Metode rješavanja kombinatoričke optimizacije

Korenić, Francesca

Sveučilište Jurja Dobrile u Puli

Fakultet informatike

FRANCESCA KORENIĆ

METODE RJEŠAVANJA PROBLEMA KOMBINATORIČKE OPTIMIZACIJE

Diplomski rad

Pula, rujan, 2019. god.

Sveučilište Jurja Dobrile u Pui

Fakultet informatike

FRANCESCA KORENIĆ

METODE RJEŠAVANJA PROBLEMA KOMBINATORIČKE OPTIMIZACIJE

Diplomski rad

JMBAG: 0242023468; redovita studentica

Studijski smjer: Sveučilišni diplomski studiji Informatike

Kolegiji: Napredni algoritmi i strukture podataka

Znanstvrno područje: Društvene znanosti

Znanstveno polje: Informacijske i komunikacijske znanosti

Znanstvena grana: Informacijski sustavi i informatologija

Mentor: doc. dr. sc. Tihomir Orehovački

Pula, rujan, 2019. god.

IZJAVA

o korištenju autorskog djela

Ja, Francesca Korenić dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj diplomski rad pod nazivom „Metode rješavanja problema kombinatoričke optimizacije“ koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.

Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, 16. rujna 2019.

Potpis

_____

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisana Francesca Korenić, kandidat za magistra informatike ovime izjavljujem da je ovaj Diplomski rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Diplomskog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Studentica

_____

U Puli, 16. rujna, 2019. godine

Sadržaj

Sažetak

Problem trgovačkog putnika je jedan od najpoznatijih i najproučavanijih optimizacijskih problema. Njegov matematički model je traženje Hamiltonovog ciklusa najmanje težine u težinskom grafu. Optimizacija kolonijom mrava je metaheuristika koja se uspješno primjenjuje za rješavanje teških optimizacijskih problema, osobito kombinatoričkih optimizacijskih problema koji pripadaju klasi NP-teških problema. Cilj ovog diplomskog rada je predstaviti problem trgovačkog putnika te proširiti znanje o načinu djelovanja algoritma koji je definiran kolonijom mrava. Također, opisan je i genetski algoritam koji svoju primjenu bazira na prirodnim pojavama.

Ključne riječi: mravi, optimizacija, genetski algoritam, optimizacija kolonijom mrava, problem trgovačkog putnika, Hamiltonov ciklus

Summary

The problem of a traveling salesman is one of the most well-known and most effective for optimization problems. His mathematical model is to search for Hamilton cycle with the least weight in the weight graph. Ant colony optimization is a metaheuristic that is successfully applied for solving difficult optimization problem, especially combinatorial optimization problems that belong to the group of NP-hard problems. The main goal of this thesis is to present the problem of traveling salesman and to expand the knowledge of the algorithm defined by ant colony method. Therefore, in this paper is described genetic algorithm which uses its application in natural phenomena.

Keywords: ants, optimization, genetic algorithms, Ant Colony Optimization, Traveling Salesman Problem, Hamilton's cycle

Uvod

Ponašanje insekata najviše proučavaju biolozi, ali u novije vrijeme sve više i ostali znanstvenici. Naime, istraživanja kolektivnog ponašanja insekata omogućila su znanstvenicima u računalstvu da razviju moćne metode i algoritme za distribuirano upravljanje i optimizaciju. Kretanje mrava i pčela inspirirali su razvoj nekih modernih znanosti kao što je primjerice kolektivna inteligencija. Predmet ovog rada su neke linije istraživanja koje se bave transformacijom znanja o modelu kolektivnog ponašanja insekata pri rješavanju određenih njihovih problema, kao što je potraga za hranom, korisne optimizacijske i upravljačke algoritme za rješavanje tehničkih i drugih problema iz prakse. Dobar primjer su mravi i njihovo kooperativno traženje hrane, odnosno mravlji algoritmi i mravlji sistemi koji su se pokazali kao neočekivano uspješni pri rješavanju i najsloženijih kombinatornih problema. U posljednjem desetljeću strukturiran je novi model optimizacije nazvan „Optimizacija pomoću mravljih kolonija“ (eng. Ant Colony Optimization, ACO) kojeg karakteriziraju specifični optimizacijski algoritmi inspirirani ponašanjem prirodnih mravljih kolonija. Navedena metoda optimizacije zasniva se na indirektnoj komunikaciji i suradnji više agenata, slično kao što u prirodi komuniciraju i ponašaju neke vrste mrava. Navedeni algoritam rješava NP-teške i NP-kompletne kombinatorne probleme. Upravo navedeno je tema ovog rada gdje je definirano nekoliko NP-teških problema te je objašnjen genetski algoritam i opisan problem trgovačkog putnika pomoću genetskog algoritma. U radu je skraćeni opis navedenog koncepta optimizacije, a zatim je opisana primjena jednog mravljeg algoritma u rješavanju problema trgovačkog putnika.

Kod problema trgovačkog putnika, cilj je pronaći put kojim će se obići sva odredišta uz najmanji trošak (primjerice, najkraći prijeđeni put). Stoga, problem trgovačkog putnika je odabran kao primjer na kojem će se objasniti optimizacija kolonijom mrava. Za problem trgovačkog putnika se može reći da je to jedan od najistraženijih i najpoznatijih optimizacijskih problema te je pri rješavanju navedenog problema došlo do kombinatoričke eksplozije nakon čega je razvijen velik broj heurističkih metoda za rješavanje. No, često se koriste genetski algoritmi, ali u novije vrijeme se koriste i teorije rojeva (eng. swarm intelligence) koji pogoduju različitim oblicima paralelizacije (primjerice, grid computing).

U ovom radu će se u kratko obraditi i problem raspoređivanja, tj. dodjele ograničenih sredstava pojedinom skupu aktivnosti s ciljem optimiranja jednog ili više parametra. Sredstva (eng. resources) su definirana ovisno o domeni kojoj pripada zadani problem. Sredstva mogu biti strojevi u proizvodnom pogonu, procesor na računalu, ljudska radna snaga i sl. Aktivnosti mogu biti operacije u proizvodnji, računalni programi koji se izvode na računalu, poslovi koji zahtijevaju ljudsku snagu. Rješenje problema raspoređivanja je redoslijed obavljanja aktivnosti na određenom sredstvu. Problem je riješen ako je napravljen „dovoljno dobar“ raspored koji će zadovoljiti određene potrebe.

U prvom poglavlju su opisani optimizacijski problemi te vrste problema optimizacije.

U drugom poglavlju su objašnjeni evolucijski procesi u prirodi.

U trećem poglavlju je opisan genetski algoritam i njegove najvažnije karakteristike. U potpoglavljima su objašnjene teme poput kodiranja svojstva jedinke i inicijalizacija, postupci selekcije genetskog algoritma, genetski operatori križanja, mutacije i selekcija te parametri genetskog algoritma.

U četvrtom poglavlju je objašnjen problem trgovačkog putnika te su prikazane metode rješavanja navedenog problema. Također, navodi se povezanost pčela i problema trgovačkog putnika te se navodi primjer.

U petom poglavlju se opisuje algoritam optimizacije kolonijom mrava te teorijska podloga optimizacijom mrava.

U šestom poglavlju je prikazan problem raspoređivanja poslova, nakon čega slijedi prilog.

1. Optimizacijski problemi

Svakim danom suvremeno društvo transformira se u informacijsko društvo te je taj proces sve brži i brži. Globalizacija mijenja povijesne obrasce ekonomskog i društvenog ponašanja te napredak u informacijskoj tehnologiji zajedno s pripadajućim smanjivanjem troškova obrade informacija predstavljaju moćan poticaj za prijelaz iz analogno-papirnate u digitalnu informacijsku domenu. Mnogi čimbenici čine tu tranziciju sve lakšom i isplatljivijom, a kao najvažniji čimbenici su sve brže i jeftinije računalno sklopovlje, sve kvalitetniji programski alati za razvoj aplikacija te sve bolje mogućnosti međusobne komunikacije informacijskih sustava pri čemu Internet ima presudnu ulogu. Ugradnja ERP (eng. Enterprise Resource Planning) i CRM (eng. Customer Relationship Management) sustava u gospodarske subjekte vodi ka kvalitetnijem planiranju i praćenju poslova, a brzi rast područja dubinske analize podataka (eng. data mining), gdje se istraživanjem sve složenijih odnosa između podataka nastoji što bolje iskoristiti dostupne informacije što predstavlja prirodnu evoluciju u cjelokupnom procesu informatizacije. Međutim, prebacivanje informacija u digitalni oblik uz laku mogućnost dohvata, pretraživanja i prikaza predstavlja tek prvi korak u iskorištavanju informacijske tehnologije. Kao krajnji cilj se nameće mogućnost analize dostupnih informacija zbog što bolje potpore u donošenju odluka, bilo u korporativnoj ili javnoj domeni.

Tehnika dubinske analize podataka predstavljaju primjer navedenog sustava te kada je ispravno ugrađena u informacijski sustav, pruža zajedničku pomoć u procesu poslovnog odlučivanja, najviše u sektorima bankarstva i trgovine gdje se generira velika količina informacija te je potrebno primijeniti složene tehnike analize kako bi se uočili relevantni uzroci između podataka. No, postoji značajan skup problema koji se javljaju u praksi, a koji nisu prikladni za rješavanje tehnikama dubinske analize. Primjerice, „digitalizacija“ cjelokupnog procesa naručivanja i isporuke robe za lokalne trgovine iz skladišta, u smislu informatizacije cijelog sustava i pripadnih procesa, donosi sama po sebi značajan doprinos u vidu boljeg praćenja i kontrole stanja skladišta. No, nije od velike pomoći kod donošenja odluke kako će se naručena roba prevesti do lokalnih trgovina koristeći ograničene resurse prijevoza, konkretne kamione, pritom zadovoljavajući različita ograničenja. Kao drugi primjer koji se može navesti je informatizacija visokih učilišta putem projekta ISVU koji sam po sebi donosi

značajan napredak mogućnostima interaktivnog praćenja različitih statistika vezanih uz nastavu. Međutim, rješavanje problema definiranja satnice, tj. određivanje koji nastavnik predaje koji kolegiji, kojim grupama studenata, u koje vrijeme i u kojoj dvorani je poprilično težak problem, pogotovo imajući u vidu vrlo ograničene prostorne i ljudske resurse na većini fakulteta (Baranović i sur., 2003).

Rješavanjem navedenih problema bavi se optimiranje/optimizacija. Optimizacija se može definirati kao postupak kojim se u ekonomiji, tehničkim i prirodnim znanostima, pri projektiranju ili planiranju, ostvaruje, tj. određuje najbolji mogući izbor ekonomskih i/ili tehničkih veličina na temelju prethodno određenih kriterija (Anić i Goldstein, 1991.). Problem optimizacije se javlja u raznim granama znanosti kao što su optimizacija funkcija u matematici (problem pronalaženja minimuma ili maksimuma), različite vrste matematičkog programiranja (linearno, kvadratično i nelinearno), kombinatorno optimiziranje kao grana računarske znanosti, itd. Iako svaka vrsta optimizacije ima svoje specifičnosti vezane uz pripadajuću strukturu problema, svima im je zajedničko postojanje precizno definiranog problema i određenog skupa optimizacijskih postupaka ili metoda koje su primjenjive na taj problem.

U matematici i računalnoj znanosti, problem optimizacije je problem pronalaženja najboljeg rješenja iz svih izvedivih rješenja. Problemi s optimizacijom mogu se podijeliti u dvije kategoriji, ovisno o tome jesu li varijable kontinuirane ili diskretne. Optimizacijski problem s diskretnim varijablama poznat je kao diskretna optimizacija. U diskretnom optimizacijskom problemu traži se objekt kao što je cijeli broj, permutacija ili graf iz konačnih ili brojčano beskonačnih skupova. Problemi s kontinuiranim varijablama uključuju ograničene probleme i multimodalne probleme. No, može se zaključiti kako na neki način treba „pobjeći“ iz lokalnih optimuma.

Metode optimizacije su u zadnjih pedesetak godina doživjele nagli razvoj ponajviše zahvaljujući dvama čimbenicima. Prvi važan čimbenik je povećanje dostupne računalne snage (Mooreov zakon) što je omogućilo širu primjenjivost optimizacijskih metoda u IT zajednici i rješavanje sve složenijih problema. Drugi bitan faktor čine razvoj heurističkih algoritama kao što su genetički algoritmi, evolucijske strategije te simulirano kaljenje koji, iako ne garantiraju nalaženje optimalnog rješenja problema, pokazuju vrlo dobre rezultate u nalaženju približno optimalnog rješenja, ali u bitno kraćem vremenu. Mnogobrojni primjeri iz prakse koji su bili tehnički nerješivi tradicionalnim matematičkim egzaktnim tehnikama ili su se rješavali određenim

aproksimacijama, korištenjem primjerice linearnog programiranja, primjenom heurističkih algoritama postaju rješivi u smislu da se može doći do kvalitetnih rješenja problema. Može se reći da je pojava heurističkih metoda optimiranja dala novi poticaj raspravi o osnovnom pitanju optimiranja, a koje glasi: „Je li bolje riješiti točno približan problem ili približno riješiti pravi problem?“.

Za razliku od egzaktnih matematičkih metoda, primjerice različitih vrsta matematičkog programiranja, algoritama iz teorije mreža i grafova, koji imaju dugu tradiciju i dobro su teorijski utemeljeni, heurističke metode su nešto novijeg datuma, ali su ih uspjesi u rješavanju teških optimizacijskih problema istaknuli u središte pozornosti. Međutim, iako je napredak u razvoju pojedinačnih algoritama vrlo brz, postoje i nekoliko sivih točaka. Jedna je vezana uz još nedovoljno razvijenu teoriju heurističkih optimizacijskih postupaka. Kao primjer se može uzeti genetski algoritmi za koje se može reći da je teorija u zaostatku s praksom. Osim pojedinačnih detaljnih analiza primjena genetskih algoritama na jednostavne probleme, što znači na način da se dobiveni teorijski rezultati o radu algoritma mogu provjeriti eksperimentom, način djelovanja genetičkih algoritama je uvelike neistraženo područje. Navedeno nije neočekivano kad se uzme u obzir širina njihove primjene i različitosti reprezentacija s kojima mogu raditi što značajno otežava općenitu teorijsku analizu. U ovom primjeru, pod načinom djelovanja se misli na precizan matematički opis rada algoritma nad dobro definiranim matematičkim strukturama. Razlozi zbog kojih je „jasno“ da bi genetski algoritmi trebali biti kvalitetan optimizacijski postupak su prilično čvrsti te za većinu istraživača posve dovoljni. Međutim, rezultat toga je velik broj članaka koji istražuju heurističkih postupaka pristupaju isključivo s eksperimentalnog gledišta. No, potrebno je napomenuti da je kod egzaktnih matematičkih postupaka situacija gotovo posve obrnuta.

Druga siva točka na području heurističke optimizacije se svodi na eksperimentalno istraživanje i uspoređivanje različitih algoritama što zahtijeva puno ulaganja u njihovu izgradnju, definiranje testnih problema, izgradnju pomoćne programske infrastrukture za usporedbu i vizualizaciju rezultata te obavljanje mnogih aktivnosti koje nisu izravno povezane s izgradnjom što kvalitetnijih optimizacijskih postupaka, a što bi trebala biti osnovna specijalnost istraživača u tom području. Nepostojanje kompletnih programskih okruženja koja bi omogućavala jednostavnu primjenu heurističkih

metoda na širok skup problema, bilo u praktične ili istraživačko-znanstvene svrhe, predstavljaju veliki problem za istraživače u cjelini.

Iz navedenih razloga su kod primjene određene optimizacijske metode na dani problem znanstvenici vrlo često prisiljeni krenuti od nule, uz eventualno korištenje neke rudimentarne biblioteke razreda. Iako postoje i sofisticirane biblioteke razreda s ugrađenim mnogobrojnim optimizacijskim algoritmima (primjerice GALib, EOLib, JEO) od kojih neke predstavljaju i kompletno okruženje za provođenje optimizacije (primjerice HeuristicLab), česti nedostatak tih rješenja je slaba upotrebljivost i s tim je povezana visoka razina programerskog znanja potrebna za primjenu. Rješenja su najčešće izrađena korištenjem jezika C++, uz napredne tehnike predloška i generičkog programiranja, a „optimizacijsko okruženje“ najčešće čini C++ prevoditelj. Također, navedena okruženja pate i od problema nedovoljne fleksibilnosti i proširljivosti s obzirom na to da su uglavnom predviđena za korištenje samo nekih optimizacijskih metoda, primjerice genetičkih algoritama te da vrlo rijetko imaju ugrađene mogućnosti za virtualizaciju i analizu dobivenih rješenja. Kad se ima u vidu heuristički karakter optimiranja, upravo nedostatak ugrađenih mogućnosti za vizualizaciju i usporedbu rješenja predstavljaju veliki problem s obzirom na to da eksperimentiranje s različitim vrstama algoritama, operatora i reprezentacija varijabli predstavljaju važnu i vremenski zahtjevnu aktivnost prilikom primjene heurističkih metoda.

Optimizacijski problemi se dijele na kontinuirane i kombinatorne probleme. Kontinuirani problemi koriste realne varijable, a cilj im je otkriti vrijednosti tih varijabli za dobivanje najboljeg (optimalnog) rješenja. S druge strane, kombinatorni problemi rade s diskretnim varijablama. Mnogi kombinatorni problemi su problemi koji se javljaju u stvarnom svijetu (eng. real-life problems). Većina takvih problema spada u NP-teške probleme što znači da ih uobičajeni algoritmi koji bi se možda koristili za pretraživanje prostora stanja ne mogu riješiti u polinomijalnom vremenu. Poznati algoritmi za rješavanje takvih problema u najboljem slučaju su eksponencijalne složenosti. Jedan od takvih problema je i problem usmjeravanja vozila, tj. mnoge njegove inačice koje se razlikuju samo u ograničenjima koja koriste pri dobivanju (optimalnih) rješenja.

Neki od problema kombinatoričke optimizacije su problem trgovačkog putnika (TSP), Boolean satisfiability problem (SAT) te linearno programiranje, a neki od primjera će se obraditi u ovom radu.

1.1. Vrste problema optimizacije

Važan korak u procesu optimizacije je razvrstavanje modela optimizacije. Naime, algoritmi za rješavanje problema optimizacije su prilagođeni određenoj vrsti problema. Slijedi prikaz smjernica koje pomažu u klasifikaciji modela optimizacije (neosGuide, 2018).

1) Kontinuirana optimizacija (eng. Continuous Optimization) u suprotnosti s diskretnom optimizacijom (eng. Discrete Optimization).

Određeni modeli imaju smisla samo ako varijable uzimaju vrijednost iz diskretnog skupa, često podskup cijelih brojeva, dok drugi modeli sadrže varijable koje mogu uzeti bilo koju stvarnu vrijednost. Modeli s diskretnim varijablama su diskretni problemi optimizacije, a modeli s kontinuiranim varijablama su kontinuirani problemi optimizacije. Kontinuirani problemi s optimizacijom lakše se rješavaju od diskretnih problema s optimizacijom.

Glatkoća (savršenost) funkcije (eng. the smoothness of the functions) se odnosi na to da se objektivna funkcija i ograničavajuće vrijednosti funkcije u točci *(x,x)* mogu upotrijebiti za zaključivanje informacija o točkama u susjedstvu *(x,x).* Međutim, poboljšanja algoritama u kombinaciji s napretkom u računalnoj tehnologiji dramatično su povećala veličinu i složenost diskretnih problema optimizacije koji se mogu riješiti učinkovito. Kontinuirani optimizacijski algoritmi važni su u diskretnoj optimizaciji jer mnogi diskretni optimizacijski algoritmi su generiraju niz kontinuiranih potproblema.

