Raspodijeljeni biološki inspiriran algoritam

Korpar, Miljenko

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Miljenko Kopar

RASPODIJELJENI BIOLOŠKI INSPIRIRAN ALGORITAM

DIPLOMSKI RAD

Varaždin, 2023.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Miljenko Korpar

Matični broj: 44920/16–R

Studij: Informacijsko i programsko inženjerstvo

RASPODIJELJENI BIOLOŠKI INSPIRIRAN ALGORITAM

DIPLOMSKI RAD

Mentor/Mentorica:

Izv. prof. dr. sc. Nikola Ivković

Varaždin, rujan 2023.

Miljenko Korpar

Izjava o izvornosti

Izjavljujem da je moj završni/diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Tema ovog rada je opis raspodijeljene implementacije odabranog biološki inspiriranog algoritma. Biološki inspirirani algoritmi predstavljaju skupinu algoritama koji simuliraju biološke procese ili ponašanje kolonije jedinki. Najčešće se koriste za rješavanje kompleksnih optimizacijskih problema u znanosti, inženjerstvu i industriji. Početni dio rada sadrži opis nekoliko poznatih algoritama iz ove klase s naglaskom na algoritme optimizacije kolonijom mrava. Drugi dio rada opisuje modele, topologije i strategije koje se mogu koristiti za raspodijeljenu implementaciju, a na kraju se nalazi i opis vlastite implementacije distribuiranog algoritma optimizacije kolonijom mrava. Ključne prednosti raspodijeljene implementacije su: skalabilnost, performanse, otpornost (uvodi se više čvorova što rješava problem centralnog mjesta ispada sustava) te isplativost. Od navedenih prednosti u radu su detaljnije testirane performanse.

Ključne riječi: biološki inspirirani algoritmi; evolucijski algoritmi; ACO; TBAS; MMAS; distribuirani algoritmi; topologije; paralelni algoritmi; otočni model;

Sadržaj

1. Uvod ... 1

2. Biološki inspirirani algoritmi ... 3

2.1. Opis, klasifikacija i motivacija za njihovo korištenje ... 3

2.2. Genetski algoritam ... 5

2.2.1. Opis algoritma ... 5

2.2.2. Genetski operatori ... 6

2.3. Algoritam optimizacije rojem čestica ... 7

2.3.1. Opis osnovnog algoritma ... 7

2.3.2. Modifikacije algoritma ... 8

2.4. Algoritmi optimizacije kolonijom mrava ... 9

2.4.1. Jednostavan algoritam optimizacije kolonijom mrava ... 12

2.4.2. Ant System ... 14

2.4.3. Max-Min Ant System ... 15

2.4.4. Three Bound Ant System ... 18

3. Raspodijeljena implementacija ... 19

3.1. Razlozi za raspodijeljenu implementaciju ... 19

3.2. Taksonomija i modeli raspodijele ... 22

3.3. Topologije i migracijske strategije ... 29

3.4. Performanse ... 34

3.5. Implementacija i rezultati ... 38

3.5.1. TBAS – slijedna implementacija ... 38

3.5.2. TBAS – raspodijeljena implementacija ... 43

3.5.3. Rezultati ... 46

4. Zaključak ... 52

Popis literature ... 54

Popis slika ... 56

Popis tablica ... 57

1. Uvod

Biološki inspirirani algoritmi spadaju u skupinu metaheuristika koje simulacijom ponašanja iz prirode rješavaju računalno zahtjevne optimizacijske probleme. Za razliku od uobičajenih algoritama s determinističkim ponašanjem, metaheuristike najčešće daju dovoljno dobra rješenja u prihvatljivom vremenu. Međutim, njihova vremenska složenost također ima značajan utjecaj na performanse pa originalne sekvencijalne verzije algoritma često nisu u stanju u prihvatljivom vremenu dati dovoljno dobra rješenja za probleme velikih dimenzija. S obzirom da danas većina računala dolazi s višejezgrenim procesorima, jedno od rješenja je paralelna , tj. distribuirana implementacija. Cilj ovog rada je dati pregled modela, topologija i strategija koje se mogu koristiti za raspodijeljenu implementaciju algoritama optimizacije kolonijom mrava (eng. *Ant Conoloy Optimization - ACO*), ali i testirati odabrane modele i strategije na vlastitom rješenju. Inicijalna pretpostavka je da će se najveće ubrzanje ostvariti usporedbom slijedne i distribuirane implementacija originalnog TBAS algoritma (bez optimizacija), dok će uvedene optimizacije smanjiti razliku u performansama.

Prvi dio rada sadrži opis, podjelu biološki inspiriranih te motivacijski primjer u kojem je prikazano zašto su algoritmi iz ove skupine značajni za rješavanje optimizacijskih problema. Nakon toga se u radu nalazi opis nekoliko poznatih algoritama iz ove klase, a naglasak je stavljen na ACO algoritme. U tom dijelu nalaze se četiri važne inačice ACO algoritama koje su poredane kronološki prema godini predstavljanja. Svaki algoritam dolazi s opisom temeljnih ideja i razlika u odnosu na prethodnika, važnim algebarskim izrazima koji se koriste za izračune u različitim koracima algoritma te opisom utjecaja članova prisutnih u tim izrazima na rad algoritma.

Drugi dio rada započinje s općenitim karakteristikama distribuiranih sustava i motivacijom za njihovu primjenu. Nakon toga se nalazi motivacijski primjer koji pokazuje probleme u vidu performansi ACO algoritama te su spomenuta dva glavna razloga za paralelizaciju ACO algoritama. Potom slijedi najvažniji dio rada u kojem se nalazi opis različitih modela za distribuiranu implementaciju te komunikacijskih strategija i topologija koje se pri tome mogu koristiti. U tom dijelu nalaze se i metrike vezane uz performanse i graf koji prikazuje ovisnost vjerojatnosti pronalaska zadovoljavajućeg rješenja o broju instanci algoritama. Nakon toga slijedi opis sekvencijalne verzije odabranog algoritma gdje su također opisane i korištene optimizacije. Nakon slijedne objašnjena je distribuirana implementacija, korišteni modeli te razlozi odabira tih modela. Na kraju se nalaze rezultati testiranja te usporedba slijedne i raspodijeljene verzije algoritma.

Za potrebe izrade ovog rada u najvećoj mjeri su korišteni znanstveni članci i knjige koje se bave tematikom prirodom inspiriranih te evolucijskih algoritama. Također su korištene knjige i članci koji općenito opisuju paralelno programiranje, paralelne algoritme te modele, probleme i preporuke za raspodijeljenu implementaciju programskih rješenja.

2. Biološki inspirirani algoritmi

2.1. Opis, klasifikacija i motivacija za njihovo korištenje

Biološki inspirirani algoritmi (eng. *bio-inspired algorithms*) spadaju u okvir biološki inspiriranog računanja (eng. *bio-inspired computing*). Biološki inspirirano računanje zasniva se na optimizacijskim algoritmima koji su inspirirani principima i evolucijskim procesima iz prirode. Razlog i motivacija za njihovo korištenje su kompleksni problemi vezani uz procesiranje informacija, odlučivanje ili optimizaciju koje navodi Darwish [1]. To su problemi za koje nemamo efikasne algoritme pa unatoč računalnoj snazi kojoj danas imamo pristup, njihovo rješavanje korištenjem dostupnih egzaktnih algoritama ne daje rješenje u zadovoljavajućem vremenu. Jedan od takvih problema je problem trgovačkog putnika (eng. *Traveling salesman problem - TSP*) – za zadanih N gradova, trgovački putnik mora pronaći najkraći put pomoću kojeg će obići sve gradove i vratiti se u grad iz kojeg je krenuo, uz ograničenje da svaki grad mora posjetiti točno jednom (iznimka je prvi grad). Naizgled jednostavan problem, ali zapravo računalno jako zahtjevan. Da bi se pronašlo optimalno rješenje, potrebno je provjeriti sve permutacije. Ako su svi gradovi međusobno povezani, znači da za N gradova, krećući od prvog grada, postoji $N-1$ susjednih gradova. Za svaki od tih gradova postoji $N-2$ susjednih gradova pa za svaki od njih još $N-3\ldots$ jasno je kako je ukupan broj permutacija jednak $(N-1)!$ i da je složenost algoritma faktorijelna. Testira li se ovakav algoritam na hardveru koji nam je dostupan danas (Intel i5-8400H), rezultati su sljedeći.

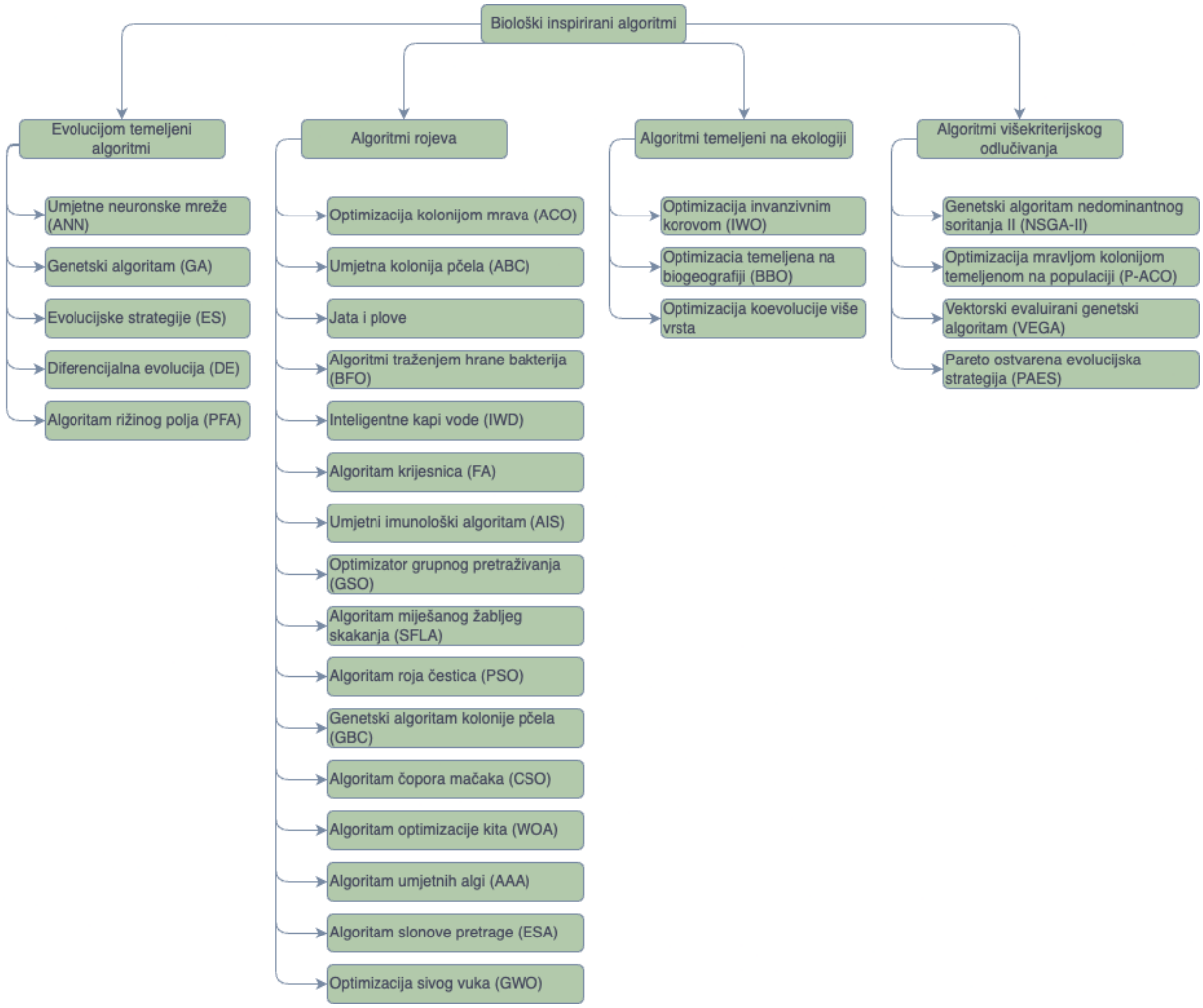
Tablica 1. Vrijeme izvršavanja TSP algoritama

Broj gradova	Vrijeme izvršavanja algoritma
12	~ 5 s
13	~ 65 s
14	~ 15 min
15	~ 4h i 20 min
16	~ 2 dana i 17 sati
17	~ 43 dana
18	~ 2 godine

Napomena – algoritam je testiran za veličine ulaza od 12 do 15 gradova, dok vremena za veći broj gradova predstavljaju procjenu izračunatu na temelju prethodnih vremena.
Iz tablice je jasno vidljivo kako je za rješavanje ovakvih problema potrebno posegnuti za novim metodama. Algoritmi koje se u takvim slučajevima često koriste spadaju u skupinu heuristika i metaheuristika. Jednostavnim rječnikom, heuristike su algoritmi koji daju dovoljno dobra

rješenja u prihvatljivom vremenu. Pri tome dobiveno rješenje može, ali i ne mora biti optimalno. Postoje i heuristike koje nisu problemski specifične zbog čega se mogu prilagoditi i upotrijebiti za rješavanje većeg skupa problema. Takvi algoritmi se nazivaju metaheuristikama [2]. Često se znaju i kombinirati s heuristikama - metaheuristika generira rješenja koja su potom ulaz za heuristiku koja mijenja to rješenje i nastoji pronaći bolje. Biološki inspirirani algoritmi, posebice algoritmi rojeva, spadaju upravu u kategoriju metaheuristika. Potrebno je još spomenuti kako su metaheuristike stohastički algoritmi, tj. algoritmi čije se ponašanje ne može u potpunosti determinirati - ako se identičan problem pokuša k puta riješiti pomoću metaheuristike, moguće je da će svih k puta rješenje biti drugačije.

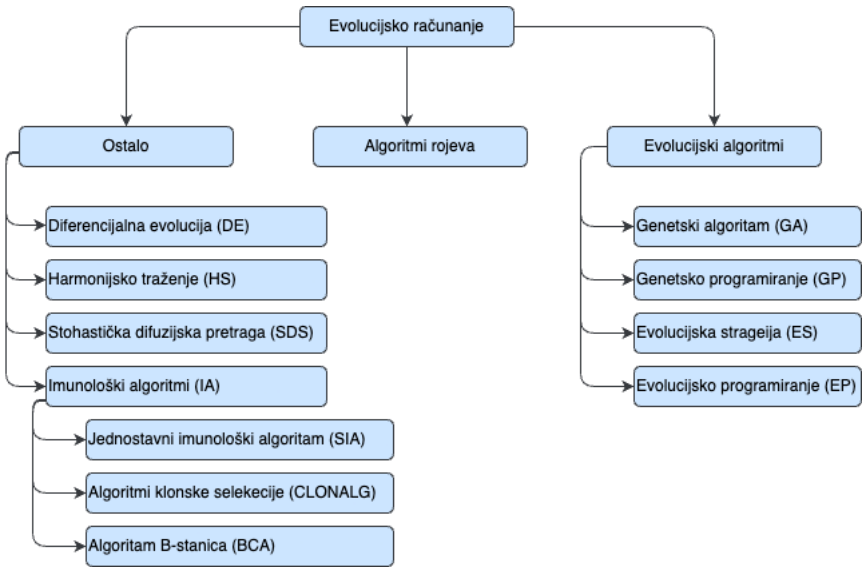
U nastavku se nalazi klasifikacija biološki inspiriranih algoritama.



Slika 1. Klasifikacija biološki inspiriranih algoritama (Prema [3])

Algoritmi rojeva često se spominju i u okviru evolucijskog računanja (eng. *evolutionary computing* - *EC*). Jedan od razloga je što jedan od izumitelja algoritma roja čestica naziva taj algoritam evolucijskim algoritmom (eng. *evolutionary algorithm* - *EA*). Drugi razlog je što se

algoritmi rojeva ponašaju slično kao i evolucijski algoritmi – postoji populacija rješenja koja se poboljšava svakom iteracijom [4]. U nastavku slijedi klasifikacija evolucijskog računanja.



Slika 2. Klasifikacija evolucijskog računanja (Prema [5])

2.2. Genetski algoritam

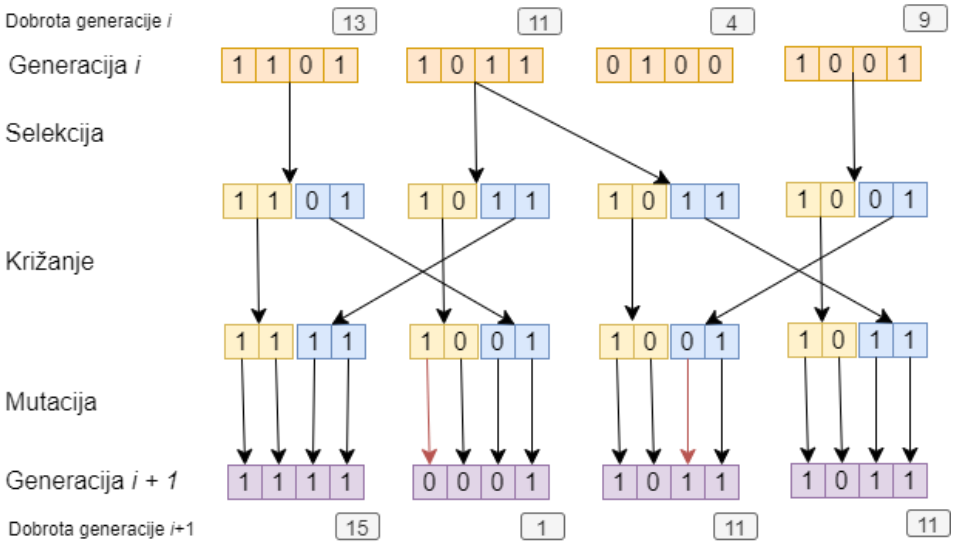
2.2.1. Opis algoritma

Genetski algoritam (eng. *genetic algorithm*) temelji se na postojanju populacije gdje potomaka ima više no što je hrane dostupno, stoga se jedinke bore za preživljavanje. Jedinke s boljim genetskim materijalom imaju veću šansu preživjeti, a djeca dijele sličan genetski materijal s roditeljima uz određeno odstupanje.

Algoritam dakle radi s populacijom jedinki, a svaka jedinka predstavlja jedno rješenje problema. Ovo je slično algoritmu optimizacije rojem čestica, a razlika je da ovdje jedinke nisu čestice nego kromosomi. U kontekstu slijednog algoritma postoje dvije verzije – eliminacijski genetski algoritam i generacijski genetski algoritam [5]. Eliminacijski podrazumijeva populaciju koja se polako mijenja i poboljšava iz iteracije u iteraciju. Primjerice, dijete nastalo u i-toj iteraciji prema određenom kriteriju zamjenjuje jedinku koja se nalazi u trenutnoj populaciji. Kriterij za zamjenu može biti kvaliteta rješenja gdje nova jedinka zamjenjuje onu s najlošijom dobrotom. Kod generacijskog genetskog algoritma u svakoj iteraciji dolazi do potpune izmjene populacije – djeca nastala u prošloj iteraciji postaju roditelji u sljedećoj.

Rješenja, tj. kromosomi u populaciji jedinki najčešće se kodiraju koristeći niz bitova, a kodirani niz zapravo predstavlja genotip kromosoma.

Svaka iteracija algoritma sastoji se od nekoliko operacija, a to su redom: selekcija, križanje, mutacija i zamjena.



Slika 3. Prikaz rada genetskog algoritma

2.2.2. Genetski operatori

Operator selekcije prvi je operator koji se primjenjuje u svakoj iteraciji. Selekcijom se odabiru jedinke (roditelji) koji sudjeluju u križanju. Postoji nekoliko načina na koje se može provesti selekcija, a tri najčešća su: proporcionalna selekcija, selekcija linearnim rangiranjem, turnirska selekcija. Kod proporcionalne selekcije je vjerojatno selekcije jedinke proporcionalna njezinoj dobroti. Što je dobrote neke jedinke veća od ostalih, to je veća vjerojatno da će ona biti selektirana. Kod selekcije linearnim rangiranjem rješenja u populaciji se sortiraju uzlazno prema vrijednostima dobrote rješenja i potom im se pridjeljuje vjerojatnost odabira koja je linearno proporcionalna poziciji rješenja [5]. Kod k-turnirske selekcije se iz populacije slučajnim odabirom uzima uzorak od k rješenja i iz uzorka se odabire rješenje s najvećom dobrotom. Postupak se ponavlja toliko puta koliko roditelja je potrebno odabrati.

Nakon što su selektirane jedinke slijedi operacija križanja. Operator križanja kombinira genetski materijal jedinki kako bi se stvorile nove jedinke. Pri tome se najčešće koriste križanja s jedinom i t točaka prekida te uniformno križanje [5]. U prvom slučaju se za par roditelja slučajnim odabirom odabire točka prekida. Primjerice, oba roditelja se sastoje od 4 bita i točka prekida se nalazi nakon prvog bita. Nakon toga se sastavljaju jedinke gdje se prva sastoji od prvog dijela bitova prvog roditelja i drugog dijela bitova drugog roditelja, dok je kod druge jedinke situacija suprotna i sastoji se od prvog dijela drugog roditelja i drugog dijela prvog roditelja. Kod križanja s t točaka razlika je samo u tome da postoji više točaka prekida gdje se

svaka točka također odabire nasumično, a t može biti u rasponu $1 \leq t \leq k-1$ gdje k predstavlja broj bitova u roditelju. Kod uniformnog križanja postoji $k-1$ točka prekida. Nakon toga se križanje najčešće nastavlja na način da se zajednički dijelovi kopiraju u dijete, a potom se generira slučajan broj na temelju kojeg se na ostale poziciji u dijete kopira nula ili jedan.

Posljednja operacija je mutacija koja ima ulogu stvaranja dodatne različitosti među jedinkama. Najčešće korišteni operator mutacije je binarna mutacija uz zadanu vjerojatnost mutacije bita p_m. Kod ovog načina mutacije definira se vjerojatnost mutacije p_m, a najčešće vrijednosti su u intervalu od 1% do 5% [5]. Nakon toga se iterira kroz sve bitove za svako stvoreno dijete. Za svaki bit se generira slučajan realan broj iz intervala $[0, 1]$ (podrazumijeva se uniformna distribucija) i provjerava se je li manji od p_m. Ako jest, tada se taj bit mutira - kod binarnog niza negira se trenutni bit pa nula prelazi u jedinicu, a jedinica u nulu.

