Rešavanje problema minimalnog k-trgovačkog putnika

Marica Bogićević, Marina Brkić

Januar 2019

Sažetak

Ovaj rad se bavi primenom genetskih algoritama na problem trgovačkog putnika TSP. U radu je implementiran genetski algoritam u programskom jeziku C.

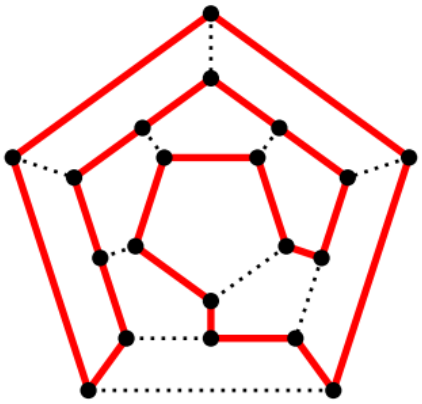
Sadržaj

1 Uvod

1.1 Problem trgovačkog putnika

1.1.1 Istorija

Problem trgovačkog putnika (eng. Travelling Salesman Problem – TSP) je klasičan problem u polju diskretne i kombinatorne optimizacije. Spada u grupu NP-teških problema a složenost mu je O(n!). Matematičke probleme slične njemu prvi je razmatrao Euler, koji se bavio pitanjem kako bi skakač na šahovskoj tabli posetio svih 64 mesta samo jednom. Početkom 20. veka matematičari William Rowan Hamilton i Thomas Kirkman su razmatrali probleme koji se svode na problem trgovačkog putnika. Hamilton je izmislio Ikozijansku igru u kojoj je cilj da se nadje zatvorena putanja ivice dodekaedra i da se svako njegovo teme pojavi tačno jednom u putanji. U današnjoj terminologiji to bi znacilo da treba da se nađe Hamiltonova putanja u grafu kome su čvorovi temena, a ivice su ivice dodekaedra.



Slika 1: Ikozijanska igra

Njegova opšta forma se pojavljuje 30-tih godina 20. veka. Pojam "trgovački putnik" prvi put je upotrebljen 1932. godine od strane Karl Mengera koji je u svom radu spomenuo brute-force algoritam i definisao TSP onako ga danas definišemo. Problem je vremenom postajao sve više popularan i izučavan od strane mnogih naučnika.

Problem trgovačkog putnika (eng. Travelling Salesman Problem – TSP) možemo neformalno da formulišemo na sledeći način: Dato je n gradova i poznate su sve udaljenosti između njih. Trgovački putnik treba da obiđe sve gradove i da se na kraju vrati u grad odakle je i krenuo, ali da pri tome razdaljina koju pređe bude najkraća. Formalna definicija problema trgovačkog putnika [Cormen et al., 2001] glasi:

Definicija 1.1 (Problem Trgovačkog putnika TSP) Ako je dat kompletan neusmeren težinki graf, sa nenegativnim celobrojnim težinama, naći Hamiltonovu putanju sa najmanjom težinom.

Osim optimizacijske varijante ovog problema, postoji i tzv. Forma problema odlučivanja:

Definicija 1.2 (TSP -odlučivanje) Ako je dat kompletan težinski graf i pozitivan realan broj *L*, treba odrediti da li postoji Hamiltonov put kraći od *L*.

2. Algoritmi za rešavanje

Kada razmišljamo o rešavanju ovakvih problema, prvo što nam pada na pamet je da se pronađu svi mogući obilasci, izračunaju njihove dužine i odabere najbolja. Ali problem ovakve metode je sto zahteva mnogo računarskog vremena za veći broj gradova. Zbog toga su razvijeni aproksimativni algoritmi koji relativno brzo daju prihvatljiva rešenja čak i ako su u pitanju problemi sa nekoliko miliona gradova. U nastavku su opisane neke vrste algoritama koji se koriste za rešavanje problema.

2.1 Egzaktni algoritmi

To su algoritmi čiji je ishod sigurno najbolje rešenje. Njihov problem je vreme izvršavanja. Osim spomente pretrage brute-force koja je zbog svoje složenosti O(n!) nepraktična za više od 10 gradova, metoda koja se još koristi je Branch and Bound (metoda grananja i ograničavanja) koja vrši procenu svih mogućih rešenja i odbacuje loša na temelju unapred postavljene gornje i donje granice. Korisna je za broj gradova između 40 i 60.

