Univerzitet u Novom Sadu

Fakultet tehničkih nauka

Dokumentacija za projektni zadatak

Studenti:	Sladaković Milica, SV 18/2020 Dutina Nemanja, SV 27/2020
Predmet:	Nelinearno programiranje i evolutivni algoritmi
Broj projektnog zadatka:	1
Tema projektnog zadatka:	Genetski algoritam, problem putujućeg trgovca

Opis problema

Problem putujućeg trgovca se sastoji iz traženja najkraće rute koja spaja sve date gradove kako bi se optimizovao (minimizovao) put koji trgovac treba da pređe. Naime, putujući trgovac ima zadatak da prođe kroz sve zadate gradove, a da pri tome najkraći mogući put i na kraju se vrati u početni grad. Ovaj problem, danas je primenljiv na mnoge savremene probleme (optimizacija rute gradskog prevoza, rute dostavljača,...). Zbog toga je rešavanje baznog problema, problema putujućeg trgovca, od izuzetnog značaja. Kako je pristup grubom silom, te prolazak kroz sve moguće puteve veoma neefikasan (za n gradova, postoji n! mogućih puteva), rešavanju ovog problema mora se pristupiti na drugačiji način.

Uvod

Genetski algoritam predstavlja algoritam koji, koristeći simulaciju razvića neke biološke vrste kroz generacije, ima ulogu da optimizuje dati problem pretraživačkom heuristikom. Naš zadatak je bio primena ovog algoritma na rešavanje problema putujućeg trgovca.

Implementacija

Implementaciju genetskog algortima za rešavanje problema putujućeg trgovca započeli smo definicijom pojmova koji su nam neophodni. Jedinku genetskog algoritma predstavlja jedna moguća ruta koja zadovoljava uslove početnog problema. Ona se sastoji od niza gradova koji su predstavljeni klasom *City*, a koja sadrži koordinate x i y svakog grada. Populacija je predstavljena nizom jedinki (tj. mogućih ruta). Njena veličina je unapred zadana, dok je generacija predstavlja jednu iteraciju programa. Na samom početku izvršavanja programa, dolazi do učitavanja niza gradova iz ulazne datoteke i formiranja prve generacije programa.

Formiranje prve generacije programa realizuje se tako što se formira populacija određene veličine koja sadrži jedinke koje su nastale nasumičnim mešanjem redosleda početnih gradova. Faze izvršavanja jedne generacije programa su sledeće: računanje fitnesa trenutne populacije i rangiranje jedinki u odnosu na izračunati fitnes, odabir roditelja za ukrštanje, ukrštanje, mutacije novonastale jedinke i odabir nove populacije.

Kriterijum optimalnosti je udaljenost između svih gradova unutar jedne rute. Kako su gradovi predstavljeni u dvodimenzionalnom sistemu, kriterijum optimalnosti se svodi na sumu Euklidskih udaljenosti između dve tačke u ravni. Cilj optimizacije je naći minimum.

Fitnes svake jedinke predstavlja recipročnu vrednost vrednosti kriterijuma optimalnosti te jedinke. Fitnes se normalizuje u odnosu na ukupni fitnes cele populacije.

Ruletska selekcija se vrši tako što se jedinke rangiraju u odnosu na izračunati fitnes nakon čega se rang svake jedinke množi sa nasumično odabranim brojem koji je između 0 i 1. Tada se za roditeljske jedinke biraju dve jedinke sa najvećim proizvodom. Ovo dovodi do raznovrsnosti u odabiru roditeljskih jedinki, pri tom zadržavajući odnos u kom se jedinke sa boljim fitnesom češće biraju. Ovo nam, zbog te raznovrsnosti omogućava da se i jedinke sa manjim fitnesom ponekad odaberu, a samim tim pokušavamo da izbegnemo situaciju u kojoj ćemo se, zbog biranja isključivo jedinki sa najboljim fitnesom, zaglaviti oko nekog lokalnog optimuma (ovome takođe pomaže i mutacija).

Ukrštanje se odvija nakon biranja roditeljskih jedinki od kojih se tokom ovog procesa stvara nova jedinka. Ukrštanje se vrši na sledeći način: Najpre se iz jednog roditelja uzima nasumično odabran, neprekidan niz gradova i direktno se prenosi u jedinku potomka. Kako su jedinke predstavljene kao niz gradova, nasumično se biraju početni i krajnji indeks. Svi gradovi iz jednog roditelja koji se nalaze između ta dva indeksa predstavljaju neprekidan niz koji se prenosi. Ostatak gradova koji nam nedostaje se redom popunjava iz drugog roditelja. Na ovaj način dobijena je nova, potomačka jedinka koja se dalje prosleđuje mutaciji.