2.3. Algoritam optimizacije rojem čestica

2.3.1. Opis osnovnog algoritma

Algoritam roja čestica (eng. *Particle Swarm Optimization*, PSO) optimizacijski je algoritam za probleme čija se rješenja mogu opisati kao točke (čestica) u n-dimenzionalnom prostoru. To je dakle populacijski algoritam koji se sastoji od skupa čestica. Čestice se gibaju kroz višedimenzionalni prostor rješenja koji se pretražuje i pri tome mijenjaju svoj položaj na temelju svojeg iskustva (individualna komponenta) te iskustva susjednih čestica (sociološka komponenta). Utjecaj pojedine komponente značajno određuje ponašanje čestice. Ukoliko je dominantna individualna komponenta, čestica istražuje širi prostor rješenja. Kod dominacije sociološke komponente čestica se fokusira na fino podešavaju pronađenog rješenja [5].

Algoritam se implementira pomoću dva dvodimenzionalna polja gdje jedno sadrži položaj čestica za svaku dimenziju, a drugo brzine čestica za svaku dimenziju. Položaj i brzina svake čestice su dakle predstavljeni n-dimenzionalnim vektorima. Čestice pretražuju ograničen prostor rješenja gdje se položaj čestice x za svaku dimenziju d nalazi u granicama $x_{min}[d] \leq x[d] \leq x_{max}[d]$, a x_{min} i x_{max} predstavljaju polja u kojima se nalaze granice za svaku dimenziju. Isto vrijedi i za ograničenje brzine, $v_{min}[d] \leq v[d] \leq v_{max}[d]$, gdje su v_{min} i v_{max} polja u kojima se nalaze ograničenja brzine za svaku dimenziju.

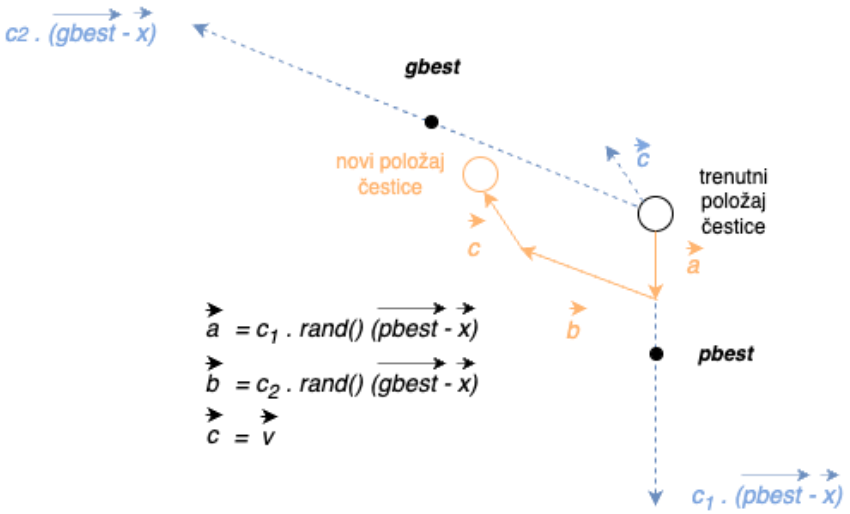
Prvi korak algoritma je inicijalizacija gdje se svaka čestica smješta na slučajno odabrani položaj te joj se postavlja slučajno odabrana brzina. Nakon toga slijedi glavni dio algoritma koji se ponavlja tako dugo dok se ne zadovolji kriterij zaustavljanja. Ovaj dio se sastoji od: evaluacije čestica tj. pronađenih rješenja, ažuriranja do sad najboljeg rješenja (položaja) svake čestice $pbest$, ažuriranje globalno najboljeg rješenja $gbest$ te postavljanje nove brzine i

položaja svake čestice. Ažuriranje brzine $v_{i,d}$ za česticu i te dimenziju d odvija se prema sljedećem izrazu

$$v_{i,d} = v_{i,d} + c_1 \cdot rand() \cdot (pbest_{i,d} - x_d) + c_2 \cdot rand() \cdot (gbest_{i,d} - x_d).$$

Faktori c_1 i c_2 zajedno s funkcijom $rand()$ koja generira slučajne vrijednosti iz intervala $[0, 1]$ služe za moduliranje individualne, odnosno socijalne komponente, a njihove vrijednosti se obično postavljaju na 2.0 [5]. Veća vrijednost komponente c_1 stavlja naglasak na individualnost jedinke i pretraživanje većeg prostora rješenja, dok veća vrijednost c_2 potiče fokusiranju pretragu oko najboljeg rješenja kojeg je pronašao čitav roj. Član $v_{i,d}$ koji se odnosu na prethodnu brzinu zapravo predstavlja inerciju čestice. Nakon ažuriranja brzina ažurira se položaj svih čestica. Novi položaj $x_{i,d}$ za česticu i te dimenziju d izračunava se prema sljedećem izrazu

$$x_{i,d} = x_{i,d} + v_{i,d}.$$



Slika 4. Ilustracija ažuriranja položaja čestice kod PSO algoritma (Prema *[5]*)

2.3.2. Modifikacije algoritma

Bilo bi poželjno kad bi algoritam radio na način da u početku pretražuje širi prostor rješenja, a da se kasnije, kad pronađe dobra rješenja, fokusira na područja oko njih kako bi pronašao još bolja. Ovakvo ponašanje može se postići uvođenjem faktora inercije ω čime se izraz za ažuriranje brzine mijenja u izraz

$$v_{i,d} = \omega(t) \cdot v_{i,d} + c_1 \cdot rand() \cdot (pbest_{i,d} - x_d) + c_2 \cdot rand() \cdot (gbest_{i,d} - x_d).$$

Faktor inercije se smanjuje s vremenom, a obično se u koraku inicijalizacije postavlja na vrijednost koja je blizu 1. Na ovaj će način čestica nakon nekog vremena prilikom ažuriranja brzine u obzir uzimati samo lokalno i globalno najbolje rješenje čime će se pretraga prostora

rješenja ograniči na zanimljivo područje. Ako se nakon T iteracija želi postići ovakva fokusirana pretraga, tada se ažuriranje faktora inercije može provoditi prema sljedećem izrazu [5]

$$\omega(t) = \begin{cases} \frac{t}{T} \cdot (\omega_{min} - \omega_{max}) + \omega_{max}, & t < T \\ \omega_{min}, & t \geq T \end{cases}$$

gdje ω_{min} i ω_{max} predstavljaju minimalnu i maksimalnu vrijednost faktora inercije, a t predstavlja broj trenutne iteraciju.

Problem PSO algoritma koji koristi $pbest$ i $gbest$ je u tome da može dovesti do prerane konvergencije ukoliko neka čestica u ranoj fazi pretrage pronađe značajno bolje rješenje od ostalih i time privuče ostale prema tom rješenju. Ovaj problem je riješen uvođenje susjedstva između čestica. Umjesto da svaka čestica prilikom ažuriranja brzine uzima globalno najbolje rješenje, u obzir uzima lokalno najbolja rješenja od čestica iz susjedstvu. Pri tome je susjedstvo definirano topologijom i ne definira se prema blizini čestica u prostoru rješenja [5]. Jedan način je da se definira parametar veličine susjedstva n_s, a čestice se slože u niz. Za $n_s = 2$, susjedi od čestice i su čestice $i-1, i-2, i+1, i+2$ te je dodatno čestica i također i susjed sama sebi. U slučaju korištenja susjedstva, najbolje rješenje u susjedstvu čestice se označava s $lbest$ (eng. *local best*). Izraz za ažuriranje brzine tada se mijenja u

$$v_{i,d} = \omega(t) \cdot v_{i,d} + c_1 \cdot rand() \cdot (pbest_{i,d} - x_d) + c_2 \cdot rand() \cdot (lbest_{i,d} - x_d)$$

gdje $lbest_i$ predstavlja najbolje rješenje u susjedstvu čestice i.

2.4. Algoritmi optimizacije kolonijom mrava

Algoritam optimizacije kolonijom mrava jedan je od najpoznatijih prirodom inspiriranih algoritama i predmet brojnim znanstvenih istraživanja, a koristi se za rješavanje kombinatoričkih optimizacijskih problema poput TSP-a. Algoritam je predstavio Marco Dorigo 1992. godine u svojoj doktorskoj disertaciji. Ovaj algoritam predstavlja metaheuristiku zbog čega se ACO koristi za rješavanje vrlo široke klase problema. Prilikom rješavanja specifičnih problema u ACO algoritme se ponekad ugrađuju problemski specifične heuristike kako bi se (što brže) došlo do prihvatljivog rješenja.

Postoji nekoliko aspekata ponašanja kolonije mrava koja služe kao temelj različitih mravljih algoritama. To su: potraga za hranom, podjela rada, sortiranje legla i kooperativni transport. Ovaj algoritam oponaša način na koji kolonija mrava dolazi do izvora hrane. Oni se prema pronađenom izvoru hrane ne kreću nasumično, nego s vremenom tvore jedan put kojeg većina mrava slijedi. Da bi to ostvarili, koriste se stigmergijom – doprinosom putem fizičkog medija. Mediji kojeg pri tome koriste su feromoni – organske tvorevine koje izlučuju, a koje služi kao medij za indirektnu komunikaciju [6]. Postoji nekoliko vrsta feromona koje mravi izlučuju u različitim situacijama: feromoni za označavanje puta i teritorija (eng. *trail pheromones*),

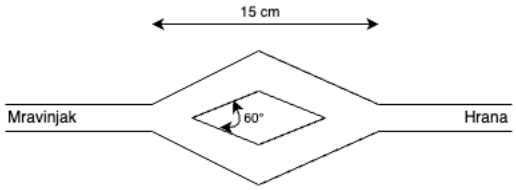
feromoni spola (eng. *sex pheromones*), alarmirajući feromoni (eng. *alarm pheromones*), primarni feromoni (eng. *primer pheromones*) i agregacijski feromoni (eng. *aggregation pheromones*) [7]. ACO algoritam temelji se na feromonima za označavanje tragova puta prilikom potrage za hranom.

Mravi prilikom kretanju od izvora prema mravinjaku te od mravinjaka prema izvoru deponiraju određenu količinu feromona pri čemu manja količina kontinuirano isparava. Mravi mogu nanjušiti te feromone, a feromonski miris utječe na njihov odabir puta - što je veća količina feromona na nekom putu, to je veća vjerojatnost da će mrav odabrati upravo taj put [6]. Na taj način će se uspostaviti put prema izvoru hrane.

Dorigo i Stützle [6] spominju tri eksperimenta koja se znanstvenici proveli, a koji pokazuju kako se mravi ponašaju u tri različite situacije prilikom potrage za hranom

a) Postoje dva puta iste dulje
b) Postoje dva puta različite duljine
c) Na početku postoji samo jedan put, a kasnije se dodaje drugi, kraći put

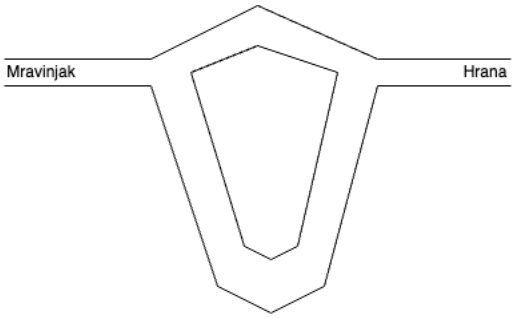
U eksperimentu a) postavljena su dva puta jednake duljine. Nakon nekog vremena, gotovo svi mravi su se kretali samo jednim putem, iako su oba iste duljine. Objašnjenje za ovakvo ponašanje je sljedeće. Na početku nema deponiranih feromona pa mravi s istom vjerojatnošću odabiru oba puta. Međutim, u nekom će trenutku malo veći broj mrava odabrati jedan put umjesto drugog. To će za posljedicu imati da će na tom put biti deponirana nešto veća količina feromona nego na drugom, što će povećati vjerojatnost da sljedeći mravi odaberu taj put. Tako će u sljedećim obilascima količina feromona na tom putu rasti brže nego na drugom što će u konačnici dovesti do dovoljno velike razlike da gotovo svi mravi biraju upravo taj put umjesto drugog. U literaturi se takav proces naziva i autokatalizacija (eng. *autocatalyzation*) ili mehanizam pozitivne povratne sprege (eng. *positive feedback*) [6].



Slika 5. Prvi eksperiment dvokrakog mosta (Prema *[6]*)

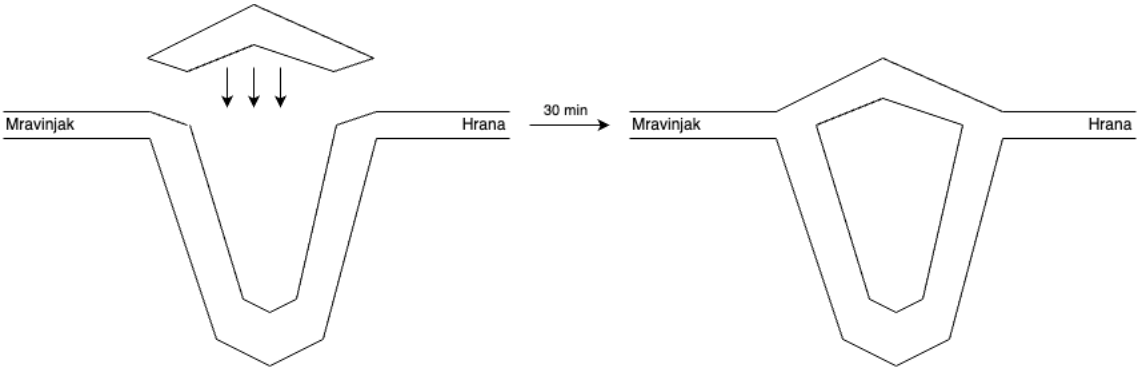
U eksperimentu b) postavljena su dva puta do izvora hrane gdje je jedan kraći od drugog. Nakon nekog vremena, gotovo svi su mravi odabrali kraći put. Kao i u primjeru a), na početku nema deponiranih feromona pa je vjerojatnost da će mravi odabrati dulji ili kraći put jednaka. Međutim, s obzirom da je jedan put kraći, mravi koji su odabrali taj put će prije stići do hrane, vratiti se do mravinjaka i ponoviti postupak. Zbog toga će količina feromona na tom putu brže

rasti nego na dužem putu. Nakon nekog vremena, količina feromona na kraćem putu će biti značajno veća nego na dužem te će gotovo svi mravi birati upravo taj put [6]. Ovaj primjer zapravo pokazuje kako su algoritmi temeljeni na ovom ponašanju pogodni za rješavanje optimizacijskih problema.



Slika 6. Drugi eksperiment dvokratkog mosta (Prema *[6]*)

U eksperimentu c), postavljan je jedan put te nakon nekog vremena drugi, kraći put. Kad je uveden kraći put, samo mali broj mrava je birao taj put. Objašnjenje za takvo ponašanje je sljedeće – s obzirom da su mravi na početku imali na raspolaganju samo duži put, svi su birali upravo njega pa je količina feromona na tom putu, nakon nekog vremena, značajno narasla. Iako je uveden kraći put, zbog sporog isparavanja feromona gotovo svi su mravi i dalje birali duži put zbog velike koncentracije feromona koja je bila prisutna na tom putu [6]. Ovaj primjer pokazuje kako se mravi ponašaju u dinamičnom okruženju. Na računalu ovo zapravo ne predstavlja velik problem jer se faktor isparavanja može kontrolirati, a može se i dinamički mijenjati tijekom izvođenja programa (npr. prilikom promjene čvorova u grafu privremeno se poveća faktor isparavanja).



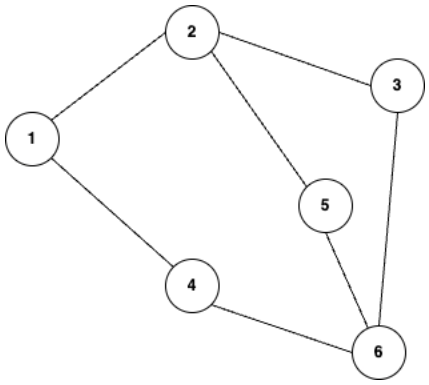
Slika 7. Treći eksperiment dvokrakog mosta (Prema *[6]*)

2.4.1. Jednostavan algoritam optimizacije kolonijom mrava

Od prvog predstavlja ACO algoritma do danas osmišljene su brojne modifikacije kako bi se dobilo što točnije, odnosno optimalnije rješenje u što kraćem vremenu. Ono što je svim varijantama zajedničko su osnovne operacije. Osnovni algoritam zadan je sljedećim pseudokodom [6]:

```
dok je broj_iteracije < maks_broj_iteracija
      Za svakog mrava u populaciji
            konstruiraj_rješenje()
      ispari_feromonske_tragove()
      Za svakog mrava u populaciji
            ažuriraj_količinu_feromona().
```

Jedan od najjednostavnijih algoritama koji radi na taj način se u literaturi spominje pod nazivom jednostavni ACO (eng. *Simple-ACO - S-ACO).*



Slika 8. Primjer strukture na kojoj se temelji rad ACO algoritma

Vjerojatnost p_{ij}^k da će mrav k odabrati brid ij računa se na sljedeći način [6]:

$$p_{ij}^k = \begin{cases} \frac{\tau_{ij}^\alpha}{\sum_{l \epsilon N_i^k} \tau_{il}^\alpha}, za\ j \in N_i^k \\ 0, \qquad za\ j \notin N_i^k \end{cases}$$

pri čemu τ_{ij} predstavlja količinu feromona na bridu ij, α predstavlja konstantu kojom se podešava utjecaj feromonskih tragova na vjerojatnost odabira sljedećeg brida/vrha, a N_i^k predstavlja okružje mrava k na vrhu i. Okružje predstavlja sve vrhove (osim prošlog) koji su bridom spojeni s vrhom i [6]. Ako se za primjer uzme graf s prethodne slike, tada je $N_1^1 = 2,4$. Konkretno, ako se mrav k nalazi na vrhu 1, tada je vjerojatnost p_{14}^k da će taj mrav odabrati vrh 4 biti jednaka

$$p_{14}^k = \frac{\tau_{14}^\alpha}{\tau_{12}^\alpha + \tau_{14}^\alpha}$$

U fazi inicijalizacije na sve se bridove postavlja ista količina feromona, a mravi se raspoređuju po čvorovima. Za raspored mrava može se koristiti nekoliko strategija – svi mravi kreću iz istog vrha, mravi se nasumično raspoređuju po vrhovima ili svaki mrav ima svoj vrh (ako je broj mrava jednak broju gradova). U svakoj iteraciji svaki mrav k krećući od početnog čvora odabire sljedeći čvor na temelju vjerojatnosti p_{ij}^{k} i ponavlja ovu proceduru sve dok ne dođe do cilja (stigne do izvora hrane ili obiđe sve gradove). S obzirom da svaki mrav rješenje gradi dio po dio, ovakvi algoritmi spadaju u kategoriju konstrukcijskih algoritama [5].

Nakon što su mravi konstruirali rješenje ažurira se količina feromona. Količina $\Delta\tau^{k}$ koju će mrav k deponirati obrnuto je proporcionalna cijeni rješenja C, tj. duljini puta, a računa se pomoću sljedećeg izraza [6]

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

dok se ažuriranje feromona izvodi prema sljedećem izrazu [6]

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

Treba primijetiti kako jedan mrav na svaki brid kojim je prošao deponira istu količinu feromonu. Odnosno, količina koja se deponira na nekom bridu ovisi samo o duljini ukupnog puta.

Na kraju svake iteracije potrebno je ispariti određenu količinu feromonskih tragova. Iako se u literaturi spominje da isparavanje nema velik utjecaj kod stvarnih mrava, kod mravljih algoritama ono je potrebno kako bi kolonija zaboravila loša rješenja – posebice ona u početnim iteracijama [6]. Količina feromona nakon isparavanja računa se prema izrazu

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

gdje ρ predstavlja faktor isparavanja. Vrijednost parametara nalaze se u intervalu $[0, 1]$. Što je vrijednost ovog parametar bliža 0, to je manja količina feromona koja isparava pa mravi „brže uče“ jer količine feromona na tragovima brže rastu. Ako se vrijednost parametra p postavi preblizu 0, tada će mravlji algoritam brže konvergirati, ali će pretražiti manji prostor rješenja te će vjerojatno zapeti u lokalnom optimumu. Druga krajnost je postavljanje parametra na vrijednost koja je preblizu 1. Tada feromonski tragovi prebrzo isparavaju što odgađa konvergenciju algoritma. Treba imati na umu da isparavanjem količina feromona na bridovima pada eksponencijalno brzo. Tako će nakon x iteracija od inicijalizacije količina feromona na bridovima na kojima nisu deponirani feromoni biti jednaka $kol_{pocetna} \cdot (1 - \rho)^{x}$. Zbog toga se faktor isparavanja ρ postavlja na relativno malu vrijednost. Kod S-ACO algoritma postavlja se na $\rho = 0.01$. Za ovu vrijednost će nakon 10 iteracija oko 10% feromona ispariti. U slučaju da je $\rho = 0.1$, tada bi nakon 10 iteracija oko 65% feromona bilo ispareno što bi značajno otežalo formiranje kolektivnog znanja (vrijednosti u feromonskoj matrici) i narušilo bi rad algoritma [6].

2.4.2. Ant System

Ant System (u nastavku AS) predstavlja prvu je verzija ACO algoritama. Inicijalno su predstavljene tri različite verzije algoritma - *ant-density*, *ant-quntity* i *ant-cycle* [6]. Prve dvije verzije ažuriraju količine feromona prilikom svakog prelaska mrava na novi čvor, dok se kod *ant-cycle* feromonska matrica ažurira na kraju iteraciju - jednom kad su svi mravi konstruirali rješenje, a količina koja se deponira korelira s kvalitetom rješenja. Kad se govori o AS algoritmu misli se na upravo na *ant-cycle* verziju zbog boljih performansi u odnosu na preostale dvije [6].

Razlika između AS i S-ACO algoritma je dodatni član η u formuli za izračun vjerojatnosti p_{ij}^k. Član η predstavlja tzv. heurističku informaciju. Vjerojatnost p_{ij}^k da će mrav k iz čvora i krenuti prema čvoru j se kod AS-a računa pomoću sljedećeg izraza:

$$p_{ij}^k = \begin{cases} \dfrac{\tau_{ij}^{\alpha} \cdot {\eta_{ij}}^{\beta}}{\sum_{l \epsilon N_i^k} \tau_{il}^{\alpha} \cdot {\eta_{ij}}^{\beta}}, za\ j \in N_i^k \\ 0, \qquad za\ j \notin N_i^k \end{cases}$$

pri čemu se η_{ij} računa kao $\frac{1}{d_{ij}}$, gdje d_{ij} predstavlja duljinu puta između dva susjedna čvora i i j. Parametar β služi za podešavanja utjecaja duljine brida na rad algoritma. Novi član doprinosi tome da će AS, za razliku od S-ACO algoritma, češće birati čvorove koji su bliže trenutnom čvoru. Umnožak članova τ_{ij} i η_{ij} u određenoj mjeri kontrolira da algoritam ne zapne u lokalnom optimumu. Možda u grafu postoji brid relativno velike duljine u odnosu na ostale, ali je u kombinaciji s ostalim bridovima dio puta koji predstavlja globalno optimalno rješenje. Kroz određen broj iteracija količina feromona na tom bridu porast će dovoljno da član τ_{ij}, nadvlada član η_{ij}, što će dovesti do toga da će sve više mrava birati upravo taj brid i algoritam će sve brže konvergirati prema optimalnom rješenju. Druga moguća situacija je da graf sadrži brid mnogo manje duljine od svojih susjeda, ali je dio puta koji predstavlja loše rješenje. Kroz određen broj iteracija će se na tom bridu nalaziti mnogo manja količina feromona nego na njegovima susjedima pa će mravi s većom vjerojatnošću prolaziti kroz njih jer će njihov umnožak $\tau^{\alpha} \cdot \eta^{\beta}$ biti značajno veći.

