Optimizacioni algoritmi za generisanje proizvoljnih grafova sa željenim svojstvima

Seminarski rad u okviru kursa
Računasrka inteligencija
Matematički fakultet

Anđela Damnjanović
mi19059@alas.matf.bg.ac.rs

7. januar 2024.

Sažetak

Grafovi su veoma značajni kako sa teorijskog, tako i sa praktičnog stanovišta. Oni nalaze svoje primene na mestima gde bi malo ko mogao pomisliti — od navigacije pa sve do generisanja rasporeda časova. Sa njima se, iako nesvesno, svakodnevno susrećemo. Upravo zato su razvijeni brojni algoritmi koji rešavaju grafovske probleme. No, kako se ne mogu svi algoritmi primeniti na sve grafove, nekad je poželjno imati softver koji će generisati graf na osnovu željenih svojstava. Upravo to je centralna tema ovog rada, tj. cilj je napisati optimizacioni algoritam koji će generisati graf na osnovu željenih svojstava.

Sadržaj

1 Uvod

Grafovi predstavljaju pogodnu strukturu podataka za modelovanje najrazličitijih problema na koje ljudi nailaze. Mogu se koristiti za pravljenje modela molekula u hemiji, za modelovanje povezanosti korisnika na društvenim mrežama, za vizualizaciju povezanosti mesta na geografskoj karti i tako dalje. Stoga ni ne treba da čudi što su svoju primenu našli i u računarstvu — grafovi imaju veoma širok spektar primena poput klasterovanja, istraživanja podataka, segmentacije slika [6]. Naravno, ovo su samo neke oblasti računarstva gde su grafovi korisni.

Za sve probleme koji se mogu predstaviti grafovima, definisani su odgovarajući algoritmi koji ih rešavaju. Međutim, ne mogu se svi algoritmi primeniti na sve grafove. Na primer, Dajkstrin algoritam neće raditi ako u grafu postoje grane sa negativnom vrednošću težine ili ako je graf netežinski, nalaženje minimalnog razapinjućeg stabla nije moguće primeniti na usmerene grafove, a nalaženje Ojlerovog ciklusa neće biti moguće ako je broj čvorova sa neparnim stepenom različit od 0 i 2 [2]. Kako ljudi ne žele da razmišljaju da li će nad njihovim grafom moći da se primeni neki od algoritama, bilo bi lepo imati softver koji će moći da generiše onakav graf kakav korisnik želi. Na taj način, korisnik ne mora da troši vreme razmišljajući na primer o stepenu čvorova ako može mašini da kaže da je potrebno generisati graf sa parnim stepenima čvorova.

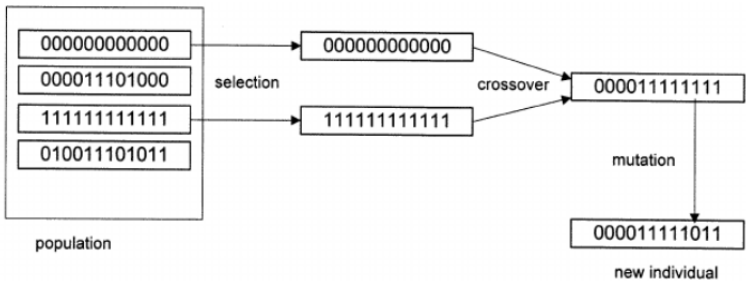
Upravo je ovaj problem poslužio kao inspiracija za pisanje ovog rada. U nastavku će biti opisani dosadašnji rezultati poznatih matematičara u sekciji 2, biće izveden odgovarajući algoritam u sekciji 3 koji će biti testiran na različitim ulazima u delu 4, te će se na osnovu njih donostiti zaključak u sekciji 5.

2 Opis problema i dosadašnji rezultati

Kao što se iz uvoda 1 može zaključiti, ideja je napisati program koji će generisati graf kakav korisnik želi. Međutim, kako je sam kurs posvećen optimizacijama, zadatak ovog rada će biti konstruisati *optimizacioni* algoritam, tj. algoritam koji minimizuje/maksimizuje ciljnu funkciju. Samim tim, ovi algoritmi ne moraju pronaći optimalno rešenje, već mogu vratiti rešenje koje je skoro optimalno [4]. Algoritam koji će biti korišćen za izradu ovog rada je *genetski algoritam*.

