Univerzitet u Novom Sadu

Fakultet tehničkih nauka

Dokumentacija za projektni zadatak

Student: Ćurčić Aleksa, SV70/2023

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**(Chromosome) predstavlja jednu moguću putanju koja obilazi sve gradove i implementirana je kao lista **gena**, gdje je svaki gen intidžer koji predstavlja identifikator grada.

Populacija(Population) predstavlja skup takvih jedinki, odnosno listu različitih permutacija ruta.

Jedna generacija označava iteraciju algoritma u kojoj se postojeća populacija razvija kroz mehanizme selekcije roditelja, ukrštanja roditelja za dobijanje djece, mutacije dječijih gena i selekcije nove generacije pri čemu se zadržavaju najbolja rješenja, a nova se generišu od postojećih.
Veličina populacije i broj generacija definišu se unaprijed.

Parametri:

FILE_NAME – putanja do fajla iz kojeg se čita data set.
NUM_GENERATIONS – maksimalan broj iteracija algoritma prije nego što zaustavi se algoritam.
POPULATION_SIZE - veličina populacije. Neka dobra praksa je da bude 10x veća nego veličina jedinke.
TOURNAMENT_SELECTION_SIZE – broj jedinki koji se bira za takmičenje u jednom krugu turnira.
MUTATION_CHANCE - šansa za mutaciju jedinke djeteta.
ELITISM_RATE – postotak elitnih jedinki koji se bira pri selekciji sljedeće generacije.
CHROMOSOME_MAX_AGE – maksimalan broj generacija koju jedinka može da proživi. Poslije toga se naprosto izostavlja iz selekcije.
RESET_RATE – postotak jedinki(najlošijih u trenutnoj populaciji) koji će se zamjeniti sa nekim nasumičnim jedinkama radi resetovanja populacije. Ovo se radi kada algoritam ne dostigne napredak kroz određeni broj iteracija, pa ne bi li izašao iz lokalnog minimuma, ako je u njemu zapeo.

Klase:

Geni su zarad jednostavnosti i performansi algoritma predstavljeni kao intidžeri, gdje taj broj predstavlja indentifikator grada.

Chromosome – od polja ima listu gena, starost, kao i fitness.

- Lista gena predstavlja dakle redoslijed kroz koji se obilaze gradovi.
- Starost jedinke predstavlja broj generacija kroz koju je ta jedinka preživila i ograničena je parametrom CHROMOSOME_MAX_AGE.
- Fitness predstavlja ukupnu distancu putanje kroz koju jedinka prolazi. Na fitness se dodaje i udaljenost od posljednjeg do prvog grada, kako bi se napravila kontura.

Parametri konstruktora – lista gena i boolean vrijednost calc_fitness

- Lista gena označava putanju koju jedinka obilazi
- Calc_fitness naznačava da li će se pri inicijalizaciji objekta računati fitness. Calc_fitness se proslijedjuje radi optimizacije, da se fitness ne računa bespotrebno(npr. u slučaju kada se u operaciji crossover-a pravi dijete, a fitness mu je potrebno izračunati tek po njegovoj mutaciji).

Metode:

- Calc_fitness računa fitness jedinke. Što manji fitness, to bolji.
- Is_child vraća da li je jedinka nastala u trenutnoj iteraciji.
- Mutate(zasnovana na displacement metodi) mutira gene jedinke kako bi proširila oblast istraženosti. Nasumična mutacija neće mnogo izmjeniti jedinku, ali će je opet odbaciti negdje kako bi mogla da istražuje još dalje. Displacement metoda radi tako što uzme dva nasumična indeksa, koja predstavljaju podskup putanje, i još jedan indeks gdje će pomjeriti tu podputanju.