2) Neograničena optimizacija (eng. Unconstrained Optimization) u suprotnosti s ograničenom optimizacijom (eng. Constrained Optimization)

Neograničeni problemi optimizacije nastaju izravno u mnogim praktičkim primjenama te se pojavljuju i u preoblikovanju ograničenih problema optimizacije u kojima se ograničenja zamjenjuju s članom u objektivnoj funkciji. Ograničeni problemi optimizacije proizlaze iz aplikacija u kojima postoje eksplicitna ograničenja na varijablama. Ograničenja varijabli mogu se široko razlikovati od jednostavnih granica

do sustava jednakosti i nejednakosti koji oblikuju kompleksne odnose između varijabli. Također, ograničeni problemi optimizacije mogu se dalje razvrstati prema prirodi ograničenja (primjerice, linearno ili konveksno) i glatkoći funkcija (primjerice, različito ili nejednako).

3) Optimizacija s niti jednom, jedinom ili više objektivnih funkcija

Većina problema s optimizacijom ima jednu objektivnu funkciju, no postoje slučajevi kada problemi optimizacije nemaju objektivnu funkciju ili višestruke objektivne funkcije. Problemi izvedivosti (eng. feasibility problems) su problemi u kojima je cilj pronaći vrijednosti za varijable koje zadovoljavaju ograničenja modela bez posebnog cilja optimizacije. Problemi komplementarnosti (eng. complementarity problems) prožimaju se ekonomijom te inženjeringom. Cilj je pronaći rješenje koje zadovoljava uvjete komplementarnosti. Na mnogim područjima, kao što su inženjering, ekonomija i logistika, pojavljuju se višestruki optimizacijski problemi (eng. multi-objective optimization), pri čemu se optimalne odluke trebaju poduzeti u nazočnosti kompromisa između dva ili više sukobljenih ciljeva. Primjerice, razvoj nove komponente može uključivati smanjenje težine dok povećava snagu ili odabir portfelja može uključivati maksimiziranje očekivanog prinosa, a istovremeno smanjuje rizik. U praksi, problemi s višestrukim ciljevima često se preoblikuju kao pojedinačni ciljevi stvaranjem težine kombinacije različitih ciljeva ili zamjenom nekih ciljeva prema ograničenjima.

4) Deterministička optimizacija u suprotnosti sa stohastičkom optimizacijom

U determinističkoj optimizaciji pretpostavlja se da su podaci za određeni problem točno poznati. Međutim, za mnoge stvarne probleme podaci se ne mogu točno znati iz više razloga. Prvi razlog je zbog jednostavne pogreške mjerenja. Drugi i temeljniji razlog je da neki podaci predstavljaju informacije o budućnosti (primjerice, potražnja za proizvodom ili cijena za buduće vremensko razdoblje) i jednostavno se ne može sa sigurnošću znati. U stohastičkoj optimizaciji (eng. stochastic optimization), nesigurnost je uključena u model. Robusne tehnike optimizacije (eng. robust optimization techniques) mogu se koristiti kada su parametri poznati samo unutar određenih granica, a cilj je pronaći rješenje koje je izvedivo za sve podatke i optimalno u nekom smislu. Stohastički modeli programiranja (eng. stochastic programming models) donose činjenicu da su distribucije vrijednosti koje upravljaju

podacima poznate ili se mogu procijeniti, a cilj je pronaći neku politiku koja je izvediva za sve (ili gotovo sve) moguće instance podataka i optimizira očekivane performanse modela.

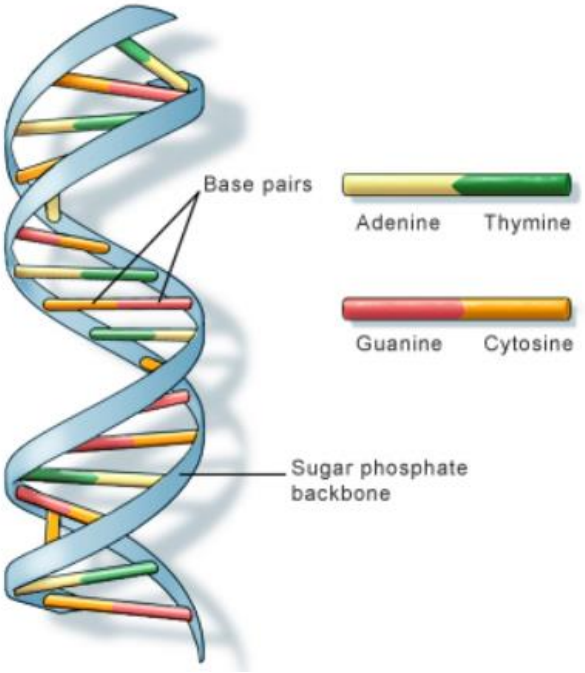
2. Evolucijski procesi u prirodi

Charles Darwin je uočio da živa bića u pravilu stvaraju više potomaka od cijele njihove populacije. Prema tome, broj jedinki koje čine populaciju bi trebalo eksponencijalno rasti iz generacije u generaciju. No, unatoč toj činjenici, broj jedinki jedne vrste teži ka konstantnom broju. Na temelju toga, Darwin je, primijetivši različitosti između jedinkama iste vrste, zaključio da prirodna selekcija jedinki regulira veličinu populacije. Naime, dobra svojstva jedinke (primjerice, otpornost na razne bolesti, sposobnost trčanja i sl.) pomažu da jedinka preživi u neprestanoj borbi za opstanak. Kako bi neka vrsta tijekom evolucije opstala, mora se prilagođavati uvjetima i okolini u kojoj živi jer se uvjeti i okolina mijenjaju te svaka sljedeća generacija neke vrste mora pamtiti dobra svojstva prethodne generacije, pronalaziti i mijenjati ta svojstva tako da ostanu dobra u neprekidno novim uvjetima.

Danas se pretpostavlja da su sva svojstva jedinke zapisana u kromosomima. Kromosomi su lančaste tvorevine koje se nalaze u jezgri svake stanice, a skup informacija koje karakteriziraju jedno svojstvo zapisano je u jedan djelić kromosoma koji se naziva gen. Potrebno je napomenuti da kromosomi dolaze uvijek u parovima (jedan je od oca, drugi od majke). Takav par gena gdje i jedan i drugi gen nose informaciju za jedno svojstvo naziva se alel.

Slika 1 prikazuje molekulu deoksiribonukleinske kiseline (DNK) u obliku dvije spirale građene od fosforne kiseline i šećera, a mostovi između spiralnih niti građeni su od dušičnih baza i to od adenina (A), gvanina (G), timina (T) i citozina (C). Pokazalo se da su upravo te dušične baze jedinice informacije. Slično kao što je u računalu najmanja jedinica informacije jedan bit, tako je u prirodi najmanja jedinica informacije jedna dušična baza (A, G, C ili T). Jedan kromosom se sastoji od dvije komplementarne niti DNK. Ovakva struktura dviju spiralnih niti DNK omogućava prenošenje informacije dijeljenjem kromosoma pri diobi stanice.

Slika 1.: Struktura DNK



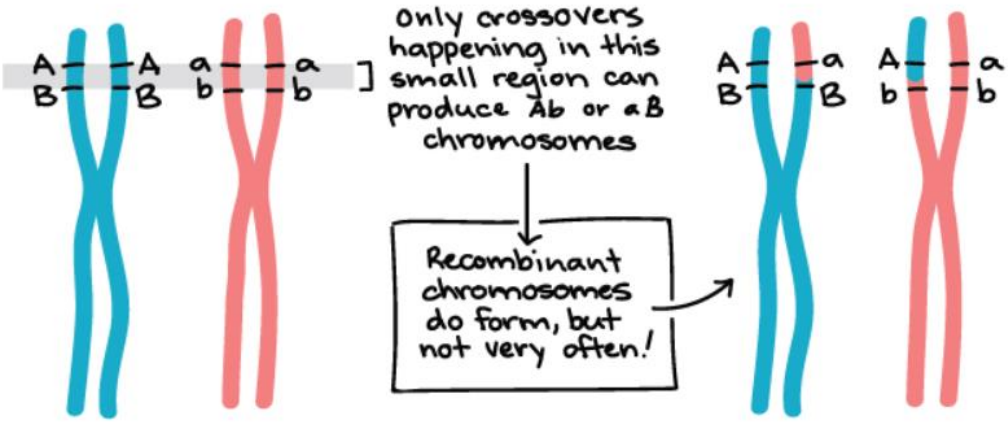
Izvor: Genetics Home reference, 2019

Evolucija se može objasniti kao prirodni proces traženja najbolje te najprilagodljivije jedinke na uvjete u prirodi, ali i samu okolinu. Stoga, za evoluciju, tj. prirodni evolucijski proces se može reći da je to metoda optimiranja. Što se tiče same ideje za genetski algoritma, može se naći u mehanizmu prirodnog odabira jer u prirodi iz skupa istobitnih pojedinaca preživljavaju oni koji su bolje, čak i najbolje prilagođeni snalaženju u okolini koja je siromašna sredstvima za život. Najbolji, tj. najsposobniji imaju priliku dominirati nad slabijima te se reproduciraju. Ukoliko se poistovjeti mjera sposobnosti pojedinca da preživi s naslijeđenim materijalom koji nosi u sebi, genima, tada se može reći da geni dominantnih pojedinaca opstaju, a geni slabijih izumiru jer nemaju potomstva. Uza sve navedeno, u prirodi kod svake reprodukcije dolazi do rekombinacije gena koja uzrokuje različitost između pojedinaca iste vrste, ali i sličnosti s roditeljima jedinke. Ukoliko se promatraju samo geni, može se zaključiti da se u svakoj novoj generaciji dobije novi skup gena gdje su neki pojedinci lošiji od onih u prethodnoj generaciji, a neki bolji. Izmjena gena koja nastaje pri reprodukciji u genetskim algoritmima naziva se križanje, iako navedeno nije sasvim u skladu s biologijom. Osim križanja, može se uočiti i još jedna pojava, a to je mutacija iako u

znatno manjem opsegu. Mutacija je slučajno mijenjanje genetskog materijala koje nastaje pod djelovanjem vanjskih uzroka. Križanje i mutacija, kod genetskih algoritama, nazivaju se genetskim operatorima, a proces izdvajanja najsposobnijih jedinki unutar svake generacije odabirom selekcije.

Križanjem se prenosi genetski materijal, a samim time i svojstva roditelja na djecu te svaka jedinka posjeduje genetski materijal oba roditelja. No, neka je broj kromosoma koje ima jedinka jednak *2n*. Sljedeća generacija jedinki treba također imati isti broj kromosoma, što znači da jedan roditelj daje polovičan broj kromosoma *n*. Navedeno se postiže mejozom, tj. sintezom spolnih stanica staničnom diobom. Za vrijeme diobe kromosoma, kromosomi se razmotaju u duge niti te se par kromosoma razdvoji, ali ne u potpunosti tako da se još drži zajedno u jednoj točki koja se naziva centromera nakon čega toga slijedi udvostručavanje kromosoma te se vežu slobodne baze s komplementarnim bazama na kromosomskim nitima. Nakon toga, između unutrašnjih niti (rjeđe i vanjskih) dolazi do izmjene segmenata kromosomskih niti. Slijedi prikaz slike koji prikazuje navedeni proces koji se naziva križanje ili izmjena faktora.

Slika 2.: Izmjena segmenata kromosomskih niti (eng. crossing-over)



Izvor: Khan Academy, 2019

Mutaciju se može opisati kao slučajnu promjenu gena, gdje neki geni mutiraju lakše, a neki teže stoga se dijele na stabilne i nestabilne gene. Vjerojatnost gena je konstantna, ali je različita za različite gene te treba napomenuti da vjerojatnost da gen *A* postane gen *B* nije ista kao i vjerojatnost da se gen *B* mutacijom promjeni u gen *A*. Mutaciju se može okarakterizirati i kao trajnu, ali i nasljednu ukoliko se dogodi u spolnoj stanici, promjenu genetskog materijala stanice (DNK ili RNK). Uzroci mutacija su mnogobrojni, poput greške kod umrežavanja genetskog materijala u procesu stanične diobe, izlaganje vanjskim čimbenicima poput radijacije, različiti kemijski spojevi ili virusi, namjerne mutacije tijekom mejoze i sl. Mutacije koje se dešavaju u nespolnim stanicama višestaničnog organizma (somatske mutacije) ne prenose se na potomstvo i mogu prouzročiti greške u odvijanju staničnih funkcija ili smrt stanica. S druge strane, mutacije u spolnim stanicama smatraju se jednim od preduvjeta evolucije jer se procesom prirodnog odabira u populaciji nakupljaju mutacije koje omogućuju bolju prilagodbu uvjetima okoliša i time utječu na bolje preživljavanje jedinke koje ih nose te prijenos na sljedeće generacije.

3. Genetski algoritam

Genetski algoritmi (eng. genetic algorithms, GA) su najpopularnija vrsta evolucijskih algoritama, a predstavljaju model optimizacije čije ponašanje potječe iz procesa evolucije koji se odvija u prirodi. Osnovni cilj genetskih algoritama je pronalaženje rješenja nekog problema koje tradicionalne determinističke metode ne mogu riješiti. Za razliku od većine determinističkih algoritama, genetski algoritmi pretragu ne započinju od jedne točke nego od cijelog niza potencijalnih rješenja. Ta potencijalna rješenja su najčešće generirana slučajnim odabirom, a predstavljaju početnu populaciju genetskog algoritma. Genetski algoritam je heuristička metoda optimizacije, a po načinu djelovanja, spada u metode usmjerenog slučajnog pretraživanja prostora rješenja (eng. guided random search techniques) u potrazi za globalnim optimumom.

Genetski algoritam (GA) predložio je John H. Holland 70-tih godina 20. stoljeća. Navedeni algoritam se pokazao moćnim i općenitim alatom za rješavanje čitavog niza problema iz inženjerske prakse. Naime, genetski algoritam služi za rješavanje teških optimizacijskih problema za koje ne postoji egzaktna matematička metoda rješavanja ili su NP-teški pa se za veći broj nepoznanica ne mogu riješiti u zadanom vremenu. Kao metoda optimizacije, genetski algoritam je nastao oponašanjem evolucije. Naime, simuliranje prirodnog evolucijskog procesa na računalu se svodi na grube aproksimacije rješenja (Golub, 2004).

Analogija evolucije kao prirodnog procesa i genetskog algoritma kao metode optimizacije očituje se u procesu selekcije i ostalim genetskim operatorima. U genetskom algoritmu, ključ selekcije je funkcija cilja koja na odgovarajući način predstavlja problem koji se rješava. Kod genetskog algoritma, jedno rješenje je jedna jedinka. Selekcijom se odabiru dobre jedinke koje se prenose u sljedeću populaciju te se manipulacijom genetskog materijala stvaraju se nove jedinke. Takav ciklus selekcije, reprodukcije i manipulacije magnetskim materijalom jedinki ponavlja se sve dok se ne zadovolji uvjet zaustavljanja evolucijskog procesa.

Može se reći da genetski algoritam simulira prirodni evolucijski proces. Za evolucijski proces, ali i za genetski algoritam može se reći da postoji populacija jedinki; neke jedinke su bolje, tj. bolje su prilagođene okolini; bolje jedinke imaju veću vjerojatnost preživljavanja i reprodukcije; svojstva jedinke zapisana su u kromosomima pomoću genetskog koda; djeca nasljeđuju svojstva roditelja; nad jedinkom može djelovati mutacija.

Snaga genetskih algoritama leži u činjenici da su oni sposobni odrediti položaj globalnog optimuma u prostoru s više lokalnih ekstrema u višemodalnom prostoru. S druge strane, klasične determinističke metode će se uvijek kretati prema lokalnom minimumu ili maksimumu pri čemu on može biti i globalni, ali se to ne može odrediti iz rezultata.

Genetski algoritam nije ovisan o nekoj početnoj točki te može svojim pretraživanjem s nekom vjerojatnošću locirati globalni optimum određene ciljne funkcije. No, za rezultat rada genetskog algoritma ne može se sa stopostotnom vjerojatnošću reći predstavlja li globalni ili samo lokalni optimum te je li isti određen sa željenom preciznošću. Sigurnost dobivenih rezultata značajno se povećava postupkom ponavljanja procesa rješavanja. Uza sve navedeno, genetski algoritam ne određuje na koji način je genetski materijal pohranjen u radni spremnik, niti kako treba manipulirati genetskim materijalom, već samo govori o tome da se genetski materijal treba razmjenjivati, slučajno mijenjati, a bolje jedinke trebaju s većom vjerojatnošću preživljavati selekciju. Genetski materijal se može opisati kao skup svojstva koji opisuje neku jedinku. Način na koji se bolje jedinke selektiraju za reprodukciju te kako će se i s kolikim vjerojatnostima obaviti križanje i mutacija, određuje se na temelju iskustva te eksperimentalno, tj. heuristički. Optimizacijom, tj. podešavanjem parametra genetskog algoritma mogu se postići zadovoljavajući rezultati. No, postupak podešavanja, tj. optimizacije je dugotrajan proces za rješavanje teških optimizacijskih problema s velikim brojem nepoznanica. Jedan od načina kako skratiti trajanje postupaka optimizacije jest paralelizacija genetskih algoritama.

Moguće je identificirati dva pristupa rješavanja određenog problema, a to su prilagoditi problem genetskom algoritmu ili genetski algoritam prilagoditi specifičnostima problema. Ukoliko se odabere prilagođavanje algoritma problemu, potrebno je modificirati njegov rad tako da rukuje s veličinama svojstvenim određenom zadatku (najčešće je ovdje riječ o upotrebi ili definiranju drugačijih

struktura podataka i operatora). Za veliki broj ovakvih slučajeva, razvijeni su posebni specijalizirani genetski algoritmi koji se tada nazivaju evolucijskim programima te se tada najčešće postižu velika povećanja djelotvornosti. No, ovakav je način problemski ovisan i zahtjeva mnogo rada na prilagođavanju te je ponekad problem ravan onome koji se želi riješiti, a za svaku novu promjenu potreban je nova prilagodba. Stoga, kompromis se postiže izradom evolucijskog programa koji će se moći primijeniti na čitavu klasu problema. S druge strane, ukoliko se želi zadani problem riješiti uz pomoć genetskog algoritma, tada je potrebno problem prilagoditi samom genetskom algoritmu. Moguće rješenje se najčešće pokazuje kao niz bitova (binarni prikaz). No, postoji čitav niz problema koje je teško ili nemoguće primijeniti binarni prikaz (primjerice, problem rasporeda). Uobičajeni genetski operatori mogu generirati značajan postotak nemogućih rješenja (primjerice, za problem najkraćeg puta) koji ne donose nikakva poboljšanja nego samo usporavaju algoritam. Osim binarnog prikaza, koriste se i razni drugi prikazi poput matrica, nizova realnih brojeva i sl. te se definiraju usko specijalizirani genetski operatori primjenjivi samo za određeni problem kako bi se izbjegla nemoguća rješenja. Primjenom prikaza i genetskih operatora dobiva se prilagođeni genetski algoritam problema koji se naziva evolucijski program.

