Implementacija algoritma optimizacije kolonijom mrava u Python programskom jeziku

Platužić, Josip

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac Z1P - Obrazac za ocjenu završnog rada na preddiplomskom sveučilišnom studiju

Osijek, 16.09.2021.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada na preddiplomskom sveučilišnom studiju

Ime i prezime studenta:	Josip Platužić
Studij, smjer:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R4118, 28.07.2017.
OIB studenta:	10674426797
Mentor:	Izv. prof. dr. sc. Emmanuel Karlo Nyarko
Sumentor:	
Sumentor iz tvrtke:	
Naslov završnog rada:	Implementacija algoritma optimizacije kolonijom mrava u python programskom jeziku
Znanstvena grana rada:	**Umjetna inteligencija (zn. polje računarstvo)**
Predložena ocjena završnog rada:	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	16.09.2021.
Datum potvrde ocjene Odbora:	22.09.2021.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

IZJAVA O ORIGINALNOSTI RADA

Osijek, 25.09.2021.

Ime i prezime studenta:	Josip Platužić
Studij:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R4118, 28.07.2017.
Turnitin podudaranje [%]:	4

Ovom izjavom izjavljujem da je rad pod nazivom**: Implementacija algoritma optimizacije kolonijom mrava u python programskom jeziku**

izrađen pod vodstvom mentora Izv. prof. dr. sc. Emmanuel Karlo Nyarko

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

IZJAVA

o odobrenju za pohranu i objavu ocjenskog rada

kojom ja __Josip Platužić_____, OIB: __10674426797_____, student/ica Fakulteta elektrotehnike, računarstva i informacijskih tehnologija Osijek na studiju _Preddiplomski sveučilišni studij Računarstvo _, kao autor/ica ocjenskog rada pod naslovom: _Implementacija algoritma optimizacije kolonijom mrava u python programskom jeziku**,**

dajem odobrenje da se, bez naknade, trajno pohrani moj ocjenski rad u javno dostupnom digitalnom repozitoriju ustanove Fakulteta elektrotehnike, računarstva i informacijskih tehnologija Osijek i Sveučilišta te u javnoj internetskoj bazi radova Nacionalne i sveučilišne knjižnice u Zagrebu, sukladno obvezi iz odredbe članka 83. stavka 11. *Zakona o znanstvenoj djelatnosti i visokom obrazovanju* (NN 123/03, 198/03, 105/04, 174/04, 02/07, 46/07, 45/09, 63/11, 94/13, 139/13, 101/14, 60/15).
Potvrđujem da je za pohranu dostavljena završna verzija obranjenog i dovršenog ocjenskog rada. Ovom izjavom, kao autor/ica ocjenskog rada dajem odobrenje i da se moj ocjenski rad, bez naknade, trajno javno objavi i besplatno učini dostupnim:

a) široj javnosti
b) studentima/icama i djelatnicima/ama ustanove
c) široj javnosti, ali nakon proteka 6 / 12 / 24 mjeseci (zaokružite odgovarajući broj mjeseci).

**U slučaju potrebe dodatnog ograničavanja pristupa Vašem ocjenskom radu, podnosi se obrazloženi zahtjev nadležnom tijelu Ustanove.*

__Osijek, 25.09.2021._____ _____

(mjesto i datum) (vlastoručni potpis studenta/ice)

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

Implementacija algoritma optimizacije kolonijom mrava u Python programskom jeziku

Završni rad

Josip Platužić

Osijek, 2021.

SADRŽAJ

1. UVOD

Optimizacija kolonijom mrava (eng. Ant Colony Optimization - ACO) je probabilistička metoda rješavanja težih, kompliciranih i inače vremenski zahtjevnih problema koji se u većini slučajeva baziraju na pronalaženje optimalnih puteva kroz težinske grafove. Sam algoritam inspiriran je na ponašanju stvarnih bioloških mrava. ACO prvi je opisao Marco Dorigo, 90-ih u svom doktorskom radu [1]. U početku, ACO je bio upotrijebljen za dobro poznat problem trgovačkog putnika (eng. Traveling Salesman Problem - TSP), te se s vremenom počeo primjenjivati za rješavanje težih problema optimizacije.

U drugom poglavlju su opisani aktualni znanstveni radovi u području rada koji se obrađuje.

U trećem poglavlju je razjašnjena inspiracija za ACO i sličnost algoritma optimizacije kolonijom mrava te stvarne kolonije mrava. Detaljnije su objašnjeni temeljni dijelovi ACO algoritma. Objašnjene su komponente umjetnih mrava koji predstavljaju agente u pronalaženju optimalnih rješenja.

U četvrtom poglavlju detaljnije je objašnjena implementacija algoritma, te parametri algoritma korištene u rješavanju problema

U petom poglavlju detaljnije se ulazi u pojedine probleme. Problem trgovačkog putnika ili TSP, te problem usmjeravanja vozila s ograničenim kapacitetom (eng. Capacitated Vehicle Routing Problem - CVRP). Te su za pojedini problem ispitane kombinacije parametara i dobivena rješenja.

1.1. Zadatak završnog rada

Zadatak završnog rada je u programskom jeziku python implementirati algoritam optimizacije kolonijom mrava, obraditi probleme TSP i CVRP, te analizirati dobivena rješenja.

2. PREGLED PODRUČJA TEME

U nastavku su nabrojeni i ukratko opisani znanstveni radovi u području rada koji se obrađuje.

U [2], istražena je paralelna implementacija algoritma Max-Min mravljeg sustava (eng. Ant Colony System - MMAS) visokih performansi. Paralelizacija predstavlja mogućnost upotrebe više međusobno umreženih računala u izvođenju velikog broja metaheurističkih algoritama u paraleli. Testirane su predložene implementacije algoritma u odnosu na postojeće serijske i paralelne algoritme.

U [3], analiziran je sustav raspoređivanja proizvodnje dodjeljivanjem zadataka pripadajućim resursima, pritom raspoređivajući ih što učinkovitije koristeći metaheuristiku. U radu je prikazan i analiziran razvoj sustava za planiranje kod postojećih prodajnih mjesta koja koriste optimizaciju kolonije mrava u problemu raspoređivanja proizvodnje.

U [4], predstavljen je pristup traženja optimalnih puteva u hipergrafu na temelju optimizacije kolonijom mrava. Usmjereni hipergrafovi su proširenja usmjerenih grafova koji obuhvaćaju složene odnose u mrežnim strukturama. U radu se predstavlja eksperimentalna procjena algoritma pomoću umjetno generiranih hipergrafova.

U [5], istraživanje se usredotočilo na implementaciju sustava optimizacije kolonije mrava za rješavanje napredne verzije problema određivanja rute vozila (eng. Vehicle routing problem – VRP), nazvane sustav upravljanja voznim parkom (eng. Fleet Management System - FMS). Razmotrene su potrebne izmjene u postojećim algoritmima FMS-a kao odgovor na brzi razvoj električnih vozila. Potrebne izmjene predstavljaju ograničeni domet električnih vozila, njihovo vrijeme punjenja i trošenje baterija.

