu sami prilagodljivi, već mijenjaju način na koji je problem reprezentiran i kako ga percipiraju drugi mravi.

1.4 ACO metaheuristika

Neformalno, ACO algoritme možemo zamišljati kao međudjelovanje tri postupka: `KonstruirajMravljeRješenje`, `AžurirajFeromone` i `CentralneAkcije`. Sljedeći pseudokod opisuje ACO metaheuristiku.

Algoritam 1: Pseudokod ACO metaheuristika

```
1 Postavi parametre, inicijaliziraj trag feromona
2 RASPOREDI_AKTIVNOSTI
3     KonstruirajMravljeRješenje
4     AžurirajFeromone
5     CentralneAkcije
6 ZAVRŠI_RASPOREDI_AKTIVNOSTI
```

Glavni postupak `RasporediAktivnosti` raspoređuje tri navedene komponente ACO algoritma. Postupak ne opisuje kako su te tri aktivnosti raspoređene i sinkronizirane. Drugim riječima, ne određuje jesu li aktivnosti izvršene paralelno i neovisno, ili je potrebna neka vrsta sinkronizacije među njima. Autor algoritma je slobodan odabrati kako se te aktivnosti izvršavaju, uzimajući u obzir karakteristike problema.

KonstruirajMravljeRješenja

`KonstruiraMravljeRješenje` izvodi kolonija mrava, koji istovremeno i asinkrono posjećuju susjedna stanja pripadnog problema, krećući se kroz susjedne čvorove konstrukcijskog grafa G_C. Oni se kreću kroz graf, primjenjujući stohastičku lokalnu strategiju odlučivanja, koja koristi trag feromona i heurističku informaciju. Mravi postupno grade rješenje optimizacijskog problema. Kad mrav završi s gradnjom rješenja, ili tijekom izgradnje, on ocjenjuje (djelomično) rješenje, što će se iskoristiti u postupku `AžurirajFeromone` za odluku koliko feromona treba naslagati.

AžurirajFeromone

`AžurirajFeromone` je proces u kojem se mijenja trag feromona. Vrijednost traga se može povećati, ako mrav odloži feromone na komponente i veze koje je koristio, ili se može smanjiti, zbog isparavanja feromona. Odlaganje nove količine feromona povećava vjerojatnost da komponente/veze koje su bile korištene u dobrim rješenjima budu ponovo izabrane kod nadolazećih mrava. Isparavanje feromona implementira oblik zaboravljanja: ono izbjegava preranu konvergenciju algoritma prema lokalno optimalnom rješenju i omogućuje istraživanje novih površina u prostoru pretraživanja.

CentralneAkcije

Postupak `CentralneAkcije` se koristi za implementaciju centralnih akcija koje ne može samostalno izvesti jedan mrav. Primjeri takvih akcija su pokretanje lokalnog optimizacijskog postupka, ili skupljanje globalnih informacija koje se mogu koristiti kod odlučivanja je li korisno odložiti dodatne feromone ili ne. Centralne akcije promatraju puteve svih mrava i odabiru one koji će odložiti dodatne feromone na komponente/veze koje su koristili.

1.5 Primjena ACO metaheuristike na TSP

Problem trgovačkog putnika je problem u kojem trgovački putnik, počevši iz nekog grada, želi pronaći najkraći mogući put kroz zadani skup gradova, tako da svaki grad posjeti točno jednom i završi u gradu iz kojeg je krenuo. Problem se može prikazati potpunim težinskim grafom $G = (N, A)$, gdje je N skup od $n = |N|$ čvorova (gradova), a A je skup bridova (cesta) koje potpuno povezuju graf. Svaki brid $(i, j) \in A$ ima pripadnu težinu d_{ij} koja predstavlja udaljenost između gradova i i j. Problem možemo predstaviti kao problem nalaženja minimalnog Hamiltonovog ciklusa u bridno-težinskom (potpunom) grafu. Hamiltonov ciklus je zatvoreni Hamiltonov put, to jest, put koji prolazi kroz sve čvorove grafa točno jednom. Možemo razlikovati simetrični problem trgovačkog putnika, za kojeg vrijedi $d_{ij} = d_{ji}$, za svaki par vrhova, i asimetrični problem trgovačkog putnika, kod kojeg za bar jedan par vrhova (gradova) i, j vrijedi $d_{ij} \neq d_{ji}$.

Rješenje problema trgovačkog putnika može se prikazati kao permutacija grafova. Ta permutacija je ciklička, to jest, apsolutna pozicija gradova u permutaciji nije bitna, samo je bitan relativni poredak (postoji n permutacija koje prikazuju isto rješenje).

Konstrukcijski graf

Konstrukcijski graf je identičan grafu problema: skup komponenti C odgovara skupu čvorova ($C = N$), skup veza je jednak skupu bridova ($L = A$), i svaka veza ima težinu koja je jednaka udaljenosti d_{ij} između čvorova i i j. Skup stanja problema je skup svih mogućih djelomičnih puteva.

Ograničenja

Jedini uvjet u problemu trgovačkog putnika je da svi gradovi trebaju biti posjećeni i to najviše jednom. Ovaj uvjet se provodi tako da mrav bira sljedeći grad koji će

posjetiti među onima koje još nije posjetio. Drugim riječima, dopustivo susjedstvo $\mathcal{N}_i^k$, mrava k u gradu i, sadrži sve gradove koje taj mrav još nije posjetio.

Trag feromona i heuristika

Trag feromona τ_{ij} u TSP predstavlja želju za posjetom grada j iz grada i. Heuristička informacija η_{ij} je, uglavnom, obrnuto proporcionalna udaljenosti između gradova i i j, a najčešće se uzima upravo $\eta_{ij} = 1/d_{ij}$. Ova heuristička informacija se koristi u većini ACO algoritama za problem trgovačkog putnika.

Konstrukcija rješenja

Svaki mrav je, na početku, postavljen u slučajno odabran grad. U svakom koraku, mrav dodaje jedan, još neposjećen grad u svoj djelomični put. Konstrukcija staje u trenutku kad su svi gradovi posjećeni (što osigurava da su posjećeni točno jednom).

Poglavlje 2

Vrste ACO algoritama

U ovom poglavlju opisat ćemo tri najpoznatija ACO algoritma: *Mravlji sustav*, *MAX–MIN mravlji sustav* i *Sustav mravlje kolonije*. Nakon toga, opisat ćemo sve korake koje je potrebno poduzeti da bi se ti algoritmi implementirali. Također, dajemo jedan primjer implementacije algoritma u programskom jeziku Python. Algoritme i implementacije primjenit ćemo i opisati na problemu trgovačkog putnika.

2.1 Mravlji sustav

U ovom odjeljku predstavljamo algoritam Mravlji sustav (engl. Ant System, u nastavku koristimo skraćenicu AS).

U početku su bile predstavljene tri različite verzije algoritma Mravljeg sustava, *ant-density*, *ant-quantity* i *ant-cycle*. U algoritmima *ant-density* i *ant-quantity*, mravi ažuriraju feromone odmah nakon prijelaza iz jednog grada u susjedni, a u *ant-cycle* algoritmu, mravi ažuriraju feromone tek nakon što su svi mravi završili konstrukciju svog rješenja i količina feromona koje svaki mrav odloži je funkcija kvalitete nađenog puta. Danas, kada spominjemo algoritam Mravljeg sustava, zapravo mislimo na algoritam *ant-cycle*, zato što su ostale verzije algoritma odbačene, jer imaju slabije performanse.

Glavne faze Mravljeg sustava su mravlja konstrukcija rješenja i ažuriranje feromona. No, za početak, opišimo inicijalizaciju feromonskog traga i heurističke informacije.

Inicijalizacija feromonskog traga i heuristike

U algoritmu Mravljeg sustava, dobra heuristika za inicijalizaciju traga feromona je da vrijednost feromona stavimo malo veću od očekivane vrijednosti feromona koju

mrav izbaci u jednoj iteraciji. Približna vrijednost može se dobiti tako da stavimo

$$\forall(i,j), \quad \tau_{ij} = \tau_0 = \frac{m}{C^{nn}},$$

gdje je m broj mrava, a C^{nn} je duljina puta generiranog heuristikom najbližeg susjeda (bilo koji drugi razumni postupak konstruiranja puta bi radio jednako dobro). Razlog zašto smo odabrali ovu vrijednost: ako je inicijalna vrijednost feromona τ_0 preniska, tada je pretraživanje sklono prema prvim putevima koje su mravi generirali, što dovodi do istraživanja malih dijelova prostora pretraživanja. No, ako je inicijalna vrijednost feromona previsoka, tada su mnoge iteracije izgubljene, čekajući da isparavanje feromona dovoljno reducira vrijednosti feromona, tako da feromoni dodani od mrava mogu voditi pretraživanje.