Parametri α i β određuju ponašanje algoritma. Ako je $\alpha = 0$, tada je utjecaj feromona u potpunosti zanemaren. Time algoritma postaje previše pohlepan i uvijek će s velikom vjerojatnoću birati najkraće bridove. S druge strane, ako je $\beta = 0$, tada heuristička informacija nema nikakav utjecaj na odabir sljedećeg čvora, što može dovesti do prerane konvergencije – mravi prerano počinju slijediti jedan drugog, a duljina puta kojim prolaze može značajno odstupati od optimalne duljine [5]. Prema tablici koja se nalazi u literaturi, $\alpha = 1$ i $\beta \in [2,5]$ su dobre vrijednosti za AS algoritam [6].

Prilikom inicijalizacije grafa na sve se bridove deponira ista količina feromona koja se računa prema izrazu

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

gdje m predstavlja broj mrava, a C^{nn} predstavlja procjenu optimalnog rješenja generiranu primjerice algoritmom najbližih susjeda. Kod AS algoritma m je obično jednak broju gradova. To je količina koja je nešto veća od količine koja će biti deponira nakon prve iteracije i služi tome kako populacija mrava ne bi odmah bila usmjereni prema području koje je istraženo u prvoj iteraciji. S druge strane, kad bi se τ_0 postavio na preveliku vrijednost, tada bi se veliki broj početnih iteracija potrošio na lutanje mrava po prevelikom prostoru rješenja – tj. feromoni koje su mravi deponirali u početnim iteracijama imali bi premali utjecaj na odabir puta [5], [6].

Nakon što su svi mravi konstruirali rješenja, prvo se ispari određena količina feromona prema izrazu

$$\tau_{ij} \leftarrow (1-\rho) \cdot \tau_{ij}, \ \forall (i,j) \in L .$$

Dobra vrijednost za ρ kod *AS* algoritma iznosi 0.5 [5]. Nakon toga se vrednuju rješenja koja su mravi pronašli te mravi tada deponiraju određenu količinu feromona prema sljedećem izrazu

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

pri čemu je $\Delta\tau_{ij}$ [5], [6]

$$\Delta\tau_{ij} = \begin{cases} \frac{1}{C^k}, & za\ (i,j) \in T^k \\ 0, & za\ (i,j) \notin T^k \end{cases},$$

gdje T^k predstavlja put koji je konstruirao mrav k.

2.4.3. Max-Min Ant System

Na temelju AS-a osmišljeno je nekoliko novih algoritama. Tako je osmišljena elitistička verzija algoritma (eng. *Elitist Ant System*, EAS) kod koje se pamti globalno najbolje rješenje te se ono koristi prilikom ažuriranja količine feromona. Još jedna verzija je rangirajući sustava mrava (eng. *Rank-based Ant System*) kod kojeg se također pamti globalno najbolje rješenje, ali se u svakoj iteraciji uzima ω najboljih mrava koji se sortiraju i koji zajedno s globalnim rješenjem sudjeluju prilikom ažuriranja količine feromona [5], [6].

Pokazalo se je kako je dobro učiniti AS malo pohlepnijim kako bi se dobilo bolje rješenje. Jedan od najpoznatijih mravljih algoritama danas, *Max-Min Ant System* (*MMAS*) nastao je na temelju AS-a i prethodno spomenutih varijanti. Glavne razlike MMAS-a u odnosu na AS su [8], [9]:

- U svakoj iteraciji samo jedan mrav deponira feromone
- MMAS uvodi parametre τ_{min} i τ_{max} koji služe za ograničavanje minimalne i maksimalne količine feromona na bridovima

- U fazi inicijalizacije na svaki se brid deponira τ_{max} feromona

Kako originalni MMAS koristi samo jednog mrava prilikom ažuriranja količine feromona, u pravilu se koriste dvije strategije – ažuriranje najboljim mravom u iteraciji ili ažuriranje najboljim mravom do sad (onaj koji ima globalno najbolje rješenje). Ukoliko se koristi ažuriranje globalno najboljim rješenjem, ono se često kombinira s najboljim rješenjem u iteraciji i to na način da se svakih x iteracija dozvoli globalno najboljem mravu da ažurira feromone, dok u ostalim iteracijama to čini najbolji mrav u iteraciji. Ova metoda se koristi kako bi se izbjegla prerana stagnacija algoritma. Kod ažuriranja feromona isparavanje se odvija na isti način kao i kod AS algoritma, dok se deponiranje količine feromona odvija prema izrazu [8]

$$\tau_{ij}(\mathrm{t}) = \Delta\tau_{ij}^{k_{najbolji}}, \qquad \forall\,(i,j) \in T^{k_{najbolji}}$$

gdje $k_{najbolji}$ predstavlja globalno najboljeg mrava ili najboljeg mrava u iteraciji, a $T^{k_{najbolji}}$ turu koju je taj mrav konstruirao.

Zbog korištenja samo jednog mrava za ažuriranje feromonske matrice, autori ovog algoritma uveli su τ_{min} i τ_{max} parametre kako bi ograničili utjecaj jačine feromonskih tragova. U suprotnom bi algoritam prerano bi konvergirao. Količina feromona $\tau_{ij}(t)$ koja se nalazi na svakom bridu (i,j) u nekom trenutku je u granicama $\tau_{min} \leq \tau_{ij} \leq \tau_{max}$.
Granica τ_{max} može se izračunati prema sljedećem izrazu

$$\tau_{max} = \frac{1}{1-\rho} \cdot \frac{1}{L_{opt}}$$

gdje L_{opt} predstavlja duljinu optimalnog puta [7].

Donju granicu τ_{min} je nešto teže odrediti, a jedan od načina na koji se može odrediti je prema izrazu kojeg su predložili sami autori MMAS algoritma

$$\tau_{min} = \frac{\tau_{max} \cdot (1 - p^{dec})}{avg - p^{dec}}$$

gdje p^{dec} predstavlja vjerojatnost da će mrav svaki put odabrati onaj vrh/luk koji je dio optimalnog rješenja, a avg predstavlja prosječan broj vrhova koji su mravu na raspolaganju prilikom prelaska s jednog vrha na drugi. Za avg se uzima vrijednost $\frac{n}{2}$ gdje n predstavlja ukupan broj vrhova u grafu. Prethodni izraz došao je od izraza

$$p^{dec} = \frac{\tau_{max}}{\tau_{max} + avg \cdot \tau_{min}}$$

kojim se određuje vjerojatnost p^{dec} da će mrav odabrat sljedeći vrh koji je dio optimalnog rješenja. Izraz se temelji na činjenici da se najbolje rješenje uglavnom pronalazi netom prije stagnacije – situacije u kojoj su vrijednosti feromona na svim bridovima kandidatima (bridovi koji vode prema sljedećem čvoru) blizu ili jednake τ_{min}, dok se na bridu koju je dio optimalnog rješenja nalazi količina koja je blizu ili jednaka τ_{max}. Vjerojatno p^{dec} zapravo je drugačija na

svakom vrhu, ali su zbog pojednostavljenja autori algoritma uzeli da je ona ista na svakom vrhu. p^{dec} izračunava se na temelju vjerojatnost da će mrav odabrati optimalni put p^{best}. Vjerojatnost p^{best} da će mrav odabrati optimalnu put jednaka je vjerojatnosti da će mrav svaki put odabrati vrh/brid koji se nalazi na optimalnom putu. Kako za n vrhova mrav $n-1$ puta bira sljedeći vrh, p^{dec} se izračunava prema sljedećem izrazu

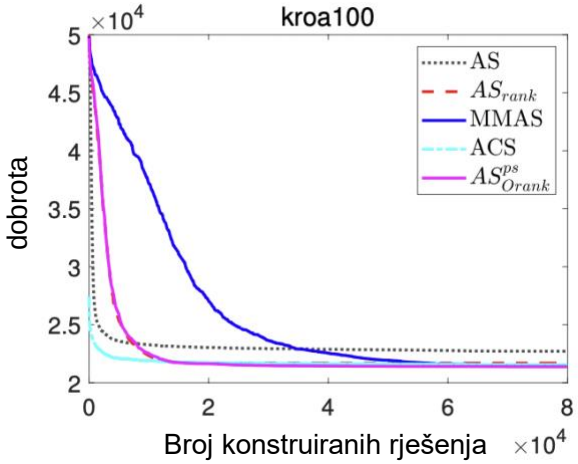
$$p^{best} = p_{dec}^{(n-1)}$$

$$p^{dec} = \sqrt[n-1]{p_{best}},$$

a pokazalo se je kako MMAS ostvaruje dobre performanse za $p_{best} = 0.05$ [8].

U fazi inicijalizacije se na svaki brid deponira količina feromona koja je jednaka τ_{max}. Ovakva početna konfiguracija ima smisla kod MMAS-a jer u svakoj iteraciji samo jedan mrav deponira feromone. Zbog toga će na kraju svake iteracije na svim bridovima količina feromona malo ispariti, dok će na onima na kojima mrav deponira feromone količina porasti. Ovime se zapravo omogućuje pretraživanje većeg prostora rješenja u početnim iteracijama.

Prema testiranjima koje su obavili autori algoritma, MMAS je u prosjeku pokazao bolje performanse u odnosu na AS i EAS [8]. Sljedeća slika objašnjava zašto MMAS generalno daje bolje rješenje od konkurentnih algoritama.



Slika 9. Usporedba kvalitete rješenja ACO algoritama (Izvor *[10])*

MMAS za razliku od AS-a, ACS-a i rangirajućeg mravljeg algoritma (AS_{rank}) zbog τ_{min} i τ_{max} parametara i načina na koji se inicijaliziraju feromonski tragovi odgađa trenutak konvergencije i pretražuje širi prostor rješenja. Pretraga šireg prostora na početku daje kvalitetnije rješenje na kraju [10]. Treba još spomenuti kako se performanse MMAS-a mogu dodatno unaprijediti primjerice korištenjem mehanizma izglađivanja tragova (eng. *trail smoothing*) i korištenjem algoritama lokalne pretrage (eng. *local search*).

2.4.4. Three Bound Ant System

Three Bound Ant System (TBAS) jedan je od noviji mravljih algoritama koji se temelju na MMAS-u. Nekoliko je značajnih razlika između TBAS i MMAS algoritma [11]:

- TBAS koristi 3 umjesto 2 granice – τ_{LB} i τ_{UB} (koje imaju istu svrhu kao i τ_{min}, τ_{max} kod MMAS-a) te τ_{CB} kojim se kontrolira potiskivanje (isparavanje) feromona
- koristi se povremeno potiskivanje umjesto konstantnog isparavanja
- TBAS koristi drugačiju proceduru prilikom isparavanja feromona, dok u fazi deponiranja koristi novi parametar ω za izračun količine feromona za deponiranje
- na sve se tragove inicijalno postavlja τ_{LB} feromona (umjesto τ_{max} kod MMAS-a)

TBAS koristi gotovo sve parametre koje koristi i MMAS, ali se ρ koristi drugačije. Za razliku od MMAS-a gdje se koristi prilikom isparavanje, kod TBAS algoritma on se koristi za izračun i ažuriranje vrijednosti Q_i (koja se koristi prilikom deponiranja feromona) i ω parametara čija se vrijednost postavlja na $1 - \rho$. Iako TBAS koristi τ_{LB}i τ_{UB} za gornju i donju granicu, i dalje se koriste τ_{min} i τ_{max} vrijednosti kako bi se izračunale granice τ_{LB} i τ_{UB}.

Za TBAS vrijedi $\tau_{LB} = \tau_{max}$, a vrijedi i sljedeće

$$\theta = \frac{\tau_{LB}}{\tau_{UB}} = \frac{\tau_{min}}{\tau_{max}}.$$

Pomoću omjera $\theta = \frac{\tau_{min}}{\tau_{max}}$ može se odrediti granica τ_{UB} prema sljedećem izrazu

$$\tau_{UB} = \frac{\tau_{LB}}{\theta}.$$

Faza deponiranja/pojačavanja feromona slična je kao i kod MMAS-a i razlikuje se samo u načinu na koji se izračunava količina koja će se deponirati. Izraz za izračun količine je sljedeći

$$\Delta\tau_{ij} = \frac{Q_i}{C^{k_{najbolji}}},$$

gdje se parametar Q_i inicijalno postavlja na 1 i zatim se u svakoj iteraciji njegova vrijednost ažurira prema izrazu [11]

$$Q_{i+1} = \frac{Q_i}{1-\rho}.$$

Kao i kod MMAS-a, TBAS također koristi najboljeg ili najbolje mrave u ovoj fazi. Tako se prilikom deponiranja može koristi najbolji mrav u iteraciji, najbolji mrav do sad, a mogu se koristiti i metode poput *k-best* i *max-k-best* [11].

Faza kontrakcije je novost u odnosu na MMAS, ali ima istu ulogu kao i faza isparavanja kod MMAS-a. Međutim, za razliku od stalnog isparavanja koje se izvodi u svakoj iteraciji, kod TBAS-a se kontrakcija događa samo ako je zadovoljen uvjet. Ova faza se može opisati sljedećim pseudokodom

```
ako maks_vrijednost_feromona > TR_UB
      omega_crtano = TR_UB / maks_vrijednost_feromona * omega
```

```
za svaki brid u feromonskoj matrici
      količina_feromona = količina_feromona * omega_crtano
Qᵢ = Qᵢ * omega_crtano
```

gdje `maks_vrijednost_feromona` predstavlja trenutno maksimalnu količinu feromona (uzimajući sve bridove u obzir). Na primjeru *kroA100* TSP problema za 3000 iteracija kontrakcija se izvodi u približno 70% iteracija. Zbog ovog je TBAS nešto brži od MMAS algoritma jer u početnim iteracijama nema isparavanja koje se zbog iteriranja kroz feromonsku matricu odvija u složenosti $O(n^2)$.

3. Raspodijeljena implementacija

3.1. Razlozi za raspodijeljenu implementaciju

Velik broj današnjih računala dolazi s višejezgrenim procesorima. Glavni razlog za to proizlazi iz sljedećeg izraza

$$P = C \cdot U^2 \cdot f$$

gdje P predstavlja snagu procesora koji radi na radnom taktu f, radnom naponu U te gdje je ukupni kapacitet njegovih komponenata i veza između njih jedak C [12], [13]. Ako se želi smanjiti frekvencija i zadržati propusnost, to se može ostvariti uvođenjem novih procesorskih jezgri. Smanjenje frekvencije f k puta i povećanje broja jezgri k puta bi trebalo osigurati približno istu propusnost. Ako se ovo preslika na slučaj jednojezgrene i dvojezgerne arkture, snaga P_2 u slučaju dvojezgrene

$$C_2 = 2.2C_1$$
$$U_2 = 0.6U_1$$
$$f_2 = 0.5f_1$$
$$P_2 = 0.396P_1$$

približno 60% manja od snage jednojezgrene arhitekture P_1 [12]. Ovo je prije nekoliko desetljeća bio pokazatelj kako će se višejezgrene arhitekture zadržati i postati prisutnije u nadolazećim godinama. U svijetu softvera to znači kako je u slučaju računalno zahtjevnih algoritama potrebno uložiti dodatan trud kako bi se identificirali dijelovi koda koji se mogu paralelizirati te je na kraju potrebno i isprogramirati rješenje koje je paralelno.

Paralelna implementacija također ima svoja ograničenja, a jedno od njih je broj procesorski jezgri koje su dostupne na računalu. Moderna osobna računala imaju najčešće 4-8 jezgri što predstavlja gornju granicu u pogledu broja dretvi koje se mogu izvršavati istovremeno. Naravno da se, ako procesor ima N jezgri, program može napraviti na način da on koristi i više od N dretvi, ali time se paralelna implementacija pretvara u djelomično konkurentnu jer se samo N dretvi može istovremeno izvršavati - vrijeme izvršavanja algoritma

u tom se slučaju neće smanjiti. Jedno rješenje za ovaj problem je vertikalno skaliranje, tj. korištenje procesora s većim brojem jezgri. Drugo rješenje je horizontalno skaliranje, tj. implementacija algoritama na način da se oni mogu distribuirani na više računala. Fokus ovog rada je upravo distribuirana odnosno raspodijeljena implementacija.

Distribuirani računalni sustavi su sustavi koje prosječan korisnik danas često koristi. Dobar primjer je Web. Poslužiteljska strana Weba (eng. *backend*) često je implementirana na distribuirani način pa se tako umjesto jednog poslužitelja koristi njih nekoliko kako bi se ostvarila funkcija cjeline. Primjerice, umjesto jednog često se koristi nekoliko *API* čvorova (računala koja poslužuju krajnje točke (eng. *endpoint*)), nekoliko čvorova koji su zaduženi za periodičke procese (npr. kreiranje izvještaja, čišćenje baze podataka), zatim sustav za raspoređivanje opterećenja (eng. *load balancer*) te baza podataka. U posljednjih nekoliko godina sve više se implementiraju sustavi u obliku mikroservisa gdje je osnovna ideja podijeliti monolitne sustave na nekoliko manjih koji međusobno komuniciraju brzim protokolima kao što je *gRPC*.

Glavne karakteristike distribuiranih sustava su [14]:

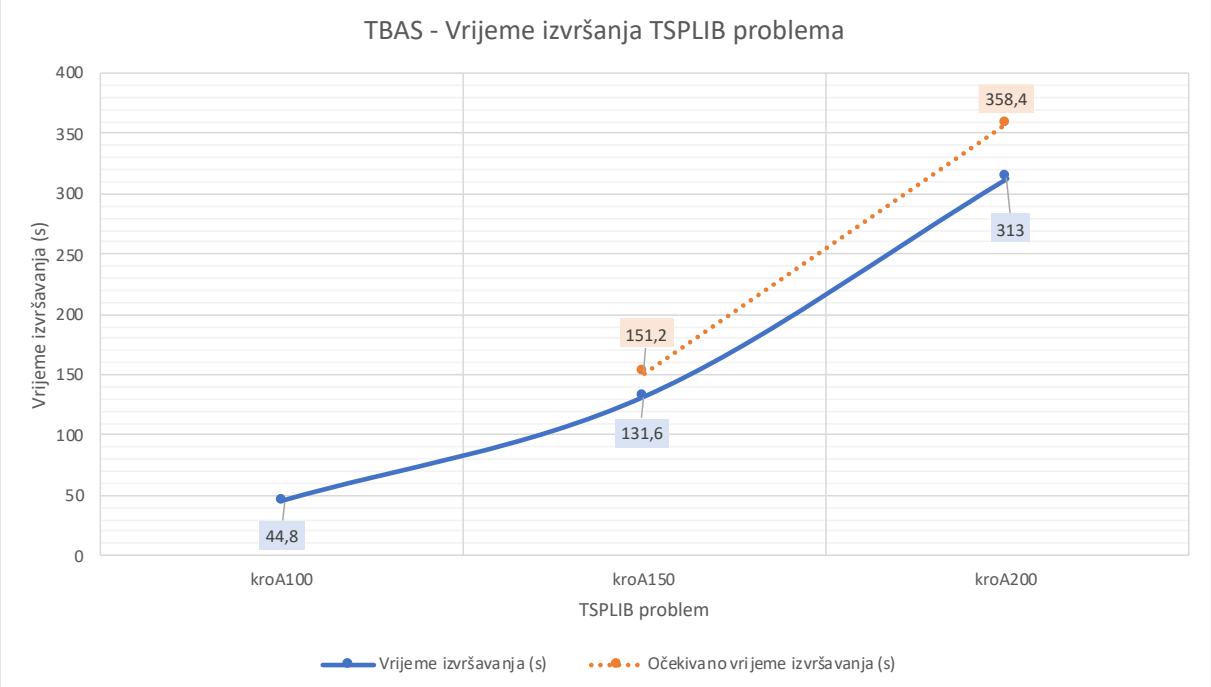
- Funkcionalno odvajanje – na temelju funkcionalnosti, mogućnosti i svrhe sustava
- Inherentna distribucija – sustav koji je, na temelju funkcionalnosti ili algoritama, prirodno takav da je pogodan za distribuciju
- Pouzdanost – očuvanje i replikacija podataka, upravljanje ispadima sustava
- Skalabilnost – povećanje resursa u svrhu povećanja performansi ili dostupnosti sustava
- Ekonomičnost – dijeljenje resursa između više entiteta kako bi se smanjio trošak vlasništa (eng. *cost of ownership*)

U nekim literaturama su performanse izdvojene kao posebna karakteristika, dok su ovdje spomenute u sklopu skalabilnosti. U nastavku ovog rada će se performanse i skalabilnost razmatrati kao dva odvojena pojma. Performanse će se odnositi na vrijeme izvršavanja algoritma i kvalitetu rješenja, dok će se skalabilnost odnositi na mogućnost distribucije algoritma na veći broj računala u svrhu povećanja performanse.

Ako se distribuirani sustav usporedi s paralelnim implementacijom koja se izvodi samo na jednom računalu, tada mu je osnovni nedostatak velika ovisnost o mreži, dok se kod paralelne varijante se osim mreže mogu koristiti i dijeljeni memorijski adresni prostor, datoteka, cjevovod te red poruka pomoću kojih se može ostvariti mnogo brža komunikacija i manje kašnjenje. Ovisnost o računalnoj mreži povećava komunikacijske troškove (povećano opterećenja sustava i/ili vrijeme izvršavanja algoritma), latenciju, uvodi dodatne sigurnosne probleme, a može i u potpunosti onemogućiti rad takvog sustava u slučaju mrežnog ispada.

Mrežna komunikacija može i usporiti rad algoritma zbog poslova serijalizacije, slanja, deserijalizacije i primanja mrežnih paketa [5].