U [6], predložena je napredna verzija optimizacije kolonijom mrava zvana križana optimizacija kolonijom mrava (eng- Crisscross Ant Colony Optimization-CCACO) koja se temelji na algoritmu optimizacije mrava za kontinuirane domene (eng. Ant Colony Optimization for Continous Domains-ACOR). Uvedena su poboljšanja kao što su vertikalno i horizontalno pretraživanje i poboljšan mehanizam izvornog ACO-ra kako bi se formirao poboljšani algoritam CCACO. CCACO se u radu primjenjuje na segmentaciji slike s više pragova na temelju 2D histograma.

3. OPIS ALGORITMA

ACO je populacijski orijentirana metaheuristika koja se primjenjuje za rješavanje problema kombinatorne optimizacije, gdje umjetni mravi zajedno surađuju da bi stvorili najbolje, to jest najkraće moguće rješenje u težinskom grafu. Za pronalazak optimalnog rješenja se koriste elementi kao što su feromoni, što su zapravo promjenjive karakteristike grafa koje su povezane s komponentama grafa te utječu na kretanje umjetnih mrava u grafu. Isti ti mravi su odgovorni za promjenu te karakteristike. U literaturi su zabilježene mnoge različite varijante osnovnog ACO načela [2-7]. Glavno svojstvo ACO algoritma je proces pronalaženja rješenja koji se svodi na iterativno mijenjanje određenih elemenata ovisno o prethodnim rezultatima kako bi se postiglo sve bolje i bolje rješenje.

Način definiranja komponenti algoritma specifičan je za pojedine probleme i može se osmisliti na različite načine, suočavajući se s specifičnosti informacija koje se primjenjuju za uvjetovanje i ograničenja koja je potrebno zadati da bi se pridonijelo pojavi dobrih rješenja.

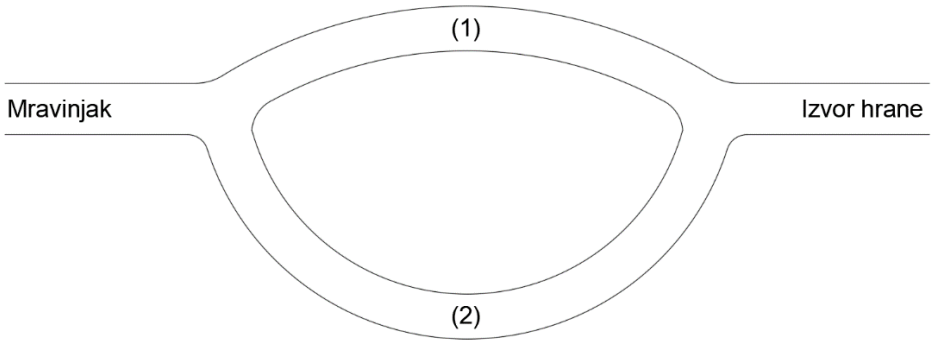
U ovome poglavlju je objašnjen izvor inspiracije iz prirode, detaljnije objašnjena ACO struktura , komponente i parametri umjetnih mrava koji su temeljan i neophodan dio ACO sustava. [7]

3.1. Inspiracija iz prirode

Algoritamski svijet je svestran s raznim strategijama i alatima koji se stalno razvijaju kako bi se zadovoljila potreba za rješenjima inače nerješivih problema konvencionalnim sredstvima. Kada su algoritmi inspirirani prirodnim zakonima, opažaju se zanimljivi rezultati. Evolucijski algoritmi su osmišljeni tako da oponašaju određena ponašanja kao i evolucijske osobine ljudskog genoma.

Takav algoritamski dizajn nije ograničen samo na ljude, već se može nadahnuti i prirodnim ponašanjem određenih životinja. Različite tehnike optimizacije su se razvile na temelju takvih evolucijskih algoritama i tako otvorile domenu metaheuristike. ACO je jedan od primjera metaheurističkih metoda. Inspiriran je hranjenjem pravih mrava, gdje mravi nasumično istražuju okolinu mravinjaka te kada nađu hranu, vraćaju se u mravinjak te tijekom istraživanja ostavljaju trag feromona, točnije kemijski trag koji mogu osjetiti i drugi mravi. Mravi slijede različite puteve do izvora hrane i natrag. Opaženo je da uzastopnim prolazima po putu, povećava se razina feromona na istom, te zato što se najkraći put najviše puta obiđe na njemu ostaje najveća razina feromona. Ako pogledamo sliku 3.1. koja predstavlja put od mravinjaka do izvora hrana i pretpostavimo da je put (1) dvostruko kraći nego put (2) te ako prvi mrav odabere put (2), to jest

duži put, mravi nakon njega će naići na trag feromona prethodnog mrava, te većina će krenuti tim putem, ali neki će odlučiti uzeti drugi put, to jest kraći put.



Sl. 3.1. Putevi različitih duljina

Iako će trenutni originalni put (2) biti pogodniji mravima zbog razine feromona, drugi kraći put će se krenuti brže razvijati, zato što ako je za duži put potrebno *x* minuta da mrav vrati hranu u mravinjak, mrav na kraćem putu će isto napraviti za *x*/2 minuta, što znači da će ostaviti dvostruko više feromona na svome putu u istom vremenu. Zbog toga, kraći i optimalniji put će uskoro poprimiti više feromona, dok će se duži put koristi manje i manje te će feromoni tijekom vremena nestajati.

Izbor puta je stoga nasumičan, iako na njega snažno utječe intenzitet feromona, ipak omogućuje slučajno odstupanje od trenutno najbolje rješenja, što vodi pronalasku novog najboljeg rješenja.

3.2. Dijelovi algoritma

ACO je prvotno bio osmišljen za rješavanje problema trgovačkog putnika (eng. Traveling Salesman Problem – TSP), ali taj algoritam nije bio bitno bolji od drugih metaheurističkih algoritama. Ipak, ACO je vrlo koristan u rješavanju kompleksnijih problema kao na primjer problem određivanja rute vozila s ograničenim kapacitetom (eng. Capacitated Vehicle Routing Problem – CVRP), koji je detaljnije objašnjen u nadolazećim poglavljima.

Većina varijanti ACO algoritama, zajedno dijeli više aktivnosti, to jest postupaka koji zajedno tvore cjelinu i temelj ACO algoritma. Te aktivnosti predstavljaju konstrukcija puta te ažuriranje feromona.

3.2.1. Izgradnja puta

Mravi su implementirani kao skup asinkronih agenata. Oni grade rješenje posjećujući niz vrhova na težinskom grafu. Odabiru put do vrhova preko bridova grafa ovisno o dva parametara, privlačnosti i udaljenosti. Kao i pravi mravi, umjetni mravi će u većini slučajeva preferirati put s najjačom koncentracijom feromona. Ipak, izbor se odlučuje slučajno pa umjetni mravi mogu odabrati i druga rješenja, što vodi traženju boljih to jest kraćih puteva.

