SEMINARSKI RAD NA KURSU

Računarska inteligencija

MINIMUM INTERVAL GRAPH COMPLETION

Mina Kovandžić

127/2019

Filip Ogrenjac

275/2019

U Beogradu

2024

Sadržaj

1 Uvod

Tema ovog rada je problem minimalnog kompletiranja grafa do intervalnog grafa (Minimum Interval Graph Completion), koji se bavi transformacijom grafova u intervalne grafove uz minimalno dodavanje novih grana. Problem spada u klasu NP-teških problema, što ukazuje na njegovu kompleksnost i potrebu za efikasnim algoritamskim pristupima. U okviru rada, biće istraženi različiti algoritmi za optimizaciju kako bi se pronašla približna rešenja za ovaj problem.

Poglavlje 2 uvodi osnovne pojmove vezane za intervalne grafove, uključujući definiciju intervalnog grafa, meru problema i algoritamsku rešivost. Opisuje se metodologija provere da li je graf intervalni, uključujući upotrebu alata iz *SageMath* okruženja za verifikaciju osobina grafova.

U poglavlju 3, predstavljeni su različiti algoritmi za rešavanje problema minimalnog kompletiranja grafa do intervalnog grafa.

- **Algoritam grube sile** - generiše sve moguće grane koje mogu biti dodate grafu uz korišćenje kombinatorne metode za pronalaženje minimalnog skupa grana.
- **Simulirano kaljenje** - korišćenje različitih tehnika hlađenja za istraživanje prostora rešenja i optimizaciju ciljne funkcije.
- **Tehnika VNS (Variable Neighborhood Search)** - kombinacija upotrebe lokalne pretrage i promene oblasti pretrage za poboljšanje rezultata optimizacije.
- **Genetski algoritmi** - selekcija, crossover (rekombinacija) i mutacija kao alati za pronalaženje optimalnih rešenja.

Na kraju rada, analizirani su rezultati izvršavanja svakog od algoritama. Posebna pažnja je posvećena poređenju dobijenih vrednosti kako bi se utvrdila efikasnost različitih pristupa u rešavanju problema minimalnog kompletiranja grafa do intervalnog grafa. Kroz analizu performansi i kvaliteta rešenja, identifikovani su najefikasniji algoritmi i strategije za ovaj problem, što pruža korisne uvide za buduća istraživanja i primene u teoriji grafova i optimizaciji.

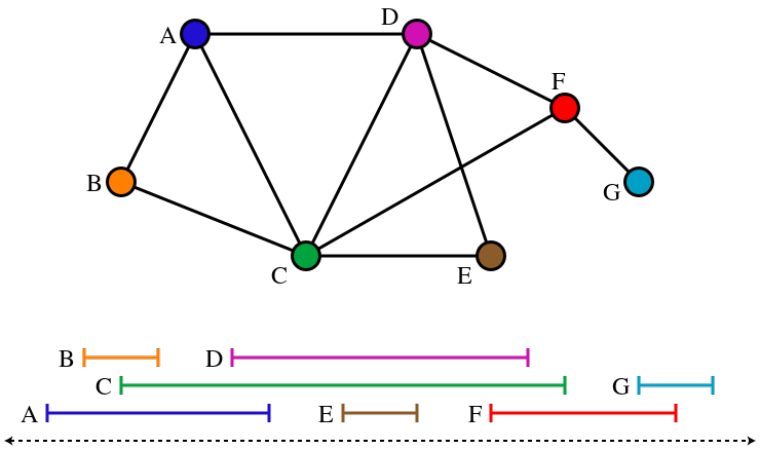
2 Opis problema

Minimum Interval Graph Completion

Problem minimalnog kompletiranja grafa do intervalnog grafa (Minimum Interval Graph Completion) bavi se grafovima i njihovim transformacijama u intervalne grafove. Formalno, neka je dat graf $G = (V, E)$, gde je V skup čvorova, a E skup grana. Cilj je pronaći intervalni graf $G' = (V, E')$ koji sadrži G kao podgraf, tj. gde je $E \subseteq E'$.