Holland je predložio u svom radu jednostavan genetski algoritam kao računarski proces koji imitira evolucijski proces u prirodi te ga primjenjuje na apstraktne jedinke. Svaki evolucijski program odražava populaciju jedinki u nekoj određenoj generaciji te svaka jedinka predstavlja potencijalno rješenje problema koji se obrađuje. Svaka jedinka je predstavljena jednakom podatkovnom strukturom, primjerice broj, niz matrica, stablo, itd. Navedene jedinke se nazivaju kromosomi. Svakom rješenju se pridjeljuje određena mjera koja se naziva dobrota, a funkcija koja tu kvalitetu određuje naziva se funkcija cilja, tj. funkcija dobrote. Tada se formira nova populacija izdvajajući bolje jedinke iz skupa postojećih. Neki članovi nove populacije podvrgnuti su utjecajima genetskih operatora koji iz njih formiraju nove jedinke. Operatori se dijele na unarne te operatore višeg reda. Unarni operatori stvaraju nove jedinke mijenjajući manji dio genetskog materijala (mutacijska grupa), a operatori višeg rada kreiraju nove individue kombinirajući osobine nekoliko jedinki (grupa križanja). Nakon određenog broja izvršenih generacija, čitav postupak se zaustavlja kada se zadovolji

uvjet zaustavljanja, a najbolji član trenutne populacije predstavlja rješenje koje bi trebalo biti sasvim blizu optimuma (Golub, 2004).

Slijedi prikaz, tj. pseudokod koji prikazuje strukturu genetskog algoritma.

Slika 3.: Pseudokod genetskog algoritma

```
START
Generate the initial population
Compute fitness
REPEAT
    Selection
    Crossover
    Mutation
    Compute fitness
UNTIL population has converged
STOP
```

Izvor: Mallawaarachchi, 2017

Iz navedenog pseudokoda, može se izvući sljedeći:

Genetski_algoritam() {

t = 0;

generiraj početnu populaciju potencijalnih rješenja P(0);

sve dok nije zadovoljen uvjet završetka evolucijskog procesa;

{

t = t + 1;

selektiraj P´(t) iz P(t-1);

križaj jedinke iz P´(t) i djecu spremi u P(t);

mutiraj jedinke iz P(t);

}

ispiši rješenje;

}

Za genetski algoritam, jedinke su potencijalna rješenja, a okolinu predstavlja funkcija cilja. Također, za genetski algoritam kao metodu optimizacije potrebno je definirati proces dekodiranja i preslikavanja podataka zapisanih u kromosomu te odrediti funkcije dobrote. Prilikom inicijalizacije generira se početna populacija jedinki. Najčešće se početna populacija generira slučajnim odabirom rješenja iz domene iako je moguće početnu populaciju generirati uniformno[1] ili usaditi početno (inicijalno) rješenje u početnu populaciju dobivenom nekom drugom optimizacijskom metodom. Slijedi proces koji se ponavlja dok ne istekne vrijeme ili je zadovoljen neki od uvjeta. Navedeni proces se sastoji od djelovanja genetskih operatora selekcije, križanja te mutacije nad populacijom jedinki. Tijekom selekcije loše jedinke odumiru, dobre opstaju te se u sljedećem koraku križaju/razmnožavaju. Križanjem se prenose svojstva roditelja na djecu. Mutacijom se mijenjaju svojstva jedinke slučajnom promjenom gena. Navedenim postupkom se postiže iz generaciju u generaciju sve veća prosječna dobrota populacije (Golub, 2004).

3.1. Svojstva jedinke i inicijalizacija

Svi podaci koji obilježavaju jednu jedinku zapisani su u jednom kromosomu. Podaci mogu biti pohranjeni na razne načine u jednom kromosomu. Primjerice, podaci mogu biti pohranjeni binarnim prikazom (niz bitova koji čine jedan kromosom), kromosom može biti realan broj (float, double), a ukoliko je problem višedimenzijski, umjesto jednog broja treba biti polje brojeva veličine dimenzije problema.

Može se zaključiti da kromosom može biti bilo kakva struktura podataka koja opisuje svojstva jedne jedinke. Za genetski algoritam značajno je da kromosom predstavlja moguće rješenje na zadani problem. Za svaku strukturu podataka potrebno je definirati genetske operatore. Genetski operatori trebaju biti definirani na način da ne stvaraju nove jedinke koje predstavljaju nemoguća rješenja jer se time znatno umanjuje učinkovitost genetskog algoritma.

Uza sve navedeno, potrebno je napomenuti da se veličina populacije ne mijenja tijekom evolucije te se proces generiranja početne populacije sastoji od slučajno

[1] Sve jedinke su iste pa u početku evolucijskog procesa, genetski algoritam nije učinkovit te se taj postupak ne preporuča

odabranih vrijednosti unutar dozvoljenog intervala. Početna populacija može biti uniformna (sve su jedinke iste). U početnu populaciju se može dodati neko međurješenje dobiveno nekom drugom optimizacijskom metodom. Evolucijski proces se neprestano ponavlja sve dok se ne zadovolji neki uvjet. Budući da je vrijeme izvođenja pojedine iteracije približno konstantno, time je zadano i trajanje optimiranja.

Potrebno je spomenuti i funkciju dobrote ili funkciju ocjene kvalitete jedinke koja se često naziva još i fitness funkcija, funkcija sposobnosti ili funkcija cilja i u najjednostavnijoj interpretaciji ekvivalent je funkcije *f* koju treba optimizirati:

$$dobrota(v)=f(x)$$

gdje *v* predstavlja potencijalno rješenje iz populacije. Što je dobrota jedinke veća, jedinka ima veću vjerojatnost preživljavanja i križanja. Može se reći da je funkcija dobrote ključ za proces selekcije. U praksi se pokazalo da se transformacija može primijeniti samo neka ograničenja nad f(*x*). Za zadani optimizacijski problem najveću poteškoću predstavlja definiranje funkcije dobrote koja treba vjerno odražavati problem koji se rješava. Tijekom procesa evolucije, dobar genetski algoritam generira, iz generacije u generaciju, populacije čija je ukupna dobrota i prosječna dobrota sve bolja.

3.2. Postupci selekcije genetskog algoritma

Genetski algoritmi koriste selekcijski mehanizam za odabir jedinki koje sudjeluju u reprodukciji. Selekcija omogućava prenošenje boljeg genetskog materijala iz generacije u generaciju. Zajedničko svojstvo svih vrsta selekcija je veća vjerojatnost odabira boljih jedinki za reprodukciju. Postupci same selekcije se međusobno razlikuju po načinu odabira boljih jedinki. Prema načinu prenošenja genetskog materijala boljih jedinki u sljedeću generaciju, postupci selekcije se dijele na generacijske selekcije i eliminacijske selekcije. Kod generacijske selekcije, samom selekcijom se direktno biraju bolje jedinke čiji će genetski materijal prenijeti u sljedeću iteraciju. S druge strane, kod eliminacijske selekcije se biraju loše jedinke za eliminaciju, a bolje jedinke prežive postupak selekcije.

Generacijskom selekcijom se odabiru bolje jedinke koje kasnije sudjeluju u reprodukciji. Između jedinki iz populacije iz prethodnog koraka biraju se bolje jedinke

i kopiraju u novu populaciju. Nova populacija, koja sudjeluje u reprodukciji, naziva se međupopulacijom. Navedeno je ujedno i prvi nedostatak generacijskih selekcija jer se u radnom spremniku odjednom nalaze dvije populacije jedinki. Broj jedinki koje prežive selekciju je manji od veličine populacije. Najčešće se ta razlika u broju preživjelih jedinki i veličine populacije popunjava duplikatima preživjelih jedinki. Navedeno je drugi nedostatak generacijske selekcije jer duplikati nikako ne doprinose kvaliteti dobivenog rješenja, već samo usporavaju evolucijski proces. Generacijska selekcija postavlja granice između generacija. Svaka jedinka egzistira točno samo jednu generaciju. Izuzetak su jedino najbolje jedinke koje žive duže od jedne generacije i to samo ako se primjeni elitizam. S druge strane, kod eliminacijske selekcije, bolje jedinke preživljavaju mnoge iteracije pa stroge granice između generacije nema, kao što je to slučaj kod genetskih selekcija. Eliminacijska selekcija briše M[2] jedinki. Izbrisane jedinke nadomještaju se jedinkama koje se dobivaju reprodukcijom. Stoga, može se reći kako se eliminacijskom selekcijom otklanjaju nedostatka generacijske selekcije. Na prvi pogled, uvođenje novog parametra M predstavlja nedostatak eliminacijske selekcije jer se svakim dodatnim parametrom značajno produžava postupak podešavanja algoritma. Međutim, uvođenjem novog parametra nestaje parametar vjerojatnosti križanja.

Selekcijom se osigurava prenošenje boljeg genetskog materijala s većom vjerojatnošću u sljedeću iteraciju. Ovisno o metodi odabira boljih jedinki kod generacijskih selekcija, odnosno loših jedinki kod eliminacijskih selekcija, postupci selekcije se dijele na proporcionalne i rangirajuće. Zajedničko obilježje svim selekcijama je veća vjerojatnost preživljavanja bolje jedinke od bilo koje druge lošije jedinke. Selekcije se razlikuju prema načinu određivanja vrijednosti selekcije određene jedinke. Proporcionalne selekcije odabiru jedinke s vjerojatnošću koja je proporcionalna dobroti jedinke, odnosno vjerojatnost selekcije ovisi o kvocijentu dobrote jedinke i prosječne dobrote populacije. Rangirajuće selekcije odabiru jedinke s vjerojatnošću koja ovisi o položaju jedinke u poretku jedinki sortiranih po dobroti. Također, rangirajuće selekcije se dijele na sortirajuće i turninske selekcije. Turninske selekcije se dijele prema broju jedinki koje sudjeluju u turniru (Golub, 2004).

[2] Parametar M naziva se mortalitetom ili generacijskim jazom

3.3. Genetski operatori križanja, mutacije i selekcije

Reprodukcija je važna karakteristika genetskog algoritma. U reprodukciji sudjeluju dobre jedinke koje preživljavaju proces selekcije. Može se reći da je reprodukcija zapravo razmnožavanje pomoću genetskog operatora križanja. Tijekom procesa reprodukcije dolazi i do slučajnih promjena nekih gena ili mutacije te zamjene gena ili inverzije.

U procesu križanja (eng. crossover) sudjeluju dvije jedinke koje se nazivaju roditelji. Križanje je binarni operator te križanjem nastaje jedna ili dvije nove jedinke koje se nazivaju djeca. Najvažnija karakteristika križanja jest da djeca nasljeđuju svojstva svojih roditelja. Ukoliko su roditelji dobri (prošli su proces selekcije), tada će najvjerojatnije i dijete biti dobro, ako ne i bolje od svojih roditelja. Uniformno križanje predstavlja jednaku vjerojatnost nasljeđivanja svojstva roditelja. P-uniformno križanje predstavlja križanje gdje su vjerojatnost nasljeđivanja svojstva razlikuje za pojedine gene. Potrebno je spomenuti i segmentno križanje s više točaka prekida, broj točaka i pozicija prekida je slučajna za svako pojedino križanje te miješajuće križanje koje se obavlja se u tri koraka. Naime, izmiješaju se bitovi svakog roditelja, nakon čega ide klasično križanje s jednom ili više točaka te na kraju se izmiješani bitovi kod roditelja vrate se stara mjesta.

Pretpostavlja se da je upravo operator križanja to što razlikuje genetski algoritam od ostalih metoda optimizacije. Drugi operator koji je karakterističan za genetski algoritam je mutacija ili slučajna promjena jednog ili više gena. Mutacija je unarni operator jer djeluje nad jednom jedinkom. Rezultat mutacije je izmjena jedinka. Parametar koji određuje vjerojatnost mutacije p_m jednog bita je ujedno i parametar algoritma. Ukoliko vjerojatnost mutacije teži jedinici, tada se algoritam pretvara u algoritam slučajne pretrage prostora rješenja. S druge strane, ako vjerojatnost mutacije teži nuli, postupak će najvjerojatnije već u početku procesa optimizacije stati u nekom lokalnom optimumu.

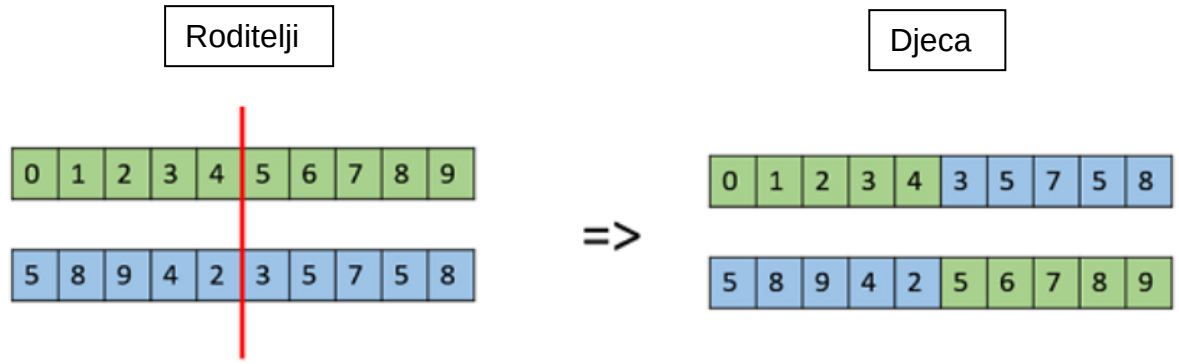
Križanjem u prirodi, genetski se materijal prenosi s roditelja na djecu. Novonastalo dijete posjeduje genetski materijal oba roditelja i ima jednak broj gena kao i oni, što znači da svaki roditelj daje samo pola svojih gena djetetu.

Isti se princip primjenjuje i u genetskim algoritmima. Procesom križanja (eng. crossover) nastaju nove jedinke koje imaju kombinirane informacije sadržane u dvoje ili više roditelja. Križanje se često naziva i rekombinacija (eng. recombination). Odabir križanja za genetski algoritam ovisi o odabranom načinu prikaza rješenja. Slijedi opis najpoznatijih izvedba križanja za jednostavni prikaz rješenja binarnim brojevima (Tutorialspont simplyeasylearning, 2019):

1) Križanje u jednoj točki
2) Križanje u dvije točke
3) Križanje rezanjem i spajanjem
4) Uniformno križanje
5) Polu-uniformno križanje (eng. half-uniform crossover)

Kod križanja u jednoj točki (eng. one point crossover), odabire se jedna točka na roditeljima (jednako udaljena od početka za oba roditelja) i mijenja se desni dio kromosoma prvog roditelja s desnim dijelom kromosoma drugog roditelja što prikazuje i sljedeća slika.

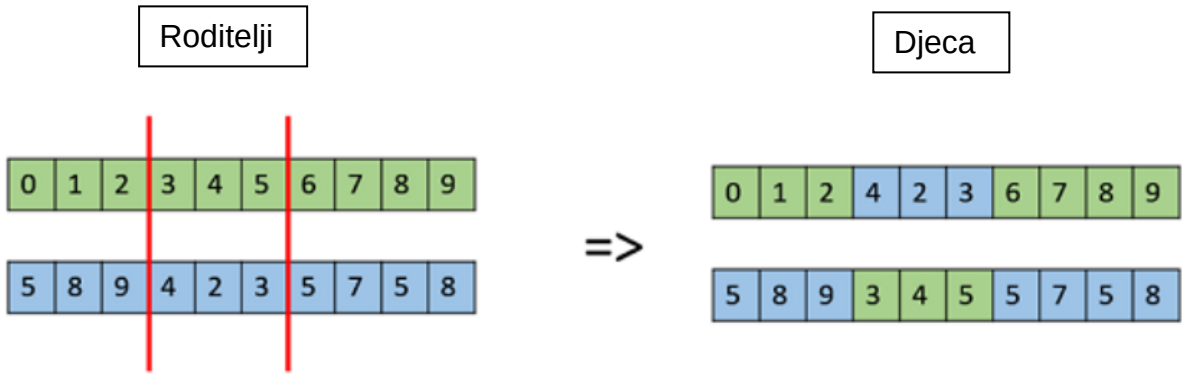
Slika 4.: Križanje u jednoj točki



Izvor: Tutorialspont simplyeasylearning, 2019

Kod križanja u dvije točke (eng. two point crossover), odabiru se dvije točke na roditeljima pazeći da je pritom lijeva točka jednako udaljena od lijevog kraja i desna točka od desnog kraja kromosoma za oba roditelja. Roditelji izmjenjuju dijelove kromosoma između tih točaka. Sljedeća slika prikazuje križanje u dvije točke.

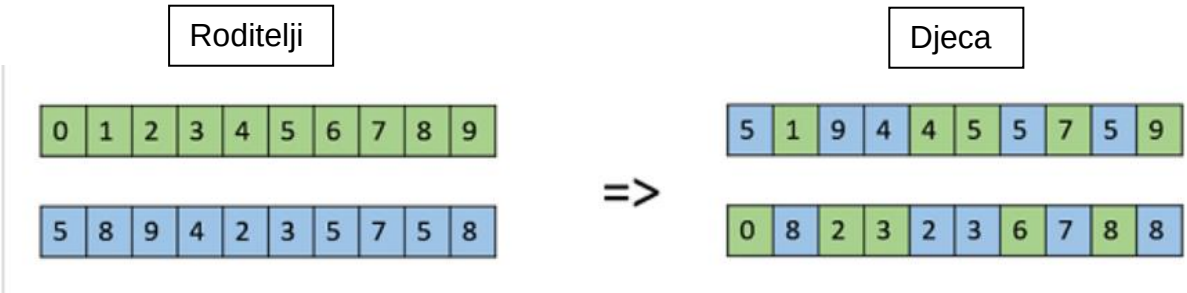
Slika 5.: Križanje u dvije točke



Izvor: Tutorialspont simplyeasylearning, 2019

Kod uniformnog križanja (eng. uniform crossover), uspoređuju se bit po bit oba roditelja i mijenjaju se s fiksnom vjerojatnošću od 50%. S druge strane, kod polu-uniformnog križanja (eng. half-uniform crossover), uspoređuje se bit po bit oba roditelja i računa se koliko ih se ne poklapa. Polovica od tih koji se ne poklapaju se mijenja. Potrebno je napomenuti kako se u uniformnom križanju, kromosomi ne dijele na segmente nego ih se tretira kao svaki gen zasebno te se u tom slučaju zapravo prebacuje novčić na svaki kromosom roditelja kako bi se odlučilo hoće li ili neće biti uključen u kromosom djeteta. Stoga, može se zaključiti kako novčić može presuditi u korist jednog roditelja gdje će kromosom djeteta imati više genetskog materijala jednog roditelja.

Slika 6.: Uniformno križanje



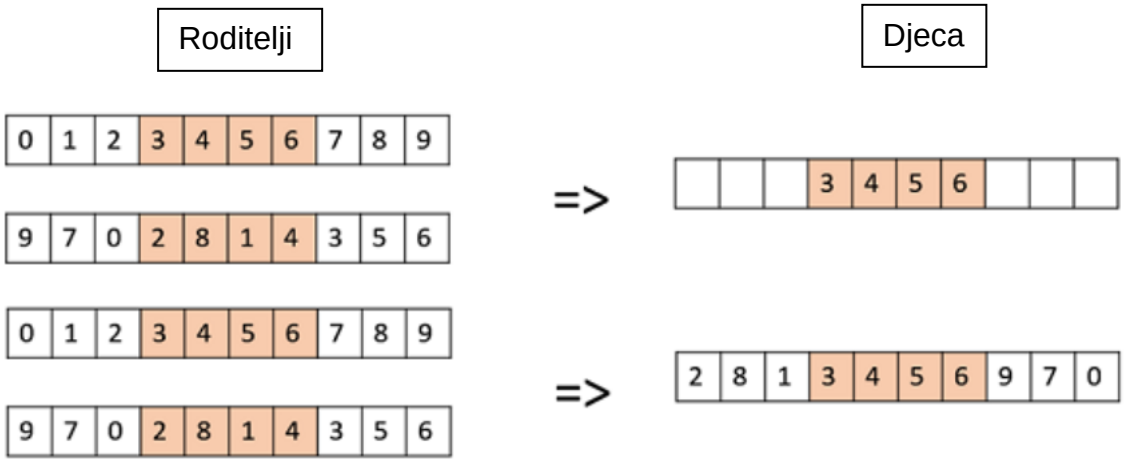
Izvor: Tutorialspont simplyeasylearning, 2019

Potrebno je spomenuti i križanje Davidsovim redoslijedom (eng. Davis' Order Crossover, OX1) te križanje rezanjem i spajanjem.

