Univerzitet u Novom Sadu

Fakultet tehničkih nauka

Dokumentacija za projektni zadatak

Studenti:	Kačarević Milan, SV73/2023 Stevanović Aleksandar, SV04/2023
Predmet:	Nelinearno programiranje i evolutivni algoritmi
Broj projektnog zadatka:	1
Tema projektnog zadatka:	Genetski algoritam, problem putujućeg trgovca

Opis problema

Problem putujućeg trgovca (TSP) predstavlja zadatak pronalaženja najkraće moguće rute koja povezuje sve zadate gradove, tako da se svaki grad posjeti tačno jednom, a završetak putovanja jeste povratak u početnu tačku. Cilj je optimizovati ukupnu dužinu puta koji trgovac mora da pređe.

Ako bi se pokušao naivan pristup koji testira sve moguće redoslijede gradova, broj kombinacija za n gradova bi bio n!, što vrlo brzo postaje nepraktično za veći broj gradova. Zbog toga se u praksi umjesto iscrpnog pretraživanja koriste heuristički i evolutivni algoritmi — među kojima je genetski algoritam posebno popularan zbog svoje sposobnosti da u razumnom vremenu pronađe dobro približno rješenje.

Ovaj problem ima široku primjenu u stvarnom svijetu, naročito u oblastima kao što su planiranje logističkih ruta, navigacione aplikacije i upravljanje saobraćajem.

Uvod

Genetski algoritam predstavlja heurističku metodu(pristup rješavanja problema ne garantuje savršeno rješenje, ali pronalazi dovoljno dobro rješenje u razumnom vremenu) inspirisanu principima prirodne selekcije i evolucije. Kroz niz generacija, populacija rješenja se postepeno poboljšava zahvaljujući mehanizmima kao što su ukrštanje, mutacija i selekcija najboljih jedinki.

Umjesto iscrpnog pretraživanja svih mogućih rješenja, algoritam primjenjuje pametnu strategiju pretrage kako bi se efikasno pronašlo optimalno rješenje. U ovom projektu, genetski algoritam je primijenjen kao strategija za rješavanje problema putujućeg trgovca, gdje je cilj pronaći najkraću rutu koja obilazi sve gradove, a da je počena i završna tačka isti grad.

Implementacija

Kako bi se razumio način implementacije genetskog algoritma, neophodno je prvo uspostaviti vezu između osnovnih pojmova teorije i konkretnih programskih struktura korišćenih u radu. U ovom kontekstu, *jedinka* predstavlja jednu moguću putanju koja obilazi sve gradove i implementirana je kao lista objekata klase `City`, gdje svaki objekat sadrži indeks grada (tipa `int`) i njegove koordinate (tipa `float`).

Populacija predstavlja skup takvih jedinki, odnosno niz različitih ruta, i implementirana je kao `list[list[City]]`. Jedna *generacija* označava iteraciju algoritma u kojoj se postojeća populacija razvija kroz mehanizme ukrštanja, mutacije i selekcije pri čemu se zadržavaju najbolja rješenja, a nova se generišu od postojećih. Veličina populacije i broj generacija definišu se unaprijed.

Na početku programa, podaci o gradovima se učitavaju iz tekstualne datoteke u listu objekata klase `City` pomoću metode `load_cities(filename)`. Zatim se formira *matrica rastojanja* između svih parova gradova, uz paralelno mapiranje svakog grada na njegov indeks u listi(metoda `compute_distance_matrix(cities)`. Ovaj korak unapređuje efikasnost algoritma, jer omogućava da se ukupna dužina (metoda `total_distance(path, dist_matrix, city_index))` bilo kog para grada izračuna konstantnim vremenom pristupom unaprijed izračunatim vrijednostima. Zahvaljujući ovoj optimizaciji, značajno se smanjuje ukupna računarska složenost u odnosu na računanje udaljenosti „u hodu" za svaku jedinku tokom svake generacije. Ukupna dužina jedne rute, odnosno zbir rastojanja između uzastopnih gradova koje ruta obuhvata jeste *kriterijum optimalnosti*. Budući da su gradovi predstavljeni u dvodimenzionalnom koordinatnom prostoru, svako pojedinačno rastojanje računa se primjenom formule euklidske metrike između dvije tačke u ravni. Primarni cilj optimizacije jeste minimizacija ukupne dužine puta, odnosno pronalaženje najkraće moguće zatvorene putanje koja povezuje sve gradove.