Intervalni graf

Intervalni graf je graf za koji postoji funkcija koja mapira čvorove na intervale na realnoj liniji, pri čemu su čvorovi povezani ivicom ako i samo ako se intervali koje predstavljaju preklapaju. Drugim rečima, intervalni graf je graf u kojem postoji funkcija $f : V \to \mathbb{R} \times \mathbb{R}$, gde je svaki čvor mapiran na interval $[a_i, b_i]$ na realnoj liniji. Dva čvora su povezana ivicom ako i samo ako se njihovi intervali $[a_i, b_i]$ i $[a_j, b_j]$ preklapaju.



Slika 2.1: Intervalni graf [2]

Mera

Za dati graf G, mera problema minimalnog kompletiranja grafa do intervalnog grafa, je kardinalnost skupa grana E' u intervalnom grafu G', tj. broj grana koje su dodate kako bi graf G postao intervalni graf. Matematički, mera se može izraziti kao $|E'|$, gde je E' skup grana u proširenom grafu G'.

Algoritamska Rešivost

Problem minimalnog kompletiranja grafa do intervalnog grafa je poznat po svojoj težini u pogledu računarske kompleksnosti. Međutim, može se približno rešiti u vremenu $O(\log |T|)$ [3]. Ovo znači da postoje algoritmi koji mogu pružiti približna rešenja problema u polinomijalnom vremenu, što je značajno za praktične primene i analize.

Primene

Značajna primena prisutna je u rešavanju problema iz različitih oblasti, uključujući optimizaciju rasporeda, analizu komunikacionih mreža, i uopšte u teoriji grafova gde se zahteva transformacija grafova u specijalizovane forme kao što su intervalni grafovi.

2.1 Provera intervalnosti grafa

Graf je intervalni ako i samo ako je *tetivni* (poznat i kao *chordal graph*) i ako je njegov komplement graf uporedivosti (*comparability graph*).[4]

Za proveru ovih svojstava korišćeni su alati iz *SageMath* okruženja[5].

Jedna od karakteristika tetivnih grafova je PEO (*Perfect Elimination Order*). PEO je specifično uređenje čvorova grafa, takvo da za svaki čvor, svi njegovi susedi koji dolaze kasnije u uređenju formiraju kliku. Klika (*clique*) je podskup čvorova u grafu u kom svaki par čvorova ima zajedničku granu, odnosno svi čvorovi u kliki su međusobno povezani. Koristimo metodu `lex_BFS()`, koja kao rezultat vraća PEO u slučaju da je ulazni graf tetivni. Njena složenost je $O(n^2)$, gde je n broj čvorova u grafu.

Da bismo proverili da li je graf tetivni, koristimo metodu `maximum_cardinality_search_M()` čija je složenost $O(n^2)$. Ova metoda kao rezultat vraća uređenu trojku (α, F, X). Parametar α predstavlja rezultujuće uređenje čvorova, a F je lista grana minimalne triangulacije grafa G prema uređenju α. Ako je početni graf zaista tetivni, on ima PEO, pa je lista triangulacije F prazna.

Na kraju, ukoliko je graf tetivni, proveravamo da li je njegov komplement graf uporedivosti, što takođe ima složenost $O(n^2)$.

Kombinovanjem ovih koraka, ukupna složenost funkcije `check_interval_graph` je:

$$O(n^2 + m)$$

gde je n broj čvorova, a m broj grana u grafu.

3 Implementacija algoritama

U ovom poglavlju biće predstavljeni i implementirani različiti algoritmi za rešavanje problema. S obzirom na složenost problema, biće korišćeni različiti pristupi kako bi se istražile efikasne metode za pronalaženje približnih rešenja.

Svaki od algoritama biće detaljno analiziran i upoređen u smislu efikasnosti i kvaliteta rešenja koje pružaju, kako bi se utvrdilo koji pristup najbolje odgovara za rešavanje problema minimalnog kompletiranja grafa do intervalnog grafa.