Slučajna vrijednost pomaka s vrha *i* na vrh *j*, η_{ij}, ima vrijednost koja je obrnuto proporcionalna udaljenosti dvaju vrhova, to jest dužinu brida koji spaja vrh *i* i vrh *j*, (izraz 3.1.):

$$\eta ij = \frac{1}{dij} \tag{3.1.}$$

Trenutna razina traga feromona, τ_{ij} , predstavlja indikaciju kvalitete grane, to jest puta. Ako umjetni mrav prepozna snažan trag feromona koji vodi do vrha, zna da je to obećavajući smjer za istraživanje. Kada umjetni mrav odabere svoju rutu i dovrši konstrukciju puta ažurira se razina feromona i onda tek kreće novi mrav. Informacije o feromonima se ažuriraju prema pravilima koje će se objasniti u sljedećem potpoglavlju.

Odabir sljedećeg vrha u putu se bazira na vjerojatnosti, a vjerojatnost obilaska određenog vrha se određuje prema sljedećoj formuli:

$$p_{ij} = \frac{[\tau_{ij}]^{\alpha} * [\eta_{ij}]^{\beta}}{\sum_{h \epsilon \Omega} [\tau_{ij}]^{\alpha} * [\eta_{ij}]^{\beta}} \tag{3.2.}$$

gdje vrijednosti α i β predstavljaju parametre koji određuju utjecaj slučajne vrijednosti η_{ij} i razine feromona τ_{ij}. Ako je $\alpha >> \beta$,onda dolazi do stagnacije. Svaki mravi će birati isti put, to jest put sa najviše feromona, što neće voditi do eventualnog poboljšanja puta, nego obično vodi lošim rješenjima. Slično, ako je $\beta >> \alpha$,onda će razina feromona na granama biti zanemariva, te u većini slučajeva rješenja neće biti kvalitetna. Skup Ω predstavlja skup svih vrhova koji zadovoljavaju zadane uvjete, ovisne o specifičnom problemu.

3.2.2. Ažuriranje feromona

Nakon što pojedini mravi izgrade svoja rješenja, ažurira se stanje feromona na grafu, točnije na bridovima. Ažuriranje feromona se dijeli na tri procesa: lokalno i globalno ažuriranje, te

isparavanje feromona. Količina feromona koji se ažurira, to jest formule koje se koriste nisu uvijek jednake. Potreba za većim, odnosno manjim utjecajem ažuriranja feromona ovisi o kompleksnosti i tipu problema. Izrazi zadani u nastavku eksperimentalno su ispitani i pokazali su se učinkoviti za probleme koji će se riješiti u ovom radu.

Lokalno ažuriranje feromona

Dok pojedini mrav iz kolonije traži rješenje, njegov put se sprema u memoriju i nakon što izgradi put u grafu, obavlja se lokalno ažuriranje feromona. To ažuriranje predstavlja povećanje razine feromona na bridovima koji su dio izgrađenog puta. Ažuriranje se vrši po sljedećoj formuli:

$$\tau_{ij} = \tau_{ij} + \rho * \frac{1}{n * J_{nn}} \tag{3.3.}$$

gdje je $0 < \rho \leq 1$ stopa isparavanje feromona. Taj se parametar upotrebljava za kontrolu iznosa povećavanja i isparavanja feromona, to jest koristi se da se vrijednost feromona ne bi neograničeno povećavala ili smanjivala. Parametar *n* predstavlja broj posjećenih vrhova u izgrađenom rješenju, a J_{nn} predstavlja duljinu izgrađenog puta, to jest prijeđeni put. Kao što vidimo količina dodanih feromona je obrnuto proporcionalna duljini prijeđenog puta, što znači da će razina dodatnih feromona biti veća što je put kraći.

Globalno ažuriranje feromona

Tek kada svaki mrav iz kolonije izgradi svoje rješenje, to jest nakon svake iteracije, izvodi se globalno ažuriranje feromona te primjenjuje se samo na trenutno najboljem dobivenom rješenju. Globalno ažuriranje ostavlja veći trag feromona nego lokalno, sa ciljem da se buduća rješenja orijentiraju više na susjedno područje trenutno najboljeg puta. Primjenjuje se po formuli:

$$\tau_{i^*j^*} = \tau_{ij} + \rho * \frac{1}{J^*} \qquad \forall (i^* j^*) \, \epsilon \, \psi^* \tag{3.4.}$$

gdje $\tau_{i^*j^*}$ predstavlja stanje feromona na bridovima koji su dio najboljeg globalnog rješenja, a J^* predstavlja prijeđenu udaljenost istog najboljeg globalnog rješenja.

Isparavanje feromona.

Kao i globalno ažuriranje feromona, isparavanje feromona se vrši nakon svake iteracije, to jest kada su svi mravi izgradili svoja rješenja. Intenzitet isparavanja feromona ovisi o vrijednosti ρ za koju je bitno da se vrijednost ne postavi premala ili previsoka. Ako je vrijednost $\rho \approx 1$,razina feromona će nekontrolirano opadati. Ako je vrijednost $\rho \approx 0$, isparavanje će biti zanemarivo i razina feromona će nekontrolirano rasti, to jest algoritam neće moći zaboraviti loše odabrane puteve.

$$\tau_{ij} = (1 - \rho) * \tau_{ij} \tag{3.5.}$$

4. IMPLEMENTACIJA ALGORITMA

U ovom su poglavlju detaljnije objašnjeni pojedini koraci i implementacija ACO algoritma u programskom jeziku python. U nastavku će se obrađivati problemi TSP i CVRP, koji dijele zajedničke aktivnosti i korake koji predstavljaju kostur ACO algoritma.

U algoritmu mravi i bridovi su implementirani klasom *Ant* i klasom *Edge*, gdje klasa *Edge* predstavlja brid u grafu, i kao atribute ima duljinu brida i trenutnu razinu feromona na bridu. Klasa *Ant* predstavlja umjetnog mrava koji kao atribute ima potrebne podatke za izgradnju puta, kao što su:

- **Matrica udaljenosti** - koja predstavlja dvodimenzionalni niz koji, kao elemente, sadrži objekte klase *Edge*, to jest sadrži sve bridove koji povezuju vrhove u grafu. Ako postoji N vrhova, ova matrica će imati veličinu N*N. Matrica udaljenosti se izračunava pomoću matrice s koordinatama vrhova.
- **Parametri za optimizaciju algoritma** - predstavljaju vrijednosti koje utječu na odabir vrhova u izgradnji puta. Te vrijednosti predstavljaju alfa i beta. (vidi izraz 3.2.)
- **Ukupna prijeđena udaljenost** - kao atribut predstavlja mogućnost mrava da pamti svoju prijeđenu udaljenost