2.2 Aproksimativni algoritmi

Njihova prednost je što u relativno kratkom vremenskom periodu mogu da se dobiju dovoljno dobra rešenja. Primeri ovih algoritama su genetski algoritam, optimizacija kolonijom mrava, Lin-Keringan potezi metoda najbližeg suseda, pohlepni algoritam idt.

U nastavku ćemo objasniti problem TSP pomoću genetskog algoritma.

3. Genetski algoritami

3.1 Sta su genetski algoritmi i čemu služe

Genetski algoritmi (*Genetic Algorithms*, GA) su porodica algoritama koja je inspirisana Darvinovom teorijom evolucije. Tvorcem ove oblasti se smatra John Holland.
GA imaju za cilj rešavanje problema kombinatorne optimizacije, tj problema u kojima se traži minimum ili maksimum neke funkcije. Svako pojedinačno rešenje se predstavlja jednom jedinkom koja se sastoji iz gena koji prestavljaju delove rešenja. Nad njima se vrše operatori mutacije i ukrštanja (kao mehanizam pretrage) i selekcije (usmerava algoritam ka perspektivnim delovima pretraživačkog prostora). Iako ne nalaze uvek optimalno rešenje, njihova prednost je što mogu da u razumnom vremenu nađu rešenje koje je svega 2-3% lošije od optimalnog.

3.2 Osnovni operatori u GA

Definicija 1.3 *Fitnes funkcija* je genetski operator koji svakoj jedinki dodeljuje vrednost f_i, koja oslikava kvalitet te jedinke.

Kod TSP ona predstavlja ukupnu dužinu puta.

Definicija 1.4 *Selekcija* je genetski operator koji bira jedinke koje će se ukrštati i/ili preneti u sledeću generaciju.
Princip rada selekcije je sličan kao u stvarnom životu, cilj je da se odabere genetski materijal koji će se preneti u narednu generaciju. Ono što je važno je da se sačuva raznovrsnost kao i kvalitet genetskog materijala, u suprotnom dobra rešenja mogu da se zauvek izgube. Ideja je da se da veća šansa boljim jedinkama.

Definicija 1.5 *Ukrštanje* je genetski operator koji konstruiše 2 nove (decu) jedinke na osnovu 2 date jedinke (roditelja). Deca su, podrazume se, nastala kombinovanjem genetskog materijala roditelja.
Cilj ovog operatora je da se ukrštanjem dva postojeća rešenja dobiju nova, obično kvalitetnija rešenja.

Definicija 1.6 *Mutacija* je genetski operator koji (uglavnom) nasumično mutira genetski materija date jedinke.
Mutacijom se vraća raznovrsnost genetskog materijala u populaciju. Ipak, prekomerna mutacija može svesti algoritam na nasumičnu pretragu, pa se iz tog razloga uvodi verovatnoća mutacije koja je uglavnom manja od 5%.

3.3 Opis algoritma

```
Genetski_algoritam(Velicina_Populacije, t, p_m, p_c, M)
{
   t = 0;
   generiši početnu populaciju potencijalnih rešenja P(0);
   sve dok nije zadovoljen uslov završetka evolucijskog procesa
   {
      t = t + 1;
      selektuj P'(t) iz P(t-1);
      ukrštaj jedinke iz P'(t) i decu sačuvaj P(t);
      mutiraj jedinke P(t);
   }
   ispiši rešenje;
}
```

Slika 2. Pseudo kod Genetskog algoritma

U implementaciji genetskog algoritma koristi se permutacijska reprezentacija čvorova grafa, tj. svaka jedinka se sastoji od niza rednih brojeva čvorova u permutaciji p i njihovih međusobnih udaljenosti izračunavanjem rastojanja primenom pitagorine teoreme. U implementaciji je primenjena random inicijalizacija, selekcija i mutaciju. Algoritam počinje sa skupom inicijalizalizovanih gradova, odnosno rešenja (predstavljenih hromozomima) koji se naziva populacija. Inicijalizacija je proces koji se odvija jedanput, na početku rada algoritma. Ovaj deo je zadužen da načini polaznu populaciju koja će daljim procesima prirodne selekcije evoluirati u bolje populacije. Jedan od pristupa je sačiniti nasumičnu populaciju. Kod problema TSP, nasumična populacija predstavlja skup nasumičnih permutacija skupa {1,2,…,n}, gde je n broj gradova.