3.1 Algoritam grube sile

Algoritam grube sile se sastoji od nekoliko ključnih koraka. Generišu se sve moguće grane koje bi mogle biti dodate grafu. Korišćenjem kombinatornih metoda nalazi se minimalni skup grana koje je potrebno dodati kako bi graf postao intervalni. Algoritam proverava sve moguće kombinacije nedostajućih grana i testira da li dodavanje svake od njih čini graf intervalnim.

Ovaj pristup, iako jednostavan za implementaciju, može biti veoma računski zahtevan zbog eksponencijalnog broja kombinacija koje treba ispitati. Može biti neefikasan za veće grafove sa velikim brojem nedostajućih grana, iako pruža tačno rešenje. Zbog toga je algoritam grube sile testiran samo na manjim grafovima.

Složenost algoritma se može analizirati kroz sledeće korake:

- **Generisanje mogućih grana:** Prolazak kroz sve parove čvorova kako bi se generisao skup mogućih grana. Složenost ovog koraka je $O(n^2)$, gde je n broj čvorova u grafu.
- **Pronalaženje postojećih grana:** Prolazak kroz listu grana u grafu, što je složenosti $O(m)$, gde je m broj grana u grafu. Kreiranje skupa postojećih grana traje takođe $O(m)$ vremena.
- **Određivanje nedostajućih grana:** Razlika između dva skupa se može izračunati u $O(n^2)$ vremenu u najgorem slučaju.
- **Generisanje i testiranje kombinacija grana:** Broj mogućih kombinacija grana se može aproksimirati kao 2^k u najgorem slučaju, što dovodi do eksponencijalne složenosti. Kopiranje grafa i dodavanje grana traje $O(n + m + r)$ za svaku kombinaciju, a provera intervalnog grafa može biti polinomijalna u najgorem slučaju, recimo $O((n + m)^2)$.

Stoga, generisanje kombinacija nedostajućih grana dominira u računanju ukupne vremenske složenosti algoritma:

$$O\left(2^k \cdot (n + m + r) \cdot (n + m)^2\right)$$

gde je k broj nedostajućih grana.

3.2 Simulirano kaljenje

Inspirisan procesom zagrevanja i postepenog hlađenja materijala, ovaj algoritam omogućava istraživanje rešenja kroz kombinaciju lokalnih i globalnih pretraga. Na početku istraživanja korišćen je najosnovniji princip simuliranog kaljenja.

Osnovni Algoritam Simuliranog Kaljenja

Osnovni algoritam simuliranog kaljenja koristi slučajnu prirodu da bi istražio prostor rešenja i optimizovao ciljni funkciju.

Proces iteracije u algoritmu simuliranog kaljenja funkcioniše tako što se u svakoj iteraciji generiše novo rešenje promenom postojećeg rešenja i procenjuje njegov kvalitet. Ako je novo rešenje bolje, prihvata se; u suprotnom, postoji verovatnoća prihvatanja lošijeg rešenja zavisno od trenutne temperature. Temperatura se postepeno smanjuje kako bi se smanjila verovatnoća prihvatanja lošijih rešenja. Algoritam se završava kada se postigne broj iteracija ili temperatura padne ispod minimuma, vraćajući najbolje pronađeno rešenje.

Tehnike Hlađenja - Cooling Schedules

Da bi se poboljšala efikasnost algoritma simuliranog kaljenja, koriste se različite tehnike hlađenja. Ove tehnike definišu kako temperatura opada tokom vremena i time utiču na sposobnost algoritma da istražuje prostor rešenja. Upotrebom različitih metoda hlađenja, istražujemo kako različite strategije utiču na efikasnost algoritma i kvalitet konačnog rešenja. Ne postoji univerzalno najbolja tehnika hlađenja za sve probleme. Svaka tehnika hlađenja ima svoje prednosti i mane, a najbolji izbor zavisi od prirode problema, kao i od ciljeva optimizacije. Nekoliko tehnika hlađenja koje su primenjene u istraživanju [6]:

Multiplikativne sheme hlađenja

Multiplikativne monotone sheme hlađenja oslanjaju se samo na početnu temperaturu T_{max}, trenutni korak k i ručno podešenu konstantu α.

