Problemi usmjeravanja vozila

Šiser, Ivan

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Ivan Šiser

PROBLEMI USMJERAVANJA VOZILA

DIPLOMSKI RAD

Varaždin, 2021.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Ivan Šiser

Identifikacijski broj (x-ica): 0016123395

Studij: *Informacijsko i programsko inženjerstvo*

PROBLEMI USMJERAVANJA VOZILA

DIPLOMSKI RAD

Mentor:

Doc. dr. sc. Nikola Ivković

Varaždin, rujan 2021.

Ivan Šiser

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

Problem usmjeravanja vozila (eng. *Vehicle Routing Problem*, skraćeno VRP) je jedan on najpopularnijih problema u kombinatornoj optimizaciji, a njegovo proučavanje utjecalo je na istraživanje nekoliko egzaktnih i heurističkih rješenja opće primjenjivosti. Zbog svoje složenosti, pripada skupini NP-teških problema. VRP ima nekoliko različitih varijanti s obzirom na vrstu i broj ograničenja, a odabir varijante ovisi o instanci problema koja se pokušava riješiti. U današnjem svijetu u kojemu transport i logistika igraju veliku ulogu, VRP postaje sve veći predmet istraživanja s ciljem minimiziranje cijene. U ovom radu najprije je obrađena tema VRP-a i njegovih varijanti, s teorijskog i matematičkog stajališta. Nakon toga su navedeni pristupi njegovom rješavanju. Jedan od takvih pristupa je i metaheuristička optimizacija kolonijom mrava (eng. *Ant Colony Optimization*, skraćeno ACO), u kojoj su glavna komponenta umjetni mravi koji postupno grade optimalno rješenje. ACO također ima različite varijante, koje je potrebno odabrati i prilagoditi ovisno o problemu koji se pokušava riješiti. Tako je obrađena i varijanta MAX-MIN mravljeg sustava (eng. *MAX-MIN Ant System*, skraćeno MMAS), koja se pokazala da daje najbolja rješenja za veliki skup vrhova i dovoljan broj iteracija. U ovom radu, implementirano je rješenje za problem usmjeravanja vozila s ograničenjem kapaciteta (eng. *Capacitated VRP*, skraćeno CVRP) pomoću MMAS varijante mravljeg algoritma. Implementacija je podvrgnuta testiranjem nad različitim instancama problema dostupnih na internetu i podešenim parametrima. Na temelju izlaznih podataka testiranja, provedena je statistička analiza prema kojoj je donesen zaključak o efikasnosti algoritma i mogućim poboljšanjima.

Ključne riječi: Problem usmjeravanja vozila, VRP, Problem usmjeravanja vozila s ograničenjem kapaciteta, CVRP, metaheuristika Optimizacija kolonije mrava, ACO, MAX-MIN mravlji sustav, MMAS

Sadržaj

1. Uvod

Prema podacima iz Europske unije, transportne usluge čine nešto više od 9% ukupnih troškova cijele Europe. Gledajući s industrijske strane, na cijenu ukupnog troška proizvodnje jednog klasičnog proizvoda utječe cijena prijevoza od minimalno 10%. [1]

Napretkom tehnologije, računalo je postalo neizostavni dio svakog sektora u poslovnome svijetu, ponajprije iz razloga što omogućava smanjenje vremena i troška pojedinih procesa. Prema istraživanjima provedenim na simulacijama rada poduzeća, korištenje računalnih optimizacijskih programa omogućuje uštedu od otprilike 5% ukupnih troškova poslovanja. [2]

U ovome radu obrađena je tema problema usmjeravanja vozila, koja upravo ima važnu ulogu u transportnim uslugama poput logistike i distribucije dobara. Rješavanje ovog problema se temelji na optimizaciji i kombinatorici, pomoću koje se na efikasan način pokušava dobiti dovoljno dobro rješenje koje bi u stvarnom svijetu smanjilo vrijeme i trošak transporta. Sam problem se najčešće uspoređuje s poznatim problemom trgovačkog putnika, ali uz dodatne parametre i ograničenja koje ga čine puno težim. Zbog različitih mogućih ograničenja, problem usmjeravanja vozila nije jedinstven, nego sadrži različite varijante.

Nakon teorijske i matematičke obrade samog problema i njegovih varijanti, u nastavku se usredotočujemo na njihovo rješavanje pomoću optimizacije mravljom kolonijom. Kako ne bi zanemarili i ostale načine rješavanja, u jednom poglavlju opisani su i drugi pristupi koji mogu dati dobro rješenje.

U sklopu ovoga rada, implementirano je rješenje za problem usmjeravanja vozila s ograničenjem kapaciteta pomoću MAX-MIN mravljeg sustava. Rješenje je testirano nad različitim instancama problema, kao i na različitim ulaznim parametrima. Izlazni podaci rješenja su statistički obrađeni, a na temelju dobivenih rezultata donesen je zaključak istraživanja i moguća poboljšanja algoritma.

2. Problem usmjeravanja vozila

Upravljanjem distribucijom i logistikom smatra se jedan od teži zadataka u današnjem svijetu. U samoj srži rješavanja zadataka nalazi se problem usmjeravanja vozila. Svakodnevno brojna poduzeća i organizacije sudjeluju u procesu dostave, razmjeni dobara i usluge te ljudi. Kako svako poduzeće posluje na svoj način, odnosno možemo reći da ima vlastite ciljeve, ograničenja i da različite varijable utječu na krajnji ishod. Upravo iz toga razloga što praksa pokazuje veliki broj različitih problema, gotovo je nemoguće pronaći jedno univerzalno teoretsko rješenje koje bi pokrilo sve slučajeve.

Najveći pomak u istraživanju ovog problema napravljen je u 1959. godine, kada su dva američka matematička znanstvenika George Dantzig i John Ramser u svojoj publikaciji objavili članak na temu „*truck dispatching*“. Unutar članka izneseno je nekoliko varijanti općenitog problema, za koje su predložene snažne formulacije, zajedno s raznim istraživanjima i algoritmima dekompozicije. Razvijena je i brojna heuristika za probleme s usmjeravanjem vozila, čije je istraživanje potaknulo pojavu i rast nekoliko metaheuristika, kojima se performanse pokušavaju konstantno unaprijediti. [3]

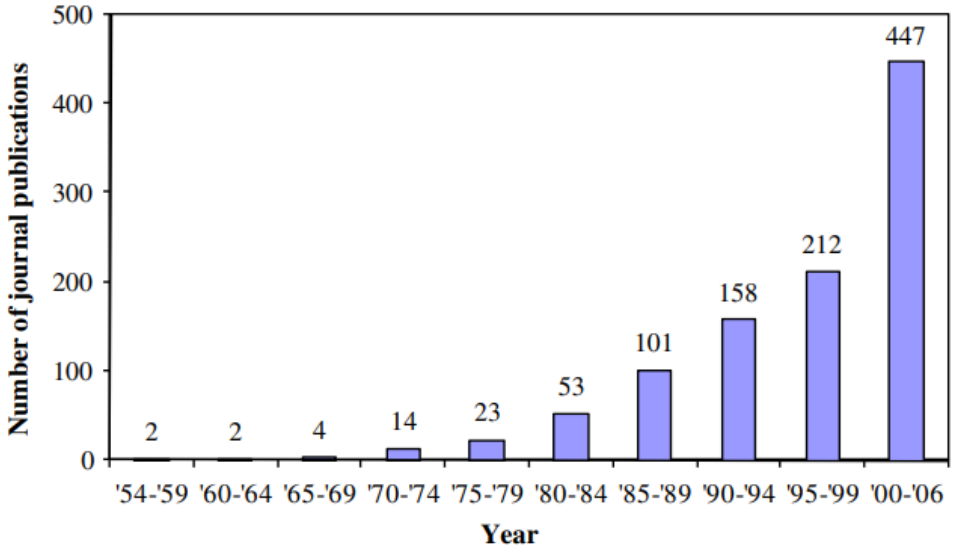
Ovo poglavlje je usmjereno na općenitu razradu problema usmjeravanja vozila i definiranje njezinog matematičkog modela, kao i na podtipove problema sukladno njezinim najpoznatijim varijantama. Osim toga, u nastavku slijedi kratak uvid u povijest prvih istraživanja i potrebe za sve većim proučavanjem, što je dovelo do razvoja metaheuristike i brojnih algoritama pomoću kojih je moguće doći do rješenja.

2.1. Kratka povijest istraživanja

Kao što je već prethodno spomenuto, istraživanje problema usmjeravanja vozila započinje objavom članka na temu „*Truck Dispatching Problem*“ 1959. godine. Članak se bavio modeliranje problema kako bi flota homogenih kamiona mogla zadovoljiti potražnju za naftom prema različitim benzinskim postajama iz središnjeg čvora (mjesta prerade naftnih sirovina u gorivo) s minimalnom prijeđenom udaljenosti. Pet godina kasnije (1964.), Clarke i Wright generaliziraju ovaj problem kao problem linearne optimizacije s kojom se obično susreće u domeni logistike i transporta. Kao naprimjer: kako poslužiti određeni skup klijenata, geografski raspršenih okolo središnjeg skladišta, koristeći vozni park kamiona različitog kapaciteta. Od ovoga trenutka, ovaj problem postaje poznat kao problem usmjeravanja vozila (eng. *Vehicle Routing Problem*, skraćeno VRP), jedna od najšire proučavanih tema u području operacijskih istraživanja. [3]

Trenutni VRP modeli uvelike se razlikuju od prethodno spomenutih, ponajprije iz razloga što nastoje uključiti složene varijable iz stvarnog života, poput vremena putovanja ovisnih o gužvi (utjecaj prometne gužve), vremenske intervale za preuzimanje i dostavu, kao i dinamički mijenjanje podataka tokom vremena (npr. podaci o potražnji). Iako su Clarke i Wright napravili prekretnicu primjenom efektivnog pohlepnog algoritma uštede, ove značajke donose znatnu veću složenost. Tako se na VRP gleda kao NP – težak problem (nedeterministički u polinomijalnom vremenu teški), za koje dotadašnji egzaktni algoritmi nisu primjenjivi jer njihova efikasnost opada s brojem instanci. Za ovakve probleme često su prikladniji heuristički i metaheuristički algoritmi. [3]

U proteklom desetljeću, broj metoda za rješavanje različitih varijanti VRP-a uvedenih u akademsku literaturu, brzo je narasla. Napredak računalne tehnologije, odnosno povećanje brzine obrade i kapaciteta sadašnjih računala, omogućilo je rješavanje većeg broja instanci VRP-a, što rezultira povećanje istraživanja i razvoja softvera za VRP u komercijalne svrhe. Danas VRP softvere koriste tisuće tvrtki, među kojima je i *Coca-Cola Enterprises*. [4]



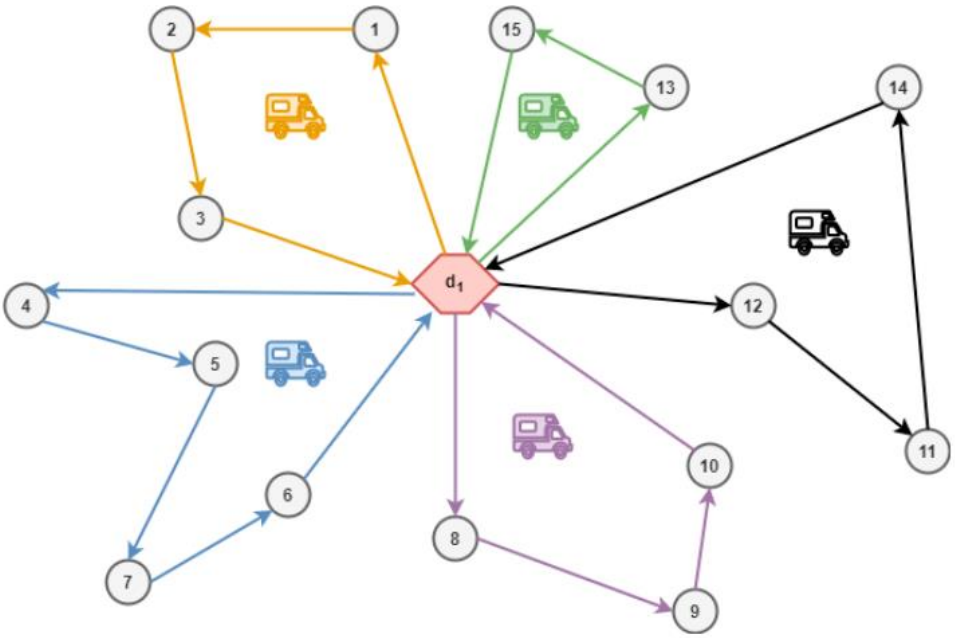
Slika 1. Prikaz porasta članaka na temu VRP (1954. - 2006.) [4]

2.2. Klasični problem usmjeravanja vozila

Problem usmjeravanja vozila (eng. *Vehicle Routing Problem*, skraćeno VRP) je jedan on najpopularnijih problema u kombinatornoj optimizaciji, a njegovo proučavanje utjecalo je na istraživanje nekoliko egzaktnih i heurističkih rješenja opće primjenjivosti. Korijen problema je jako sličan problemu trgovačkog putnika (eng. *Traveling Salesman Problem*, skraćeno TSP), s dodatnim složenim ograničenjima što ga itekako otežava. Iz toga razloga, možemo reći da VRP generalizira TSP. VRP pokušava dati odgovor na pitanje: „Koji je optimalan skup puteva za grupu vozila u cilju isporuke robe određenom broju klijenata?“. Pronalaženje optimalnog odgovara na navedeno pitanje je NP – težak problem, odnosno vrlo je teško doći do optimalnog rješenja. [5]

Pod klasičnim problemom usmjeravanja vozila smatramo općeniti naziv za cijeli skup problema kojima je cilj pronaći najbolje rješenje s određenom skupinom vozila koji kreću iz definiranog skladišta obići sve klijente uz uvjet da svakog klijenta posjeti samo jedno vozilo i da se na kraju sva vozila vrate u definirano skladište. Najbolje rješenje najčešće ovisi o stvarnom primjeru koji se pokušava riješiti, a neki od najčešćih su [5]:

- Minimizirati globalne troškove prijevoza na temelju prijeđene udaljenosti
- Minimizirati broj vozila potrebno za usluživanje svih klijenata
- Smanjiti varijaciju u vremenu putovanja i opterećenju vozila
- Smanjiti penale za lošu kvalitetu usluge
- Maksimizirati profit



Slika 2. Klasični problem usmjeravanja vozila (VRP) [6]

2.3. Matematički model

Klasični VRP možemo prikazati na usmjerenom težinskom grafu $G = (V, E)$, gdje je $V = \{0, \ldots, n\}$ skup vrhova. Svaki vrh $i \in V \setminus \{0\}$ predstavlja kupca koji ima potražnju q_i (npr. količinu dobara) koja predstavlja pozitivan broj, dok vrh 0 predstavlja centralno središte odnosno skladište. Za svaki brid $e \in E = \{(i,j) : i,j \in V,\ i < j\}$ je povezan s putnim troškovima $c_e\ ili\ c_{ij}$ koji mogu predstavljati cijenu rute, udaljenost između dva grada, vrijeme razmak i slično. Sve se cijene mogu izraziti pomoću simetrične matrice za koju vrijedi $c_{ij} = c_{ji}$, dok se na dijagonali nalaze vrijednosti $c_{ii} = c_{jj} = 0$. Skupina od m identičnih vozila, svako vozilo s vlastitim kapacitetom Q, u početku se nalazi u središtu odnosno vrhu $i = 0$. VRP zahtijeva određivanje skupa od m ruta gdje su ukupni troškovi putovanja svedeni na minimum i da za svaki vrijedi slijedeće [7]:

1. Svaki klijent je posjećen točno jednom rutom
2. Svaka ruta započinje i završava u centralnom središtu (skladištu)
3. Ukupna potražnja usluženih kupaca unutar rute ne prelazi kapacitet vozila Q
4. Ukupna duljina rute svakog vozila ne prelazi unaprijed postavljeno ograničenje L (opcionalno – npr. slučaj u kojemu bridovi predstavljaju vrijeme kao cijenu rute, ukupna ruta ne smije prelaziti više od 8 sati odnosno radno vrijeme vozača)

Rješenje se može promatrati kao skup m ciklusa koji dijele zajednički vrh skladišta.

