Univerzitet u Novom Sadu

Fakultet tehničkih nauka

Dokumentacija za projektni zadatak

Student:	Privalov Maksim, SV81/2023
Predmet:	Nelinearno programiranje i evolutivni algoritmi
Broj projektnog zadatka:	14
Tema projektnog zadatka:	Ant colony optimization algoritam, problem putujuceg trgovca

Opis problema

Travelling salesman problem sastoji se u tome da trgovac mora proći kroz sve gradove i vratiti se u polazni grad, trošeći što je manje moguće napora na putu i posjećujući svaki grad samo jednom.

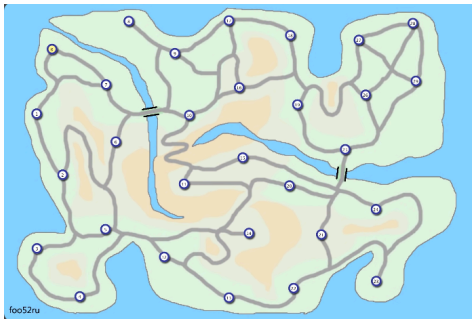
Ako su putovi između gradova jednaki u oba smjera, tada je složenost pretrage $\frac{(n-1)!}{2}$, što dovodi do nepremostive zadatka kada je broj gradova >66, čime se problem ne može riješiti ni za milijarde godina. Zbog toga su osmišljeni algoritmi koji omogućavaju uvjetno rješenje problema uz minimalan napor i vrijeme. Među svim algoritmima mogu se izdvojiti:

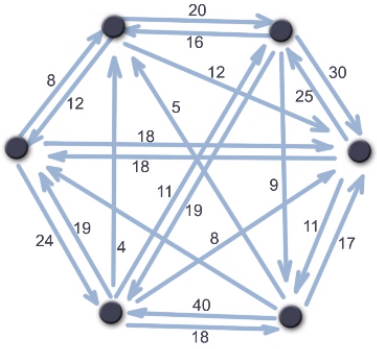
Genetic Algorithm - evolutivna metoda; *Ant Colony Optimization* - metoda "rojevoj" inteligenciji; *Simulated annealing* - metoda imitiranja fizičkih procesa.

U ovom radu razmotriću rješenje problema korišćenjem algoritma **Ant Colony Optimization**

Kada biste mi rekli "rojeva inteligencija", zamislio bih neku genijalnu kraljicu koja kontrolira sve svoje podređene mrave.

Nažalost ili na sreću, stvari stoje drugačije. Kako onda mravi s minimalnim mozgom uspevaju da pronađu najkraće puteve, i to bolje od ljudi? (Čovek može rešiti ovaj problem samo kada vizuelno može proceniti udaljenost između gradova, ali ponekad putevi nisu ravni, i obilazni put može biti kraći od direktnog.)





Sve je jednostavno - oni prolaze kroz sve "gradove" (hrana i td.) prvo prilično haotično, vođeni samo udaljenošću do najbližeg objekta, ali ostavljajući trag od feromona duž puta. Što je konačni put duži, to će manje feromona ostati na putu kojim je mrav prošao. Feromon s vremenom postepeno isparava. Sledeći mrav se ne oslanja samo na blizinu, već i na količinu feromona koju oseća na putu. Nakon nekog vremena, feromona na najkraćem putu će biti znatno više nego na drugim putevima, i mravi će češće birati najkraći put. Pozajmićemo algoritam od naših mrava i implementiracemo njega!

Uvod

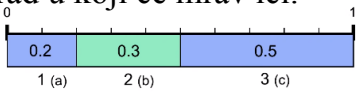
Ant Colony Algorithm

Podaci u našem algoritmu se čuvaju u tabeli veličine n*n (gde je n broj gradova), u svakoj ćeliji tabele postoje vrednosti: d - udaljenost od grada i do grada j i τ - količina feromona na putu. Verovatnoća P prelaska iz čvora i u čvor j se računa kao:

$P_{ij} = \frac{\tau_{ij}^{\alpha}\eta_{ij}^{\beta}}{\sum_{m}\tau_{im}^{\alpha}\eta_{im}^{\beta}}$, gde α, β = const, τ = kolicina feromona, $\eta = \frac{const}{d}$, m = allowed nodes

Dobijamo n verovatnoća prelaska, čija suma iznosi 1 (100%). Postavljamo brojeve na rulet, nasumično dobijamo neki broj od 0 do 1 i određujemo grad u koji će mrav ići.