1. **Linearno multiplikativno hlađenje**:

$$T(k) = T_{\text{max}} - \alpha k$$

2. **Prirodni Logaritam Eksponencijalno Multiplikativno Hlađenje**:

$$T(k) = T_{\text{max}} \alpha^k$$

3. **Logaritamsko Multiplikativno Hlađenje**:

$$T(k) = \frac{T_{\text{max}}}{1 + \alpha \log(k+1)}$$

4. **Kvadratno Multiplikativno Hlađenje**:

$$T(k) = \frac{T_{\text{max}}}{1 + \alpha k^2}$$

Aditivni Monotoni Rasporedi Hlađenja

Aditivne monotone sheme hlađenja zavise od dva dodatna parametra: ukupnog broja koraka n i konačne temperature T_{min}.

1. **Linearno Aditivno Hlađenje**:

$$T(k) = T_{\text{min}} + (T_{\text{max}} - T_{\text{min}}) \left(\frac{n-k}{n} \right)$$

2. **Eksponencijalno Aditivno Hlađenje**:

$$T(k) = T_{\text{min}} + (T_{\text{max}} - T_{\text{min}}) \left(\frac{1}{1 + e^{\frac{2 \ln(T_{\text{max}} - T_{\text{min}})}{n} (k - \frac{1}{2} n)}} \right)$$

3. **Kvadratno Aditivno Hlađenje**:

$$T(k) = T_{\text{min}} + (T_{\text{max}} - T_{\text{min}}) \left(\frac{n-k}{n} \right)^2$$

Objašnjenje Parametara

- $T(k)$: Temperatura u koraku k.
- T_{max}: Početna (maksimalna) temperatura.
- T_{min}: Minimalna temperatura do koje se hladi.
- k: Trenutni korak.
- n: Ukupan broj koraka.
- α: Konstanta za prilagođavanje brzine hlađenja.

3.3 Tehnika VNS (Variable Neighborhood Search)

Tehnika **Variable Neighborhood Search (VNS)** je metaheuristički algoritam koji koristi princip sistematskog istraživanja različitih oblasti rešenja kako bi poboljšao rezultate optimizacije. VNS se oslanja na kombinaciju lokalne pretrage i promene oblasti pretrage kroz promenljive veličine. U našem istraživanju, VNS tehnika je implementirana kroz nekoliko ključnih koraka:

- **Shaking**: Generisanje nove varijacije trenutnog rešenja kroz nasumične promene. Grane grafa se dodaju ili uklanjaju iz rešenja na osnovu slučajnog izbora i unapred definisanog broja promena k. Ovaj korak pomaže u istraživanju različitih delova prostora rešenja.
- **Lokalna pretraga**: Primenjuje se lokalna pretraga sa algoritmom najboljeg poboljšanja. Svaka grana se nasumično dodaje ili uklanja iz trenutnog rešenja i procenjuje se novo rešenje. Ako se pronađe bolje rešenje, ono se usvaja, a pretraga se ponavlja dok ne dođe do daljih poboljšanja. Ovaj proces omogućava fino podešavanje rešenja unutar trenutne oblasti.

- Proces se ponavlja u okviru definisanog vremenskog ograničenja, pri čemu se istražuju različite veličine oblasti (od $k_{\min}$ do $k_{\max}$). Ako novo rešenje nudi poboljšanje u odnosu na trenutno najbolje rešenje ili se sa određenom verovatnoćom prihvata, koristi se kao novo trenutno rešenje.

Kombinovanjem ovih koraka, VNS tehnika omogućava efikasno istraživanje velikih prostora rešenja i poboljšanje kvaliteta rešenja tokom vremena.

3.4 Genetski algoritmi

Algoritam korišćen u radu oslanja se na ključne tehnike genetskog programiranja, uključujući selekciju, crossover (rekombinaciju) i mutaciju.

