Poređenje primjene algoritma optimizacije kolonijom mrava i genetičkog algoritma za rješavanje problema TSP-a (engl. Travelling salesman problem)

Edina Kovač
Faculty of Electrical Engineering
University of Sarajevo
Sarajevo, Bosnia and Herzegovina
ekovac2@etf.unsa.ba

Damir Pozderac
Faculty of Electrical Engineering
University of Sarajevo
Sarajevo, Bosnia and Herzegovina
dpozderac1@etf.unsa.ba

Zerina Ramić
Faculty of Electrical Engineering
University of Sarajevo
Sarajevo, Bosnia and Herzegovina
zramic1@etf.unsa.ba

***Sažetak*—Problem trgovačkog putnika (TSP) je jedan od najpoznatijih problema kombinatorne optimizacije. Sa povećanjem broja gradova njegovo rješavanje primjenom matematičkih metoda postaje gotovo nemoguće. Iz tog razloga bolje je koristiti heurističke metode za rješavanje ovog problema kao što su genetički algoritam (GA) i optimizacija kolonijom mrava (ACO), koji su upravo iskorišteni u ovom radu. Za njihovu implementaciju koristi se programski jezik Python. Izvršeno je pokretanje implementiranih algoritama nad različitim brojem gradova i sa različitom kompleksnošću pronalaska optimalne rute. Na osnovu dobijenih rezultata, izvršeno je poređenje učinkovitosti ovih algoritama u pogledu vremena izvršenja algoritma i najkraće pronađene udaljenosti. Pokazalo se da genetički algoritam daje poprilično dobre rezultate za probleme male ili srednje veličine, ali je sporiji što se tiče vremena izvršenja u odnosu na ACO. Sa druge strane algoritam optimizacije kolonijom mrava se pokazao dominantnijim u odnosu na genetički algoritam kada se primjenjuje za rješavanje kompleksnih problema, ali je teži za implementaciju i zahtijeva više resursa.**

Ključne riječi—problem trgovačkog putnika, genetički algoritam, optimizacija kolonijom mrava, Python

***Abstract*—The Traveling Salesman Problem (TSP) is one of the most well-known combinatorial optimization problems. With the increase in the number of cities, solving it by applying mathematical methods becomes almost impossible. For this reason, it is better to use heuristic methods to solve this problem such as the Genetic Algorithm (GA) and Ant Colony Optimization (ACO), which were used in this paper. Python programming language is used for implementation. The implemented algorithms were compared over different cities and different complexity of finding the optimal route. Based on the obtained results, a comparison of these algorithms was performed over the execution time of the algorithm and the shortest found distance. The Genetic Algorithm (GA) gives pretty good results for small or medium size problems, but is slower in terms of execution time compared to ACO. On the other hand, the Ant Colony Optimization (ACO) has proven to be more dominant than the Genetic Algorithm (GA) when applied to solve complex problems, but is more difficult to implement and requires more resources.**

Keywords—The Traveling Salesman Problem, Genetic Algorithm; Ant Colony Optimization, Python

I. UVOD

Optimizacija predstavlja jedan od osnovnih zadataka inženjera, gdje se od inženjera traži da osmisli što bolji i jeftiniji sistem kao i tehnike za poboljšanje rada samog sistema. Problem trgovačkog putnika (TSP) predstavlja problem kombinatorne optimizacije, te je dokazano da je ovaj problem *NP-hard* što znači da je za njegovo rješavanje potreban jako efikasan algoritam.

Osnovna postavka ovog problema se sastoji od toga da imamo određeni broj gradova, te troškove za prelazak iz jednog grada u drugi, odnosno težine ili udaljenosti između gradova. Trgovački putnik treba posjetiti sve gradove, ali u što god moguće kraćem vremenu, pa prema tome potrebno je da pronađe odgovarajuću putanju obilaska gradova koja će minimizirati trošak, težine, odnosno udaljenosti.

Rješavanje ovog problema zahtijeva korištenje algoritama koji na efikasan način mogu riješiti ovakav tip problema. Neki od takvih algoritama su genetički algoritam (GA) i algoritam optimizacije kolonijom mrava (ACO). Ovi algoritmi pripadaju porodici metaheurističkih algoritama u koju još spadaju simulirano hlađenje, tabu pretraživanje, optimizacija rojem čestica i evolucione strategije. Svi oni teže pružiti dovoljno dobra rješenja za izazovne probleme.

Ovaj rad ima za cilj uporediti performanse genetičkog algoritma i algoritma optimizacije kolonijom mrava za različite primjere problema trgovačkog putnika. U tu svrhu korišteni su simetrični problemi. Poređenje performansi je bazirano na vremenu izvršenja algoritma i najkraćoj pronađenoj udaljenosti između gradova za određeni problem.

Postoji određeni broj autora koji su, također, zainteresovani za poređenje performansi ACO i GA u rješavanju problema trgovačkog putnika. Haroun i ostali [1] dali su značajan doprinos u poređenju ova dva algoritma za rješavanje problema trgovačkog putnika. ACO i GA su pokretali za tri različita primjera mijenjajući veličinu problema i kompleksnost. Zukhri i Paputungan [2] su predložili hibridnu verziju ACO algoritma i genetičkog algoritma pod nazivom *Genetic Ant Colony Optimization* (GACO), te su došli do zaključaka da ova verzija algoritma ima bolje performanse u odnosu na GA i ACO. Alhanjouri i Alfarra [3] su pokušali oba algoritma pokrenuti nad istim skupom podataka, te ustanovili da je GA bolji u odnosu na ACO algoritam za rješavanje problema trgovačkog putnika. Chandekar i Pillai [4] su testirali performanse ACO i GA algoritama nad primjerima koji se sastoje od slučajno generisanih brojeva, te su ustanovili da ACO algoritam ima bolje performanse u odnosu na GA za ovakvu vrstu problema. Afreen i ostali [5] su poredili performanse ACO algoritma i još pet često korištenih algoritama za rješavanje problema trgovačkog putnika, te su došli do zaključka da algoritam koji treba koristiti zavisi od samog problema na koji se dati algoritam primjenjuje.

TSP ima svoju primjenu u medicine gdje omogućava način integracije lokalnih mapa u jednu hibridnu mapu zračenja za genom. TSP je, također, korišten u okviru NASA misije za smanjenje potrošnje goriva prilikom ciljanja i izvođenja manevara za par satelita, koji su uključeni u samu misiju. Zatim, korišten je i za formiranje univerzalne DNK sekvence, koja se sastoji od većeg broja manjih sekvenci, sa ciljem dobijanja što kraće univerzalne DNK sekvence. Jedna od starijih primjena TSP-a jeste osmišljavanje rute za prikupljanje kovanica sa telefonskih govornica na određenom području [6]. Osim prethodno navedenih primjena, postoji još veliki broj problema koji se mogu predstaviti pomoću problema trgovačkog putnika.