Primer sa 3 destinacije (a, b, c). Ako je nasumični broj 0.4, ići ćemo do "b":



Dodavanje feromona na stazu kojom je mrav prošao i isparavanje su računaju kao:

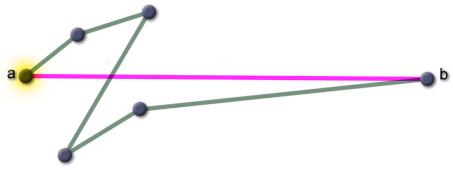
$$\Delta\tau_{ij,k}(t) = \begin{cases} \frac{Q}{L_k(t)}, & (i,j) \in T_k(t), \\ 0, & (i,j) \notin T_k(t), \end{cases}$$

$$\tau_{ij}(t+1) = (1-p)\cdot\tau_{ij}(t) + \sum_{k=1}^{m}\Delta\tau_{ij,k}(t),$$

Δτ = kolicina feromona koj dodamo, k = indeks mrava, t = redni broj iteracije, Q = const, L = dužina svih ruta, p = koeficijent isparavanja feromona.

S obzirom da su gotovo u svim slučajevima koeficijenti τ i η u formuli za izračunavanje verovatnoće prelaska u grad manji ili jednaki 1, konstante (koje se postavljaju ručno) α i β će, pri njihovom povećanju, značajno potisnuti izabrani parametar. Na taj način, povećanjem vrednosti β, najkraći putevi će imati veći koeficijent, dok će udaljeni putevi imati, obrnuto, veoma mali. Isto važi i za konstantu α.

Kod mrava ne postoji "početni" grad, tj. uvek kreću iz različitih gradova, jer ako se stalno kreće iz istog grada, grad koji je najudaljeniji imaće veću verovatnoću da bude izabran poslednji, često stvarajući manje optimalan put. Primer:



Takođe, radi poboljšanja pronalaženja najboljeg puta, algoritam ima modifikacije. U svom programu sam implementirao neke od njih *Ant Colony Algorithm with Elite Strategy in Process Route* - pored običnih mrava, postoje i elitni koji idu samo najboljim rutama, time dodajući veću količinu feromona najboljoj ruti; *Max-Min Ant System* - postavljaju se ograničenja na količinu feromona, na primer, ne može biti manje od 0.1, što značajno proširuje spektar mogućih puteva.

Kriterijum optimalnosti:

Uvedimo dodatnu binarnu promenljivu x_{ij}, koja uzima vrednost 1 ako trgovac ide iz grada *i* u grad *j*, i 0 inače. Onda:

$$F = min \sum_{i=0}^{n} \sum_{j=0}^{n} \left(d_{ij} \cdot x_{ij}\right),$$

Svaki grad mora biti posetjen tačno jednom:

$$g_1 = \sum_{i=0}^{n} x_{ij} = 1, \forall j ; \qquad g_2 = \sum_{j=0}^{n} x_{ij} = 1, \forall i$$

Implementacija

Struktura programa:

1) metoda *build_source_table(filename)*, koja se koristi za kreiranje tabele koristeći datoteku u formatu koji je naveden u zadatku.
2) metoda *solve(table, alpha, betta, n, Q)*, koja sadrži osnovnu logiku i rešenje problema.
3) metoda *updateferomon(table, routes, amountOfFeromonToAdd)*, koja služi za ažuriranje feromona: isparavanje, dodavanje ako je mrav prošao tom stazom i uzimanje u obzir granica feromona (Max-Min Ant System).
4) metoda *update_coefficient(table)*, koja ažurira binarne koeficijente (x_{ij})
5) metoda *draw(file_name, route, length)*, koja crta gradove i krajnju najbolju rutu.

Način implementacije:

Faza 1: inicializacija, metoda *build_source_table(filename)*

Kreira se dvodimenzionalni niz veličine n*n, gde svaki red i predstavlja put od grada i do grada j ($0 \leq j < n$, $0 \leq i < n$). Put - niz od 3 elementa - udaljenosti d između gradova *i* i *j*, količine feromona na putu i bitnog elementa sa vrednostima 0, ako mrav nije prošao tim putem, i 1, ako jeste.

Podaci se upisuju u tabelu pomoću dve petlje: spoljni petlja za *departure* i ugnježdena petlja za *destination*. Računa se evklidska udaljenost *d* između gradova, početna vrednost feromona postavlja se na 0.5, a bitni element postavlja se na 0.

$$d_{ij} = \sqrt{(x_i - x_j) + (y_i - y_j)}$$

Faza 2: algoritm, metoda *solve(table, alpha, betta, n, Q)*

Kreira se string *best_route*, gde će se čuvati najbolji put, *best_length* za najbolju dužinu i `*n*` - čuva broj gradova kako ne bi bilo potrebno više puta ga izračunavati.