Matematički model:

T_i – redoslijed isporuka vozila i

$T_{i,j}$ – vrijednost polja isporuka vozila i na j indeksu

TQ_i – ukupna duljina rute vozila i

$e_{ij} \in \{0,1\}$ – 0 ne postoji brid, 1 brid postoji ($i \neq j;\ i,j \in \{0,1,2,\ldots,n\}$)

Cilj:

$$Min = \sum_{i=1}^{m} \left(c_{\,0\,,\,T_{i,0}} + \sum_{j=1}^{n-1} c_{\,T_{\,i\,,j}\,,T_{\,i,\ \,j+1}} + c_{\,|T_i|-1\ ,\ \ 0} \right)$$

Navedena ograničenja poštujući redoslijed:

(1)

$$\sum_{i \in V} (\, j \in T_i \,) \;=\; 1 \;\; (\, j \in N \backslash \{0\} \,)$$

(2)

$$\sum_{j=1}^{n} e_{0j} \;=\; 1 \;\; (T_{i,0} \;, \mathrm{i} \in \{0, 1 \ldots, \mathrm{m}\}) \;\cup\; \sum_{j=1}^{n} e_{j0} \;=\; 1 \;\; (T_{i,\; |T_i|-1} \;, \mathrm{i} \in \{0, 1 \ldots, \mathrm{m}\})$$

(3)

$$\sum_{j \in T_i} q_j \;\leq\; q_i \,(\, i \in V \,)$$

(4)

$$TQ_i \;\leq\; L \,(\, i \in V \,)$$

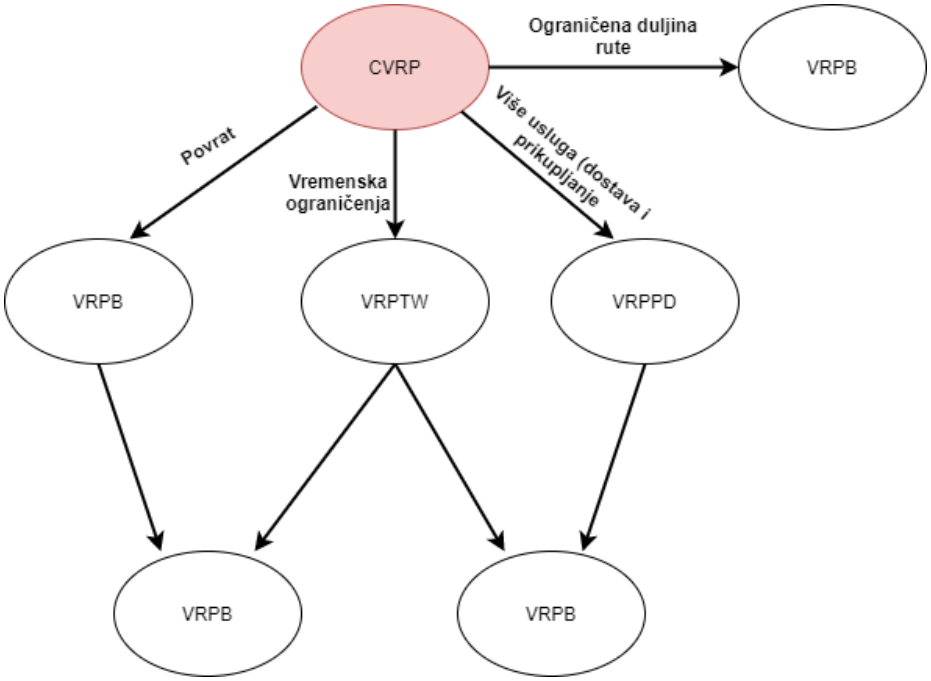
Traženo rješenje:

$$R \;=\; \{T_1, T_2, \ldots, T_m\}$$

$$R \;=\; \sum_{i=1}^{m} \left(e_{\,0\,,\,T_{i,0}} \;+\; \sum_{j=1}^{n-1} e_{\,T_{\,i,j}\,,\,T_{\,i,\;\,j+1}} + e_{\,|T_i|-1\;,\;\;0} \right)$$

2.4. Varijante problema

Prilikom modeliranja problema usmjeravanja vozila za konkretan primjer možemo se susresti s raznim ograničenjima i varijablama. Kako je danas sam transportni sektor kategoriziran prema određenoj usluzi koje rješavaju određene specijalizirane probleme, tako su nastale i različite poznate varijante VRP za njihovo rješavanje. Na sljedećoj slici prikazan je mapa poznatih tipova VRP te način na koji su pojedini tipovi povezani, odnosno kako jedan tip problema proširuje drugi:



Slika 3. Varijante problema usmjeravanja vozila

U nastavku je opisana pojedina varijanta problema, s detaljnijim opisom VRP s ograničenjem kapaciteta. Upravo iz razloga što ju većina ostalih primjera proširuje kao što je vidljivo sa slike, ali također jer je na temelju te varijante implementirano rješenje.

2.4.1. Kapacitativni VRP

Kapacitativni problem usmjeravanja vozila (eng. *Capacitated VRP*, skraćeno CVRP) utemeljen je na ograničenju kapaciteta, a smatra se ujedno i osnovnim tipom problema usmjeravanja vozila. Unutar problema definirani su klijenti koje treba poslužiti (zajedno sa svojim potraživanjima iskazanim u količinama robe) i vozila s istim karakteristikama (brzina, kapacitet i sl.). Svako vozilo u početku se nalazi u centralnom skladištu iz kojega kreću prema klijentima. Jedino ograničenje koje postoji je kapacitet pojedinog vozila. Cilj je pronaći

najbolje rješenje koji se očituje kao minimiziranje ukupnoga troška, odnosno pronaći rutu za svako pojedino vozilo na takav način da svi korisnici budu posluženi na najbolji mogući način. Ukupan trošak dobiva se kao suma ukupnog troška puta svakog vozila. [8]

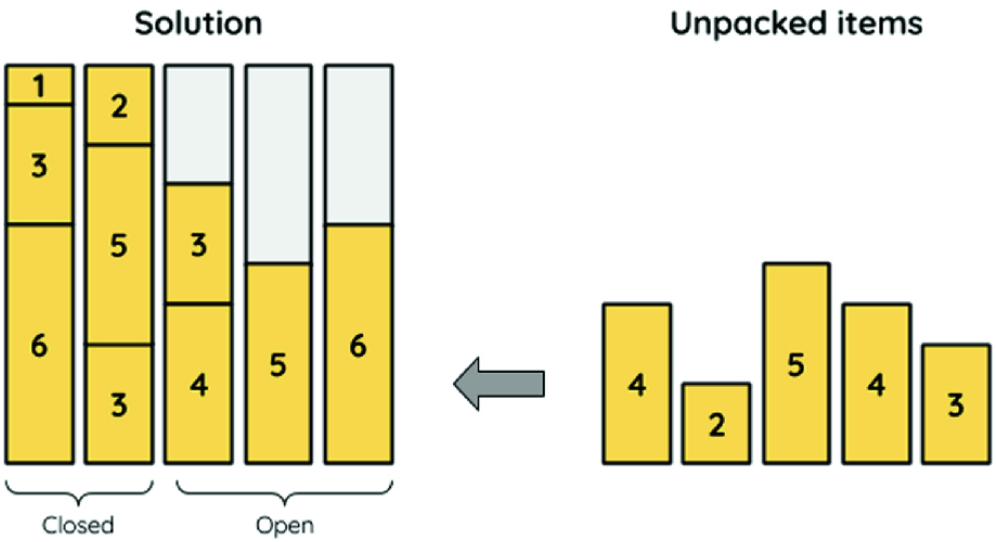
Kako se CVRP zapravo smatra osnovnim oblikom VRP-a, svi uvjeti ograničenja su identični klasičnom VRP-u pa iz toga razloga matematički model objašnjen u prethodnom poglavlju također vrijedi i za ovu inačicu. Da ponovimo, korištenjem teorije grafova već smo definirali graf $G = (V, E)$, gdje je $V = \{0, \ldots, n\}$ skup vrhova (0 predstavlja centralno skladište, ostatak skupa predstavlja korisinke) , a E skup lukova između navedenih vrhova. Za lukove $(i, j) \in E$ se zapravo veže nenegativna cijena c_{ij} koja predstavlja prijevoza između korisnika i i j. U praksi se ne koriste lukovi (i, i) i pretpostavlja se da cijena takvog luka $c_{ij} = +\infty$ za sve $i \in V$. Prilikom implementacije, troškovi prijevoza se zapisuju u matricu troškova c veličine n, a radi jednostavnosti u lukove (i, i) umjesto $+\infty$ upisuju se nule. Ako na G gledamo kao usmjereni graf, matrica troškova c je asimetrična matrica, pa se na ovakav CVRP naziva asimetrični VRP s ograničenjima kapaciteta (eng. *Asymetric CVRP*, skraćeno ACVRP). Ako matrica troškova izgleda na način da su $c_{ij} = c_{ji}$ za svaki $(i, j) \in E$, tada se problem naziva simetrični VRP s ograničenjima kapaciteta (eng. *Symetric CVRP*, skraćeno SCVRP). Općenito, kada kažemo CVRP onda se podrazumijeva da mislimo na SCVRP. [5] [8]

Unutar CVRP također se pronalazi problem minimalnog broja vozila odnosno ukupnog kapaciteta svih vozila potrebnih da se mogu ispuniti zahtjevi prema svim korisnicima i na taj način uspješno riješiti CVRP. Kako doći do toga podatka? Najprije, obrazložimo i ovaj problem s matematičke strane. Svaki od korisnika i gdje je $i \in \{1, .., n\}$ povezan je s unaprijed poznatim nenegativnim zahtjevom robe d_i, dok za početno centralno skladište se postavlja fiktivni zahtjev d_0. Uzmemo li skup S kao podskup $S \subseteq V$, ukupni zahtjev skupa odnosno zahtjev svi klijenata izgleda ovako:

$$d(S) = \sum_{i \in V} d_i$$

U skladištu se u početku nalazi skup od predefiniranih K identičnih vozila, svatko sa svojim kapacitetom C (koji može biti isto za svako vozilo ili različito). Da bi rješenje problema bilo izvedivo, mora vrijediti $d_i \leq C$ za svaki $i \in \{1, .., n\}$. Kako je već poznato, svako vozilo može izvesti najviše jednu rutu. Pretpostavlja se da K nije manji od K_{min}, gdje je K_{min} definiran kao minimalni broj vozila potrebnih da se posluže svi korisnici. I dalje se pitamo kako odrediti taj minimalni broj vozila K_{min}? Postoje različiti algoritmi za to, a jedna od preporuka je rješavanje problema pakiranja koša (eng. *Bin Packing Problem*, skraćeno BPP), koji ovdje

rješava problem određivanja minimalnog broj vozila kapaciteta C potrebnog za ispunjenje svih zahtjeva d_i. Kako se BPP smatra NP-teškim problemom, njegovo rješavanje je poprilično kompleksno pa se u tome slučaju za manji broj zahtijeva (otprilike do nekoliko stotina) preporučuje korištenje neke druge poznate metode koje efikasno pronalaze optimalno rješenje. [8] [9]



Slika 4. Prikaz rješavanja BPP-a [9]

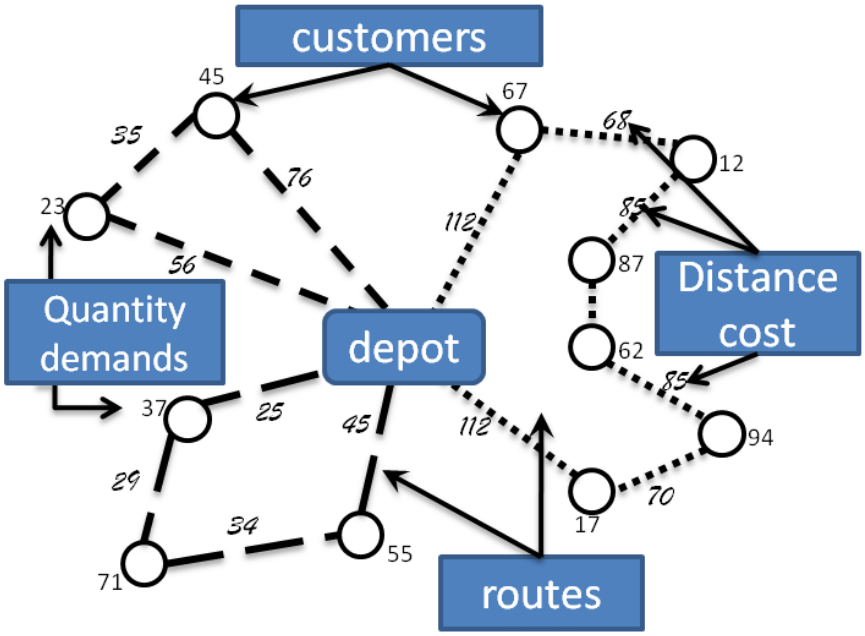
U literaturi postoji nekoliko varijacija ovog problema. Prva varijanta odnosi se na broj raspoloživih vozila K koji je veći od K_{min}, što bi značilo da postoji mogućnost da će pojedina vozila ostati neiskorištena. Smanjenje troška se može postići tako da se uz minimiziranje ukupne cijene ruta svih vozila, smanji i broj korištenih vozila, te se na taj način uštedi i na troškovima vozila. Kada ograničenje kapaciteta zamijenimo s ograničenjem maksimalnog puta ili vremenskog trajanja, VRP dobivamo varijantu koju zovemo problem usmjeravanja vozila ograničen udaljenošću (eng. *Distance Constrained VRP*, skraćeno DVRP). Takvu udaljenost (ili vrijeme) označavamo s t_{ij}, gdje su $(i,j) \in V$, a u pravilu ona ne smije biti veća od maksimalnog dozvoljenog prevaljenog puta na ruti odnosno vremena provedenog na ruti T. Ako je duljina lukova izražena kao vrijeme putovanja, tada tome vremenu dodajemo i vrijeme posluživanja ς_i koje je pridruženo za svakog korisnika i, a predstavlja vrijeme zadržavanja vozila kod korisnika odnosno vrijeme izražavanja usluge. Ako vrijeme kretanja na luku bez vremena zadržavanja kod korisnika označimo s t'_{ij}, tada ukupno vrijeme kretanja na luku između vrha $(i,j) \in V$ dobijemo prema formuli [8]:

$$t_{ij} = t'_{ij} + \frac{\varsigma_i + \varsigma_j}{2}$$

Do sada smo uvijek govorili o varijantama gdje je svako vozilo identično. Također postoji varijanta u kojoj sva vozila nisu jednaka, već imaju različite kapacitete C_k, za vozila $k \in \{1,\ldots,K\}$, gdje K predstavlja ukupan broj vozila. Tada su maksimalni dozvoljeni putevi odnosno vremena definirana kao T_k za svako k. [8]

Varijanta CVRP-a gdje su prisutna dva ograničenja: ograničenje kapaciteta i ograničenje maksimalne duljine rute zovemo problem usmjeravanja vozila ograničen udaljenošću i kapacitetom (eng. *Distance-Constrained CVRP*, skraćeno DCVRP) [5]

Također, u literaturi nije strogo navedeno kako se odvija posluživanje klijenta unutar CVRP modela, odnosno je li kamion odlazi iz skladišta do svakog klijenta dostavljajući mu robu ili skupljajući robu, ali u konačnici rješenje se svede na isto.



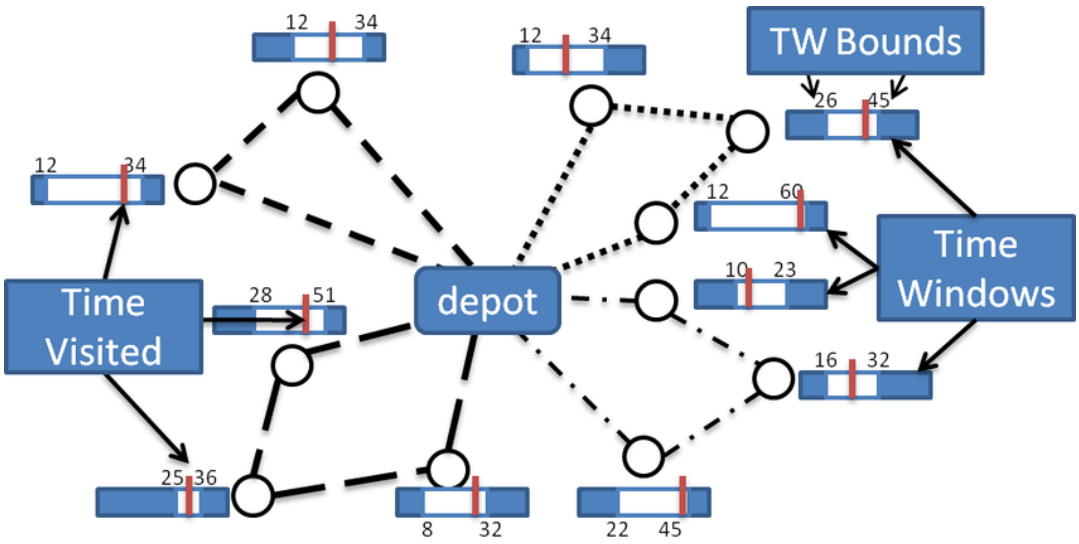
Slika 5. VRP s ograničenjem kapaciteta [10]

2.4.2. VRP s vremenskim ograničenjima

Problem usmjeravanja vozila s vremenskim ograničenjem (eng. *VRP with Time Windows*, skraćeno VRPTW) predstavlja proširenje CVRP gdje osim kapacitivnih ograničenja svakog vozila postoje i vremenska ograničenja vezana za svakog korisnika. Posluživanje svakog korisnika mora započeti u vremenskom razmaku $[a_i, b_i]$, a vozilo za vrijeme posluživanja t_i. U slučaju da vozilo dođe do korisnika prije vremena a_i, vozilu je u pravilu dozvoljeno da čeka do trenutka a_i, kada može započeti posluživanje. Osim tih

vremena, zadano je i vrijeme kada vozilo izlazi iz skladišta te vrijeme koje vozilo provede na svakom od lukova t_{ij} $(i,j) \in V$. [5]

Uobičajeno je da se matrica troškova i matrica putnih vremena podudaraju, a da su vremenski prozori korisnika tako definirani da se pretpostavlja kako sva vozila napuštaju skladište u vremenskom trenutku 0. Jedino proširenje ograničenja s obzirom na CVRP je uvjet posluživanja svakog korisnika u točno određenom vremenskom intervalu $[a_i, b_i]$ u kojemu je vozilo zaustavljeno s obzirom na definirano vrijeme t_i. [8]