U odjeljku 2 se nalaze osnovni podaci o problemu koji se rješava, te način rada korištenih algoritama. Odjeljak 3 pruža uvid u implementaciju algoritama koji su korišteni za rješavanje problema trgovačkog putnika, te u rezultate testiranja. U odjeljku 4 predstavljen je zaključak formiran na osnovu svega opisanog kroz rad.

II. Korišteni Algoritmi

A. Genetički algoritam

Koncept genetičkog algoritma su prvi uveli J. H. Holland i njegove kolege sredinom 1960-ih godina [7].

Genetički algoritam (ili GA) je tehnika pretraživanja koja se koristi u računarstvu kako bi se pronašla stvarna ili približna rješenja za optimizaciju i probleme pretraživanja. Spada u posebnu klasu evolucijskih algoritama koji koriste tehnike inspirisane evolucijskom biologijom kao što su nasljeđivanje, mutacija, selekcija i ukrštanje (koji se također naziva rekombinacija).

Evolucija obično počinje od populacije nasumično generisanih jedinki. Evolucija se događa kroz generacije koje predstavljaju iteracije genetičkog algoritma.

U svakoj generaciji se procjenjuje fitness svake jedinke u populaciji, više jedinki se bira iz trenutne populacije (na temelju njihovog fitnessa) koja se modificira tako da se formira nova populacija. To je motivisano nadom da će nova generacija biti bolja od stare na osnovu svojih karakteristika.

Nova populacija se koristi u sljedećoj iteraciji algoritma. Algoritam se prekida kada je dosegnut ili najveći broj generacija ili je postignut zadovoljavajući nivo fitnessa za populaciju [8].

Na osnovu opisanog predstavljen su koraci algoritma:

1) (*Početak*) Generisanje slučajne populacije n jedinki.

2) (*Fitness*) Procjena fitnessa f(x) svake jedinke x u populaciji.

3) (*Nova populacija*) Stvaranje nove populacije primjenom sljedećih koraka (sve dok se populacija ne popuni):

a) (*Selekcija*) Odabir dvije roditeljske jedinke iz populacije u skladu sa njihovim fitnessom (što je bolji fitness, veća je mogućnost da jedinka bude izabrana).

b) (*Ukrštanje*) Proces ukrštanja sa zadanom vjerovatnoćom nad roditeljima kako bi se formiralo novo potomstvo (djeca). Ako nije izvršeno nikakvo ukrštanje, potomstvo je identična kopija roditelja.

c) (*Mutacija*) Proces mutacije sa zadanom vjerovatnoćom mutira novo potomstvo (skup jedinki) na svakom lokusu (položaj u hromozomu jedinke).

d) (*Elitizam*) Zadržavanje jedne ili više jedinki sa najboljim fitnessom i njihovo bezuslovno prenošenje u novu populaciju.

e) (*Prihvatanje*) Dodavanje novog potomstva u novu populaciju.

f) (*Zamjena*) Korištenje novo formirane populacije u sljedećim koracima algoritma.

g) (*Testiranje*) Ako je krajnji uvjet zadovoljen, zaustavlja se i vraća najbolje rješenje u trenutnoj populaciji.

h) (*Petlja*) Povratak na korak 2 [9].

B. Algoritam optimizacije kolonijom mrava (ACO)

Algoritam optimizacije kolonijom mrava (ACO) predstavlja jednu od najpopularnijih metaheurističkih metoda, koje se koriste za rješavanje problema diskretne optimizacije. Postoje različite varijante algoritma, međutim, prva uspješna verzija ovog algoritma za pronalazak najkraće staze je predstavljena od strane Marca Doriga 1991. godine pod nazivom *Ant System* (AS).

Ponašanje mrava u prirodi, tačnije njihova potraga za hranom predstavlja inspiraciju za nastanak ovog algoritma. Iako je kretanje mrava u početku nasumično oni uspijevaju komunikacijom, u vidu ostavljanja feromonskog traga, usmjeriti traganje prema područjima u kojima je pronađena hrana. Ovakvo ponašanje, gdje je bitna kolonija kao cjelina, a ne mrav kao individua u direktnoj je vezi sa pronalaskom rješenja problema.

Kada mrav pronađe hranu, prilikom svog povratka pojačava prethodno ostavljeni feromonski trag. Prema tome, tokom izbora puta kojim će se kretati, mravi uvijek biraju onaj put na kojem je ostavljen jači feromonski trag. Pokazalo se da ostavljanje feromonskog traga, odnosno indirektna komunikacija omogućava mravima pronalazak najkraćeg puta do izvora hrane.

Kod jednostavnog algoritma optimizacije kolonijom mrava (S-ACO) postoje četiri osnovne operacije:

- kretanje mrava unaprijed,
- kretanje mrava unazad,
- ažuriranje feromonskog traga,
- isparavanje feromonskog traga.

Prilikom kretanja unaprijed svaki mrav formira jedno potencijalno rješenje kerečući se od početnog ka odredišnom čvoru, pri čemu iza sebe ne ostavlja feromonski trag. Kretanjem unazad mrav ostavlja feromonski trag na svim granama kojima je prošao. Ažuriranjem feromonskog traga se sama vrijednost feromonskg traga povećava što ujedno povećava i vjerovatnoću izbora date grane prilikom formiranja nekog od narednih rješenja. Da ne bi došlo do preuranjene konvergencije, koju uzrokuje stalno povećavanje feromonskog traga, vrši se smanjenje feromonskog traga kroz iteracije što omogućava da algoritam postepeno konvergira ka rješenju.

Ant System (AS) algoritam se od S-ACO algoritma razlikuje po načinu ažuriranja feromonskog traga, koji se ažurira nakon što svi mravi završe konstruiranje potencijalnih rješenja od kojih zapravo i zavisi jačina fermonskog traga kojeg će ostaviti. Zbog ove male izmjene AS ima bolje performanse u odnosu na S-ACO. Jedna od varijanti AS algoritma jeste i *elitistički AS* kod kojeg se dozvoljava najboljem mravu, od početka izvršavanja algoritma, da ostavi jači feromonski trag na grane najbolje pronađene staze. Osim elitističkog AS značajan je i *MAX-MIN AS* algoritam kod kojeg se uvodi nekoliko izmjena. Kod ovog algoritma dozvoljava se isključivo najboljem mravu trenutne iteracije ili najboljem mravu od pokretanja algoritma da ostave feromonske tragove stazama kojima su prošli. Zatim, vrijednosti feromonskog traga na granama moraju biti između vrijednosti τ_{min} i τ_{max}, koje se računaju na poseban način. Na početku se vrijednost feromonskog traga inicijalizira na τ_{max} za sve grane. Kada dođe do stagnacije pretraživanja, odnosno kada ne dolazi do poboljšanja samog rješenja izvrši se reinicijalizacija vrijednosti feromona na svim granama [10].

