Univerzitet u Novom Sadu

Fakultet tehničkih nauka

Dokumentacija za projektni zadatak

Studenti: Roganović Veselin, SV36/2022

Cvjetković Tamara, SV48/2022

Predmet: Nelinearno programiranje i evolutivni algoritmi

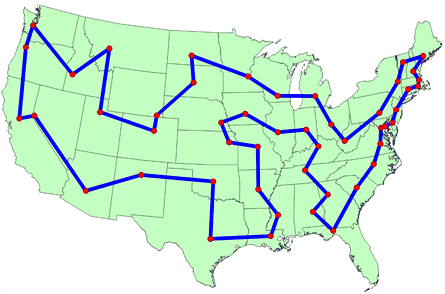
Broj projektnog zadatka: 1

Tema projektnog zadatka: Genetski algoritam, problem putujućeg trgovca

# Opis problema

**Problem putujućeg trgovca** (*Traveling Salesman Problem,* ***TSP***) je problem diskretne i kombinatorne optimizacije. Spada u skup NP-teških problema (problemi čija rješenja mogu da se provjere u polinomijalnom vremenu), a složenost mu je O(n!).

Sam problem je vrlo jednostavan – unaprijed je dat skup gradova u kojem je svaki grad određen pomoću koordinata x i y. Potrebno je odrediti redoslijed kojim bi putujući trgovac obilazio gradove, tako da svaki grad obiđe jednom i svoje putovanje završi u gradu iz kojeg je krenuo, ali da je ukupan pređeni put **minimalan**.



Slika 1: primjer problema putujućeg trgovca

Na prvi pogled ovaj se problem može činiti lakim, ali ako se uzme u obzir da mu je složenost O(n!), računanjem vrlo lako dobijemo da već za 10 gradova postoji 3628800 mogućih redoslijeda obilaženja gradova.

# Uvod

Problem putujućeg trgovca je izuzetno težak problem zbog faktorijelne složenosti, pogotovo ako imamo skup podataka koji broji više od 50 ili 100 gradova. Postoji više rješenja za ovaj problem (pretraga grubom silom - brute force, pohlepni algoritam - greedy, linearno programiranje..), međutim, u ovom projektnom zadatku izabran je **genetski algoritam**, koji je takođe vrlo često rješenje.

Genetski algoritmi (GA) su dio evolucionih algoritama, koji su inspirisani Čarls Darvinovom teorijom prirodne evolucije. Evolucioni algoritmi ne nađu uvijek optimalno rješenje, ali je njihova prednost što u vrlo razumnom vremenu mogu naći rješenje koje je 2% ili 3% lošije od optimalnog.

GA koristi prirodnu selekciju, koja započinje selekcijom „najboljih“ jedinki iz jedne populacije. Oni dalje „proizvode“ jedinke koje koje će naslijediti njihove karakteristike i koje će se dodati u sledeću generaciju. Ako roditelji imaju dobre karakteristike, njihova djeca će imati bolje karakteristike od njih i imaće veću šansu da „prežive“. Ovaj proces se ponavlja, a na kraju dobijemo generaciju u kojoj su jedinke „najbolje“.

Proces možemo podijeliti u sledeće faze:

1. **Generisanje** **populacije** - na osnovu kakva je populacija, postoje dva tipa algoritama: *realni* i *binarni*. U problemu putujućeg trgovca, binarni kod ne bi bio prikladan, jer je često potrebno permutovati cijele gradove.
2. **Selekcija** - Da bi selekcija bila ispravna, a rješenje optimalno, potrebno je da postoji *kriterijum optimalnosti* pomoću kojeg ćemo rangirati sve jedinke, od „najbolje“ do „najgore“. Kod problema putujućeg trgovca, kriterijum je najmanje Euklidsko rastojanje između svih gradova jedne rute. Nakon rangiranja, potreban je način na koji ćemo implementirati proces selekcije (npr. ruletska ili turnirska). Prije procesa selekcije, vrlo je korisno „prenijeti“ nekoliko najboljih jedinki u sledeću generaciju (*elitizam*).
3. **Ukrštanje** - ukrštaju se dva roditelja i dobiju se dvije nove jedinke.
4. **Mutacije** - proces kojim se izbjegava dupliranje jedinki. Predstavlja slučajnu promjenu gena jedinke sa vrlo malom šansom za dešavanje.