Population – od polja ima listu jedinki
Parametri konstruktora – lista jedinki
Metode:

- Get_next_population prolazi kroz niz operatora radi računanja sljedeće populacije. Prvo se radi selekcija roditelja(objašnjeno u nastavku). Potom se od tih selektovanih roditelja radi operator ukrštanja(eng. crossover) da se dobiju djeca(takođe objašnjeno u nastavku). Nakon ukrštanja se radi mutacija dobijene djece i na kraju se radi selekcija sljedeće generacije.
- Get_best_chromosome vraća jedinku sa najboljim fitnesom.
- Mutate_children prolazi kroz proslijedjenu listu djece i za određenu šansu mutira neko od djece.
- Select_population prvo spaja djecu i roditelje u jednu totalnu populaciju za tu generaciju. Sortira ih po vrijednosti fitnesa i potom bira ELITISM_RATE * POPULATION_SIZE elitnih jedinki(mogu biti djeca, mogu biti roditelju, svejedno). Za ostatak nove populacije uzima samo djecu radi napretka algoritma(ne želimo da roditelju dugo ostanu u igri, već da djeca dalje napreduju i istražuju). Select population metoda kao parametar dobija boolean reset vrijednost koja se koristi kada algoritam ne napreduje veliki broj iteracija. Cilj je da se populacija "osvježi" tako što će najgore jedinke zamjeniti sa nasumičnim jedinkama da da algoritmu šansu da se isčupa iz lokalnog minimuma.
- Reset_population dakle resetuje najgore jedinke populacije sa nasumičnim jedinkama.

Selekcija:

TournamentSelection – klasa bez stanja, koja se koristi za selekciju roditelja. Radi po principu turnira(no shit) tako što za svaku od n(veličina populacije) iteracija uzima TOURNAMENT_SELECTION_SIZE nasumičnih jedinki i pušta ih da se takmiče(pobjeđuje ona sa najmanjim fitnesom). Turnirska selekcija je veoma intuitivna metoda za selekciju, a i veoma je jednostavna za implementirati. Može se reći da je pogodna za problem putujućeg trgovca.

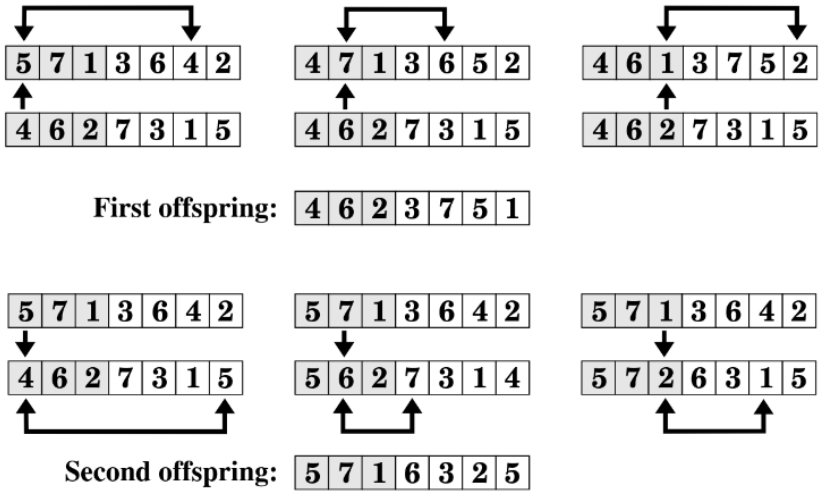
RouletteSelection - takođe klasa bez stanja i isto se koristi za selekciju roditelja. Selekcija je bazirana na rangiranom ruletu. Za proslijedjenu populaciju se najprije izračunaju rankovi(rankiranje po manjoj vrijednosti fitnesa). Potom se za n/2 iteracija, gdje je n veličina populacije, biraju po dva roditelja iz tabele(kako bi na kraju broj izbranih roditelja bio jednak n). Roditelji se ne biraju nasumično(doduše u neku ruku se biraju nasumično), već se gledaju njihovi rezultati(eng. scores). Rezultati se računaju tako što se rang jedinke pomnoži sa nasumičnom vrijednošću u opsegu (0,1). Svakako da veću prednost imaju jedinke sa većim rangom, ali opet i lošije jedinke imaju šanse da budu izabrane.