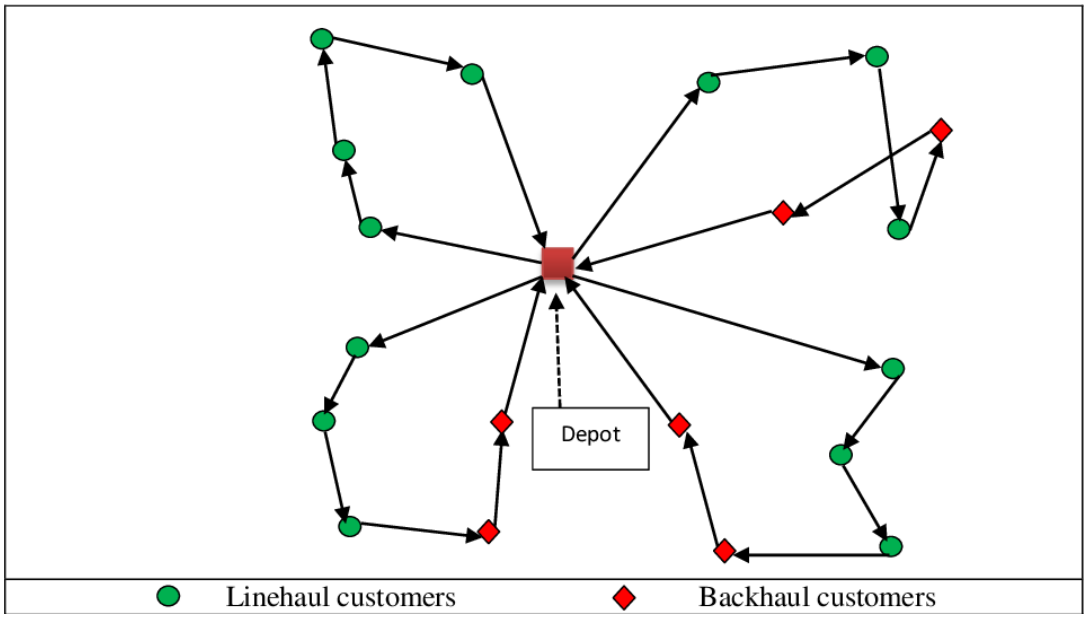
Slika 6. Prikaz VRP s vremenskim ograničenjem [10]

2.4.3. VRP s povratom

Problem usmjeravanja vozila s povratom (eng. *VRP with Backhauls*, skraćeno VRPB) je proširenje CVRP gdje je skup korisnika $V \setminus \{0\}$ podijeljen na dva podskupa. U prvom podskupu A nalazi se n korisnika koji zahtijevaju određene količine robe za isporuku. Drugi podskup B sadrži m korisnika kod koji kojih se zahtijeva prikupljanje određene količine iste robe. Korisnici su indeksirani na način da je $A = \{1, \ldots, n\}$ i $B = \{n+1, \ldots, n+m\}$. Prema navedenom, VRP s povratom nije baš najpreciznije ime pošto se ne radi samo o povratu robe, nego i posluživanju korisnika. [11]

Unutar VRPB pronalazi se dodatno ograničenje prednosti posluživanja, u kojem se svi korisnici skupa A moraju poslužiti prije korisnika B skupa. Nenegativni zahtjevi d_i svakog vrha imaju dvostruko značenje, ovisno radi li se o posluživanju ili prikupljanju. Skladištu se pridružuje fiktivni zahtjev d_0. S obzirom na CVRP, VRPB može definirati dva dodatna ograničenja: kapacitet odabranog vozila Q mora biti veći ili jednak zbroju pojedinačnih

zahtijeva klijenta iz oba podskupa A i B, kao i ograničenje da u svakoj ruti vozila svi korisnici iz podskupa A su posluženi prije korisnika iz podskupa B ako B nije prazan skup. Uobičajeno se smatra da podskup A nije prazan skup, a implicitno se pretpostavlja da svaka ruta sadrži klijente iz oba podskupa. [11]



Slika 7. Prikaz VRP s povratom [12]

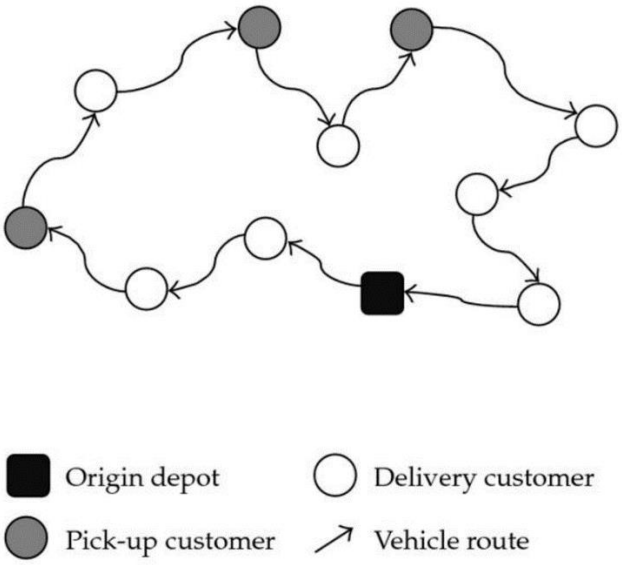
2.4.4. VRP s prikupljanjem i dostavom

Problem usmjeravanja vozila s prikupljanjem i dostavom (eng. *VRP with Pickup and Delivery*, skraćeno VRPPD) predstavlja problem u kojem je svaki korisnik i povezan s dvije vrijednosti: zahtjevom za dostavu d_i i zahtjevom za prikupljanje iste robe p_i. U većini slučajeva, koristi se samo jedna vrijednost $d_i = d_i - p_i$ za svakog korisnika i, koja predstavlja razliku zahtijeva dostave i prikupljanja. Kako zahtjevi korisnika za prikupljanje mogu biti veći od zahtijeva za dostavu, d_i također možete biti i negativne vrijednosti. Svaki zahtjev sadrži podatke o mjestu prikupljanja, mjestu dostave i količini robe koju treba transportirati. Zahtjevi su unaprijed definirani za svakog korisnika i, kao podaci o mjestu dostave O_i i mjestu prikupljanja D_i, u slučaju da O_i i D_i nisu početno skladište. [13]

Prilikom rješavanja VRPPD - a, pretpostavlja se da je vozilo izvršilo dostavu prije posjetu mjesta prikupljanja. Uz do sada standardna ograničenja CVRP - a, ovaj problem se proširuje s dva nova uvjeta [13]:

- Za svakog korisnika i postoji u istoj ruti korisnik O_i (mjesto dostave koje se razlikuje od skladišta) kojeg treba poslužiti prije korisnika i
- Za svakog korisnika i postoji u istoj ruti korisnik D_i (mjesto prikupljanja koje se razlikuje od skladišta) kojeg treba poslužiti poslije korisnika i

Najčešće se centralno skladište uzima kao izvor i odredište zahtijeva. Tada govorimo o varijanti VRP – a sa simultanom dostavom i prikupljanjem (eng. *VRP with Simultaneous Pickup and Delivery*, skraćeno VRPSPD). [8]



Slika 8. Primjer jedne rute vozila za VRPPD [13]

2.4.5. Izvedene varijante

U prethodnim poglavljima, objašnjenje su četiri glavne varijante problema usmjeravanja vozila, čije kombinacije najčešće tvore i ostale podtipove. Na slici 3. vidimo dvije najčešće kombinacije koje se susreću u većini literatura:

1. Problem usmjeravanja vozila s povratom i vremenskim ograničenjem (eng. *VRP with Backhauls and Time Windows*, skraćeno VRPBTW)
2. Problem usmjeravanja vozila s prikupljanjem i dostavom te vremenski ograničenjem (eng. *VRP with Pickup And Delivery and Time Windows*, skraćeno VRPPDTW)

Kako svaka varijanta problema ovisi o specifičnom zadatku koje nastoji riješiti, tako postoje i brojna druga ograničenja. U većini slučajeva, nova ograničenja proširuju standardni VRP s kapacitativnim ograničenjima, a najpoznatiji takvi VRP su:

1. Problem usmjeravanja vozila s više skladišta (eng. *Multiple depots VRP*, skraćeno MDVRP) [14]
2. Otvoreni problem usmjeravanja vozila (eng. *Open VRP*, skraćeno OVRP) u kojemu se vozila ne moraju vraćati u skladište [15]
3. Problem usmjeravanja vozila s više ruta (eng. *VRP with Multiple Trips*, skraćeno VRPMT) u kojemu se ukida ograničenje da vozilo može odraditi samo jednu rutu [16]
4. Problem usmjerava električnih vozila (eng. *Electric VRP*, skraćeno E-VRP), u zadnje vrijeme sve popularniji VRP, a kao glavno ograničenje uzima se baterija vozila koja ograničava njegovu turu. [17]

Konstantni izazovi i promjene u distribuciji i transportu zahtijevaju brza rješenja, a samim time dolazi do stvaranja i pojave novih modela problema usmjeravanja vozila, koji se sve više pojavljuju unutar znanstvenim istraživanjima i postaju predmet proučavanju unutar obrazovanju.

2.5. Pristupi rješavanju problema usmjeravanja vozila

Suvremeni pristupi rješavanju VRP mogu se podijeliti prema različitim motivacijama kao što je prikazano na slijedećoj slici:

EGZAKTNI	HEURISTIČKI	METAHEURISTIČKI
pristup zasniva se na modelima protoka vozila, protoka tereta ili SP modelu	pristup zasniva se na heurističkoj konstrukciji ruta	pristup zasniva se najčešće na lokalnoj pretrazi vođenoj procesima koji su preuzeti iz prirode
GRANAJ I REŽI **GENERIRANJE STUPACA** **LAGRANGE-OVO OPUŠTANJE**	**CLARK AND WRIGHT** **SWEEP** **CHRISTOFIDES MIGNOZI TOTH**	**SIMULIRANO KALJENJE** **GENETIČKI ALGORITMI** **KOLONIJA MRAVI**

Slika 9. Pristupi rješavanju problema usmjeravanja vozila [8]

Prvi pristup rješavanja je potraga za egzaktnim rješenjem problema. Praktična primjena ovog pristupa vrlo je ograničena zbog svoje primjenjivosti na manji broj korisnika. Za veći broj ruta nije moguće očekivati da ovaj pristup generira rješenja uporabljiva u realnom vremenu koja se zahtijevaju u praksi. U specijalnim slučajevima kada je bitno sužen skup mogućih rješenja, često se koriste egzaktni algoritmi poput „*branch and cut*“, „*column generation*“ i „*lagrange relaxation“.* [8]

Heuristički pristup predstavlja korištenje iskustva, intuicije i vlastite procjene prilikom rješavanja nekog problema. Za razliku od egzaktnih metoda, heurističke metode ne predstavljaju znanje o strukturi ili odnosima unutar modela problema koji rješavamo, nego predstavljaju pravilo izbora, filtriranja i odbacivanja rješenja. Često se zasnivaju na konstrukciji ruta gdje se konstruiranje i poboljšavanje ruta s obzirom na ciljanu funkciju vrši iterativno. [8]

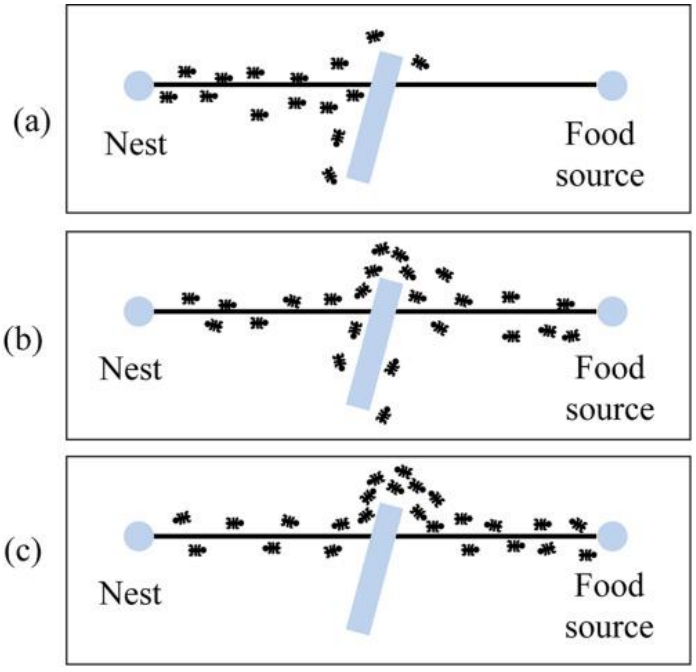
Najnoviji pristup rješavanja NP-teških problema kombinatorne optimizacije je metaheuristički, koji se u praksi definira kao skup algoritama koji se koriste pri rješavanju više različitih optimizacijskih problema gdje se samo algoritam vrlo malo mijenja u ovisnosti o problemu koji se rješava. Često se zasniva na lokalnoj pretrazi vođenoj procesima koji su preuzeti iz prirode poput *simuliranog kaljenja*, *genetičkih algoritama* i *kolonije mravi*. [8]

Prilikom rješavanja problema usmjeravanja vozila u ovom radu korišten je metaheurstički pristup utemeljen na koloniji mrava. U idućem poglavlju ukratko je obrađena optimizacija pomoću kolonije mrava kako bih čitatelj mogao što lakše pratiti kasnije poglavlje implementacije rješenja.

3. Optimizacija kolonijom mrava

Optimizacija kolonijom mrava (eng. *Ant Colony Optimization*, skraćeno ACO) je metaheuristika koja se koristi za rješavanje teških optimizacijskih problema, poput kombinatornih optimizacijskih problema koji pripadaju klasi NP-teških problema. Osnovna značajka algoritma je surađivanje skupa umjetnih mravi, koji postupno grade dovoljno dobro rješenje. Rješenje se očituje kao najbolji put na težinskome grafu. [18]

Razvijanje i proučavanje ovakve vrste optimizacije potaknula je motivacija iz prirode. Kolonija mrava predstavlja uspješnu i kompleksnu organizaciju upravo zbog međusobne suradnje mrava. Uspješnost se očituje u rješavanju složenih zadataka, za koje pojedini mrav nije u stanju sam riješiti. Gledajući s biološke strane, sposobnost vida je kod većine mravi samo djelomično razvijena, dok su neke vrste i potpuno slijepe. Proučavajući argentinske mrave, dokazano je da se većina komunikacije između mrava ostvaruje ispuštanjem feromona. Kretanjem od mravinjaka u potrazi za hranom i vraćanjem nazad, mravi iza sebe ostavljaju trag feromona. Količina feromona koje ostavljaju na pojedinom tragu može ovisiti o broju mravi koji su prošli navedenim tragom, kao i blizini, kvaliteti i količini hrane koje pronalaze u navedenoj ruti. Ostali mravi mogu namirisati feromone i slijediti njegov trag do hrane. Prilikom odabira puta kojim žele ići do hrane, s većom vjerojatnošću odabiru put s većom koncentracijom feromona. Upravo ovakav način komunikacije omogućio je mravima da pronađu što kraći put između hrane i mravinjaka. Kako bi potvrdio ovakvo ponašanje mravi, Deneubourg je razvio eksperiment dvokrakog mosta. [19]



Slika 10. Prikaz motivacije za optimizaciju kolonijom mrava [20]

ACO algoritmi koriste se za rješavanje statičkih i dinamičkih kombinatornih problema. Razlika je u tome što statički problemi su definirani samo jednom i ne mijenjaju se za vrijeme njegovog rješavanja, dok su dinamički definirani kao funkcija nekih veličina čije vrijednosti mijenjaju dinamiku ishodišnog cilja. U statičke se može svrstati dobro poznati problem trgovačkog putnika, a za primjer dinamičkog problema usmjeravanje podataka kroz mrežu računala. [18]

3.1. ACO Metaheuristika

Na većoj razini apstrakcije, ACO algoritam se može objasniti pomoću jednostavnog pseudokoda:

Algorithm 1 The Ant Colony Optimization Metaheuristic

```
Set parameters, initialize pheromone trails
while termination condition not met do
    ConstructAntSolutions
    ApplyLocalSearch (optional)
    UpdatePheromones
endwhile
```

Slika 11. Opća metaheuristika za ACO [21]

U glavnoj *while* petlji nalaze se tri glavne komponente ACO algoritma: konstruiraj mravlje rješenje, primjeni lokalno pretraživanje (opcionalno) te ažuriraj feromone. Sam postupak ne određuje direktno hoće li navedene komponente biti izvršene neovisno ili paralelnom, nego je autor algoritma ima slobodu sam odabrati, uzimajući u obzir karakteristike problema. [18]

Prije generiranja rješenja unutar *while* petlje, najprije je potrebno pročitati instance problema, pripremiti odgovarajuće strukture podataka, inicijalizirati feromonsku matricu i heurističke vrijednosti i postaviti sve parametre (ovisno o varijanti algoritma). Nakon toga započinje rad petlje. Uvjet zaustavljanja petlje također nije strogo definiran, a u većini slučajeva predstavlja odabrani broj iteracija, dozvoljeno vrijeme računanja, postizanje dovoljno dobrog rješenja i sl. [21]

U nastavku je ukratko opisana pojedina metoda koja se ponavlja unutar petlje.