Konstrukcija puta

U Mravljem sustavu mravi istovremeno grade rješenje TSP-a. U početku, mravi su postavljeni u slučano odabran grad. U svakom koraku konstrukcije, mrav k primijenjuje vjerojatnosno pravilo odlučivanja, nazvano *slučajno proporcionalno* pravilo, da odluči koji će grad posjetiti sljedeći. Detaljnije, vjerojatnost s kojom će mrav k, koji je trenutno u gradu i, posjetiti grad j, dana je s

$$p_{ij}^k = \frac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\sum_{l \in \mathcal{N}_i^k} [\tau_{ij}]^\alpha [\eta_{ij}]^\beta}, \quad \text{ako je } j \in \mathcal{N}_i^k, \tag{2.1}$$

gdje je $\eta_{ij} = 1/d_{ij}$ heuristička vrijednost, a α i β su dva parametra koji određuju relativan utjecaj traga feromona i heurističke vrijednosti. $\mathcal{N}_i^k$ su dopustivi susjedi mrava k u gradu i, tj. skup gradova koje mrav k još nije posjetio. Vjerojatnost odabira nekog grada izvan $\mathcal{N}_i^k$ je 0.

Prema vjerojatnosnom pravilu, vjerojatnost izbora brida (i,j) se povećava s povećanjem vrijednosti traga feromona τ_{ij} i heuristike η_{ij}. Uloga parametara α i β je sljedeća. Ako je $\alpha = 0$, najvjerojatnije će biti odabrani najbliži gradovi, što odgovara klasičnom stohastičkom pohlepnom algoritmu (s višestrukim počecima, jer su mravi, na početku, slučajno distribuirani po gradovima). Ako je $\beta = 0$, na izbor utječe jedino vrijednost feromona, bez heuristike. Općenito, to vodi do jako loših rezultata. Posebno, za vrijednosti $\alpha > 1$, to ubrzo dovodi do stagnacije, to jest, do situacije u kojoj svi mravi slijede isti put i konstruiraju isto rješenje, koje je (vrlo često) jako loše.

Svaki mrav k ima memoriju $\mathcal{M}^k$, koja sadrži gradove koje je mrav posjetio, u poretku u kojem ih je posjetio. Memorija se koristi kod definiranja skupa mogućih susjeda $\mathcal{N}_i^k$, u pravilu konstrukcije (2.1). Dodatno, memorija $\mathcal{M}^k$ omogućuju mravu k da izračuna duljinu puta T^k i da zna gdje treba odložiti feromone.

Moguća su dva načina implementacije konstrukcije rješenja: paralelno i sekvencijalno. U paralelnoj implementaciji, u svakom koraku konstrukcije svi mravi se pomiču u susjedni grad, dok u sekvencijalnoj implementaciji mrav gradi kompletan put prije nego sljedeći mrav počinje graditi svoj. U algoritmu AS, oba načina implementacije su ekvivalentna, u smislu da ne utječu značajno na ponašanje algoritma. No, to nije slučaj za sve algoritme, jedan od takvih je Sustav mravlje kolonije.

Ažuriranje feromonskog traga

Nakon što su svi mravi završili s kontrukcijom puta, ažurira se feromonski trag. Prvo se snizi vrijednost feromona na svim bridovima za neki konstantan faktor, a nakon toga dodaju se feromoni na bridove koje su mravi posjetili. Isparavanje feromona se implementira na sljedeći način:

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij}, \quad \forall (i,j) \in L, \tag{2.2}$$

gdje je $0 < \rho \leq 1$ stopa isparavanja feromona. Parametar ρ se koristi da se izbjegne neograničeno povećanje traga feromona i omogućuje algoritmu da zaboravi na loše odluke koje je donio. Zapravo, ako mravi nisu odabrali neki brid, njegova vrijednost feromona se smanjuje eksponencijalno u broju iteracija. Poslije isparavanja, svi mravi ispuštaju feromone na bridove koje su posjetili u svojoj turi:

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k-1}^{m} \Delta\tau_{ij}^{k}, \quad \forall (i,j) \in L, \tag{2.3}$$

gdje je $\Delta\tau_{ij}^{k}$ količina feromona koju mrav k odloži na bridove koje je posjetio. Definira se na sljedeći način:

$$\Delta\tau_{ij}^{k} = \begin{cases} 1/C^{k}, & \text{ako brid } (i,j) \text{ pripada } T^{k}, \\ 0, & \text{inače}, \end{cases} \tag{2.4}$$

gdje je C^{k} duljina ture T^{k} koju je sagradio mrav k, a izračunata je kao suma duljina svih bridova koji pripadaju turi T^{k}. Prema jednadžbi (2.4), što je tura bolja, to ona poprimi više feromona. Općenito, bridovi kojima je prošlo puno mrava i koji su dio kratkih tura, poprime više feromona i time je vjerojatnije da će ih mravi u budućnosti odabrati.

Dobri parametri za Mravlji sustav

Eksperimentalno je dobiveno da su sljedeći parametri dobri za algoritam Mravljeg sustava u slučaju rješavanja problema trgovačkog putnika:

- $\alpha = 1$.
- β između 2 i 5.
- $\rho = 0.5$.
- $m = n$ (n je broj gradova u problemu trgovačkog putnika).
- $\tau_0 = m/C^{nn}$.

Relativna performansa algoritma Mravljeg sustava, ako ju usporedimo s drugim metaheuristikama, se značajno smanji kad se poveća veličina ulaza. Upravo zato se veliki dio istraživanja ACO fokusira na poboljšanje algoritma Mravljeg sustava.

2.2 MAX–MIN mravlji sustav

MAX–MIN mravlji sustav (engl. MAX–MIN Ant System, u nastavku koristimo skraćenicu MMAS) uvodi četiri nove modifikacije u algoritam Mravljeg sustava [4]. Algoritam se fokusira na najbolje nađene ture: samo najbolji mrav u iteraciji ili najbolji mrav do sada može odložiti feromone. Nažalost, takva strategija može dovesti do stagnacije u kojoj svi mravi slijede istu turu, zbog pretjeranog rasta vrijednosti traga feromona na bridovima dobrog, ali ne optimalnog, rješenja. Kako bi se izbjegao taj efekt, druga modifikacija u MMAS je da se ograniče moguće vrijednosti traga feromona na interval $[\tau_{min}, \tau_{max}]$. Treća modifikacija algoritma je da je trag feromona inicijaliziran na gornju granicu traga feromona. Ova modifikacija, zajedno s niskom stopom isparavanja feromona, povećava istraživanje puteva na početku pretraživanja. Konačno, u MMAS, trag feromona se ponovo inicijalizira, ako u sistemu dođe do stagnacije ili nema poboljšanja u određenom broju uzastopnih iteracija.

Ažuriranje feromonskog traga

Nakon što svi mravi završe s konstrukcijom rješenja, feromoni se ažuriraju primjenjivanjem isparavanja, kao i u algoritmu Mravljeg sustava (jednadžba (2.2)), a zatim se odlažu novi feromoni na sljedeći način:

$$\tau_{ij} = \tau_{ij} + \Delta\tau_{ij}^{best}, \tag{2.5}$$

gdje je $\Delta\tau_{ij}^{best} = 1/C^{best}$. Mrav kojem je dopušteno da odloži feromone je ili najbolji mrav dosad, i u tom slučaju je $\Delta\tau_{ij}^{best} = 1/C^{bs}$, ili najbolji u trenutnoj iteraciji, pa je $\Delta\tau_{ij}^{best} = 1/C^{ib}$, gdje je C^{ib} duljina ture najboljeg mrava u iteraciji. U implementacijama MMAS se koriste oba pravila ažuriranja (najbolji dosad ili najbolji u iteraciji)

i to naizmjenično. Očito, izbor relativne frekvencije kojom se ta dva pravila primijenjuju ima utjecaj na to koliko je algoritam pohlepan. Kada ažuriranje feromona uvijek primijenjuje najbolji mrav dosad, tada je pretraživanje ubrzo fokusirano oko T^{bs} ture, a ako feromone ažurira najbolji mrav u iteraciji, tada je broj bridova koji dobiju feromone veći i pretraživanje je manje usmjereno.

Eksperimentalni rezultati pokazuju da je za male TSP probleme možda najbolje koristiti samo ažuriranje feromona od mrava najboljeg u iteraciji. Za velike TSP probleme, sa stotinama gradova, najbolje performanse su dobivene tako da se povećava utjecaj najboljeg dosad. Ovo se može postići, na primjer, postepenim povećanjem frekvencije kojom je najbolja tura dosad T^{bs} odabrana za ažuriranje traga.

Dobri parametri za MMAS

Eksperimentalno je dobiveno da su sljedeći parametri dobri za algoritam MMAS u slučaju rješavanja problema trgovačkog putnika:

- $\alpha = 1$.
- β između 2 i 5.
- $\rho = 0.02$.
- $m = n$ (n je broj gradova u problemu trgovačkog putnika).
- $\tau_0 = 1/\rho C^{nn}$.
- Granice traga feromona su $\tau_{max} = 1/\rho C^{bs}$ i

$$\tau_{min} = \tau_{max} \frac{1 - \sqrt[n]{0.05}}{(avg - 1) \cdot \sqrt[n]{0.05}},$$

  gdje je avg prosječan broj različitih izbora dostupnih mravu u svakom koraku dok konstruira rješenje (za objašnjenje pogledati rad [4]).

Ograničenja vrijednosti feromonskog traga

U MMAS, donja i gornja granica τ_{min} i τ_{max} vrijednosti feromona na bilo kojem bridu su stavljene zato da bi se izbjegla stagnacija pretraživanja. Točnije, ove granice imaju efekt ograničavanja vjerojatnosti p_{ij} odabira grada j, kada se mrav nalazi u gradu i, na interval $[p_{min}, p_{max}]$, gdje je $0 < p_{min} \leq p_{max} \leq 1$. Samo kada mrav k ima samo jedan mogući izbor za sljedeći grad, to jest, kada je $|\mathcal{N}^k| = 1$, imamo da je $p_{min} = p_{max} = 1$.