Mutacija je predstavljena kao jednostavna zamena dva grada unutar jedne jedinke. Ukoliko je nasumično izabran broj manji od unapred zadate stope mutacije, dogodiće se mutacija. Tada se nasumično biraju dva indeksa, a zatim se gradovi na tim indeksima zamenjuju.

Formiranje nove generacije predstavlja poslednji korak unutar jedne generacije tokom koga se vrši izbor jedinki koje će preći u sledeću generaciju. Za ovu selekciju iskorišćen je elitizam. Naime roditeljska i potomačka populacija se najpre spajaju i rangiraju. Sada dozvoljavamo unapred zadatom broju jedinki iz roditeljske populacije da prežive i to isključivo ukoliko su bolji od svih potomačkih jedinki. Ostatak populacije ćemo popuniti najboljim jedinkama iz potomačke populacije. Time je formirana nova populacija koja prelazi u sledeću generaciju.

Takođe, kako bismo pratili najbolju jedinku ikad, algoritam najpre čuva najbolju jedinku prve populacije, a zatim, nakon svake generacije, poredi tu jedinku sa najboljom iz novonastale populacije te, ukoliko je nova jedinka bolja, zamenjuje je.

Unutar algoritma nalazi se i brojač koji će se povećavati ukoliko ne dođe do promene najbolje jedinke ikad nakon generacije. Ukoliko taj brojač dostinge unapred definisanu vrednost pre završetka algoritma, smatraćemo da je algoritam ili pronašao optimalno rešenje ili upao u lokalni optimum iz kog ne može da izađe.

Zaključak

Kao parametre koje je potrebno proslediti algoritmu unapred, definisani su: veličina populacije (*POPULATION_SIZE*), broj generacija algoritma (*TOTAL_GENERATIONS*), stopa mutacije (*MUTATION_RATE*), broj roditeljskih jedinki koje mogu da prežive (*ELITISM_SIZE*),

vrednost nakon koje će se algoritam prekinuti, ukoliko nije došlo do promene najbolje jedinke unutar tog broja generacija (*ALGORITHM_STOP*).

Parametar *POPULATION_SIZE* odabran je po formuli deset jedinki za svaku promenljivu, te je, kako se u ulaznoj datoteci nalazi 52 grada (promenljive), izabran broj 520.

Parametar *TOTAL_GENERATIONS* postavljen je na 500, međutim u najčešćim slučajevima algoritam se prekida nakon 100-140 (143, 100, 128, 112, 118, 114, 105, 140, 140, 145, 108) iteracija zbog toga što ne dolazi do promene globalnog optimuma.

Parametar *MUTATION_RATE* je najpre postavljen na 0.2, međutim, ta vrednost u velikom broju pokretanja algoritma se jako teško ili nikako ne izbori sa lokalnim optimumima, te smo, uz testiranje vrednosti od 0.2-0.8, došli do zaključka da se algoritam najbolje ponaša kada je ova vrednost postavljena na 0.4.

Parametar *ELITISM_SIZE* je izabran da bude 5, zbog toga što svaka veća vrednost najčešće dovodi do završavanja algoritma usled dolaska do lokalnog optimuma, a zbog zadržavanja velikog broja roditeljskih jedinki, algoritam ne može da formira potomačku jedinku koja se nalazi van neposredne okoline tog lokalnog optimuma.

Parametar *ALGORITHM_STOP* je postavljen na vrednost 40, zbog toga što smo, nakon testiranja algoritma došli do zaključka da, ukoliko se najbolje rešenje ne promeni u 40 iteracija, to rešenje se ili neće promeniti uopšte ili će se promeniti za vrlo mali, praktično zanemarljiv broj.

Rezultat algoritam se najčešće nalazi između 9800 i 8400 uz mali broj rezultata koji izlaze van tog opsega (8776.651642262277, 9880.965679317545, 9224.348504095025, 8455.883323333752, 8376.214668560946, 9035.928336410925, 9795.356629276599, 9468.262930818582, 9529.822555212313, 10023.989222511347, 8966.038445581173), a kao najbolje rešenje do kog smo došli koristeći ranije definisane parametre je 8327.768027039978.

Obzirom da raznolikost u rešenjima nije prevelika, može se zaključiti da su rešenja prilično uravnotežena, te da implementacija genetskog algoritma daje sasvim solidno rešenje ovog problema.