Jedan od problema ACO algoritama poput AS-a, MMAS-a i TBAS-a je relativno dugo vrijeme izvršavanja algoritma. U svakoj iteraciji m mrava konstruira turu u vremenskoj složenosti $O(n^2)$, što vodi do ukupne složenosti od $O(m \cdot n^2)$. Kako je u slijednima varijantama algoritama bez korištenja algoritama lokalnih pretraga $m = n$, ukupna složenost svake iteracije je $O(n^3)$ [5]. Broj iteracija koji se koristi ovisi o parametrima algoritma, korištenim optimizacijama, broju gradova i broju mrava, a često se nalazi u rasponu od nekoliko stotina do nekoliko tisuća. Na primjeru TSP problema od 100 gradova, vrijeme algoritma se mjeri u desecima sekundi, dok se za probleme s 200 i više gradova vrijeme najčešće mjeri u minutama. U nastavku se nalazi graf koji prikazuje vrijeme izvršavanja slijednog TBAS algoritma za TSP probleme sa 100, 150 i 200 gradova. U svim slučajevima korišteno je 3000 iteracija, a broj mrava odgovara broju gradova. Za svaki problem izvršeno je 5 mjerenja, a prikazane vrijednosti u grafu predstavljaju aritmetičku sredinu. Vidljivo je kako porast izmjerenog vremena odgovara porastu očekivanog vremena izvršavanja koje je izračunato na temelju složenosti algoritma – za oba problema je odstupanje od očekivanog vremena približno 13%. Na temelju ovih mjerenja i odstupanja od očekivanog vremena, vrijeme izvršavanja za problem od 400 gradova bi iznosilo otprilike 40 minuta, dok bi za problem od 800 gradova iznosilo približno 5 sati i 30 minuta.



Slika 10. TBAS - porast vremena izvršavanja algoritma

Ovako veliko vrijeme izvršavanja je uglavnom posljedica korištenja algoritma koji sekvencijalno simulira ponašanje mrava koje je inherentno paralelno [8], [15] – u prirodi mravi istovremeno konstruiraju put do hrane. Zbog toga su ACO algoritmi pogodni za paralelnu implementaciju.

Još jedan razlog paralelizacije ACO algoritma je pronalaženje kvalitetnijeg rješenja i povećanje robusnosti algoritma (smanjiti vjerojatnost ostanka u lošijim područjima). U tom slučaju se često koriste modeli s više populacija (eng. *multicolony*) kako bi se pronašlo što bolje rješenje. Stützle [16], ujedno i autor MMAS algoritma, autor je jednog od prvih radova koji se bavi tom tematikom. Ovaj rad opisuje *paralelne nezavisne primjerke algoritma* (eng. *parallel independent run - PIR*) gdje se istovremeno pokreće nekoliko instanci algoritama te se na kraju uzima najbolje rješenje svih instanci. To je najjednostavnija strategija paralelizacije jer algoritmi ne koriste nikakav oblik komunikacije, a već se je i takav oblik pokazao učinkovit - osnovna ideja je povećanje vjerojatnosti pronalaska zadovoljavajućeg rješenja. Međutim, ovaj pristup ne rješava problem stagnacije i zapinjanja u lokalnog optimumom. Zato je dobro uvesti nekakav oblik komunikacije između algoritama gdje će oni zajedničkom suradnjom doprinijeti smanjenju vjerojatnosti da zapnu u lokalnom optimumu. Michel i Middendorf [17] autori su jednog od prvih radova koji opisuje implementaciju ACO algoritma korištenjem otočnog modela (eng. *island model*) koji je često korišten model kod paralelnog genetskog algoritam. Otočni model podrazumijeva korištenje nekoliko subpopulacija (otoka) koje periodično, svakih nekoliko iteracija razmjenjuju svoja (najbolja) rješenja. Kasnije je ovakav model primijenjen i za rješavanje TSP problema korištenjem različitih migracijskih topologija [18]. U međuvremenu je objavljeno još nekoliko radova koji se bave višepopulacijskim paralelnim pristupom [19], [20].

3.2. Taksonomija i modeli raspodijele

Prvi korak u izgradnji distribuiranog algoritma je odabir modela. Jedna od podjela modela nalazi se u nastavku [21]:

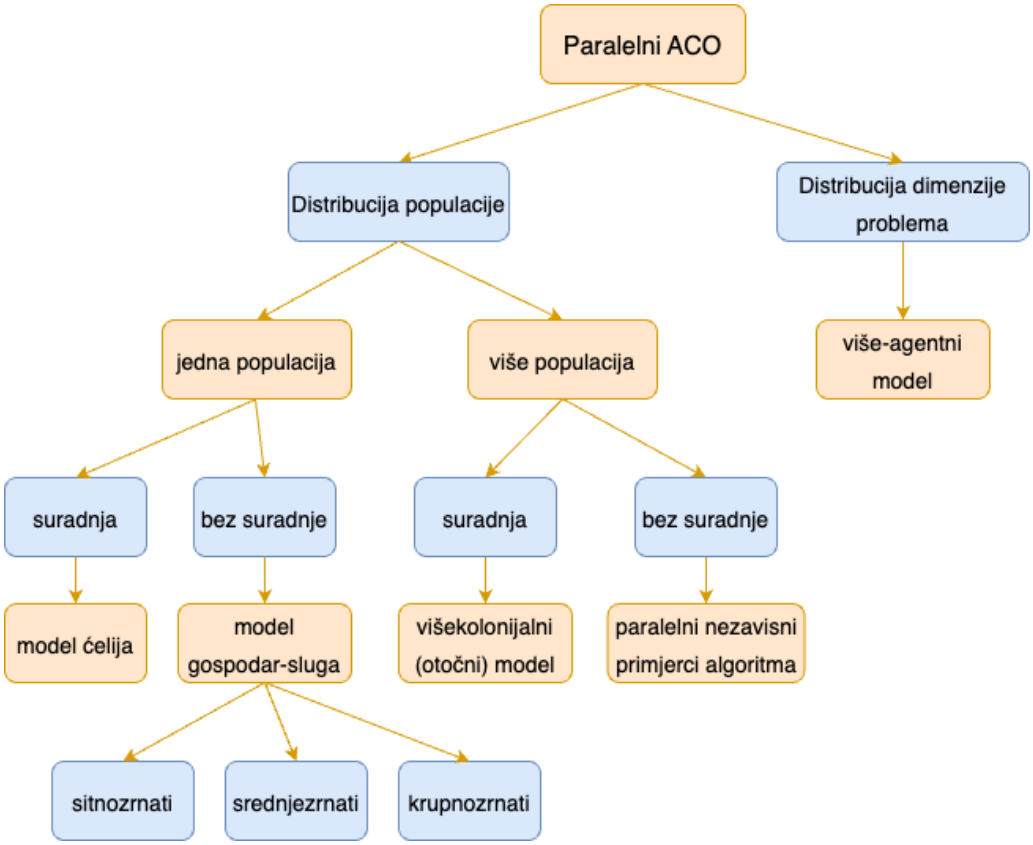
- Krupnozrnata podjela – podjela populacije na nekoliko subpopulacija (primjer je otočni model)
- Sitnozrnata podjela – podjela populacija na vrlo male subpopulacije, najčešće veličine jedne jedinke (primjer je model ćelije)
- Jedna populacija s paraleliziranim operatorima – umjesto podjele populacije na manje populacije, paraleliziraju se pojedine operacije (evaluacija rješenja, konstrukcija rješenja, korištenje lokalne pretrage [16]) ili operatori korištenjem modela gospodar-sluga (eng. *master-slave model*)

Čupić [5] navodi nešto drugačiju podjela, ali ona u sebi zapravo sadrži iste kategorije kao i podjela iznad:

- Paralelizacija na razini algoritma – istovremeno izvođenje više instanci algoritma; ovdje spadaju krupnozrnati i sitnozrnati modeli
    - Nesuradna paralelizacija – primjerci algoritama ne razmjenjuju informacije (primjer je PIR)
    - Suradna paralelizacija – primjerci algoritama razmjenjuju informacije (primjer su otočni model i model ćelije)
- Paralelizacija na razini populacije – jednopopulacijski model, cilj je omogućiti bržu izmjenu populacije (paralelizacija izgradnje jedinke/rješenja)
- Paralelizacija na razini iteracije algoritma – jednopopulacijski model, čitav algoritam se odvija slijedno, a paraleliziraju se operatori (npr. operatori križanja, mutacije, vrednovanja rješenja, paralelizacija operacija u kojima se ažuriraju feromonski tragovi)

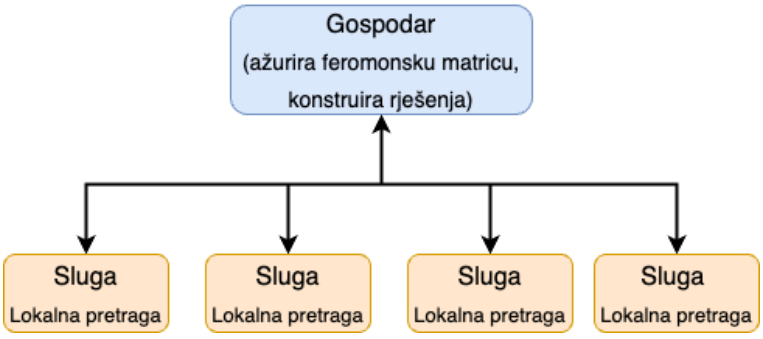
Kad je riječ o distribuiranim evolucijskim algoritmima, koristi se paralelizacija na razini algoritama i to najčešće krupnozrnati modeli jer je kod njih trošak mrežne komunikacije značajno manji nego kod sitnozrnatih modela.

U nastavku se nalazi taksonomija paralelnih ACO algoritama. Podjela je napravljena na temelju Pedemonte, Nesmachnow, Cancela [15] i Gong *et al.* [22], a u osnovni se modeli dijele na distribuciju populacije ili dimenzije problema. Kod distribucije populacije se jedna populacija dijeli na nekoliko subpopulacija, a njihov broj ovisi o modelu i konkretnoj implementaciju. Kod distribucije dimenzije problema čitav se problem dekomponira na manje dijelove koji se potom distribuiraju. Kod ACO algoritama to primjerice znači da mrav na jednom procesoru ne radi nad cijelom matricom udaljenost, nego samo nad dijelom te matrice.

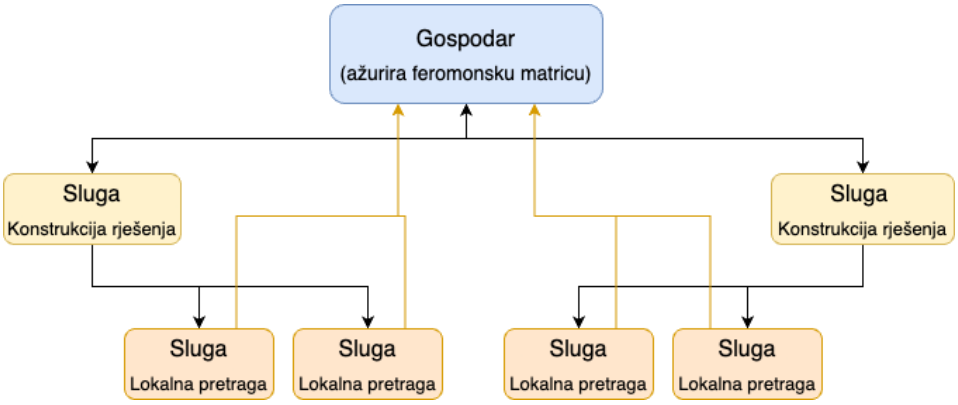


Slika 11. Hijerarhijski pregled taksonomije distribuiranih ACO modela

Model gospodar-sluga predstavlja hijerarhijski model. Model se sastoji od gospodara koji ima matricu feromona i koordinira sluge koji se najčešće bave poslovima konstrukcije rješenja, iako postoje i pristupi u kojima su sluge iskorišteni primjerice za lokalnu pretragu. Ovaj model je često korišten u paralelnim implementacijama koje se izvode na jednom računalu zbog dijeljene memorije pa se ažuriranje feromonske matrice može realizirati vrlo brzo. Model se dijeli na tri granulacije - sitno, srednje i krupnozrnatu, a granulacije predstavljaju količinu posla kojeg jedan sluga obavlja [15].



Slika 12. Ilustracija modela gospodar-sluga - prvi pristup (Prema [16])



Slika 13. Ilustracija modela gospodar-sluga - drugi pristup (Prema *[16]*)

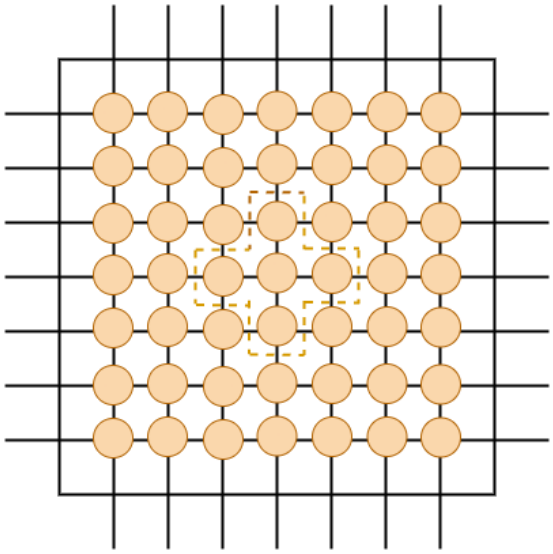
Kod krupnozrnate varijante modela sluge obavljaju posao cjelokupne konstrukcije, evaluacije i slanja rješenja gospodaru, a gospodar održava matricu feromona. Svakom slugi je dodijeljen jedan ili grupa mrava te oni najčešće sinkrono pristupanju matrici feromona. Dodatne optimizacije poput korištenja algoritama lokalne pretrage također obavljaju sluge. Ovo je ujedno i najčešće korištena varijanta zbog relativno jednostavne implementacije i dobrih performansi. Model se može iskoristi za vrlo skalabilne implementacije, a pokazalo se je kako je u tom slučaju dobro uvesti nekoliko podgospodara kako bi se stvorila hijerarhijska organizacije i smanjio komunikacijski trošak [15].

Kod srednjezrnate varijante problem se dekomponira na manje dijelove. Sluge umjesto potpunih pronalaze parcijalna rješenja koja potom šalju gospodaru koji ih sastavlja u cjelovita [15]. Za isti problem srednjezrnata implementacija u pravilu koristi više slugu nego krupnozranta. Zbog ovog je i komunikacijski trošak veći što je jedan od glavnih nedostataka ovog modela.

Kod sitnozrnate implementacije sluge procesiraju vrlo male zadatke poput odabir pojedinih komponenti rješenja (npr. pojedinih bridova u TSP-u) koje potom šalju gospodaru [15]. Najveći nedostatak ovog modela je učestala komunikacija što često predstavlja uskoro-grlo i značajno utječe na performanse.

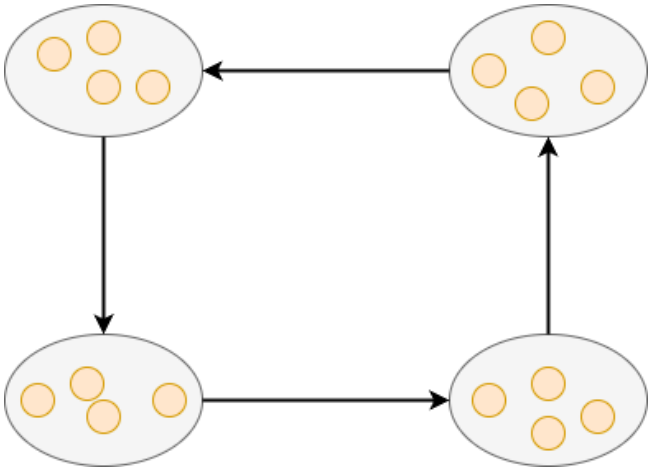
Model ćelija ili celularni model (eng. *celluar model*) spada u sitnozrnate modele, a podrazumijeva jednu populaciju čije su jedinke raspoređene u mrežu - idealno jedna ćelija (jedinka) po procesoru. Kod ovog modela ćelije komuniciraju samo s onim ćelijama koje su u njihovom susjedstvu, a način komunikacije u susjedstvu je definiran korištenom topologijom. Slika ispod prikazuje ilustraciju ovog modela. Isprekidanim linijama označeno je susjedstvo ćelije koja se nalazi u centru. Nedostatci ovog modela su veliki komunikacijski troškovi s obzirom da se distribuiraju pojedine jedinke. Također, za razliku od otočnog model, kod

celularnog nema unutar-ćelijske evolucije kao što je to unutar-otočna (eng. *inter-island*) kod otočnog [23], a pokazalo se kako je prednost otočnog modela upravo postojanja različitih populacija koje se razlikuju po kolektivnom znanju, gdje ta raznolikost na kraju rezultira kvalitetnijim rješenjem jer takvi modeli pretražuju širi prostor rješenja.



Slika 14. Ilustracija celularnog modela

Višekolonijalni model podrazumijeva nekoliko kolonija koje pretražuju prostor rješenja, a svaka do njih ima svoju matricu feromona. Informacije između kolonija razmjenjuju se periodično ili kad se za to pokaže potreba. Model spada u krupnozrnate modele i istovjetan je otočnom modelu često korištenom kod paralelnog genetskog algoritma.



Slika 15. Ilustracija višekolonijalnog modela

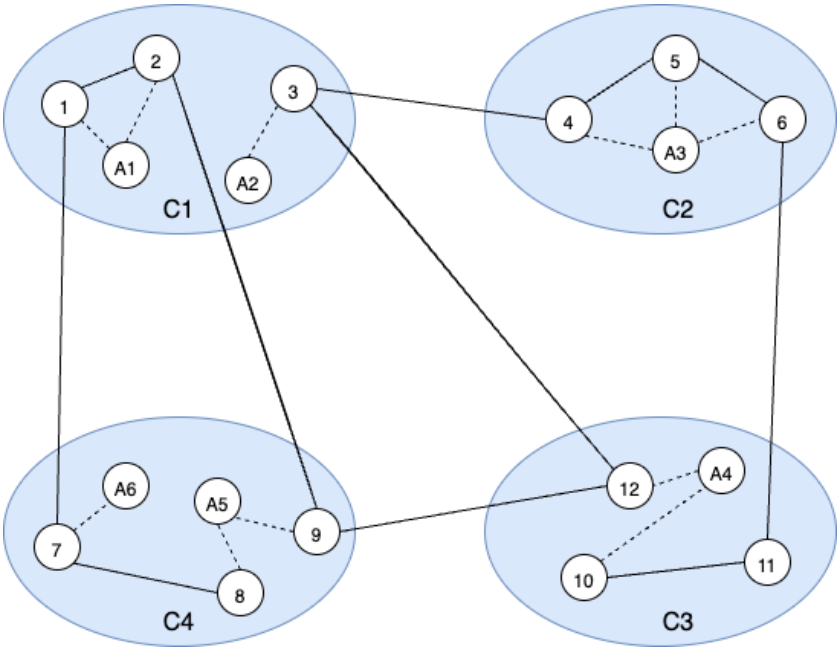
Randall i Lewis [24] opisali su model paralelne interaktivne kolonije mrava (eng. *parallel interacting ant colony*) – varijante višekolonijalnog modela kod koje trenutno najbolja kolonija šalje svoju feromonsku matricu svim ostalima. U istom radu su autori spomenuli kako se model može primijeniti na probleme manjih dimenzija, ali nije pogodan za TSP probleme zbog velikih

komunikacijskih troškova. Zato kolonije najčešće razmjenjuju najbolja rješenja (mrave). Kod razmjene rješenja može se primjerice koristit globalno najbolje rješenje, lokalno najbolje rješenje, k-najboljih rješenja do sad, a mogu se i kombinirati prethodne dvije metode. Od nabrojanih, pokazalo se je kako su najbolji rezultati ostvareni slanjem lokalno najboljih mrava [18]. Naravno, mogu se koristiti i drugačije razmjene, a jedna od njih je utvrđivanje najboljeg rješenja u susjedstvu i slanje takvog rješenja svim kolonijama u susjedstvu [25]. Kao i kod otočnog modela, inter-kolonijalna evolucija pozitivno utječe na raznolikost između kolonija što omogućuje pretraživanje šireg prostora rješenja i rezultira kvalitetnijem rješenjem. Na raznolikost između kolonija uvelike utječe učestalost komunikacije kao i korištena komunikacijska topologija. Pokazalo se je kako je dobro spriječiti pretjeranu komunikaciju između kolonija kako bi se održala raznolikost između kolonija, što rezultira pretragom šireg prostora rješenja te daje kvalitetne finalno rješenje. Kako bi se dodatno povećale performanse, predlaže se implementacija adaptivnih modela u kojim se komunikacijska topologija te učestalost komunikacije mijenjaju tijekom izvođenja algoritma te na temelju specifičnog problema [15]. Zbog jednostavne implementacije i dobre kvalitete rješenja, ovo je jedan od najčešće korištenih modela.

Ovaj model se kao i ostali spomenuti modeli može dodatno okarakterizirati kao homogeni ili heterogeni te sinkroni ili asinkroni. Kod homogenog pristupa svi procesi, tj. kolonije koriste iste parametre. Ovaj pristup je češće korišten od heterogenog, ali ima dva nedostatka. Ako se kolonije izvode na računalnim sustavima koji su heterogeni (npr. distribuirano istovremeno izvođenje algoritma na mobilnom uređaju te stalnom računalnu), to će smanjiti učinkovitost i efikasnost algoritma jer će uglavnom samo brži uređaji doprinositi pronalasku novih, kvalitetnijih rješenja [22]. Drugi nedostatak je da ovaj pristup možda neće najbolje uravnotežiti globalno pretraživanje i lokalnu eksploataciju [22]. Drugim riječima, da bi se dobilo kvalitetno rješenje, poželjno je pretražiti širi prostor rješenja, a jednom kad se pronađu dobri kandidati tada je dobro imati fokusiraniju pretragu je se optimalno rješenje često nalazi blizu onih koja su jako dobra. Kod heterogenog modela različite kolonije imaju različite parametre ili neke druge karakteristike. Tako je moguće koristiti heterogeni model koji prilagođava parametre, primjerice faktor isparavanja, prema uređaju na kojem se izvodi kako bi se ujednačio trenutak stagnacije u slučaju distribucije algoritma na različite uređaje. Moguće je i kombinirati drugačije algoritme, primjerice TBAS i MMAS [23]. Moguće je koristiti i različite parametre na način da se određen broj kolonija više fokusira na pretragu šireg prostora rješenja, dok je kod ostalih veći fokus na lokalnoj pretrazi. Sinkroni i asinkroni pristup se odnosi na način razmjene rješenja. Ako se rješenja razmjenjuju periodično, svakih nekoliko iteracija, tada je riječ o sinkronom modelu, dok se kod asinkronog modela rješenja razmjenjuju kad za

to postoji potreba [22]. Primjerice, kod asinkronog modela kolonija može poslati svoje dosad najbolje rješenje samo onda kad pronađe novo najbolje rješenje.