Pokazat ćemo da je gornja granica traga feromona na svakom bridu ograničena s $1/\rho C^*$, gdje je C^* duljina optimalne ture (Propozicija 2.2.1). Prema tome rezultatu, MMAS koristi procjenu te vrijednosti, $1/\rho C^{bs}$, za definiranje τ_{max}: svaki put kad je nova najbolja tura pronađena, vrijednost τ_{max} se mijenja. Donja granica vrijednosti traga feromona je postavljena na $\tau_{min} = \tau_{max}/a$, gdje je a parametar. Eksperimentalni rezultati sugeriraju da, za izbjegavanje stagnacije, donja granica feromona igra važniju ulogu od τ_{max}. S druge strane, τ_{max} ostaje koristan za postavljanje vrijednosti feromona kod povremenih reinicijalizacija traga.

Propozicija 2.2.1. *U iteraciji θ, za svaki $\tau_{ij}(\theta)$ vrijedi:*

$$\lim_{\theta\to\infty} \tau_{ij}(\theta) \leq \tau_{max} = \frac{q_f(s^*)}{\rho},$$

gdje je $q_f(s)$ neka funkcija kvalitete rješenja s, nerastuća obzirom na f. Često se uzima $q_f(s) = 1/f(s)$ i tada je desna strana jednaka $1/\rho f(s^*)$.

Dokaz. Maksimalna moguća količina feromona koju je moguće dodati nekom bridu (i, j) nakon svake iteracije je $q_f(s^*)$. Očito, u prvoj iteraciji maksimalni trag feromona je $(1-\rho)\tau_0 + q_f(s^*)$, u drugoj iteraciji $(1-\rho)^2\tau_0 + (1-\rho)q_f(s^*) + q_f(s^*)$ i tako dalje. Prema tome, zbog isparavanja feromona, trag feromona u iteraciji θ je ograničen s

$$\tau_{ij}^{max}(\theta) = (1-\rho)^\theta \tau_0 + \sum_{i=1}^{\theta} (1-\rho)^{\theta-i} q_f(s^*).$$

Kako je $0 < \rho \leq 1$, ova suma konvergira prema

$$\tau_{max} = \frac{q_f(s^*)}{\rho}.$$

□

Inicijalizacija traga feromona i reinicijalizacija

Na početku algoritma, početni trag feromona je postavljen na procijenjenu vrijednost gornje granice traga. Ovakvo definiranje traga feromona, u kombinaciji s malom stopom isparavanja feromona, uzrokuje sporo povećanje u relativnoj razlici razina traga feromona, pa je početna faza pretraživanja MMAS jako opširna (pretražuje veliki dio prostora pretraživanja).

Da bi se povećao prostor pretraživanja puteva koji imaju malu vjerojatnost da budu odabrani, trag feromona se ponekad ponovo inicijalizira. Trag feromona se

ponovo inicijalizira kada algoritam dosegne fazu stagnacije (prema nekim statistikama traga feromona) ili ako za neki zadani broj iteracija algoritam nije popravio najbolju nađenu turu.

MMAS je jedan od najistraživanijih ACO algoritama i postoje njegova mnoga proširenja. Jedno od proširenja je, da pravilo ažuriranja traga feromona ponekad koristi najbolju nađenu turu od zadnje reinicijalizacije traga feromona, umjesto najbolje nađene ture dosad. Druga varijanta koristi isto pseudoslučajno proporcionalno pravilo izbora, kao u algoritmu Sustav mravlje kolonije.

2.3 Sustav mravlje kolonije

Sustav mravlje kolonije (engl. Ant Colony System, ili skraćeno ACS) razlikuje se od algoritma Mravljeg sustava u 3 točke. Prvo, ovaj algoritam više iskorištava samo iskustvo pretraživanja mrava i to tako da koristi jače, agresivnije pravilo odabira. Drugo, isparavanje feromona i odlaganje feromona događa se samo na najboljoj turi dosad. Treće, svaki put kad mrav koristi brid (i, j) da se makne iz grada i u grad j, on izbriše dio feromona na tom bridu, da poveća istraživanje drugih puteva.

Konstrukcija puta

U ACS, kada se mrav k nalazi u gradu i i želi se pomaknuti u grad j, on odabire grad primijenjujući pseudoslučajno proporcionalno pravilo, dano na sljedeći način

$$j = \begin{cases} \arg\max_{l \in \mathcal{N}_i^k} \{\tau_{il}[\eta_{il}]^\beta\}, & \text{ako je } q \leq q_0, \\ J, & \text{inače,} \end{cases} \tag{2.6}$$

gdje je q slučajna varijabla uniformno distribuirana na $[0, 1]$, q_0 $(0 \leq q_0 \leq 1)$ je parametar, i J je slučajna varijabla odabrana prema vjerojatnosnoj distribuciji danoj formulom (2.1) (uz $\alpha = 1$).

Drugim riječima, mrav s vjerojatnošću q_0 napravi najbolji mogući potez, prema postojećem tragu feromona i heurističkoj informaciji, dok s vjerojatnošću $(1-q_0)$ radi vođeno istraživanje bridova. Optimiziranje parametra q_0 dozvoljava modificiranje stupnja istraživanja i odabira hoće li se pretraživanje fokusirati na najbolje nađeno rješenje dosad ili na istraživanje novih puteva.

Globalno ažuriranje traga feromona

U ACS samo jedan mrav (najbolji mrav dosad) može dodati feromone nakon svake iteracije. Ažuriranje feromona u ACS je implementirano sljedećom jednadžbom:

$$\tau_{ij} \leftarrow (1 - \rho)\tau_{ij} + \rho\Delta\tau_{ij}^{bs}, \quad \forall (i, j) \in T^{bs}, \tag{2.7}$$

gdje je $\Delta\tau_{ij}^{bs} = 1/C^{bs}$. Najvažnije je primijetiti da se ažuriranje traga feromona u ACS, pritom mislimo na isparavanje feromona i odlaganje novih feromona, odnosi samo na bridove iz T^{bs}, a ne na sve bridove, kao u AS. Ovo je važno zato što smanjuje složenost izračunavanja ažuriranja feromona za svaku iteraciju s $\mathcal{O}(n^2)$ na $\mathcal{O}(n)$, gdje je n veličina instance problema koji se rješava. Kao i inače, parametar ρ označava stopu isparavanja feromona. Za razliku od (2.2) i (2.3) kod AS, u pravilu (2.7) je odloženi feromon smanjen za faktor ρ, tako da je novi trag feromona težinski prosjek između stare vrijednosti feromona i količine odloženih feromona.

U početnim eksperimentima, bilo je razmatrano i korištenje najbolje ture u iteraciji za ažuriranje feromona. Za male TSP instance, minimalna je razlika u rezultatima ako se koristi najbolja tura dosad ili najbolja tura u iteraciji. Za instance problema s više od 100 gradova, korištenje najbolje ture dosad dalo je bitno bolje rezultate.

Lokalno ažuriranje traga feromona

Uz globalno ažuriranje traga feromona, u ovom algoritmu mravi koriste i lokalno ažuriranje feromona, i to odmah nakon što mrav prijeđe brid (i, j) tijekom konstrukcije ture:

$$\tau_{ij} \leftarrow (1 - \xi)\tau_{ij} + \xi\tau_0, \tag{2.8}$$

gdje su ξ, $0 \leq \xi \leq 1$, i τ_0 dva parametra. Vrijednost τ_0 je postavljena da bude ista kao početna vrijednost traga feromona. Eksperimentalno je dobiveno da je dobra vrijednost za parametar ξ jednaka 0.1, a dobra vrijednost za parametar τ_0 je $1/nC^{nn}$, gdje je n broj gradova u instanci TSP problema i C^{nn} je duljina ture najbližih susjeda. Posljedica lokalnog ažuriranja feromona je da svaki put kad mrav koristi brid (i, j), trag feromona tog brida τ_{ij} je smanjen pa taj brid postaje manje poželjan da ga slijede mravi koji dolaze nakon njega. Drugim rječima, ovo omogućuje da se poveća pretraživanje bridova koji još nisu bili posjećeni i, u praksi, ima efekt da algoritam ne dolazi u fazu stagnacije (to jest, mravi ne konvergiraju tome da generiraju iste puteve). Važno je primijetiti da za prijašnje varijante algoritma Mravljeg sustava nije bilo bitno ako mravi konstruiraju turu paralelno ili sekvencijalno. To se mijenja u algoritmu Sustava mravlje kolonije, zbog lokalnog ažuriranja feromona.

Dobri parametri za ACS

Eksperimentalno je dobiveno da su sljedeći parametri dobri za algoritam ACS u slučaju rješavanja problema trgovačkog putnika:

- α parametar nema.
- β između 2 i 5.

- $\rho = 0.1$.
- $m = 10$.
- $\tau_0 = 1/nC^{nn}$ (n je broj gradova u problemu trgovačkog putnika).
- U lokalnom ažuriranju feromona: $\xi = 0.1$. U pseudoslučajnom proporcionalnom pravilu odabira je $q_0 = 0.9$.