Ant System (AS) algoritam kao i njegove varijante su pogodni za rješavanje kompleksnih problema među koje spada i problem trgovačkog putnika (TSP).

Kod problema kombinatorne optimizacije optimalno rješenje se traži u okviru diskretnog prostora pretrage. Jedan od najpoznatijih CO problema jeste problem trgovačkog putnika (TSP) kod kojeg broj potencijalnih rješenja raste sa povećanjem veličine problema što dodatno otežava pronalazak optimalnog rješenja.

C. *Problem trgovačkog putnika (TSP)*

Problem trgovačkog putnika predstavlja problem koji opisuje situaciju u kojoj je za zadani broj gradova i udaljenosti između tih gradova potrebno pronaći najkraću moguću rutu koja prolazi kroz sve gradove tačno jednom i vraća se u polazni grad. Sam naziv problema potiče od analogije sa putnikom koji mora proći kroz gradove u određenom području pri čemu svaki grad mora posjetiti tačno jednom i na kraju svog puta se mora vratiti u grad iz kojeg je krenuo.

Iako su problemi traženja Hamiltonovog ciklusa i problem trgovačkog putnika slični problemi, treba naglasiti koje su njihove bitne razlike. Naime, problem pronalaska Hamiltonovog ciklusa predstavlja problem pronalaska rute koja obilazi svaki grad samo jednom. Međutim, problem trgovačkog putnika se sastoji u pronalasku najkraćeg Hamiltonovog ciklusa od svih mogućih [11].

Ovaj problem predstavlja jedan od najinteresantnijih i najpoznatijih problema optimizacije svih vremena. Postoje brojni razlozi za to. Naime, ovaj problem je jako jednostavno definirati, ali ga je izuzetno teško riješiti. Osim toga, on ima veliki značaj u nauci. Iako je riječ o transportnom problemu, TSP ima i razne primjene u drugim oblastima koje nisu vezane za transport. Ekvivalentan je sa velikom klasom bitnih problema optimizacije. To znači da bi pronalazak optimalnog rješenja za problem trgovačkog putnika značio rješavanje i velikog broja drugih problema optimizacije [12].

Problem trgovačkog putnika se može definirati nad grafovima pri čemu čvorovi grafa predstavljaju gradove a grane grafa su udaljenosti između njih. Ovaj problem spada u klasu *NP-hard* problema. NP problemi su problemi za koje nije poznat algoritam koji može pronaći rješenje u polinomijalnom vremenu ali rješenje ovakvog problema je moguće verificirati u polinomijalnom vremenu. *NP-hard* (NP-teški) problemi imaju svojstvo da se svaki problem iz klase NP može polinomijalno reducirati na njih, a da pri tome ti problemi ne moraju nužno pripadati klasi NP [13].

TSP je naizgled jednostavan problem. Naime, naivni pristup rješavanju ovog problema bi se sastojao u pronalasku svih Hamiltonovih ciklusa u nekom grafu nakon čega bi se izabrao onaj sa najmanjom dužinom (najmanjom cijenom). Međutim, ovakav način pretrage grafa je izuzetno vremenski zahtjevan. Ukoliko bismo imali graf sa 10 čvorova (gradova), tada bi bilo moguće kreirati preko 150000 različitih Hamiltonovih ciklusa. Ovaj broj rapidno raste te dostiže vrijednost od nekoliko desetina milijardi za samo 15 čvorova [14].

TSP je problem kojim se bave naučnici već mnogo godina unazad. Predložene su različite metode za rješavanje ovog problema. Naravno, i dalje se traga za boljim rješenjem. Neki od predloženih pristupa koriste heurističke i metaheurističke algoritme kao što su genetički algoritam (GA) i algoritam optimizacije kolonijom mrava (ACO).

1) *TSP koristeći genetički algoritam (GA)*

Da bi se genetički algoritam mogao primijeniti na problem trgovačkog putnika potrebno je izvršiti određene izmjene. Postoje mnogi pristupi, a za potrebe ovog rada izvršene su izmjene koje će biti navedene u nastavku.

Za najlakšu manipulaciju kroz algoritam koristi se genetički algoritam sa realnim kodiranjem.

Potrebno je prilagoditi funkciju cilja, tako da vrši računanje puta između čvorova koji su joj proslijeđeni.

Prvo pitanje koje se postavlja jeste kako predstaviti ciklus u obliku hromozoma neke jednike. Hromozom je moguće predstaviti kao bilo koju permutaciju čvorova. Kako problem generisanja permutacija raste sa porastom čvorova, potrebno je iskoristiti suženi skup permutacija. Okolina nekog rješenja, odnosno suženi skup permutacija, se može formirati mijenjanjem pozicije svakog čvora sa pozicijom svih ostalih čvorova trenutnog rješenja.

Kako bi se postiglo najbolje rješenje u najmanje koraka početnu populaciju je potrebno popuniti sa hromozomima koji predstavljaju cikluse dobijene pohlepnim algoritmom. Da se ne bi desio slučaj da sve jedinke u početnoj populaciji budu iste, u početnu populaciju se sa malom vjerovatnoćom dodaju i drugi ciklusi iz suženog skupa permutacija.

Nakon što je definisana početna populacija, potrebno je definisati operatore mutacije i ukrštanja. Ovo je izuzetno bitno jer ovi operatori moraju kreirati cikluse koji su dozvoljeni, jer u suprotnom algoritam neće moći naći način da se vrati u skup dozvoljenih rješenja.

Kao operator mutacije se može uzeti operator koji uzima slučajnim odabirom dva čvora u ciklusu i mijenja im mjesta. Na ovaj način se dobija dozvoljena permutacija.

Da bi se izbjegla potreba za popravljanjem, operator ukrštanja je moguće definisati kao operator ukrštanja u dvije

tačke, gdje se dvije tačke biraju nasumično, ali sa izmjenama. Dio hromozoma koji se nalazi između te dvije tačke se prenosi u novi hromozom na iste pozicije, dok se za ostale pozicije odabiru čvorovi jednog od roditelja vodeći računa da čvor koji se umeće ne postoji već u hromozomu.

Kao najuspješniji i najbrži mehanizam selekcije se pokazao mehanizam selekcije proporcionalan fitnessu ili poznatiji kao mehanizam selekcije na bazi točka ruleta (Roullete Wheel Selection – RWS) [10].

2) TSP koristeći algoritam optimizacije kolonijom mrava (ACO)

Algoritam optimizacije kolonijom mrava (ACO) se može direktno primijeniti za rješavanje problema trgovačkog putnika, odnosno nije potrebno izvršavati određene prilagodbe kao što je to bio slučaj za genetički algoritam.

