Genetski algoritem na problemu potujočega trgovca

Tina Bertok
Neža Habjan
Gašper Letnar

17. december 2018

1 Uvod

Naša naloga je, da implementiramo genetski algoritem na problemu potujočega trgovca. Pri tem bomo uporabljali različna križanja, spreminjali velikosti populacije in verjetnosti za mutacijo. Primerjali bomo rezultate pri različnih pogojih. Naš algoritem bomo testirali na podatkih, ki jih bomo generirali sami in na podatkih najdenih na internetu.

Genetski algoritem je metahevristika, navdihnjena s strani procesov naravne selekcije in spada v razred razvojnih algoritmov. Uporablja se za generiranje kvalitetnih rešitev v optimizaciji, ki temeljijo na operatorjih kot so mutacija, križanje in selekcija.
V genetskem algoritmu se uporabi množica kandidatov za rešitev, ki jih nato razvijamo do čim boljše rešitve. Vsak kandidat ima določene lastnosti, katere lahko spremenimo oziroma lahko mutirajo. Evolucija rešitev se ponavadi začne na naključni izbiri kandidatov, katere potem s pomočjo iteracije razvijamo. Na vsakem iterativnem koraku se potem oceni primernost novih kandidatov za optimizacijski problem. Najboljše kandidate potem uporabimo za naslednji korak iteracije. Algoritem se zaključi, ko je izpolnjen zaustavitveni kriterij. Ena možnost je, da algoritem ustavimo po tem, ko preteče določeno število generacij.

Problem trgovskega potnika oz. *traveling salesman problem (TSP)* je NP-težek problem v kombinatorični optimizaciji, pomemben pri operacijskih raziskavah, matematični optimizaciji in teoretičnemu računalništvu. Trgovski potnik mora obiskati določeno množico mest tako, da bo pri tem prehodil čim krajšo pot in se vrniti v izhodišče.

2 Potek in opis dela

Že dobro poznan problem potujočega , smo se odločili predstaviti z grafom v obliki *nxn* matrike cen povezav. Graf smo generirali tako, da smo elemente matrike, ki predstavljajo celoštevilske cene povezav, izbrali naključno.
Nato je bilo potrebno definirati *ciljno funkcijo* oz. *fitness function*, s pomočjo katere bomo ocenjevali primernost rešitev. V našem primeru bo vrednost te funkcije za neko pot kar dolžina te poti.

Naključno smo ustvarili začetno populacijo izbrane velikosti. Vsak element populacije ali *kromosom* predstavlja neko pot, ki obišče vsa vozlišča grafa.

V nadaljevanju izberemo starše, katerih gene bomo uporabili za nastanek naslednje generacije - določimo paritveni bazen. Tega se bomo lotili postopoma in sicer izbiramo po 2 starša za 2 otroka. Za selekcijo imamo dve možnosti, in sicer selekcijo s turnirjem ter proporcionalno selekcijo. Odločili smo se za turnirsko. Vsakega starša bomo izbrali tako, da bomo iz trenutne populacije naključno izbrali *k* kromosomov oz. poti. Naša funkcija oz. *selekcija* nam bo vrnila zmagovalca izmed teh *k* poti oziroma pot z najkrajšo dolžino. Torej bomo za *n* staršev v bazenu imeli *n* turnirjev.

Za ustvarjanje otrok bomo uporabili različna križanja oz. *crossoverje* staršev (poti) iz paritvenega bazena.

Različne variacije križanj:

- urejeno križanje oz. *ordered crossover*
- delno mapirano križanje oz. *partially mapped crossover*
- ciklično križanje oz. *cycle crossover*

Tako bomo dobili otroke, ki so neke nove poti ustvarjene iz dveh staršev. Da pa ohranjamo diverziteto v populaciji, bodo nekatere poti mutirane in sicer s *SWAP mutacijo* (zamenjali bomo dve vozlišči v poti). Vsako vozlišče poti z neko verjetnostjo mutiramo, torej zamenjamo položaj mutiranega vozlišča z nekim naključnim vozliščem te poti.

3 Načrt za nadaljnje delo

V nadaljevanju imamo namen naš program preizkusiti na različnih velikostih populacije in primerjati rezultate. Prav tako bomo opazovali, kaj se spremeni, če uporabimo drugačno različico križanja. Postavili si bomo tudi vprašanje, kakšen bo rezultat, če povečamo verjetnost mutacije.