Formiranje *inicijalne populacije* se vrši pomoću metode `initial_population(cities, population_size)`. Ova metoda prima listu gradova i veličinu populacije za argumente i vraća listu lista

tipa `City`. Inicijalna populacija ili prva generacija se stvaraju permutacijom(bez ponavljanja) svih gradova `population_size` puta. Na kraj svake putanje se dodaje početni grad kako bi se zatvorila ruta. Ruta mora da se zatvori – završi u gradu u kom je započeta jer je to karakteristično za problem putujućeg trgovca.

U implementaciji genetskog algoritma, ključnu ulogu ima *fitnes metoda*, čiji je zadatak da ocijeni kvalitet svake jedinke, odnosno rute, u populaciji. Kriterijum na osnovu kojeg se fitnes određuje jeste ukupna dužina rute, pri čemu kraće rute predstavljaju bolja rješenja. Za svaku jedinku iz populacije izračunava se ukupna distanca, a zatim se iz nje dobija fitnes vrijednost kao recipročna vrijednost te distance. Time se osigurava da manja distanca dovodi do većeg fitnesa, čime se preferiraju rješenja koja nude kraći put. Međutim, same fitnes vrijednosti nemaju smisao bez međusobnog poređenja, zbog čega se vrši normalizacija. Svaki fitnes se dijeli sa sumom svih fitnesa u trenutnoj populaciji, čime se dobija normalizovani fitnes — vrijednost između 0 i 1 koja izražava relativnu vjerovatnoću da će određena jedinka biti selektovana za reprodukciju. Ova vrijednost se direktno koristi u selekcionim mehanizmima, kao što je turnirska selekcija. Računanje ukupnih distanci i odgovarajućih normalizovanih fitnesa implementirano je u metodi `compute_fitnesses(population, dist_matrix, city_index)` koja vraća tuple dvije liste, `distances` i `normalized` koje redom predstavljaju ukupne dužine ruta za sve jedinke i normalizovane fitnes vrijendosti koje služe kao vjerovatnoće selekcije.

U procesu *selekcije jedinki(roditelja)* za dalju reprodukciju, ključnu ulogu ima metod *turnirske selekcije*, koji predstavlja jedan od popularnijih i jednostavnijih mehanizama izbora u genetskim algoritmima. Ova metoda funkcioniše tako što se iz populacije nasumično odabere podskup od `k` jedinki, nakon čega se među njima bira ona sa najvećom vrijednošću normalizovanog fitnesa. Na ovaj način se osigurava da jedinke boljeg kvaliteta imaju veću šansu da budu selektovane za ukrštanje, ali se pritom ne isključuje mogućnost da i slabija rješenja dobiju priliku(u izabranim jedinkama nisu one najbolje), što doprinosi očuvanju raznolikosti populacije. Implementacija turnirske selekcije nalazi se u metodi `tournament_selection(population, normalized_fitnesses, k)` koja kao ulazne parametre prima populaciju jedinki, listu normalizovanih fitnes vrijednosti i veličinu turnira `k`. Povratna vrijednost metode je jedinka (putanja) sa najvećim fitnesom među izabranim kandidatima. Ova metoda se koristi prilikom formiranja nove generacije, pri čemu se odabrani roditelji dalje koriste za ukrštanje i generisanje potomaka.

U fazi *ukrštanja*, za generisanje novih potomaka koristi se metoda *ordered crossover*, koja je prilagođena problemima kao što je problem putujućeg trgovca, gdje je redoslijed elemenata jedinke od suštinskog značaja. Ideja ove metode je da se iz jednog roditelja nasumično izdvoji segment rute (definisan dvjema nasumično izabranim pozicijama), a zatim se taj segment kopira u novog potomka. Nakon toga, preostali gradovi se redom dodaju iz drugog roditelja, pri čemu se preskaču oni koji su već uključeni u potomka, čime se osigurava da se svaki grad u ruti pojavi tačno jednom. Na ovaj način se dobija nova jedinka (dijete) koja sadrži strukturalne karakteristike oba roditelja, ali zadržava validnu permutaciju gradova. Budući da se sve rute u algoritmu tretiraju kao zatvorene (tj. završavaju se ponavljanjem početnog grada), u završnom koraku ukrštanja početni grad se ponovo dodaje na kraj nove rute, jer smo od roditelja uklonili taj posljednji element. Bitno je napomenuti da početni i završni element djeteta ne moraju biti isti kao početni i završni element roditelja. Ova metoda omogućava efikasnu kombinaciju genetskog materijala roditelja uz očuvanje ispravnosti rješenja, što doprinosi postepenom unapređenju kvaliteta populacije kroz generacije. Dakle, metoda ove funkcionalnosti jeste `ordered_crossover(parent1, parent2)` čija je povratna vrijednost lista objekata tipa `City`.