Davidsov redoslijed se koristi za permutacije koje su bazirane na križanju s namjerom prijenosa informacija o relativnom poretku izvora te radi na sljedeći način (Tutorialspont simplyeasylearning, 2019):

- Stvaraju se dvije slučajne točke na kromosomu roditelja i segment se kopira između njih od prvog roditelja na prvog potomka.
- Nakon navedenog, počinje se od druge točke drugog roditelja; omatajući se oko liste, kopiraju se preostali neiskorišteni brojevi drugog roditelja prvog djeteta
- Ponavlja se postupak za drugo dijete kojemu je uloga obrnuta.

Slika 7.: Križanje Davidsovim redoslijedom (OX1)



Izvor: Tutorialspont simplyeasylearning, 2019

Kod križanja rezanjem i spajanjem (eng. cut and splice crossover), na roditeljima se odabire točka koja nije jednako udaljena od početka roditeljskih kromosoma. Mijenja se desni dio kromosoma jednog roditelja s desnim dijelom kromosoma drugog. Na taj način nastaju djeca koja imaju različite duljine kromosoma.

Mutacijom se pretražuje prostor rješenja i upravo je mutacija mehanizam za izbjegavanje lokalnih minimuma. Ukoliko cijela populacija završi u nekom od lokalnih minimuma, jedino slučajnim pretraživanjem prostora rješenja pronalazi se bolje

rješenja. Dovoljno je da jedna jedinka (nastala mutacijom) bude bolja od ostalih, pa da se u nekoliko sljedećih generacija sve jedinke presele u prostor gdje se nalazi bolje rješenje. Uloga mutacije je također i u obnavljanju izgubljenog genetskog materijala. Dogodi li se primjerice da sve jedinke populacije imaju isti gen na određenom mjestu u kromosomu, samo križanjem se taj gen nikada ne bi mogao promijeniti.

U prirodi se mutacija definira kao slučajna promjena gena. Vjerojatnost promjene za gene su različite, pa geni mogu biti stabilni, tj. nestabilni. Vjerojatnost da gen *A* postane gen *B* nije ista kao da gen *B* postane gen *A*.

U genetskim algoritmima, mutacija (eng. mutation) je genetski operator koji se upotrebljava za dobivanje genetske raznolikosti sljedeće generacije rješenja od one postojeće. Najjednostavniji primjer mutacije je vjerojatnost da neki bit u genetskom kodu (rješenju) promjeni iz svog originalnog stanja u novo stanje. Navedeno se postiže uvođenjem neke varijable za svaki bit u nizu. Ta varijabla govori hoće li bit biti modificiran ili neće. Ukoliko se koristi binaran prikaz rješenja, mutacija je vjerojatnost da neki bit iz nule prijeđe u jedinicu ili iz jedinice u nulu (slika 8). Mutacija sprječava rješenja da postanu preslična i na taj način uspore ili zaustave evoluciju. Kada se mutacija ne bi koristila, rješenja genetskog algoritma najčešće bi konvergirala prema nekom lokalnom optimumu.

Slika 8.: Mutacija

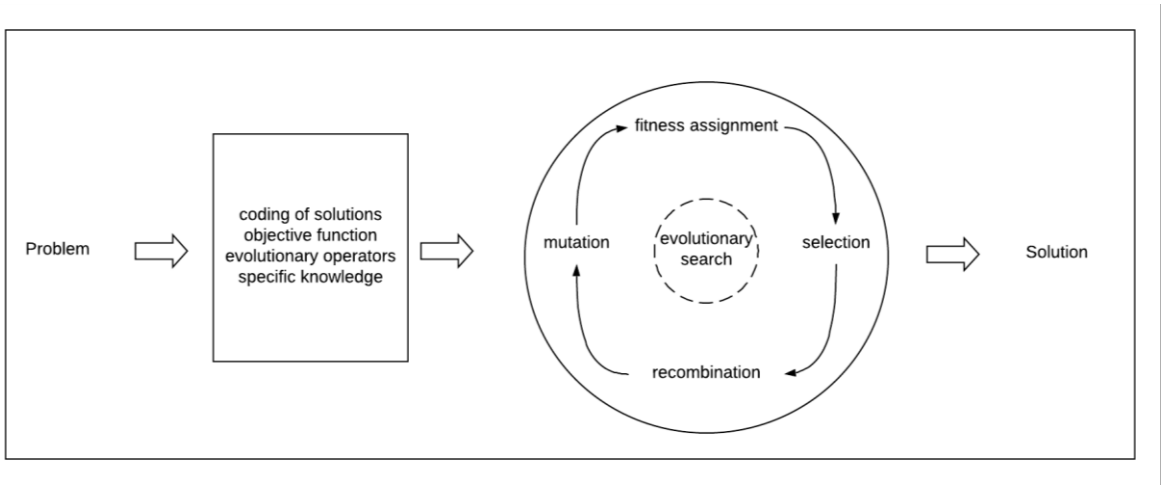
Izvor: Tutorialspoint simplyeasylearning, 2019

U prirodi preživljavaju jedinke koje se prilagode novim okolnosti. Naime, jedinke s vanjskim karakteristikama bolje prilagođenim okolišu imaju veće šanse za preživljavanje i povećanje svoje vrste. S druge strane, manje prilagođene jedinke izumiru, uzrokujući tako izumiranje čitavih vrsta. Na taj način priroda provodi prirodnu selekciju i osigurava da geni bolje prilagođenih jedinki imaju veće šanse za

preživljavanje od onih koji su lošije prilagođeni. Upotrebom tog osnovnog biološkog modela nastali su evolucijski algoritmi (eng. evolutionary algorithms) čiji su dio genetskih algoritama.

Početnoj se populaciji određuje dobrota (eng. fitness) što predstavlja mjeru kvalitete te govori koliko je neko rješenje dobro (bolja rješenja će imati veću dobrotu i obrnuto. Dobrota se određuje funkcijom cilja (funkcijom dobrote) koja ovisi o problemu koji se rješava. Kada se odredi dobrota, može započeti genetski proces. Početna populacija rješenja se iterira kroz generacije, a primjenjuje se princip pretraživanja najspremnijeg (eng. survival of the fittest). U svakoj se generaciji odabiru bolja rješenja i odbacuju ona lošija. Odabrana rješenja su podvrgnuta genetskim operatorima. Najprije se križaju, a zatim s određenom vjerojatnošću i mutiraju. Na takav način nastaju potencijalna rješenja koja predstavljaju novu generaciju genetskog algoritma. Ta nova rješenja su u pravilu bolja nego stara rješenja od kojih su nastala (Golub, 2004). Cijeli se proces temelji na prirodnom procesu evolucije što prikazuje slika 9.

Slika 9.: Rješavanje problema genetskim algoritmom



Izvor: Pohlheim, 2006, str. 1

Stvaranje novih rješenja se nastavlja dok se ne zadovolji kriteriji završetka genetskog procesa koji je definirao korisnik što prikazuje već spomenuti pseudokod genetskog algoritma. Kada se kriteriji zadovolji, genetski algoritam je završio s radom. Međutim, to ne znači da je pronađeno optimalno rješenje koje može, ali i ne mora biti najbolje. Završetak rada genetskog algoritma znači samo da je nađeno dovoljno dobro rješenje. Ukoliko se koristi elitizam u genetskom algoritmu, najbolja jedinka iz

trenutne generacije ide direktno u sljedeću bez križanja i mutiranja. Unatoč tome, ta jedinka može sudjelovati i u stvaranju djece. Može se koristiti elitizam jedne jedinke ili elitizam više jedinki.

Selekcija (eng. selection) je odabir jedinki koje stvaraju potomstvo te se na taj način čuvaju i prenose dobra svojstva (dobri geni) na sljedeću generaciju jedinki. Selekcija se vrši prema dobroti gdje jedinke s većom dobrotom imaju veće šanse da budu odabrane za roditelje u sljedećoj generaciji. Vjerojatnost odabira jedinke ovisi o njenoj dobroti, ali i o dobroti ostalih jedinki. Što je jedinka bliže ciljanom rješenju u odnosu na druge jedinke, njezina vjerojatnost razmnožavanja se povećava. Na temelju navedenoga, ukoliko je iznos dobrote jedinke manji u odnosu na onaj drugih jedinki, njezine mogućnosti preživljavanja se smanjuju.

Postupak selekcije može se ostvariti jednostavnim odabirom N[3] najboljih jedinki, ali takav odabir bi doveo do prerane konvergencije rješenja. Problem je u tome što se takvim odabirom gubi dobar genetski materijal koji mogu sadržavati loše jedinke te iz tog razloga i loše jedinke moraju imati određenu vjerojatnost preživljavanja (veću od nule).

Postoji mnogo načina odabira roditelja (eng. parent selection scheme) za sljedeću generaciju, a najpoznatiji su proporcionalni odabir (eng. proportionate selection), odabir po rangu (eng. ranking selection) te turnirski odabir (eng. tournament selection). Kod proporcionalnog odabira, jedinke se biraju proporcionalno prema dobroti. Jedinka s najvećom dobrotom ima najveće šanse da bude izabrana, dok najlošija jedinka ima najmanje šanse preživljavanja. Vjerojatnost preživljavanja ostalih jedinki je negdje između. Kod odabira po rangu, populacija se sortira od najboljih jedinki prema najgorima. Broj kopija koje jedinka dobije određuje se funkcijom (eng. assignment function) i proporcionalna je rangu jedinke. Rang jedinke je pozicija u odnosu na generaciju. Najbolja jedinka ima rang 1, dok najlošija ima rang N što predstavlja veličinu populacije. Kod turnirskog odabira, odabire se slučajan broj jedinki iz populacije (s ili bez zamjena, ovisno o implementaciji), a najbolje od njih postaju roditelji sljedeće generacije. Postupak se ponavlja dok se ne popuni cijela sljedeća generacija, tj. dok broj odabranih jedinki ne postane jednak

[3] N je veličina populacije

veličini populacije. Ovaj se odabir često naziva i k-turnirski odabir, gdje *k* označava broj jedinki čije se dobrote uspoređuju te se taj broj naziva veličina prozora.

3.4. Parametri genetskog algoritma

Genetski algoritam ima sljedeće parametre (Golub, 2004, str. 17):

1) Veličina populacije
2) Broj generacija ili iteracija
3) Vjerojatnost mutacije
4) Vjerojatnost križanja – za generacijski genetski algoritma
5) Broj jedinki za eliminaciju – za eliminacijski genetski algoritam umjesto vjerojatnosti križanja

Za različite vrijednosti parametra algoritma, algoritam daje različite rezultate (brže ili sporije dolazi do boljeg ili lošijeg rješenja). Optimizacija parametra genetskog algoritma je složen postupak i zahtjeva izvođenje velikog broja eksperimenata. No, dovoljno je optimizirati parametre poput veličine i vjerojatnosti populacije, vjerojatnosti križanja te broj jedinki za eliminaciju.

Parametri genetskog algoritma ne moraju biti konstantni za vrijeme evolucije. Dinamični parametri mogu biti funkcija vremena ili broj iteracija, ili mogu biti funkcija veličine raspršenosti rješenja. Ukoliko su rješenja raspršena, treba povećati učestalost križanja, a smanjiti mutaciju. No, ukoliko su rješenja uniformna, potrebno je učiniti upravo suprotno.

Za optimiranje višemodalnih funkcija, genetski algoritmi trebaju posjedovati dvije važne karakteristike, a to su sposobnost konvergiranja k optimumu (lokalni ili globalni) nakon pronalaženja područja u kojem se nalazi te sposobnost da se konstantno istražuju nova područja problemskog prostora u potrazi za globalnim optimumom.

Ukoliko genetski algoritam upotrebljava samo križanje, on će moći konvergirati prema ekstremu kod kojega se zatekao. Ukoliko postoji samo mutacija kao genetski operator, genetski algoritam će nakon dovoljnog broja generacija vrlo vjerojatno naći područje globalnog optimuma. Strategija prilagodbe traži kompromis između ove

dvije krajnosti mijenjajući vjerojatnost mutacije i križanja, ovisno o trenutnom stanju genetskog algoritma.

3.5. Prikaz rješenja

Svojstva živih bića zapisana su u njihovim kromosomima. Kao što je već i spomenuto, kromosomi su lančaste tvorevine koje se nalaze u jezgri svake stanice, a sastoje se od molekula DNA koje su nositelji informacija. Skup informacija koje karakteriziraju jedno svojstvo naziva se gen. Živa bića dobivaju jedan kromosom od oca, a jedan od majke što znači da se svako svojstvo postoje dva gena. Ti geni mogu biti ravnopravni ili jedan može biti dominantan, a drugi recesivan. Ukoliko su geni ravnopravni, svojstvo potomaka je negdje između oca i majke, a ako geni nisu ravnopravni onda potomak zadržava svojstvo dominantnog gena.

Postavlja se pitanje kako preslikati ove prirodne operacije na model strojnog učenja, tj. kako prikazati informaciju (rješenje) u genetskom algoritmu. Rješenja se najčešće prikazuju nizom bitova i binarnim brojevima. Glavna prednost takvog prikaza je u tome što se njegovi dijelovi zbog fiksne veličine mogu lako poravnati i uskladiti što omogućuje jednostavnu operaciju križanja. Na isti način se mogu koristiti i druge, složenije strukture poput matrica ili stabla. Korištenje rješenja različite duljine otežava operaciju križanja. Kod problema određivanja rasporeda, rješenja se najčešće prikazuju nizovima cijelih brojeva.

3.6. Kriteriji završetka

Genetski algoritam ponavlja proces generiranja sljedeće generacije sve dok se ne zadovolji kriteriji završetka evolucijskog procesa koji se još naziva i uvjet kraja ili kriteriji kraja/završetka. Kriteriji završetka je najčešće jedan od sljedećih (Milajić i sur., 2010, str. 754):

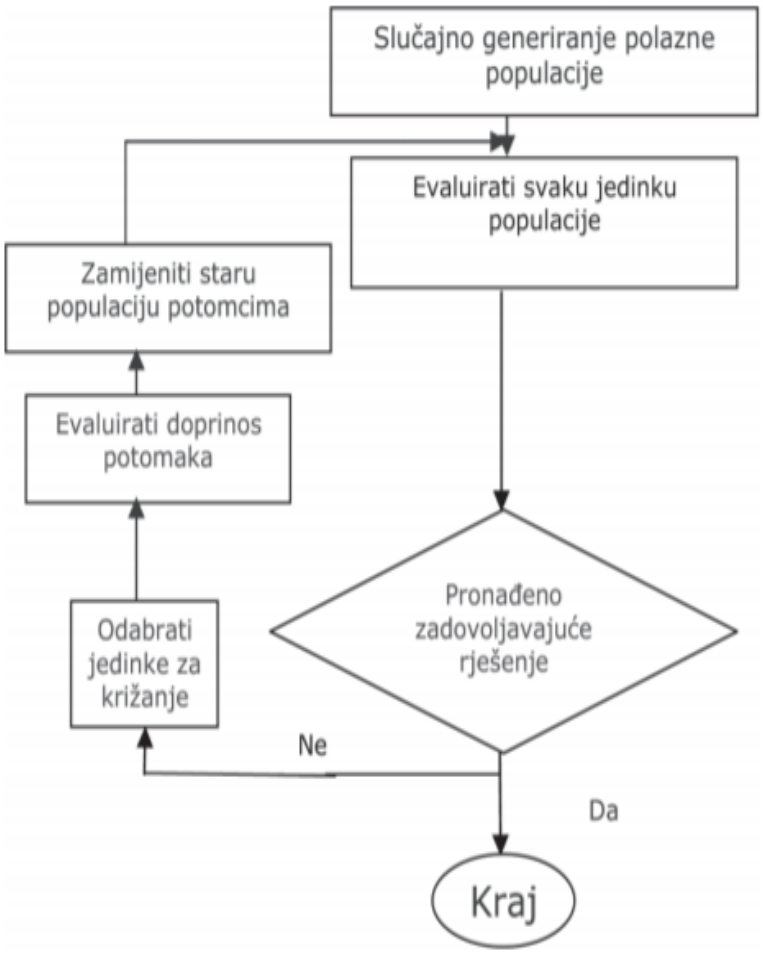
1) Pronađeno je rješenje koje zadovoljava minimalne kriterije
2) Dosegnut je unaprijed određeni broj generacija
3) Potrošen je budžet (vrijeme ili novac)
4) Rješenje s najvećim stupnjem dobrote dostiže ili je već dostiglo takvu granicu da daljnje iteracije ne daju bolje rezultate
5) Ručno ispitivanje
6) Kombinacije gore navedenih

3.7. Višepopulacijski genetski algoritam

Genetski algoritmi mogu biti jednopopulacijski i višepopulacijski (eng. multi-population). Višepopulacijski genetski algoritmi imaju više populacija pa se u njima javljaju neki novi genetski operatori kao što su ubacivanje (eng. reinsertion), migracija (eng. migration) i natjecanje (eng. competition). Svaka se populacija razvija izolirano nekoliko generacija prije nego se pojedine jedinke između populacija zamijene. Postupak zamjene naziva se migracija. Pri natjecanju se dostupna sredstva dijele tako da uspješnije populacije dobivaju više, a one manje uspješne manje sredstva. Ubacivanje je se može objasniti kao ubacivanje jedinki iz jedne populacije u drugu.