Više-agenti model (eng. multi-agent model) zasniva se na više-agentnoj paradigmi. Softverski agent je računalni sustav ili dio sustava koji ima unutarnje stanje kojeg mijenja na temelju promatranja i interakcije s okolinom. Temeljem unutarnjeg stanja agent izvodi akcije koje mijenjaju okolinu. Više-agentni sustavi podrazumijevaju više agenata koji u međusobnoj interakciji pokušavaju riješiti zajednički problem. Ako se zanemare implementacijski detalji tada ovaj model predstavlja prirodni model distribucije ACO algoritma. Naime, kod višekolonijalnog modela slijedni se algoritam distribuira na više računala, što predstavlja i svojevrsno ograničenje u pogledu ubrzanja koje se može ostvariti. Više-agentni model podrazumijeva distribuciju cjelokupne dimenzije problema. Jedan takav više-agentni model je ACODA [26]. Kod ovog modela svaki agent održava samo dio TSP gradova i s njima povezanih podataka kao što su matrica feromona i matrica udaljenosti, a mravi predstavljaju mobilne agente koji kao mrežni objekti migriraju od jednog agenta prema drugom.



Slika 16. Ilustracija više-agentnog modela (Prema *[26]*)

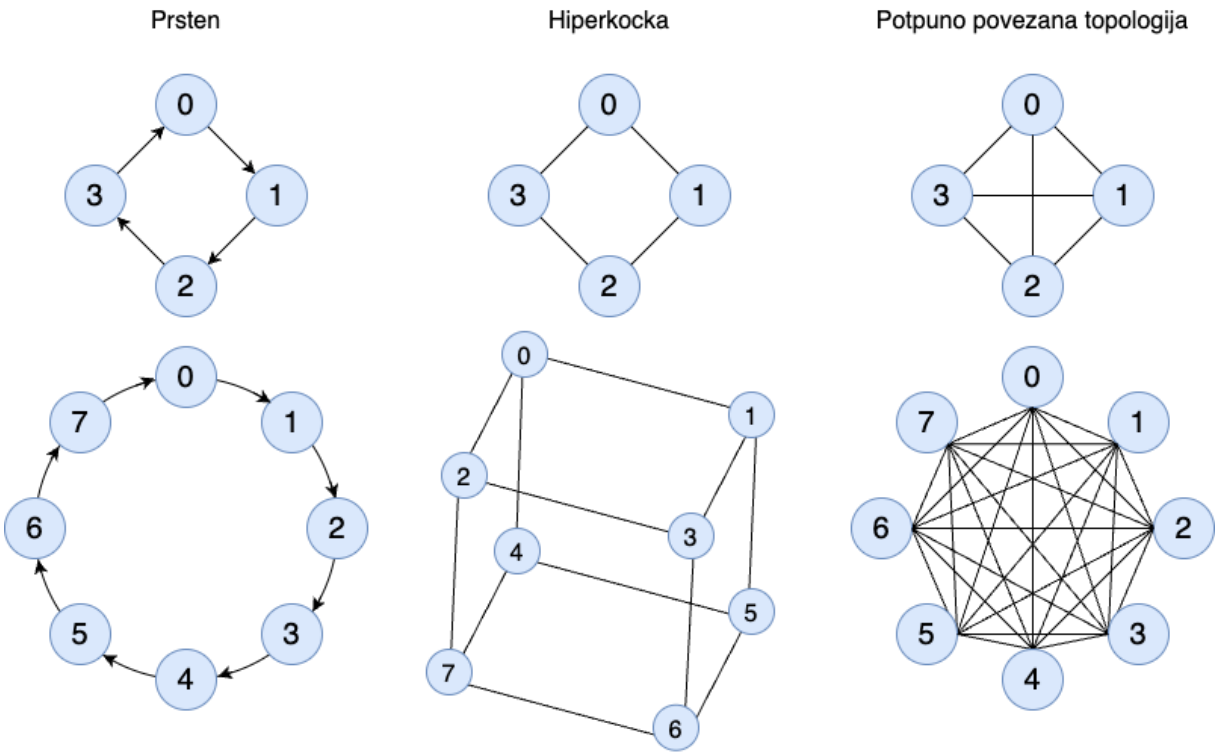
U ovom modelu svako računalo pokreće jednog ili više agenta. Svaki agent ima: faktor isparavanja, cijenu najboljeg rješenja (na temelju mrava koji su prošli kroz njegove čvorove), listu čvorova i njima susjednih čvorova uključujući količinu feromona koja je deponirana na pojedinim bridovima i mapu koja za svaki čvor sadrži identifikator agenta koji njime upravlja. Svaki mrav ima informaciju o svojem najboljem rješenju do sad. Kad mrav dođe do čvora, agent koji upravlja tim čvorom ažurira informaciju o cijeni najboljeg rješenja. Ako je taj čvor posljednji čvor u turi, tada agent postavlja zastavicu vraćanja, šalje ovu informaciju ostalim agentima i ažurira feromonsku matricu. Kad mrav dođe do mravinjaka (ishodišnog čvora),

agent ga reinicijalizira i mrav ponavlja proces izgradnje rješenja. Početnu populaciju mrava generira svaki čvor. U radu se spominje konfiguracija za TSP gdje se generira jedan mrav po čvoru [26].

Model je testiran na nekoliko TSP problema. Pokazano se je da povećanje broja računala i povećanja čvorova kojima upravlja jedan mrav daje dobre rezultate i smanjuje vrijeme konstrukcije jedne populacije rješenja. Razlog za to je da, u slučaju distribucije na veći broj računala, povećanje broja čvorova kojima upravlja jedan agent utječe na smanjenje mrežnih troškova jer u tom slučaju veći broj mrava migrira na lokalnom području (unutar agenta na jednom računalu). Najbolji rezultat za TSP problem veličine 666 gradova je ostvaren skaliranjem na 11 računala. Nakon toga je komunikacijski trošak postao prevelik i vrijeme izvršavanja je počelo rasti [26].

3.3. Topologije i migracijske strategije

Topologija se odnosi na logički raspored čvorova u komunikacijskoj mreži i usko je povezana s distribuiranim modelima jer oni podrazumijevaju razmjenu informacija između čvorova korištenjem računalne mreže. Postoji nekoliko topologija koje su često korištene kod evolucijskih i ACO algoritama, a to su popuno povezana topolgija, prsten i hiperkocka (eng. *hipercube*) [18], [27]. Kod otočnog modela spominje se još i torus [22].



Slika 17. Komunikacijske topologije

U potpuno povezanoj topologiji svaki čvor ima direktnu komunikacijsku vezu sa svim ostalim čvorovima. Ovakva topologija je prikladna u situacijama kad je potrebno poslati informaciju svim ostalim čvorovima u jednom koraku. Prednost ove topologije je što smetnje ili ispad komunikacijske veze između jednog para čvorova ne blokira komunikaciju između ostalih čvorova [28]. Nedostatak je velik broj veza između čvorova te povećani komunikacijski trošak.

Topologija prstena predstavlja suprotnost potpuno povezanoj topologiji. Ovdje svaki čvor komunicira sa svojim sljedbenikom kojem šalje informacije i svojim prethodnikom od kojeg prima informacije. Prednost ove topologije je mali broj komunikacijskih veza i mali komunikacijski trošak. Nedostaci su sporo propagiranje informacije svim čvorovima u mreži - potrebno je k koraka da se informacija propagira od prvog do k-tog čvora. Također, nakon mrežnog ispad jedne komunikacijske poveznice propagacija informacije po cijelom prstenu više nije moguće (iznimka vrijedi za čvor koji se nalazi na odredišnoj strani poveznice koja je blokirana). Postoji nekoliko varijanti ove topologije poput usmjerenog i neusmjerenog prstena (svaki čvor može slati informaciju sljedbeniku i prethodniku), a moguća je i topologija ugniježđenih prstena koja predstavlja prsten prstenova [5].

Hiperkocka spada u klasu $k - d$ mreže (eng. $k - d$ *mesh*) – klasa topologija koje se sastoje od d dimenzija i k čvorova po svakoj dimenziji. Hiperkocka se sastoji od dva čvora po svakoj dimenziji i predstavlja $2 - d$ mrežu. 0-dimenzionalna hiperkocka sastoji se od jednog čvora, jednodimenzionalna od dva, dvodimenzionalna od četiri, a trodimenzionalna od osam čvorova. Općenito, d-dimenzionalna hiperkocka konstruira se spajanjem odgovarajućih čvorova dvije $(d-1)$-dimenzionalna hiperkocke. Svaki čvor je povezan s $\log_2 k$ čvorova, a isto toliko je i koraka potrebno da se informacija od jednog čvora proslijedi svim ostalima [28].

Karakteristike koje se obično uzimaju u obzir prilikom evaluacije topologija su dijametar (eng. *diameter*) i povezanost (eng. *connectivity*). Dijametar predstavlja maksimalnu udaljenost između bilo koja dva čvora u topologiji [28], tj. najveći broj bridova (veza) na najkraćem putu između bilo bilo koja dva čvora [23, str. 933]. Formalno se dijametar definira kao

$$diam(T) = max_{u,v \in V} d(u,v)$$

gdje $d(u,v)$ predstavlja najkraći put između čvorova u i v, a $T = (V,E)$ predstavlja topologiju [23, str. 933].. Primjerice, dijametar kod potpuno povezanog grafa je 1, dok je kod prstena s k čvorova jednak k. Povezanost u topologiji predstavlja broj bridova (veza) koji pripadaju čvoru.

Tablica 2. Karakteristike topologija za k čvorova

Topologija	Dijametar	Povezanost	Broj veza
Potpuno-povezana	1	$k-1$	$\frac{n \cdot (n-1)}{2}$
Hiperkocka	$\log_2 k$	$\log_2 k$	$\frac{k \cdot \log_2 k}{2}$
Prsten	k	1	k

Na temelju topologija formiraju se različite migracijske strategije. U pravilu se prilikom razmjene informacija koriste (najbolja) pronađena rješenja (mravi) jer je već u ranijim radovima koji su se bavili paralelnim ACO algoritma uočeno da razmjena potpune matrice feromona ne ostvaruje dobre rezultate [18]. Middendorf i Schmeck [18] usporedili su performanse četiri strategije koje se temelje na topologiji prstena ili potpuno povezanoj topologiji:

a) Razmjena globalno najboljeg rješenja: Prilikom svake razmjene informacija pronalazi se globalno najbolje rješenje koje se šalje svim kolonijama i postaje njihovo lokalno najbolje rješenje. Strategija koristi potpuno povezanu topologiju.
b) Kružna razmjena lokalno najboljih rješenja: Ova strategija podrazumijeva topologiju usmjerenog prstena. Prilikom razmjene svaki čvor svoje dosad najbolje rješenje šalje svojem sljedbeniku u prstenu. Sljedbenik ažurira svoje lokalno najbolje rješenje u slučaju da je primljeno rješenje bolje.
c) Kružna razmjena migranata: Koristi se topologija kao i u slučaju b), a svaka kolonija uspoređuje m_b svojih mrava s m_b mrava svojeg sljedbenika. Od ukupno $2m_b$ mrava odabire se m_b mrava koji se koriste za ažuriranje matrice feromona.
d) Kružna razmjeno lokalno najbolje rješenja zajedno s migrantima: Kombinacija strategija b) i c).

Strategije su testirane koristeći sinkroni model razmjene svakih $I = 10$ i $I = 50$ iteracija, a veličine kolonija su $k \in \{1, 5, 10, 20\}$. Napravljen su tri usporedbe: razlika feromonskih matrica između kolonija, prosječan broj gradova koji su na raspolaganju mravu za odabir te usporedba kvalitete rješenja za TSP.

Prva usporedba pokazuje kako je za sve strategije razlika između kolonija rasla do prve razmjene rješenja. Nakon toga su uočene dvije krajnosti. Kod strategije a) je za obje frekvencije razmjene uočeno naglo smanjenje razlike nakon prve razmjene, a najveća razlika je postignuta neposredno prije prve razmjene. Veća frekvencija (kraći interval) razmjene značajno je utjecala na brzinu smanjenja razlike između kolonija. Druga krajnost je strategija c) kod koje je uočeno da razmjena migranta nema veliki učinak na razliku između kolonija. Za

oba intervala razlika je sporo rasla čak i nakon prve razmjene sve do trenutka kad je započela stagnacija što pokazuje da migracijski interval nije imao velik učinak. Neznatan pad izmjeren je samo za $I = 10$. Krivulja strategije b) nalazi se između krivulja za a) i c). Razlika između kolonija je rasla do trenutka prve razmjene, a nakon toga je uočena velika razlika za $I = 10$ i $I = 50$. Manja frekvencija utjecala je na značajno sporije smanjenje razlike između kolonija. Strategija d) pokazala je slično ponašanje za obje frekvencije što pokazuje da je korištenje razmjene lokalno najboljih rješenja dominantni faktor u ovoj strategiji.

U drugoj usporedbi prikazana je brzina konvergencije algoritma za spomenute strategije. Ovdje je izmjeren prosječan broj odabira (alternativa) koje mrav ima na raspolaganju prilikom prelaska s jednog grada na drugi. U obzir su uzeti samo oni gradovi kod kojih je vjerojatno odabira p značajna, tj. $p > \lambda$ (spominje se $\lambda = 0.01$). Za sve strategije je za obje frekvencija uočen značajan pad alternativa koji se pao na manje od 2 prije 80. iteracije (od 500 iteracija ukupno). Također je uočeno kako se broj alternativa povećava u iteracija u kojima je izvršena razmjena (krivulje imaju lokalne ekstreme i šiljasti oblik u tim iteracijama), što pokazuje da su mravi istražili rješenja na temelju dobivenih informacija, tj. da su dobivena rješenja u slučaju strategija a), b) i d) potaknule mrave na pretraživanje novih rješenja koja se nalazi blizu dobiveno. Nakon 150 iteracija broj alternativa je najmanji za strategiju a), što znači da ova strategija prosječno ne pridonosi raznolikosti i pretrazi šireg prostora rješenja. Općenito, najveći broj alternativa za $I = 10$ ostvaren je strategijom c), dok je za $I = 50$ najveći broju alternativa ostvaren strategijama b) i d).

Također je napravljena i usporedba kvalitete rješenja za 500 iteracija i $I = 50$, za 1,5,10 i 20 kolonija. Za kolonije veličine 100 korištena su tri, za 10-20 dva, a za koloniju veličine 5 mrava samo jedan mrav za deponiranje feromona. Za navedene postavke pokazalo se je kako je bolje koristiti jednu veću kolonija umjesto nekoliko manjih ukoliko nema komunikacije između njih. Ukoliko se uvede komunikacija, strategija b) ostvaruje najbolje rezultate za 5 i 10 kolonija, dok strategija c) ostvaruje najgore. Dok se je u prethodnim usporedbama strategija a) pokazala najlošijom, u ovom testu je ostvarila bolje rezultate od strategije c) te ujedno i bolje rezultate od onih koji su ostvareni pomoću kolonija bez komunikacije.

Twomey, Stützle, Dorigo, Manfrin i Birattari [27] testirali su strategije a) i b) te dodatno strategije:

e) Razmjena lokalno najboljih rješenja u hiperkocki: Svaka kolonija koja se nalazi na jednom vrhu hiperkocke razmjenjuje najbolja rješenja sa svojima susjedima.
f) Zamijeni najgoreg (eng. *replace-worst*): Jedna kolonija ima ulogu gospodara te u svakoj razmjeni od ostalih kolonija prikuplja informaciju o najboljem rješenju. Gospodar svim kolonijama šalje identifikatore kolonije s najboljim i najgorim rješenje. Kolonija s najboljim šalje svoje rješenje koloniji s najgorim rješenjem.

Prilikom testiranja je za razmjenu rješenja korišten sinkroni pristup s dvije varijante:

- fiksna frekvencija razmjene
- razmjena s povećavajućom frekvencijom – učestalost razmjene se povećava s iteracijama

U oba pristupa prvih se T iteracija izvodi bez razmjene rješenja te se nakon toga rješenja razmjenjuju svakih $c > 0$ iteracija. Drugi pristup ima dodatno ograničenje $c < T$ te minimalno c iteracija treba proći između dvije razmjene. Potonje ograničenje je uvedeno kako se razmjena nakon početnih i iteracija ne bi odvijala prečesto. Kod razmjene s povećavajućom frekvencijom i-ta razmjena se događa u iteraciji

$$\sum_{k=0}^{i-1} \lfloor b^k \cdot T \rfloor_c$$

gdje je $0 < b \leq 1$, a $\lfloor x \rfloor_c = \lfloor b^k \cdot T \rfloor_c$ se računa kao

$$\lfloor x \rfloor_c = \begin{cases} c, & za\ x < c \\ \lfloor x \rfloor, & za\ x \geq c \end{cases}$$

i predstavlja broj iteracija između dvije razmjene. U radu je testiranje obavljeno za $T = 100$, $c = 25$ za fiksnu razmjenu i $T = 1000$, $b = 0.9$, $c = 25$ za razmjenu s povećavajućom frekvencijom na MMAS algoritmu s 1000, 3162 i 10 000 iteracija.

Tablica 3. Rezultati testiranja migracijskih strategija u odnosu na PIR (Izvor [27])

Frekvencija	**Strategija**	**Mali broj iteracija**	**Srednji broj iteracija**	**Veliki broj iteracija**
Fiksna	FC	+2 -0	+2 -1	+1 -4
	HC	+1 -0	+2 -1	+1 -4
	RW	+5 -0	+3 -0	+1 -0
	R	+4 -0	+3 -0	+1 -1
Povećavajuća	FC	+0 -0	+2 -1	+1 -3
	HC	+0 -0	+2 -1	+1 -3
	RW	+0 -0	+2 -0	+1 -0
	R	+0 -0	+2 -0	+2 -1

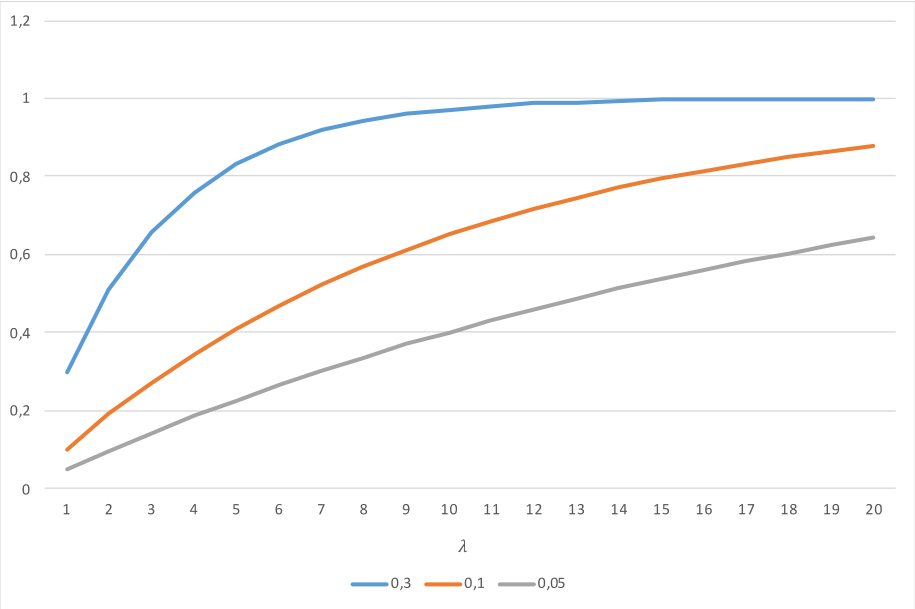
Tablica predstavlja rezultate testiranja različitih strategija u odnosu na paralelno nezavisne algoritme. Strategije FC, HC, RW, R predstavljaju potpuno-povezanu, hiperkocku, zamijeni najgoreg i strategiju kružne razmjene lokalno najboljih rješenja. U stupcima 3-5 +N označava

da ju kod N TSP problema strategija ostvarila značajno bolje rezultate od PIR, a -N da je ostvarila značajno lošije rezultate od PIR.

Za mali broj frekvencija (1000), najbolje rezultate ostvarile strategije su strategije s fiksnom frekvencijom razmjene. U odnosno na PIR gotovo sve strategije su u 6 od 7 TSP problema pridonijele značajno kvalitetnijem rješenju. Razmjena s povećavajućom frekvencijom se aktivira tek od 1000. iteracije pa očekivano nije pridonijela kvaliteti rješenja. Kod testova sa srednjim brojem iteracija (3162) vidi se učinak i razmjene s povećavajućom frekvencijom. U toj su vrsti razmjene značajno bolje rezultate ostvarile strategije koje se temelje na topologijama s manjim dijametrom (potpuno povezana i hiperkocka), pretežito zato jer su za 3162 iteracije napravljene samo tri razmjene (u 1000., 1900. i 2710. iteraciji). I u ovom slučaju su strategije s fiksnom frekvencijom razmjene za veći broj algoritama ostvarile statistički značajno bolje performanse od strategija s povećavajućom frekvencijom. Kod testova s velikim brojem iteracija (10000) za obje su frekvencije ostvareni slični rezultati. Generalno, najbolji rezultate ostvarile su strategije kružne razmjene i zamijene najgoreg rješenja koristeći fiksnu frekvenciju razmjene. Potrebno je napomenuti kako su ovo rezultati za MMAS algoritam bez korištenja algoritma lokalne pretrage. Korištenje algoritama lokalne pretrage značajno utječe na rezultate ovih testova i taj dio je djelomično opisan u poglavlju koje sadrži opis TBAS implementacije.

3.4. Performanse

ACO algoritmi distribuiraju se zato jer se želi postići bolja kvaliteta rješenja i kraće vrijeme izvođenja algoritma. Jedan od najjednostavnijih modela, PIR, podrazumijeva postojanje nekoliko kolonija koje međusobno ne komuniciraju. Zanimljivo je da već i takav model daje dobre rezultate. Neka je p vjerojatnost da će slijedni algoritam ili jedna instanca PIR modela pronaći zadovoljavajuće rješenje. Tada je vjerojatnost da ni jedna od λ instanci PIR-a neće pronaći zadovoljavajuće rješenje jednaka $(1 - p)^{\lambda}$ iz čega proizlazi da je vjerojatnost da će bar jedna pronaći zadovoljavajuće rješenje jednaka $1 - (1 - p)^{\lambda}$ [23, str. 931], [29].



Slika 18. Vjerojatnost $1 - (1 - p)^{\lambda}$ u ovisnosti o p i broju instanci algoritama λ

Slika prikazuje povećanje vjerojatnosti pronalaska zadovoljavajućeg rješenja skaliranjem na više procesa za različite vjerojatnosti p. Primjerice, za p = 0.05 se skaliranjem na 10 procesa vjerojatnost pronalaska zadovoljavajućeg rješenja povećava s 0.05 za 1 proces na 0.4 za njih 10. Može se primijetiti kako za svaki p na početku postoji gotovo linearan rast, a trenutak u kojem dolazi do zasićenja skaliranjem uvelike o p. Ovaj izraz nije samo koristan za PIR nego može poslužiti i kao donja granica vjerojatnosti pronalaska dobrog rješenja kod modela koji razmjenjuju rješenju poput višekolonijalnog.