Uz atribute, klasa *Ant* sadrži i metode :

- **probability_to_node()** – vidi prilog 6.1. između linija 21-27. Metoda koja kao povratnu vrijednost vraća vjerojatnost odabira pojedinog vrha koja se izračunava po formuli 3.2., u kojoj se koriste vrijednosti iz matrice udaljenosti.
- **roulette_wheel()** – vidi prilog 6.1. između linija 27-40, predstavlja sustav selekcije odabira sljedećeg vrha u konstrukciji puta. Vjerojatnost odabira pojedine jedinke(vrha) v_i ovisi o iznosu dobrote. Određuje omjer dobrote ili kvalitete jedinke i ukupne dobrote svih jedinki. [3]

$$p_k = \frac{dobrota(v_i)}{\sum_{i=0}^{n} dobroda(v_i)} \tag{4.1.}$$

Klasa *ACO* povezuje i ujedinjuje dijelove algoritma, pamti najbolje rješenje, sadrži sve metode ažuriranja feromona, kao i metodu aco() (vidi prilog 6.1. između linija 98-109) koja pokreće algoritam, stvara objekte mravi koji grade rješenja i pri tome izvršava ažuriranje feromona. Bitni atributi ACO klase su veličina kolonije i broj iteracija gdje veličina kolonije predstavlja broj

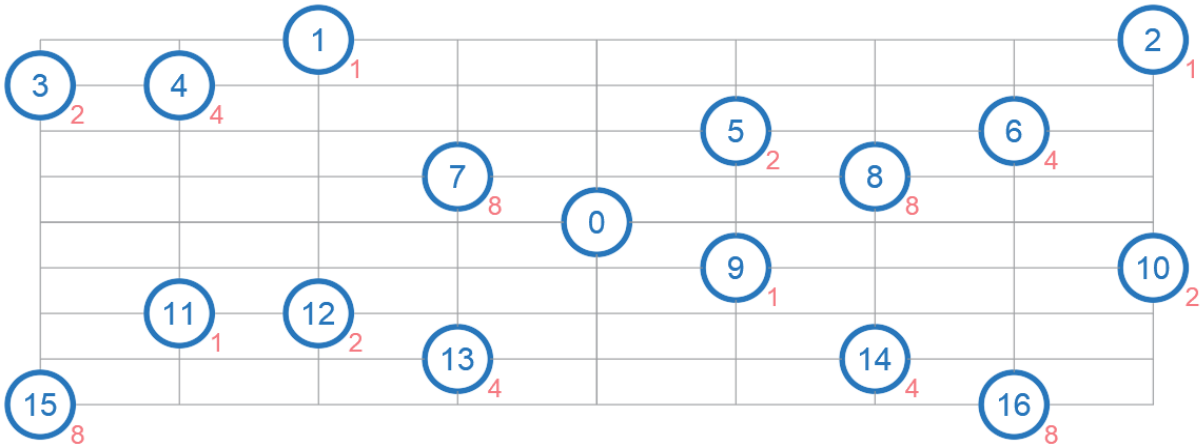
umjetnih mravi koji se koriste u određivanju rješenja, a broj iteracija predstavlja koliko će ta ista kolonija mrava ponoviti proces određivanja rješenja.

5. EKSPERIMENTALNI REZULTATI

U ovom poglavlju prikazani su rezultati dobiveni testiranjem ACO algoritma u rješavanju TSP i CVRP. Detaljnije se ulazi u pojedini problem i objašnjava se način implementacije problema u ACO algoritam.

5.1. Problem trgovačkog putnika

TSP igra važnu ulogu u optimizaciji kolonije mrava budući da je prvi ACO algoritam bio primijenjen na TSP, lako se primjenjuje i daje dobra rješenja na problem. TSP je problem prodavača koji želi pronaći, počevši od svog izvornog grada, najkraće moguće putovanje kroz zadani skup klijenata (gradova), to jest vrhova, te povratak u svoj početak putovanja. [3]. Slika 5.1. prikazuje primjer jednog takvog problema. Plavi kružići, unutar kojih su upisani brojevi, predstavljaju gradove i lokacije koje prodavač mora obići. Širina jednog pravokutnika iznosi 3, dok visina iznosi 1. Redni broj 0 na slici predstavlja početnu lokaciju putnika, odnosno ishodište koordinatnog sustava.



Sl. 5.1. Graf sa lokacijama vrhova

Teorijski, rješenje problema se može predstaviti kao kompletni težinski graf $G=(N,A)$ gdje N predstavlja skup vrhova i u ovom konkretnom slučaju iznosi 17, a A predstavlja skup bridova koji međusobno povezuju vrhove N. Bridova A ima $(N(N\text{-}1))/2$, što znači da je svaka lokacija međusobno povezana, to jest iz svake lokacije se može direktno pristupiti svakoj drugoj. Svaka grana sadrži vrijednost d_{ij} što predstavlja dužinu grane $(i, j) \in N$, tj. udaljenosti između grada i i j.

TSP je problem pronalaženja Hamiltonovog ciklusa najmanje dužine. Hamiltonov ciklus je put u grafu koji posjećuje svaki vrh točno jednom, osim prvoga i zadnjega vrha. Što znači da počinje i završava u istoj točci. U TSP, udaljenost između vrhova je neovisna o smjeru, tj. $d_{ij}=d_{ji}$.[3]

5.1.1. Dobiveni rezultati

Kao što je navedeno u trećem poglavlju, algoritmu se pri inicijalizaciji moraju predati parametri za optimizaciju algoritma, čije su vrijednosti ovisne o kompleksnosti samoga problema. Za TSP, kompleksnost ovisi o broju lokacija koje se moraju posjetiti i njihov međusobni raspored. U svrhu testiranja algoritma, analizirat će se algoritam kroz više kombinacija dvaju parametra, broja mravi i broja iteracija, dok će vrijednosti alfe i bete biti konstante,α=1.5 i β= 2.0, te će se zaključiti optimalni parametri za zadani problem.

Algoritam je pokrenut za 6 kombinacija parametara, to jest za sve kombinacije gdje je broj mravi u koloniji $\in$ {20,50,100} i broj iteracija $\in$ {10,30}. Za svaku kombinaciju eksperiment je ponovljen 10 puta. Kao rezultat dobiva se ukupna prijeđena udaljenost najboljeg pronađenog puta, kao i prosjek svih udaljenosti pojedine kombinacije parametara.

