SVEUČILIŠTE U RIJECI

**TEHNIČKI FAKULTET**

Diplomski sveučilišni studij računarstva

Inženjerstvo kompleksnih programskih sustava

**INICIJALIZACIJSKE FUNKCIJE U GENETSKIM ALGORITMIMA**

2. Kontrolna točka

Rijeka, lipanj 2021. Ingo Kodba

Leo Domitrović

paziti da sve bude na hrvatskom (npr fitness function). slobodno ispravljaj sve što ti oko poželi.

# **UVOD**

Do sada smo implementirali heurističke funkcije najbliži susjed i umetanje, te osnovu genetskih algoritama - mutaciju i elitizam. Tijekom rada smo naišli na greške u starom algoritmu za elitizam pa smo ga uz pomoć asistenta Erika Otovića uspjeli ispraviti.

U ovoj kontrolnoj točki implementirali smo algoritam križanja, točnije Davisovo križanje, Solomonovu heuristiku i prikazivanje detaljne statistike izvršavanja različitih algoritama. Također smo izmijenili inicijalizaciju programa tako da se TSP problem-čvorovi ne skidaju svaki put kod inicijalizacije programa nego smo ih hardkodirali u klasu Gradovi. I dalje ne znamo zašto se TSP problemi nazivaju gradovima i što su te točke koje svaki “grad” sadrži, no to je izvan dosega našeg istraživanja.

# **METODE RJEŠAVANJA ZADATKA**

Za prikazivanje tablica direktno u python programu koristili smo dodatnu knjižnicu tabulate koja tekstualno u konzoli prikazuje tablice. Knjižnicu tabulate instalirali smo pomoću programa pip naredbom “pip install tabulate”.

Za privremenu pohranu rezultata eksperimenata inicijalizirali smo prazno polje oblika (3, 4, 15, 50) što označava broj heurističkih funkcija, broj različitih omjera heuristike i nasumičnosti, broj gradova i broj koliko će eksperimenata biti provedeno za svaku postavku prethodnih triju parametara, respektivno. Inicijalizirali smo polje takvog oblika posebno za fitness function eksperimenata i posebno za vrijeme izvršavanja eksperimenata.

Za trajnu pohranu rezultata koristili smo funkciju iz numpy knjižnice save. Ona nam je omogućila pohranu cijelog 4D polja dimenzija (3, 4, 15, 50) koji se onda sa funkcijom load iz numpy knjižnice u drugom programu može pomoću funkcije tabulate može pregledno ispisati.

Kao što je u radu zadano, za križanje smo koristili poznati order crossover odnosno davisovo križanje koje nakon nasumičnog odabira dvije prijelomne točke prekopira sadržaj prvog roditelja između te dvije točke u novostvoreno dijete, a od druge prijelomne točke pa sve do kraja i s početka ukrug dodaje u dijete čvorove koji još nisu dodani sve dok u djetetu ne budu svi čvorovi iz nekog TSP problema.

Kod implementiranja križanja naišli smo na drastično pogoršanje rezultata. Budući da je šansa za križanjem roditelja 90%, rezultati su nakon nekoliko iteracija križanja, mutiranja i selekcije loši isto kao da bi čvorovi bili nasumično povezani.

Mislili smo da je problem srodan TSP problemu, takav da prvi i drugi roditelji nemaju istu putanju, a to znači da je nemoguće nastaviti uzimati čvorove za dijete iz drugačijeg roditelja bez posljedice velikog porasta fitness funkcije. (bolje opisati problem, možda grafički, slike priložiti)

Misleći da je problem u gornje opisanom slučaju, javili smo se asistentu s pitanjima. Assistant nam je poslao modifikaciju davisovog križanja. Umjesto da je drugi roditelj odabran nasumično (kao što je opisano negdje na internetu), odabran je roditelj koji je sljedeći slabiji po fitness funkciji. Na taj način se križanje odvija između roditelja istog “društvenog statusa”.

Također je dao savjet da elitist-random algoritam populaciju sortira rastuće (prve jedinke imaju najpovoljniji fitness function), pa prvu polovicu rješenja ne provodimo kroz križanje i mutiranje, a drugu da.

Nailazimo na problem u izvršavanju 2-opt algoritma. Algoritam savršeno radi međutim vrijeme izvršavanja je predugo. Za problem Berlin52 nam treba 96 sekundi, dok je u znanstvenom radu to vrijeme 1.71 sekundu. Ako zakomentiramo izvršavanje 2-opta, vrijeme nam se smanji na 10 sekundi.