3.1.1. Konstruiraj mravlje rješenje

Proces konstruiranja rješenja svake iteracije odrađuje kolonija mravi m, gdje je m oznaka za ukupan broj mravi, definiran prilikom inicijalizacije parametara te ostaje konstantan prilikom izvršavanja cijelog algoritma. Svaki mrav konstruira svoje rješenje, neovisno o drugim mravima, ali pritom koristi matricu feromonskih tragova i heurističke vrijednosti koje dijele svi mravi u koloniji. U jednoj iteraciji ACO algoritma, m mravi pronalazi ukupno m rješenja. Nakon što svaki mrav uspješno pronađe svoje rješenje i ako se to rješenje smatra dovoljno dobrim, on u zadnjem koraku (metoda ažuriraj feromonske tragove) ažurira matricu feromonskih tragova. [21]

Konstruiranje rješenja može se odvijati redoslijedno ili paralelno. Prilikom redoslijedne implementacije, svaki mrav konstruira svoje rješenje jedan iz drugog, dok svi mravi u koloniji ne pronađu svoje rješenje. U paralelnom konstruiranju rješenja, svaki mrav paralelno s drugim mravima konstruira rješenje korak po korak. U svakom koraku, pojedini mrav odabire jednu komponentu rješenja iz skupa svih komponenti koristeći feromonske tragove i heurističke informacije. Komponente koje imaju jači feromonski trag i veću heurističku vrijednost imaju veću šansu za odabir, ali to ipak ovisi i o pravilu odlučivanja varijante ACO algoritma. Ovaj postupak se ponavlja skroz dok mrav ne konstruira rješenje koje sadrži sve moguće komponente. Broj mogućih komponenti varira od problema do problema, a može biti fiksne ili varijabilne veličine. Paralelno konstruiranje rješenja može itekako ubrzati vrijeme izvršavanja programa kada se koristi višedretveno izvođenje. [21]

3.1.2. Primjeni lokalno pretraživanje

Nakon konstrukcije rješenja, slijedi primjena lokalnog pretraživanja kako bi se poboljšale cjelokupne performanse algoritma. Lokalno pretraživanje je tehnika pomoću koje se efikasno pretražuju susjedi originalnog rješenja s ciljem pronalaska još boljeg rješenja, a obično je niske računalne složenosti kako ne bi došlo do dodatnog usporavanja algoritma. Iako se ovaj korak u većini slučajeva odnosi na lokalno pretraživanje, najčešće se u literaturi zove i metoda **centralnih akcija**, iz razloga što se odvijaju akcije poput sakupljanja globalnih informacija koje se mogu koristiti kod odlučivanja o uvjetu za odlaganje dodatnih feromona. [18] [21]

3.1.3. Ažuriraj feromone

Matrica feromona smatra se najbitnijom komponentom ACO algoritma iz razloga što njezine vrijednosti imaju utjecaj na rješenje koje će konstruirati mravi u prvoj metodi petlje. Glavni cilj ove metode povećati utjecaj komponenti koje grade dobro rješenje, a smanjiti utjecaj loših komponenti. Korištenje feromosnkih tragova, algoritam može naučiti o instanci problema koju pokušava riješiti na način da umjetni mravi koriste matricu feromonskih tragova kako bi dobili znanje i iskustvo prilikom potrage za rješenjem. Svaka varijanta ACO algoritma ima svoj način za ažuriranje feromonskih tragova, a u većini slučajeva prate slijedeće korake [18]:

1. Identificiraj najbolje konstruirano rješenje
2. Primjeni isparavanje feromonskih tragova
3. Izvrši odlaganje feromonskih tragova dobrih komponenti

Početne varijante ACO algoritma zadnji korak odlaganja feromonskih tragova izvršavale su za svako konstruirano rješenje u trenutnoj iteraciji. Većina modernih varijanti ažurira komponente feromonskih tragova samo za jedno konstruirano rješenje, najčešće za najbolje rješenje, što se pokazalo kao dobra praksa. Za najbolje rješenje se može uzeti najbolje rješenje trenutne iteracije ili najbolje globalno rješenje (od početka rada algoritma odnosno do trenutne iteracije). [22]

3.2. MAX – MIN mravlji sustav

Kao što smo do sada spomenuli, ovisno o tome na koji će se način implementirati pojedini dijelovi ACO metaheuristike (inicijalizacija feromonskih tragova, način odabira komponente rješenje, isparavanje feromona, pravila odlaganja fermonskih tragova..), razlikujemo tri glavne varijante [18]:

1) Mravlji sustav (eng. *Ant System*, skraćeno AS)
2) Sustav mravlje kolonije (eng. *Ant Colony System*, skraćeno ACS)
3) MAX-MIN mravlji sustav (eng. *MAX-MIN Ant System*, skraćeno MMAS)

Osim glavnih varijanti, postoje i brojne druge izvedene varijante te se istražuju nove. Najpopularnija varijanta ACO algoritma za rješavanje standardnih kombinatornih problema je MMAS, koja s većim brojem iteracija algoritma daje najbolje rješenje. [23] Iz toga razloga je

upravo MMAS odabran za implementaciju rješenja VRP s ograničenjem kapaciteta te je obrađen u nastavku ovog poglavlja, dok su ostale varijante samo navedene.

MAX -MIN mravlji sustav naprednija je verzija mravljeg sustava AS, koja pokušava riješiti raniju stagnaciju koristeći efektivan mehanizam odlaganja feromonskih tragova. MMAS se razlikuje u tri ključna aspekta od svojih prethodnika [24]:

1) Nakon svake iteracije samo jedan mrav dodaje feromonski trag: mrav koji je pronašao najbolje rješenje trenutne iteracije ili mrav koji je našao najbolje rješenje od početka algoritma (tzv. globalni mrav)
2) Kako bi se izbjegla stagnacija pretraživanja, raspon mogućih tragova feromona na svakoj komponenti je ograničena intervalom $[\tau_{min}, \tau_{max}]$
3) Prilikom inicijalizacije, namjerno postavljamo feromonske tragove na τ_{max}, postižući na ovaj način veće istraživanje na početku algoritma

Nakon svake iteracije algoritma, jedan mrav ažurira feromonske tragove. Ažuriranje feromonskih tragova odvija se prema formuli:

$$\tau_{ij} = \tau_{ij} + \Delta\tau_{ij}^{best}$$

gdje je $\Delta\tau_{ij}^{best} = \frac{1}{C^{best}}$. C^{best} može predstavljati duljinu ture najboljeg mrava do sada (C^{bs}) i u tom slučaju vrijedi $\Delta\tau_{ij}^{best} = \frac{1}{C^{bs}}$ ili duljinu ture najboljeg mrava u iteraciji (C^{ib}) pa je $\Delta\tau_{ij}^{best} = \frac{1}{C^{ib}}$. U implementacijama MMAS se koriste oba pravila ažuriranja i to naizmjenično. Izbor relativne frekvencije kojom se ta dva pravila primjenjuju ima utjecaj na to koliko je algoritam pohlepan. Kada ažuriranje feromona uvijek primjenjuje najbolji mrav dosad, tada je pretraživanje ubrzo fokusirano oko T^{bs} ture, a ako feromone ažurira najbolji mrav u iteraciji, tada je broj bridova koji dobiju feromone veći i pretraživanje je manje usmjereno. Eksperimentalni rezultati pokazuju da je za male probleme najbolje koristiti ažuriranje feromona od mrava najboljeg u iteraciji, a za velike probleme najbolje performanse daje dinamička kombinacija najboljeg globalnog i najboljeg u iteraciji s postepenim povećanjem frekvencije učestalosti najbolje ture T^{bs} do sada. [18] [24]

Neovisno o tome koju duljinu ture mrava odaberemo za ažuriranje feromonskih tragova, potraga za rješenjem i dalje može stagnirati. Stagnacija se odvija u onom trenutku kada je feromonski trag znatno veći za jednu komponentu za razliku od ostalih. Prilikom odabira iduće komponente u konstruiranju rješenja, mrav će preferirati navedenu komponentu rješenja nad svim ostalim alternativama. Kada nastupi ovakva situacija, mravi će svaku iteraciju konstruirati isto rješenje te u ovom slučaju prestaje istraživanje novih

mogućnosti. Kako bi se izbjegao ovakav scenarij, postavlja se raspon mogućih tragova feromona za svaku komponentu rješenja. Raspon se određuje tako da se postavlja donja i gornja granica τ_{min} i τ_{max}. Ovakve granice imaju efekt ograničavanja vjerojatnosti ρ_{ij} odabira grada j kada se mrav nalazi u gradu i, na interval $[\rho_{min}, \rho_{max}]$, gdje je $0 < \rho_{min} \leq \rho_{max} \leq 1$. Nakon svake iteracije izvršava se provjera vrijednosti feromonskih tragova te u slučaju da je $\tau_{ij} > \tau_{max}$ postavlja se vrijednost $\tau_{ij} = \tau_{max}$, odnosno ako je $\tau_{ij} < \tau_{min}$ postavlja se slijedeća vrijednost $\tau_{ij} = \tau_{max}$. Kako određivanje vrijednosti τ_{min} i τ_{max} izlaze van okvira ovoga rada, formule za njih dane su u nastavku [24]:

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

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

gdje je avg prosječan broj različitih izbora dostupnih mravu prilikom konstruiranja rješenja, a n ukupan broj komponenti odnosno broj gradova u problemu usmjeravanja vozila (za dodatna objašnjenja kako su dobivene navedene formule, pogledati članak MAX-MIN Ant System u časopisu Future Generation Computer Systems T. Stützlea, izvor [24])

U MMAS algoritmu, početni feromonski trag postavlja se na vrijednost τ_{max}. Ova vrsta inicijalizacije je odabrana iz razloga što će se povećati istraživanje širokog broja rješenja u prvoj iteraciji (veći prostor pretrage). Da bi se povećao prostor pretraživanja puteva koji imaju malu vjerojatnost da budu odabrani, u praksi se trag feromona ponovno inicijalizira. Ponovna inicijalizacija okida se kada se dosegne određena razina stagnacije (uvjet je definirani broj iteracija algoritma za koje se nije došlo do boljeg rješenja) [18] [24]

4. Implementacija ACO za CVRP

Problem usmjeravanja vozila s ograničenjem kapaciteta je odabran za implementaciju iz razloga što je ovaj problem osnovni problem te je sve ostale varijante moguće izvesti preko njega uvodeći dodatno ograničenje. Kao što je u prethodnom poglavlju rečeno, za rješavanje CVRP-a odabrana je MAX-MIN varijanta mravljeg sustava, za koju je dokazano u raznim literaturama da daje najbolje rezultate. Na sljedećoj slici prikazan je pseudokod konstrukcije rješenja koji je poslužio kao smjernica prilikom implementacije algoritma:

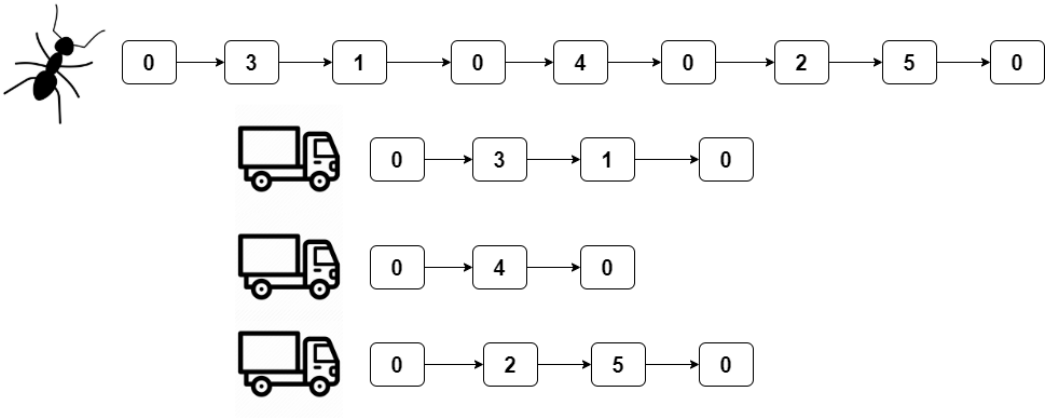
```
1.  while not terminated
2.    for each ant in the colony
3.      set all nodes as unvisited
4.      while number of unvisited nodes > 0 do
5.        compute ant's probability of going to
          unvisited nodes
6.        select a node according to the probability
7.        if ant.load + node.load > ant.capacity then
8.          return to the depot
9.        else
10.         visit the selected node
11.     return to the depot
12.   save the best solution if found
13.   evaporate pheromone trails
14.   update pheromone trails
15. return the best solution
```

Slika 12. Pseudokod algoritma ACO za CVRP [25]

Uvjet za zaustavljanje *while* petlje je određeni broj iteracija, a u svakoj iteraciji svi mravi konstruiraju rješenje. Unutar konstrukcije rješenje najprije se postavlja da su svi vrhovi grafa koji predstavljaju gradove neposjećeni. Zatim se pokreće nova *while* petlja koja za uvjet ima zaustavljanje kada se posjete svi gradovi. Temeljni dio konstrukcije rješenja prema ACO metaheuristici upravo se odvija unutar ove petlje. Tako se najprije za trenutnog mrava računaju vjerojatnosti odlaska u idući neposjećeni grad i na temelju metode **kolo sreće** odabire se vrh koji će pripasti komponenti rješenja. Upravo ovdje nastupa jedino ograničenje CVRP-a, gdje se mora provjeriti je li trenutni kapacitet mrava dovoljan (manji ili jednak) da se posjeti dobiveni odabrani grad. U slučaju da uvjet prolazi, mrav dodaje grad u svoju komponentu rješenje, u protivnom se vraća u centralno skladište. Kada svaki mrav posjeti

sve gradove, pokreću se metoda statistike, koja zapisuje najbolje rješenje trenutne iteracije i s obzirom na ostale iteracije provjerava je li takvo rješenje ujedno i najbolje globalno rješenje. Zadnje metode unutar iteracije koje su ostale prate pseudokod ACO metaheuristike, odnosno ažuriraju se feromonski tragovi.

Potrebno je naglasiti da pojedini mrav ovdje ne predstavlja jedno vozilo, nego zapravo konstruira cjelokupno rješenje. Svaki mrav sadrži listu vozila određenih kapaciteta. Unutar konstrukcije rješenja, mrav koristi jedno vozilo iz liste vozila te prema ograničenjima CVRP-a uzima druge vozilo iz liste ako trenutno ne sadrži dovoljno kapaciteta da može posjetiti odabrani grad. Zamjena vozila se vrši prilikom povratka mrava u centralno skladište. Takav postupak se ponavlja skroz dok se ne obiđu svi gradovi ili se ne iskoriste sva vozila.



Slika 13. Prikaz izgleda rješenja dobivenog implementacijom

U nastavku ovog poglavlja opisani su detaljno koraci koje je potrebno napraviti da bi se implementirao MAX-MIN mravlji sustav za CVRP prema uzoru na objašnjeni pseudokod. Implementacija će se odviti u jeziku Pythonu prema objektno-orijentiranim principima, koristeći određene biblioteke za rad s matematičkim funkcijama.

4.1. Metode i tehnike rada

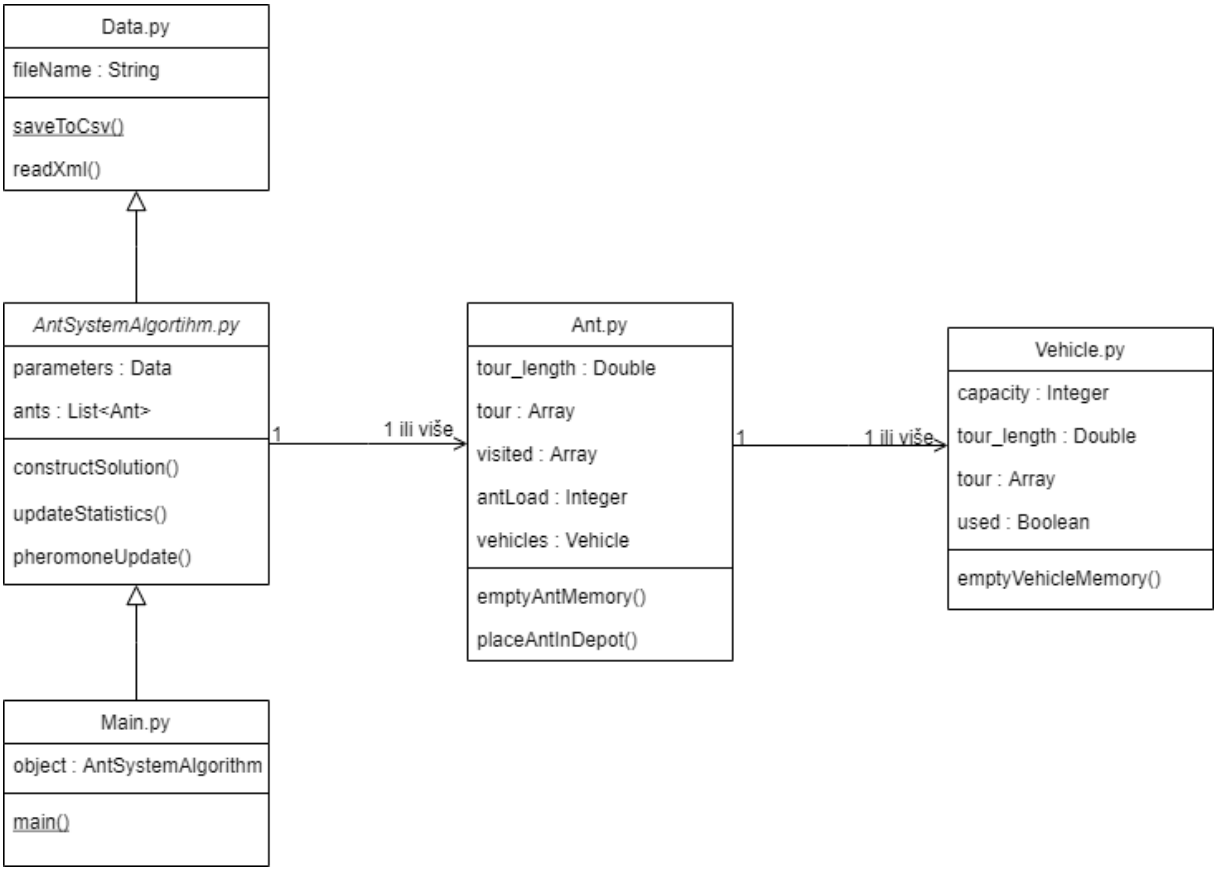
Za prethodno istraživanje teoretskog dijela problema usmjeravanja vozila, kao i opisa optimizacije mravljom kolonijom, korišteni su svi sadržaji dostupni na internetu poput knjiga i znanstvenih članaka.