Genetski algoritam je optimizacioni algoritam koji je zasnovan na procesu evolucije i prirodne selekcije [1]. Pripadaju metaheuristikama zasnovanim na populaciji, što znači da se najpre pravi cela populacija jedinki, pa se iz te populacije, na kraju, kao rešenje problema, dobije najbolja jedinka. Pošto algoritam oponaša prirodnu selekciju, potrebno je definisati i funkcije koje će oponašati osnovne aspekte prirodne selekcije: selekciju, ukrštanje, mutaciju i preživljavanje najboljih jedinki. Nakon kreiranja populacije, logično je da će najbolje prilagođene jedinke u tom trenutku preživeti i dočekati sledeću generaciju, pa je najpre potrebno pronaći određeni broj najboljih jedinki i staviti ih i u narednu generaciju. Zatim, vrši se proces selekcije — iz cele populacije bira se određeni podskup iz koga će biti izabrana najbolje prilagođena jedinka. Selekcija se vrši dva puta, a dobijene jedinke poslužiće kao roditelji za dobijanje jedinke u sledećoj generaciji. Naravno, treba napomenuti da se selekcijom neće nužno izabrati dve najbolje jedinke iz celog skupa. Zatim, sledi faza ukrštanja u

kojoj nova jedinka dobija po deo gena od oba roditelja. Ova faza se sprovodi u nadi da će nova jedinka naslediti dobre osobine od svojih roditelja i imati još veći stepen prilagođenosti od njih. Naravno, ovo se može desiti, ali i ne mora. Na kraju, na jedinke populacije može uticati i mutacija. Promene u genima mogu imati negativne posledice i smanjiti prilagodljivost jedinke, ali mogu imati i suprotan efekat. Na slici 1 može se videti kako izgledaju procesi selekcije, ukrštanja i mutacije na primeru gena.



Slika 1: Ilustracija procesa selekcije, ukrštanja i mutacije [5]

Naravno, ne treba izostaviti ni dosadašnje uspehe matematičara i informatičara. Tim sa francuskog Univerziteta u Parizu uspeo je da razvije aplikaciju koja koristi upravo evolucione algoritme da na osnovu korisničkog unosa o željenim svojstvima grafa generiše nekoliko mogućih grafova koji imaju željena svojstva. Njihov pristup se razlikuje od pristupa koji je korišćen u ovom radu. Pristup koji su francuski naučnici koristili podrazumevao je da se optimizuju hiperparametri koji su korišćeni za generisanje grafa, dok je u ovom radu optimizovana vrednost funkcije cilja. Pomenuti projekat može se pronaći ovde.

3 Implementacija

Nakon što je problem definisan, potrebo je još opisati koja svojstva grafa će korisnik moći da podešava pre nego što se pređe na opis implementacije. Odlučeno je da se odgovarajući graf generiše na osnovu sledećih osobina: broj čvorova, prosečni stepen čvora, prosečni koeficijent klasterovanja grafa, broj komponenata povezanosti i koeficijenta tranzitivnosti. Radi razumevanja algoritma i rezultata, potrebno je malo detaljnije razumeti navedene osobine:

- Čvorovi u grafu su osnovne jedinice od kojih se formiraju grafovi, pa određivanjem *broja čvorova* korisnik određuje koliko ćvorova želi u svom grafu
- *Stepen čvora* govori o tome koliko je grana incidentno sa tim čvorom u grafu. *Prosečan stepen čvora*, kao što mu ime kaže, govori o tome koliko je grana incidentno sa jednim čvorom grafa u proseku. To ostavlja mogućnost da jedan čvor nema nijednu granu incidentnu sa njim, dok neki drugi čvor ima dosta incidentnih grana, čime se može povećati broj komponenata povezanosti ukoliko je to potrebno.
- *Prosečni koeficijent klasterovanja* je mera stepena do kojeg čvorovi grafa teže da se grupišu, tj. sklonost da dva suseda istog čvora takođe budu susedi [3]

- *Komponenta povezanosti* je podgraf početnog grafa za koji važi da u njemu postoji put između svaka dva čvora, pa je *broj komponenata povezanosti* broj takvih komponenata u grafu
- *Stepen tranzitivnosti grafa* je udeo svih mogućih trouglova u grafu. On se računa kao: $T = \frac{brTrouglova}{brTrijada}$, gde je trougao skup od tri čvora koji su svi međusobno povezani, a trijada podgraf grafa koji se sastoji od tri čvora.