Selekcija je proces koji se odvija između svake dve uzastopne generacije. U ovom delu se iz cele populacije izdvajaju jedinke koje su podobne da pređu u sledeću generaciju, ili ekvivalentno, one jedinke koje nisu dovoljno jake, te stoga neće pripadati sledećoj generaciji odnosno potomstvu. Selekcija u našem slučaju je funkcija koja određuje podobnost neke jedinke računajući dužinu rastojanja između dva grada, odnosno dužina puta koju bi trgovački putnik morao da pređe ako bi gradove posećivao u poretku zadatom permutacijom te jedinke. Konstantan procenat p_e jedinki biće eliminisan. Postoje različiti pristupi za izbor ovog procenta. Zamka u koju se često upada je odbaciti p_e procenat jedinki sa najnižim odnosom podobnosti odnosno rastojanja. Ovo u većini slučajeva nije najoptimalnije rešenje, sa obzirom da u selekciji treba obratiti pažnju ne samo na podobnost jedinki (međusobno najkraće rastojanje) već i na raznovrsnost budućih generacija.

Ukrštanje je ekvivalent biološkom ukrštanju (parenju). Biramo po dve jedinke i od njih formiramo druge dve - nalik detetu dve roditeljske jedinke. Ovim procesom stvaramo onoliko jedinki koliko nam je potrebno da bi generacija ponovo sadržala *M* jedinki, tj. onoliko koliko smo eliminisali prethodnim korakom. Za izbor roditelja odabira se slučajani broj jedinki iz populacije (sa ili bez zamena), a najbolje od

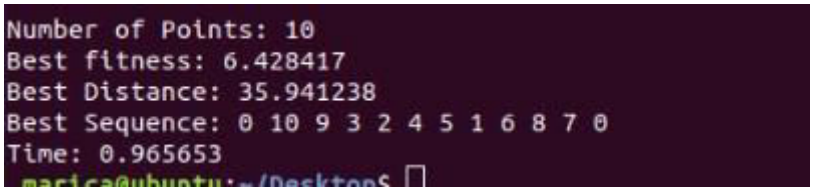
njih postaju roditelji sledeće generacije.
Ukrštanje je sledeći genetski operator koji učestvuje u procesu. U ukrštanju učestvuju dve jedinke, roditelji, a rezultat su deca. Najvažnija karakteristika je da deca nasleđuju svojstva roditelja, dakle, ako je roditelj prošao selekciju dete će biti dobro, a moguće I bolje od roditelja. Ako postoje dva roditelja te jedno ili dvoje dece tada se većinom govori o binarnom operatoru ukrštanja. Ukrštanje može biti definisano s proizvoljnim brojem prekidnih tačaka, stoga postoji ukrštanje s jednom tačkom, ukrštanje s dve tačke i uniformno ukrštanje. Ukrštanje s jednom tačkom razvijeno je da nasumično odabere tačku ukrštanja te se nizovi bitova iza te tačke zamjenjuju između dva roditelja. Kod ukrštanja s dve tačke, nasumično se biraju dve pozicije, a nizovi bitova zamenjuju se između te dve tačke, odnosno pozicije. Uniformno ukrštanje je ukrštanje sa b-1 prekidnih točaka (b predstavlja bitove). Takođe, kod ukrštanja se vrlo često mora kreirati, tj. zadati maska. Maska se zadaje ako je verovatnoća nasljeđivanja različita za pojedine gene, maska definiše verovatnoću nasljeđivanja za svaki gen posebno. Ono što je bitno za operator ukrštanja je to da se pretpostavlja kako je upravo taj operator ono što razlikuje genetske algoritme od ostalih metoda optimizacije.

Mutacija je poslednji korak u jednom ciklusu. U ovom koraku se bira procenat p_i jedinki nasumično, i one se na neki način promene, opet simulirajući mutaciju u biološkoj evoluciji. Značaj ovog koraka je opet povećanje raznovrsnosti populacije, jer se mutacijom dobijaju nove jedinke sa manje ili nimalo korelacije u odnosu na ostale jedinke iste populacije.