Prilikom ocjene performansi distribuiranog algoritma često se koristi mjera ubrzanje (eng. *speedup*). Ubrzanje se može podijeliti na [30]:

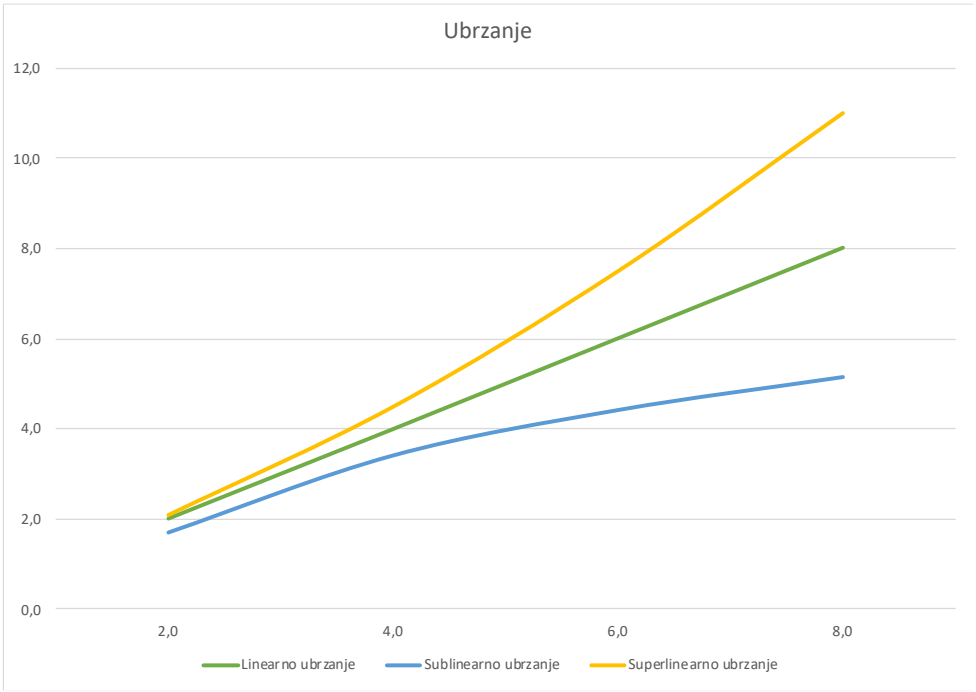
- Jako ubrzanje (eng. *strong speedup*) – vrijeme distribuiranog algoritma uspoređeno je s najboljom poznatom verzijom sekvencijalnog algoritma (npr. distribuirani AS algoritam uspoređuje se sa slijednim MMAS algoritmom s optimalnim vrijednostima parametra)
- Slabo ubrzanje (eng. *weak speedup*) – vrijeme distribuiranog algoritma uspoređuj se s vremenom sekvencijalnog algoritma iste verzije (npr. distribuirani MMAS uspoređuje se sa svojom slijednom verzijom)
    - U odnosu na kanonsku verziju – vrijeme izvršavanja algoritama distribuiranog na m računala uspoređeno je s vremenom kanonskog, sekvencijalnog algoritma
    - Ortodoksno - vrijeme izvođenja algoritam distribuiranog na m računala uspoređuje se s distribuiranom verzijom algoritma pokrenutog samo na jednom računalu (relativno ubrzanje)

Ubrzanje se računa pomoću sljedećeg izraza [30]

$$s_m = \frac{T_1}{T_m}$$

gdje T_1 predstavlja vrijeme izvođenja algoritma na jednom računalu, a T_m vrijeme izvođenja algoritma distribuiranog na m računala. Ovisni o vrijednosti s_m, ubrzanje se kategorizira kao:

- Sublinearno ubrzanje – u slučaju $s_m < m$ gdje je ukupno vrijeme izvođenja algoritma na jednom računalu manje nego zbroj ukupnih vremena izvođenja na svih m računala
- Linearno ubrzanje – u slučaju $s_m = m$ gdje je vrijeme izvođenja algoritma distribuiranog na m računala točno m putanja manje nego vrijeme izvođenje algoritma na jednom računalu
- Superlinearno ubrzanje – u slučaju $s_m > m$ gdje je ukupno vrijeme izvođenja algoritma na jednom računalu veće nego zbroj ukupnih vremena izvođenja na svih m računala



Slika 19. Prikaz sublinearnog, linearnog i superlinearnog ubrzanja

Općenito, neka je T ukupno vrijeme algoritma, T_s vrijeme utrošeno na dio algoritma koji se ne može paralelizirati i $T_p = T_s - T$ vrijeme utrošeno na dio algoritma koji se može paralelizirati. Neka dodatno $r_s = \frac{T_s}{T}$ predstavlja postotak vremena utrošenog na slijedni dio algoritma, a $r_p = r_s - 1$ postotak vremena utrošenog na dio algoritma koji se može paralelizirati. Ako se paralelni dio raspodijeli na k radnika, tada će izvođenje paralelnog dijela trajati $T_p' = \frac{T_p}{k}$, a ukupno vrijeme izvođenja algoritma će iznosti $T' = T_s + T_p'$. Ubrzanje se u ovom slučaju definira kao [5]

$$S = \frac{T}{T'} = \frac{T}{T_s + T_p'} = \frac{T}{T \cdot r_s + T \cdot \frac{r_p}{k}} = \frac{T}{T \cdot (r_s + \frac{r_p}{k})} = \frac{1}{r_s + \frac{r_p}{k}}.$$

Dakle, u idealnom slučaju kad nema troškova paralelizacije maksimalno ubrzanje iznosi

$$\lim_{k \to \infty} S = \frac{1}{r_s}$$

što znači da je u tom slučaju ubrzanje limitirano sekvencijalnim dijelom algoritma. To za posljedicu ima da koliki god trud se uloži u kvalitetnu implementaciju paralelnog dijela, ako je sekvencijalno vrijeme dominantni faktor neće se postići značajno ubrzanje. U stvarnosti pa tako i kod paralelnih i distribuiranih ACO algoritama paralelizacija sa sobom donosi dodatne troškove poput komunikacije i sinkronizacije koji produljuju vrijeme izvršavanja algoritma. Neka trošak paralelizacije po svakom radniku predstavlja dio ukupnog vremena algoritma $T \cdot \tau$. Tada je ukupno vrijeme izvođenja algoritma jednako $T'' = T_s + T_p''$, gdje je $T_p'' = \frac{T_p}{k} + k \cdot (T \cdot \tau)$, a ubrzanje je [5]

$$S = \frac{T}{T''} = \frac{T}{T_s + T_p''} = \frac{T}{T \cdot r_s + T \cdot \frac{r_p}{k} + k \cdot (T \cdot \tau)} = \frac{T}{T \cdot (r_s + \frac{r_p}{k} + k \cdot \tau)} = \frac{1}{r_s + \frac{r_p}{k} + k \cdot \tau}.$$

Da bi se paralelizacija isplatila, iz izraza iznad proizlazi da član $k \cdot \tau$ mora biti manji od člana $\frac{r_p}{k}$, tj. trošak paralelizacije po radniku τ mora biti

$$k \cdot \tau < \frac{r_p}{k}$$

$$\tau < \frac{r_p}{k^2}.$$

U suprotnom će algoritam više vremena utrošiti na troškove paralelizacije nego na posao koji se paralelizira, iako je i dalje moguće postići ubrzanje u odnosu na slijednu verziju. U slučaju da je $k \cdot \tau > (r_p - \frac{r_p}{k})$, tj.

$$\tau > \frac{r_p(k-1)}{k^2}.$$

tada vrijeme utrošeno na troškove paralelizacije nadilazi uštedu vremena ostvarenu paralelizacijom $(r_p - \frac{r_p}{k})$ i čitav algoritam traje više od slijedne verzije. Zbog ovog se u distribuiranim verzijama algoritma izbjegavaju sitno-zrnati modeli jer komunikacijski trošak može vrlo brzo postati dovoljno velik da poništi ubrzanje dobiveno paralelizacijom poslova.

Efikasnost je još jedna mjera koja se često koristi prilikom mjerenja performansi, a predstavlja normalizirano ubrzanje [30]

$$e_m = \frac{s_m}{m}.$$

Za $e_m = 1$ riječ je o linearnom ubrzanju, a za vrijednosti manje i veće od 1 o sublinearnom, odnosno superlinearnom ubrzanju. Kod distribuiranih algoritama najčešće je riječ o sublinearnom ubrzanju gdje se teži tome da je efikasnost što bliža 1. Povezano s efikasnošću postoji i mjera inkrementalne efikasnosti koja mjeri ubrzanje dobiveno prelaskom s m' na m procesa, gdje je $m' < m$ [23, str. 937]

$$ie_m = \frac{m\prime \cdot T_{m\prime}}{m \cdot T_m}.$$

Prilikom mjerenja performansi dva se pristupa često koriste. U prvom je vrijeme izvođenja algoritma fiksno te se na kraju uspoređuje kvaliteta rješenja. Obratna situacija je kod drugog pristupa gdje se unaprijed postavlja tražena kvaliteta rješenja (npr. traži se optimalno rješenje ili ono koje odstupan određen postotak od optimalnog), a na kraju se uspoređuje vrijeme algoritama. Izrazi koji se nalaze iznad i koji su vezani uz ubrzanje odnose se na potonji pristup.

3.5. Implementacija i rezultati

Za potrebe izrade finalnog programskog rješenja korišten je programski jezik Java, JDK 17. Rezultati testiranja, osnovni statistički izračuni te grafovi napravljeni su pomoću tabličnih kalkulatora Microsoft Excel i Google Sheets.

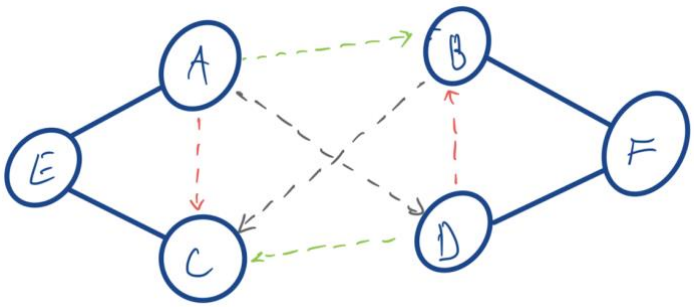
3.5.1. TBAS – slijedna implementacija

Kako bi se mogle izmjeriti performanse i usporedi rješenja dobivena raspodijeljenom implementacijom, prvo je napravljena slijedna verzija koja se izvršava samo na jednom računalu. Odabrana varijanta algoritma je TBAS koji je pokazao nešto bolje performanse (bolje kvaliteta rješenja i kraće vrijeme izvršavanja) u odnosu na MMAS koji je najčešće korišten. Prednosti TBAS-a su što u početnim iteracijama izbjegava kontrakciju (isparavanje) pa je izvođenje brže, a količina feromona koja se deponira raste s iteracijama čime se na početku naglasak stavlja na pretragu šireg prostora rješenja (isto kao MMAS), a na kraju se potiče konvergencija algoritma.

Parametri algoritma koji su korišteni su $\alpha = 1$, $\beta = 4$, $\rho = 0.02$, $\omega = 1 - \rho$. Vrijednosti τ_{LB} i τ_{UB} za svaki se TSP problem računaju na isti način koji je opisan u poglavlju 3.4.4. Prilikom deponiranja feromona korištena je kombinacija ažuriranja najbolji rješenjem u iteraciji i globalno najboljim rješenjem na način da se u svakih 20 iteracija koristi globalno, a u ostalima najbolje rješenje u iteraciji. Kratko testiranje na nekoliko prsoblema (kroA100 i kroA150) pokazalo je kako TBAS pronalazi bolja rješenja od MMAS algoritma, ali da bi bilo poželjno uvesti dodatne optimizacije kako bi se dodatno povećala prosječna kvaliteta rješenja. Najčešći algoritmi koji se kombiniraju s ACO heuristikama su 2-opt i 3-opt algoritmi lokalne pretrage (eng. *local search*) [8], [9], [27], [31].

Algoritmi lokalne predstavljaju heuristike koje kao ulaz primaju konstruirano rješenje te zatim iterativno nad tim rješenjem rade male modifikacije kako bi se pronašlo bolje. Konkretno, 2-opt i 3-opt rade na način da iteriraju kroz sve bridove u težinskom grafu, u svakoj iteraciji iz

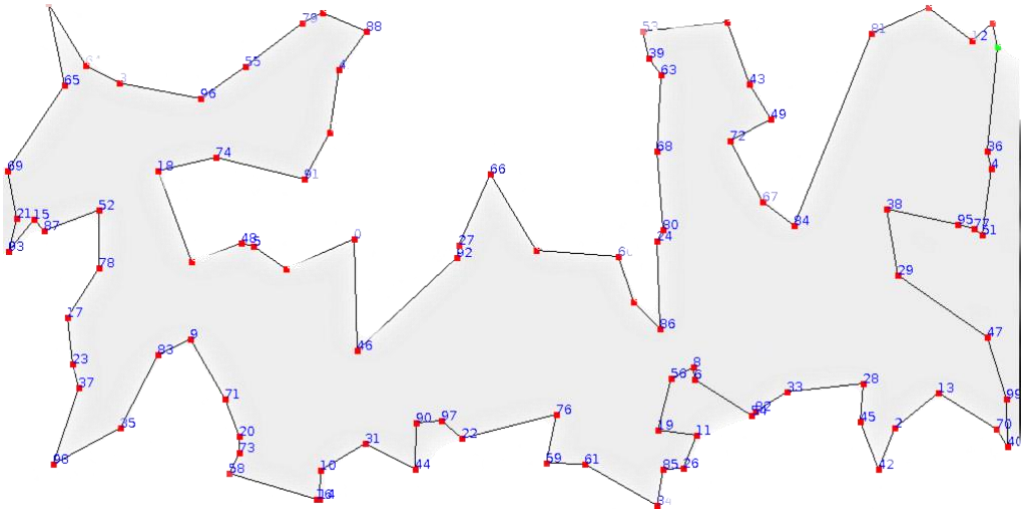
postojećeg rješenja uklanjaju dva ili tri brida (ovisno o tome radi li se o 2-opt ili 3-opt) koji nemaju zajedničke vrhove te time stvaraju nepovezani graf. Nakon toga uzimaju isti broj bridova i ponovno stvaraju Hamiltonov graf, ali bridovi ovog puta povezuju drugačije vrhove. Ako je duljina novog ciklusa kraća od prethodne ona se prihvaća, u suprotnom se odbacuje i pokušava se sa sljedećom zamjenom sve dok se ne dostigne kriterij zaustavljanja. Kod 2-opt algoritma u svakoj zamjeni bridova postoji samo jedan način zamjene koji ponovno stvara Hamiltonov graf, dok je kod 3-opt algoritma takvih načina 7. Zbog toga je 3-opt učinkovitiji i uobičajeno dalje bolja rješenja, ali ima veću vremensku složenost od 2-opt - $O(n^3)$ u odnosu na $O(n^2)$. Zbog manje složenosti odabran je 2-opt. Dodatno, 2-opt se zaustavlja čim se pronađe prvo bolje rješenje po uzoru na Ivković, Kudelić, Golub [31].



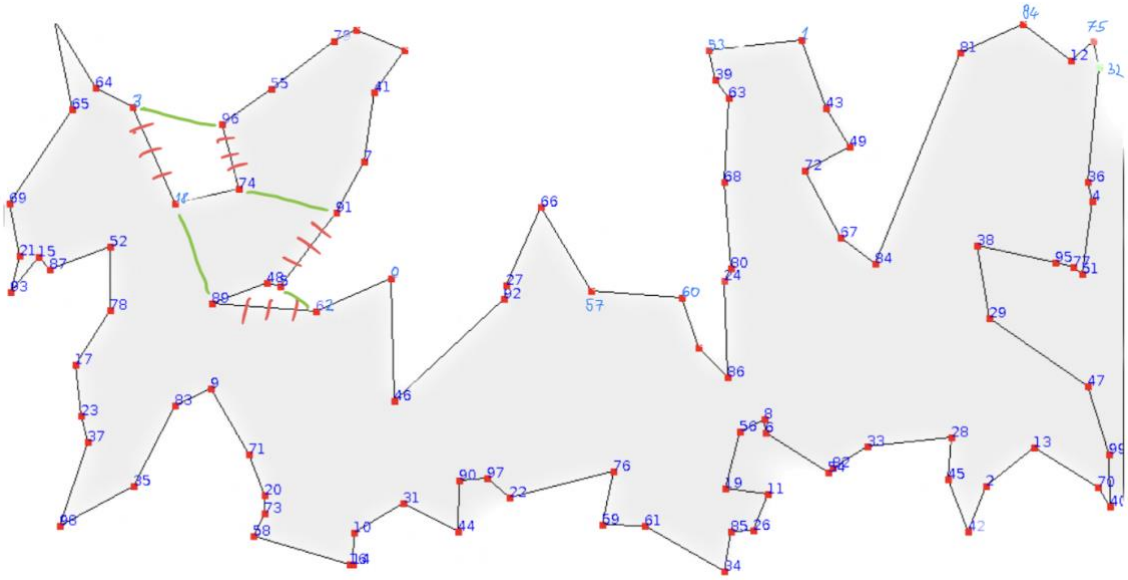
Slika 20. 2-opt zamjena bridova

Slika prikazuje način na koji 2-opt algoritam provodi zamjenu bridova. Sivom bojom su nacrtani bridovi na početku. Postoje ukupno 3 načina na koji se mogu napraviti zamjene. Jedan način postavlja bridove na isto mjesto, drugi način stvara nepovezani graf (crveni bridovi), a treći način stvara novi Hamiltonov graf (zeleni bridovi).

TBAS u kombinaciji s 2-opt algoritmom daje kvalitetnija rješenja, ali postoje slučajevi u kojima 2-opt nije u stanju poboljšati rješenje. Jedan takav slučaj je uočen prilikom testiranja kroA100 problema. TBAS je često zapeo u lokalnom optimumu s rješenjem duljine 21382 i nije bio u stanju pronaći bolje rješenje (duljina optimalnog rješenja je 21282). Nakon vizualizacije, uočeno je da je rješenje duljine 21382 takvo da nijedna izmjena od 2 brida mogla pronaći bolje



Slika 21. kroA100 optimalno rješenje duljine 21282



Slika 22. kroA100 rješenje duljine 21382

rješenje. Na slici se primjerice vidi kako bi trebalo zamijeniti 4 brida kako bi se došlo do optimalnog rješenja. Zato je po uzoru na Stützle, Hoos [8] dodatno uvedeno izglađivanje feromonskih tragova. U radu su spomenute dvije mogućnosti:

- proporcionalno ažuriranje - količina feromona na svakom bridu se podiže proporcionalno razlici između trenutne količine i gornje granice τ_{max}

$$\tau_{ij}(\mathrm{t}) = \tau_{ij}(t) + \delta \cdot \left(\tau_{max}(\mathrm{t}) - \tau_{ij}(\mathrm{t})\right), \qquad 0 < \delta < 1$$

- podizanje donje granice – povećava se donja granica τ_{min}

$$\tau_{min}(\mathrm{t}) = \tau_{min} + \delta \cdot \left(\tau_{max}(\mathrm{t}) - \tau_{min}(\mathrm{t})\right), \qquad 0 < \delta < 1$$

Druga opcija je već kod lokalnog testiranja na MMAS algoritmu pokazala lošije rezultate pa je odabrano proporcionalno ažuriranje. Izraz je za potrebe TBAS-a malo modificiran pa se

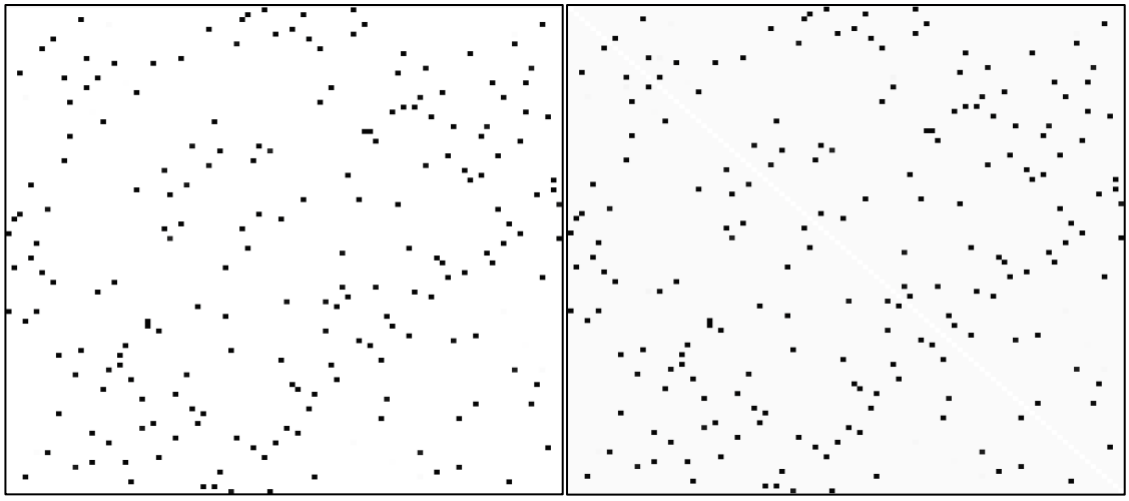
prilikom ažuriranja koristi τ_{UB} umjesto τ_{max}. I s ovim pristupom je bio problem da bi vrijednosti feromona na lošim putevima nakon nekog vremena postale prevelike što bi dovelo do nasumičnog lutanja mrava po prostoru rješenja – umjesto konvergencije, algoritam je počeo generirati sve lošija rješenja. Bilo bi dakle dobro odgoditi izglađivanje do nekog trenutka. Na taj način bi mravi u početnim iteracijama imali dovoljno vremena formirati kolektivno znanje kolonije, a kasnije bi aktivacija izglađivanja služila za fino podešavanje pronađenog rješenja. Ovakav pristup objasnili su Pérez-Carabaza, Gálvez, Iglesias [10]. Kao kriterij za aktivaciju navedene su tri mjere, a autori su predložili pristup koji mjeri standardnu devijaciju duljine rješenja zbog manje računske složenosti u odnosu na ostala dva pristupa. Kod ovog pristupa se izglađivanje tragove aktivira tada kad je standardna devijacija određenog postotka pronađenih rješenja u iteraciji jednaka nuli. Autori rada su taj postotak odredili eksperimentalno na način da su za nekoliko odabranih TSPLIB problema dostupnih na http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/ pokretali algoritam 30 puta i mjerili koliko puta je algoritam uspio pronaći bolje rješenje nakon što bi standardna devijacija određenog postotak rješenja bila jednaka nuli. Sličan pristup iskorišten je i u ovom radu i obavljeno je slično testiranje. Razlika je u tome da je umjesto standardne devijacije korišten nešto jednostavniji pristup – gleda se postotak svih rješenja u iteraciji čije su duljine, zaokružene na dvije decimale, jednake. U nastavku se nalazi tablica s rezultatima.