Za implementaciju algoritma i rješavanja jedne varijante problema usmjeravanja vozila korišteno je programsko okruženje *Visual Studio Code* s jezikom *Python*. Kako su u pojedinim koracima rješavanja algoritma korištene matrice, prvenstveno za učitavanje podataka, korištena je biblioteka *Numpy* (https://numpy.org/) koja sadrži brojne funkcije za rad s matricama. Također, za dodatnu inspiraciju prilikom pisanja algoritma korištena je kombinacija različitih pseudokodova dostupnih na internetu s naznačenom poveznicom preuzimanja.

Na kraju je izvedena statistika na temelju samoga rada algoritma, poput analize između definiranih parametara i rješenja, vremenske složenosti i sl. Odabrana instanca problema na kojoj je izvršeno testiranje preuzeta je s dostupnih online izvora. Za prikaz vrhova i analizu ruta, korištena je online aplikacija *GeoGebra* (https://www.geogebra.org/classic?lang=en). Statistika je prikazana pomoću kreiranih tablica i grafova izrađenih u programu MS Excelu. Pojedina statistika je preuzeta i s interneta te naznačena da se ne radi o vlastitim izmjerenim podacima.

4.2. Dijagram klasa

Implementacija rješenja sastoji se od pet klasa koje su prikazane na sljedećem dijagramu:



Slika 14. Dijagram klasa implementacije rješenja

Klasa *Main.py* sadrži samo glavnu metodu za pokretanje programa, a ona izgleda slično kao pseudokod za ACO metheuristiku:

```
def main():
      #Inicijalizacija podataka
      antSystemAlgorithm = AntSystemAlgorithm(Data(sys.argv[1]))
      i = 0
      while(i < brojIteracija):
            antSystemAlgorithm.ConstructSolution()
            antSystemAlgorithm.UpdateStatics()
            antSystemAlgorithm.ASPheromoneUpdate()
            i = i + 1
```

Klasa *Data.py* sadrži metodu za čitanje instanci iz .xml datoteka čije se ime unese preko argumenta prilikom pokretanja programa. Također sadrži i za metodu za spremanje dobivene statistike unutar .csv datoteke. Osim toga, unutar nje se definiraju vrijednosti parametara potrebnih za rješavanje algoritma. Vrijednosti parametara se ručno unose kako bi se olakšalo testiranje za različite kombinacije.

Klasa *Ant.py* služi za kreiranje istih mrava ovisno o definiranom broju mrava unutar parametara. Objašnjenje njezinih atributa definirano je u sljedećoj tablici:

Tablica 1. Prikaz atributa klase Ant

Atribut	Objašnjenje
tour_length	Duljina ture koju je prošao mrav, inicijalno je vrijednost postavljena na -1,
tour	Memorija mrava koja pamti trenutni/konačni prevaljeni put, odnosno polje koje sadrži posjećene gradove.
visited	Polje veličine n, gdje je n broj gradova. Svaki indeks polja označava o kojem se vrhu/gradu radi. Služi za provjeru prilikom odabira grada koje će mrav posjetit i implementaciju uvjeta da se svaki vrh može posjetiti samo jednom Inicijalno sve vrijednosti polja su postavljene na *False*.
antLoad	Označava koliko je mrav trenutno napunjen. Prilikom posjeta određenom gradu, nova vrijednost se računa suma kao trenutne vrijednosti atributa i količine potražnje grada ($\boldsymbol{antload} = antload + c_{ij}$). Također služi za provjeru ograničenja kapaciteta vozila prilikom odabira grada. Inicijalno je vrijednost postavljena na 0.
vehicles	Atribut koji sadrži listu od K vozila (u programskom smislu, Vehicle objekata). Broj K ovisi o instanci problem za koju se testira, točnije kako je u instanci definirano
vehicleInUse	Objekt tipa Vehicle u koji se sprema vozilo koje se trenutno koristi. Inicijalno se postavlja kao prvo vozilo iz liste vozila.

Klasa *Vehicle.py* služi za kreiranje vozila ovisno o definiranom broj vozila K i njihovim kapacitetima C_i (gdje je i redni broj vozila). Ove vrijednosti ovise o instanci problema za koju

se algoritam primjenjuje. Svaki mrav sadrži listu vozila K. Objašnjenje atributa klase *Vehicle.py* nalazi se u slijedećoj tablici:

Tablica 2. Objašnjenje atributa klase Vehicle.py

Atribut	Objašnjenje
capacity	Određuje kapacitet vozila definiran u instanci problema za koje se traži rješenje.
tour_length	Duljina ture koju je prošlo vozilo, inicijalno je vrijednost postavljena na -1,
tour	Memorija vozila koja pamti trenutni/konačni prevaljeni put, odnosno polje koje sadrži posjećene gradove.
usedVehicle	Atribut koji služi kao indikator (tipa *Boolean*) je li trenutno vozilo korišteno. Inicijalno je postavljeno na vrijednost *False*, a vrijednost mijenja kada se vozilo zamjenjuje s trenutnim.

Klasa *AntSystemAlgorithm.py* sadrži glavne metode koje se primjenjuju unutar MAX-MIN mravljeg sustava za rješavanje CVRP-a, a koristi podatke iz klase *Data.py*, koje po potrebi dodatno obrađuje.

Nakon što smo objasnili koja je svrha pojedine klase, u nastavku ćemo objasniti pojedine metode iz klase *AntSystemAlgorithm.py*. Poštujući ACO metaheuristiku, krenuti ćemo od prvog važnog koraka - inicijalizacije podataka i parametra algoritma.

4.3. Inicijalizacija parametara i feromonskih tragova

Metoda inicijalizacije algoritma može se podijeliti na čitanje parametara koje predstavljaju gotove vrijednosti (ručno unesene ili automatski učitane iz instance problema) i vrijednosti koje treba izračunati preko ulaznih podataka. Pod parametre algoritma smatramo α, β, ρ i broj mravi (m). Ove vrijednosti se određuje prema vlastitoj intuiciji, vrsti problema koji se rješava i stečenom iskustvu prilikom implementiranja sustava optimizacije mravima. U raznim literaturama postoje ponuđene vrijednosti ovih parametara, a eksperimentalno je dokazano da daju dobra rješenja. Tako naprimjer, dobri parametri za algoritam MMAS prilikom rješavanja problema trgovačkog putnika su [24]:

- $\alpha = 1$
- β u intervalu između 2 i 5
- $\rho = 0.02$
- Broj mrava (m) = Broj gradova (n)

Kada bi CVRP-u maknuli ograničenje kapaciteta i postavili samo jedno vozilo umjesto njih više, zapravo bismo dobili problem trgovačkog putnika. Iz toga razloga, prethodne vrijednosti parametra mogu biti dovoljno dobre i za rješavanje CVRP-a. Tijekom testiranja algoritma za pronalaženje dobrih rješenja, parametri su prilagođeni vlastitim eksperimentiranjem i kombinacijama, o čemu će više biti riječ u poglavlju eksperimentalnog istraživanja.

Ulazne podatke instance problema potrebno je obraditi kako bi se dobile vrijednosti poput udaljenosti između gradova, potražnje pojedinog grada, broju vozila i njihovih kapaciteta te inicijalizacije početnih feromonskih tragova.

Udaljenosti između gradova se zapisuje u simetričnu matricu dimenzija $n\,x\,n$, gdje je n broj gradova učitanog CVRP-a. Kako su gradovi u instancama problema definirani kao vrhovi u koordinatnom sustavu, odnosno točke $N(x, y)$, za izračun udaljenosti koristi se Pitagorin poučak:

$$d(N_1, N_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

A u programskom kodu to izgleda ovako:

```
    def CalculateDistanceMatrix(self):
        distanceMatrix = np.zeros(
            (self.numberOfCities, self.numberOfCities))
        for i in range(self.numberOfCities):
            for j in range(i, self.numberOfCities):
                distanceMatrix[i][j] = math.sqrt((self.coordinates[i][0] -
self.coordinates[j][0]) ** 2 + (
                    self.coordinates[i][1] - self.coordinates[j][1]) ** 2)
                distanceMatrix[j][i] = distanceMatrix[i][j]
        return distanceMatrix
```

Feromonski tragovi se također zapisuju u simetričnu matricu dimenzija $n\ x\ n$, ali do njezinog izračuna se dolazi na malo teži način. Najprije je potrebno izračunati listu najbližih gradova odnosno $nn\_list[i][k] = j$, koja nam daje indeks k-tog grada najbližeg gradu i. U programskom kodu koristeći biblioteku *Numpy* i njezinu funkciju *argsort*, na lagani način pronalazimo listu najbližih gradova [26]:

```
def findNearestNeighbourList(self):
        listOfNeighbour = []
        for i in range(self.numberOfCities):
            listOfNeighbour.append(np.argsort(self.distanceMatrix[i]))
        return listOfNeighbour
```

U idućem koraku je potrebno izračunati duljinu puta generiranog heuristikom najbližega susjeda C^{nn}. Označimo li put generiran heuristikom najbližega susjeda s T_h. U svakoj iteraciji se odabire vrh od ukupnog broja vrhova n koji još nije dio ture T_h, takav da je najbliži zadnjem dodanom vrhu u T_h. Odabrani vrh se dodaje u turu T_h te se ovaj postupak ponavlja skroz dok svi vrhovi nisu dio ture T_h. C^{nn} je na kraju suma svih bridova između susjeda navedene ture. [27]

```
def NearestNeighbourHeuristic(self):
    visited_cities = set()
    current_city = 0
    visited_cities.add(0)
    distance_passed = 0
    i = 0
    while i < self.numberOfCities:
        for city in self.findNearestNeighbourList()[current_city]:
            if city not in visited_cities:
                distance_passed = distance_passed + \
                    self.distanceMatrix[current_city, city]
                current_city = city
                visited_cities.add(current_city)
                break
        i = i + 1
    return distance_passed
```

Kako ovaj algoritam implementira MMAS varijantu, potrebno je odrediti početne vrijednosti gornje i donje granice, odnosno τ_{max} i τ_{min}. Kao što je prethodno objašnjeno, u MMAS varijanti početna vrijednost traga feromona iznosi τ_{max}, kako bi algoritam u prvoj iteraciji istraživao širi spektar rješenja. Vrijednosti početne gornje i donje granice određuju se pomoću formula [28]:

$$\tau_{max} = \frac{1}{\rho * C^{nn}}$$

$$\tau_{min} = \frac{\tau_{max}}{2 * n}$$

Sve varijable unutar formula su nam poznate pa sada možemo izračunati početnu feromonsku matricu:

```
def InitializePheromoneTrail(self):
    self.trail_max = 1 / (self.rho * self.NearestNeighbourHeuristic())
    self.trail_min = self.trail_max / (2 * self.numberOfCities)
    self.pheromoneMatrix = np.full(
        (self.numberOfCities, self.numberOfCities), self.trail_max)
```

Osim inicijalizacije navedenih parametara i obrade ulaznih podataka, također su inicijalizirane varijable koje čuvaju podatke za obradu statistike, kao npr. rješenje svake iteracije, najbolje rješenje do sada, vrijeme potrebno za izvršavanje pojedine iteracije i sl.

4.4. Konstruiranje rješenja

Konstrukcija rješenja se sastoji od nekoliko koraka:

1) Najprije je potrebno isprazniti memoriju svakog mrava i vozila, na način da se očiste sve ture te vozila označe kao nekorištena

```
# pražnjenje memorije mrava
        for k in range(self.numberOfAnts):
            for i in range(self.numberOfCities):
                self.ant[k].visited[i] = False
            self.ant[k].tour.clear()
            self.ant[k].antLoad = 0
            # pražnjenje memorije vozila
            for v in range(self.numberOfVehicles):
                self.ant[k].vehicle[v].vehicleTour.clear()
                self.ant[k].vehicle[v].usedVehicle = False
```

2) Svi mravi se postavljaju u centralno skladište te se uzima prvo vozilo iz liste vozila i postavlja za korištenje

```
# Postaviti mrave u depot
        for k in range(self.numberOfAnts):
            self.ant[k].tour.append(self.depot)
            self.ant[k].visited[self.depot] = True
# Postaviti prvo vozilo za korištenje
            self.ant[k].vehicleInUse = self.ant[k].vehicle[0]
            self.ant[k].vehicleInUse.usedVehicle = True
            self.ant[k].vehicleInUse.vehicleTour.append(0)
```

3) Svaki mrav konstruira kompletan put dok ne posjeti sve gradove. U svakom koraku konstrukcije, mrav primjenjuje metodu pravila odabira idućeg vrha.

Metoda pravila odabira kao argumente prima indeks trenutnog mrava k i indeks trenutnog koraka konstrukcije rješenja, kako bi se moglo saznati u kojem se gradu i trenutno nalazi

mrav k. Nakon toga se računa vjerojatnost odlaska mrava k iz grada i u neposjećene gradove j prema formuli [27]:

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

gdje je:

- η_{ij} heuristička vrijednost koja se računa kao recipročna udaljenost između dva vrha prema formuli $\eta_{ij} = \frac{1}{d_{ij}}$,
- N_i^k skup dopuštenih susjeda mravak k u gradu i,
- τ_{ij} feromonski trag između gradova.
- α i β dva paramtera koji određuju relativan utjecaj tragova feromona i heurističke vrijednosti

Kao što je vidljivo iz formule, odabir vrha većinom ovisi o tragu feromona i vrijednosti heuristike. U programskom kodu, metoda za izračun vjerojatnosti izgleda na slijedeći način:

```
def CalculateProbability(self, current, choice, k):
        probability = 0.0
        brojnik = self.pheromoneMatrix[current][choice] ** self.alpha * \
            (1 / self.distanceMatrix[current][choice]) ** self.beta
        nazivnik = 0.0
        for i in range(self.numberOfCities):
            if self.ant[k].visited[i] == False:
                nazivnik = nazivnik + self.pheromoneMatrix[current][i] ** s
elf.alpha * \
                    (1 / self.distanceMatrix[current][i]) ** self.beta
        probability = brojnik / nazivnik
        return probability
```

Na temelju izračunatih vjerojatnosti, primjenjuje se metoda „*kola sreće*“ koja radi na principu da se najprije izračuna kumulativna suma svih vjerojatnosti. Nakon toga se generira uniformno distribuiran slučajan broj r iz intervala $[0, 1]$, iz razloga što kumulativna suma završava s vrijednosti 100% odnosno jedan. Na kraj se odabire onaj grad kojemu je vrijednost kumulativne sume veća ili jednaka broju r. [29]

Prije samo odabira grada, odnosno dodavanja vrha turi mrava, potrebno je provjeriti uvjet ograničenja kapaciteta koji je prikazan u pseudokodu algoritma. U slučaju da je uvjet

zadovoljen, mrav se pomiče u odabrani grad i označuje se kao posjećen, a u protivnom mrav se vraća u centralno skladište.

```
def DecisionRule(self, k, step):
        …
            cumulativ = np.cumsum(selection_probability)
            r = np.random.uniform(0, 1)
            j = 0
            p = cumulativ[j]
            while(p <= r):
                j = j + 1
                p = cumulativ[j]
            if(self.ant[k].antLoad + self.demands[j -
1] > self.ant[k].vehicleInUse.capacity):
                self.ReturnToDepot(k)
            else:
                self.ant[k].tour.append(j)
                self.ant[k].visited[j] = True
                self.ant[k].antLoad = self.ant[k].antLoad + self.demands[j-
1]

                self.ant[k].vehicleInUse.vehicleTour.append(j)
```

4) Na kraju, mrav se vraća natrag u početni grad (isto kao i trenutno vozilo) te se računa duljina ukupnog prijeđenog puta

```
self.ant[k].tour.append(self.ant[k].tour[0])
self.ant[k].vehicleInUse.vehicleTour.append(self.ant[k].tour[0]
```

Metoda koja računa ukupni prijeđeni put mrava:

```
    def ComputeTourLength(self, k):
        tour = self.ant[k].tour
        numOfVisitedNodes = len(tour)
        tour_length = 0.0
        for i in range(numOfVisitedNodes-1):
            indexSource = tour[i]
            indexDestination = tour[i+1]
            tour_length = tour_length + \
                self.distanceMatrix[indexSource][indexDestination]
        return tour_length
```

Ovi koraci se ponavljaju toliko dugo dok mravi ne završe cijelu turu.