2.4 Implementacija ACO algoritama

U ovom poglavlju opisat ćemo detaljne korake koje je potrebno napraviti da bi se implementirao ACO algoritam za problem trgovačkog putnika. Zato što su sve vrste ACO algoritama jako slične, uglavnom se fokusiramo na implementaciju algoritma AS, a gdje je potrebno, istaknut ćemo koje je promjene potrebno napraviti za druge algoritme.

Strukture podataka

Strukture podataka trebaju sadržavati podatke koji opisuju instancu problema trgovačkog putnika, podatke o tragu feromona i strukturu koja predstavlja mrave. Opći opis glavnih struktura možemo vidjeti u kodu 2.1.

```
integer dist[n][n]  #matrica udaljenosti
integer nn_list[n][n]   #matrica sa listama najblizih susjeda
real pheromone[n][n] #matrica vrijednosti feromona
real choice_info[n][n] #spajanje vrijednosti feromona i heuristike
structure single_ant
begin
  integer tour_length #duljina ture
  integer tour[n+1] #memorija mrava (cuva djelomican i konacan put)
  integer visited[n] #posjeceni gradovi
end

single_ant ant[m] #lista mrava
```

Kod 2.1: Glavne strukture podataka za implementaciju ACO algoritma.

Detaljan opis glavnih struktura:

- **Matrica udaljenosti**. Matrica udaljenosti *dist* je simetrična matrica dimenzija $n \times n$, gdje je n broj gradova u problemu trgovačkog putnika. Ona čuva udaljenosti između gradova.

- **Lista najbližih gradova**. Lista najbližih gradova, $nn\_list[i][k] = j$, daje nam indeks k-tog najbližeg grada gradu i.
- **Trag feromona**. U matrici dimenzija $n \times n$ se nalazi vrijednost traga feromona τ_{ij} koja pripada bridu (i, j), to jest, vezi između gradova i i j. U početku je vrijednost matrice traga feromona jednaka m/C^{nn}, gdje je m broj mrava, a C^{nn} je duljina puta generiranog heuristikom najbližeg susjeda.
- **Spajanje vrijednosti feromona i heuristike**. Kada se mrav kreće iz grada i u grad j, on odabire grad s vjerojatnošću koja je proporcionalna vrijednosti $[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}$. Zato što ove vrijednosti treba izračunati svaki od m mrava, vrijeme izračunavanja se može značajno smanjiti ako ove vrijednosti čuvamo u simetričnoj matrici *choice_info*, gdje $choice\_info[i][j]$ sadrži vrijednost $[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}$.
- **Struktura mrav**. Struktura mrav treba čuvati duljinu ture koju mrav generira, mora moći spremati gradove koje je mrav posjetio i, također, treba imati memoriju u koju sprema djelomičan ili konačan put.

Osnovni pregled algoritma

Glavni zadaci u ACO algoritmu su konstrukcija rješenja, ažuriranje traga feromona i neke dodatne tehnike, poput lokalne optimizacije. Također, treba inicijalizirati podatke i parametre algoritma i treba voditi statistiku o radu algoritma.

```
1 procedure ACOforTSP
2   InitializeData
3   while (not terminate) do
4     ConstructSolutions
5     LocalSearch
6     UpdateStatistics
7     UpdatePheromoneTrail
8   end-while
9 end-procedure
```

Kod 2.2: Osnovni pregled ACO algoritma za TSP.

Inicijalizacija podataka

U inicijalizaciji podataka potrebni su sljedeći koraci:

- Potrebno je učitati instancu problema trgovačkog putnika.
- Izračunati matricu udaljenosti između gradova.

- Izračunati matricu najbližih susjeda za sve gradove.
- Potrebno je inicijalizirati matricu feromona i *choice_info* matricu.
- Zadati parametre algoritma.
- Inicijalizirati varijable koje sadrže informacije o nekim statistikama, na primjer, broju iteracija, najboljem rješenju dosad, o korištenom vremenu procesora, itd.

```
1 procedure InitializeData
2   ReadInstance
3   ComputeDistances
4   ComputeNearestNeighborList
5   ComputeChoiceInformation
6   InitializeAnts
7   InitializeParameters
8   InitializeStatistics
9 end-procedure
```

Kod 2.3: Inicijalizacija podataka.

Uvjet zaustavljanja

Program staje ako je barem jedan uvjet zaustavljanja zadovoljen. Mogući uvjeti zaustavljanja su: (1) algoritam je našao rješenje unutar zadane granice, dovoljno blizu optimalnog rješenja, (2) dosegnut je maksimalan broj konstrukcija rješenja ili maksimalan broj iteracija algoritma, (3) potrošeno je maksimalno vrijeme procesora, ili (4) algoritam stagnira.

Konstrukcija rješenja

Konstrukcija rješenja ima sljedeće djelove:

- Na početku je potrebno isprazniti memoriju mrava, tako da sve gradove označimo kao neposjećene. Ovaj dio vidimo između linija 2 i 6 u kodu 2.4.
- Sve mrave trebamo postaviti u početne gradove. Jedna od mogućnosti je da svakom mravu dodijelimo slučajno odabran grad. To radi dio između linija 7 i 12 u kodu 2.4.
- Svaki mrav konstruira kompletan put. U svakom koraku konstrukcije mrav primijenjuje AS pravilo odabira. Metoda `ASDecisionRule` implementira pravilo odabira, a kao parametre prima redni broj mrava (k) i trenutni indeks koraka konstrukcije rješenja.

- Na kraju, mrav se kreće natrag u početni grad i računa se duljina puta svakog mrava.

Ovi koraci se ponavljaju tako dugo dok mravi ne završe cijelu turu. Zato što svaki mrav mora posjetiti točno n gradova, svi mravi završe konstrukciju rješenja nakon jednakog broja koraka konstrukcije.

```
1  procedure ConstructSolution
2    for k=1 to m do
3      for i=1 to n do
4        ant[k].visited[i] = false
5      end-for
6    end-for
7    step=1
8    for k=1 to m do
9      r = random{1,..., n}
10     ant[k].tour[step] = r
11     ant[k].visited[r] = true
12   end-for
13   while(step<n) do
14     step = step + 1
15     for k=1 to m do
16       ASDesicionRule(k,step)
17     end-for
18   end-while
19   for k=1 to m do
20     ant[k].tour[n+1] = ant[k].tour[1]
21     ant[k].tour_length = ComputeTourLength(k)
22   end-for
23 end-procedure
```

Kod 2.4: Pseudokod metode KonstruirajRješenje.

AS pravilo odlučivanja

U pravilu odlučivanja, mrav koji se nalazi u gradu i, vjerojatnosno odabire da će se pomaknuti u neposjećeni grad j. Odabir ovisi o tragu feromona i vrijednosti heuristike. Metoda za pravilo odlučivanja radi sljedeće: prvo se određuje trenutni grad c mrava k. Vjerojatnosni izbor sljedećeg grada funkcionira analogno postupku odabira zvanom "kolo sreće" u evolucijskom računanju: svaka vrijednost $choice\_info[c][j]$, za grad j kojeg mrav k još nije posjetio, određuje jedan dio "kola sreće", a veličina dijela je proporcionalna težini pripadnog izbora. Tada se kolo zavrti i grad na koji je pokazalo je sljedeći grad za mrava k. Ovo se implementira tako da:

- Sumiramo težine različitih izbora u varijablu *sum_probabilities*.

- Pozivom funkcije *random* izvlačimo uniformno distribuiran slučajan broj r iz intervala $[0, sum\_probabilities]$.
- Prolazimo redom kroz dozvoljene izbore, sve dok suma nije veća ili jednaka r.

Na kraju, mrav se pomiče u odabrani grad i označimo da je taj grad posjećen.

```
procedure ASDecisionRule(k,i)
  input k # redni broj mrava
  input i # broj iteracije
  c = ant[k].tour[i-1]
  sum_probabilities = 0.0
  for j=1 to n do
    if ant[k].visited[j] then
      selection_probability[j] = 0.0
    else
      selection_probability[j] = choice_info[c][j]
      sum_probabilities = sum_probabilities + selection_probability[j]
    end-if
  end-for
  r = random[0, sum_probabilities]
  j = 1
  p = selection_probability[j]
  while(p < r) do
    j = j + 1
    p = p + selection_probability[j]
  end-while
  ant[k].tour[i] = j
  ant[k].visited[j] = true
end-procedure
```

Kod 2.5: Pseudokod za pravilo odlučivanja.