Formira se petlja po grupama mrava (250 grupa je dosta), u svakoj grupi ima `*n*` mrava, svaki od kojih započinje svoj put od svojeg grada. Svaka 10 grupa su *elitni mravi*, koji idu samo najboljim putevima. U petlji se formiraju nizovi *routes = []* i *amountOfFeromonToAdd* = []`, jer će feromon na puteve biti dodavan tek nakon što svaki mrav u grupi završi svoj put (stoga je potrebno negde čuvati puteve i feromon za svaki put).

Za svaku grupu se formira petlja kroz svakog mrava, u kojoj se: ažuriraju binarni koeficijenti (postaju jednaki 0, metoda *update_coefficient(table)*), formira niz *probabilities = []* u kojem će se čuvati verovatnoće prelaska iz trenutnog grada u preostale, kreira promenljiva *current_row* koja sadrži indeks grada u kojem se trenutno nalazi mrav, promenljiva za čuvanje puta *route = str(current_row) + " "*, i *length* za čuvanje dužine.

```
# Counting the probabilities of choosing local destination
for road in table[current_row]:
    if not (road[0] == 0) and (road[2] == 0):
        summ += (c / (road[0])) ** betta * road[1] ** alpha

for road in table[current_row]:
    if not (road[0] == 0) and (road[2] == 0):
        w = (c / (road[0])) ** betta
        r = road[1] ** alpha
        probabilities.append((w * r) / summ)
    else:
        probabilities.append(0)
```

Nakon toga slijedi petlja od 0 do *n* dok mrav obilazi gradove:

Formira se promenljiva *summ = 0;* Izračunava se suma verovatnoća prema formuli iz Uvoda (za gradove gde je binarna promenljiva == 0, odnosno za gradove u koje mrav još nije bio) i za svaki put izračunava se verovatnoća prelaska u grad i zapisuje u niz *probabilities*.

Strategija odabira putanja mrava:

Kreira se promenljiva counting = 0, za imitaciju ruleta iz Uvoda.

Ako mrav NIJE elitni: Izračunava se slučajan broj od 0 do 1, i prolazimo kroz petlju *probabilities*, dodajući trenutnu vrednost *P* promenljivoj counting. Kada *random_number* postane manji ili jednak *counting*, to znači da je trenutni grad - onaj grad koji smo odabrali. Brisa se niz *probabilities*, dužina izabrane staze se dodaje dužini, *route* se povećava za broj grada, i petlja se završava sa *break*.

Ako je mrav elitni: Izračunava se najveći P iz niza *probabilities*, i prelazimo u grad sa tom vrednošću *P*. Kad petlja se zavrsava, brišu se niz *probabilities*, dužina izabrane staze se dodaje dužini, *route* se povećava za broj grada.

Izabrali smo grad, sada postavljamo binarne promenljive za taj grad na vrednost 1 u svim delovima tabele:

```
for i, _ in enumerate(table):
    table[i][current_row][2] = 1

# Making the row element 1 on the column we
for i, _ in enumerate(table[current_row]):
    table[current_row][i][2] = 1

current_row = counting
```

current_row sada je broj ovog grada.

Kada mrav stigne do poslednjeg grada, petlja se završava, mrav se vraća nazad u polazni grad, put se dodaje u niz *routes*, proverava se dužina puta, ako je pronađen kraći put od *best_length*: *best_length* i *best_route* se ažuriraju. Računa se feromon koji ćemo nakon obilaska cele grupe dodati na pređeni put, i dodaje se u niz *amountOfFeromonToAdd*. Takođe, ako je mrav bio elitni, na njegov put se dodaje 1.7 puta više feromona (tako da najbolji put bolje miriše).

Faza 3: isparavanje i dodavanje feromona.

Kada poslednji mrav u grupi završi svoj put, ažurira se feromon na granama putem metode *update_feromon(...)*, u kojoj:

1) Feromon isparava sa koeficijentom isparavanja 0.3 i proverava se količina feromona. Ako je količina manja od 0.1, postavlja se vrednost na 0.1.

2) Za svaki put iz niza *routes*: Formira se niz gradova, pravi se petlja kroz niz gradova i počevši od druge iteracije počinje se ažurirati feromon (screenshot).

```
for i in range(len(route_arr)):
    if i > 0:
        table[int(route_arr[i - 1])][int(route_arr[i])][1] \
            += amountOfFeromonToAdd[index]  # Add feromon
        table[int(route_arr[i])][int(route_arr[i - 1])][1] \
            += amountOfFeromonToAdd[index]  # Add feromon
```

Metod vraća izmenjenu tabelu *table*.