Čvorovi odgovaraju gradovima, a grane koje povezuju čvorove odgovaraju putevima koji povezuju gradove. Svakoj grani je dodijeljena težina, koja odgovara udaljenosti između gradova.

Jedino ograničenje koje postavlja TSP je što svi gradovi moraju biti posjećeni barem jednom. U okviru ACO algoritma ovaj korak podrazumijeva da mrav prilikom kreiranja rješenja prelazi isključivo u onaj čvor koji prethodno nije posjetio.

Vrijednost feromonskog traga $\tau_{i,j}$, koji mravi ostavljaju po stazama kojima su prošli, u okviru TSP-a predstavlja poželjnost posjete određenog grada, a vrijednost za koju će se uvećati feromonski trag je gotovo uvijek obrnuto proporcionalna udaljenosti između gradova.

Svaki mrav kreće od slučajno odabranog grada (čvora) i prilikom svog kretanja, iterativno dodaje po jedan neposjećeni grad u svoje rješenje. Konstruiranje rješenja se završava nakon što mrav posjeti sve gradove, odnosno čvorove i vrati se u grad iz kojeg je i krenuo.

III. Simulacijski Rezultati

U radu se vrši poređenje rada algoritma optimizacije kolonijom mrava i genetičkog algoritma. U nastavku su prikazani pseudokodovi i kratak opis implementacije svakog od algoritama, a nakon toga mjerenja i rezultati eksperimenata.

A. Genetički algoritam (GA – Genetic Algorithm)

U nastavku je priložen pseudokod na kojem se temelji implementacija programa. U prethodnom poglavlju odjeljak C detaljno opisuje logiku implementacije svake funkcije pojedinačno i na koji način je bilo potrebno prilagoditi inicijalni genetički algoritam za primjenu na TSP.

Algoritam 1 Genetički algoritam

```
populacija ← FormiratiPocetnuPopulaciju()
velicinaPopulacije ← VelicinaPocetnePopulacije()

repeat
  novaPopulacija = []
  j ← 0
  repeat
     individua ← populacija.SelekcijaRTocak()
     Uvrstiti individuu izabranu pomoću SelekcijaRTocak() u
          novaPopulacija
     j ← j + 1
  until j = velicinaPopulacije

  Sortiraj(novaPopulacija)
  j ← 0
  repeat
     u ← Random(0, 1)
     if u < VjerovatnocaUkrstanja() then
        individua1, individua2 ←
           novaPopulacija.OpUkrstanjaDvijeTacke ()
        Uvrstiti individue dobijene pomoću
           OpUkrstanjaDvijeTacke () u novaPopulacija
     j ← j + 1
  until j = velicinaPopulacije

  j ← 0
  repeat
     u ← Random(0, 1)
     if u < VjerovatnocaMutacije() then
        novaPopulacija[u] ← OpMutacije()
     j ← j + 1
  until j = velicinaPopulacije

  j ← 0
  repeat
        p ← Elitizam()
        Zamijeniti inviduu s najmanjim fitnessom sa p
        j ← j + 1
  until j = VelicinaElite()

  j ← 0
  repeat
        novaPopulacija[j].Evaluiraj()
        j ← j + 1
  until j = VelicinaPopulacije
until UslovZaustavljanja()
```

B. *Mravlji sistem (AS – Ant System)*

U ovom radu su implementirane tri verzije algoritma optimizacije kolonijom mrava. Pošto se u ovom radu opisuje primjena ACO algoritma na rješavanje problema trgovačkog putnika, najprije je implementiran mravlji sistem (AS – Ant System). Funkcija u kojoj je implementiran ovaj mehanizam se naziva *ACO*. Njen pseudokod je prikazan kao Algoritam 1 i u nastavku će detaljnije biti opisan.

Algoritam 2 Algoritam optimizacije kolonijom mrava

```
xZvijezda = []
vZvijezda = float ("inf")
τ = []
v0 = f (x0, tezinskaMatrica)
pocetnoTau = brojMrava / v0
Inicijalizirati matricu feromonskog traga tau na osnovu vrijednosti pocetnoTau
repeat
    omegaPrim = []
    omegaV = []
    for k in range (brojMrava):
        xPrim = KonstruirajRjesenje (α, β, τ, tezinskaMatrica)
        vPrim = f (xPrim, tezinskaMatrica)
        if vPrim < vZvijezda then
            xZvijezda = xPrim
            vZvijezda = vPrim
        end if
        omegaPrim.append (xPrim)
        omegaV.append (f (xPrim, tezinskaMatrica))
    τ = IsparavanjeFeromonskogTraga (τ, ρ)
    for all xPrim in omegaPrim do
        AzurirajFeromonskiTrag (xPrim)
    end for
until UslovZaustavljanja ()
```

Funkciji se kao ulazni parametri šalju funkcija *f* koja računa dužinu pređenog puta, *x0* kao početno rješenje koje je dobijeno pohlepnim algoritmom, *tezinskaMatrica* koja sadrži udaljenosti između pojedinih čvorova (ova matrica je simetrična), *brojMrava* koji predstavlja veličinu same kolonije mrava, *brojIteracija* koji predstavlja broj iteracija koje je potrebno izvršiti prije završetka algoritma te parametri *α, β* i *ρ* koji predstavljaju parametre algoritma. Parametar *α* određuje značaj trenutnih vrijednosti smještenih u matrici *τ*, dok parametar *β* određuje koliko je poželjan prelazak preko grane *(i, j)*. Oba ova parametra se koriste prilikom konstruiranja rješenja dok se parametar *ρ* koristi prilikom isparavanja feromonskog traga.

Prilikom pokretanja funkcije, najprije se inicijalizira vrijednost početnog rješenja na osnovu *x0*. Nakon toga, inicijalizira se matrica feromonskog traga korištenjem vrijednosti varijable *pocetnoTau*. Ova varijabla se dobije kao količnik veličine kolonije mrava i vrijednosti početnog rješenja. Matrica feromonskog traga *τ* ima vrijednosti 0 na glavnoj dijagonali dok su joj ostale vrijednosti ispunjene vrijednosti *pocetnoTau*.

Centralni dio algoritma se izvršava onoliko puta koliko iznosi vrijednost varijable *brojIteracija*. U ovom dijelu, *omegaPrim* i *omegaV* predstavljaju liste koje se sastoje od čvorova koje su posjetili svi mravi u toku jedne iteracije i dužine svakog od tako kreiranih puteva, respektivno. U svakoj iteraciji algoritma svi mravi (kojih ima onoliko koliko iznosi vrijednost varijable *brojMrava*) konstruiraju po jedno rješenje. Ovaj dio posla obavlja funkcija *KonstruirajRjesenje*. Ova funkcija svakog mrava inicijalno postavlja u neki čvor grafa (grad) te nakon toga na osnovu matrice feromonskog traga i parametara *α* i *β* određuje putanju koju će taj mrav pratiti. Da bi se znalo koji je sljedeći čvor koji će posjetiti k-ti mrav, kreira se dozvoljena okolina za čvor u kojem se trenutno nalazi taj mrav. Dozvoljena okolina se sastoji od tačaka koje mrav do sada nije posjetio u trenutnoj iteraciji algoritma. Nakon što svi mravi kreiraju rješenja (putanje kojima su se kretali), određuje se da li je neko od rješenja dobivenih u trenutnoj iteraciji bolje od trenutno najboljeg rješenja. Ukoliko jeste, to rješenje postaje novo najbolje rješenje.