Metoda *mutacije* implementirana je kroz metodu `mutate(path, mutation_rate)` čija je povratna vrijednost `list[City]`,a čiji je cilj da unese određeni stepen varijacije u jedinke unutar populacije i time spriječi prerano konvergiranje algoritma ka lokalnom optimumu. Mutacija se primjenjuje nad putanjom tako što se, uz vjerovatnoću definisanu parametrom `mutation_rate`, izvrši modifikacija jednog segmenta rute. Najprije se iz postojeće putanje privremeno uklanja posljednji grad, koji predstavlja zatvaranje rute. Zatim se nasumično biraju dvije različite pozicije unutar preostale rute, koje definišu granice segmenta koji će biti mutiran. U zavisnosti od dodatnog slučajnog izbora, mutacija se sprovodi na jedan od dva načina: ili se segment između odabranih pozicija reverzuje (tj. obrne redoslijed), ili se njegova

unutrašnja struktura nasumično permutuje. Nakon izvršene mutacije, početni grad se ponovo dodaje na kraj putanje kako bi se očuvala njena zatvorenost. Ova vrsta operatora omogućava da se tokom evolutivnog procesa istraže nove kombinacije gradova u ruti, čime se doprinosi diverzitetu populacije i povećava vjerovatnoća pronalaska boljih rješenja.

Formiranje nove generacije implementirano je kroz metodu `next_generation(parents, parent_dists, children, child_dists, elitism_size)` čija je povratna vrijednosot `list[list[City]]`. Prvo se postojeći roditelji uparuju sa odgovarajućim vrijednostima ukupne distance, nakon čega se sortiraju po rastućem redoslijedu dužina ruta. Na osnovu parametra `elitism_size`, određeni broj najboljih jedinki iz te liste direktno se prenosi u novu generaciju bez ikakvih promjena, čime se osigurava da se najkvalitetnija rješenja sačuvaju. Zatim se, na sličan način, potomci sortiraju po svojoj udaljenosti i bira se onoliko najboljih koliko je potrebno da se popuni preostali dio populacije. Kombinovanjem roditelja i najuspješnijih potomaka dobija se nova generacija čija je ukupna veličina jednaka veličini inicijalne populacije. Ovaj pristup omogućava stabilan napredak algoritma, jer balansira između očuvanja postojećih dobrih rješenja i uvođenja novih potencijalno boljih kombinacija kroz reprodukciju.

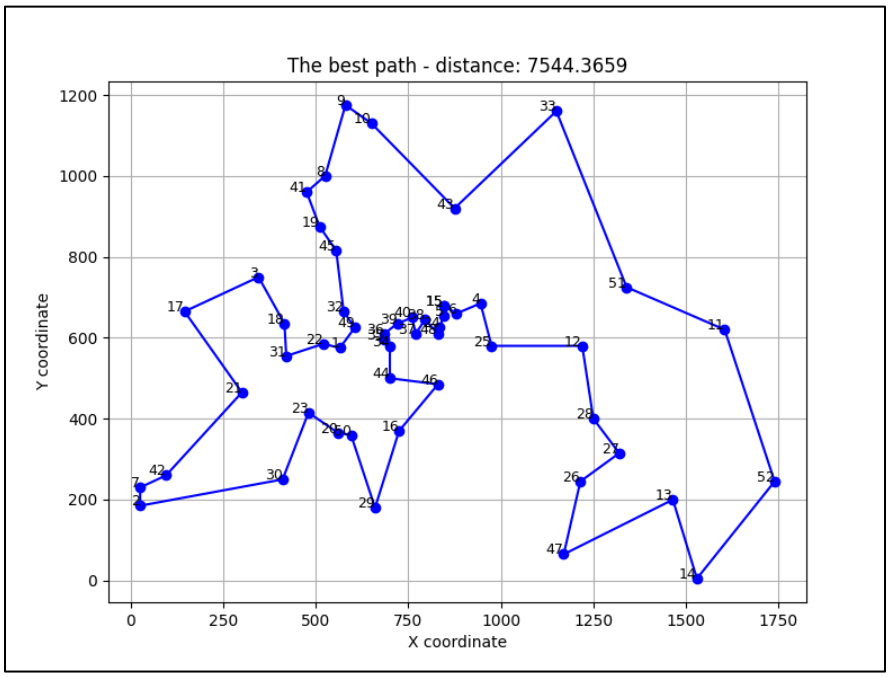
Glavna logika algoritma implementirana je u metodi `genetic_algorithm(cities)` čija je povratna vrijednost `tuple[list[City], float]`. Kao ulaz metoda prima listu objekata klase `City`, dok kao izlaz vraća najbolju pronađenu rutu (kao listu gradova) i njenu ukupnu dužinu izraženu kao realan broj. Na samom početku, vrši se izračunavanje matrice rastojanja između svih parova gradova i njihovo indeksiranje. Zatim se generiše inicijalna populacija jedinki, nakon čega se u svakoj iteraciji (generaciji) računa fitnes svake jedinke na osnovu ukupne dužine rute, a na osnovu normalizovanih vrijednosti fitnesa vrši se selekcija roditelja turnirskom selekcijom. Izabrani roditelji se ukrštaju, dok se generisani potomci podvrgavaju mutaciji sa unaprijed definisanom vjerovatnoćom. Najbolja pronađena ruta se ažurira u svakom koraku ukoliko se dobije rješenje kraće dužine, a broj uzastopnih generacija bez poboljšanja prati se kroz promjenljivu `no_improvement`. U slučaju da broj takvih generacija premaši zadatu granicu (MAX_STAGNATION), algoritam se zaustavlja ranije. Na kraju izvršavanja, metoda vraća najbolju jedinku iz populacije i njenu pripadajuću distancu.