4.5. Ažuriranje feromonskih tragova

Ažuriranje matrice feromonskih tragova je posljednji korak u iteraciji algoritme, a prema MMAS varijanti sastoji se od tri dijela: isparavanje feromona za parametar ρ, odlaganje novih feromona te provjere gornje i donje granice feromona.

```
    def MMASPheromoneUpdate(self):
        self.Evaporate()
        self.DepositPheromone()
        self.CheckPhermoneLimits()
```

U postupku isparavanja feromona, vrijednost traga feromona τ_{ij} se smanjuje za konstantni faktor ρ prema formuli [27]:

$$\tau_{ij} = (1 - \rho)\tau'_{ij}$$

Tako da metoda izgleda poprilično jednostavno:

```
def Evaporate(self):
        self.pheromoneMatrix = (1 - self.rho) * self.pheromoneMatrix
```

Odlaganje novih feromona je postupak pri kojemu se dodaju novi feromoni na bridove koji su konstruirali mravi u svojim turama. U poglavlju MMAS opisano je nekoliko načina odlaganja feromona, a u ovoj implementaciji rješenja odabrana je varijanta u kojoj samo najbolji mrav odlaže svoje feromone. U implementaciji, odlaganje feromona globalno najboljeg mrava izgleda ovako:

```
    def DepositPheromone(self):
        deltaTau = 1 / self.bestAnt.tour_length
        numberOfVisitedNodes = len(self.bestIterationAnt.tour)
        for i in range(numberOfVisitedNodes - 1):
            j = self.bestIterationAnt.tour[i]
            l = self.bestIterationAnt.tour[i + 1]
            self.pheromoneMatrix[j][l] = self.pheromoneMatrix[j][l] + delta
Tau
            self.pheromoneMatrix[l][j] = self.pheromoneMatrix[j][l]
```

Zadnji korak u MMAS varijanti je provjera gornje i donje granice feromona, koja kontrolira da rješenja budu unutar tražene granice $[\tau_{min}, \tau_{max}]$.

```
def CheckPhermoneLimits(self):
    for i in range(self.numberOfCities):
        for j in range(i):
            if self.pheromoneMatrix[i][j] < self.trail_min:
                self.pheromoneMatrix[i][j] == self.trail_min
                self.pheromoneMatrix[j][i] == self.trail_min
            elif self.pheromoneMatrix[i][j] > self.trail_max:
                self.pheromoneMatrix[i][j] == self.trail_max
                self.pheromoneMatrix[j][i] == self.trail_max
```

Ovim korakom završena je cjelokupna implementacija algoritma, a koliko su dobra rješenja dobivena pokazati će iduće poglavlje. Jedina metoda koja nije objašnjena je ažuriranje statistike, iz razloga što ona ovisi o problemu vlastitim preferencijama mjerenja te ne utječe na rad i efikasnost rješenja.

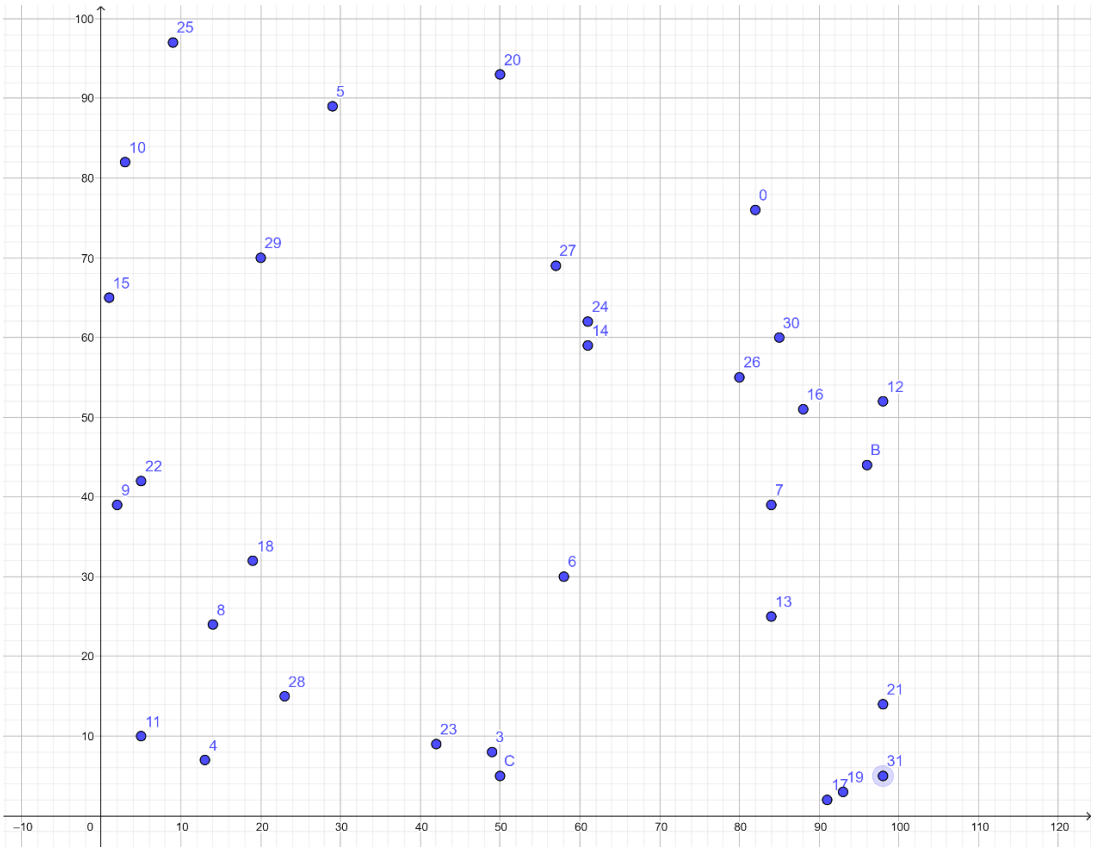
5. Eksperimentalno istraživanje

Eksperimentalno istraživanje provodilo se na instancama problema *Augerat 1995 – Set A* s izvora [30]. Posebno je odabrana instanca problema *A-32-k05.xml* na kojoj je izveden eksperiment s različitim postavkama parametara (ukupno 5 kombinacija) koji su navedeni u slijedećoj tablici:

Tablica 3. Postavke parametara za testiranje

	α	**β**	**ρ**	**Broj iteracija**	**Broj mravi**	**Broj gradova**
1)	1	2	0.02	1000	32	32
2)	1	4	0.02	1000	32	32
3)	1	2	0.02	500	64	32
4)	1	2	0.1	1000	32	32
5)	1	2	0.1	1000	32	32

Instanca problema *A-32-k05.xml* sadrži ukupno 32 grada i 5 vozila, svako vozilo ima jednaki kapacitet koji iznosi 100. Pojedini gradovi prikazani su u koordinatnom sustavu na slijedećoj slici:



Slika 15. Prikaz vrhova instance problema

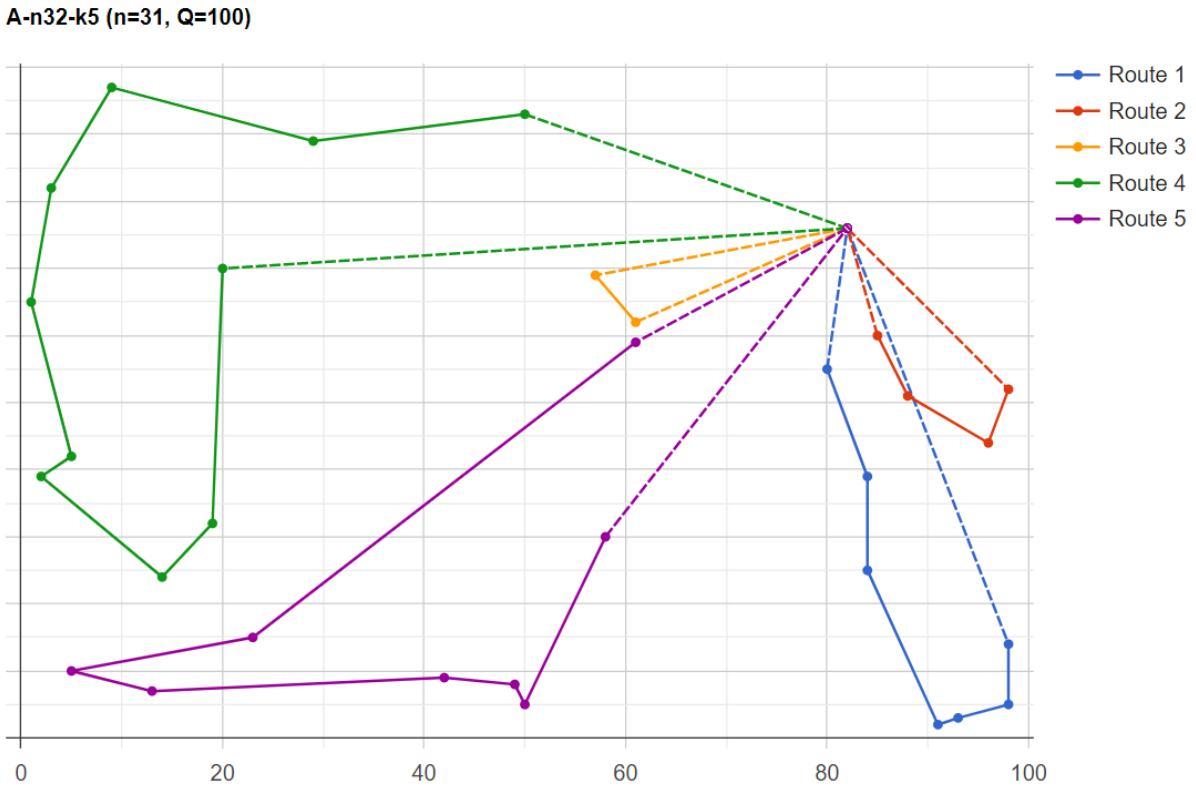
Svaki od ukupno 32 grada ima svoje zahtjeve prikazane u slijedećoj tablici:

Tablica 4. Prikaz potražnje za pojedini grad

Rbr. grada	Potražnja	Rbr. grada	Potražnja	Rbr. grada	Potražnja	Rbr. grada	Potražnja
0	0	**8**	6	**16**	18	**24**	24
1	19	**9**	16	**17**	19	**25**	24
2	21	**10**	8	**18**	1	**26**	2
3	6	**11**	14	**19**	24	**27**	20
4	19	**12**	21	**20**	8	**28**	15
5	7	**13**	16	**21**	12	**29**	2
6	12	**14**	3	**22**	4	**30**	14
7	16	**15**	22	**23**	8	**31**	9

Iz tablice možemo primijetiti da je centralno skladište grad označen s rednim brojem nula te iz toga razloga njegova potražnja iznosi 0.

Kao najbolje moguće rješenje navodi se duljina puta od 784 (npr. kilometara), a svaka ruta vozila rješenja prikazana je na slijedećoj slici:



Slika 16. Prikaz optimalnog rješenja za A-32-k5.xml [31]

Za svaku kombinaciju parametara eksperiment je ponovljen točno 21 put. Prikupljali su se podaci o najboljem rješenju svake iteracije te Na temelju dobivenih podataka, provedena je statička analiza, pri kojom su izračunate aritmetička sredina i medijan najboljih rješenja te aritmetička sredina i medijan iteracija u kojoj je pronađeno najbolje rješenje. Medijan se uzima iz razloga što je manje osjetljiv na ekstremne vrijednosti od aritmetičke sredine, ali osim toga postoje različite prednosti korištenja medijana za mjerenje performansi ovakvih algoritama. [32] Kako bismo lakše svrstali sve podatke u tablicu, uvodimo legendu oznaka:

- R_{best} – najbolje rješenje
- $\overline{R_{best}}$ – aritmetička sredina najboljeg rješenja
- $\overline{\overline{R_{best}}}$ – medijan najboljeg rješenja
- I_{tr} – iteracija u kojoj je pronađeno najbolje rješenje
- $\overline{I_{tr}}$ – aritmetička sredina iteracija u kojoj je pronađeno najbolje rješenje
- $\overline{\overline{I_{tr}}}$ – medijan iteracija u kojoj je pronađeno najbolje rješenje

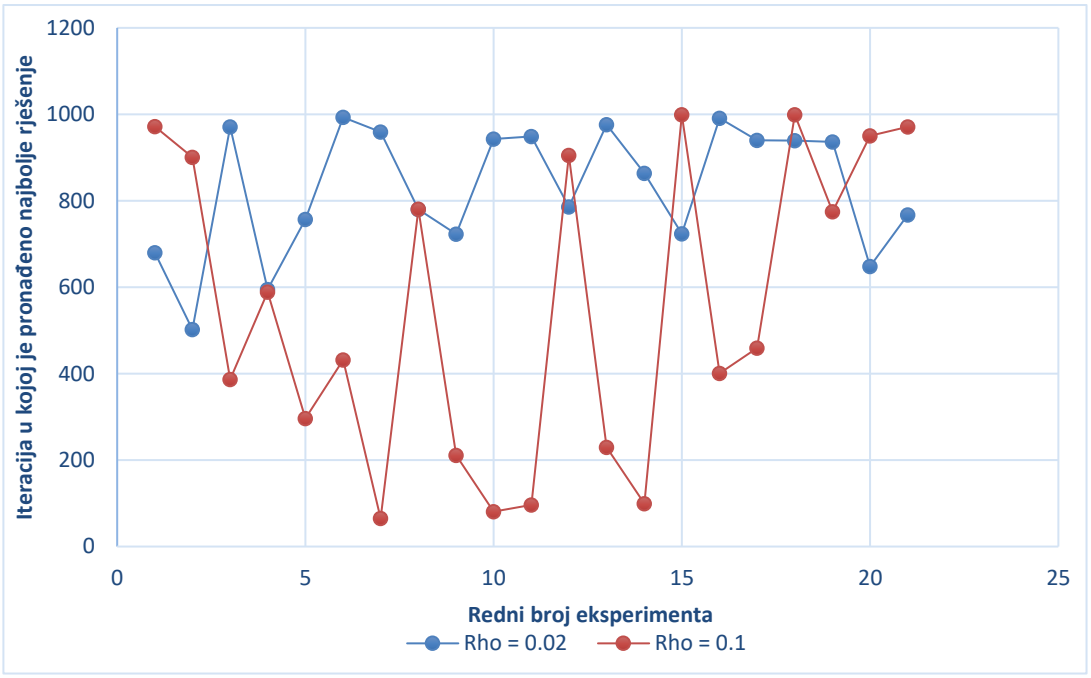
5.1. Ovisnost o parametrima β i ρ

Najprije ćemo promatrati ovisnost implementiranog algoritma o parametrima β i ρ, uz ostale konstante definirane vrijednosti. U slijedećoj tablici prikazani su dobiveni rezultati:

Tablica 5. Prikaz rezultata ovisnosti algoritma o parametrima β i ρ

	β	**ρ**	$\mathbf{R_{best}}$	$\overline{\mathbf{R_{best}}}$	$\overline{\overline{\mathbf{R_{best}}}}$	$\overline{\mathbf{I_{tr}}}$	$\overline{\overline{\mathbf{I_{tr}}}}$
1)	2	0.02	822,74	828,7123	828,69	829,67	864
2)	4	0.02	837,79	845,601	838,11	729,1429	765
4)	2	0.1	821,43	834,881	831,94	552,3333	459
5)	4	0.1	859,62	855,7571	859,62	548,619	686

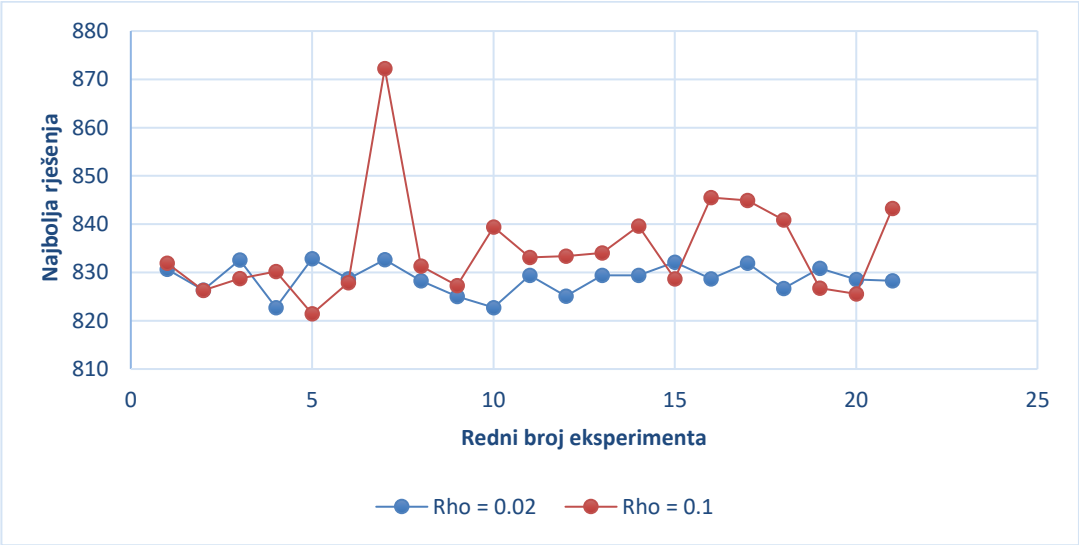
Kao što je vidljivo iz tablice, najbolje rezultate daje kombinacija parametara za $\beta = 2$ i $\rho = 0.02$ ako promatramo aritmetičku sredinu i medijan. Razlog tome je što MMAS varijanti pogoduju niži postavljeni parametri pa algoritam ne radi toliko pohlepno te se više fokusira na dobivanje najboljeg rješenja preko matrice feromonskih tragova. Za ovu kombinaciju parametara također možemo primijetiti da se najbolje rješenje dobiva tek u kasnijim iteracijama, što znači da će se što bolje rješenje dobiti ako algoritam radi duže. Na to itekako utječe parametar ρ, kao što je vidljivo u slijedećem grafu:



Grafikon 1. Prikaz odnosa rednog broja eksperimenta i I_{tr}

Zaključujemo da za manji parametar ρ dobivamo najbolja rješenje većinom u kasnijim iteracijama. Veća vrijednost parametar ρ daje veću raspršenost uzorka u tome kada će se rješenje pronaći pa iz toga ne možemo izvući zaključak.