Tablica 5.1. Rezultati za broj iteracija=10

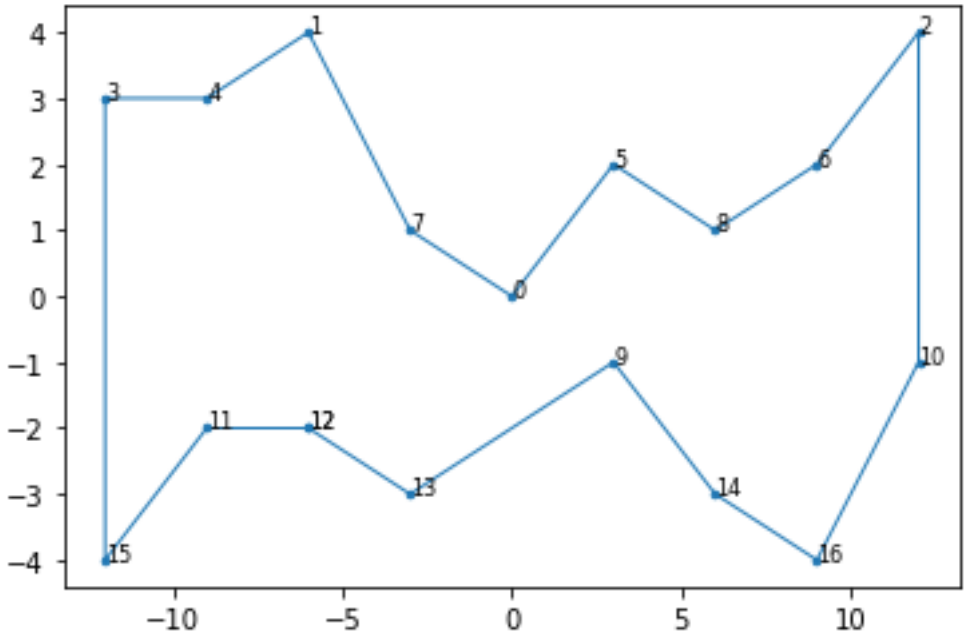
		Broj mravi		
	Redni broj eksperimenta	20	50	100
Prijeđena udaljenost (najbolje rješenje)	1.	73.96307	69.35921	67.24813
	2.	82.53367	71.73957	66.20571
	3.	79.0574	71.39028	68.41478
	4.	79.9836	72.48213	66.601
	5.	73.96307	71.20482	68.36121
	6.	78.50578	70.96152	66.20571
	7.	77.51302	71.20335	67.83233
	8.	72.06657	69.76155	66.20571
	9.	81.60466	69.56773	66.20571
	10.	78.35507	69.38957	66.601
Prosjek		78.27985	70.70597	66.98813

Tablica 5.2. Rezultati za broj iteracija=30

		Broj mravi		
	Redni broj eksperimenta	20	50	100
Prijeđena udaljenost (najbolje rješenje)	1.	66.601	66.20571	66.20571
	2.	66.20571	66.20571	66.20571
	3.	66.20571	66.20571	66.20571
	4.	66.20571	66.20571	66.20571
	5.	66.20571	66.20571	66.20571
	6.	66.20571	66.20571	66.20571
	7.	66.601	66.20571	66.20571
	8.	66.601	66.20571	66.20571
	9.	66.20571	66.20571	66.20571
	10.	67.24813	66.20571	66.20571
Prosjek		66.42854	66.20571	66.20571

Kao što je vidljivo iz tablice 5.1. i tablice 5.2. najbolja rješenja su dobivena za kombinacije s 30 broja iteracija i 50 i 100 broja mravi gdje najkraća prijeđena udaljenost iznosi 66.20571.

Na slici 5.2. je prikazano optimalno dobiveno rješenje, to jest najkraći put. Vidljivo je na grafu da „trgovac“ obilazi sve lokacije točno jedanput i vraća se u početnu lokaciju.



Sl. 5.2. Najbolje rješenje TSP

U nastavku je objašnjen utjecaj veličine kolonije, to jest, broja mravi na izvođenje algoritma. Kao što je vidljivo po rezultatima eksperimenta, povećanjem broja mravi i broja iteracije, povećava se preciznost algoritma. Povećavanjem broja iteracija, preciznost algoritma će uvijek biti veća, dok povećanje broja mravi može imati i negativne utjecaje na preciznost algoritma što je detaljnije objašnjeno u nastavku. Spomenuto je u drugom poglavlju da se dodavanje feromona vrši tek nakon što svaki mrav u koloniji izgradi svoj put, dok se isparavanje feromona vrši tek kada cijela kolonija mrava izgradi svoja rješenja, to jest nakon svake iteracije. Prema tome, što je veći broj mrava u koloni, veća će se količina feromona dodati na bridove prije nego što se primjeni isparavanje feromona. U sljedećim primjerima će se promatrati količina feromona na bridu a_{34}, to jest na bridu između vrhova 3 i 4,vidi sliku 5.1. Odabrani brid pripada putu najboljeg izgrađenog puta, pa se očekuje da stanje feromona raste kako se program izvodi. Prije nego što se razmotre rješenja, bitno je znati da početna razina feromona na svim bridovima u grafu iznosi 100.

Tablica 5.3. Stanje feromona na bridu a_{34} za broj mrava=20

Redni broj iteracije	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.
Razina feromona na a_{34}	99	97	94	94	93	87	86	83	84	83

Tablica 5.4. Stanje feromona na bridu a_{34} za broj mrava=50

Redni broj iteracije	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.
Razina feromona na a_{34}	108	109	114	120	129	133	137	140	147	154

Tablica 5.5. Stanje feromona na bridu a_{34} za broj mrava=100

Redni broj iteracije	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.
Razina feromona na a_{34}	112	130	155	168	188	205	223	238	252	263

Iz tablice 5.3. vidljivo je da razina feromona na bridu a_{34} se konstantno smanjuje. Razlog tome je to što nema dovoljno mrava po iteraciji da ostave dovoljno feromona. Te zbog toga isparavanje feromona ima dominantan učinak.

Iz tablice 5.5. gdje ima 100 mrava po iteraciji, vidljivo je da razina feromona između iteracija ubrzano raste, što znači da ako mravi odabiru krivi put, šanse za ispravak će biti sve manje i manje.

Jer isparavanje feromona služi upravo tome da se ispravi loše odabrani put, ali ako utjecaj isparavanja nije značajan, šanse za to su male.

U tablici 5.4. vidimo optimalni rast razine feromona. Dovoljno nizak da utjecaj mrava na odabir tog puta ne bude dominantan, ali i dovoljno visok rast da isparavanje feromona ne utječe na razvijanje feromona.

5.1.2. Utjecaj ostalih parametara

Za ispitivanje utjecaja parametara alfe i bete, pokrenut je algoritam dva puta, gdje će se u jednom slučaju postaviti vrijednosti $\alpha = 2$ i $\beta = 0$. Te drugi puta vrijednosti $\alpha=0$ i $\beta=2$ uz broj iteracija = 30 i broja mravi = 50.

Tablica 5.6. Rezultati ovisno o alfi i beti

	$\alpha = 2$ $\beta = 0$	$\alpha = 0$ $\beta = 2$
Prijeđena udaljenost	117.60725818821605	79.14899553235915

U prvom slučaju gdje je $\alpha >> \beta$, mravi pri odabiru vrhova u putu gledaju samo razinu feromona na bridovima, a dužinu bridova zanemaruju, što znači da je mravu jednako ako odabere najbližeg susjeda ili najudaljenijeg.