U našem istraživanju, svaka jedinka u populaciji predstavlja jedan mogući skup dodatnih grana koje se mogu dodati u originalni graf. Detaljno objašnjenje jedinke je sledeće:

- **Jedinka** je lista ili skup grana koje se dodaju u grafu kako bi se formirao novi graf.
- Svaka jedinka sadrži različit broj i kombinaciju grana koje se mogu dodati u originalni graf.
- Cilj je pronaći jedinku koja dodaje tačno onoliko grana koliko je potrebno da graf postane intervalni graf, pri čemu se teži minimizovanju broja potrebnih grana.

Kvalitet svake jedinke određen je fitness funkcijom. Ova funkcija meri kvalitet rešenja na osnovu toga da li dodate grane čine graf intervalnim. Ako graf postane intervalni, fitness vrednost je broj dodatih grana, gde manje grana označava bolji rezultat. U suprotnom, ako graf ne postane intervalni, rešenje se penalizuje beskonačnom vrednošću, čime se osigurava da se takvi rezultati ne koriste u sledećim generacijama algoritma.

Dakle, fitness funkcija favorizuje rešenja koja dodaju tačno onoliko grana koliko je potrebno da graf postane intervalni, dok rešenja koja ne uspevaju u tome bivaju kažnjena.

Turnirska selekcija je metoda odabira roditelja u genetskom algoritmu. Nasumično se izdvaja podskup jedinki iz populacije za učestvovanje na ”turniru”. Unutar ovog turnira, jedinke se takmiče na osnovu svoje fitness vrednosti. Biraju se dve jedinke sa najboljim fitnessom. Tokom ovog procesa vodi se računa da jedan roditelj ne bude izabran dva puta tokom iste selekcije, čime se osigurava raznovrsnost u procesu reprodukcije. Ovaj metod omogućava efikasnu selekciju boljih rešenja uz kontrolisanu količinu slučajnosti.

Ruletska selekcija predstavlja još jednu od metoda selekcije roditelja. Na osnovu vrednosti fitnessa, svaka jedinka u populaciji dobija verovatnoću koja određuje njenu šansu da bude izabrana kao roditelj. Roditelji se zatim biraju nasumično, ali sa težinom koja favorizuje rešenja sa boljim fitnessom, čime se omogućava da bolja rešenja imaju veću šansu da budu deo sledeće generacije.

Crossover ima ključnu ulogu u generisanju novih rešenja. Ova tehnika omogućava kombinovanje ”gena”različitih roditelja kako bi se stvorila nova rešenja, dok **mutacija** služi za uvođenje nasumičnih promena u okviru jedinke, čime se omogućava istraživanje novih mogućnosti koje možda nisu bile prisutne u prethodnim generacijama.

Zajedno, ove tehnike doprinose pronalaženju optimalnih rešenja za probleme sa grafovima.

Složenost genetskog algoritma zavisi od nekoliko ključnih faktora, uključujući veličinu populacije, broj generacija, složenost fitness funkcije i složenost operacija kao što su selekcija, crossover i mutacija. Analiza složenosti može se prikazati na sledeći način:

- **Inicijalizacija populacije:** Ako je veličina populacije P i broj grana koje se dodaju u grafu E, složenost inicijalizacije populacije je $O(P \cdot E)$, jer za svaku jedinku treba nasumično odabrati grane iz skupa svih mogućih grana.
- **Računanje fitness vrednosti:** Složenost provere da li je graf intervalni može zavisiti od specifičnog algoritma za proveru intervalnih grafova. U našem slučaju, neka je složenost $O(f)$, gde f zavisi od broja grana i čvorova u grafu. Ako se fitness funkcija računa za sve P jedinke, ukupna složenost je $O(P \cdot f)$.
- **Turnirska selekcija:** Za turnir veličine T i broj roditelja N, složenost selekcije može biti $O(N \cdot T)$, jer se za svakog roditelja mora izabrati podskup individua za turnir.
- **Ruletska selekcija:** Za populaciju veličine P i broj roditelja N, složenost je $O(P)$ za računanje sume fitnessa i verovatnoća. Zatim, za odabir roditelja, složenost je $O(P \cdot N)$, jer se za svakog roditelja verovatnoće koriste za nasumičan odabir iz populacije. Dakle, ukupna složenost ruletske selekcije je $O(P \cdot N)$.
- **Crossover:** Pretpostavljajući da se crossover operacije obavljaju za sve parove roditelja, složenost može biti $O(N \cdot C)$, gde je C složenost operacije crossovera za jedan par roditelja.
- **Mutacija:** Ako je broj mutacija u svakoj jedinki proporcionalan broju grana, a postoji P jedinki, složenost mutacije može biti $O(P \cdot E \cdot m)$, gde je m stopa mutacije.
- **Ukupna složenost:** Ako imamo G generacija i pretpostavimo da su svi ostali koraci (selekcija, crossover, mutacija) dominantni, ukupna složenost genetskog algoritma može se izraziti kao:
$$O(G \cdot P \cdot (f + T + C + E \cdot m))$$

Ova složenost uzima u obzir vreme potrebno za procenu fitness vrednosti, selekciju roditelja, izvođenje crossover operacija, i primenu mutacije tokom svih generacija.

4 Rezultati

Problematika pronalaženja minimalnog intervalnog grafa za proizvoljan graf predstavlja NP-težak problem, zbog čega je vremenski zahtevan, posebno za veće grafove. Međutim, na manjim primerima, algoritam može da pruži značajne rezultate u razumnom vremenskom roku. Skup svih grafova je podeljen u tri grupe: mali grafovi na koje može da se primeni i algoritam grube sile, nasumično generisani grafovi srednje veličine (oko 10 čvorova) i grafovi sa velikim brojem čvorova (između 20 i 50). Efikasnost različitih algoritama biće posmatrana u odnosu na vrstu grafova nad kojim su testirani.

4.1 Rezultati izvršavanja algoritama na manjim grafovima

Tabela 4.1 prikazuje prosečan broj dodatih grana za svaki od testiranih algoritama. Kolona V označava broj čvorova u grafu, a kolona E broj grana grafa. Zanimljivo je da su svi algoritmi, uključujući i genetski algoritam (GA) i algoritam grube sile (BF), ostvarili jednake rezultate kada je reč o broju dodatih grana u svim manjim grafovima. Ova uniformnost može sugerisati da su mali grafovi dovoljno jednostavni da svi algoritmi mogu uspešno pronaći najbolja rešenja bez značajnih razlika u performansama.

Tabela 4.2 prikazuje prosečno vreme izvršavanja svakog od algoritama za dati graf. Primećujemo da se najbrže izvršavaju BF i GA, koji imaju prednost u odnosu na ostale algoritme. Detaljan izveštaj o vremenu izvršavanja kao i o rezultatima svake kombinacije mogućih ulaznih parametara nalazi se u propratnoj dokumentaciji.