AS pravilo odlučivanja s listom kandidata

Kada se koristi lista kandidata, postupak `ASDecisionRule` se treba prilagoditi. Prva promjena je da kad biramo sljedeći grad, mora se naći prikladan grad iz liste kandidata za trenutni grad c. Sljedeća promjena je potrebna kako bismo riješili situaciju ako su svi gradovi posjećeni. U tom slučaju varijabla *sum_probabilities* zadržava inicijalnu vrijednost 0.0 i jedan grad koji nije u listi kandidata je odabran: postupak `ChooseBestNext` se koristi kako bi se odabrao grad s maksimalnom vrijednosti $[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}$.

```
procedure ASDecisionRule(k,i)
  input k # redni broj mrava
```

```
3   input i # broj iteracije
4   c = ant[k].tour[i-1]
5   sum_probabilities = 0.0
6   for j=1 to n do
7     if ant[k].visited[nn_list[c][j]] then
8       selection_probability[j] = 0.0
9     else
10      selection_probability[j] = choice_info[c][nn_list[c][j]]
11      sum_probabilities = sum_probabilities + selection_probability[j]
12    end-if
13  end-for
14  if(sum_probabilities = 0.0) then
15    ChooseBestNext(k, i)
16  else
17    r = random[0, sum_probabilities]
18    j = 1
19    p = selection_probability[j]
20    while(p < r) do
21      j = j + 1
22      p = p + selection_probability[j]
23    end-while
24    ant[k].tour[i] = nn_list[c][j]
25    ant[k].visited[nn_list[c][j]] = true
26  end-if
27 end-procedure
```

Kod 2.6: Pseudokod za pravilo odlučivanja s listom kandidata.

```
1  procedure ChooseBestNext(k,i)
2    input k # redni broj mrava
3    input i # broj iteracije
4    v = 0.0
5    c = ant[k].tour[i-1]
6    for j=1 to n do
7      if not ant[k].visited[j] then
8        if choice_info[c][j] > v then
9          nc = j
10         v = choice_info[c][j]
11       end-if
12     end-if
13   end-for
14   ant[k].tour[i] = nc
15   ant[k].visited[nc] = true
16 end-procedure
```

Kod 2.7: Pseudokod za metodu OdaberiSljedećegNajboljeg.

Ažuriranje feromona

Ovo je posljednji korak u iteraciji AS algoritma. Ovaj postupak se sastoji od dva dijela: isparavanja feromona i odlaganja feromona. U postupku isparavanja feromona, vrijednost traga feromona se smanji na svim bridovima (i, j) za konstantni faktor ρ. Postupak odlaganja feromona dodaje feromone na bridove koji pripadaju turama koje su konstruirali mravi. Dodatno, ažuriranje feromona sadrži i postupak `ComputeChoiceInformation` koji računa matricu *choice_info*, koja se zatim koristi u sljedećoj iteraciji algoritma.

```
procedure ASPheromoneUpdate
  Evaporate
  for k=1 to m do
    DepositPheromone(k)
  end-for
  ComputeChoiceInformation
end-procedure
```

Kod 2.8: Pseudokod za AžuriranjeFeromona.

```
procedure Evaporate
  for i=1 to n do
    for j=i to n do
      pheromone[i][j] = (1-rho)*pheromone[i][j]
      pheromone[j][i] = pheromone[i][j]
    end-for
  end-for
end-procedure
```

Kod 2.9: Pseudokod za IsparavanjeFeromona.

```
procedure DepositPheromone(k)
  input k #redni broj mrava
  delta_tau = 1/ant[k].tour_length
  for i=1 to n do
    j = ant[k].tour[i]
    l = ant[k].tour[i+1]
    pheromone[j][l] = pheromone[j][l] + delta_tau
    pheromone[l][j] = pheromone[j][l]
  end-for
end-procedure
```

Kod 2.10: Pseudokod za OdlaganjeFeromona.

Promjene potrebne za implementaciju ostalih ACO algoritama

Kada implementiramo ostale verzije ACO algoritama, većina postupaka koje smo opisali ostaje netaknuta. Neke od potrebnih promjena su sljedeće:

- MMAS treba kontrolirati jesu li vrijednosti traga feromona unutar granica. Najbolji način da se to učini da se granice dodaju u postupak `ASPheromoneUpdate`.
- Neke varijante AS algoritma zahtijevaju manje promjene u kontroli pretraživanja prostora. U MMAS, trag feromona se ponovo inicijalizira kada algoritam dosegne fazu stagnacije (prema nekim statistikama traga feromona) ili ako kroz neki zadani broj iteracija algoritam nije popravio najbolju nađenu turu. Također, u MMAS se mogu koristiti razna pravila ažuriranja (najbolji do tada, najbolji u iteraciji).
- U ACS, implementacija pseudoslučajnog proporcionalnog pravila odabira zahtijeva generiranje slučajnog broja q, uniformno distribuiranog u intervalu $[0, 1]$. Ako je $q < q_0$, koristimo postupak `ChooseBestNext`, inače koristimo postupak `ASDecisionRule`.
- Lokalno ažuriranje feromona provodi se postupkom `ACSLocalPheromoneUpdate` (opisanim u kodu 2.11) i taj postupak pozivamo odmah nakon što se mrav pomakne u novi grad.
- Globalno ažuriranje feromona implementira se slično kao i lokalno ažuriranje, no ono se poziva nakon što su svi mravi završili s konstrukcijom rješenja i trag feromona se ažurira samo na bridovima koji pripadaju najboljoj turi dosad.

```
1  procedure ACSLocalPheromoneUpdate(k, i)
2    input k #redni broj mrava
3    input i #korak konstrukcije
4    h = ant[k].tour[i-1]
5    j = ant[k].tour[i]
6    pheromone[h][j] = (1-xi)pheromone[h][j] + xi*tau_0
7    pheromone[j][h] = pheromone[h][j]
8    pheromone[h][j] = pheromone[h][j]*exp(1/dist[h][j], beta)
9    choice_info[j][h] = choice_info[h][j]
10 end-procedure
```

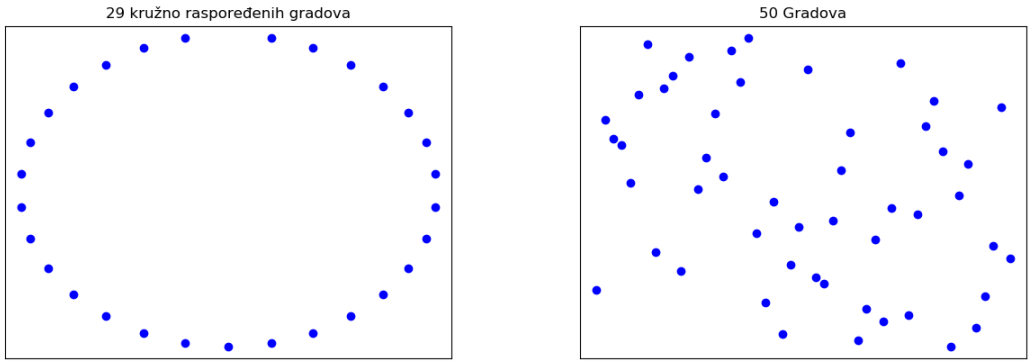
Kod 2.11: Pseudokod za LokalnoAžuriranjeFeromona u algoritmu ACS.

Poglavlje 3

Eksperimentalni rezultati

3.1 Rezultati algoritama nad kružno i slučajno raspoređenim gradovima

Algoritmi su implementirani u programskom jeziku Python. Testirani su na podacima s 29 kružno raporeñenih gradova i na slučajno generiranim podacimo za 50 gradova. Cilj ovog poglavlja je pokazati kakva rješenja nalaze algoritmi, koji su uspješniji u rješavanju i kako njihova uspješnost ovisi o sljedećim parametrima: broju mrava, broju iteracija, te parametrima α, β i ρ.



(a) 29 kružno raspoređenih gradova. (b) 50 gradova.

Slika 3.1: Skice testnih primjera.

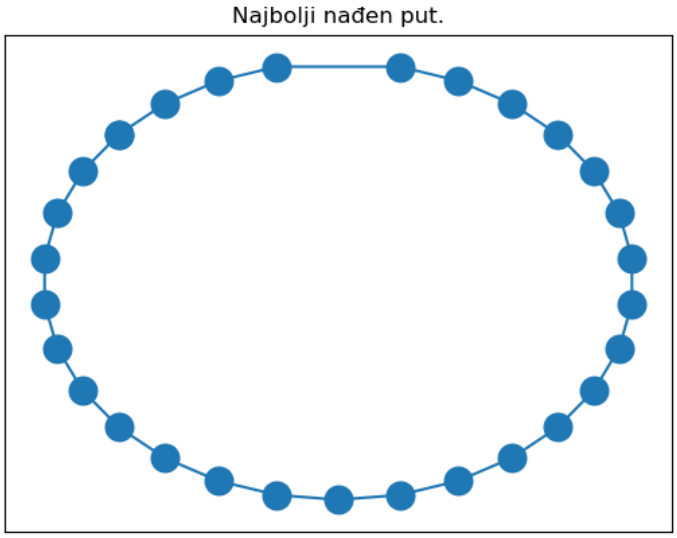
Na slikama 3.1 prikazane su skice testnih podataka. Ove skice predstavljaju problem koji je dan mravima da ga riješe.

Za početak, pokrenimo algoritme za parametre koji su se pokazali najboljima prema knjizi [3]. Uvjet zaustavljanja algoritma bilo je 1500 iteracija i svaki algoritam smo pokrenuli 5 puta. U tablici 3.1 možemo vidjeti prosjek najboljih rezultata za 5 pokretanja algoritma za slučaj kružno raspoređenih i slučajno raspoređenih gradova. U oba slučaja najbolje rezultate daje algoritam MMAS, a najgore ACS.

ACO algoritam	α	β	ρ	m	29 gradova	50 gradova
AS	1	2 do 5	0.5	n	188.954	920.699
MMAS	1	2 do 5	0.02	n	141.074	598.445
ACS	-	2 do 5	0.1	10	314.628	932.590

Tablica 3.1: Prosjek najboljih rezultata u 5 pokretanja algoritma.