Sada kada su dati i opis problema i opis osobina grafa koje će biti razmatrane, može se preći na detalje implementacije algoritma. Postoje dve opcije za pokretanje algoritma: jedan jeste poziv sa željenim argumentima, dok druga opcija podrazumeva učitavanje ulaznih vrednosti iz unapred pripremljenih fajlova i tek onda pozivanje funkcije sa odgovarajućim argumentima. U ovom radu su pokrivena oba načina. Na samom početku, učitane su vrednosti iz odgovarajućih dokumenata, nakon čega su date vrednosti organizovane u liste koje će kasnije služiti da bi se algoritam izvršavao u petlji za sve učitane ulazne vrednosti.

Nakon određivanja ulaznih parametara, poziva se funkcija

```
1000 def ga(num_nodes, avg_degree, avg_cluster_coeff, population_size,
         num_connected_components, transitivity, num_generations,
         tournament_size, elitism_size, mutation_prob):
```

koja radi tako što generiše populaciju i vrši procese selekcije, ukrštanja i mutacije, kao što je opisano u delu 2 i na kraju vraća najbolje prilagođenu jedinku kao rezultat.

Sama klasa *Individual*, koja odgovara jedinki, implementirana je korišćenjem standardnog šablona za implementaciju genetskog algoritma. Najznačajniji detalj ove implementacije jeste način na koji će se jedinka reprezentovati i način za računanje nivoa prilagođenosti jedinke, tj. implementacija *fitness* funkcije. Kao kôd jedinke, odlučeno je da se čuva odgovarajući graf. Početni graf dobijen je pozivom funkcije

```
1000 self.graph = nx.erdos_renyi_graph(self.num_nodes, 0.5)
```

koja generiše Erdoš-Renji graf [1]. Kao što se može videti, broj čvorova generisanog grafa odgovara broju čvorova koji je korisnik zadao. Još jedan bitan momenat jeste način računanja vrednosti prilagođenosti jedinke. Kako se graf generiše na osnovu želja korisnika, jedino bi imalo smisla da se meri upravo odstupanje od željenog grafa. Tako je i urađeno. Međutim, dâ se primetiti da ne utiču svi zahtevi korisnika podjednako na grešku. Naime, broj čvorova i komponenti su celi brojevi koji mogu biti veoma veliki jer je biblioteka *networkx* dovoljno moćna da i pod tim uslovima generiše odgovarajući graf, dok su vrednosti prosečnog koeficijenta klasterovanja grafa i stepen tranzitivnosti iz intervala [0,1] pa znatno manje utiču na grešku. Stoga je broj čvorova ograničen na 100, prosečan stepen čvora na 50, broj komponenti na 5 i sve potrebne vrednosti su sklairane tako da svaka komponenta proizvodi grešku iz intervala [0,1] i odgovarajući kôd za izračunavanje fitness vrednosti glasi:

[1] Nasumični graf koji se sastoji od zadatog broja čvorova, dok je mogućnost da se grana između dva čvora pojavi u grafu p=0.5

```
return -((abs(num_of_graph_nodes - self.num_nodes))/max_nodes +
    abs(graph_avg_degree - self.avg_degree)/max_degree + abs(
    graph_clustering_coefficient - self.avg_clustering_coeff) +
    abs(graph_transitivity - self.transitivity)+ (abs(
    graph_connected_components-self.num_components)/
    max_components))
```

Naravno, kao i u svakom genetskom algoritmu, najbolja je ona jedinka koja ima najveću vrednost prilagođenosti.