Pošto je matrica feromonskog traga jedan od ključnih dijelova algoritma, u nastavku se vrši isparavanje pa ažuriranje feromonskog traga. Ispravanje feromonskog traga se obavlja množenjem matrice feromonskog traga vrijednošću *(1-ρ)*. Ovo je bitan korak jer se na ovaj način onemogućava nagomilavanje feromona na određenim granama.

Naredni korak je ažuriranje feromonskog traga koje se obavlja tako što se na svim granama koje su mravi posjetili u trenutnoj iteraciji algoritma uveća vrijednost feromonskog traga. Vrijednost za koju će se uvećati feromonski trag iznosi $\frac{1}{d_{i,j}}$ gdje $d_{i,j}$ predstavlja dužinu grane *(i, j)*.

Uslov zaustavljanja u ovom algoritmu predstavlja ukupan broj iteracija. Optimalno rješenje koje je *ACO* funkcija pronašla je smješteno u varijabli *xZvijezda* dok se u varijabli *vZvijezda* nalazi dužina optimalnog puta kroz čvorove (gradove) u grafu.

C. *Elitistički mravlji sistem (EAS – Elitist Ant System)*

Implementacija koja koristi elitistički mravlji sistem je slična implementaciji klasičnog mravljeg sistema. Glavna razlika između ove dvije implementacije leži u načinu ažuriranja feromonskog traga. Funkcija *elitističkiACO* implementira elitistički mravlji sistem. Ova funkcija ima iste parametre kao i funkcija *ACO* uz jedan dodatni parametar *e*. Ovaj parametar se koristi prilikom ažuriranja matrice feromonskog traga. Naime, omogućava se ostavljanje dodatnog feromonskog traga na putanju koja je najbolja od početka izvršavanja algoritma. Vrijednost feromonskog traga koja se dodaje granama koje se nalaze na toj putanji je jednaka količniku parametra *e* i dužine najbolje staze.

Za vrijednost parametra *e* se najčešće koristi broj čvorova (gradova) u grafu. Korištenjem elitističkog mravljeg sistema algoritam brže konvergira nego što je to slučaj kada se koristi klasični mravlji sistem.

D. *MAX-MIN mravlji sistem (MMAS – MAX-MIN Ant System)*

Implementacija koja koristi MAX-MIN mravlji sistem je jako slična implementaciji pomoću klasičnog mravljeg sistema. Postoje ukupno četiri razlike koje uvodi ova implementacija. Prva razlika je u tome što u okviru trenutne iteracije feromonski trag ostavlja samo najbolji mrav od pokretanja algoritma ili najbolji mrav trenutne iteracije. Koji

mrav će ostaviti feromonski trag se bira slučajno. Vrijednost feromonskog traga na granama mora biti između vrijednosti τ_{min} i τ_{max}, pri čemu se prilikom prekoračenja vrijednosti τ_{max} vrijednost feromonskog traga vraća na vrijednost τ_{max}, a ukoliko vrijednost feromonskog traga postane manja od τ_{min}, onda se sama vrijednost vrati na τ_{min}. τ_{max} se računa kao količnik broja jedan i proizvoda od ρ i vrijednosti do tada najboljeg pronađenog rješenja, a τ_{min} se računa kao proizvod τ_{max} i 0.001. Varijabla *pocetnoTau* se računa kao količnik broja jedan i proizvoda od ρ i vrijednosti početnog rješenja. Kao dodatni parametar funkciji se prosljeđuje broj iteracija u okviru kojih može rješenje stagnirati, međutim nakon tog broja iteracija vrši se reinicijalizacija vrijednosti feromonskog traga na svim granama, postavljajući same vrijednosti na τ_{max}. Ovim korakom se pretraživanje prevodi u slučajno pretraživanje i time se naglašava proces istraživanja.

E. Simulacijski rezultati

U svrhu testiranja i mjerenja performansi implementiranih algoritama korišteni su grafovi sa različitim brojem čvorova (gradova). Preciznije, korišteno je 5 različitih grafova koji imaju 7, 13, 17, 26 i 29 čvorova, respektivno. Kako bi se performanse mogle mjeriti na pravilan način, testiranje je urađeno sa različitim brojem iteracija i različitim brojem jedinki/mrava. Korišteno je 100, 1000 i 2000 iteracija za prvi graf dok je za ostala četiri grafa korišteno 100, 1000 i 3000 iteracija. Osim broja iteracija, korišten je različit broj jedinki/mrava. Korišteno je 5 jedinki/mrava za sve grafove te je za svaki od grafova posebno korišten i broj jedinki/mrava koji je jednak broju čvorova tog grafa. Kombinacijom različitih iteracija i različitog broja jedinki/mrava ukupno je kreirano 6 testnih slučajeva za svaki pojedinačni graf. Ovi testni slučajevi su pokrenuti nad prethodno implementiranim genetičkim algoritmom kao i na sve tri verzije algoritma optimizacije kolonijom mrava (mravlji sistem, elitistički mravlji sistem i MAX-MIN mravlji sistem).

Svaki testni slučaj je pokrenut po 10 puta te je kao konačni rezultat uzeta najbolja vrijednost koju je određeni algoritam vratio za dati testni slučaj. Osim toga, mjereno je i srednje vrijeme izvršavanja algoritma. Nakon što su svi testni slučajevi pokrenuti nad svim implementiranim algoritmima sa različitim brojem iteracija i različitim brojem jedinki/mrava, izračunat je stepen greške. Stepen greške je izražen u postocima i govori nam koliko se rješenje dobiveno nekim od algoritama razlikuje u odnosu na stvarno rješenje koje se trebalo dobiti.

Naravno, svaki od algoritama koji su opisani i implementirani u ovom radu, pored broja iteracija i broja jedinki/mrava, zahtijevaju i dodatne parametre. Ovi dodatni parametri su izabrani tako da daju najbolje rezultate za konkretni testni slučaj. Dakle, dodatni parametri su fiksni za određenu implementaciju u okviru jednog testnog slučaja.

Vjerovatnoća mutacije kod genetičkog algoritma je postavljena na vrijednost 0.99 za sve testne slučajeve. Osim toga, vjerovatnoća ukrštanja ima istu vrijednost 0.99 dok je broj najboljih jedinki koji se iz trenutne generacije prenose u novu generaciju (elitizam) jednak 2.