Sljedeće što želimo testirati je da li svi algoritmi pronalaze optimalno rješenje. Za slučaj kružno postavljenih gradova, očito je koje je rješenje optimalno. Algoritme smo pokrenuli za 1080 kombinacija parametara, to jest za sve kombinacije gdje je $\alpha \in \{0, \ldots, 5\}$, $\beta \in \{0, \ldots, 5\}$, $\rho \in \{0.1, 0.2, 0.5, 0.6, 0.7, 0.9\}$ i broj mrava $m \in \{1, 3, 5, 10, 20\}$. Za sve algoritme, duljina najkraće nađene ture, za bar jednu kombinaciju parametra, je 62.694. Pretpostavljamo da je tura te duljine optimalna.



Slika 3.2: Najbolji put u slučaju 29 kružno raspoređenih gradova.

U tablici 3.2 navedene su neke vrijednosti parametara za koje su algoritmi, u slučaju kružno raspoređenih gradova, dali optimalno rješenje.

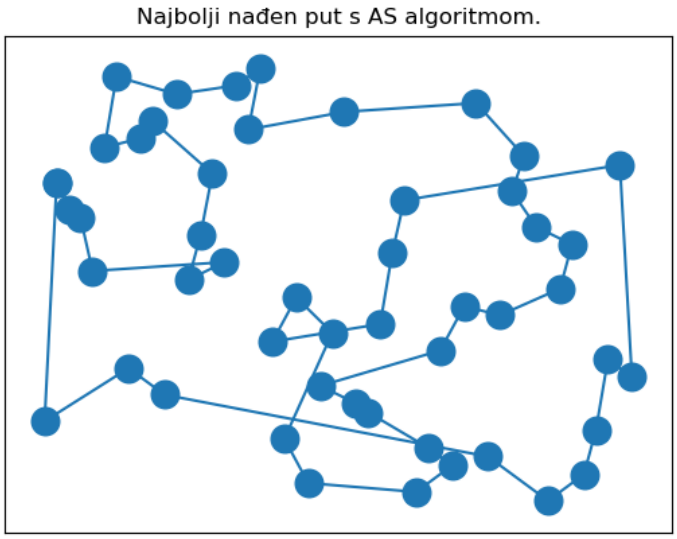
ACO algoritam	α	β	ρ	m
AS	2	5	0.2	5
AS	4	5	0.7	3
AS	5	5	0.9	5
MMAS	2	4	0.1	20
MMAS	2	5	0.5	5
MMAS	4	5	0.7	3
ACS	5	5	0.5	1
ACS	4	3	0.7	1
ACS	1	4	0.1	1

Tablica 3.2: Neke kombinacije parametara za koje algoritmi daju najbolje rješenje.

U slučaju 50 slučajno raspoređenih gradova, algoritmi ne nalaze isto najbolje rješenje za različite kombinacije parametara. Najbolje rješenje pronašao je algoritam AS, a najgore MMAS. Razlika između duljina najboljih puteva je jako mala pa možemo pretpostaviti da, iako ovo rješenje najvjerojatnije nije optimalno, ono je ipak jako dobro. U tablici 3.3 možemo vidjeti parametre za koje su algoritmi postigli najbolje rješenje, a na slici 3.3 prikaz najboljeg puta kojeg je pronašao algoritam AS.

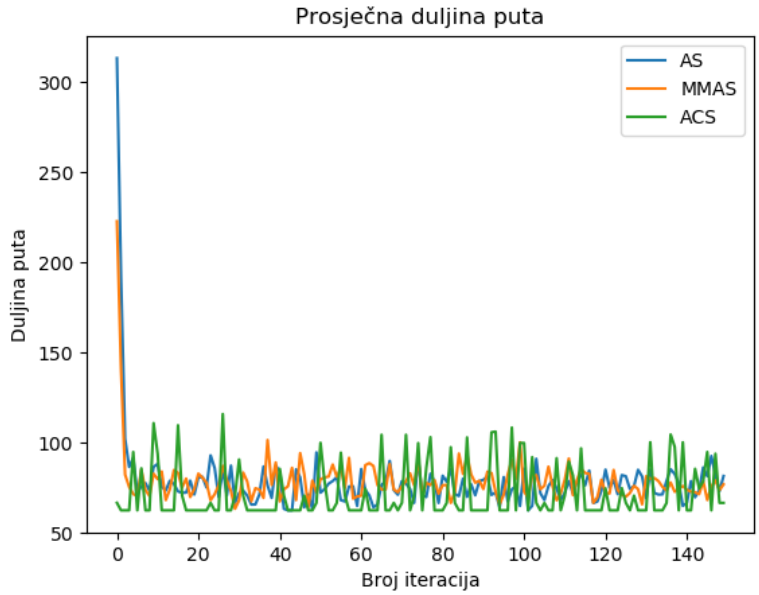
ACO algoritam	α	β	ρ	m	Duljina puta
AS	5	5	0.7	10	339.487
MMAS	2	5	0.2	20	345.775
ACS	2	3	0.9	1	342.004

Tablica 3.3: Parametri najboljih rezultata za slučaj 50 slučajno raspoređenih gradova.

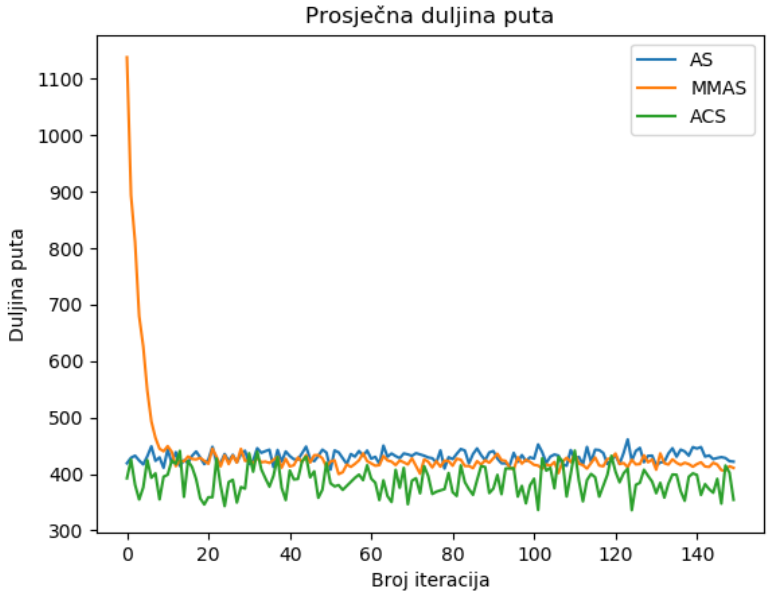


Slika 3.3: Najbolji put u slučaju 50 slučajno raspoređenih gradova.

Na slikama 3.4 i 3.5 prikazano je kako se kreće prosječna duljina puteva po iteracijama. Očekivano je da se prosječna duljina tijekom vremena smanjuje i da se kreće blizu vrijednosti najboljeg rješenja. Upravo to se može vidjeti na slikama 3.4 i 3.5. Parametri algoritma postavljeni su na vrijednosti koje su navedene u tablici 3.2 i tablici 3.3, to jest, na one parametre za koje se pokazalo da daju najbolje rezultate.



Slika 3.4: Prosječna duljina puteva kroz iteracije za 29 kružno raspoređenih gradova.



Slika 3.5: Prosječna duljina puteva kroz iteracije za 50 slučajno raspoređenih gradova.

3.2 Ovisnost o parametrima

Parametar α označava važnost feromonskog traga u algoritmu. Kod analize parametra α, algoritam smo ponovo pokrenuli za dva slučaja navedena u prethodnom poglavlju. Parametri β, ρ i m su konstantni ($\beta = 1$, $\rho = 0.6$, i $m = 5$). Iz tablice 3.4 možemo vidjeti da za algoritme AS i MMAS, za oba slučaja, povećanjem vrijednosti parametra α dobivamo sve bolje rezultate.

α	β	ρ	m	AS	MMAS	ACS	AS50	MMAS50	ACS50
0	1	0.6	5	333	318.415	156.3682	1155.9196	1241.4730	621.386
1	1	0.6	5	249	214.432	206.2208	918.7079	926.3780	658.302
2	1	0.6	5	230	247.395	169.9718	902.2239	895.6477	623.824
3	1	0.6	5	217	218.101	194.2293	845.8509	957.4234	627.064
4	1	0.6	5	227	250.780	193.5186	904.9570	926.5360	604.321

Tablica 3.4: Ovisnost o parametru α.

Parametar β označava jakost heuristike u algoritmu. Ponovo smo pokrenuli algoritam za iste slučajeve. Konstantni parametri su α, ρ i m ($\alpha = 1$, $\rho = 0.6$ i $m = 5$). U rezultatima ove analize (tablica 3.5) možemo vidjeti da parametar β znatno utječe na dobivene rezultate svih algoritama. Također, možemo primijetiti da je važnost parametra β veća od važnosti parametra α.

α	β	ρ	m	AS	MMAS	ACS	AS50	MMAS50	ACS50
1	0	0.6	5	343.814	369.864	271.668	1313.105	1257.228	1090.780
1	1	0.6	5	248.570	214.432	206.220	918.707	926.378	658.302
1	2	0.6	5	148.393	126.739	194.024	632.166	683.940	595.227
1	3	0.6	5	99.187	121.845	214.272	619.166	509.938	668.913
1	4	0.6	5	125.240	114.585	187.064	652.116	534.325	781.262
1	5	0.6	5	104.693	92.918	224.132	719.649	694.493	847.990

Tablica 3.5: Ovisnost o parametru β.