Što se tiče funkcije koja imitira selekciju, ona ostaje nepromenjena, dok su funkcije koje mutiraju gene i vrše ukrštanja promenjene. Prilikom ukrštanja, pošto je gen zapravo graf, potrebno je da dete jedinka nasledi deo grafa od jednog, a drugi deo grafa od drugog roditelja. Da bi to bilo moguće, rešenje koje je ilustrovano u ovom radu podrazumeva da se jedan deo matrice incidencije nasledi od jednog roditelja, a drugi od drugog. Zato je potrebno najpre roditeljske grafove predstaviti kao matrice incidencije, u čemu pomaže funkcija

```
nx.to_numpy_array(parent1.graph)
```

nakon čega dete jedinka dobija genom od oba svoja roditelja. Količinu genoma koji će dobiti od svakog od roditelja odabrana je nasumično, generisanjem dve nasumične vrednosti koje predstavljaju poziciju u matrici do koje se nasleđuje materijal jednog od roditelja, a od koje se nasleđuje genom drugog roditelja. Na kraju, potrebno je dobijene matrice incidencije ponovo pretvoriti u graf, za šta služi funkcija

```
nx.from_numpy_array(adj_mat_cld1)
```

Sličan pristup korišćen je i prilikom implementacije funkcije mutacije. Najpre je graf prebačen u matricu incidencije, koja je zatim menjana u zavisnosti od toga da li je nasumično generisana vrednost manja ili veća od verovatnoće da dođe do mutacije. Na kraju, izmenjena matrica je vraćena u grafovski oblik.

Ovime je zaokružena priča o implementaciji algoritma. Jedino je još ostalo pokrenuti algoritam za konkretne vrednosti i zabeležiti rezultate, te ih na kraju analizirati, što će i biti urađeno u narednom poglavlju ovog rada.

4 Eksperimenti

Nakon upoznavanja sa algoritmom, ovo poglavlje biće posvećeno testiranju programa i analizi dobijenih rezultata. Kao što je i napomenuto u prethodnoj sekciji, ulazi za algoritam učitani su iz odgovarajućih fajlova koji se nalaze u folderu *testPrimeri*. Svaki dokument imenovan je sa *testN.txt*, gde je N redni broj ulaza i uzima vrednosti iz skupa {1,2,3,4,5,6,7}.

Kao broj čvorova odabrane su simbolične vrednosti: 6, 10, 25, 50, 75 i 100, dok su vredosti ostalih parametara odabrani na osnovu prethodno nasumično generisanih grafova. Radi detaljnije analize, sve ulazne vrednosti biće testirane na populaciji od 50 i od 100 jedinki. Rezultati izvršavanja

programa mogu se naći unutar foldera *rezultati*. Za svaki test primer generisan je po jedan fajl koji u svom nazivu sadrži informaciju koji je željeni broj čvorova grafa i na kojoj je populaciji testiran radi lakšeg snalaženja. Svaki ovaj dokument sadrži informacije o željenim karakteristikama grafa koje je korisnik zadao, kao i vrednosti datih parametara nakon generisanja finalnog grafa i vrednost *fitness* funkcije. U tabeli 1 mogu se naći informacije o vrednosti *fitness* funkcije za odgovarajući broj čvorova i veličinu populacije (detaljniji rezultati mogu se naći u pomenutim fajlovima u direktorijumu *rezultati*).

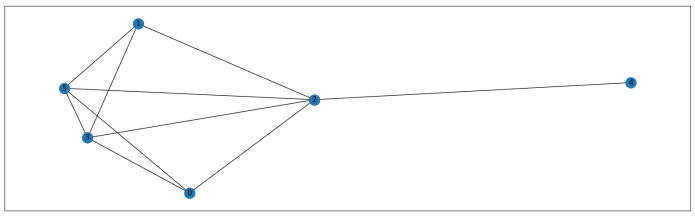
Tabela 1: Vrednosti fitness funkcije

Broj čvorova	**Veličina populacije**	**Fitness**
6	50	-0.005622222222222093
6	100	-0.005622222222222093
10	50	-0.02
10	100	-0.015999999999999997
10	50	-0.01085664682539699
10	100	-0.006736282964629461
25	50	-1.2832926628926629
25	100	-1.2528776855378152
50	50	-0.45898283539623946
50	100	-0.44200014007646093
75	50	-0.015890695284337637
75	100	-0.007531980904111335
100	50	-2.597159336271236
100	100	-2.6122425497135984