Različite verzije algoritma optimizacije kolonijom mrava su koristile vrijednosti parametra α u iznosu 1. Osim toga, vrijednost parametra β je 2, dok je za vrijednost parametra ρ korišteno 0.2.

Sumarni rezultati su prikazani u Tabeli 1.

TABELA I. SUMARNI REZULTATI TESTIRANJA GA, AS, EAS I MMAS NAD GRAFOVIMA SA RAZLIČITIM BROJEM ČVOROVA (GRADOVA) TE SA PROMJENLJIVIM BROJEM ITEERACIJA I BROJEM JEDINKI/MRAVA

Grafovi		**Parametri algoritma**		**GA**			**AS**			**EAS**			**MMAS**		
Broj gradova	***Najbolja udaljenost***	***Broj mrava/ jedinki***	***Broj iteracija***	***Nađena najkraća udaljenost***	***Stepen greške %***	***Vrijeme izvršenja (s)***	***Nađena najkraća udaljenost***	***Stepen greške %***	***Vrijeme izvršenja (s)***	***Nađena najkraća udaljenost***	***Stepen greške %***	***Vrijeme izvršenja (s)***	***Nađena najkraća udaljenost***	***Stepen greške %***	***Vrijeme izvršenja (s)***
7	19	5	100	19	0.0	0.074	19	0.0	0.047	21	10.5	0.046	19	0.0	0.122
		5	1000	19	0.0	0.796	19	0.0	0.489	19	0.0	0.484	19	0.0	1.197
		5	2000	19	0.0	1.463	19	0.0	0.947	19	0.0	0.971	19	0.0	2.689
		7	100	19	0.0	0.113	19	0.0	0.065	21	10.5 26	0.064	19	0.0	0.116
		7	1000	19	0.0	1.138	19	0.0	0.641	19	0.0	0.673	19	0.0	1.652
		7	2000	19	0.0	2.242	19	0.0	1.319	19	0.0	1.322	19	0.0	3.248
13	7293	5	100	8111	11.2	0.203	7310	0.23	0.156	7310	0.23	0.156	7534	3.30	0.239
		5	1000	7534	3.30	1.969	7586	4.01	1.469	7310	0.23	1.552	7312	0.26	2.692
		5	3000	7293	0.0	5.868	7586	4.01	4.799	7618	4.45	4.788	7310	0.23	8.366
		13	100	8111	11.2	0.592	7586	4.01	0.365	7618	4.45	0.396	7534	3.30	0.562
		13	1000	7534	3.3	5.593	7670	5.16	3.767	7310	0.23	4.34	7310	0.23	5.849
		13	3000	7534	3.3	16.732	7586	4.01	12.185	7543	3.42	12.109	7310	0.23	17.836

17	2085	5	100	2187	4.89	0.349	2149	3.07	0.238	2149	3.07	0.251	2149	3.07	0.386
		5	1000	2088	0.14	3.534	2149	3.07	2.450	2149	3.07	2.412	2149	3.07	4.071
		5	3000	2085	0.0	10.315	2149	3.07	7.822	2149	3.07	7.844	2149	3.07	13.537
		17	100	2178	4.46	1.472	2178	4.46	0.816	2178	4.46	0.794	2149	3.07	1.256
		17	1000	2085	0.0	15.721	2149	3.07	8.043	2149	3.07	8.016	2149	3.07	12.748
		17	3000	2085	0.0	54.278	2178	4.46	26.075	2178	4.46	26.242	2149	3.07	45.447
26	937	5	100	982	4.80	1.353	955	1.92	0.565	955	1.92	0.543	937	0.0	1.001
		5	1000	955	1.92	10.234	955	1.92	5.457	941	0.42	5.484	937	0.0	10.604
		5	3000	955	1.92	34.359	948	1.17	17.627	941	0.42	17.633	937	0.0	32.608
		26	100	1100	17.3	6.069	962	2.66	2.759	955	1.92	2.801	961	2.56	5.323
		26	1000	955	1.92	69.217	955	1.92	27.941	955	1.92	27.722	937	0.0	57.536
		26	3000	955	1.92	225.01	955	1.92	89.135	955	1.92	89.867	937	0.0	124.98
29	2020	5	100	2207	9.25	1.509	2085	3.21	0.689	2085	3.21	0.711	2085	3.21	1.2295
		5	1000	2096	3.76	14.375	2090	3.46	6.867	2085	3.21	7.031	2085	3.21	17.018
		5	3000	2096	3.76	45.315	2094	3.66	21.898	2134	5.64	22.572	2038	0.89	44.869
		29	100	2179	7.87	11.041	2134	5.64	3.857	2107	4.30	3.896	2060	1.98	9.0030
		29	1000	2111	4.50	103.03	2134	5.64	38.968	2107	4.30	38.816	2038	0.89	70.422
		29	3000	2096	3.76	329.51	2111	4.50	123.77	2085	3.21	125.23	2038	0.89	184.40

Ukoliko pogledamo rezultate prikazane u Tabeli 1. možemo vidjeti da su svi implementirani algoritmi u većini slučajeva uspjeli naći tačno rješenje za graf sa 7 čvorova. Izuzetak su samo dvije situacije kod elitističkog mravljeg sistema kada se dobije rezultat 21 umjesto 19. Uzrok ovome jeste brža konvergencija samog algoritma. U ovom slučaju je jako mali stepen greške te ga možemo zanemariti. Dakle, možemo zaključiti da se svi implementirani algoritmi dobro ponašaju za prvi graf bez obzira na broj iteracija.

Rezultati dobiveni za drugi graf koji sadrži 13 čvorova se razlikuju zavisno od algoritma. Jasno je da je jedino genetički algoritam uspio pronaći tačno rješenje kada je broj iteracija bio 3000 a broj jedinki 5. Elitistički i MAX-MIN mravlji sistem su dali 7310 kao najbolje rješenje. Ovo rješenje je jako blizu stvarnom rješenju i od njega se razlikuje za samo oko 0.2%. Bitno je napomenuti da je genetički algoritam u određenim situacijama za ovaj graf davao rješenja sa stepenom greške od oko 11% dok su sve verzije algoritma optimizacije kolonijom mrava imale najveći stepen greške oko 4-5%.