Parametar ρ označava faktor isparavanja feromonskog traga. On povećava utjecaj heuristike i smanjuje utjecaj feromonskog traga, bez da se mijenjaju parametri α i β. Iz rezultata analize, koji se nalaze u tablici 3.6, vidimo da faktor isparavanja ne utječe na rezultate istom jačinom kao što to čine parametri α i β.

α	β	ρ	m	AS	MMAS	ACS	AS50	MMAS50	ACS50
1	1	0.1	5	242.366	217.486	196.829	919.482	838.229	629.965
1	1	0.2	5	235.186	242.924	192.309	922.905	915.180	622.732
1	1	0.5	5	205.565	238.516	185.357	908.033	991.299	612.585
1	1	0.6	5	248.570	214.432	206.220	918.707	926.378	658.302
1	1	0.7	5	267.344	236.142	190.367	929.191	888.425	570.473
1	1	0.9	5	239.411	247.996	174.522	964.112	909.953	603.629

Tablica 3.6: Ovisnost o parametru ρ.

Parametar m označava broj mrava u algoritmu. Iz tablice 3.7 možemo vidjeti da je ovisnost o broju mrava velika u algoritmima MMAS i ACS. Za algoritam MMAS, povećanjem broja mrava se povećava i efikasnost algoritma, dok za algoritam ACS, smanjenjem broja mrava dobivamo bolje rješenje.

α	β	ρ	m	AS	MMAS	ACS	AS50	MMAS50	ACS50
1	1	0.6	1	273.569	314.586	86.984	918.838	1039.410	408.429
1	1	0.6	3	235.562	247.346	103.593	866.246	784.007	505.776
1	1	0.6	5	248.570	214.432	206.220	918.707	926.378	658.302
1	1	0.6	10	220.189	233.080	261.839	929.653	933.103	739.322
1	1	0.6	20	199.771	198.984	282.853	908.104	834.036	853.285

Tablica 3.7: Ovisnost o parametru m.

Poglavlje 4

Kod implementacije ACO algoritama u Pythonu

Implementacije napisana po uzoru na pseudokod iz knjige [3] i uz pomoć koda sa stranice aco-metaheuristic [1]

```
from Ant import Ant
from utils import *

class ACO:
    def __init__(self, number_of_cities, number_of_ants, coordinates, alpha, beta, rho, as_flag, mmas_flag, acs_flag):
        self.number_of_cities = number_of_cities
        self.number_of_ants = number_of_ants
        self.ant = [Ant(number_of_cities) for i in range(number_of_ants)]
        self.coordinates = coordinates
        self.distance_matrix = calculate_distance_matrix(coordinates, number_of_cities)
        self.nn_list = nearest_neighbor_list(number_of_cities, self.distance_matrix)
        self.pheromone_trail = initialize_pheromone_trail(number_of_ants, number_of_cities, self.distance_matrix, self.nn_list)
        self.heuristic = initialize_heuristic(number_of_cities, self.distance_matrix)
        self.choice_info = initialize_choice_info_matrix(number_of_cities, self.pheromone_trail, self.heuristic, alpha, beta)
        self.alpha = alpha
        self.beta = beta
        self.rho = rho
        self.average_tour_length = []
        self.as_flag = as_flag
        self.mmas_flag = mmas_flag
```

```
21          self.acs_flag = acs_flag
22          self.trail_0 = -1
23          self.trail_max = -1
24          self.trail_min = -1
25          self.best_so_far_ant = Ant(self.number_of_cities)
26          self.q_0 = 0.9
27
28      def InitializePheromoneTrail(self):
29          if self.as_flag:
30              self.trail_0 = self.number_of_ants / (
    nearest_neighbor_heuristic(self.number_of_cities, self.
    distance_matrix, self.nn_list))
31              self.pheromone_trail = np.full((self.number_of_cities, self.
    number_of_cities), self.trail_0)
32
33          if self.mmas_flag:
34              self.trail_max = 1 / (
35                          self.rho * nearest_neighbor_heuristic(self.
    number_of_cities, self.distance_matrix, self.nn_list))
36              self.trail_min = self.trail_max / (2 * self.number_of_cities
    )
37              self.pheromone_trail = np.full((self.number_of_cities, self.
    number_of_cities), self.trail_max)
38
39          if self.acs_flag:
40              self.trail_0 = self.number_of_ants / (
41                          self.number_of_cities *
    nearest_neighbor_heuristic(self.number_of_cities, self.
    distance_matrix, self.nn_list))
42              self.pheromone_trail = np.full((self.number_of_cities, self.
    number_of_cities), self.trail_0)
43
44          self.choice_info = initialize_choice_info_matrix(self.
    number_of_cities, self.pheromone_trail, self.heuristic, self.alpha,
45                                                           self.beta)
46
47      def ACSLocalPheromoneUpdate(self, k, i):
48          h = self.ant[k].tour[i-1]
49          j = self.ant[k].tour[i]
50          xi = 0.1
51          self.pheromone_trail[h][j] = (1-xi)*self.pheromone_trail[h][j]+
    xi*self.trail_0
52          self.pheromone_trail[j][h] = self.pheromone_trail[h][j]
53          self.choice_info[h][j] = self.pheromone_trail[h][j] * (1/self.
    distance_matrix[h][j])**self.beta
54          self.choice_info[j][h] = self.choice_info[h][j]
55
```

```
56      def UpdateStatistics(self):
57          self.BestSolutionSoFar()
58          iteration_best = self.ant[0]
59
60          if iteration_best.tour_length < self.best_so_far_ant.tour_length
        or self.best_so_far_ant.tour_length == -1:
61              self.best_so_far_ant = iteration_best
62
63              if self.mmas_flag:
64                  self.trail_max = 1/(self.rho*self.best_so_far_ant.
    tour_length)
65                  self.trail_min = self.trail_max*(1-(0.05**(1/self.
    number_of_cities)))/(((self.number_of_cities/2)-1)*(0.05**(1/self.
    number_of_cities)))
66
67
68      def ChooseBestNext(self, k, i):
69          v = 0.0
70          c = self.ant[k].tour[i - 1]
71          nc = self.ant[k].tour[0]
72          for j in range(self.number_of_cities):
73              if not self.ant[k].visited[j]:
74                  if self.choice_info[c][j] > v:
75                      nc = j
76                      v = self.choice_info[c][j]
77
78          self.ant[k].tour[i] = nc
79          self.ant[k].visited[nc] = 1
80
81      def ASDecisionRule(self, k, i):
82
83          if self.acs_flag:
84              q = np.random.uniform()
85              if q < self.q_0:
86                  self.ChooseBestNext(k, i)
87                  return
88          c = self.ant[k].tour[i - 1]
89          sum_probabilities = 0.0
90          selection_probability = np.zeros(self.number_of_cities)
91          for j in range(self.number_of_cities):
92              if self.ant[k].visited[self.nn_list[c][j]]:
93                  selection_probability[j] = 0.0
94              else:
95                  selection_probability[j] = self.choice_info[c][self.
    nn_list[c][j]]
96                  sum_probabilities = sum_probabilities +
    selection_probability[j]
```

```
97
98          if sum_probabilities == 0.0:
99              self.ChooseBestNext(k, i)
100         else:
101             r = np.random.uniform(0, sum_probabilities)
102             j = 1
103             p = selection_probability[j]
104
105             while p < r:
106                 j = j + 1
107                 p = p + selection_probability[j]
108
109             self.ant[k].tour[i] = self.nn_list[c][j]
110             self.ant[k].visited[self.nn_list[c][j]] = 1
111
112     def ComputeTourLength(self, k):
113         tour = self.ant[k].tour
114         tour_length = 0.0
115
116         for i in range(self.number_of_cities):
117             tour_length = tour_length + self.distance_matrix[tour[i]][
    tour[i + 1]]
118         return tour_length
119
120     def ConstructSolution(self):
121         for k in range(self.number_of_ants):
122             for i in range(self.number_of_cities):
123                 self.ant[k].visited[i] = False
124         step = 0
125         for k in range(self.number_of_ants):
126             r = random.randint(0, self.number_of_cities - 1)
127             self.ant[k].tour[step] = r
128             self.ant[k].visited[r] = True
129
130         while step < self.number_of_cities:
131             step = step + 1
132             for k in range(self.number_of_ants):
133                 self.ASDecisionRule(k, step)
134                 if self.acs_flag:
135                     self.ACSLocalPheromoneUpdate(k, step)
136         step = self.number_of_cities
137         for k in range(self.number_of_ants):
138             self.ant[k].tour[self.number_of_cities] = self.ant[k].tour
    [0]
139             self.ant[k].tour_length = self.ComputeTourLength(k)
140             if self.acs_flag:
141                 self.ACSLocalPheromoneUpdate(k, step)
```

```
142
143     def Evaporate(self):
144         for i in range(self.number_of_cities):
145             for j in range(self.number_of_cities):
146                 self.pheromone_trail[i][j] = (1 - self.rho) * self.
    pheromone_trail[i][j]
147                 self.pheromone_trail[j][i] = self.pheromone_trail[i][j]
148
149     def GlobalACSPheromoneUpdate(self):
150         d_tau = 1 / self.best_so_far_ant.tour_length
151         for i in range(self.number_of_cities):
152             j = self.best_so_far_ant.tour[i]
153             h = self.best_so_far_ant.tour[i + 1]
154
155             self.pheromone_trail[j][h] = (1 - self.rho) * self.
    pheromone_trail[j][h] + self.rho * d_tau
156             self.pheromone_trail[h][j] = self.pheromone_trail[j][h]
157
158             self.choice_info[h][j] = (self.pheromone_trail[h][j] ** self
    .alpha) * (self.heuristic[h][j] ** self.beta)
159             self.choice_info[j][h] = self.choice_info[h][j]
160
161     def DepositPheromone(self, k):
162         delta_tau = 1 / self.ant[k].tour_length
163
164         for i in range(self.number_of_cities):
165             j = self.ant[k].tour[i]
166             l = self.ant[k].tour[i + 1]
167             self.pheromone_trail[j][l] = self.pheromone_trail[j][l] +
    delta_tau
168             self.pheromone_trail[l][j] = self.pheromone_trail[j][l]
169
170     def MMAS_DepositPheromone(self):
171         delta_tau = 1 / self.best_so_far_ant.tour_length
172
173         for i in range(self.number_of_cities):
174             j = self.best_so_far_ant.tour[i]
175             l = self.best_so_far_ant.tour[i + 1]
176             self.pheromone_trail[j][l] = self.pheromone_trail[j][l] +
    delta_tau
177             self.pheromone_trail[l][j] = self.pheromone_trail[j][l]
178
179     def ComputeChoiceInformation(self):
180         for i in range(self.number_of_cities):
181             for j in range(i, self.number_of_cities):
182                 self.choice_info[i][j] = self.pheromone_trail[i][j] **
    self.alpha + self.heuristic[i][j] ** self.beta
```

```
183                 self.choice_info[j][i] = self.choice_info[i][j]
184         return self.choice_info
185
186     def CheckPheromoneTrailLimits(self):
187         for i in range(self.number_of_cities):
188             for j in range(i):
189                 if self.pheromone_trail[i][j] < self.trail_min:
190                     self.pheromone_trail[i][j] = self.trail_min
191                     self.pheromone_trail[j][i] = self.trail_min
192                 elif self.pheromone_trail[i][j] > self.trail_max:
193                     self.pheromone_trail[i][j] = self.trail_max
194                     self.pheromone_trail[j][i] = self.trail_max
195
196     def ASPheromoneUpdate(self):
197
198         if self.as_flag or self.mmas_flag:
199             self.Evaporate()
200
201         if self.as_flag:
202             for k in range(self.number_of_ants):
203                 self.DepositPheromone(k)
204         if self.mmas_flag:
205             self.MMAS_DepositPheromone()
206             self.CheckPheromoneTrailLimits()
207         if self.acs_flag:
208             self.GlobalACSPheromoneUpdate()
209
210         self.ComputeChoiceInformation()
211
212     def BestSolutionSoFar(self):
213         self.ant.sort(key=operator.attrgetter('tour_length'))
```