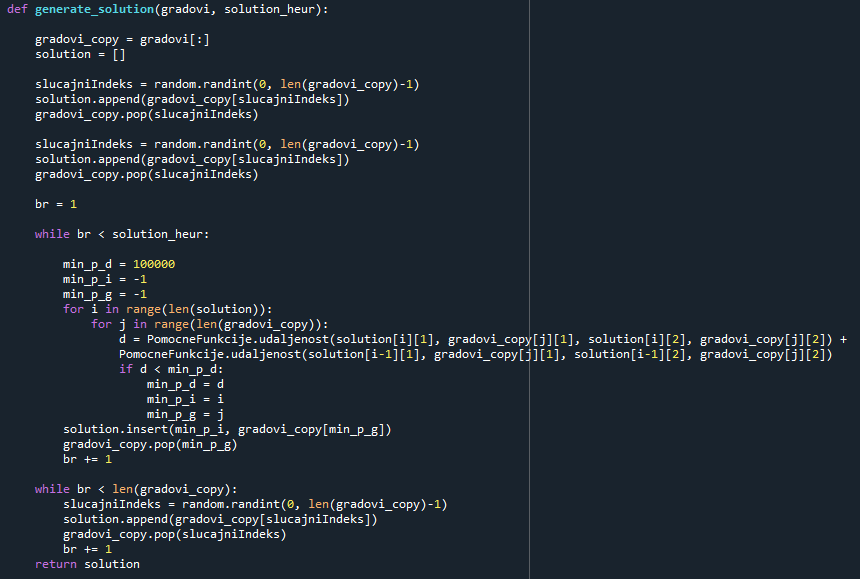
Ovim provođenjem eksperimenta s uključenim 2-optom, očekivano vrijeme za sve eksperimente bi bilo 240 sati (3\*4\*15\*50\*96 sekundi). Molimo odgovor što nam je činiti, da li da onemogućimo 2-opt ili pokušamo napraviti 2-opt manje optimalnim ili nešto treće?

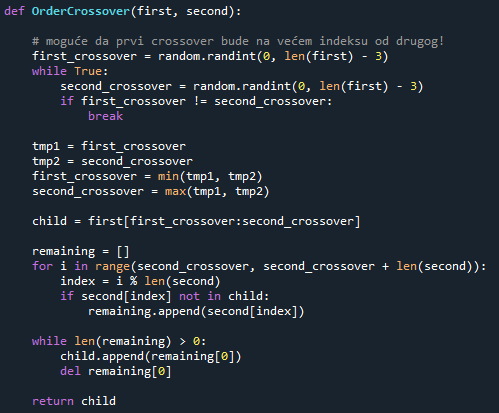
# **REZULTATI**

Do ove kontrolne točke smo završili sa programskim dijelom našeg projekta. Dodali smo implementaciju Solomonove heuristike I1 te križanje. Također, započeli smo sa prikupljanjem rezultata.

Pošto svaku inicijalizacijsku funkciju sa svakim genetskim algoritmom ponavljamo 50 puta ovisno o računalu izvršavanje može potrajati pa smo osigurali pohranu rezultata u datoteku za vrijeme izvršavanja kako bi spriječili gubitak podataka u slučaju nekih nepredviđenih događaja. Npr. u slučaju nestanka struje nećemo morati sve ispočetka izvršavati već samo nastaviti tamo gdje smo stali. Prije izvršavanja je cijelo polje sa rezultatima popunjeno nulama pa tako jednostavno algoritam može znati treba li nešto izvršavati ili je već prije izvršeno. Uz trajnu pohranu ova datoteka nam omogućuje i detaljniju analizu ukoliko je potrebna.

Implementaciju Solomonove heuristike i križanja je najlakše opisati direktno u kodu te se ispod nalaze isječci koda.





# **ZAKLJUČAK**

Sa završenim programskim dijelom projekta slijedi nam izvršavanje algoritama te detaljna analiza rezultata kao i provođenje Friedmanovog testa.

# **LITERATURA**

[1] <https://towardsdatascience.com/how-to-easily-create-tables-in-python-2eaea447d8fd>

[2] https://www.tutorialspoint.com/genetic\_algorithms/genetic\_algorithms\_crossover.htm