Tablica 4. Broja pokretanja algoritma (od 30 pokretanja) u kojima je pronađeno bolje rješenje

TSPLib problem	**Postotak rješenja s istom duljinom**									
	10%	**20%**	**30%**	**40%**	**50%**	**60%**	**70%**	**80%**	**90%**	**100%**
kroA100	30	28	24	19	18	17	15	12	2	0
ch130	1	0	0	0	0	0	0	0	0	0
tsp225	30	30	30	29	28	27	24	21	16	0
Pr76	28	19	14	13	10	9	3	0	0	0

Primjerice, u slučaju kroA100 TSP problema, kad je 10% konstruiranih puteva jednake duljine, algoritam je u svih 30 pokretanja uspio pronaći bolje rješenje. Kad je 50% konstruiranih puteva bilo jednake duljine, tada je algoritam u $18/30$ pokretanja uspio pronaći bolje rješenje. Kod svih problema vidi se slično ponašanje i očekivani pad kako broj rješenja s istom duljinom raste i algoritam se približava fazi stagnacije. Ponašanje algoritma se poklapa s ponašanjem u spomenutom radu. Značajna razlika uočena je jedino kod ch130 problema kod kojeg je i nakon nekoliko testiranja vidljivo da algoritam ne stagnira. Vjerojatno bi se do konvergencije došlo kad bi se broj iteracija značajno povećao (testirano je na 3000 iteracija), ali s obzirom da je svaki problem testiran 30 puta, to bi značajno produljilo vrijeme testiranja. Na temelju dobivenih rezultata, kao postotak za aktivaciju uzeto je 80%. Dakle, kad 80% pronađenih rješenja u iteraciji ima istu duljinu, tada je algoritam jako blizu ili u fazi stagnacije i aktivira se izglađivanje

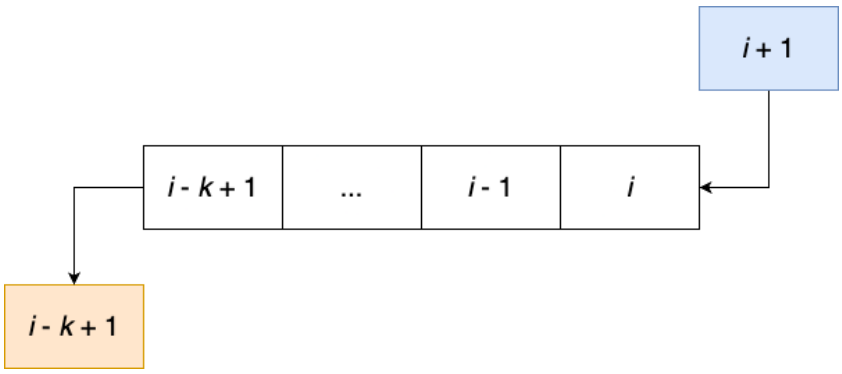
tragova. Izglađivanje najviše utječe na bridove s manjom količinom feromona zbog čega mravi prilikom sljedeće iteracije s većom vjerojatnošću razmatraju i takve bridove što često dovodi do pronalaska boljih rješenja. Autori su faktor δ postavili na vrijednost 0.1, dok se je u ovom radu eksperimentalno pokazalo kako TBAS najbolje performanse ostvaruje za $0.01 \leq \delta \leq 0.07$. Da bi se lakše uočio trenutak i efekt koji se postiže izglađivanjem tragova, prilikom implementacije dodana je vizualizacije feromonske matrice, a u nastavku se može vidjeti slika koja prikazuju stanje matrice neposredno prije i neposredno poslije aktivacije izglađivanja. Bijela bolja se koristi za količine jednake τ_{LB}, crna za τ_{UB}, a sve ostale vrijednosti unutar tih granica prikazane su nijansom sive koja je tamnija što je količina bliža τ_{UB}. Na desnoj matrici se može uočiti kako se na dijagonali i dalje nalazi količina feromona koja je jednaka 0 (očekivano jer su to količine na bridovima koji povezuju gradove sa samima sobom), ali se u ostatku matrice može primijetiti blago siva nijansa koja upućuje na to da su količine feromona na svim bridovima između različitih gradova podignute i veće od τ_{LB}.



Slika 23. *Prikaz feromonske matrice neposredno prije (lijevo) i neposredno poslije (desno) aktivacije izglađivanja feromona*

U TBAS je također uvedena i dodatna strategija za ažuriranje feromona koja umjesto jednog rješenja (najbolje u iteraciji ili najboljeg do sad) koristi k-najboljih, a opisana je i testirana u [31]. Strategija podrazumijeva strukturu podatka koja pamti k-najboljih rješenja do sad. U svakoj iteraciji u strukturu se dodaje najbolje rješenje u iteraciji. Ukoliko je struktura puna, tj. već sadrži svih k rješenje, tada se iz nje prvo uklanja ono najstarije i potom se dodaje novo. Ukoliko je novo rješenje bolje od svih rješenja koje se trenutno nalaze u strukturi, tada se sva prethodna rješenja uklanjaju i u strukturu se dodaje novo najbolje. Prikladna struktura za opisane operacije je red (eng. *queue*). U toj strukturu se operacije dodavanja i uklanjanja

rješenja odvijaju u $O(1)$, dok se pronalazak najboljeg rješenja odvija u linearnoj složenosti $O(k)$.



Slika 24. Strategija k-najbolji – prikaz dodavanja i uklanjanja rješenja iz reda u $i + 1$ iteraciji (Prema *[31]*)

Strategija je u TBAS uključena na način da algoritam provodi ažuriranje globalno najboljim rješenjem jednom svakih 20 iteracija, a u preostalim iteracijama se izvodi ažuriranje strategijom k-najbolji ili najbolji u iteraciji (ovisno o tome je li k-najbolji strategija uključena ili ne). U ovom radu je odabran $k = 4$ zbog dobrih rezultata u testovima sa i bez korištenja 2-opt heuristike koji su provedeni u [31].

Nabrojane optimizacije i strategije uvedene su u algoritam kako bi se stvorila što bolja varijanta slijednog TBAS algoritam te kako bi se u konačnici usporedbom rezultata između slijedne i raspodijeljene implementacije mogli donijeti relevantni zaključci o performansama između dvije implementacije.

3.5.2.TBAS – raspodijeljena implementacija

Kod raspodijeljene implementacije pažnja je posvećena tome da se smanji negativni utjecaj komunikacijske na performanse algoritma. Slijedna varijante algoritma testirana na kroA100 problemu je na testnom računalu u jednoj sekundi procesuirala 80 iteracija pa se je već eksperimentalno pokazalo da bi slanje velike količine podataka ili prečesto slanje manje količine moglo značajno usporiti algoritam. Zato u obzir nisu uzeti sitno zrnati modeli s velikom migracijskom frekvencijom ili modeli kod koji se šalje cijela matrica feromona, nego je u konačnici odabran višekolonijalni model. Ovaj model je odabran zbog dobrih rezultata postignutih u [18] te istraživanja u kojem je navedeno kako je najviše radova objavljeno upravo za taj model [15]. Također, po uzoru na [18], prilikom razmjene svaka kolonija šalje svoje trenutno najbolje rješenje. Ako je primljeno rješenje bolje od trenutno najboljeg rješenje u odredišnoj koloniji, kolonija ažurira svoje najbolje rješenje do sad. Da bi primljeno rješenje imalo utjecaj na ponašanje kolonija, u algoritam je ukomponirano ažuriranje feromona

najboljim rješenje do sad i najboljim rješenjem u iteraciji na isti način koji je opisan kod slijedne verzije.

Kako bi se pojednostavilo testiranje, višekolonijalni model je uklopljen u model gospodar-sluga. Poslovi gospodara su:

- TCP poslužitelj
- inicijalizacija sluga - učitava TSPLIB problem i šalje ga slugama koji su poslali zahtjev
- inicijalna koordinacija – brine da sve kolonije započnu s izvršavanjem problema u isto vrijeme
- odabir i prikaz najboljeg rješenja – prima najbolje finalno rješenje od svake kolonije, od primljenih odabire najbolje i vizualizira ga na ekranu

S obzirom da se komunikacija između slugu i gospodara odvija prije i poslije, ali ne i za vrijeme izvršavanja TBAS algoritma, kao protokol za komunikaciju je odabran TCP jer latencija u ovom slučaju nije bitna, ali je bitno da sve odaslane poruke stignu na odredište kako bi se ispravno pokrenuli svi sluge i kako bi sva finalna rješenja stigla do gospodara. Poslovi sluga su:

- iniciranje veze s gospodarom – prijavljuju se kod gospodara i dobivaju podatke potrebne za inicijalizaciju (primjerice listu gradova)
- imaju ulogu kolonije u višekolonijalnom modelu - izvršavaju TBAS algoritam slično kao kod slijedne verzije
- razmjenjuju rješenja s ostalim slugama – način razmjene ovisi o topologiji
- šalju finalno najbolje rješenje gospodaru

Da bi se ostvarila opisana komunikacija između gospodara i slugu, osmišljen je jednostavan algoritam koji podržava tri operacije:

- INIT – inicijalni zahtjev koji šalje svaki sluga da bi se prijavo kod gospodara i od njega dobio potrebne podatke za rad (listu gradova, topologiju, duljinu optimalnog rješenja)
- START – zahtjev kojeg sluga šalje gospodaru nakon INIT zahtjeva, a koji signalizira da je sluga sprema za pokretanje
- SOLUTION – zahtjev u kojem sluga gospodaru šalje najbolje rješenje na kraju izvođenja programa

Da bi se ostvarila komunikacija između kolonija korištene su dvije topologije:

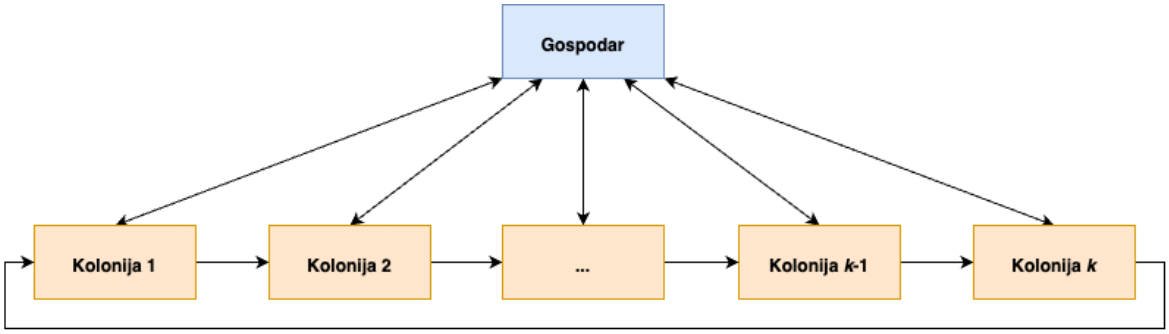
- Potpuno povezana
- Topologija prstena

S obzirom da je prilikom razmjene rješenja bitno da je komunikacija što brža, na transportnom sloju je odabran protokol UDP. Nedostatak ovog protokola je što ne garantira da će poslani paket stići na odredište. Međutim, u ovoj implementaciji to ne predstavlja problem jer kolonije svaki put razmjenjuju najbolje rješenje do sad. Ako se i dogodi da se neki paket izgubi i da

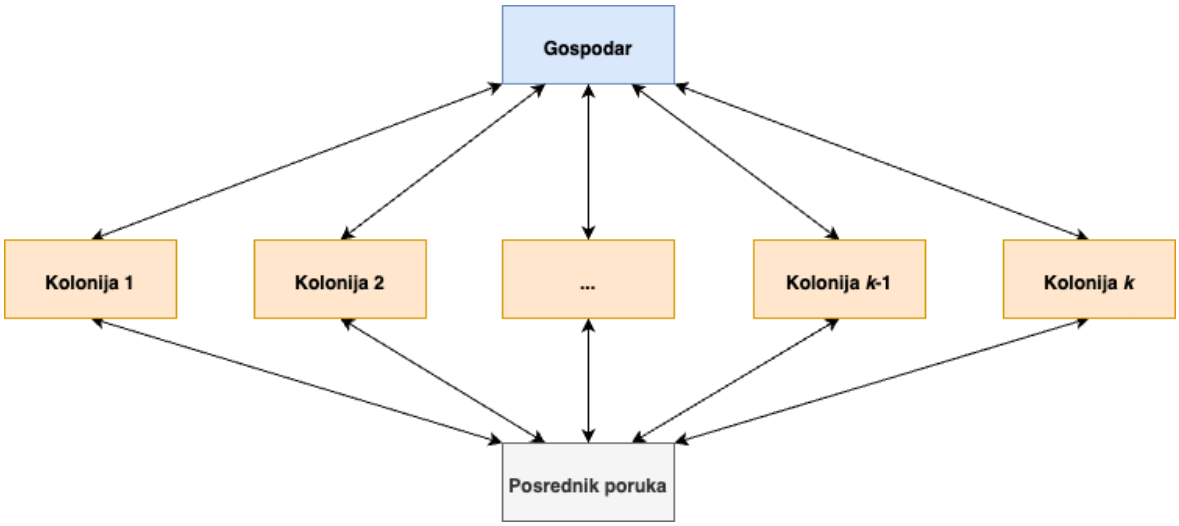
odredišna kolonija u jednom koraku razmjene ne primi rješenje, u sljedećoj razmjeni će primiti rješenje iste i bolje kvalitete, tj. nije moguće da primi lošije rješenje od onog koje je izgubljeno. Problem s potpuno povezanom topologijom je velik komunikacijski trošak jer bi svaka kolonija u svakom koraku razmjene trebala poslati svoje rješenje svim ostalima. Dakle, za k kolonija bi ukupan broj poslanih rješenja u svakom koraku razmjene bio $k \cdot (k-1)$. Da bi se ublažio ovaj nedostatak, napravljen je jednostavan posrednik poruka (eng. *message broker*). Posrednik predstavlja jednostavan UDP poslužitelj koji je implementiran na način da ima bazen dretvi (eng. *thread pool*) zahtjeva. On obavlja sljedeće poslove:

- primanje UDP paketa od kolonija koje šalju rješenje
- višeodredišno *(eng. multicast)* prosljeđivanje primljenih paketa svim ostalim kolonijama

Na taj način i kod potpuno povezane topologije svaka kolonija u svakom koraku razmjene šalje svoje rješenje samo jednom. Posrednik to rješenje šalje višeodredišno što znači da zapravo šalje rješenje samo jednom do preklopnika, a mrežni preklopnik radi replikaciju tog paket i šalje ga svim ostalim računalima, tj. kolonijama u mreži. U nastavku se nalaze ilustracije opisanih modela i topologija.



Slika 25. Kombinirani višekolonijalni i gospodar-sluga model s topologijom prstena



Slika 26. Kombinirani višekolonijalni i gospodar-sluga model s posrednikom poruka

3.5.3.Rezultati

U ovom dijelu radu nalaze se rezultati testiranja slijedne i distribuirane verzije algoritma. Za verzije bez 2-opt optimizacije korišteno je 10 000 iteracija prilikom svakog testa te su odabrani sljedeći TSPLIB problemi: kroA100, kroA150, kroA200, ch130, pr76, lin318. Duljine optimalnih rješenja su redom: 21282, 26524, 29368, 6110, 108159, 42029. Za verzije koje sadrže 2-opt heuristiku testiranje je napravljeno i za pcb444 TSBLIB problem (duljina optimalnog rješenja je 50778), a većina TSPLIB problema testirana je s 5000 iteracija. U oba slučaja svaki TSPLIB problem testiran je 10 puta kako bi se dobile najbolje i srednje vrijednosti koje se potom koriste za usporedbu performansi. Za algoritme bez 2-opt lokalne pretrage korišteni su parametri $\alpha = 1$, $\beta = 4$, $\rho = 0.02$, $\delta = 0.05$. Svakih 20 iteracija se za ažuriranje količine feromona koristi najbolje rješenje do sad, a izglađivanje feromonskih tragova aktivira se kad 80% svih mrava u iteraciji konstruira rješenje jednake duljine. Kod verzije algoritama s uključenom 2-opt optimizacijom korišten je $\rho = 0.8$. Dodatno, u slijednim verzijama algoritma je broj mrava m jednaku broju gradova n, dok je kod distribuirane verzije $m = \frac{n}{2}$ (iznimka su varijante s uključenom 2-opt lokalnom pretragom gdje je $m = 25$). Ovakav broj mrava u distribuiranoj implementaciji odabran je da bi se smanjilo vrijeme izvršavanja algoritma, ubrzalo testiranje te da bi se provjerilo hoće li više manjih kolonija pronaći isto ili bolje rješenje u kraćem vremenu (u odnosu na slijednu verziju). S obzirom da je distribuirana implementacija testirana skaliranjem algoritma na 4 procesora, pokušalo se je s $m = \frac{n}{4}$, ali je već na manjim problemima, kao što je kroA100, uočeno da takva konfiguracija daje lošije rješenje od slijedne verzije. Pretpostavka je da bi kvaliteta rješenja bila još lošija za probleme s većim brojem gradova. U nastavku se nalaze rezultati testiranja u tabličnom obliku. Sadržaj svake tablice je sljedeći:

- duljina najboljeg pronađenog rješenje
- duljina najgoreg pronađenog rješenja
- medija rješenja dobivenih u svih 10 pokretanja algoritma
- aritmetička sredina rješenja dobivenih u svih 10 pokretanja algoritma
- odstupanje prosječne kvalitete rješenja od najboljeg pronađenog rješenja – nalazi se u zagradama uz prosječnu kvalitetu rješenja
- medijalno vrijeme $t_{0.5}$ i iteracija $i_{0.5}$ za pronalazak rješenja koje odstupa 0,5% od optimalnog
- medijalno vrijeme t_1 i iteracija i_1 za pronalazak rješenja koje odstupa 1% od optimalnog
- medijalno vrijeme t_2 i iteracija i_2 za pronalazak rješenja koje odstupa 2% od optimalnog

Zbog zaokruživanja prilikom izračuna duljine pronađenog rješenja za neke probleme postoji manje odstupanje između izračunate i stvarne duljine. Tako primjerice duljine od 21285 za kroA100, 26625 za kroA150 te 29369 za kroA200 zapravo predstavljaju duljine optimalnih rješenja.

Tablica 5. Rezultati testiranja slijedne verzije bez uključenih optimizacija (2-opt, k-best, izglađivanje tragova)

Instanca	Najbolje	Najgore	Medijan	Aritmetička sredina	$i_{0.5}$	$t_{0.5}$	i_1	t_1	i_2	t_2
kroA100	**21285**	21381	21344	21338 (**0,47%**)	959	17,5	620	11,5	447	8,5
kroA150	**26625**	27019	26844	26816 (**0,72%**)			4589	271	1559	95
kroA200	**29400**	29617	29540	29519 (**0,4%**)	863	132,5	877	121	671	92
ch130	**6173**	6260	6220	6213 (**0,64%**)					3948	155,5
pr76	**108159**	109653	108159	108308 (**0,43%**)	2334	20	2305	19,5	1155,5	10
lin318	**42185**	42430	42313	42313 (**0,19%**)	4387	2405,5	1852	1040	1187	652,5

U ovom testu je vidljivo kako je slijedna verzija TBAS algoritma uspjela pronaći optimalno rješenja za kroA100, kro150 i pr76. Za kroA150 te ch130 TBAS nije uspio pronaći rješenje koje odstupa 0,5% od optimalnog, a za ch130 nije pronađeno ni rješenje čija duljina odstupa 1% od optimalne.

Tablica 6. Rezultati testiranja slijedne verzije s uključenim 2-opt (prihvati prvo bolje rješenje) i izglađivanjem tragova

Instanca	Najbolje	Najgore	Medijan	Aritmetička sredina	$i_{0.5}$	$t_{0.5}$	i_1	t_1	i_2	t_2
kroA100	**21285**	21285	28285	21285 (**0%**)	238	2	123	1	95	1
kroA150	**26525**	26741	26644	26634 (**0,29%**)	3055	36,5	1365	16	607	7,5
kroA200	**29369**	29732	29550	29535 (**0,44%**)	1910	41	1082	23	374	8
ch130	**6154**	6247	6182	6188 (**0,48%**)			3535	31	930	8

pr76	**108159**	109086	108602	108610 (**0,44%**)	544	2	491	2	245	1
lin318	**42141**	54927	42624	44787 (**10,57%**)	1706	91	2189	117	854	45
pcb444	**51147**	52092	51672	51633 (**0,48%**)			2304	247	3205	338

Iz rezultata je vidljivo kako uključivanje 2-opt algoritma lokalne pretrage u značajnoj mjeri pozitivno utječe na pronalazak boljih rješenja u kraćem vremenu. U odnosu na tablicu 5, vidi se da je ovdje TBAS uspio za sve algoritme pronaći isto ili bolje rješenje u značajno kraćem vremenu.

Tablica 7. Rezultati testiranja slijedne verzije s uključenim 2-opt (provjeri sve izmjene) i izglađivanjem tragova

Instanca	Najbolje	Najgore	Medijan	Aritme-tička sredina	$i_{0.5}$	$t_{0.5}$	i_1	t_1	i_2	t_2
kroA100	**21285**	21285	21285	21285 (**0%**)	9	0	6	0	5	0
kroA150	**26525**	26525	26525	62525 (**0%**)	69	2	24	1	10	0
kroA200	**29369**	29369	29369	29369 (**0%**)	42	2	23	1	10	0
ch130	**6110**	6128	6128	6120 (**0,15%**)	85	1,5	20	0	9	0
pr76	**108159**	108159	108159	108159 (**0%**)	19	0	8	0	4	0
lin318	**42042**	42176	42073	42093 (**0,14%**)	357	51,5	135	18	33	4
pcb444	**50949**	51189	50998	51017 (**0,14%**)	2221	603	308	82	55	14,5

Ovim testom provjerava se utjecaj ograničavanja broja koraka 2-opt algoritma Rezultati su pokazali kako je bolje omogućiti izmjenu svih bridova u 2-opt algoritmu jer se u tom slučaju značajno skraćuje vrijeme pronalaska optimalnog rješenja (slučaj za kroA100, kroA150, kroA200, ch130, pr76), vrijeme pronalaska rješenja koje odstupa od optimalnog za 0,5% (vrijedi za sve instance), a povećava se i kvaliteta finalnog rješenja (lin318, pcb44). Zbog boljih performansi u daljnjim je testovima korištena upravo ova varijanta.