Ukrštanje(eng. crossover):

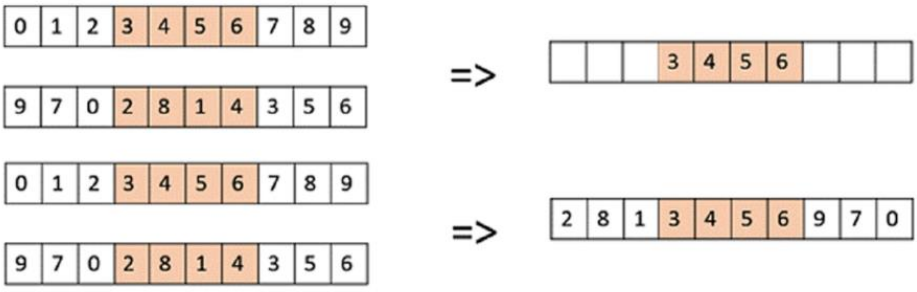
Ponuđena su dva operatora ukrštanja - **PMX**(Partially Mapped Crossover) i **OX1(**Order Crossover 1).

PMX najprije nasumično odredi tačku ukrštanja. Radi ubrzanja i pojednostavljenog rada se prvo mapiraju svi geni od oba roditelja na njihove indekse unutar liste(biće ubrzo objašnjeno zašto). Operator radi tako

što najprije prekopira gene prvog roditelja u prvo dijete, a isto to uradi i sa drugim djetetom. Zatim ide od početka liste do tačke ukrštanja i radi sljedeće: ako se na trenutnoj poziciji u djetetu1 ne nalazi ista vrijednost kao i u roditelju2, onda će u mapi roditelja1 pretražiti gdje je pozicionirana ta vrijednost i zamjeniće je sa trenutnom vrijednošću kako bi se održala validnost jedinke(mapa se u ovom slučaju koristi radi O(1) pretrage indeksa vrijednosti). Isto to će uraditi za i drugo dijete. Primjer:



OX1 bira dva nasumična indeksa za kreiranje podputanje. Ta podputanja će se prekopirati u djecu na istoj poziciji kao i u roditelju(za dijete1 se uzima podputanja iz roditelja1, i tako za dijete2). Potom se iterira od drugog indeksa i radi se umetanje vrijednosti u dijete od suprotnog roditelja(za dijete1 se umeće od roditelja2, i obratno) u onom redoslijedu u kojem su nađene. Kada se iteriranjem dođe do kraja liste, ciklično se vraća na početak. Primjer:



Pomoćne metode:

- Get_starting_population učitava data set iz fajla, pravi listu gena i popunjava globalni riječnik gradova CITIES. Potom poziva funkciju za dobavljanje nasumične jedinke i time pravi inicijalnu populaciju za početak algoritma. Neki su prijedlozi bili da se inicijalna selekcija prve populacije

radi sa heuristikom, međutim zbog nedostatka vremena inicijalna populacija je kompletno nasumična.
- Get_random_chromosome mješa listu gena i vraća nasumičnu jedinku.
- Random_list_index(list_size, current_index) vraća nasumičan index unutar granica liste, ali tako da index nije current_index.
- Get_distance vraća udaljenost izmedju koordinata dva grada.

Zaključak

U okviru ovog projektnog zadatka uspješno je implementiran genetski algoritam za rješavanje problema putujućeg trgovca (TSP), koristeći kombinaciju evolutivnih operatora kao što su selekcija, ukrštanje i mutacija. Prikazana je funkcionalna implementacija osnovnih metoda selekcije (turnirska i rulet selekcija), kao i dva različita pristupa ukrštanju jedinki (PMX i OX1), uz dodatne mehanizme kao što su elitizam, kontrola starosti jedinki i resetovanje populacije.

Rezultat implementacije je fleksibilan i nadogradiv okvir koji omogućava eksperimentisanje sa različitim parametrima i strategijama, što može biti korisno u daljoj analizi i optimizaciji heurističkih metoda. Iako algoritam ne garantuje pronalazak optimalnog rješenja, pokazao je sposobnost da brzo pronađe kvalitetna približna rješenja, što ga čini pogodnim za složene kombinatorne probleme poput TSP-a.

U budućem radu mogla bi se uvesti pametna inicijalizacija populacije pomoću heuristika, dodatna adaptivna kontrola parametara i analiza konvergencije rešenja.