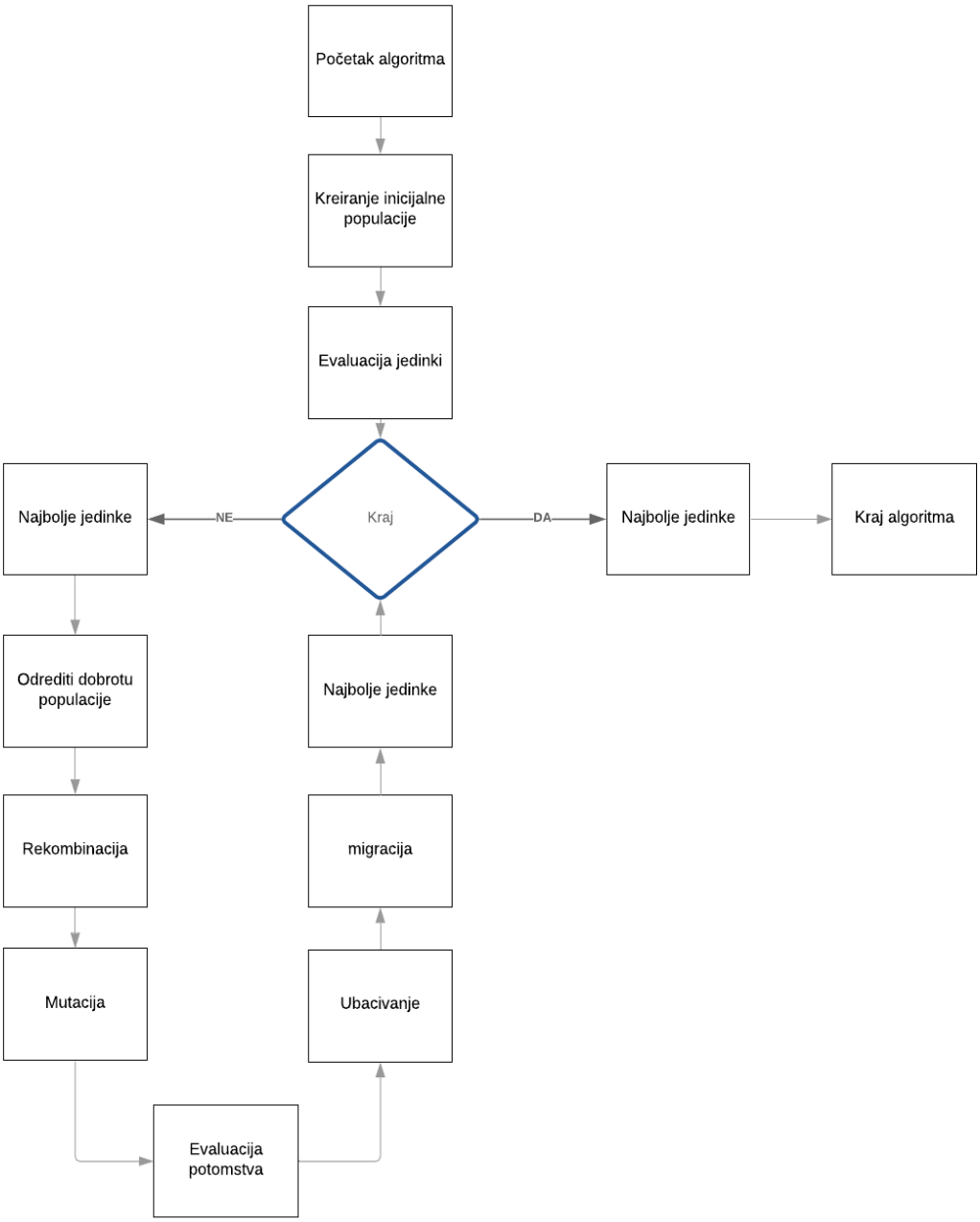
Višepopulacijski genetski algoritmi sličniji su prirodnom tijeku evolucije (slike 10 i 11) pa samim time daju bolje rezultate nego jednopopulacijski. S obzirom na to da imaju više populacija u svakoj generaciji, zahtijevaju više memorije i usporavaju brzinu iteriranja po generacijama.

Slika 10.: Višepopulacijski genetski algoritam



Izvor: Markić, 2008, str. 269

Slika 11.: Detaljniji prikaz višepopulacijskog genetskog algoritama



Izvor: rad autorice

4. Problem trgovačkog putnika

Problem trgovačkog putnika (eng. Traveling Salesman Problem, TSP) je problem diskretne i kombinatorne optimizacije koji spada u skupinu NP-teških problema čija je složenost O(n!). Matematičke probleme slične problemu trgovačkog putnika počeo je razmatrati Euler kojega je zanimalo kako bi skakač na šahovskoj ploči posjetio sva 64 mjesta samo jednom. Početkom 20. st. matematičari William Rowan Hamilton i Thomas Kirkman su razmatrali probleme koji se svode na problem trgovačkog putnika, a njegova opća formula se pojavljuje 30-tih godina 20. st. Pojam „trgovački putnik“ prvi put je upotrijebljen 1932. godine. Može se reći kako je problem trgovačkog putnika jedan od najčešće proučavanih problema u matematičkoj optimizaciji te jedan je od najpoznatijih problema algoritamske teorije brojeva. Naime, mogućnost da se ovaj problem primijeni na različite ljudske aktivnosti je ono što čini jedno od najvažnijih prepoznatljivih matematičkih problema bez, do sad, idealnog rješenja. Ovaj problem spada u kategoriju NP-teških problema i ima faktorijelnu složenost stoga predstavlja izazov i za ljude i za računala. Zbog njegove složenosti, tradicionalne metode rješavanja nisu se pokazale pretjerano uspješnim. Iz navedenog razloga na red dolaze genetski algoritmi, stohastičke metode pretraživanja koje oponašaju prirodni tijek biološke evolucije. Prednost genetskih algoritama prilikom rješavanja ovakvog problema i njemu sličnih problema je da ne započinju pretragu od jedne točke nego od niza točaka koje predstavljaju početnu populaciju (Bryant, 2010).

Sam problem je vrlo jednostavan. Trgovački putnik ima unaprijed definirane gradove i sve međusobne udaljenosti između njih te mora posjetiti svaki grad samo jednom i vratiti se u početni grad. Iz navedenog se može postaviti problem koji uključuje pitanje kojim bi redoslijedom trgovački putnik morao obilaziti gradove, a da ukupna duljina puta bude minimalna, tj. potrebno je da trgovački putnik prođe kroz određene gradove te se vratiti na polaznu točku njegovog putovanja što je moguće uz najkraće vrijeme putovanja. No, umjesto da se ispituje najkraća/najbrža ruta, može se ispitati i ona ruta koja košta najmanje, ona koji omogućuje najveći protok informacija i sl. Može se zaključiti kako se ovaj problem ne čini preteškim, ali ako se uzme u obzir da ima faktorijelnu složenost, računanjem se dobije da već za 10 gradova postoji 1 814 400 mogućih redoslijeda obilazaka. Naime, problem trgovačkog putnika po svojoj

prirodi spada u kategoriju NP-potpunih problema što znači da nema algoritma koji bi pružio rješenje problema u smislu polinomnog vremena, no ukoliko postoji rješenje, može se testirati i zaključiti je li optimalno.

Ukratko, u rješavanju problema trgovačkog putnika, potrebno je riješiti optimizacijski problem samog trgovačkog putnika. Ukoliko trgovac treba obići nekoliko gradova, a u svakom se zadržava samo jednom te se na kraju vrati u početni grad, potrebno je naći najkraći put kojim on to može učiniti.

Sama populacija se iterira kroz generacije nakon čega nastaju sve bolja rješenja, a genetski algoritam se zaustavlja kada nađe rješenje koje zadovoljava kriteriji optimizacije, ali to rješenje ne mora nužno biti optimalno. Problem trgovačkog putnika vrlo je važan jer se na njega svode mnogi drugi problemi, uključujući problem postavljanja sklopovskih pločica u kojem treba osigurati konstante intervale između slanja signala; proces dobivanja sirovina, primjerice papira iz drva; grupiranje podataka iz velikih polja; analiziranje kristalnih struktura; raspoređivanja redoslijeda poslova i sl.

Kod problema trgovačkog putnika zadatak je pronaći najmanji Hamiltonov ciklus u grafu. Ovaj problem spada u NP-teške (eng. NP-hard) probleme gdje je broj stanja otprilike (n-1)!. Algoritmi koji mogu dati točna rješenja za ovaj problem su pretraga grubom silom (engl. brute force (enumeracija)) čija je složenost O(n!), dinamičko programiranje čija je složenost O(n^2+2^n) te grananje i ograničavanje (engl. Branch and Bound) koji radi za 40 do 60 gradova.Godine 2006. je točno riješen graf sa 85,900 gradova (Michalewicz i Fogel, 2004).

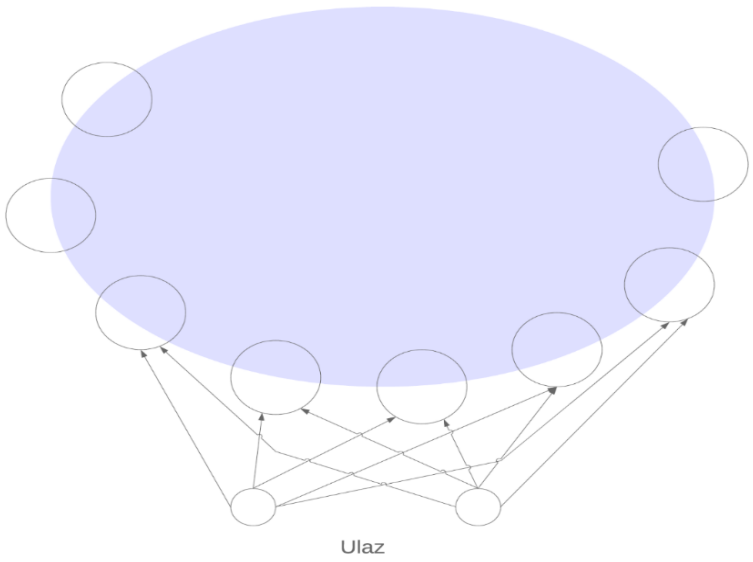
Postoji mnogo problema koji su povezani s problemom trgovačkog putnika ili se svode na njega. Neki od sličnih problema su pronaći Hamiltonov ciklus najmanje težine u težinskom grafu, problem trgovačkog putnika s uskim grlom te opći problem trgovačkog putnika.

Kod problema kako pronaći Hamiltonov ciklus najmanje težine u težinskom grafu, Hamiltonov ciklus (eng. Hamilton cycle) predstavlja ciklus koji u težinskom grafu posjećuje svaki čvor samo jednom i vraća se u početni čvor. Razlika od problema trgovačkog putnika je u tome da svi čvorovi ne moraju biti međusobno povezani, dok su kod trgovačkog putnika svi gradovi povezani. Kod problema trgovačkog putnika s uskim grlom (eng. Bottleneck Traveling Problem) potrebno je pronaći Hamiltonov

ciklus u težinskom grafu, ali takav da udaljenost na najtežem bridu bude minimalna. Ovaj problem ima veliku praktičnu važnost za prijevoznike i logističare, a javlja se i prilikom određivanja redoslijeda poslova. Kod općeg problema trgovačkog putnika (eng. Generalized Traveling Salesman Problem), trgovački putnik ima na raspolaganju gradove, ali i države te mora posjetiti točno jedan grad iz svake države, a da prevaljeni put bude minimalan. Ovako definiran problem često se naziva i problem putujućeg političara (eng. Traveling Politician Problem), a može se svesti na obični problem trgovačkog putnika s istim brojem gradova, ali modificiranim udaljenostima.

Za rješenje problema može se koristiti Kohonenova neuronska mreža gdje je jednodimenzionalni niz neurona topološki organiziran u obliku prstena, prema slici 12.

Slika 12.: Kohonenova mreža za rješenje problema trgovačkog putnika



Izvor: rad autorice

Mreža ima dva ulaza na koje se dovode koordinate gradova. Na ulaz mreže se naizmjence postavljaju koordinate gradova i taj se proces iterativno ponavlja. Onaj neuron čiji vektor težina je najbliži koordinatama grada koji je trenutno na ulazu je pobjednik u kompetitivnom učenju. Vektor težina pobjedničkog neurona mijenja se u smjeru vektora koordinata grada. Prema Kohonenovom principu, osim pobjedničkog

neurona vektore težina mijenjaju i susjedni neuroni, ali s manjim iznosom. Najjače se modificiraju vektori težina najbližih susjeda, a najslabije najdaljih. Promjena iznosa težina susjednih neurona definirana je Gaussovom funkcijom. Rezultat je da će se neuroni u prostoru težina pomicati prema koordinatama gradova. Neurona ima dva puta više nego gradova zbog toga da se višak neurona postavi između dva susjedna grada. U protivnom, jedan neuron bi mogao oscilirati između ta dva grada.

Sam algoritam ne garantira da je dobiveni rezultat optimalan, tj. moguće je da rezultat predstavlja lokalni, a ne globalni minimum. Rezultat ovisi i o početnim vrijednostima težina neurona, a eksperiment se može ponoviti i za veći broj gradova (primjerice, 300).

4.1. Metode rješavanja

Zbog svoje faktorijelne složenosti, problem trgovačkog putnika predstavlja pravi izazov za rješavanje. Pojavilo se mnogo algoritama za rješavanje, a izumom računala uspješnost se uvelike popravila. Neki algoritmi za rješavanje navedeni su u nastavku.

4.1.1. Pretraga grubom silom

Pretraga grubom silom (eng. brute force search) je najjednostavniji način rješavanja problema trgovačkog putnika. Međutim, zbog svoje jednostavnosti ovo je ujedno i najlošiji (najsporiji) način. Ideja je da se jednostavno isprobaju sve permutacije i pamti najbolja kombinacija. S obzirom na to da je složenost problema faktorijelna, ovakva pretraga gubi praktično značenje već kod malog broja gradova. Može se koristiti za probleme do desetak gradova, a dobiveno konačno rješenje uvijek je optimalno.

4.1.2. Pohlepni algoritam

Rješavanje problema trgovačkog putnika pohlepnim algoritmom (eng. Greedy Algorithm) najčešće ljudima „prvo padne na pamet“. Pohlepni algoritam pretragu započinje od proizvoljnog početnog grada te je svaki sljedeći grad onaj koji je najbliži trenutnom gradu, a da još nije posjećen. Također, pohlepni algoritam najčešće ne daje optimalno rješenje jer ne koristi sve dostupne podatke. Iz navedenog razloga često se posjećuje neki grad prerano što onemogućuje pronalazak boljeg rješenja kasnije. Važnost pohlepnog algoritma je u tome što se može brzo provesti, a daje odličnu aproksimaciju optimuma. Većinom se koristi za usporedbu i testiranje drugih algoritama. Pohlepni se algoritam naziva i algoritam najbližeg susjeda (eng. Nearest Neighbour Algorithm) jer se u svakom koraku posjećuje grad koji je najbliži trenutnom gradu.

4.1.3. Grananje i ograničavanje

Grananje i ograničavanje (eng. Branch And Bound) je opći algoritam za traženje optimalnih rješenja mnogih optimizacijskih problema. Sastoji se od brojanja svih potencijalnih rješenja pri čemu se veliki podskupovi loših rješenja odbacuju. Loša rješenja su ona koja su unutar nekog intervala, tj. između gornje i donje granice.

Redoslijed obilaska gradova trgovačkog putnika se najprije rekurzivno grana na dva dijela te nastaje struktura stabla nakon čega se računaju gornja i donja granica za svaki dobiveni podskup. Ukoliko je donja granica nekog čvora bolja od gornje granice nekog drugog čvora, taj drugi čvor se izbacuje iz pretrage. Rekurzija se zaustavlja kad se skup reducira na jedan element ili kad njegova gornja granica postane jednaka njegovoj donjoj granici.

4.1.4. Linearno programiranje

Linearno programiranje (eng. Linear Programming) je optimiziranje linearne funkcije cilja koja je podvrgnuta ograničenjima linearne jednakosti i nejednakosti. Ono traži način kako postići najbolji rezultat upotrebljavajući listu zahtjeva koja je opisana linearnim jednadžbama. Omogućuje uspješno rješavanje problema veličine do 200 gradova.

4.1.5. Najbliže ubacivanje

Najbliže ubacivanje (eng. Nearest Insertion) uzima neki početni grad koji predstavlja početnu rutu nakon čega se traži drugi grad koji je najbliži početnome te ga ubacuje u početnu rutu. Ruta tada sadrži dva grada, a sljedeći grad koji se dodaje je onaj grad koji je najbliži bilo kojem gradu u ruti, a da već nije sadržaj u njoj. Taj grad se ubacuje u rutu na optimalno mjesto, tako da dobivena udaljenost bude minimalna. Postupak se ponavlja dok se ne ubace svi gradovi.

4.1.6. Najdalje ubacivanje

Najdalje ubacivanje (eng. Farthest Insertion) je slično najbližem ubacivanju. Razlika je u tome što se kod najdaljeg ubacivanja u rutu dodaje grad koji je najudaljeniji od bilo kojeg grada u ruti, a ne onaj koji je najbliži. Najudaljeniji grad se dodaje na optimalno mjesto tako da nova udaljenost bude minimalna.

4.1.7. Evolucijski algoritam

Rješavanje problema trgovačkog putnika evolucijskim algoritmom je vrlo često (Hornby i Pollack, 2002.). Iako evolucijski algoritmi ne nalaze uvijek optimalno rješenje, njihova prednost je što mogu u razumnom vremenu naći rješenje koje je svega 2 ili 3% lošije od optimalnog. Mogu se primijeniti na probleme različite veličine, od onih malih do onih ogromnih koji se sastoje od nekoliko stotina tisuća gradova (Schrijver, 2005.). Dio evolucijski algoritama su genetski algoritmi.

4.2. Povezanost pčela i problema trgovačkog putnika

Kao što je to i napomenuto u članku „Pčele nemaju problema s „problemom putujućeg trgovca““, pčele rješavaju problem putujućeg trgovca svakodnevno i u tome su čak i vještije od računala. Naime, pčele posjećuju cvjetove na različitim lokacijama te troše mnogo energije za let, one pronalaze rute koje skraćuje letenje na minimum. Stoga, može se zaključiti da mozgovi pčela mogu riješiti kompleksne matematičke probleme s kojima se računala često muče danima (Ko., 2010).

Radišne pčele vrlo brzo i uspješno uče kako letjeti najkraće rute nasumično odabranih cvjetova te se može reći da time efektivno savladavaju problem trgovačkog putnika (TSP), tj. metodu odabira najjeftinije rute između odabranih gradova pod uvjetom da se svaki grad posjeti samo jednom i da putovanje bude ciklično. No, može se reći da čak i u ljudskom mozgu navedeno izaziva mentalni napor.

Znanstvenik Nigel Raine iz Škole za biološke znanosti, objasnio je da pčele rješavaju problem putujućeg trgovca svakodnevno. Naime, Raine objašnjava da pčele posjećuju cvjetove na različitim lokacijama te samim time troše mnogo energije za let. Na temelju toga, pronalaze rute koje skraćuju letenje na minimum. Raineov tim proveo je istraživanje u kojima je proučavano ponašanje pčela na nizu umjetnih cvjetova jer ih je zanimalo hoće li one odabrati najjednostavniju rutu definiranu redoslijedom po kojem su pčele nailazile na cvjetove ili će potražiti najkraći put. Tim je zaključio da nakon kratkog pregleda cvjetnog polja, pčele ipak odabiru najbolju rutu za uštedu energije i vremena te unatoč njihovim malim mozgovima, pčele su sposobne za izvanredne podvige u ponašanju (Ko., 2010).

4.3. Rješavanje problema trgovačkog putnika uz pomoć genetskog algoritma

Problem putujućeg trgovca (eng. travelling salesman problem) sastoji se u odabiru najkraćeg puta za obilazak *N* gradova i povratak u početni grad. Za svaki grad definirana je njegova lokacija tako da je moguće izračunati udaljenost između gradova. Ovaj problem spada u grupu teških problema i ima NP (eng. non-polynomial) složenost. Zbog posebnosti problema putujućeg trgovca potrebno je koristiti posebne funkcije za križanje i mutaciju, takve da rezultirajućim jedinkama nema ponavljanja gradova. U ovom eksperimentu križanje se obavlja tako da se od

prve ulazne jedinke uzme prvi grad te se bira sljedeći između gradova na drugoj poziciji obje ulazne jedinke. Odabire se onaj grad koji je manje udaljen od početnog grada. Postupak se nastavlja za sljedeće gradove, a ukoliko se oba dva ponuđena grada već nalaze u jedinki, slučajno se odabire jedan od preostalih gradova. Na isti način se sastavlja i druga izlazna jedinka koja za prvi grad uzima prvi grad druge ulazne jedinke. Treba napomenuti da osim navedenog, mogući su i alternativni načini križanja. Mutacija se vrši tako da dva slučajno izabrana grada zamjene mjesta. Potrebno je stvoriti matricu koordinata gradova te početnu populaciju „pokazatelja“ na gradove.