Tablica 8. Rezultati testiranja slijedne verzije s uključenim 2-opt (provjeri sve izmjene), izglađivanjem tragova i k-najbolji strategije

Instanca	Najbolje	Najgore	Medijan	Aritmetička sredina	$i_{0.5}$	$t_{0.5}$	i_1	t_1	i_2	t_2
kroA100	**21285**	21285	21285	21285 (**0%**)	10	0	7	0	4	0
kroA150	**26525**	26525	26525	26525 (**0%**)	69	2	32	1	9	0
kroA200	**29369**	29369	29369	29369 (**0%**)	41	2	20	1	9	0
ch130	**6110**	6128	6110	6117 (**0,15%**)	56	1	22	0	9	0
pr76	**108159**	108159	108159	108159 (**0%**)	11	0	7	0	5	0
lin318	**42042**	42176	42104	42101 (**0,13%**)	343	42	77,5	10	24	3
pcb444	**50988**	51235	51037	51061 (**0,14%**)			382	99	61	15

U ovom testu provjerava se utjecaj strategije *k*-najbolji na kvalitetu rješenja i performanse algoritma. Za probleme manjih dimenzija (kroA100, kroA150, kroA200, ch130, pr76) vidljivo je kako je algoritam uspio pronaći optimalna rješenja kao i varijanta algoritma iz tablice 8. Kod problema s većom dimenzijom (lin318, pcb444) je ova strategija pokazala nešto lošije performanse, tj. medijan kvalitete rješenja je veći u odnosu na algoritam iz tablice 8. Također je vidljivo kako ova strategija ostvarila bolje medijalno vrijeme potrebno za pronalazak rješenja koje odstupaju od optimalnog za 0,5%, 1% i 2%. Zbog spomenutog u testovima distribuirane implementacije nije korištena ova strategija jer se tamo očekuje najveća razlika upravo za probleme s većim brojem gradova.

Tablica 9. Rezultati testiranja raspodijeljene verzije bez uključenih optimizacija (prsten topologija, razmjena svakih 10 iteracija)

Instanca	Najbolje	Najgore	Medijan	Aritmetička sredina	$i_{0.5}$	$t_{0.5}$	i_1	t_1	i_2	t_2
kroA100	**21285**	21307	21285	21287 (**0,03%**)	1170	12	706	7	485	5
kroA150	**26625**	26981	26764	26787 (**0, 45%**)			3236	109	1773	59,5
kroA200	**29369**	29574	29406	29417 (**0,2 %**)	1264	92	906	65	726	54

ch130	**6183**	6260	6221	6224 (**0,36%**)					2627	57
pr76	**108159**	108159	108159	108159 (**0%**)	1402	13,5	1147	11	700	6
lin318	**42213**	42319	42239	42258 (**0,1%**)			2570	745	1259	362

U ovim testu provjerava se koliko utjecaj distribucije originalnog algoritma (bez optimizacija) na njegove performanse. S obzirom da je algoritam distribuiran na 4 procesora, gdje je veličina svake kolonije duplo manja nego kod slijedne verzije, ukupno trajanja izvođenja algoritma bi se u trebalo prepoloviti. Iz rezultata je vidljivo kako ima smisla distribuirati algoritam i bez korištenja 2-opt heuristike. Za 4/6 probleme medijan kvalitete finalnog rješenja je značajno manji. Za preostala dva problema medijan je jednak (pr76) ili zanemarivo veći (ch130). Prosječno ubrzanje prilikom pronalaska rješenja koje odstupa 2% od optimalnog je $s_m = 1{,}87$ što predstavlja sublinearno ubrzanje s efikasnošću od $e_m = \frac{1{,}87}{4}\ x\ 100\% = 46{,}75\%$.

Tablica 10. Rezultati testiranja raspodijeljene verzije bez uključenih optimizacija (prsten topologija, razmjena svakih 50 iteracija)

Instanca	Najbolje	Najgore	Medijan	Aritme-tička sredina	$i_{0.5}$	$t_{0.5}$	i_1	t_1	i_2	t_2
kroA100	**21285**	21381	21285	21300 (**0,15%**)	1173	11	694	7	506	5
kroA150	**26608**	26823	26718	26713 (**0,31%**)			4721	153	1385	45,5
kroA200	**29404**	29599	29410	29430 (**0,2%**)	1176	85	955	69	810	58
ch130	**6118**	6241	6197	6197 (**0,55%**)					4254	86
pr76	**108159**	109653	108159	108622 (**0,66%**)	1545	14	1150	10	721	7
lin318	**42194**	42382	42239	42255 (**0,15%**)			3229	918	1527	434

U ovom testu provjerava se utjecaj manje frekvencije razmjene rješenja na performanse algoritma. Iz rezultata se vidi kako je za probleme s manjim brojem gradova bolje imati manju frekvenciju razmjene. Usporedbom rezultata iz tablice 9 i tablice 10 vidi se kako je manjoj frekvencijom razmjene ostvaren manji ili jednak medijan duljine rješenja za sve probleme (iznimka je kroA200) . Međutim, razmjenom svakih 10 iteracija, TBAS je uspio u

medijalno kraćem vremenu pronaći rješenja koja odstupaju 2% i 1% od optimalnog (iznimka je kroA150).

Tablica 11. Rezultati testiranja raspodijeljene verzije bez uključenih optimizacija (potpuno povezana topologija, razmjena svakih 50 iteracija)

Instanca	Najbolje	Najgore	Medijan	Aritmetička sredina	$i_{0.5}$	$t_{0.5}$	i_1	t_1	i_2	t_2
kroA100	**21285**	21381	21285	21306 (**0,19%**)	1232	13	684	7	522	5,5
kroA150	**26608**	26878	26676	26719 (**0,39%**)			3791	127	1078	35,5
kroA200	**29394**	29416	29410	29408 (**0,02%**)	1056	73,5	910	66	738	51,5
ch130	**6188**	6273	6221	6225 (**0,43%**)					3178	72
pr76	**108159**	109653	108159	108757 (**0,71%**)	1882	18	1681	16	720	7
lin318	**42194**	42398	42273	42276 (**0,18%**)			2576	738	1356	382

U ovom testu provjerava se utjecaj potpuno povezane topologije na performanse algoritma. Pomoću ove topologije algoritam je uspio pronaći rješenja iste kvalitete kao i kod topologije prstena, a iznimka su kroA200 i ch130 gdje je pronađeno bolje, odnosno lošije rješenje. Iz rezultata se vidi da je TBAS algoritmu koristeći ovu topologiju bilo potrebno manje vremena da pronađe rješenje koje odstupan 2% od optimalnog (iznimka je kroA150).

Tablica 12. Rezultati testiranja raspodijeljene verzije s uključenim 2-opt (isprobaj sve permutacije) i izglađivanjem tragova (prsten topologija, razmjena svakih 50 iteracija)

Instanca	Najbolje	Najgore	Medijan	Aritmetička sredina	$i_{0.5}$	$t_{0.5}$	i_1	t_1	i_2	t_2
kroA100	**21285**	21285	21285	21285 (**0%**)	7	0	5	0	3	0
kroA150	**26525**	26525	26525	26525 (**0%**)	43	1	17	0,5	7	0
kroA200	**29369**	29369	29369	29369 (**0%**)	38	2	11	1	8	0
ch130	**6110**	6110	6110	6110 (**0%**)	36	1	13	0	7	0

pr76	**108159**	108159	108159	108159 (**0%**)	9	0	7	0	3	0
lin318	**42042**	42165	42042	42072 (**0,11%**)	184	25,5	60	8	14	2
pcb444	**50959**	51025	50986	50989 (**0,05%**)	2010	558	277	75,5	29	8,5

Ovaj test zapravo predstavlja najrealniji slučaj u kojem se uspoređuje najbolja varijanta TBAS algoritma (prema provedenim testovima) s najboljom distribuiranom implementacijom (prema testovima autora). Iako je razmjena svakih 50 iteracija u prethodnom testu pokazala nešto lošije rezultate od razmjene svakih 10 iteracija, u ovom je slučaju odabrana jer su Twomey, Stützle, Dorigo, Manfrin, Birattari [27] pokazali kako je u slučaju korištenja 2-opt lokalne pretrage bolje koristit strategije s manjom frekvencijom razmjene jer 2-opt sam po sebi stavlja veći naglasak na pretragu užeg prostora rješenja pa bi pretjerana razmjena rješenja stvorila efekt jedne velike kolonije te bi algoritam prerano stagnirao. Testovi su pokazali kako je distribuirana implementacija uspjela pronaći rješenja iste kvalitete kao i slijedna (iznimka je pcb444), ali u značajno kraćem vremenu. S obzirom da su oba algoritma uspjela pronaći rješenja čija kvaliteta odstupa 0,5% od optimalnog, vremena $t_{0,5}$ korištena su za izračun ubrzanja i efikasnost algoritma. Postignuo prosječno ubrzanje u tom slučaju iznosi $s_m = 1{,}55$ što predstavlja sublinearno ubrzanje s efikasnošću od $e_m = \frac{1{,}55}{4}\ x\ 100\% = 38{,}75\%$. Za vremena t_2 postignuto je ubrzanje od $s_m = 1{,}71$ s efikasnošću $e_m = \frac{1{,}71}{4}\ x\ 100\% = 42{,}75\%$, dok je za t_1 postignuto ubrzanje od $s_m = 1{,}67$ s efikasnošću od $e_m = \frac{1{,}67}{4}\ x\ 100\% = 41{,}75\%$. Ubrzanju su najviše pridonijeli testovi na problemima s većim brojem gradova (lin318, pcb444).

4. Zaključak

U ovom radu naglasak je stavljen na distribuiranu implementaciju TBAS algoritma koji spada u kategoriju ACO algoritama. Odabrani višekolonijalni model pokazao se je kao dobar odabir u smislu jednostavnost implementacije i performansi koje se njime ostvaruju. Zbog nedostataka ozbiljnije infrastrukture algoritmi su testirani na problemima s manjim brojem gradova (najveći je pcb444 s 444 grada). U skladu s očekivanjima, najveće prosječno ubrzanje od $s_m = 1{,}87$ postignuto je usporedbom performansi između slijedne i raspodijeljene verzije algoritama koji ne koriste optimizacije poput 2-opt, dok je korištenjem 2-opt lokalne pretrage postignuto prosječno ubrzanje od $s_m = 1{,}71$ za vremena t_2. U potonjem slučaju su značajna ubrzanja ostvarena samo za probleme s većim brojem gradova. To znači da distribucija ACO algoritama u svrhu smanjenja vremena izvršavanja ima najviše smisla za probleme većih

dimenzija gdje korišteni algoritmi lokalne pretrage više nisu toliko učinkoviti u odnosu na probleme s manjim brojem gradova.

Očekivalo se je da će distribuirane varijante pronaći bolja rješenja za probleme većih dimenzija (lin318, pcb444), a rezultati su pokazali da su uglavnom pronađena rješenja iste kvalitete, ali u kraćem vremenu. Neki od načina povećanja kvalitete rješenja su korištenje boljih algoritama lokalne pretrage poput 3-opt, ali i korištenje asinkronih modela razmjena rješenja ili sinkronih s povećavajućom frekvencijom. Na taj način bi se stvorila dovoljno velika raznolikost između kolonija koji bi omogućila pretraživanje dovoljno velikog prostora rješenja kako bi algoritam na kraju generirao bolje rješenje. Također, iako izglađivanje tragova doprinosi boljoj kvaliteti rješenja, reinicijalizacija tragova nakon određenog broja iteracija bez poboljšanja bi mogla dati bolje rezultate kod ACO algoritama koji koriste lokalnu pretragu. Posebice ukoliko se koristi velik broj iteracija poput 10 000.

Distribucija ACO algoritama nudi i određenu fleksibilnost u pogledu korištenih modela, strategija i topologija. Tako je primjerice moguće koristiti otočne modele i povezati nekoliko različitih biološki inspiriranih algoritama s različitim karakteristikama kako bi se ostvario zadani cilj. Moguće je i pokrenuti isti algoritam s različitim parametrima gdje se primjerice može kontrolirati brzina konvergencije algoritma. Takve metode bi trebale biti korisne u heterogenim sustavima. Danas se sve češće koriste i implementacije na grafičkim podsustavima pomoću kojih se mogu ostvariti velika ubrzanja s faktorom 10-20 ovisno o veličini problema [32].

Popis literature

[1] A. Darwish, “Bio-inspired Computing: Algorithms review, deep analysis, and the scope of applications,” *Future Computing and Informatics Journal*, vol. 3, no. 2, pp. 231–246, 2018. doi:10.1016/j.fcij.2018.06.001

[2] A. P. Castaño, *Practical Artificial Intelligence: Machine Learning, Bots, and Agent Solutions Using C*. New York: Apress, 2018.

[3] X. Fan *et al.*, “Review and classification of bio-inspired algorithms and their applications,” *Journal of Bionic Engineering*, vol. 17, no. 3, pp. 611–631, 2020. doi:10.1007/s42235-020-0049-9

[4] A. Y. Alanis, N. Arana-Daniel, and C. Lopez-Franco, *Bio-Inspired Algorithms for Engineering*. Oxford, UK: Butterworth-Heinemann, an imprint of Elsevier, 2018.

[5] M. Čupić, *Prirodom inspirirani optimizacijski algoritmi. Metaheuristike,* 2013

[6] M. Dorigo and T. Stützle, *Ant Colony Optimization*. Cambridge, MA: MIT Press, 2004.

[7] S. Bailey, „The Secret Lives of Ants“, 1990. [Na internetu]. Dostupno: https://www.uky.edu/Ag/Entomology/ythfacts/allyr/ants.htm [pristupano 3.8.2023.].

[8] T. Stützle and H. Hoos, “Improvements on the ANT-system: Introducing the max-min ant system,” *Artificial Neural Nets and Genetic Algorithms*, pp. 245–249, 1998. doi:10.1007/978-3-7091-6492-1_54

[9] T. Stützle and H. H. Hoos, “Max-Min ant system,” *Future Generation Computer Systems*, vol. 16, no. 8, pp. 889–914, 2000. doi:10.1016/s0167-739x(00)00043-1

[10] S. Pérez-Carabaza, A. Gálvez, and A. Iglesias, “Rank-based ant system with originality reinforcement and pheromone smoothing,” *Applied Sciences*, vol. 12, no. 21, p. 11219, 2022. doi:10.3390/app122111219

[11] N. Ivković and M. Golub, “A new ant colony optimization algorithm: Three bound ant system,” *Lecture Notes in Computer Science*, pp. 280–281, 2014.

[12] T. Mattson, *A „Hands-on“ Introduction to OpenMP*, 2013. [Na internetu]. Dostupno: OpenMP, https://openmp.org/wp-content/uploads/Intro_To_OpenMP_Mattson.pdf [pristupano 10.8.2023].

[13] M. Potkonjak and J. Rabaey, “Optimizing Resource Utilization using transformations,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, no. 3, pp. 277–292, 1994. doi:10.1109/43.265670

[14] K. Nadiminti, M. D. de Assunção, and R. Buyya, “Distributed systems and recent innovations: Challenges and benefits,” *InfoNet Magazine*, pp. 1–5, 2006.

[15] M. Pedemonte, S. Nesmachnow, and H. Cancela, “A survey on parallel ant colony optimization,” *Applied Soft Computing*, vol. 11, no. 8, pp. 5181–5197, 2011. doi:10.1016/j.asoc.2011.05.042

[16] T. Stützle, “Parallelization strategies for Ant Colony Optimization,” *Lecture Notes in Computer Science*, pp. 722–731, 1998. doi:10.1007/bfb0056914

[17] R. Michel and M. Middendorf, “An island model based ant system with lookahead for the shortest supersequence problem,” *Lecture Notes in Computer Science*, pp. 692–701, 1998. doi:10.1007/bfb0056911

[18] M. Middendorf, F. Reischle, and H. Schmeck, *Journal of Heuristics*, vol. 8, no. 3, pp. 305–320, 2002, doi: https://doi.org/10.1023/a:1015057701750.

[19] L. Melo, F. B. Pereira, and E. Costa, “MC-ANT: A Multi-Colony Ant Algorithm,” *Lecture Notes in Computer Science,* pp. 25–36, Jan. 2010, doi: https://doi.org/10.1007/978-3-642-14156-0_3

[20] P. González, R. Prado-Rodriguez, A. Gábor, J. Saez-Rodriguez, J. R. Banga, and R. Doallo, “Parallel ant colony optimization for the training of cell signaling networks,” *Expert Systems with Applications*, vol. 208, p. 118199, Dec. 2022, doi: https://doi.org/10.1016/j.eswa.2022.118199.

[21] M. Golub, *Genetski algoritmi - Drugi dio*, 2004.
[22] Y.-J. Gong et al., “Distributed evolutionary algorithms and their models: A survey of the state-of-the-art,” *Applied Soft Computing*, vol. 34, pp. 286–300, Sep. 2015, doi: https://doi.org/10.1016/j.asoc.2015.04.061.
[23] J. Kacprzyk and W. Pedrycz, *Springer Handbook of Computational Intelligence*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015.
[24] M. Randall and A. Lewis, “A parallel implementation of ant colony optimization,” *Journal of Parallel and Distributed Computing*, vol. 62, no. 9, pp. 1421–1432, 2002. doi:10.1006/jpdc.2002.1854
[25] E. Alba, *Parallel Metaheuristics: A New Class of Algorithms*. Hoboken, NJ: John Wiley, 2005.
[26] S. Ilie and C. Bădică, “Multi-agent approach to distributed ant colony optimization,” *Science of Computer Programming*, vol. 78, no. 6, pp. 762–774, 2013. doi:10.1016/j.scico.2011.09.001
[27] C. Twomey, T. Stützle, M. Dorigo, M. Manfrin, and M. Birattari, “An analysis of communication policies for homogeneous multi-colony ACO algorithms,” *Information Sciences*, vol. 180, no. 12, pp. 2390–2404, 2010. doi:10.1016/j.ins.2010.02.017
[28] Ananth Grama, *Introduction to parallel computing*. Harlow, England ; New York: Addison-Wesley, 2003.
[29] N. Ivkovic, M. Malekovic, and M. Golub, “Extended trail reinforcement strategies for ant colony optimization,” *Swarm, Evolutionary, and Memetic Computing*, pp. 662–669, 2011. doi:10.1007/978-3-642-27172-4_78
[30] E. Alba, “Parallel evolutionary algorithms can achieve super-linear performance,” *Information Processing Letters*, vol. 82, no. 1, pp. 7–13, 2002. doi:10.1016/s0020-0190(01)00281-2
[31] N. Ivković, R. Kudelić, and M. Golub, “Adjustable pheromone reinforcement strategies for problems with efficient heuristic information,” *Algorithms*, vol. 16, no. 5, p. 251, 2023. doi:10.3390/a16050251
[32] G. D. Guerrero *et al.*, “Comparative evaluation of platforms for parallel ant colony optimization,” *The Journal of Supercomputing*, vol. 69, no. 1, pp. 318–329, 2014. doi:10.1007/s11227-014-1154-5

Popis slika

Slika 1. Klasifikacija biološki inspiriranih algoritama (Prema [3]) 4
Slika 2. Klasifikacija evolucijskog računanja (Prema [5]) 5
Slika 3. Prikaz rada genetskog algoritma 6
Slika 4. Ilustracija ažuriranja položaja čestice kod PSO algoritma (Prema *[5]*) 8
Slika 5. Prvi eksperiment dvokrakog mosta (Prema *[6]*) 10
Slika 6. Drugi eksperiment dvokratkog mosta (Prema *[6]*) 11
Slika 7. Treći eksperiment dvokrakog mosta (Prema *[6]*) 11
Slika 8. Primjer strukture na kojoj se temelji rad ACO algoritma 12
Slika 9. Usporedba kvalitete rješenja ACO algoritama (Izvor *[10])* 17
Slika 10. TBAS - porast vremena izvršavanja algoritma 21
Slika 11. Hijerarhijski pregled taksonomije distribuiranih ACO modela 24
Slika 12. Ilustracija modela gospodar-sluga - prvi pristup (Prema [16]) 24
Slika 13. Ilustracija modela gospodar-sluga - drugi pristup (Prema *[16]*) 25
Slika 14. Ilustracija celularnog modela 26
Slika 15. Ilustracija višekolonijalnog modela 26
Slika 16. Ilustracija više-agentnog modela (Prema *[26]*) 28
Slika 17. Komunikacijske topologije 29
Slika 19. Prikaz sublinearnog, linearnog i superlinearnog ubrzanja 36
Slika 20. 2-opt zamjena bridova 39
Slika 21. kroA100 optimalno rješenje duljine 21282 40
Slika 22. kroA100 rješenje duljine 21382 40
Slika 23*. Prikaz feromonske matrice neposredno prije (lijevo) i neposredno poslije (desno) aktivacije izglađivanja feromona* 42
Slika 24. Strategija k-najbolji – prikaz dodavanja i uklanjanja rješenja iz reda u $i + 1$ iteraciji (Prema *[31]*) 43

Popis tablica

Tablica 1. Vrijeme izvršavanja TSP algoritama 3
Tablica 2. Karakteristike topologija za k čvorova 31
Tablica 3. Rezultati testiranja migracijskih strategija u odnosu na PIR (Izvor [27]) 33
Tablica 4. Broja pokretanja algoritma (od 30 pokretanja) u kojima je pronađeno bolje rješenje 41
Tablica 5. Rezultati testiranja slijedne verzije bez uključenih optimizacija (2-opt, k-best, izglađivanje tragova) 47
Tablica 6. Rezultati testiranja slijedne verzije s uključenim 2-opt (prihvati prvo bolje rješenje) i izglađivanjem tragova 47
Tablica 7. Rezultati testiranja slijedne verzije s uključenim 2-opt (provjeri sve izmjene) i izglađivanjem tragova 48
Tablica 8. Rezultati testiranja slijedne verzije s uključenim 2-opt (provjeri sve izmjene), izglađivanjem tragova i k-najbolji strategije 49
Tablica 9. Rezultati testiranja raspodijeljene verzije bez uključenih optimizacija (prsten topologija, razmjena svakih 10 iteracija) 49
Tablica 10. Rezultati testiranja raspodijeljene verzije bez uključenih optimizacija (prsten topologija, razmjena svakih 50 iteracija) 50
Tablica 11. Rezultati testiranja raspodijeljene verzije bez uključenih optimizacija (potpuno povezana topologija, razmjena svakih 50 iteracija) 51
Tablica 12. Rezultati testiranja raspodijeljene verzije s uključenim 2-opt (isprobaj sve permutacije) i izglađivanjem tragova (prsten topologija, razmjena svakih 50 iteracija) 51