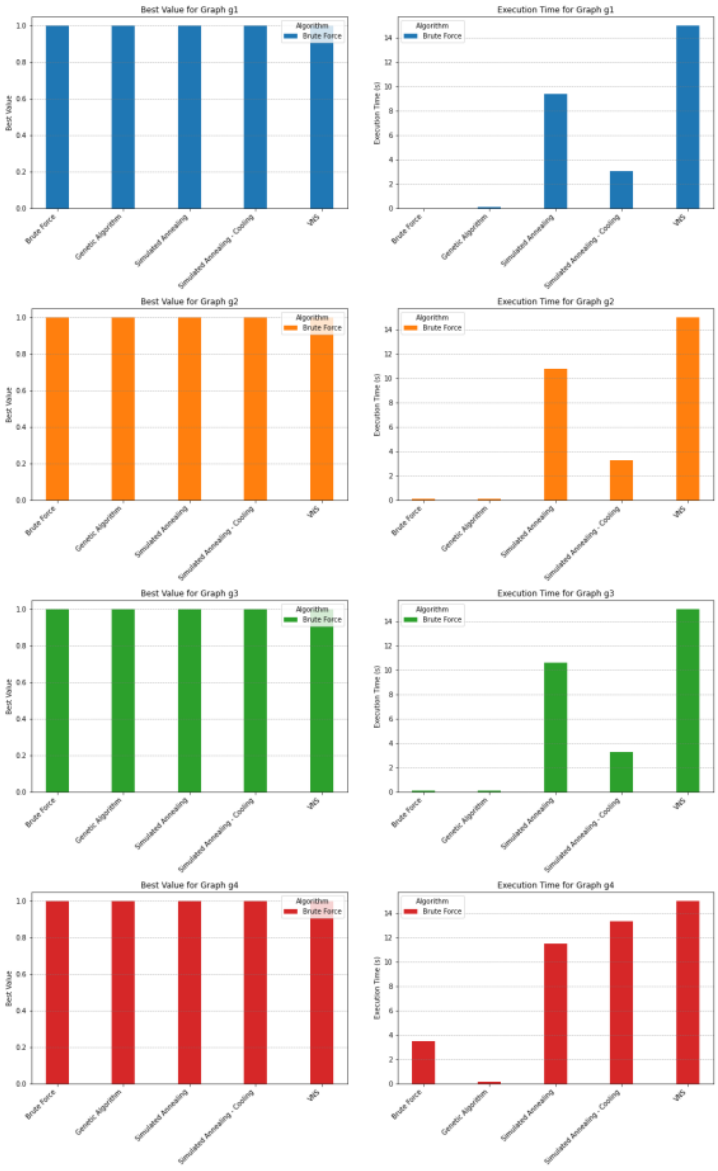
Vizualizacija rezultata izvršavanja svih algoritama na slici 4.1. u nastavku dodatno oslikava razlike u performansama. GA se ističe kao najbrži algoritam, dok VNS i simulirano kaljenje (SA, SAC) pokazuju duža vremena izvršavanja, što može biti indikativno za njihove složenije pretrage prostora rešenja.

Naziv	V	E	BF	SA	SAC	VNS	GA
G1	4	4	1	1	1	1	1
G2	5	4	1	1	2	1	1
G3	5	5	1	1	1	1	1
G4	7	6	1	1	1	1	1

Tabela 4.1: Prosečan broj dodatih grana

Naziv	V	E	BF	SA	SAC	VNS	GA
G1	4	4	0.0068s	9.3957s	8.63s	15s	0.1161s
G2	5	4	0.1064s	10.7620s	10.62s	15s	0.1262s
G3	5	5	0.1057s	10.5886s	9.72s	15s	0.1220s
G4	7	6	3.4807s	11.4943s	10.75s	15s	0.1309s

Tabela 4.2: Prosečno vreme izvršavanja



Slika 4.1: Rezultati izvršavanja svakog algoritma

4.2 Rezultati izvršavanja algoritama na nasumičnim grafovima

U tabelama 4.3 i 4.4 koje prikazuju rezultate na nasumičnim grafovima, primetne su dodatne razlike između algoritama. Algoritam grube sile nije davao zadovoljavajuće rezultate, što ukazuje na izazovnost ovih grafova i složenost rešenja, što je u skladu sa očekivanjima s obzirom na NP-težak karakter problema. Naredne tabele sadrže i oznaku „/“ za slučajeve kada je algoritam identifikovao početni graf kao intervalni, dok je „-“ označavao neuspeh u pronalaženju rešenja.

Zanimljivo je da su SA i SAC pokazali različite rezultate u broju dodatih grana, pri čemu je često SA postizao bolje rezultate. Efikasnost algoritama je u velikoj meri zavisila od karakteristika grafova, što može biti posledica različitih pristupa rešavanju problema. Na primer, za veće grafove (sa više čvorova i grana), primetno je povećanje vremena izvršavanja, posebno za SA i SAC, dok su GA i VNS i dalje zadržavali relativno kratka vremena izvršavanja.