4. Rezultati

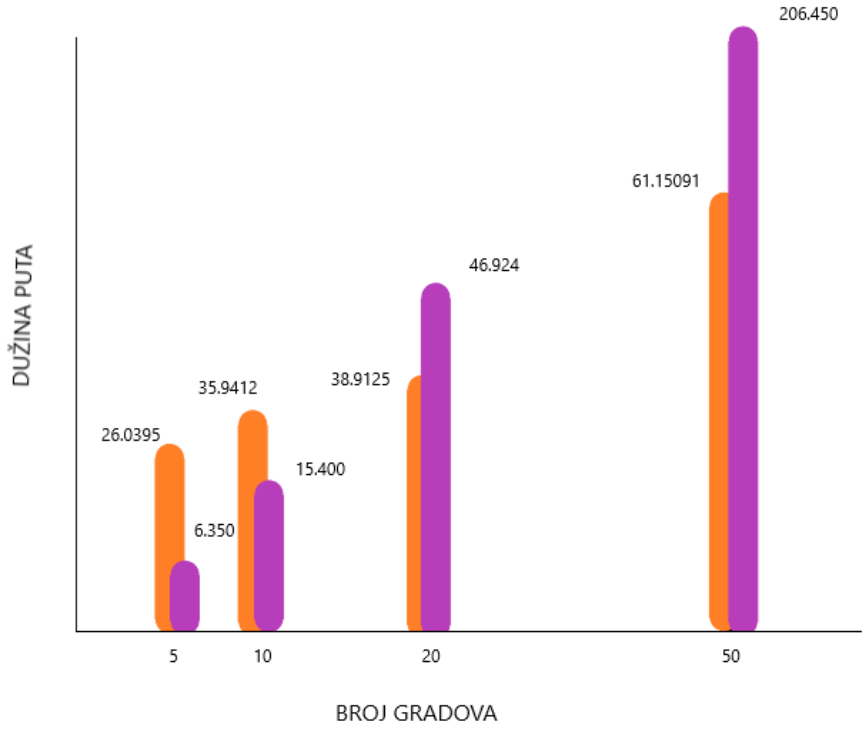
4.1. Tehnički detalji

Program je napisan u programskom jeziku C na operativnom sistemu UBUNTU 18.04.1 LTS i pokreće pomoću Makefile-a. Pri pokretanju programa *ga.c*, unosi se broj gradova. Nakon toga će program ispisati broj gradova, najbolju fitnes funkciju, najkraću rutu, gradove označene brojevima i vreme potrebno da se program izvrši.



4.2 Zaključak

Na osnovu testiranih primera i algoritma, vidi se da GA za manji broj gradova daje optimalno rešenje i to u realnom vremenu. Za veći broj gradova (par stotina), algoritam radi oko 3 minuta. U poredjenju sa algoritmom iz literature [*] dobili smo sledeće rezultate:



Narandžasta boja predstavlja naš algoritam, a ljubičasta GA algoritam iz literature. Iz priloženog vidimo da se naš algoritam lošije ponaša za manje gradova, a bolje za više gradova.

Literatura

Velga Bosancic, Anka Golemac, Tanja Vojkovic – Kako pomoći trgovačkom putniku

D. Applegate, R. Bixby, V. ChvÃ¡tal, and W. Cook. Implementing the dantzigfulkerson-johnson algorithm for large traveling salesman problems. Mathematical Programming, 97:91_153, 2003. ISSN 0025-5610.

T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. Introduction to Algorithms.McGraw-Hill Higher Education, 2nd edition, 2001. ISBN 0070131511.

G. Dantzig, R. Fulkerson, and S. Johnson. Solution of a large-scale travelingsalesman problem. Journal of the operations research society of America, pages 393_410, 1954.

J. Holland. Adaptation in natural and arti_cial systems: an introductory analysis with applications to biology, control, and arti_cial intelligence. MITpress, 1992.

S. Lin and B. Kernighan. An e_ective heuristic algorithm for the travelingsalesman problem. Operations research, 21(2):498_516, 1973.

[*] Selver Pepić , Zoran Lončarević , Goran Miodragović , Slobodan Aleksandrov IMPLEMENTACIJA REŠENJA PROBLEMA TRGOVAČKOG PUTNIKA GENETSKIM ALGORITMIMA U JAVA PROGRAMSKOM JEZIKU