Kod 4.1: Implementacije klase ACO u Pythonu.

```
1  import numpy as np
2  import random
3
4  class Ant:
5
6      def __init__(self, n):
7          self.n = n
8          self.tour_length = -1
9          self.tour = np.full(n + 1, -1)
10         self.visited = np.full(n, False)
11
12     def ant_empty_memory(self):
```

```
        for i in range(self.n):
            self.visited[i] = False

    def place_ant(self, step):
        rnd = random.randint(0, self.n - 1)
        self.tour[step] = rnd
        self.visited[rnd] = True
```

Kod 4.2: Implementacija klase Ant u Pythonu.

```
import random
import matplotlib.pyplot as plt
import numpy as np
import math


def generate_cities(number_of_cities):

    x_coordinate = random.sample(range(1, number_of_cities+1), number_of_cities)
    y_coordinate = random.sample(range(1, number_of_cities+1), number_of_cities)
    coordinates = np.column_stack((x_coordinate, y_coordinate))

    return x_coordinate, y_coordinate, coordinates


def calculate_distance_matrix(coordinates, number_of_cities):

    distance_matrix = np.zeros((number_of_cities, number_of_cities))
    for i in range(len(coordinates)):
        for j in range(i, len(coordinates)):
            distance_matrix[i][j] = math.sqrt((coordinates[i][0] - coordinates[j][0]) ** 2 + (coordinates[i][1] - coordinates[j][1]) ** 2)
            distance_matrix[j][i] = distance_matrix[i][j]
    return distance_matrix


def nearest_neighbor_list(number_of_cities, distance_matrix):
    nn_list = []
    for i in range(number_of_cities):
        nn_list.append(np.argsort(distance_matrix[i]))
    return nn_list


def nearest_neighbor_heuristic(number_of_cities, distance_matrix, nearest_neighbor_list):
```

```
34      visited_cities = set()
35      current_city = 0;
36      visited_cities.add(0)
37      distance_passed = 0
38      i = 0
39      while i < number_of_cities:
40          for city in nearest_neighbor_list[current_city]:
41              if city not in visited_cities:
42                  distance_passed = distance_passed + distance_matrix[
    current_city, city]
43                  current_city = city
44                  visited_cities.add(current_city)
45                  break
46          i = i+1
47
48      return distance_passed
49
50
51  def initialize_pheromone_trail(number_of_ants, number_of_cities,
    distance_matrix, nearest_neighbor_list):
52      distance_passed = nearest_neighbor_heuristic(number_of_cities,
    distance_matrix, nearest_neighbor_list)
53      pheromone_trail = np.full((number_of_cities, number_of_cities),
    number_of_ants/distance_passed)
54      return pheromone_trail
55
56
57  def initialize_heuristic(number_of_cities, distance_matrix):
58      heuristic_information = np.zeros((number_of_cities, number_of_cities
    ))
59      for i in range(number_of_cities):
60          for j in range(i, number_of_cities):
61              if i != j:
62                  heuristic_information[i][j] = 1/distance_matrix[i][j]
63                  heuristic_information[j][i] = heuristic_information[i][j
    ]
64      return heuristic_information
65
66
67  def initialize_choice_info_matrix(number_of_cities, pheromone_trail,
    heuristic, alpha, beta):
68      choice_info = np.zeros((number_of_cities, number_of_cities))
69      for i in range(number_of_cities):
70          for j in range(i, number_of_cities):
71              choice_info[i][j] = pheromone_trail[i][j]**alpha + heuristic
    [i][j]**beta
72              choice_info[j][i] = choice_info[i][j]
```

```
73      return choice_info
```

Kod 4.3: Implementacija pomoćnih funkcija u Pythonu.

Bibliografija

[1] M. Dorigo, *ACO: Public Software*, `http://www.aco-metaheuristic.org/aco-code/`, 2018, posjećeno u studenom 2018.

[2] ———, *Ant Colony Optimization*, `http://www.scholarpedia.org/article/Ant_colony_optimization`, 2018, posjećeno u studenom 2018.

[3] M. Dorigo i T. Stützle, *Ant Colony Optimization*, The MIT Press, Cambridge, Massachusetts, 2004.

[4] T. Stützle i H. H. Hoos, *MAX–MIN Ant System*, Future Generation Computer Systems **16** (2000), br. 8, 889–914.

Sažetak

U ovom radu bavimo se algoritmima optimizacije kolonijom mrava (engl. Ant colony optimization, skraćeno ACO). Na početku objašnjavamo od kuda dolazi motivacija za ovakvu vrstu algoritama i dajemo opis ACO metaheuristike i njezine primjene na problem trgovačkog putnika. U nastavku opisujemo tri najpoznatije vrste ACO algoritama, to su: Mravlji sustav, MAX–MIN mravlji sustav i Sustav mravlje kolonije. Također, navodimo sve njihove najvažnije karakteristike. Na kraju, navodimo eksperimentalne rezultate dobivene primjenom algoritama na problem trgovačkog putnika. Na kraju se nalazi i kod ACO algoritama, implementiran u programskom jeziku Python, iz kojeg su dobiveni navedeni rezultati.

Summary

This paper presents the Ant colony optimization algorithms (ACO). At the beginning, we explain where does the motivation for this kind of algorithm come from and we give a definition of ACO metaheuristics and its application to the travelling salesman problem. Further, we describe the three most successful ACO algorithms: Ant System, MAX–MIN Ant System and Ant Colony System. Also, we describe all their most important characteristics. At the end of this paper we present experimental results obtained by applying algorithms to the travelling salesman problem. Finally, we also give a code, written in Python, which is used to get the experimental results.

Životopis

Rođena sam 6. prosinca 1993. godine u Zagrebu, odrastala sam u Krapini gdje sam pohađala Srednju školu Krapina, smjer prirodoslovno-matematička gimnazija. Godine 2012. upisala sam Preddiplomski sveučilišni studij matematike na Matematičkom odsjeku Prirodoslovno-matematičkog fakulteta u Zagrebu, koji sam završila 2016. godine. Iste godine upisujem diplomski studij Računarstvo i matematika, također, na Prirodoslovno-matematičkom fakultetu u Zagrebu.

Tijekom ljeta 2017. godine sudjelovala sam na Ericsson Summer Campu na projektu Smart IoT Analytics, a od ožujka 2018. godine radim u tvrtki AVL-AST u Zagrebu.