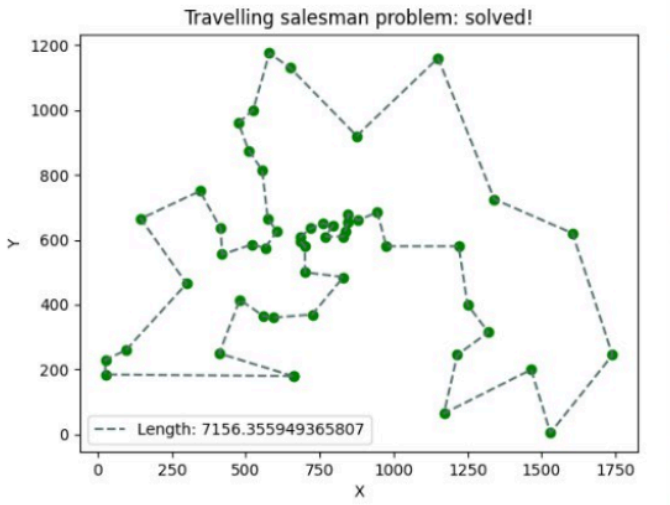
Nakon ažuriranja feromona, kreće se sledeća grupa. Kada sve grupe završe svoj rad, metoda će vratiti promenljive *best_length* i *best_route*, koje ćemo proslediti metodi *draw(file_name, route, final_length)*, gde ćemo koristeći *matplotlib.pyplot* lepo prikazati najbolji put na ekranu.

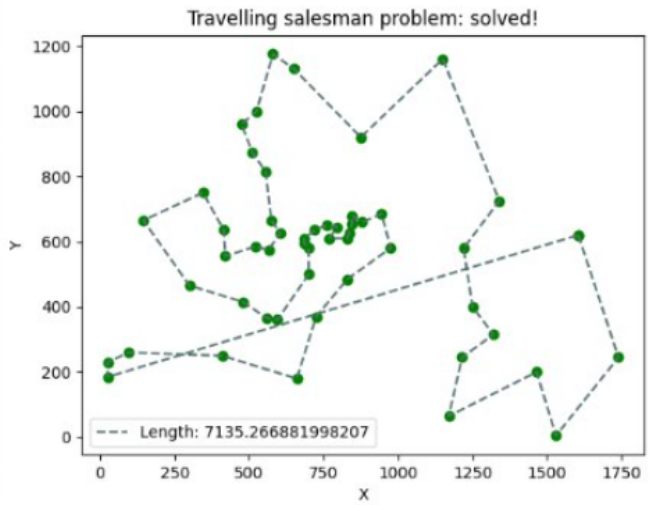
Faza 0: Konstante

Konstante α, β, *c* (broj koji deli dužinu *d* prilikom računanja verovatnoće *P*), *Q* (količina feromona koja će se podeliti na dužinu i dodati feromonu tokom prolaska rute) se biraju eksperimentalno. Potrošio sam 2 dana prateći procentualne odnose τ_{ij}^{α} i η_{ij}^{β} kako bih pravilno odabrao sve ove koeficijente, i u svom radu sam predstavio one koji najbolje odgovaraju našem zadatku.

alpha = 1.0, *beta* = 4.0, *c* = 450, *Q* = 320

Resultat:





Algoritam ne uvek dolazi do ove odluke, ali dosledno prikazuje puteve koji su bliski po dužini.

Zaključak

Ova verzija Python algoritma je konačna; prethodno sam koristio različite strategije obilaska, provere da li je mrav bio u gradu ili ne, i tako dalje. Sa 2 minuta izvršavanja programa (sa 250 grupa mrava) u prvoj verziji, došao sam **do 23 sekundi** izvršavanja u konačnoj verziji. Proces možete pratiti u mom GitHub repozitorijumu.

Rešavajući ovaj interesantan projekat, naučio sam mnogo novih stvari o različitim načinima rešavanja TSP, produbio sam se u ACO algoritam i njegova poboljšanja, naučio sam kako bolje optimizovati kod, istraživati podatke i raditi sa debagovanjem.

Literatura

Thomas Stutzle, Holger H. Hoos - MAX –MIN Ant System

FengYun Huang, ShiQiu Jiang - Research of Ant Colony Algorithm with Elite Strategy in Process Route

Travelling salesman problem - Wikipedia

Yong Wang, Zunpu Han - Ant colony optimization for traveling salesman problem