Funkcija cilja za problem trgovačkog putnika je vrlo jednostavan i prirodno se nameće. Naime, dva se rješenja mogu uspoređivati prema duljini obilaska svih gradova te se lako može zaključiti koje od njih je bolje. Međutim, stvar s izborom prikaza rješenja je sasvim suprotna. Binarni prikaz puta je potpuno neprikladan jer je potrebno permutirati cijele gradove pa njihov binarni kod nema nikakvo značenje. Binarni kod bi uzrokovao i kreiranje nekih ilegalnih rješenja pa bi bili potrebni algoritmi za popravljanje (eng. Repair Algorithm). Iz navedenog se razloga, rješenja problema trgovačkog putnika najčešće prikazuju vektorski i matrično. Kod vektorskog prikaza rješenja su prikazana kao lista gradova. Vrste vektorskih prikaza su zbijeni prikaz, redni prikaz i prikaz po redoslijedu obilaska.

4.3.1. Zbijeni prikaz

Pri zbijenom prikazu (eng. Adjacency Representation) gradovi su prikazani listom, a grad *j* se nalazi na poziciji *i* ako i samo ako put vodi iz grada *i* u grad *j*. Primjerice, vektor

(2 4 8 3 9 7 1 5 6)

predstavlja redoslijed obilaska

1-2-4-3-8-5-9-6-7.

Svaki redoslijed obilaska ima jedinstveni zbijeni prikaz, ali neki vektori predstavljaju ilegalne rute. Tako vektor

(2 4 8 1 9 3 5 7 6)

predstavlja ilegalnu rutu

1-2-4-1.

Navedena ruta je dovela do preranog zatvaranja ciklusa što znači da bi putnik posjetio jedan grad dvaput, a to nije u skladu s uvjetom u opisu problema.

4.3.2. Redni prikaz

Kod rednog prikaza (eng. Ordinal Representation) postoji uređena lista gradova koja služi kao referentna točka za gradove u rednom prikazu. Primjer takve liste je:

C=(1 2 3 4 5 6 7 8 9).

Put je prikazan listom gradova, a *i*-ti element liste je broj iz intervala [1,n-1+1] gdje je *n* broj elemenata. Primjer takvog prikaza je

L=(1 1 2 1 4 1 3 1 1).

Prvi broj u prikazu je 1 pa se uzima prvi grad iz liste C i briše iz nje. Na taj način se dobiva da je prvi grad 1. Nakon toga, čita se sljedeći broj iz liste *L* koji je opet 1 pa se uzima prvi broj iz promijenjene liste *C*. Prvi grad je sada broj 2 što znači da se on briše iz liste *C* i da postoje drugi grad traženog obilaska. Sljedeći broj u listi *L* je 2, pa se čita drugi grad iz liste *C* i dobiva 4. Postupak se ponavlja za sve gradove i dobiva obilazak 1-2-4-3-8-5-9-6-7.

Glavna prednost rednog prikaza je što se s njim može koristiti klasično križanje. Bilo koje dvije ture rednog prikaza klasičnim križanjem u jednoj točki daju djecu koja su legalne ture.

4.4. Prikaz po redoslijedu obilaska

Prikaz po redoslijedu obilaska (eng. Path Representation) je najprirodniji prikaz problema trgovačkog putnika. U njemu prikaz

(4 5 1 2 3 9 8 6 7)

predstavlja redoslijed obilaska

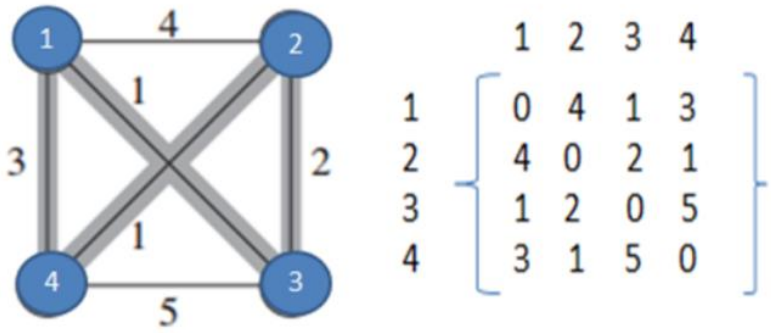
4-5-1-2-3-9-8-6-7.

4.5. Primjer problema trgovačkog putnika

U ovom poglavlju je predstavljen primjer problema trgovačkog putnika. Problem kreće od pretpostavke da postoje četiri grada (1, 2, 3, 4). Kako bi se približilo najgorem slučaju, postoji pretpostavka da je svaki grad spojen s ostalim gradovima. Također, postoji i trgovački putnik koji živi u gradu 1 te mora posjetiti svaki grad i vratiti se u početni grad, tj. grad u kojem živi. Svaki grad treba posjetiti točno jednom jer u suprotnom gubi vrijeme i energiju da više puta posjećuje isti grad. Navedeno se može poistovjetiti i s posjećivanjem svakog čvora u grafu točno jednom, što predstavlja Hamiltonov ciklus. No, u ovom slučaju, problem je nešto veći od Hamiltonovog ciklusa jer to nije samo pronalaženje Hamiltonovog puta, nego pronalaženje i najkraćeg puta. Uza sve navedeno, potrebno je posjetiti svaki grad (vrh u grafu) točno jednom s minimalnim troškom ruba u grafu.

Pristup grubom silom (eng. Brute Force) traje O (n^n) vrijeme, jer se treba provjeriti (n-1)! staze, tj. sve permutacije i mora se naći minimum među njima. Jedan od boljih pristupa je rješavanje problema pomoću dinamičkog programiranja. Dinamičko programiranje se može primijeniti ukoliko se glavni problem može podijeliti na potprobleme. Slijedi grafički prikaz problema.

Slika 13.: Grafički prikaz problema trgovačkog putnika



Izvor: The Crazy Programmer, 2019

Priložena slika prikazuje cjeloviti usmjereni graf i matricu troškova koja uključuje udaljenost između svih gradova. Može se uočiti kako je matrica troškova (eng. cost matrix) simetrična, što znači da je udaljenost između grada 2 i 3 ista kao udaljenost

između grada 3 do 2. Postavlja se pitanje putovanja trgovačkog putnika uz minimalne troškove. Ukoliko je T(1, {2, 3, 4}), navedeno prikazuje da je trgovački putnik na početku u gradu 1 i onda može otići na bilo koju lokaciju od {2, 3, 4}. Odatle do ne-posjećenih vrhova/gradova postaje novi problem. U ovom slučaju se može primijetiti kako glavni problem ulazi u potproblem što je svojstvo dinamičkog programiranja.

Treba se napomenuti kako prilikom izračunavanja ispod desne strane, vrijednosti se izračunavaju na način odozdo prema gore. Vrijednosti crvene boje preuzete su iz danih izračuna.

T (1, {2,3,4}) = minimum of

= { (1,2) + T (2, {3,4}) 4+**6**=10
= { (1,3) + T (3, {2,4}) 1+**3**=4
= { (1,4) + T (4, {2,3}) 3+**3**=6

Navedeno je odgovor za najmanje od tri staze, ali se znaju samo vrijednosti (1, 2), (1, 3), (1, 4), a preostale su staze poput T (2, {3, 4} što predstavlja novi problem što je prikazano sljedeće.

T (2, {3,4}) = minimum od

= { (2,3) + T (3, {4}) 2+**5**=7
= { (2,4) + T {4, {3}) 1+**5**=6

T (3, {2,4}) = minimum od

= { (3,2) + T (2, {4}) 2+**1**=3
= { (3,4) + T {4, {2}) 5+**1**=6

T (4, {2,3}) = minimum od

= { (4,2) + T (2, {3}) 1+**2**=3
= { (4,3) + T {3, {2}) 5+**2**=7

T (3, {4}) = (3,4) + T (4, {}) 5+0=5

T (4, {3}) = (4,3) + T (3, {}) 5+0=5

T (2, {4}) = (2,4) + T (4, {}) 1+0=1

T (4, {2}) = (4,2) + T (2, {}) 1+0 = 1

T (2, {3}) = (2,3) + T (3, {}) 2+0 = 2

T (3, {2}) = (3,2) + T (2, {}) 2+0=2

Ovdje T(4, {}) doseže osnovni uvjet u rekurziji, koji vraća vrijednost 0 (nula). Slijedi prikaz konačnog rješenja.

T (1, {2,3,4}) = minimum od

= { (1,2) + T (2, {3,4}) 4+**6**=10 na ovoj stazi/putu mora se dodati +1 jer se put završava s 3. Naime, odatle je potrebno stići 3->1 udaljenost 1 će biti dodana na ukupnu udaljenost koja je 10+1=11

= { (1,3) + T (3, {2,4}) 1+**3**=4 na ovoj stazi/putu se treba dodati +3 jer ova staza završava s 3. Odatle je potrebno stići do jedan na način 4->1 gdje će se distance 3 nadodati na ukupnu vrijednost te ona glasi 4 + 3 = 7

= { (1,4) + T (4, {2,3}) 3+**3**=6 na ovoj stazi/putu dodaje se +1 jer put završava s 3 te odatle se treba doći do 1, stoga 3->1 distanca će se dodati ukupnoj distanci koja glasi 6 + 1 = 7.

Minimalna udaljenost je 7 što uključuje put **1->3->2->4->1.**

Nakon rješavanja primjera problema, slijedi rekurzivna jednadžba:

T (i , s) = min ((i , j) + T (j , S – { j })) ; S!= Ø ; j € S ;

S je postavljen kako bi sadržavao neposjećene vrhove:

= (i, 1) ; S=Ø, što je osnovni uvjet za ovu rekurzivnu jednadžbu.

Ovdje, T (i, S) znači da se putuje iz vrha *i* i potrebno je posjetiti skupo ne-postojećih vrhova *S* te se potrebno vratiti na vrh 1 (ukoliko se kreće iz vrha 1); (i, j) označava trošak puta od čvora *i* do čvora *j*. Ukoliko se promatra prva rekurzivna jednadžba iz čvora, nalaze se troškovi za sve ostale čvorove (i, j) i od tog čvora do preostalog korištenja rekurzije (T (j, {S-j})). No, navedeno nije jamstvo da je svaki vrh povezan s drugim vrhom te tada se trošak smatra beskonačnim nakon čega se uzima minimum od svih tako da put koji nije povezan dobiva beskonačnost u izračunu i ne uzima se u obzir. Ukoliko je *S* prazan, to znači da su posjećeni svi čvorovi, tj. udaljenost od zadnjeg postojećeg čvora do čvora 1 (prvi čvor) jer nakon posjećivanja svakog čvora, trgovački putnik se mora vratiti u početni čvor, tj. grad (The Crazy Programmer, 2019).

Što se tiče složenosti vremena, budući da se radi o rješavanju problema trgovačkog putnika pomoću dinamičkog programiranja, potrebno je napomenuti da navedeni pristup sadrži potprobleme. Naime, nakon pronalaženja i-tog čvora, pronalaze se preostale minimalne udaljenosti do i-tog čvora što predstavlja potproblem. Ukoliko se riješi rekurzivna funkcija, dobit će se ukupno $\mathbf{(n-1)\ 2^{(n-2)}}$ potproblema čija je složenost $\mathbf{O\ (n2^n)}$. Svaki potproblem traje O(n) vrijeme (pronalaženje puta do preostalih (n-1) čvorova). Stoga, ukupna složenost vremena je $\mathbf{O\ (n2^n) * O\ (n) = O\ (n^2 2^n).}$ Prostorna složenost je broj potproblema što iznosi $\mathbf{O\ (n2^n)}$. U prilogu se može naći prikaz problema trgovačkog putnika u C++.

5. Algoritam optimizacije kolonijom mrava

Algoritam optimizacije kolonijom mrava (engl. Ant Colony Optimization, skraćeno ACO) je populacijski orijentirana metaheuristika koja se koristi za rješavanje optimizacijskih problema, ponajviše kombinatornih NP-teških problema. Prvi algoritam optimizacije kolonijom mrava korišten je za nalaženje optimalnog puta u grafu, a algoritam je oponašao ponašanje mrava u potrazi za hranom. Danas algoritam optimizacije kolonijom mrava ima svestranu primjenu, a najpoznatiji je za rješavanje problema raspoređivanja, problema usmjeravanja vozila (engl. vehicle routing), detekciju rubova na slikama, problem trgovačkog putnika i sl. Kod optimizacije kolonijom mrava, skup umjetnih mrava traži dobro rješenje za dani optimizacijski problem, a surađivanje je osnovna značajka algoritma optimizacije kolonijom mrava iz kojega dobro rješenje proizlazi iz međusobnog surađivanja mrava. Za primjenu ovakve optimizacije, optimizacijski problem opisuje se problemom nalaženja najboljeg puta u težinskom grafu. Umjetni mravi postupno grade rješenje, krećući se kroz graf. Proces konstrukcije rješenja je stohastičan i pod utjecajem feromon modela, a to je skup parametara povezanih s komponentama grafa (ili čvorovima, ili bridovima), čije vrijednosti su modificirane prolazima mrava. Algoritmi optimizacije kolonijom mrava mogu se koristiti za rješavanje statističkih i dinamičkih kombinatornih problema. Karakteristike statičnog problema su definirane samo jednom, i to samo na početku, i ne mijenjaju se za vrijeme njegovog rješavanja/traženja rješenja. Klasičan primjer je problem trgovačkog putnika gdje se lokacije gradova i njihove relativne udaljenosti ne mijenjaju. S druge strane, dinamički kombinatorni problemi su definirani kao funkcija nekih veličina, čije vrijednosti mijenja dinamika ishodišnog sistema, tj. veličine se mijenjaju u stvarnom vremenu, čemu se algoritam optimizacije kolonijom mrava uspješno prilagođava. Definicija problema mijenja se za vrijeme rješavanja i optimizacijski algoritam mora se moći mijenjati, ovisno o promjenama definicije. Primjer dinamičkog problema je mrežno usmjeravanje (eng. network routing), jer se u mrežama čvorovi mogu dodavati i uklanjati.

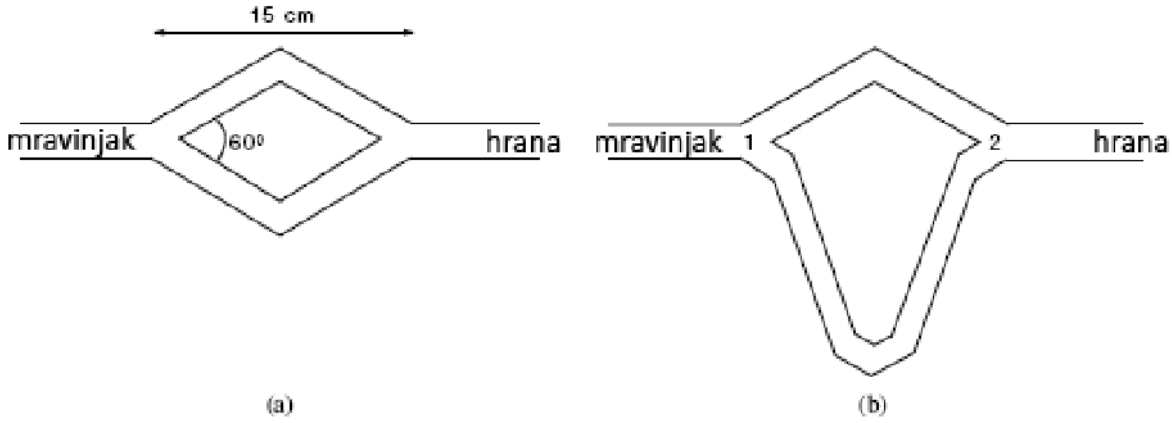
Stoga, može se zaključiti kako algoritam optimizacije kolonijom mrava proučava umjetne sustave inspirirane ponašanjem pravih mravljih kolonija, koji se koriste za rješavanje optimizacijskih problema, najčešće pronalaženja najkraćeg puta u

zadanom grafu. Pojam algoritma optimizacije kolonijom mrava (eng. ant colony optimization) uvodi Marco Dorigo u svome doktoratu iz 1992., kao rezultat istraživanja pristupa kombinatoričkoj optimizaciji. U početku se algoritam primjenjivao na problemu trgovačkog putnika i na problem kvadratičnog pridruživanja. Od 1995. Dorigo sa suradnicima radi na raznim proširenim verzijama početne ideje, počevši sa značajnim eksperimentom Deneubourga iz 1990. gdje se koristio dvostruki most koji je povezivao mravinjak i izvor hrane. Eksperiment je uključivao variranje omjera duljina grana mosta r = l1 / l2, a početno je postavljeno r = 1. S obzirom da na početku ne postoje nikakvi tragovi feromona, mravi nasumično biraju jednu od grana mosta. Čim je veći broj mrava, može se pretpostaviti da neće jednak broj mrava krenuti na obje grane, nego će slučajno jedna grana imati nešto više mrava. Nešto više mrava povlači nešto više feromona, dok nešto više feromona povlači nešto više novih mrava koji se privlače. Pojam „nešto više“ se ne taj način u nekoliko iteracija pretvori u „puno više“ sve dok se nakon nekog vremena ne odluče svi mravi za jednu granu, i to upravo za onu na koju je slučajno prešlo nešto više mrava na samom početku eksperimenta. Dalje se omjer postavio na r = 2, s time da mravi ne mogu unaprijed znati koja je grana kraća. Situacija počinje identično prethodnom slučaju na način da se mravi nasumično odlučuju za neku od grana. No, oni koji su krenuli kraćom granom, ti će prvi stići do hrane i početi ostavljati trag feromona i time pozivati nove mrave. U ovom će se slučaju ponovno svi mravi odlučiti nakon nekog vremena za jednu granu, i to onu kraću, ali ovog puta ne zbog nasumičnosti, već zbog bržeg obnavljanja traga feromona na kraćoj grani. Navedeno se eksperimentom intuitivno može objasniti glavna ideja algoritma optimizacije kolonijom mrava. Sljedeća slika prikazuje postavke eksperimenta dvostrukog mosta (a) s jednakim granama te (b) s različitim granama (Dorigo i sur., 1996.).

Cilj eksperimenta je bio proučiti ponašanje kolonija mrava, a dokazano je kako, ukoliko oba mosta imaju istu duljinu, mravi konvergiraju prema korištenju jednog od ta dva mosta. Nakon što je eksperiment ponovljen više puta, pokazalo se da je svaki od dva mosta korišten jednako mnogo puta. Rezultat objašnjava činjenica da mravi, dok se kreću, ispuštaju feromone na tlo, a kad trebaju odlučiti koji put će slijediti, njihov izbor temelji se na feromonima. Ukoliko je veća količina feromona nađenih na tlu, onda je i veća vjerojatnost da će slijediti taj put. Naime, u početku mravi istražuju okolinu mravinjaka. Kada dođu do trenutka odluke koji most će odabrati, biraju s

vjerojatnošću temeljenoj na količini feromona koju osjećaju na dva mosta. Svaki mrav na početku bira jedan od dva mosta s jednakom vjerojatnošću, zato što još nema feromona. Ipak, nakon nekog vremena, zbog slučajnih varijacija, jedan od dva mosta sadrži veću količinu feromona pa zbog toga privlači više mrava. Zbog navedenog mehanizma, nakon nekog vremena, cijela kolonija konvergira prema samo jednom mostu.