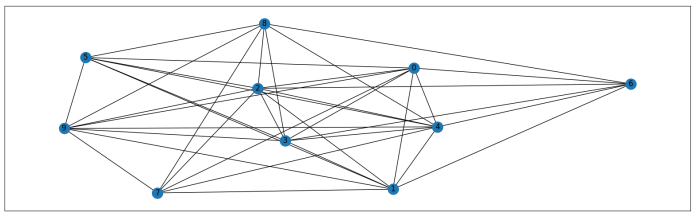
Kao što se iz navedene tabele može videti, vrednosti funkcije su veoma bliske bilo da je prvobitna populacija imala 50 ili 100 jedinki. Međutim, zanimljiva je činjenica da je negde manja populacija dala bolje rezultate nego veća (kao na primer u jednom od slučajeva sa 10 čvorova [2] i u primeru sa 100 čvorova), dok se negde rezultati čak i potpuno poklapaju (primer sa 6 čvorova).

Dalje, može se zaključiti da, *za postojeće test primere*, broj čvorova ne utiče na vrednost fitness-a, jer su grafovi sa 50 i 75 čvorova koji su dobijeni bliži željenim grafovima od grafa sa samo 25 čvorova. Takođe treba pomenuti da su sve dobijene vrednosti relativno niske, osim slučaja sa 25 i sa najvećim brojem čvorova, što bi bilo zanimljivo dalje analizirati. No, pre toga, na slikama ispod mogu se naći neki od generisanih grafova (ostatak rezultata može se naći u direktorijumu *slike*) i to: na slici 2 graf sa 6 čvorova, na slici 3 graf sa 10 čvorova, na slici 4 graf sa 50 čvorova i na slici 5 graf sa 100 čvorova.

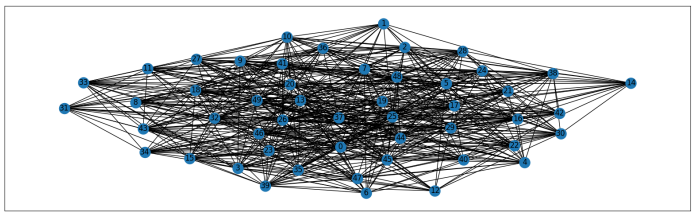
[2] Iako može delovati da se podaci u tabeli ponavljaju, to nije slučaj, jer je u dva test slučaja samo broj čvorova isti, dok se parametri poput prosečnog stepena čvora razlikuju



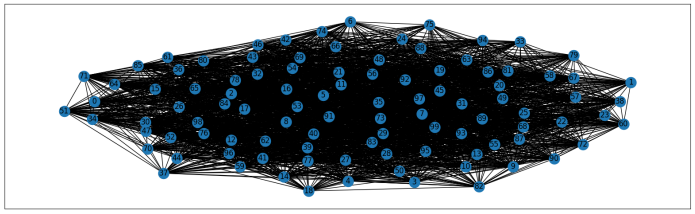
Slika 2: Graf sa 6 čvorova dobijen iz populacije od 50 jedinki



Slika 3: Graf sa 10 čvorova dobijen iz populacije od 100 jedinki



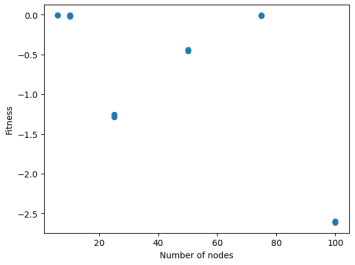
Slika 4: Graf sa 50 čvorova dobijen iz populacije od 100 jedinki



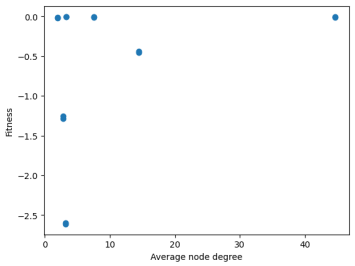
Slika 5: Graf sa 100 čvorova dobijen iz populacije od 50 jedinki

Sada, može se postaviti pitanje da li postoji linearni model koji bi na osnovu vrednosti nekog od mogućih parametara mogao da predvidi