U drugom slučaju gdje je $\beta >> \alpha$, mravi će preferirati samo vrhove do kojih moraju prijeći najmanju udaljenost. Znaci ovisno na kojem vrhu se mrav početno nalazi, u većini slučajeva uvijek će odabrati isti put, te kvaliteta rješenja nikad neće rasti.

Slično kao što algoritam mora imati dobre kombinacije ulaznih parametara broja mravi i broja iteracije, bitna je i kombinacija parametara alfe i bete.

5.2. Problem određivanja rute vozila s ograničenim kapacitetom

Problem određivanja rute vozila (eng. Vehicle routing problem - VRP) odnosi se na prijevoz proizvoda između skladišta i kupaca pomoću grupe vozila. VRP se može primijeniti na mnoge probleme u stvarnom svijetu. Općenito rješavanje VRP-a znači pronaći najbolji put za obilazak svih kupaca točno jedanput koristeći više vozila, pritom poštujući ograničenja. CVRP proširena je verzija VRP-a. Naziv potječe od ograničavanja vozila sa određenim kapacitetom. Ako pogledamo sliku 5.1., vidimo da pored svakog vrha označenog plavom bojom, vidimo i težinu proizvoda na istome vrhu označene crvenom bojom. Kako se vozilo kreće po lokacijama i skuplja proizvode, tako se smanjuje kapacitet vozila koji u našem konkretnom slučaju iznosi 15. Svako vozilo se

ponaša na način da, ako dostigne maksimalni kapacitet ili ima nedovoljan preostali kapacitet da može posjetiti ijedan vrh, vraća se u početnu točku koja se može nazvati skladište koje se u ovom konkretnom slučaju nalazi na vrhu 0. Optimalan put je najkraći put koji obiđe sve lokacije pritom poštivajući ograničenja vozila i ograničenja broja vozila. Za problem sa slike 5.1. koristit ćemo 4 vozila za rješavanje problema. Problem se može formulirati kao teoretski problem grafova, gdje je $G=(V,A)$ potpuni graf, dok V predstavlja skup vrhova (kupci i skladište), a A predstavlja skup bridova koje povezuju sve vrhove međusobno.

5.2.1. Dobiveni rezultati

Kao i pri testiranju TSP-a, za testiranje CVRP-a koristit će se više kombinacija ulaznih parametara broja mrava i broja iteracija, dok će vrijednosti α=1.5 i β=2.5 biti konstante kao i kapacitet svih vozila=15 i broj vozila=4.

Algoritam smo pokrenuli za 9 kombinacija parametara, to jest za sve kombinacije gdje je broj mravi $\in$ {10,20,50} i broj iteracija $\in$ {10,30,200}.Eksperiment se ponavlja 10 puta za svaku kombinaciju parametara. Kao rezultat dobiva se ukupna prijeđena udaljenost najboljeg puta.

Tablica 5.7. CVRP Rezultati za broj iteracija=10

		Broj mravi		
	Redni broj eksperimenta	20	50	100
Prijeđena udaljenost (najbolje rješenje)	1.	104.5092	103.894	102.0483
	2.	104.5092	101.867	102.0483
	3.	104.5092	103.894	101.892
	4.	104.5092	101.8925	101.867
	5.	104.1298	102.0483	101.867
	6.	103.932	104.8139	101.9813
	7.	105.0332	103.2051	102.0483
	8.	104.8394	103.7275	103.2812
	9.	101.8925	104.2864	101.8925
	10.	105.5547	101.7875	101.9433
Prosjek		104.3418	103.1416	102.087

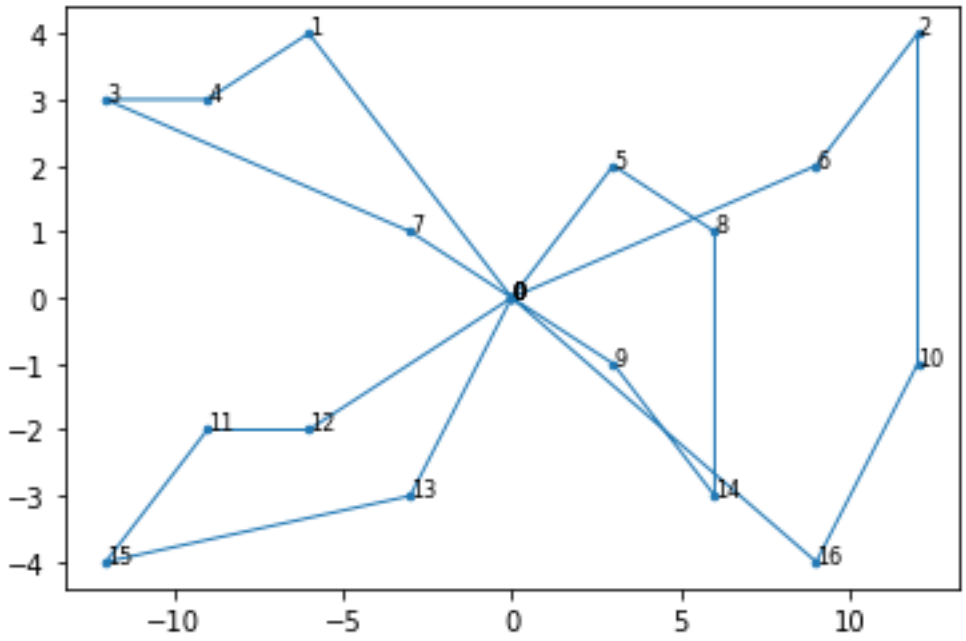
Tablica 5.8. CVRP Rezultati za broj iteracija=30

		Broj mravi		
	Redni broj eksperimenta	20	50	100
Prijeđena udaljenost (najbolje rješenje)	1.	102.0483	101.6169	102.0483
	2.	101.6169	102.0483	101.867
	3.	104.1589	101.867	101.4611
	4.	102.0483	102.0483	102.0483
	5.	101.867	101.8925	101.867
	6.	102.0483	102.0483	101.867
	7.	102.0483	101.6169	101.4611
	8.	103.789	104.1589	101.8925
	9.	102.0483	102.0483	102.0483
	10.	102.0483	102.0483	101.867
Prosjek		102.3722	102.1394	101.8428

Tablica 5.9. CVRP Rezultati za broj iteracija=200

		Broj mravi		
	Redni broj eksperimenta	20	50	100
Prijeđena udaljenost (najbolje rješenje)	1.	102.0483	101.9433	101.867
	2.	101.867	101.4356	101.4356
	3.	102.0483	101.4356	101.8925
	4.	102.0483	101.8925	101.4356
	5.	101.6169	101.9433	101.6169
	6.	102.0483	101.6169	101.9433
	7.	102.0483	101.4356	101.4356
	8.	102.0483	101.8925	101.813
	9.	101.9813	101.4356	101.6169
	10.	101.6169	102.0483	101.6169
Prosjek		101.9372	101.7079	101.6673

Kao što je vidljivo iz tablice 5.7.,5.8. i 5.9. najbolja rješenja su dobivena za kombinacije sa 200 broja iteracija i 50 i 100 broja mravi gdje najkraća udaljenost dobivenog puta iznosi 101.4356 prikazan na slici 5.9.