Povećamo li parametar ρ, dobivamo poprilično slične rezultate za medijan najboljih rješenja, ali ipak malo različitu aritmetički sredinu. Također možemo primijetiti da je upravo za veću vrijednost parametra ρ, dobiveno i najbolje optimalno rješenje u iznosu od 821. Najbolji prikaz ovisnosti algoritma o parametru ρ prikazan je na slijedećem grafu:



Grafikon 2. Prikaz odnosa R_{best} i broja eksperimenta s obzirom na parametar ρ

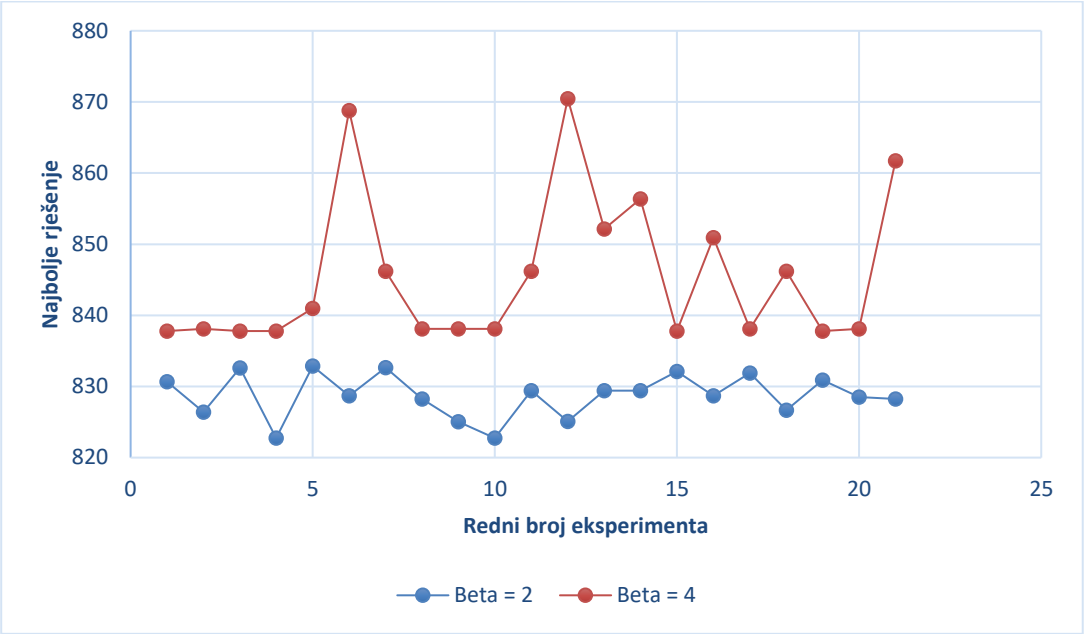
Kao što je vidljivo iz grafa, veći parametra ρ također utječe na veću raspršenost uzorka rješenja što u ovom slučaju ne možemo smatrati dobrom implementacijom rješenja iz razloga što je velika vjerojatnost da svakim pokretanjem programa dobijemo drugačije rješenje. Dobivena rješenja za $\rho = 0.02$ pokazuju se kao slična što pokazuje da manja vrijednost parametra ρ daje dobro rješenje.

Ostalo nam je pokazati ovisnost rješenja o parametru β. Kao što je vidljivo iz tablice rezultata ovisnosti algoritma o paramterima, za veću vrijednost parametara β dobiva se lošije rješenje. Razlog tome je što parametar β utječe na pravilo odabira prilikom konstrukcije puta, na način da za veću vrijednost povećava vjerojatnost odabira najbližeg grada, odnosno tada kažemo da je algoritam pohlepniji. Pohlepnost algoritma za posljedicu ima stagnaciju rješenja, odnosno mravi će pritom slijediti uvijek isti put i konstruirati isto rješenje. Ponavljanje istog najboljeg rješenja u većini slučajeva vidljivo je u slijedećoj tablici:

Tablica 6. Prikaz najboljih rješenja za eksperiment ρ = 0.02 i β = 4

Redni broj eksperimenta	R_{best}	I_{tr}
1	837,79	887
2	838,11	735
3	837,79	765
4	837,79	963
5	840,98	530
6	868,78	250
7	846,21	780
8	838,11	284
9	838,11	713
10	838,11	463
11	846,21	746
12	870,46	950
13	852,15	734
14	856,35	770
15	837,79	832
16	850,94	836
17	838,11	610
18	846,21	981
19	837,79	853
20	838,11	871
21	861,72	759

Ovisnost implementiranog algoritma o parametru β pri davanju najboljeg rješenja također je prikazano i u slijedećem grafikonu:



Grafikon 3. Ovisnost najboljeg rješenja o parametru β

MMAS varijanta ima efikasan mehanizam kako bi se izbjegla stagnacija algoritma koja u ovoj osnovnoj verziji nije implementirana, a o njoj će biti više riječi u idućem poglavlju. Parametar β nije pokazao znatan utjecaj na iteraciju u kojoj je pronađeno najbolje rješenje.

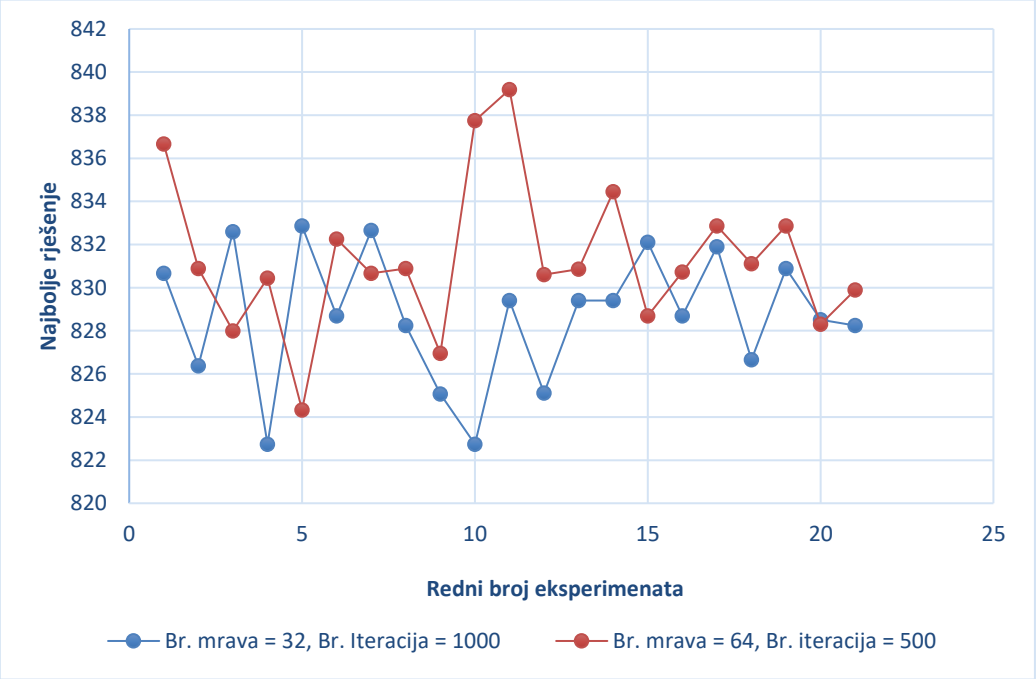
5.2. Ovisnost o broju mravi

Testiranjem ovisnosti implementacije o broju mravi prikazano je u slijedećoj tablici, poštujući pritom da je umnožak broja mravi i broja iteracija za testirane eksperimente jednak:

Tablica 7. Prikaz rezultata ovisnosti algoritma o broju mravi

	Broja mravi (k)	**Broj iteracija (i)**	$\mathbf{R_{best}}$	$\overline{\mathbf{R_{best}}}$	$\overline{\overline{\mathbf{R_{best}}}}$
1)	32	1000	822,74	828,7123	828,69
3)	64	500	824,33	831,35	830,86

Kao što možemo vidjeti iz tablice, kombinacija parametara s manjim brojem mrava i većim brojem iteracija daje malo bolje rješenje. Ova ovisnost uvelike ovisi o načinu na koji će biti implementiran MMAS, odnosno koji će mrav odlagati svoje feromonske tragove. U ovoj implementaciji, samo najbolji mrav odnosno globalni mrav odlaže svoje feromone i to na kraju iteracije. Tako da se u trenutnoj iteraciji, svi mravi (koliko god njih bilo) primjenjuju pravilo odabira s obzirom na istu feromonsku matricu, što može rezultirati sličnim turama. Ako bismo uzeli drugu varijantu, odnosno da najbolji mrav svake iteracije odlaže svoje feromonske tragove, tada bismo vjerojatno dobili veći raspon najboljih rješenja. S obzirom na prethodna istraživanja, možemo zaključiti da se veliki raspon najboljih rješenja u ovom slučaju ne smatra kao cjelokupno dobro rješenje algoritma. Ponekad, veća raspršenost uzorka može se smatrati i kao prednost, ali većinom se radi o načinu implementacije. [32]. Važno je napomenuti da druga varijanta implementacije nije testirana, nego samo teoretski zaključena.



Grafikon 4. Ovisnost najboljeg rješenja o parametru broja mrava

5.3. Vremenska složenost

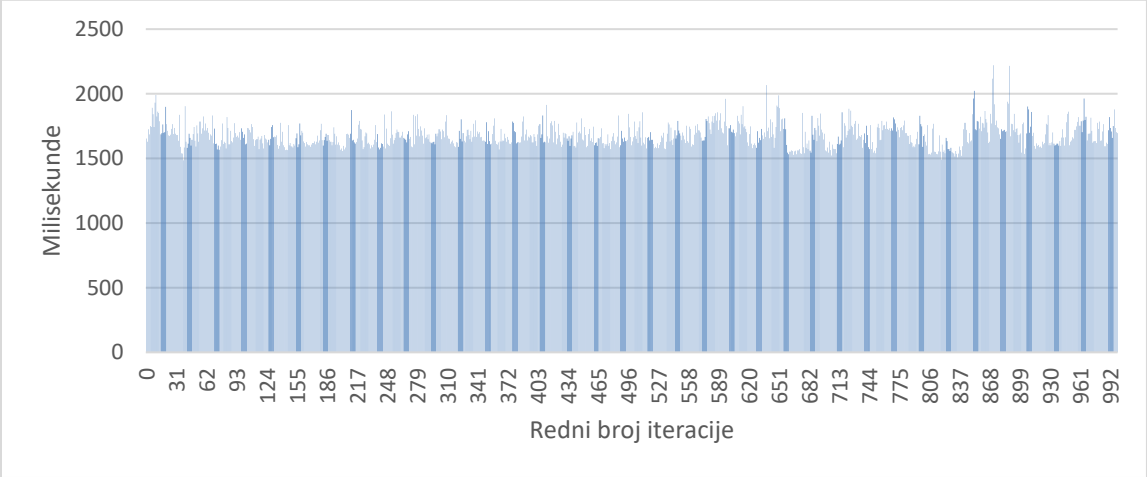
Vremenska složenost algoritma prikazana je slijedećom formulom [25]:

$$O(n_g \cdot (n_a \cdot N^2 + N^2),$$

gdje n_g predstavlja broj iteracija, n_a broj mrava u koloniji i N broj gradova koje treba poslužiti. Kao što možemo vidjeti iz formule, broj iteracije i broj mrava unutar kolonije prikazuju linearnu ovisnost, dok broj gradova kvadratnu ovisnost. Ažuriranje feromonskih tragova predstavljena je s zadnjom oznakom N^2.

Vremensko testiranje pojedine iteracije konstrukcije rješenja je izvršeno na temelju 1000 ponavljanja za 32 mrava, prema postavkama prve kombinacije parametara iz tablice 3. Kao rezultat, dobiveni su slijedeći podaci:

- Aritmetička sredina vremena pojedine iteracije= 1678,57 ms
- Medijan vremena pojedine iteracije = 1665 ms
- Najkraće vrijeme obavljanja iteracije = 1486 ms
- Najbolje vrijeme obavljanja iteracije = 2221 ms

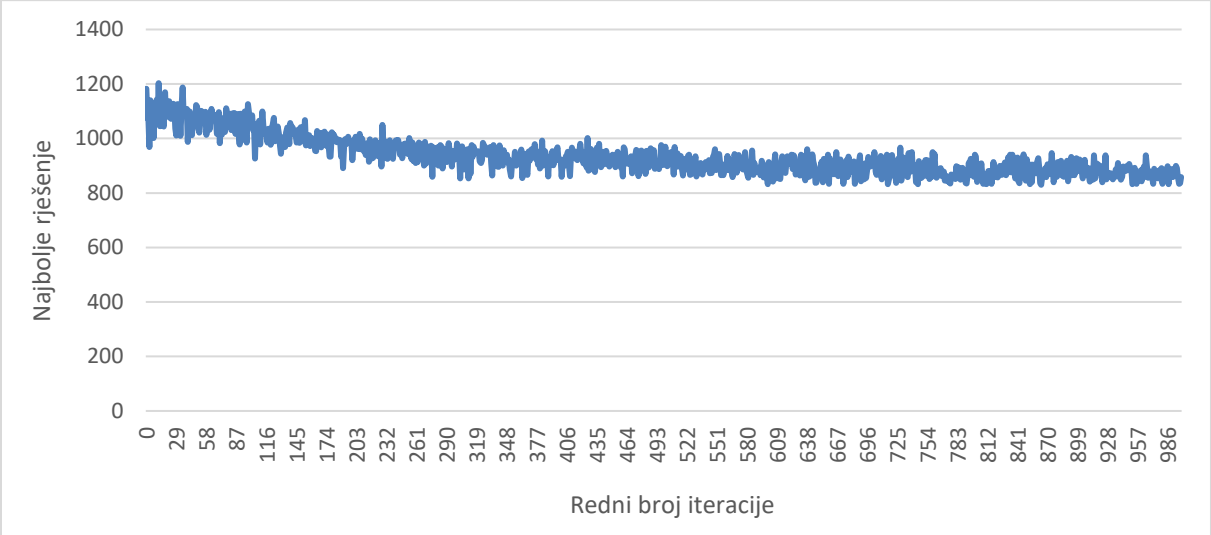


Grafikon 5. Prikaz vremenskog trajanja pojedine iteracije

Slijedno izvršavanje algoritma za 1000 iteracija i 32 mrava rezultira trajanju pronalaska optimalnog rješenja od otprilike 28 minuta, što itekako ovisi o brzini takta procesora. Ovo vrijeme smatra se poprilično dugačkim, ali ga je moguće lagano pretvoriti u paralelno izvršavanje, na način da konstrukcija rješenja pojedinog mrava predstavlja jednu dretvu. Takav način izvršavanja je moguće primijeniti iz razloga što pojedini mrav u koloniji konstruira put neovisno o drugim mravima.

6. Rasprava o rezultatima istraživanja

Kako je optimalno rješenje za ovu instancu problema 784, a najbolje rješenje dobiveno ovom implementacijom 821, postavlja se pitanje možemo li takvo rješenje smatrati kao dovoljno dobro? Odgovor se krije u idućem grafikonu, koji prikazuje konstruiranje najboljeg rješenja u pojedinoj iteraciji za kombinaciju postavki parametra 1) koji se unutar istraživanja pokazalo najboljim.



Grafikon 6. Prikaz R_{best} u pojedinoj iteraciji za kombinaciju parametara 1)

U grafu je vidljivo kako funkcija pronalaska najboljeg rješenja u pojedinoj iteraciji blago pada prema dole, odnosno s porastom broja iteracije pronalazi se bolje rješenje. Također se može reći kako algoritam uči k boljem rješenju. Prema tome, možemo zaključiti da je velika vjerojatnost u slučaju povećanja broja iteracija, doći će se još do boljeg rješenja. S toga je odgovor na postavljeno pitanje: DA, trenutno najbolje rješenje u iznosu od 821 se smatra dovoljno dobrim jer ima mogućnost napretka. Naravno, optimizacijom trenutnog algoritma može se doći puno prije do optimalnog rješenja. Kako bi se optimizacija ostvarila, donosi se nekoliko prijedloga za poboljšanje performansi algoritma:

1) Modificiranje postojećeg pseudokoda i pojačavanje utjecaja feromosnkih tragova

Prilikom implementacije rješenja prema uzoru na već pojašnjeni pseudokod algoritma iz članka [25], uočeno je moguće poboljšanje kod postavljanja ograničenja kapaciteta. Naime, nakon što je pravilom odluke određen idući grad koje će vozilo posjetiti, provjerava se je li kapacitet vozila veći od sumi trenutnog kapaciteta vozila i kapaciteta zahtjeva grada

(algoritam je implementiran za varijantu u kojoj vozilo vrši uslugu prikupljanja robe iz pojedinog grada). Ako je uvjet zadovoljen, posjećuje se odabrani grad, a u protivnom vozilo odlazi natrag u centralno skladište i uzima drugo vozilo. Moguće poboljšanje se upravo pronalazi u slučaju nezadovoljavanja uvjeta : umjesto da se vozilo odmah vraća u centralno skladište, treba provjeriti je li se neki od drugih mogućih gradova može posjetiti s trenutnim kapacitetom (prilikom provjere kreće se od najbližeg grada). Tako bi se mogao smanjiti ukupan prevaljeni jer novo vozilo koje bi krenulo iz skladišta ne bi trebalo putovati do navedenog vrha. Osim toga, ako na problem gledamo da i svako pojedino vozilo predstavlja trošak (npr. trošak amortizacije), a ne samo trošak u vidu puta između gradova, također ćemo biti na boljem putu pri optimalnom rješenju. Programski kod za koji je preporučeno poboljšanje:

```
            if(self.ant[k].antLoad + self.demands[j-
1] > self.ant[k].vehicleInUse.capacity):
                self.ReturnToDepot(k)
            else:
                self.ant[k].tour.append(j)
                self.ant[k].visited[j] = True
                self.ant[k].antLoad = self.ant[k].antLoad + self.demands[j-
1]
                self.ant[k].vehicleInUse.vehicleTour.append(j)
```

Sljedeći dio optimizacije postojećeg programskog koda odnosi se na proces odlaganja novih feromonskih tragova. U algoritmu, nove feromonske tragove odlaže samo najbolji mrav do sada, odnosno tzv. mrav s najboljim globalnim rješenjem. Kao što je objašnjeno u poglavlju MAX-MIN mravlji sustav, bolja implementacija bi bila dinamičko odlaganje feromonskih tragova. Tako naprimjer, nakon svake iteracije konstrukcije rješenja naizmjenice feromonske tragove ostavljaju najbolji mrav trenutne iteracije C^{ib} i najbolji mrav do sada C^{bs}. Količinu feromonskih tragova koje će pojedini mrav ostaviti definira se koeficijentima. U praksi se najčešće kreće s niskim koeficijentom za najboljeg mrava, koji se postepeno povećava, odnosno rješenje se određenom funkcijom usmjerava prema najboljem rješenju. Također se bilježi u koliko se iteracija nije pronašlo novo najbolje rješenje. Ova informacija kasnije služi kako bi se riješio problem stagnacije rješenja, na način ako broj iteracija u kojemu se nije pronašlo najbolje rješenje dosegne određenu vrijednost (npr. 100), tada se feromonska matrica restartira na svoju početnu vrijednost τ_0.