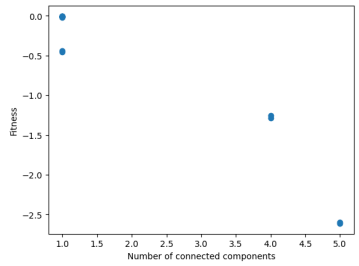
vrednosti fitness funkcije. Ukoliko bi odgovor bio potvrdan, to bi pomoglo u razumevanju zašto su dva pomenuta primera dala toliko loše rezultate. Dati test je sproveden i dobijeni su sledeći rezultati: na slikama 6, 7, 8, 9 i 10 mogu se redom videti grafici zavsnosti vrednosti fitness funkcije od broja čvorova, prosečnog stepena čvora, broja komponenti povezanosti, vrednosti klaster koeficijenta i tranzitivnosti.



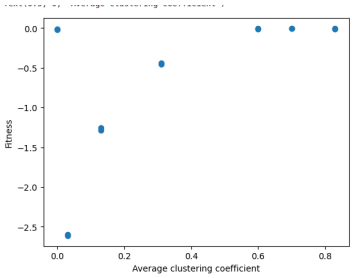
Slika 6: Graf zavisnosti fitness funkcije od broja čvorova



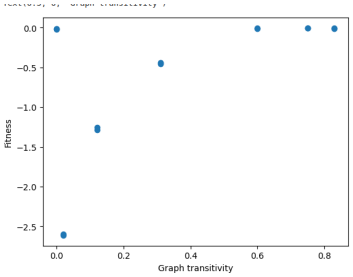
Slika 7: Graf zavisnosti fitness funkcije od prosečnog stepena čvora



Slika 8: Graf zavisnosti fitness funkcije od broja komponenti povezanosti



Slika 9: Graf zavisnosti fitness funkcije od klaster koeficijenta



Slika 10: Graf zavisnosti fitness funkcije od tranzitivnosti

Dakle, odgovor na postavljeno pitanje je odričan, tj. ne postoji pojedinačni faktor od koga će vrednost funkcije zavisiti. No, ipak postoji mogućnost da se sazna koji elementi najviše doprinose — matrica korelacije. Ona može pokazati kolika je korelacija između vrednosti fitness funkcije i ostalih željenih osobina, ali i korelaciju između svaka dva para osobina, što takođe može biti veoma korisno. Izgled matrice povezanosti dobijene na test primerima može se videti na slici 11. Kao što se sa slike može videti, vrednost fitness funkcije je najjače korelisana sa tranzitivnošću i koeficijentom klasterovanja, a nešto manje sa prosečnim stepenom čvora, dok je korelacija sa ostalim elementima veoma mala. Sa ciljem da se navedena tvrdnja testira, dodata su dva nova test primera. Kod oba navedena test primera, promenjeni su prosečni stepen čvora, klaster koeficijent, tranzitivnost i broj komponenti u odnosu na polazni primer. I zaista, dobijeni su bolji rezultati: u jednom primeru krajnja vrednost fitness funkcije iznosi -0.4521703030374372 što je već znatno poboljšanje, dok je u drugom primeru dobijena još bolja vrednost: -0.19286999575305058. Dakle, dobar izbor vrednosti tranzitivnosti, koeficijenta klasterovanja i prosečnog stepena čvora zaista najviše utiču na krajnji rezultat. Vrednosti koje su doprinele poboljšanju rezultata su redom:

```
1000 res = ga(100, 36, 0.43, 2, 0.45, 100, 20, 7, 10, 0.05)
     res = ga(100, 42, 0.47, 1, 0.53, 100, 20, 7, 10, 0.05)
```

```
Correlation matrix is :
                    fitness  num_of_nodes  avg_degree  num_components  \
fitness            1.000000     -0.658670    0.323508       -0.958681
num_of_nodes      -0.658670      1.000000    0.442679        0.521044
avg_degree         0.323508      0.442679    1.000000       -0.354475
num_components    -0.958681      0.521044   -0.354475        1.000000
transitivity       0.615751     -0.329264    0.343455       -0.602434
clustering_coeff   0.608040     -0.311776    0.362305       -0.594298

                   transitivity  clustering_coeff
fitness                0.615751          0.608040
num_of_nodes          -0.329264         -0.311776
avg_degree             0.343455          0.362305
num_components        -0.602434         -0.594298
transitivity           1.000000          0.998739
clustering_coeff       0.998739          1.000000
```