Nakon završetka procesa, dobićemo generaciju koja sadrži jedinke sa najboljim karakteristikama, a u našem problemu to predstavlja rutu između gradova sa minimalnim pređenim putem.

# Implementacija

Implementacija ovog problema svodi se na suštinsko prenošenje procesa prirodne selekcije u računarski kontekst. Rješenje problema implementirano je u Pythonu.

Na početku je potrebno učitavanje podataka iz datoteke i smještanje koordinata gradova u rječnik pod odgovarajućim ključem (naziv grada), kojim ćemo lako moći da rukujemo.

Nakon toga, generiše se početna populacija koja predstavlja grupu više nasumično generisanih ruta gradova. Poziva se funckija **generate\_initial\_population(pop\_size)**, gdje proslijeđeni parametar „pop\_size“ označava veličinu generisane populacije.

A black and white math equation

Description automatically generated

Slika 2: udaljenost gradova g1 i g2, sa koordinatama (x1, y1) i (x2, y2)

Početna udaljenost dobija se računanjem kriterijuma optimalnosti svih ruta prvobitne populacije (kao suma Euklidskih udaljenosti između svake dvije tačke), te se one rangiraju i dobijamo niz sortiranih vrijednosti (indekse jedinki u samoj populaciji i udaljenosti) po udaljenosti od najmanje do najveće. Ovaj niz vrijednosti vraća funkcija **rank\_routes(population)**,gdje parametar „population“ označava populaciju koja je proslijeđena, a fitness predstavlja neke rute, što je u stvari u ovom slučaju udaljenost.

A computer code with colorful text

Description automatically generated

Slika 3: funkcija za rangiranje udaljenosti

Rezultat početne i svake naredne udaljenosti se pamti da bi znali kada je rezultat iskonvergirao, a iteriranje kroz sledeće generacije se nastavlja. Svakim iteriranjem se kreira nova populacija funkcijom **create\_new\_generation(cur\_population, elitism\_size, mutation\_rate)**, gdje parametri redom predstavljaju trenutnu (prethodnu) populaciju, broj elitnih jedinki koje će se „prebaciti“ iz prethodne generacije i nivo mutacije. Povratna vrijednost ove funkcije je nova generacija koja se očekuje nakon sledeća tri procesa: **selekcija**, **ukrštanje** i **mutiranje** populacije.

a) Proces selekcije obavlja funkcija **route\_selection(ranked \_population, elitism\_size)**.  
Parametar *ranked\_population* predstavlja rangiranu populaciju prethodne generacije, a *elitism\_size* je broj elitnih jedinki koje će „preživjeti“ ovaj proces selekcije. Za svaku preostalu jedinku se računa *parent\_score*, koji predstavlja njen „skor“ pomnožen sa nasumičnim brojem. Nakon toga se sve jedinke rangiraju po *parent\_score* i izaberu se dvije najbolje koje će biti korišćene kao roditelji za ukrštanje. Ovo se naziva **ruletska** selekcija.

A screen shot of a computer code

Description automatically generated

Slika 4: ruletska selekcija

b) Nakon selekcije, potrebno je ukrštanje, kako bismo dobili „djecu“ jedinke. Funkcija koja obavlja ovaj dio zadatka je **cross\_over\_population(co\_population, elitism)**, gdje parametri redom predstavljaju populaciju koja je izabrana u prethodnom procesu selekcije i broj elitnih jedinki koje će opet biti prve izabrane, kao najbolje. Preostale jedinke se ukrštaju (po dva roditelja izabrana selekcijom) i dobije se po dvoje djece, koje se dodaju u prethodno napravljenu novu populaciju, zajedno sa elitnim jedinkama. Ukrštanje se vrši tako što se od jednog roditelja kopira prvih *border* gena, gdje *border* predstavlja neki nasumičan broj u rasponu od 0 do broj gradova minus 1. Zatim se iz drugog roditelja kopiraju svi preostali geni (gradovi) i nadodaju na kopirane gene iz prvog roditelja i prenose u dijete. Tako se dobija prvo dijete, a za drugo dijete ponavljamo identičan proces samo što prvo gledamo drugog roditelja. Ovaj način ukrštanja je pogodan za ovaj problem zato što se održava jedinstvenost svakog gena (grada) u jedinki (ruti), kao i da je broj gena u jedinki uvijek isti (svi gradovi su dio jedne rute).