2) Uvođenjem lokalne optimizacije

O važnosti lokalne optimizacije govori i sama činjenica da je svrstana kao opcionalni dio unutar ACO metaheuristike. Nakon konstruiranja rješenja, postoje brojni algoritmi lokalne optimizacije koji trenutnu konstruiranu turu mrava poboljšaju na način da dodatno smanje ukupni trošak ture, proučavajući susjede svakog vrha rješenja.

3) Automatsko podešavanje najboljih parametara IRACE metodom

IRACE metoda podrazumijeva *irace* paket koji služi za automatsko generiranje konfiguracije parametara na temelju instance problema i vrijednosti maksimalnog vremena izvođenja. Takva konfiguracija omogućuje najbolje rješenje, odnosno najnižu cijenu kompletnog puta ako gledamo sa stajališta CVRP. Najčešće se koristi za moderne optimizacijske algoritme koji obično zahtijevaju veliki broj parametara za poboljšanje njihovih performansi. [33]

```
# Best configurations (first number is the configuration ID)
    algorithm  ls alpha beta  rho ants nnls   q0 dlb rank eants
530       acs   3  1.93 8.27 0.36   21   10 0.78   1  NA   NA
635       acs   3  2.52 9.69 0.29   42   11 0.83   1  NA   NA
552       acs   3  1.81 7.86 0.25   20   16 0.72   1  NA   NA
741       acs   3  1.78 5.74 0.23   31   13 0.64   1  NA   NA
700       acs   3  1.85 5.97 0.23   34   12 0.68   1  NA   NA
# Best configurations as commandlines (first number is the configuration ID)
530  --acs --localsearch 3 --alpha 1.93 --beta 8.27 --rho  0.36 \
     --ants 21 --nnls 10 --q0 0.78 --dlb 1
635  --acs --localsearch 3 --alpha 2.52 --beta 9.69 --rho  0.29 \
     --ants 42 --nnls 11 --q0 0.83 --dlb 1
552  --acs --localsearch 3 --alpha 1.81 --beta 7.86 --rho  0.25 \
     --ants 20 --nnls 16 --q0 0.72 --dlb 1
741  --acs --localsearch 3 --alpha 1.78 --beta 5.74 --rho  0.23 \
     --ants 31 --nnls 13 --q0 0.64 --dlb 1
700  --acs --localsearch 3 --alpha 1.85 --beta 5.97 --rho  0.23 \
     --ants 34 --nnls 12 --q0 0.68 --dlb 1
```

Slika 17. Primjer generiranja konfiguracije IRACE metodom za TSP [33]

4) Korištenjem naprednijih varijanti algoritama ACO

Teorijska analiza MMAS-a dovela je do stvaranja njezine bolje varijante ACO nazvane mravlji sustav tri granice (eng. *Three Bound Ant System*, skraćeno TBAS), koja je manje vremenske složenosti od prethodne varijante te poboljšava performanse konstruiranih rješenja. Glavna karakteristike koje razlikuju TBAS s obzirom na prethodne varijante [34]:

- Sastoji se od tri granice (donje granice τ_{LB}, gornje granice τ_{UB} i granice skraćivanja $\tau_{CB} = \omega \cdot \tau_{LB}$
- Umjesto isparavanja feromosnkih tragova svake iteracije, ponekad dolazi do skraćivanja feromona

- Inicijalna vrijednost feromonskih tragova jednaka je donjoj granici
- Koristi se dodatan parametar ω za koji vrijedi $\frac{\tau_{LB}}{\tau_{CB}} \leq \omega \leq 1$

Ažuriranje feromonskih tragova započinje funkcijom odlaganja feromona nakon koje slijedi metoda skraćivanja feromona, koja se poziva samo u slučaju ako je jedan od feromonskih tragova veći od gornje granice τ_{UB}. Odlaganje feromona odvija se prema formuli slijedećoj formuli:

$$\tau_c = \tau_c + \frac{Q_i}{f(s^{bs})}, \forall c \in s^{bs} \ (odabrana\ komponenta, npr. najbolji\ mrav\ do\ sada),$$

gdje je Q_i za pravu iteraciju jednak $Q_0 = 1$, a za svaku ostalu se računa prema formuli $Q_{i+1} = \frac{Q_i}{(1-\rho)}$. Skraćivanje feromona se odvija na način da se množi svaki feromonski trag i Q_i s $\omega' = \omega \frac{\tau_{UB}}{\tau_{max}}$ te prisiljava svaki feromonski trag τ_c da budu unutar intervala $[\tau_{LB}, \tau_{CB}]$. Upravo zbog različitog načina ažuriranja feromona, TBAS ima manju vremensku složenost od MMAS-a te može biti brži čak i do 20% . [34]

O tome kakvo će rješenje dati implementacija uvelike ovisi o znanju kombinatorke i iskustvu u području optimizacije. Ovdje su navedene samo neke od brojnih metoda koji mogu dovesti do optimalnog rješenja.

7. Zaključak

U ovome radu obrađena je tema problema usmjeravanja vozila, kao jednog od najpopularnijih problema u kombinatornoj optimizaciji. Kako je ovaj problem poprilično složen, pripada skupini NP-teških problema. Idealno rješenje ovakvih problema nije poznato pa se iz toga razloga njegovom rješavanju pristupa pomoću različitih egzaktnih i heurističkih metoda. U današnjem svijetu gdje je globalizacija dosegla svoj vrhunac, izazovi u transportu i logistici postali su sve kompleksniji. Dodatni izazovi pojavili su se i za vrijeme COVID-19 pandemije, u kojemu je online kupovina postali primarni dio života većine ljude. Dostavne službe počeli su se susretati s prevelikom potražnjom, što je predstavilo dodatan problem koji je utjecao na kvalitetu njihove usluge. Upravo su ovi izazovi potaknuli još veća istraživanja VRP-a, kako bi se optimizirale rute u logistici u vidu smanjenja cijena i vremena, što na kraju rezultira poboljšanju usluge. S obzirom da je danas transport širok pojam, tako su nastale i nekoliko najpoznatijih varijanti VRP-a, čija kombinacija se može primijeniti na široki spektar specifičnih problema.

Od raznih pristupa rješavanju VRP-a, odabran i je metahuristički pristup optimizacije kolonijom mrava, koji se pokazao uspješan u rješavanju NP-teških problema. Na temelju njegove varijante MAX-MIN mravlji sustav, koji se pokazao uspješan za rješavanje problema s velikim brojem vrhova i dovoljnim brojem iteracija, implementirano je rješenje za VRP s ograničenjem kapaciteta. Implementacija je napravljena u programskom jeziku *Python*, koristeći neke od ugrađenih biblioteka poput *Numpy*. Kako je CVRP osnovni oblik, ovaj algoritam može poslužiti kao temelj za izgradnju ostalih varijanti problema. Kao motivacija za implementaciju, poslužio je pseudokod iz izvora [25].

Rješenje dobiveno implementacijom nije optimalno, ali ga i dalje možemo smatrati dovoljno dobrim rješenjem iz razloga što s većim brojem iteracija se dobivaju još bolji rezultati. Koliko će rješenje biti dobro, uvelike ovisi o iskustvu i znanju optimizacijske kombinatorike. Kako se ovaj rad više fokusirao na rješavanje VRP-a i njegovu obradu, napravljeni algoritam nije do kraja optimiziran te služi kao temelj za daljnje razvijanje k boljem rješenju. Na temelju dobivene statističke analize na određenoj instanci problema, izvedeni su zaključi o mogućem poboljšanju algoritma. Tako je vidljivo da s podešavanjem parametara, modifikacijom ažuriranja feromonske matrice i uvođenjem lokalne optimizacije možemo dobiti još bolje rješenje. Idealno rješenje kao zaključak istraživanju ovoga rada moguće je dobiti implementacijom mravljeg sustava s tri granice, najnovije varijante ACO, u kombinaciji s dodatnim metodama lokalne optimizacije.

Popis literature

[1] „Transport | European Union“. https://europa.eu/european-union/topics/transport_en (pristupljeno kol. 30, 2021).

[2] G. Hasle, K.-A. Lie, i E. Quak, *Geometric Modelling, Numerical Simulation, and Optimization: Applied Mathematics at SINTEF*. Berlin Heidelberg: Springer e-books, 2007.

[3] K. Braekers, K. Ramaekers, i I. Van Nieuwenhuyse, „The vehicle routing problem: State of the art classification and review“, *Comput. Ind. Eng.*, sv. 99, str. 300–313, ruj. 2016, doi: 10.1016/j.cie.2015.12.007.

[4] B. Eksioglu, A. V. Vural, i A. Reisman, „The vehicle routing problem: A taxonomic review“, *Comput. Ind. Eng.*, sv. 57, izd. 4, str. 1472–1483, stu. 2009, doi: 10.1016/j.cie.2009.05.009.

[5] P. Toth i D. Vigo, Ur., *The vehicle routing problem*. Philadelphia, Pa: Society for Industrial and Applied Mathematics, 2002.

[6] L. Kovács, A. Agárdi, i T. Bányai, „Fitness Landscape Analysis and Edge Weighting-Based Optimization of Vehicle Routing Problems“, *Processes*, sv. 8, izd. 11, str. 1363, lis. 2020, doi: 10.3390/pr8111363.

[7] C. Barnhart i G. Laporte, *Handbooks in Operations Research & Management Science - Transportation, Volume 14.* Burlington: Elsevier, 2006. Pristupljeno: ruj. 02, 2021. [Na internetu]. Dostupno na: http://www.123library.org/book_details/?id=36171

[8] Mr. sc. Tonči Carić, „Unapređenje organizacije transporta primjenom heurističkih metoda“, str. 123, 2004.

[9] A. Silva-Gálvez, E. Lara-Cárdenas, I. Amaya, J. M. Cruz-Duarte, i J. C. Ortiz-Bayliss, „A Preliminary Study on Score-Based Hyper-heuristics for Solving the Bin Packing Problem“, u *Pattern Recognition*, sv. 12088, K. M. Figueroa Mora, J. Anzurez Marín, J. Cerda, J. A. Carrasco-Ochoa, J. F. Martínez-Trinidad, i J. A. Olvera-López, Ur. Cham: Springer International Publishing, 2020, str. 318–327. doi: 10.1007/978-3-030-49076-8_30.

[10] A. Konstantinidis, S. Pericleous, i C. Charalambous, „Adaptive Evolutionary Algorithm for a Multi-Objective VRP“, str. 14.

[11] S. Salhi, N. Wassan, i M. Hajarat, „The Fleet Size and Mix Vehicle Routing Problem with Backhauls: Formulation and Set Partitioning-based Heuristics“, *Transp. Res. Part E Logist. Transp. Rev.*, sv. 56, str. 22–35, ruj. 2013, doi: 10.1016/j.tre.2013.05.005.

[12] N. A. Wassan, „Meta-Heuristics for the Multiple Trip Vehicle Routing Problem with Backhauls“, str. 262.

[13] G. Martinovic, I. Aleksi, i A. Baumgartner, „Single-Commodity Vehicle Routing Problem with Pickup and Delivery Service“, *Math. Probl. Eng.*, sv. 2008, str. 1–17, 2008, doi: 10.1155/2008/697981.

[14] N. Mahmud i Md. M. Haque, „Solving Multiple Depot Vehicle Routing Problem (MDVRP) using Genetic Algorithm“, u *2019 International Conference on Electrical, Computer and Communication Engineering (ECCE)*, Cox'sBazar, Bangladesh, velj. 2019, str. 1–6. doi: 10.1109/ECACE.2019.8679429.

[15] F. Li, B. Golden, i E. Wasil, „The open vehicle routing problem: Algorithms, large-scale test problems, and computational results“, *Comput. Oper. Res.*, sv. 34, izd. 10, str. 2918–2930, lis. 2007, doi: 10.1016/j.cor.2005.11.018.

[16] A. Chbichib, R. Mellouli, i H. Chabchoub, „Profitable Vehicle Routing Problem with Multiple Trips: Modeling and constructive heuristics“, u *2011 4th International Conference on Logistics*, Hammamet, Tunisia, svi. 2011, str. 500–507. doi: 10.1109/LOGISTIQUA.2011.5939450.

[17] T. Erdelić i T. Carić, „A Survey on the Electric Vehicle Routing Problem: Variants and Solution Approaches“, *J. Adv. Transp.*, sv. 2019, str. 1–48, svi. 2019, doi: 10.1155/2019/5075671.

[18] M. Dorigo, „Ant colony optimization“, *Scholarpedia*, sv. 2, izd. 3, str. 1461, ožu. 2007, doi: 10.4249/scholarpedia.1461.

[19] E. Bonabeau, M. Dorigo, i G. Theraulaz, *Swarm intelligence: from natural to artificial isystems*. New York: Oxford University Press, 1999.

[20] Z. Tang, M. Sonntag, i H. Gross, „Ant colony optimization in lens design“, *Appl. Opt.*, sv. 58, izd. 23, str. 6357–6364, kol. 2019, doi: 10.1364/AO.58.006357.

[21] M. Dorigo, M. Birattari, i T. Stutzle, „Ant colony optimization“, *IEEE Comput. Intell. Mag.*, sv. 1, izd. 4, str. 28–39, stu. 2006, doi: 10.1109/MCI.2006.329691.

[22] N. Ivkovic, M. Malekovic, i M. Golub, „Extended Trail Reinforcement Strategies for Ant Colony Optimization“, u *Swarm, Evolutionary, and Memetic Computing*, sv. 7076, B. K. Panigrahi, P. N. Suganthan, S. Das, i S. C. Satapathy, Ur. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, str. 662–669. doi: 10.1007/978-3-642-27172-4_78.

[23] S. A. Adubi i S. Misra, „A comparative study on the ant colony optimization algorithms“, u *2014 11th International Conference on Electronics, Computer and Computation (ICECCO)*, Abuja, Nigeria, ruj. 2014, str. 1–4. doi: 10.1109/ICECCO.2014.6997567.

[24] Thomas Stützle i Holger H. Hoos, „MAX–MIN Ant System“, u *Future Generation Computer Systems*, sv. 16, 8, 2000, str. 889–914. [Na internetu]. Dostupno na: https://doi.org/10.1016/S0167-739X(00)00043-1

[25] P. Stodola, J. Mazal, M. Podhorec, i O. Litvaj, „Using the Ant Colony Optimization algorithm for the Capacitated Vehicle Routing Problem“, u *Proceedings of the 16th*

International Conference on Mechatronics - Mechatronika 2014, Brno, Czech Republic, pros. 2014, str. 503–510. doi: 10.1109/MECHATRONIKA.2014.7018311.

[26] „numpy.argsort — NumPy v1.21 Manual“. https://numpy.org/doc/stable/reference/generated/numpy.argsort.html (pristupljeno ruj. 07, 2021).

[27] M. Dorigo i T. Stützle, *Ant colony optimization*. Cambridge, Mass: MIT Press, 2004.

[28] „ACO: Public Software“. http://www.aco-metaheuristic.org/aco-code/ (pristupljeno ruj. 07, 2021).

[29] A. Lipowski i D. Lipowska, „Roulette-wheel selection via stochastic acceptance“, *Phys. Stat. Mech. Its Appl.*, sv. 391, izd. 6, str. 2193–2196, ožu. 2012, doi: 10.1016/j.physa.2011.12.004.

[30] „Augerat 1995 - Set A - VRP-REP: the vehicle routing problem repository“. http://www.vrp-rep.org/datasets/item/2014-0000.html (pristupljeno ruj. 08, 2021).

[31] „CVRPLIB - Plotted Instances“. http://vrp.atd-lab.inf.puc-rio.br/index.php/en/plotted-instances?data=A-n32-k5 (pristupljeno ruj. 08, 2021).

[32] N. Ivkovic, D. Jakobovic, M. Golub, "Measuring Performance of Optimization Algorithms in Evolutionary Computation," *International Journal of Machine Learning and Computing* vol. 6, no. 3, pp. 167-171, 2016. DOI: 10.18178/ijmlc.2016.6.3.593

[33] M. López-Ibáñez, J. Dubois-Lacoste, L. Pérez Cáceres, M. Birattari, i T. Stützle, „The irace package: Iterated racing for automatic algorithm configuration“, *Oper. Res. Perspect.*, sv. 3, str. 43–58, 2016, doi: 10.1016/j.orp.2016.09.002.

[34] N. Ivkovic i M. Golub, „A New Ant Colony Optimization Algorithm: Three Bound Ant System“, 2014, str. 280–281.

Popis slika

Popis tablica

Popis grafikona