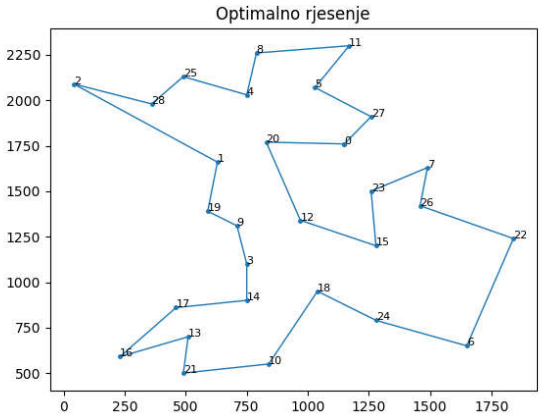
Treći graf se sastoji od 17 čvorova i optimalna staza ima dužinu 2085. Možemo primijetiti da je ponovo genetički algoritma jedini koji je dao optimalno rješenje i to u nekoliko različitih testnih slučajeva. Sa povećanjem broja iteracija i broja jedinki preciznost ovog algoritma se također povećavala. Verzije algoritma optimizacije kolonijom mrava su za ovaj graf u većini slučajeva davale rezultat u iznosu 2149. Stepen greške ovog rješenja je oko 3%. Treba istaći da je MAX-MIN mravlji sistem u svim testnim slučajevima dao upravo ovaj rezultat dok su mravlji sistem i elitistički mravlji sistem u više od 50% slučajeva davali taj rezultat.

Testiranje algoritama na četvrtom grafu sa 26 čvorova je pokazalo je da MAX-MIN mravlji sistem dao najbolje rezultate. Tačnije, samo u jednom testnom slučaju nije pronašao optimalno rješenje. Pri tome, stepen greške kod genetičkog algoritma ide čak do 17% u situaciji kada se algoritam pokretao sa 100 iteracija i 26 jedinki. Mravlji sistem i elitistički mravlji sistem su se uspijevali približiti rješenju, ali je stepen greške u većini slučajeva oko 2%.

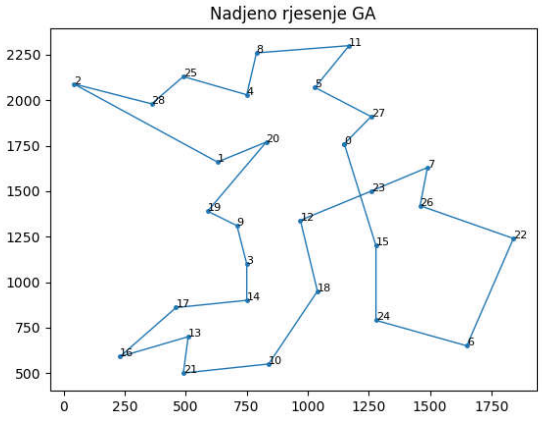
Konačno, testiranje algoritama na petom grafu sa 29 čvorova je pokazalo da su u ovoj situaciji verzije algoritma optimizacije kolonijom mrava superiornije u odnosu na genetički algoritam. Naime, najveći stepen greške ovih algoritama ne prelazi 6% dok genetički algoritam u pojedinim situacijama ima stepen greške od skoro 10%. Ipak, bitno je napomenuti da niti jedan algoritam nije uspješno pronašao optimum, odnosno svaki algoritam ima određeni ne-nulti stepen greške. Najmanji stepen greške je imao MMAS, te je samim time i došao do vrijednosti koja je najbliža optimumu. U situacijama kada je MMAS radio sa 3000 iteracija i 5 jedinki, 1000 iteracija i 29 jedinki ili sa 3000 iteracija i 29 jedinki, rezultat je iznosio 2038 što je relativno blizu optimumu koji iznosi 2020.

Optimalno rješenje za graf sa 29 čvorova je prikazano na Slici 1. Najbolja rješenja koja su pronašli genetički algoritam i MAX-MIN mravlji sistem su prikazana na Slici 2 i Slici 3, respektivno. Može se primijetiti da putanje u svim situacijama imaju zajedničke dijelove.

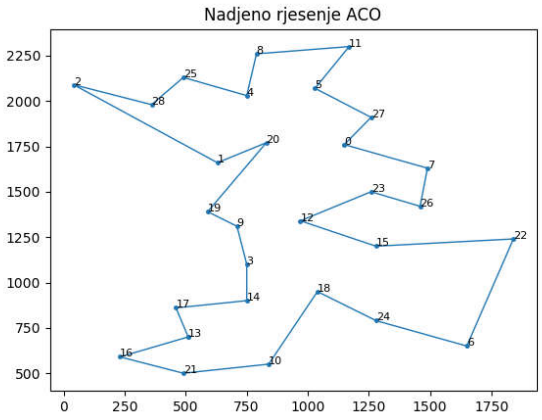
Razlike se javljaju kada genetički algoritam nakon čvora 12 izabere da posjeti čvor 23, odnosno kada MMAS algoritam izabere da posjeti čvor 25 nakon čvora 22 što utiče na dužinu putanje koja se mijenja u odnosu na optimalno rješenje te rezultira ukupnom putanjom dužine 2096 u slučaju GA, odnosno 2038 u slučaju MMAS umjesto optimalne putanje koja iznosi 2020.



Slika 1. Optimalno rješenje za graf sa 29 čvorova



Slika 2. Rješenje koje je genetički algoritam pronašao za graf sa 29 čvorova



Slika 3. Rješenje koje je MAX-MIN mravlji sistem pronašao za graf sa 29 čvorova

Osim stepena greške, potrebno je uporediti i vrijeme izvršavanja svakog od algoritama. U Tabeli 1. se jasno može vidjeti da se vrijeme izvršavanja kod svih algoritama povećava sa povećanjem broja iteracija i sa povećanjem broja jedinki/mrava. Također, bitno je uočiti da najkraće vrijeme izvršavanja imaju mravlji sistem i elitistički mravlji sistem. Slijede ih MAX-MIN mravlji sistem i genetički algoritam. Genetički algoritam je najsporiji dok je mravlji sistem najbrži algoritam. Ovo je očekivano s obzirom da je broj operacija koje je potrebno izvršiti unutar svake iteracije genetičkog algoritma manji od broja operacija potrebnih kod mravljeg sistema. Elitistički mravlji sistem i MAX-MIN mravlji sistem su sporiji od mravljeg sistema ali brži od genetičkog algoritma s obzirom na to da ipak imaju nešto više operacija od klasičnog mravljeg sistema ali istovremeno i nedovoljan broj operacija da bi sustigli genetički algoritam.

Iz dobivenih rezultata možemo zaključiti da je genetički algoritam bolji od svih verzija algoritma optimizacije kolonijom mrava za grafove sa manjim brojem čvorova dok se situacija mijenja kod onih grafova koji imaju veći broj čvorova. Također, od svih verzija algoritama optimizacije kolonijom mrava najbolje rezultate daje MAX-MIN mravlji sistem. Ovo je očekivano, jer je ova verzija algoritma upravo osmišljena da bi se izbjegli problemi prilikom implementacije standardne verzije algoritma.