c) Poslednji proces stvaranja nove generacije je mutacija, koja se izvodi u funkciji **mutate\_population(population, rate, elitism)**, gdje je *population* populacija nad kojom se radi mutacija (djeca dobijena ukrštanjem), *rate* predstavlja nivo mutacije, a *elitism* broj elitnih jedinki. Elitne jedinke se odabiraju na početku, a onda se vrši proces mutacije nad preostalim jedinkama, gdje se, sa malom (rate) šansom dešavanja, vrši mutacija zamjenom nasumičnih gena.

A computer code with colorful text

Description automatically generated

Slika 5: mutacija zamjenom nasumičnih gena

Dobija se nova generacija, kojoj se računa udaljenost i ispisuje najbolja jedinka (prvi grad se ne ispisuje i na kraju jedinke, smatra se da se povratak u prvi grad podrazumijeva i računa se u funkciji za računanje udaljenosti) i njen rezultat. Proces se ponavlja, nova generacija postaje prethodna, a na osnovu nje se generiše ponovno neka druga nova generacija.

Ako se desi da se tokom procesa neka udaljenost ponovi 30 puta, proces se zaustavlja, jer se pretpostavlja da je tu konvergencija. Takva vrijednost se uzima za konačnu najbolju vrijednost, tj. minimalnu udaljenost.

# Zaključak

Genetski algoritmi pružaju efikasno rješenje za problem putujućeg trgovca, čak i kada se suočavamo sa velikim brojem gradova. Važno je naglasiti da uspješnost genetskog algoritma mnogo zavisi od odabira parametara, uključujući veličinu populacije, vjerovatnoću mutacije, kao i broj elitnih jedinki.

Manja populacija ima veće šanse da "zaglavi" u lokalnom optimumu, ali zato iterira puno brže kroz generacije nego veća populacija. Sa druge strane, veća populacija je korisnija za rješavanje problema sa većim brojem gradova. Ovde je napravljen neki kompromis i izabrano je 500 jedinki za populaciju. Ovaj broj davao je bolje rezultate nego 200 jedinki i brzina algoritma nije mnogo lošija, dok je za broj 1000 algoritam znatno sporiji, a rezultati slični.

Povećana vjerovatnoća mutacije doprinosi raznolikosti jedinki u populaciji i sprečava prebrzu konvergenciju rješenja. Ovde je izabrana mutacija od 3%. Generalno rezultati od 0.5% do 5% su bili vrlo slični, mada je ovaj djelovao najbolje. Mutacija preko 5% donosila je loše rezultate zbog čestih izmjena, a premala mutacija je dovodila do konvergencije u lokalnom minimumu i opet loših rezultata.

Ako se prenese veći broj elitnih jedinki, veća je šansa da se sačuva visok nivo kvaliteta rješenja i da se ubrza konvergencija algoritma ka boljim rješenjima. S druge strane, manji broj elitnih jedinki može omogućiti veću raznolikost u novoj generaciji, ali takođe može dovesti do gubitka najboljih rješenja. Broj najboljih jedinki koje čuvamo kroz generacije kod nas je 20 (4%). Broj 50 donosio je malo raznolikosti, a broj 5 previše malo sačuvanih dobrih jedinki, te je 20 neki kompromis.

Rješenja (udaljenosti) za odabrane parametre su bili brojevi od 8900-13000, najčešće u rasponu od 9500-10500 (80% ukupnog broja testiranih rješenja programa).

Dakle, odabir ovih parametara direktno utiče na balans između očuvanja kvaliteta rješenja i raznolikosti u populaciji, što ima ključnu ulogu u efikasnosti genetskog algoritma.