Slika 14.: Postavke eksperimenta dvostrukog mosta



Izvor: Stambolić, 2018

Na slici, na lijevoj strani (a) je prikaz eksperiment dvostrukog mosta u slučaju kada su oba mosta su iste dužine, a na desnoj strani je prikaz mostova koji su različitih dužina.

Stoga, kako je algoritam optimizacije kolonijom mrava nastao po uzoru na ponašanje iz prirode, najbolje je princip rada objasniti upravo na stvarnom ponašanju. Stave li se mravi u nepoznatu okolinu da pronađu izvor hrane, najprije će se kretati u potpunosti nasumično. U trenutku kada mrav pronađe hranu, tada će pri povratku u koloniju ostavljati trag feromona koji će privlačiti druge mrave upravo u smjeru hrane. Tako će se postepeno „neodlučni“ mravi koji se kreću nasumično početi kretati u smjeru traga feromona te će se pojam „slučajno“ pretvoriti u „optimalno“. Svaki novi mrav koji se priključi pronalasku hrane, pojačavat će trag feromona i tako privlačiti još više novih mrava. Riječ je o strategiji novačenja koji radi na principu povratne veze. Kroz vrijeme, trag feromona počinje slabiti, što je dobra stvar, obzirom da je pozicija hrane u okolišu dinamična. Naime, nije poželjno imati trag koji navodi ostale mrave u smjeru gdje je prije dva dana bila hrana, ali sada više nije. No, činjenica slabljenja

traga ukazuje na drugu zanimljivu činjenicu. Ukoliko se pretpostavi da postoje dva izvora hrane, jedan bliži i jedan udaljeniji od kolonije i ukoliko se svi mravi u početku kreću nasumično, u jednom će trenutku oba izvora biti pronađena te će mravi početi ostavljati trag feromona koji usmjerava ostale mrave u tom smjeru. No, kraći put povlači kraće vrijeme prolaska, što znači da će trag manje slabiti jer će se brže obnavljati. Za duži put treba duže vrijeme prolaska, odnosno više će feromona oslabiti. Kako mrave privlači jači trag feromona, tako će se više mrava odlučiti upravo za kraći put. Bitno je istaknuti i da isparavanje feromona ima prednost u smislu da se izbjegava konvergencija lokalno optimalnog rješenja, pa se ostavlja mjesto i za istraživanje širih prostora, na kojima se možda nalaze bolji izvori hrane. Ideja algoritma upravo oponaša spomenuto ponašanje mrava, a najčešće se opisuje primjerom rješavanja problema putujućeg trgovca. Prednost algoritma optimizacije kolonijom mrava je upravo ta što se može prilagoditi dinamičkoj situaciji (npr. promjeni grafa u vremenu). Osim navedenih sličnosti, postoje i bitne razlike između stvarnog svijeta i simulacije umjetnih mrava. Tako primjerice umjetni mravi žive u diskretnom svijetu, dok se njihovi pokreti sastoje od prijelaza iz diskretnog u diskretno stanje te imaju interno stanje koje sadrži informacije o svojim prethodnim akcijama. Algoritam optimizacije kolonijom mrava se često unaprjeđuje dodatnim sposobnostima poput lokalne optimizacije ili gledanja unaprijed, čime se ne mogu pohvaliti mravi iz stvarnog svijeta. Iz svega navedenog, može se zaključiti kako kolonija mrava predstavlja kompleksnu, moćnu organizaciju i to upravo zbog međusobne suradnje mrava. Zahvaljujući toj kompleksnosti, kolonija može riješiti zadatke koji nadmašuju sposobnosti pojedinog mrava. Sposobnost vida kod određenih vrsta mrava je djelomično razvijena, a neke vrste su i potpuno slijepe, no količina feromona koju ostavljaju može ovisiti o kvaliteti i količini hrane. Ostali mravi mogu namirisati feromone i slijediti njihov trag do hrane. Mravi, na putu prema hrani, s većom vjerojatnošću biraju put s većom koncentracijom feromona. Upravo takva komunikacija omogućuje mravima da nađu najkraći put između hrane i mravinjaka.

5.1. Teorijska podloga optimizacije kolonijom mrava

Dok se kreću (ili trče), mravi polažu na tlo određenu količinu kemijske supstance poznate kao feromon. Istraživanja biologa i inženjera su pokazala da svaki mrav s određenom vjerojatnošću preferira praćenje putanje (traga) na kojoj postoji deblji (bogatiji) sloj feromona. Ova jednostavna činjenica objašnjava zašto su mravi sposobni da se prilagode promjenama u svom okruženju, kao što je npr. pojava neke prepreke na njihovom putu. No postavlja se pitanje što se događa kada kolona mrava koja se kreće najkraćim putem između staništa i hrane, bude spriječena da nastavi takvo kretanje zato što se negdje na putanji pojavila prepreka.

Ono što se realno događa je da kada mravi naiđu na prepreku, oni slučajno biraju način da je zaobiđu tako što kreću lijevo, desno, gore ili dolje. No, zbog pojednostavljenja objašnjenja, može se pretpostaviti da mravi jedino mogu lijevo ili desno u odnosu na prepreku. S velikom sigurnošću se može reći da će približno polovina mrava u početku ići na jednu, a druga polovina na drugu stranu. Mravi koji krenu kraćim putem, brže će formirati jači trag feromona nego oni koji idu dužim putem što će dalje izazvati da sve više i više mrava kreće za jačim feromonskim tragom dok konačno svi mravi ne nađu taj kraći put. Mravi koji prate najkraći put prvi i vraćaju se odakle su krenuli (gnijezdo ili hrana), njihov feromonski trag postepeno nestaje uslijed isparavanja, ali ga novi mravi istovremeno i pojačavaju. Treba napomenuti da iako jači trag privlači više mrava koji taj trag još više pojačavaju, uvijek ima mrava-solista koji u međuvremenu idu svojim putem bez obzira na trag i tako istražuju nove puteve. Kada prepreka na putu izazove gužvu, mravi su spremni da slijede rezervne pravce. Pravci koji se manje koriste konačno se eliminiraju iz razloga što feromon koji je na njih položen jednostavno ispari.

Novija istraživanja su dokazala sposobnost mrava da brzo nađu sljedeći najkraći put kada prepreka blokira direktan (najkraći) put od gnijezda do hrane (Bonabeau i sur., 1999). Kada se ova sposobnost mrava preslika na mrežu komunikacijskih pravaca na Internetu, ili mrežu puteva ili sistem telefonskih linija, tada mravi nude neka zanimljiva rješenja. Jedan od primjera je i preusmjeravanje prometa ukoliko dođe do zagušenja ili prekida na postojećim pravcima. Ideje o paralelizmu prometa i kretanju kolonija mrava dovele su do razvoja „mravlje metodologije“ (eng. Ant Methodology) i „algoritma mravljih kolonija“ (eng. Ant Colony Algorithms). Tvorcima mravlje

metodologije optimizacije smatraju se belgijski znanstvenik Dorigo i njegov suradnik Gambardella (Dorigo i sur., 1996.). Glavna ideja novog pristupa diskretnoj optimizaciji je da se simulira ponašanje vještačkih mrava (pokretljivi agenti inspirirani ponašanjem stvarnih mrava) i generiraju nova rješenja za određeni problem. Vještački mravi koriste informaciju sakupljenu tijekom prethodnih simulacija i usmjeravaju tok daljeg traženja rješenja problema, s tim da je ta informacija raspoloživa i promjenljiva u danom okruženju. Ni jedan od mravljih algoritama se ne temelji na jedinstvenoj definiciji umjetnog mrava, niti potpuno imitira ponašanje stvarnih mrava. Štoviše, pošto se radi o novom području, nisu unificirani ni terminologija, definicije niti matematička notacija, prije svega zbog heurističke komponente koja dopušta maštovitost znanstvenika u traženju najboljih simulatora prirode.

Umjetni mravi u algoritmu optimizacije kolonijom mrava su stohastički konstrukcijski postupci koji inkrementalno grade rješenje, dodajući odgovarajuće komponente rješenja na parcijalno rješenje koje se gradi. Iz tog razloga, metaheuristika algoritama optimizacije kolonijom mrava se može primijeniti na sve kombinatorne optimizacijske probleme za koje se može definirati konstrukcijska heuristika. Iako se navedeno odnosi na to da se metaheuristika algoritma optimizacije kolonijom mrava može primijeniti na bilo koji zanimljiv kombinatorni optimizacijski problem, pravi problem je kako preslikati problem kojeg se promatra, u prikaz kojeg umjetni mravi mogu koristiti kod konstrukcije rješenja.

Insekti međudjeluju bez nadzora te se njihove akcije temelje na interakcijama između pojedinaca. Iako su njihove interakcije primitivne na nižoj razini, na onoj višoj, globalnoj razini često daju impresivne rezultate. Navedeno kolektivno ponašanje koje izranja iz grupe naziva se inteligencija roja. Glavne prednosti su fleksibilnost (brza prilagodljivost promjenjivoj okolini), robusnost (otpornost na manja odstupanja) te samo-organizacija (sposobnost funkcioniranja bez nadziranja). Stoga, algoritam optimizacije kolonijom mrava upravo pripada klasi inteligencije roja (eng. Swarm Intelligence).

6. Problem raspoređivanja poslova

Problemi raspoređivanja poslova (eng. Scheduling Problems) su problemi u kojima treba alocirati ograničena sredstva za obavljanje nekih aktivnosti tijekom vremena. Oni predstavljaju kompliciran zadatak koji može biti definiran pomoću niza ograničenja koja se trebaju ispuniti kako bi zadatak bio uspješno obavljen. Konačno rješenje mora zadovoljavati sve postavljene uvjete. Kada su svi uvjeti zadovoljeni, raspored se može optimizirati prema nekom kriteriju. Kriteriji može biti cijena, kašnjenje, vrijeme obrade dijela proizvoda, vrijeme obrade cijelog proizvoda i sl.

Na raspolaganju je konačan skup od *n* poslova i *m* strojeva. Svaki se posao sastoji od skupa operacija, a svaki stroj može obavljati najviše jedan posao odjednom. Svaki posao se mora obavljati neko vrijeme na nekom stroju i za to vrijeme ne smije biti prekidan. Potrebno je pronaći optimalan raspored, tj. alocirati takav redoslijed obavljanja zadanih poslova na zadanim strojevima da ukupna duljina obavljanja posla bude minimalna.

Neki poslovi imaju velika ograničenja zbog nedostupnosti sredstva (primjerice, novca ili opreme) i/ili malog vremenskog roka u kojem se moraju obaviti. Poslovi su najčešće disjunktni, što znači da dva zadatka ne mogu upotrebljavati isto sredstvo u isto vrijeme. Problem raspoređivanja poslova je NP-težak problem. U literaturi se navodi da je za određivanje mogućeg i učinkovitog (eng. feasible) rješenja potrebno definirati postupke kako bi se složenost problema smanjila (Garrido i sur., 2000).

Problemi raspoređivanja poslova uz vrijeme postavljanja strojeva (eng. Job Scheduling Problems with setup times) formalno su prikazani u literaturi kao niz poslova J={J1, J2,…, Jn} i niz sredstava R={R1, R2,…,Rn}. Svaki se posao *Ji* sastoji od skupa iteracija Oi={Oi1, Oi2,…,Oin} koje moraju biti obavljene između početnog (*rti*) i krajnjeg trenutka (*dti*). Obavljanje svake operacije *Oi* zahtjeva upotrebu skupa sredstva *Rik R* tijekom nekog vremenskog intervala. Postoji mnogo varijacija na temu problema raspoređivanja poslova uz vremena postavljanja strojeva. Tako primjerice strojevi mogu biti povezani ili neovisni, jednaki ili različiti, mogu ili ne moraju zahtijevati određeni razmak između obavljanja dvaju poslova. Poslovi često imaju takva graničenja da jedan posao mora završiti prije nego se drugi počne izvršavati. Neki poslovi ovise o strojevima na kojima se izvode pa se ne mogu izvoditi na svima

nego na točno određenim strojevima. S druge strane, drugi se pak brže izvode na jednoj vrsti strojeva, a sporije na nekoj drugoj (Xu, 2001).

6.1. Metode rješavanja

Prilikom rješavanja problema raspoređivanja poslova, postavljaju se pitanja kada maknuti trenutni posao sa stroja, koji posao staviti sjedeći te kada ga staviti. Posao se miče sa stroja kada je gotov, a može se maknuti i kao je dozvoljeno prekidanje uz uvjet da je posao višeg prioriteta spreman za izvođenje.

Kad tvornica proizvodi nekoliko vrsta proizvoda ili kad stroj ima više poslova za obaviti, potrebno je odrediti redoslijed izvođenja poslova. Upotreba jedinstvenog sustava za proizvodnju različitih komponenti najčešće traži neko vrijeme pripreme stroja između obavljanja dvaju poslova. Vrijeme pripreme stroja je neproduktivno vrijeme jer se za njegova trajanja na stroju ne obavlja niti jedan posao stoga ga je potrebno minimizirati. Ono ovisi o stroju koji obavlja četiri posla, moguća situacija je prikazana u tablici 1.

Tablica 1.: Vrijeme postavljanja stroja

	Posao 1	Posao 2	Posao 3	Posao 4
Posao 1	0	30	40	40
Posao 2	55	0	25	30
Posao 3	20	10	0	10
Posao 4	10	50	15	0

U tablici su prikazana vremena u minutama, potrebna da se stroj pripremi za obavljanje novog posla. Primjerice, ukoliko je stroj obavljao posao 3, tada mu je potrebno 25 minuta da se pripremi za obavljanje posla 2. Ukoliko se isti posao obavlja dva puta za redom, vrijeme pripreme stroja jednako je nuli. Ako stroj mora obaviti sva 4 posla, a početni posao je posao 1, moguće kombinacije i potrebna vremena predstavljena su:

1-2-3-4: 55+10+15=80

1-2-4-3: 55+50+10=115

1-3-2-4: 20+25+50=95

1-3-4-2: 20+15+30=65

1-4-2-3: 10+30+10=50

1-4-3-2: 10+10+25=45

Može se zaključiti da je optimalna kombinacija 1-4-3-2 čije je ukupno utrošenog vremena. U praksi se najčešće poslovi proizvode ciklično što znači da bi u prethodnom primjeru trebalo izvršiti s poslom s kojim se i počelo. U tom slučaju, rezultati su:

1-2-3-4-1 55+10+15+40=120

1-2-4-3-1: 55+50+10+40=155

1-3-2-4-1: 20+25+50+40=135

1-3-4-2-1: 20+15+30+30=95

1-4-2-3-1: 10+30+10+40=90

1-4-3-2-1: 10+10+25+30=75

Potrebno vrijeme se povećalo, ali optimalna kombinacija je ostala ista iako se i ona mogla promijeniti. Ukoliko se malo bolje pogledaju sve kombinacije i način izračunavanja optimuma, može se zaključiti da se problem raspoređivanja poslova na jednom stroju uz vremena postavljena svodi na problem trgovačkog putnika. Stoga, on se rješava istim metodama kao i problem trgovačkog putnika.

Kao što je to već napomenuto, problem rasporeda posla (eng. Job Shop Scheduling Problem) predstavljen je standardnim modelom *m*-strojeva i *n*-poslova koji se sastoje od sekvence operacija koje izvode strojevi. U ovom problemu, pokušavaju se dodijeliti operacije strojevima i vremenskim intervalima tako da ukupno vrijeme izvođenja bude minimalno, s time da u nekom intervalu jedan stroj može istovremeno raditi samo jedan posao. Problem rasporeda posla može se riješiti pomoću mravljeg algoritma gdje je osnovna ideja imati populaciju *m* umjetnih mrava koji iterativno grade rješenje primjenjujući odluke kretanja *n* puta dok rješenje nije pronađeno. Mravi koji pronađu dobro rješenje, označit će svoj prijeđeni put tragom feromona na rubove grafa kojim su prošli. Mravi također imaju memoriju (tzv. tabu lista) koja pamti posjećene komponente njihovog puta. Ovdje se uvodi i tehnika istraživanja koračanja

nogom (eng. Foot Stepping, FS). Ključna karakteristika strategije koračanja nogom je da mijenja informaciju koju su prikupili mravi. Mravi se koriste kako bi unaprijedili rješenje koje je pronašla kolonija. Za razliku od mnogih tehnika, tehnika koračanja nogom je fleksibilna i neovisna o problemu, a često se uspoređuje s MMAS tehnikom (eng. Max-Min Ant System). Kročenje nogom je inspirirano mogućnošću pravih mrava koji mogu modificirati svoje putove ukoliko iznenadno naiđu na neku prepreku. U ovom slučaju, ako se to dogodi, pretpostavlja se da će svaki trag feromona na tom području nestati. Stoga, mravi su primorani pronaći novo rješenje. Cilj koračanja nogom jest stvoriti oscilacije na postojećim dobrim rješenjima koje je pronašla kolonija te se primjenjuje tek nakon ili blizu globalne stagnacije. Preciznije, tehnika koračanja nogom uzrokuje promjene traga feromona nekih rubova. S obzirom na to da tehnika koračanja nogom počinje kada je pronađeno dobro rješenje, ono će uzrokovati lokalnu pretragu u okolini trenutnog rješenja.

Spomenuti Max-Min sustav mrava (eng. Max-Min Ant System, MMAS) razvili su Stutzle i Hoos kako bi se više iskoristila najbolja rješenja u procesu pretrage, a istovremeno izbjegla preuranjena stagnacija. Tako je Max-Min sustav za razliku od običnog Ant Systema, elitistički model koji ne ističe samo globalno najbolje rješenje, nego i najbolje rješenje trenutne iteracije. Koncept minimuma i maksimuma vrijednosti feromona na rubovima se uveo zbog preuranjene stagnacije, dok se inicijalno svi postave na maksimalne vrijednosti. Može se reći kako je jedna od bitnih karakteristika MMAS-a elitizam koji potiče rješenja koja se često pojavljuju kao iteracijski najbolja.

Zaključak

Problem usmjeravanja vozila je NP-težak problem, a ujedno i problem kojeg se susreće u realnom životu. Razvijeni su i prilagođeni mnogi algoritmi koji taj problem mogu riješiti, a jedan od njih je i genetski algoritam. Kako bi genetski algoritam bio učinkovitiji, potrebno je odabrati dobre parametre algoritma. No, svaki problem se može riješiti na više načina od kojih će neka rješenja biti lošija, a neka bolja. Rješavanje problema može biti uspješno, ali u isto vrijeme ne mora biti optimalno. Čovjek ima prirodnu sklonost da teži prema optimalnosti, odnosno pronalaženju najboljeg rješenja problema, tj. rješenja s najmanjom cijenom. Postojeći algoritmi za rješavanje nekih kombinatoričkih problema (primjerice pretraživanje u širinu, pretraživanje u dubinu, algoritam A*) na žalost, u praksi često nisu iskoristivi. Problemi koji imaju eksponencijalnu složenost, NP, ne mogu se riješiti postojećim algoritmima iscrpnog pretraživanja. Povećanjem broja mogućih stanja složenosti raste toliko da se može reći da svemir nije dovoljno star koliko treba da se riješi zadani problem.