Zaključak

Parametri koji su unaprijed definisani za proslijeđivanje algoritmu su: veličina populacije - POPULATION_SIZE (zadata vrijednost je 700), broj generacija - TOTAL_GENERATIONS (zadata vrijednost je 600), stopa mutacije - MUTATION_RATE (zadata vrijednost je 0.04), broj roditeljskih jedinki koje mogu da prežive ELITISM_SIZE (zadata vrijednost 4) , vrijednost nakon koje će se algoritam prekinuti, ako nije došlo do promjene najbolje jedinke unutar tog broja generacija - MAX STAGNATION (zadata vrijednost 200) i broj jedinki koji se bira u svakom turniru prilikom turnirske selekcije – TOURNAMENT_SELECTION_SIZE (zadata vrijednost 5).

Svi navedeni parametri pažljivo su odabrani kako bi se postigla ravnoteža između efikasnosti algoritma i kvaliteta rješenja. Veća veličina populacije (700) omogućava veću raznovrsnost jedinki i smanjuje rizik od konvergencije ka lokalnom optimumu. Broj generacija (600) pruža dovoljno iteracija da se kvalitetne rute bolje razvijaju. Niska stopa mutacije (0.04) čuva stabilnost dobrih rješenja, a istovremeno uvodi neophodnu varijaciju u populaciju. Uvođenjem elitizma sa 4 jedinke obezbjeđuje se očuvanje najboljih rješenja iz generacije u generaciju. Parametar MAX_STAGNATION postavljen je na 200 da bi se izbjeglo nepotrebno dalje izvršavanje u slučaju stagnacije razdaljine. Konačno, parametar TOURNAMENT_SELECTION_SIZE postavljen na 5 određuje koliko se jedinki nasumično bira da bi među njima bio izabran najbolji za ukrštanje.

Prilikom testiranja programa dolazi do pojave cikličnih rezultata, tj. pojave dvije različite rute obilaska koje imaju istu najkraću distancu kao što je prikazano u nastavku:




```
Generation 540: The shortest distance = 7544.37
Generation 541: The shortest distance = 7544.37
An early stop in a generation 541
The best path: [25, 12, 28, 27, 26, 47, 13, 14, 52, 11, 51, 33, 43,
 10, 9, 8, 41, 19, 45, 32, 49, 1, 22, 31, 18, 3, 17, 21, 42, 7, 2,
30, 23, 20, 50, 29, 16, 46, 44, 34, 35, 36, 39, 40, 37, 38, 48, 24,
 5, 15, 6, 4, 25]
The shortest distance: 7544.365901904086
```

```
Generation 447: The shortest distance = 7544.37
Generation 448: The shortest distance = 7544.37
An early stop in a generation 448
The best path: [15, 6, 4, 25, 12, 28, 27, 26, 47, 13, 14, 52, 1
1, 51, 33, 43, 10, 9, 8, 41, 19, 45, 32, 49, 1, 22, 31, 18, 3,
17, 21, 42, 7, 2, 30, 23, 20, 50, 29, 16, 46, 44, 34, 35, 36, 3
9, 40, 37, 38, 48, 24, 5, 15]
The shortest distance: 7544.365901904086
```

Kao što se vidi na priloženim slikama, postoji cikličnost ruta tj. pojava da dvije različite putanje imaju istu distancu. Inače, distanca 7544.365901904086 je najbolja distanca koja je rezultat izvršavanja ovog algoritma. Neke od ostalih distanci koje predstavljaju rezultat rada algoritma su:

7782.984436148981,	7661.0132721261625,	7916.5118985281815,
8242.371168453326,	8201.593321066197,	7993.165381910276.
8285.806916325237,	8034.177794480282,	
7821.740401666496,	8471.401814377452,	