Odluka o tome koji od ovih algoritama treba koristiti u određenoj situaciji zavisi od toga šta želimo postići. Ukoliko nam je bitna preciznost, onda možemo koristiti genetički algoritam koji sa većim brojem iteracija daje preciznije rezultate. Međutim, vrijeme izvršavanja ovog algoritma ponekad može predstavljati ograničenje. S obzirom da se u većini praktičnih primjena traži rješenje koje je dovoljno blisko optimumu, verzije algoritama optimizacije kolonijom mrava daju sasvim zadovoljavajuće rezultate. Oni su u mogućnosti da sa manjim brojem iteracija u odnosu na genetički algoritam daju rezultate koji su dovoljno bliski optimumu te je stepen greške u većini situacija oko 2-5%.

IV. ZAKLJUČAK

U ovom radu izvršeno je poređenje performansi jednih od najčešće korištenih algoritama optimizacije ACO i GA za rješavanje problema trgovačkog putnika. Performanse koje su mjerene jesu vrijeme izvršenja i najkraća pronađena udaljenost za konkretan problem.

Algoritmi su implementirani korištenjem Python programskog jezika, te su mjerenja izvršena pokretanjem pet testnih primjera uz korištenje različitih parametara.

Genetički algoritam (GA) je mnogo jednostavniji za implementaciju, zahtijeva manju količinu resursa, ali je sa druge strane sporiji što se tiče vremena izvršenja u odnosu na ACO. Algoritam optimizacije kolonijom mrava (ACO) u većini slučajeva daje bolje rezultate pogotovo za veće probleme, odnosno probleme sa većim brojem gradova.

GA je bolje koristiti za male ili probleme srednje veličine, dok se kao bolji za probleme sa izrazito velikim brojem gradova pokazao ACO. ACO je pogodan za rješavanje kompleksijih problema, ali je i teži za implementaciju od GA, te zahtijeva veću količinu resursa.

Oba algoritma imaju svoje prednosti i nedostatke, te zavisno od problema na koji se primjenjuju treba odrediti koji je od ova dva algoritma pogodniji. Međutim, oba algoritma imaju veliki potencijal za rješavanje stvarnih problema koji su bazirani na TSP-u.

GA kao i ACO su jako osjetljivi na promjenu parametara. U ovom radu su uzeti parametri za koje se algoritmi najbolje ponašaju u gotovo svim situacijama, te su se jedino mijenjali parametri koji se odnose na broj itreacija i broj jedinki, odnosno mrava. Na taj način dvije najbolje verzije oba algoritma su ravnopravno testirane promjenom zajedničkih parametara.

U budućem radu se može više fokusirati na parametre samih algoritama, a koji nisu broj iteracija i broj jedinki (mrava), te u odnosu na njihovu promjenu mjeriti peformanse samih algoritama.

LITERATURA

[1] S. Haroun, B. Jamal i E. Hicham, "A Performance Comparison of GA and ACO Applied to TSP", International Journal of Computer Applications, vol. 117, no. 20, pp. 28-35, 2015. Dostupno: 10.5120/20674-3466.

[2] Z. Zukhri i I. Paputungan, "A Hybrid Optimization Algorithm based on Genetic Algorithm and Ant Colony Optimization", International Journal of Artificial Intelligence & Applications, vol. 4, no. 5, pp. 63-75, 2013. Dostupno: 10.5121/ijaia.2013.4505.

[3] Alhanjouri, M. i B. Alfarra, "Ant Colony versus Genetic Algorithm based on Travelling Salesman Problem", International Journal of Computer Technology and Applications, 2011.

[4] N. Chandekar i M. Jayachandran Pillai, "A Comparative Study of GA and ACO for Solving Travelling Salesman Problem", International Journal of Mechanical And Production Engineering, ISSN: 2320-2092, 2017.

[5] S. Afreen, G. Rozario i A. Islam, "Travelling Salesman Problem: Complexity Analysis of ACO with the Existing Algorithms", American Journal of Engineering Research (AJER) e-ISSN: 2320-0847 p-ISSN : 2320-0936, vol. -8, no. -7, pp. pp-148-153, 2019.

[6] Math.uwaterloo.ca, 2020. [Online]. Dostupno: http://www.math.uwaterloo.ca/tsp/apps/genome.html. [Pristupano: 19- Jan- 2020].

[7] J. Holland, "Adaptation in Natural and Artificial Systems. London", England: MIT Press, 1992. [ebook]. Dostupno: https://books.google.ba/books?hl=en&lr=&id=5EgGaBkwvWcC&oi=fnd&pg=PR7&dq=Adaptation+in+Natural+and+Artificial+Systems&ots=mImr4ZOsqn&sig=pIFCCT0ihwZaz_uAvTrMmnNAuMw&redir_esc=y#v=onepage&q=Adaptation%20in%20Natural%20and%20Artificial%20Systems&f=false. [Pristupano: 19- Jan- 2020].

[8] Cs.cmu.edu, 2020. [Online]. Dostupno: http://www.cs.cmu.edu/~02317/slides/lec_8.pdf. [Pristupano: 19- Jan- 2020].

[9] H. Mukhairez i A. Maghari, "Performance Comparison of Simulated Annealing, GA and ACO Applied to TSP", International Journal of Intelligent Computing Research, vol. 6, no. 4, pp. 647-654, 2015. Available: 10.20533/ijicr.2042.4655.2015.0080.

[10] Samim Konjicija, Predavanja na predmetu: Optimizacija resursa, Elektrotehnički fakultet, Univerzitet u Sarajevu, ak. 2019/2020. godina.

[11] "Travelling Salesman Problem | Set 1 (Naive and Dynamic Programming) - GeeksforGeeks", GeeksforGeeks, 2020. [Online]. Dostupno: https://www.geeksforgeeks.org/travelling-salesman-problem-set-1/. [Pristupano: 19- Jan- 2020].

[12] H. Miller i S. Shaw, "Geographic Information Systems for Transportation: Principles and Applications", Oxford University Press, 2001. [ebook]. Dostupno: https://books.google.ba/books?hl=en&lr=&id=2XjtJIUG-gMC&oi=fnd&pg=PA3&dq=Geographic+Information+Systems+for+Transportation:+Principles+and+Applications&ots=JHBqKmx3Vs&sig=2yhCZl8g6R-E7kvmTSpVLflB6e8&redir_esc=y#v=onepage&q=Geographic%20Information%20Systems%20for%20Transportation%3A%20Principles%20and%20Applications&f=false. [Pristupano: 19- Jan- 2020].

[13] Haris Šupić, Predavanja na predmetu: Napredni algoritmi i strukture podataka, Elektrotehnički fakultet, Univerzitet u Sarajevu, ak. 2019/2020. godina.

[14] E. Klarreich et al., "Computer Scientists Find New Shortcuts for Infamous Traveling Salesman Problem", WIRED, 2020. [Online]. Dostupno: https://www.wired.com/2013/01/traveling-salesman-problem/. [Pristupano: 19- Jan- 2020].