Sl. 5.3. Najbolje rješenje CVRP

Kao što je vidljivo sa slike 5.9., postoje točno 4 rute koje počinju i završavaju u nuli, to jest točno 4 vozila koji se nakon obilaska i dostizanja svog kapaciteta vraćaju u skladište.

6. ZAKLJUČAK

Cilj rada je bio pobliže objasniti strukturu ACO algoritma, kako se algoritam ponaša na principu bioloških mrava da bi se riješili pravi problemi iz svakodnevnog života. Implementirana je najosnovnija verzija ACO algoritma u Python programskom jeziku, te je algoritam analiziran primjenom na dva problema - TSP i CVRP. Na TSP i CVRP primjeru je pobliže objašnjen izvor problema, te su dobiveni bolji rezultati što je broj mrava veći kao i broj iteracija. Također je analiziran utjecaj broja mravi u koloniji na promjenu razine feromona na bridovima i zaključen je optimalan broj mravi za pravilno razvijanje feromona kroz tijek algoritma. Analiziran je utjecaj parametara alfe i bete na konačna rješenja, i utvrđen je njihov utjecaj pri izgradnji puta u grafu.

LITERATURA

[1] M.Dorigo, V. Maniezzo i A. Colorni, Ant System, Optimization by a colony of cooperating agents,1996

[2] S.Kelly, Ant Colony Optimisation in Parallel,2021.

[3] L.Pereira, Santos, G.Vieira, H.Vinicius, R.Leite, M.Teresinha, A.Steiner, Ant Colony Optimisation for Backward Production Scheduling,2020

[4] M.Comuzzi, Optimal directed hypergraph traversal with ant-colony optimisation, 2018

[5] J.Muriel, A.Fotouhi, Electric Vehicle Fleet Menagement Using Ant Colony Optimisation,2020

[6] Z.Dong, L.Lei, F.Yu,A-H.Ali,W. Mingjing,O. Diego, M. Khan, C. Huiling, Ant colony optimization with horizontal and vertival crossover search:Fundemental visions for multi-threshold image segmentation,2021

[7] A.-E.Rizzoli, O.F., R.Montemanni i L. M.Gambardella, Ant Colony Optimisation for vehicle routing problems: from theory to applications, 2003.

[8] Wikipedia,Fitness proportionate selection [online],dostupno na: https://en.wikipedia.org/wiki/Fitness_proportionate_selection

[9] St.Thomas i M.Dorigo,Aco Algoritms for the Traveling Salesman Problem,1999.

SAŽETAK

Problem rada je bilo objasniti i izvršiti analizu algoritma optimizacije kolonijom mravi rješavanjem problema trgovačkog putnika i problema usmjeravanja vozila s ograničenim kapacitetom. Objašnjena je izvorna inspiracija ACO algoritma. Detaljno su opisani ključni dijelovi i aktivnosti algoritma te na temelju dobivenih rezultata eksperimenata, prikazana su najbolja dobivena rješenja. Prikazana je osjetljivost ACO algoritma na promjene parametara, posebno na sam tijek algoritma kao i na konačno rješenje.

Ključne riječi: Algoritam optimizacije kolonijom mravi, problem trgovačkog putnika, problem određivanja rute vozila s ograničenim kapacitetom

Implementation of ant colony optimization algorithm in Python programming language

The ant colony optimization algorithm is explained and analyzed by solving the traveling salesman problem and capacitated vehicle routing problem. The original inspiration of ACO algorithm and the key parts and activities of ACO are explained. Based on the obtained experimental results, the best solution to the problems are presented. The influence and the sensitivity of the ACO algorithm parameters were shown on the flow of the algorithm as well as the final solution.

Keywords: Ant colony optimization, traveling salesman problem, Capacitated vehicle routing problem

PRILOG

P.6.1. Izlistanje koda u python programskom jeziku

```
import math
import random
from matplotlib import pyplot as plt
import pandas as pd

class Edge:
    def __init__(self, a, b, edge_distance, initial_pheromone):
        self.a = a
        self.b = b
        self.edge_distance = edge_distance
        self.pheromone = initial_pheromone

class Ant:
    def __init__(self, alpha, beta, num_nodes, dist_matrix):
        self.alpha = alpha
        self.beta = beta
        self.num_nodes = num_nodes
        self.dist_matrix = dist_matrix
        self.tour = None
        self.distance = 0.0

    def probability_to_node(self,unvisited_node):
      dist=pow(self.dist_matrix[self.tour[-1]][unvisited_node].pheromone,\
               self.alpha)*pow((1/self.dist_matrix[self.tour[-1]]\
                               [unvisited_node].edge_distance),self.beta)
      return dist

    def roulette_wheel(self,unvisited_nodes):
        roulette_wheel = 0.0
        for unvisited_node in unvisited_nodes:
            roulette_wheel += self.probability_to_node(unvisited_node)

        random_value = random.uniform(0.0, roulette_wheel)
        wheel_position = 0.0

        for unvisited_node in unvisited_nodes:
            wheel_position += self.probability_to_node(unvisited_node)
            if wheel_position >= random_value:
                return unvisited_node

    def calculate_tour_distance(self):
        self.distance = 0.0
        for i in range(len(self.tour)-1):
            self.distance +=\
            self.dist_matrix[self.tour[i]][self.tour[(i + 1)]].edge_distance
        return self.distance

class ACO:
    def __init__(self, mode='ACO', colony_size=10, alpha=1.5, beta=3.0,
                 rho=0.1, initial_pheromone=1.0, steps=100, nodes=None,\
                 labels=None):
        self.mode = mode
```

```
        self.colony_size = colony_size
        self.rho = rho
        self.steps = steps
        self.num_nodes = len(nodes)
        self.nodes = nodes
        self.global_best_tour = None
        self.global_best_distance = float("inf")
        self.dist_matrix = self.get_distance_matrix(initial_pheromone)
        if labels is not None:
            self.labels = labels
        else:
            self.labels = range(0, self.num_nodes + 0)

    def get_distance_matrix(self,initial_pheromone):
        dist_matrix = [[None] * self.num_nodes for i in range(self.num_nodes)]
        for i in range(self.num_nodes):
            for j in range(i + 1, self.num_nodes):
              dist_matrix[i][j]=self.Edge(i, j,self.get_distance_two_nodes(i, j),
                                                          initial_pheromone)
              dist_matrix[j][i]=dist_matrix[i][j]
        return dist_matrix

    def get_distance_two_nodes(self,i,j):
        distance = math.sqrt(pow(self.nodes[i][0] - self.nodes[j][0], 2.0) +
                             pow(self.nodes[i][1] - self.nodes[j][1], 2.0))
        return distance

    def add_pheromone(self, tour, distance, pheromone_factor=1.0):
        pheromone_to_add = self.rho/ (self.num_nodes / distance)
        self.local_pheromone_update(tour,pheromone_to_add)

    def local_pheromone_update(self,tour,pheromone_to_add):
        for i in range(len(tour)-1):
            self.dist_matrix[tour[i]][tour[(i + 1)]].pheromone +=pheromone_to_add

    def pheromone_disintegration(self):
        for i in range(self.num_nodes):
            for j in range(i + 1, self.num_nodes):
                self.dist_matrix[i][j].pheromone *= (1.0 - self.rho)

    def global_pheromone_update(self):
        for i in range(len(self.global_best_tour)-1):
         self.dist_matrix[self.global_best_tour[i]]\
                         [self.global_best_tour[(i+1)]].pheromone\
                         += self.rho*(1/self.global_best_distance)

    def aco(self):
        for step in range(self.steps):
            for ant in self.ants:
                self.add_pheromone(ant.find_tour(), ant.calculate_tour_distance())
                if ant.distance < self.global_best_distance:
                    self.global_best_tour = ant.tour
                    self.global_best_distance = ant.distance
                self.pheromone_disintegration()

                self.global_pheromone_update()