Razvojem heurističkih algoritama mogu se dobiti rješenja NP problema koja nisu optimalna, ali su prihvatljiva za određene potrebe. Genetsko programiranje je jedan od načina rješavanja takvih problema, bez davanja konkretnih uputa o postupku rješavanja, uz mogućnost definiranja načina na koji će računalo naučiti učinkovito riješiti problem. Treba napomenuti da se genetsko programiranje svrstava u širu skupinu algoritama koja se zove evolucijsko računanje (eng. Evolutionary computing). Temelji se na Darwinovoj teoriji o postanku vrsta (razvoj svih životnih oblika procesom prirodne selekcije), a metodama križanja, mutacije i prirodne selekcije se dobiva najbolja jedinku iz populacije, odnosno dobiva se najbolje rješenje za dani problem. Mravlje kolonije su bile inspiracija znanstvenicima da istraže i pokažu da je moguće iskoristiti neke prirodne mehanizme kolektivne inteligencije i indirektne razmjene informacije da bi se po analogiji izgradile metode i rješavala jedna klasa složenih zadataka kombinatorne optimizacije te su na taj način nastale grupe algoritama specijalne namjene koji pripadaju obitelji heurističkih i metaheurističkih metoda. Navedena nova podobitelj se često i najpreciznije naziva i „mravlja optimizacija“. Primjena se nalazi u mrežnoj komunikaciji i distribuiranoj memoriji, a glavni problem je kako prevesti originalni optimizacijski zadatak u zadatak

kombinatorne (diskretne) optimizacije. No, može se zaključiti kako je algoritam optimizacije kolonije mrava relativno novi pojam u svijetu, pa je područje primjene i istraživanja dosta aktivno i aktualno u današnje vrijeme. Nove izazove predstavljaju višeciljni i dinamički problemi, za razliku od uglavnom statičkih koji su razmatrani u prošlosti. No, treba se naglasiti kako se ne može reći da je ovdje riječ o plagijatu jer je algoritam kolonijom mrava inspiriran jednostavnošću individualnog mrava i kompleksnošću mravlje kolonije kao cjeline. Od mrava je samo posuđena dobra ideja i prenesena na novu razinu. Na taj način, algoritam mravlje kolonije se postepeno bavio problemima najkraće udaljenosti i matematičkog pridruživanja, da bi kasnije počeo stvarati rasporede, bojati grafove, usmjeravati vozila, sve do naprednih primjena poput mrežnog usmjeravanja, gradskog planiranja ili DNK istraživanja. Uza sve navedeno, može se zaključiti kako problem trgovačkog putnika se može koristiti u mnogim ljudskim aktivnostima/problemima od kojih među najvažnijima treba istaknuti organizaciju proizvodnje i lociranje najoptimalnijeg slijeda operacija i zadaća. Potrebno je napomenuti kako je navedeni problem jedan od najpoznatijih i najproučavanijih problema kombinatorne optimizacije. Sam problem trgovačkog putnika se odnosi na permutacijski problem u kojem je cilj pronalaženje rute koja pokriva sve gradove na način da ukupna prijeđena udaljenost bude minimalna. Problem trgovačkog putnika je NP-težak problem, ali ga se može dovoljno dobro riješiti u razumnom vremenu koristeći heuristike te iz navedenog razloga su razvijeni su mnogi heuristički algoritmi koji daju približno optimalna rješenja, a izvode se u polinomijalnom vremenu.

Literatura

1) Adleman, L., Department of Computer Science and Institute for Molecular Medicine and Technology, University of Southern California (1994.), "Molecular Computation of Solutions To Combinatorial Problems", https://web.archive.org/web/20050206144827/http:/www.usc.edu/dept/molecular-science/papers/fp-sci94.pdf (preuzeto: 1.4.2018.)
2) Anić, V., Goldstein, I., Novi Liber, Zagreb (1991.), „Rječnik stranih riječi“
3) Baranović, M., Borčić, M., Hunjet, D., Kalafatić, V., Kranjčec, D., Mesarić, J., Peh, B., MZT, Zagreb (2003.), „Informacijski sustav visokih učilišta“
4) Bonabeau, E., Dorigo M., Theraulaz, G. (1999.), „Swarm Intelligence: From Natural to Artificial Systems“, Oxford Univ. Press, New York
5) Bryant, K., Department of Mathematic, Harvey Mudd College (2010.) „Genetic Algorithms and the Traveling Salesman Problem“, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.91.9167&rep=rep1&type=pdf, (preuzeto 11.4.2018.)
6) CODE PROJECT For those who code (2013.), „Applying Ant Colony Optimization Algorithms to Solve the Traveling Salesman Problem“, https://www.codeproject.com/Articles/644067/Applying-Ant-Colony-Optimization-Algorithms-to-Sol, (preuzeto:11.10.2018.)
7) Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C., MIT Press, Cambridge, Massachusetts London, Englanf, (2009.) „Introduction to Algorithms (3rd ed.), https://books.google.hr/books?id=i-bUBQAAQBAJ&printsec=frontcover&hl=hr&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false (preuzeto: 4.5.2018.)
8) Dorigo, M., Gambardella L. M. (1997), „Ant Colonies for the Traveling Salesman Problem“, BioSystems
9) Dorigo, M., Maniezzo, V., Colorni, A., IEEE Transactions on Systems, Man, and Cybernetics-Part B, Vol. 26 (1996.), „The Ant System: Optimization by a colony of cooperating agents“
10) Garrido, A., Salido, M. A., Barber, F., Lopez, M. A., Dpto. Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, Valencia, Spain (2000.)

„Heuristic Methods for Solving Job-Shop Scheduling Problems“, https://pdfs.semanticscholar.org/3607/c47e413bc32a6d1fe66d7411529d2cf73fc6.pdf (preuzeto 21.4.2018.)

11) Github (2018)., TSP/aco, https://github.com/arjun-krishna/TSP/blob/master/aco.cpp (preuzeto: 21.8.2019.)
12) Genetics Home reference (2019.), Your Guide to Understand Genetic Conditions „What is DNA“, https://ghr.nlm.nih.gov/primer/basics/dna (preuzeto: 6.8.2019.)
13) Golub M., Fakultet elektrotehnike i računarstva. Zavod za elektroniku, mikroelektroniku, računalne i inteligentne sustave (2004.), „Genetski algoritmi – Prvi dio“, http://www.zemris.fer.hr/~golub/ga/ga_skripta1.pdf (preuzeto 11.4.2018.)
14) Golub, M., Fakultet elektrotehnike i računarstva, Zavod za elektroniku, mikroelektroniku, računalne i inteligentne sustave (2004.), „Genetski algoritmi – Drugi dio“, http://www.zemris.fer.hr/~golub/ga/ga_skripta2.pdf (preuzeto 11.4.2018.)
15) Hornby,G.S., Pollack, J.B., Artificial life (2002.), „Creating high-level components with a generative representation for body-brain evolution“
16) Khan Academy (2019.), „Genetic linkage & mapping“, https://www.khanacademy.org/science/biology/classical-genetics/chromosomal-basis-of-genetics/a/linkage-mapping (preuzeto: 14.5.2019.)
17) Ko., D., Dnevno hr (2010.), „Pčele nemaju problema s „problemom putujućeg trgovca““, http://www.dnevno.hr/magazin/techno/znanost/pcele-nemaju-problema-s-problemom-putujuceg-trgovca-13807/ (preuzeto: 6.3.2018.)
18) Lawler, E. L., Shmoys, D. B., Kan, A. H. G. R., Lenstra, J. K., John Wiley & Sons, Incorporated (1985.) „The Traveling Salesman Problem“, https://books.google.hr/books?id=BXBGAAAAYAAJ&redir_esc=y (preuzeto: 4.5.2018.)
19) Mallawaarachchi, V., Towards Data Science (2017.), Introduction to Genetic Algorithms — Including Example Code, https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3 (preuzeto: 5.3.2019.)
20) Markić, B., Ekonomski fakultet, Sveučilište u Mostaru, Mostar, Bosna i Hercegovina (2008.), „Optimizacija portfolija i evloucijski algoritmi, Problem trgovačkog putnika“, https://hrcak.srce.hr/34348 (preuzeto: 24.5.2018.)

21) Michalewicz, Z., Fogel, D. B., Springer, Berlin (2004.), “How to Solve It: Modern Heuristics”
22) Milajić, A., Beljaković, D., Petronijević, P., Hrčak (2010.), „Genetski algoritmi za raspoređivanje rukovatelja građevinskih strojeva“, https://hrcak.srce.hr/file/107381 (preuzeto: 7.7.2019.)
23) neosGuide, „Types of Optimization Problems“ (2018), https://neos-guide.org/optimization-tree, preuzeto: 20.5.2018.
24) Nigel Raine Lab, https://1in3mouthfuls.org/ (preuzeto 28.8.2018.)
25) Pohlheim, H., GEATbx.com (2006.), Genetic and Evolutionary Algorithm Toolbox for Matlab, „GEATbx: Introduction; Evolutionary Algorithms: Overview, Methods and Operators“, http://www.geatbx.com/download/GEATbx_Intro_Algorithmen_v38.pdf (preuzeto 11.4.2018.)
26) Sapna Katiyar, Ibraheem, Abdul Quaiyum Ansari, ResearchGate (2015.), „Ant Colony Optimization: A Tutorial Review“, https://www.researchgate.net/publication/281432201_Ant_Colony_Optimization_A_Tutorial_Review (preuzeto: 2.4.2018.)
27) Schrijver, A., (2005.), „On the history of combinatorial optimization (till 1960)“, https://homepages.cwi.nl/~lex/files/histco.pdf (preuzeto: 12.8.2018.)
28) Stambolić, I., Klub mladih hemičara Srbije (2018.), „Kolonije mrava“, http://www.kmhem.net/antcolony/ (preuzeto: 5.5.2019.)
29) The Crazy Programmer (2019), Travelling Salesman Problem in C and C++, https://www.thecrazyprogrammer.com/2017/05/travelling-salesman-problem.html (preuzeto: 14.7.2019.)
30) Tutorialspont simplyeasylearning, „Genetic Algorithms – Crossover“, https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_crossover (preuzeto: 17.4.2019.)
31) Tutorialspoint simplyeasylearning, „Genetic Algorithms – Mutation“ https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_mutation (preuzeto: 14.5.2018.)
32) Xu, Q, (2001.), „Introduction to Job Shop Scheduling Problem“, http://www.cs.umbc.edu/671/fall01/class-notes/jobshop.ppt (preuzeto: 12.4.2018)

Popis slika

Popis tablica

Prilog

Slijedi prikaz problema trgovačkog putnika u C++ pomoću naivnog pristupa (The Crazy Programmer, 2019).

```
#include<iostream>
using namespace std;

int ary[10][10],completed[10],n,cost=0;

void takeInput()
{
	int i,j;

	cout<<"Enter the number of villages: ";
	cin>>n;

	cout<<"\nEnter the Cost Matrix\n";

	for(i=0;i < n;i++)
	{
		cout<<"\nEnter Elements of Row: "<<i+1<<"\n";

		for( j=0;j < n;j++)
			cin>>ary[i][j];

		completed[i]=0;
	}

	cout<<"\n\nThe cost list is:";

	for( i=0;i < n;i++)
	{
		cout<<"\n";

		for(j=0;j < n;j++)
			cout<<"\t"<<ary[i][j];
	}
}

int least(int c)
{
	int i,nc=999;
	int min=999,kmin;

	for(i=0;i < n;i++)
	{
		if((ary[c][i]!=0)&&(completed[i]==0))
			if(ary[c][i]+ary[i][c] < min)
			{
				min=ary[i][0]+ary[c][i];
				kmin=ary[c][i];
				nc=i;
			}
	}
```

```
	if(min!=999)
		cost+=kmin;

	return nc;
}

void mincost(int city)
{
	int i,ncity;

	completed[city]=1;

	cout<<city+1<<"--->";
	ncity=least(city);

	if(ncity==999)
	{
		ncity=0;
		cout<<ncity+1;
		cost+=ary[city][ncity];

		return;
	}

	mincost(ncity);
}

int main()
{
	takeInput();

	cout<<"\n\nThe Path is:\n";
	mincost(0); //passing 0 because starting vertex

	cout<<"\n\nMinimum cost is "<<cost;

	    system("pause");
	    return 0;
}
```

Slijedi prikaz trgovačkog putnika pomoću genetičkog algoritma:

```
#include<stdio.h>
#include<conio.h>
using namespace std;

int a[10][10],visited[10],n,cost=0;
int least (int);

void get()
{
     int i,j;
     printf("Enter No. of Cities: ");
     scanf("%d",&n);
     printf("\nEnter Cost Matrix\n");
     for(i=0;i < n;i++)
```

```
     {
               printf("\nEnter Elements of Row # : %d\n",i+1);
               for( j=0;j < n;j++)
               scanf("%d",&a[i][j]);
               visited[i]=0;
     }
     printf("\n\nThe cost list is:\n\n");
     for( i=0;i < n;i++)
     {
          printf("\n\n");
          for(j=0;j < n;j++)
          printf("\t%d",a[i][j]);
     }
}

void mincost(int city)
{
     int i,ncity;
     visited[city]=1;
     printf("%d -->",city+1);
     ncity=least(city);
     if(ncity==999)
     {
                    ncity=0;//printf("hii");
                    printf("%d",ncity+1);
                    cost+=a[city][ncity];
                    return;
     }
     mincost(ncity);
}

int least(int c)
{
    int i,nc=999;
    int min=999,kmin;
    for(i=0;i<n;i++)
    {
                     if((a[c][i]!=0)&&(visited[i]==0))
                     if(a[c][i] < min)
                     {
                               min=a[i][0]+a[c][i];
                               kmin=a[c][i];
                               nc=i;
                     }
    }

    if(min!=999)
    cost+=kmin;
    return nc;
}

void put()
{
     printf("\n\nMinimum cost:");
     printf("%d",cost);
}

int main()
{
    get();
    printf("\n\nThe Path is:\n\n");
```

```
    mincost(0);
    put();
    getch();
    return 0;
}
```

Sljedi prikaz problema trgovačkog putnika u C++ pomoću mravljeg algoritma (Github, 2018.).

```
#include <vector>
#include <cstdio>
#include <ctime>
#include <cmath>
#include <algorithm>
#include <set>
#include <iostream>
using namespace std;

class Data {
public:
      vector< pair<double,double> > cityCoords;
      vector< vector<double> > cost;
      vector< vector<double> > visibility;
      vector< vector<double> > T;
      int N;
      char s[30];

      Data() {    // N cities
            scanf("%s",s);
            scanf("%d",&N);

            // index from 1
            cityCoords.push_back(make_pair(0,0));
            vector< double> L,M,U;
            cost.push_back(L);
            T.push_back(M);
            visibility.push_back(U);

            for(int i=0;i<N;i++) {
                  double x,y;
                  scanf("%lf%lf",&x,&y);
                  cityCoords.push_back(make_pair(x,y));
            }
            for(int i=0;i<N;i++) {
                  vector<double> V(N+1);
                  vector<double> t(N+1);
                  vector<double> l(N+1);
                  for(int j = 1; j<=N ; j++) {
                        scanf("%lf",&V[j]);
                        t[j] = 1.0;
                        if(V[j] != 0) {
                              l[j] = 1/ V[j];
                        }
                  }
                  cost.push_back(V);
                  T.push_back(t);
                  visibility.push_back(l);
```

```
		}
	}

	double tourCost(vector<int> C) {
		int l = C.size();
		double tourCost = 0.0;
		l = l-1;
		for (int i=0;i <l; i++) {
			tourCost += cost[C[i]][C[i+1]];
		}
		tourCost += cost[C[l]][C[0]];
		return tourCost;
	}

	void print(vector<int> C) {
		for(int i=0;i<C.size();i++)
			printf("%d ",C[i]);
		printf("\n");
	}
};

Data d;
	// global data construct

class Ant {
public:
	vector<int> trail;
	set<int> available;
	double alpha;                                          //
importance of the pheromone level
	double beta;                                           //
importance of the visibility
	Ant(double a,double b) {
		alpha = a;
		beta = b;
		trail.push_back(1);                                // always
start from the nest (1)
		for(int i=2;i<=d.N;i++) {
			available.insert(i);
		}
	}
	void reset() {
		vector<int> L;
		L.push_back(1);
		trail = L;                                         // reset
to nest.
		for(int i=2;i<=d.N;i++) {
			available.insert(i);
		}
	}
	void deposit() {
		double tourCost = d.tourCost(trail);
		int Q = 100;
		double depositAmount = Q / tourCost;
		int l = trail.size();
		l = l-1;
		for (int i=0;i <l; i++) {
			d.T[trail[i]][trail[i+1]] += depositAmount;
		}
		d.T[trail[l]][trail[0]] += depositAmount;
	}
```

```
	vector<int> stop() {
		deposit();
		vector<int> temp = trail;
		reset();
		return temp;
	}

	void step() {
		int currentCity = trail[trail.size()-1];
		double norm = probabilityNorm(currentCity);
		double p,gp;
		bool moved = false;
		double highestProb = 0;
		double cityHighest = 0;
		for(set<int>::iterator i=available.begin(); i !=
available.end() ; i++) {
			p = moveProbability(currentCity,*i,norm);
			if (p > highestProb) {
				cityHighest = *i;
				highestProb = p;
			}
			gp = getRand();
			if (gp <= p) { // move
				moved = true;
				trail.push_back(*i);
				available.erase(i);
				break;
			}
		}
		if(!moved) {
			// make a move to the highest available prob city
			// move to cityHighest
			trail.push_back(cityHighest);
			available.erase(cityHighest);
		}
	}

	double getRand() {
			double p = (rand() / (RAND_MAX + 1.0));
	    return p;
	}
	double moveProbability(int i,int j,double norm) {
		double p =
(pow(d.T[i][j],alpha))*(pow(d.visibility[i][j],beta));
		p /= norm;
		return p;
	}

	double probabilityNorm(int currentCity) {
		int size = available.size();
		double norm = 0.0;
		for(set<int>::iterator i=available.begin(); i !=
available.end() ; i++) {
			norm +=
(pow(d.T[currentCity][*i],alpha))*(pow(d.visibility[currentCity][*i],beta))
;
		}
		return norm;
	}
```

```
};

class ACO {
public:
	int N;                                  // cities
	int M;                                  // no.of ants
	vector<Ant> ant;                  // ants
	double evaporation;                     // evaporation rate
	double alpha;                           // importance of the pheromone
level
	double beta;                            // importance of the visibility

	ACO(double a,double b,double e) {
		alpha = a;
		beta = b;
		evaporation = e;
		N = d.N;
		M = 30;
                                  // ants
		for(int i=0; i<M; i++) {
			Ant a(alpha,beta);
			ant.push_back(a);
		}
	}

	void run() {
		vector<int> PATH;
		double minTour,tourC;
		for(int n=0;n < 30; n++) {
			for(int p=0; p<(N-1); p++){
				for(int i=0;i<M;i++) {
					ant[i].step();
				}
			}
			for(int i=0;i<M;i++) {
				vector<int> p = ant[i].stop();
				if(!PATH.size()) {
					PATH = p;
					minTour = d.tourCost(p);
					continue;
				}
				tourC = d.tourCost(p);
				if(tourC < minTour) {
					minTour = tourC;
					PATH = p;
				}
			}
			for(int i=1;i<=N;i++) {
				for(int j=1;j<=N;j++) {
					d.T[i][j] *= evaporation;
				}
			}
		}
		printf("%lf\n",minTour);
		d.print(PATH);
	}
};

int main(void) {
	time_t t;
	srand(time(&t));
```

```
	double alpha = 1;					// pheromone
importance
	double beta = 2;					// visibility
importance
	double evaporation = 0.1;				// evaporation
rate
	ACO colony(alpha,beta,evaporation);
	colony.run();

	system ("pause");
	return 0;
}
```