Slika 11: Izgled matrice korelacije

Takođe, bilo bi interesantno proveriti koliko neki spoljašnji parametri, poput verovatnoće za mutaciju i verovatnoće pri izboru grana prilikom generisanja Erdoš-Renji grafa utiču na dobijene rezultate. Stoga je sprovedeno i ovo istraživanje. Odgovarajući rezultati nalaze se u tabelama 2 i 3. Kao što se može videti, prilikom smanjenje verovatnoće za grupisanje grana, dva problematična primera davala su uglavnom bolje rezultate, što bi se i moglo očekivati. Sa druge strane, povećanje verovatnoće za mutaciju nije doprinelo poboljšanju rezulta.

Tabela 2: Rezultati smanjenja verovatnoće za generisanje grana

Broj čvorova	**Veličina populacije**	**Fitness**
6	50	-0.005622222222222093
6	100	-0.005622222222222093
10	50	-0.1395151515151515
10	100	-0.012000000000000002
10	50	-0.008549953314659122
10	100	-0.005387076615423028
25	50	-0.6687809523809524
25	100	-0.48526616541353385
50	50	-0.420012338376997
50	100	-0.46940357205102456
75	50	-0.0125517293241569
75	100	-0.012004898528627055
100	50	-1.428400037947298
100	100	-2.6324747759596843

Tabela 3: Rezultati povećanja verovatnoće za mutaciju

Broj čvorova	Veličina populacije	Fitness
6	50	-0.005622222222222093
6	100	-0.06117777777777778
10	50	-0.4827692307692308
10	100	-0.21071794871794872
10	50	-0.036000000000000136
10	100	-0.006736282964629461
25	50	-1.4037338762346439
25	100	-1.283509261380821
50	50	-0.4757752270329916
50	100	-0.44939682755602156
75	50	-0.0099214931224301
75	100	-0.013052419602693637
100	50	-2.595627933058239
100	100	-2.6392986448062357

5 Zaključak

Kao što je u radu i izloženo, problem generisanja grafa koji ima željene osobine može biti veoma značajan. Optimizacioni algoritam koji je bio izložen je genetski algoritam koji se pokazao relativno dobro. Naravno, nije se pokazao dobar koliko i neki postojeći algoritmi koje su konstruisali svetski stručnjaci. Ipak, dodatnim istraživanjem je utvrđeno da postoje komponente koje su međusobno zavisne te stoga vrednost fitness funkcije zavisi od toga koliko smisla ove vrednosti imaju zajedno. Samim tim algoritam se našao u problemu i (nadamo se) odabrao najmanje loše rešenje žrtvovanjem najmanje bitnih uslova. Sa druge strane, zaključeno je i da spoljašnji faktori imaju malo uticaja na dobijene vrednosti. Naravno, uvek postoji prostor za napredak pa bi u budućnosti, na primer, mogala biti razmotrena ideja generisanja nasumičnih povezanih grafova jer se neki zanimljivi algoritmi ne mogu izvršiti ako postoji veći broj komponenti grafa.

Literatura

[1] Annu Lambora, Kunal Gupta, and Kriti Chopra. Genetic algorithm-a literature review. *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (Com-IT-Con)*, 2019. on-line at: https://sci-hub.se/10.1109/comitcon.2019.8862255.

[2] Vesna Marinković, Filip Marić, Strahinja Stanojević, and Sana Stojanović-Đurđević. *Konstrukcija i analiza algoritama*. Matematički fakultet, Univerzitet u Beogradu, 2019.

[3] M.E.J. Newman. Random graphs with clustering. *Physical review letters*, 2009.

[4] Optimization algorithms. Complexica, 2023. on-line at: https://www.complexica.com/narrow-ai-glossary/optimization-algorithms.

[5] Vili Podgorelec, Janez Brest, and Peter Kokol. Power of heterogeneous computing as a vehicle for implementing e3 medical decision support systems. *International Journal of Medical Informatics*, 2000.

[6] Ferozuddin Riaz and Khidir Ali. Applications of graph theory in computer science. *Third International Conference on Computational Intelligence, Communication Systems and Networks*, 2011.