```

```
    def run(self):
     print('Started')
     if self.mode == 'ACO':
       self.aco()
       print('Ended')
       print('Sequence:<-{0}->'.format('-'.join(str(self.labels[i])\
             for i in self.global_best_tour)))
       print('Total distance travelled:{0}\n'.format\
             (round(self.global_best_distance,2)))

    def plot(self, line_width=1, point_radius=math.sqrt(2.0),
             annotation_size=8, dpi=120, save=True, name=None):
        x = [self.nodes[i][0] for i in self.global_best_tour]
        y = [self.nodes[i][1] for i in self.global_best_tour]
        plt.plot(x, y, linewidth=line_width)
        plt.scatter(x, y, s=math.pi * (point_radius ** 2.0))
        plt.title(self.mode)
        for i in self.global_best_tour:
            plt.annotate(self.labels[i], self.nodes[i], size=annotation_size)
        if save:
            if name is None:
                name = '{0}.png'.format(self.mode)
            plt.savefig(name, dpi=dpi)
        plt.show()
        plt.gcf().clear()

class TSP(ACO):       #Traveling salesman problem
    class Edge(Edge):
        def __init__(self, a, b, edge_distance, initial_pheromone):
            super().__init__(a, b, edge_distance, initial_pheromone)
    class Ant(Ant):
        def __init__(self, alpha, beta, num_nodes, dist_matrix):
            super().__init__(alpha, beta, num_nodes, dist_matrix)

        def select_node(self):
            unvisited_nodes = [node for node in range(self.num_nodes)\
                               if node not in self.tour]
            selected_node=self.roulette_wheel(unvisited_nodes)
            return selected_node

        def find_tour(self):
            self.tour = [random.randint(0, self.num_nodes - 1)]
            while len(self.tour) < self.num_nodes:
                self.tour.append(self.select_node())
            self.tour.append(self.tour[0])
            return self.tour

    def __init__(self,mode='ACO',colony_size=10,alpha=1.5,beta=2.0,
                 rho=0.1,initial_pheromone=100.0,steps=100,nodes=None,\
                 labels=None):

        super().__init__(mode, colony_size, alpha, beta,
                         rho,initial_pheromone, steps, nodes, labels)

        self.ants = [self.Ant(alpha, beta, self.num_nodes, self.dist_matrix)\
                     for _ in range(self.colony_size)]

```

```
167
168 class CVRP(ACO):      #Capacitated vehicle routing problem VRP
169     class Node:
170         def __init__(self,weight):
171             self.weight=weight
172     class Edge(Edge):
173         def __init__(self, a, b, edge_distance, initial_pheromone):
174             super().__init__(a, b, edge_distance, initial_pheromone)
175     class Ant(Ant):
176         def __init__(self, alpha, beta, num_nodes, dist_matrix,
177                      weight_matrix,vehicle_num,load_capacity):
178             super().__init__(alpha, beta, num_nodes, dist_matrix)
179             self.weight_matrix=weight_matrix
180             self.current_load=0.0
181             self.load_capacity=load_capacity
182             self.vehicle_num=vehicle_num
183             self.current_vehicle_num=vehicle_num
184         def select_node(self):
185             unvisited_nodes = [node for node in range(self.num_nodes)\
186                                if node not in self.tour]
187             unvisited_nodes_selection=[ ]
188             for next_node in unvisited_nodes:
189                 if(self.current_load + self.weight_matrix[next_node].weight>\
190                    self.load_capacity):
191                     continue
192                 unvisited_nodes_selection.append(next_node)
193             if not unvisited_nodes_selection:
194                 self.current_load=0;
195                 self.current_vehicle_num-=1
196                 return 0;
197             selected_node=self.roulette_wheel(unvisited_nodes_selection)
198             self.current_load+=self.weight_matrix[selected_node].weight
199             return selected_node
200
201         def find_tour(self):
202             self.current_vehicle_num=self.vehicle_num
203             self.current_load=0
204             self.tour =[0]
205             while len(self.tour) < self.num_nodes+self.vehicle_num :
206                 self.tour.append(self.select_node())
207             if(self.current_vehicle_num < 0 or self.tour[-1]!=0 ):
208                 return self.find_tour()
209             return self.tour
210
211     def __init__(self, mode='ACO', colony_size=10, alpha=1.5, beta=2.5,
212                  rho=0.1,initial_pheromone=100, steps=100, nodes=None,
213                  nodes_weight=None,vehicle_num=4,vehicle_capacity=15,labels=None):
214       super().__init__(mode, colony_size, alpha, beta,
215                        rho,initial_pheromone,steps,nodes,labels)
216       self.nodes_weight=self.get_weight_matrix(nodes_weight)
217       self.vehicle_num=vehicle_num
218       self.vehicle_capacity=vehicle_capacity
219       self.ants=[self.Ant(alpha, beta, self.num_nodes, self.dist_matrix,\
220                           self.nodes_weight,self.vehicle_num,\
221                           self.vehicle_capacity)for _ in range(self.colony_size)]
222
223     def get_weight_matrix(self,nodes_weight):
```

```
224         weight_matrix=[[None] * self.num_nodes for i in range(self.num_nodes)]
225         for i in range (self.num_nodes):
226             weight_matrix[i]=self.Node(self.get_edge_weight(nodes_weight,i))
227         return weight_matrix
228 
229     def get_edge_weight(self,nodes_weight,i):
230         weight=nodes_weight[i]
231         return weight
```