V	E	SA	SAC	VNS	GA
6	7	3	4	3	3
7	12	2	3	2	2
7	15	1	1	1	1
8	15	2	4	2	2
8	16	2	4	2	2
8	19	/	/	/	/
9	21	5	8	5	5
9	21	6	7	5	5
9	21	6	9	6	6
9	23	4	6	3	3
10	20	-	17	-	7
10	26	-	12	7	7

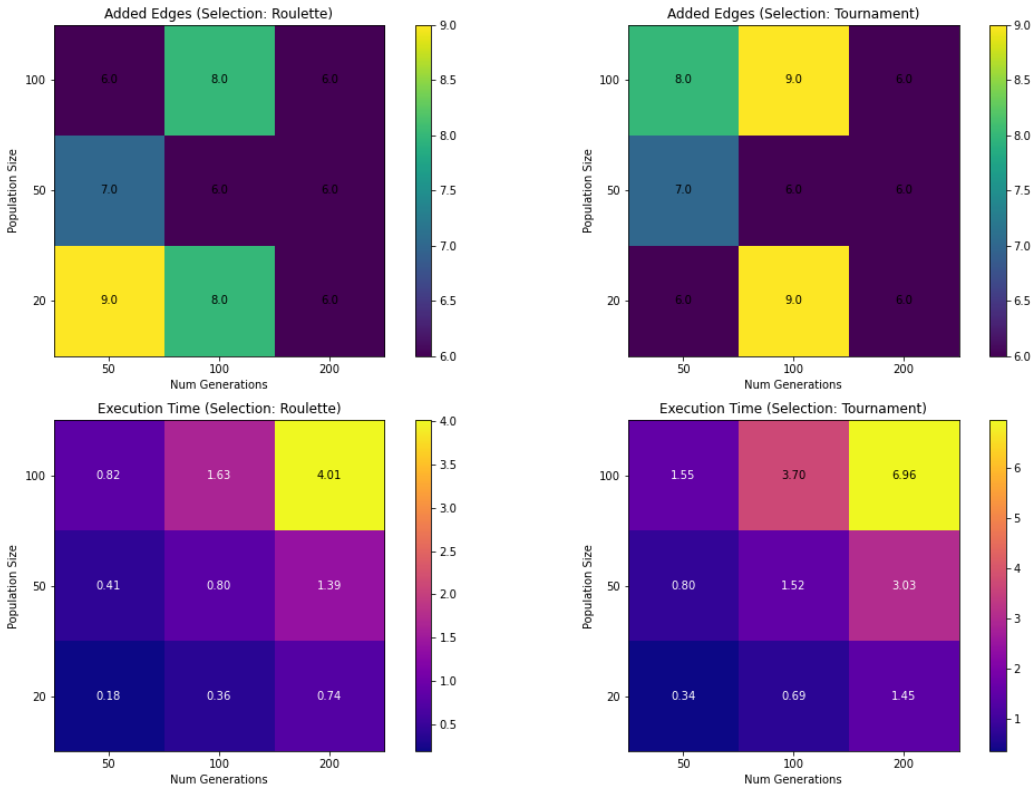
Tabela 4.3: Prosečan broj dodatih grana

V	E	SA	SAC	VNS	GA
6	7	10.9590s	10.833s	15s	0.1375s
7	12	11.2987s	10.9148s	15s	0.1517s
7	15	14.1091s	12.2046s	15s	0.1895s
8	15	12.9164s	11.8133s	15s	0.3194s
8	16	13.6216s	12.3723s	15s	0.1488s
8	19	/	/	/	/
9	21	15.2925s	13.9151s	15s	0.3078s
9	21	14.7493s	13.7071s	15s	0.2043s
9	21	16.6224s	13.6s	45s	0.3398s
9	23	20.4286s	12.9857s	15s	0.3768s
10	20	15.2074s	14.006s	-	0.6427s
10	26	16.7679s	14.7433s	15s	0.3886s

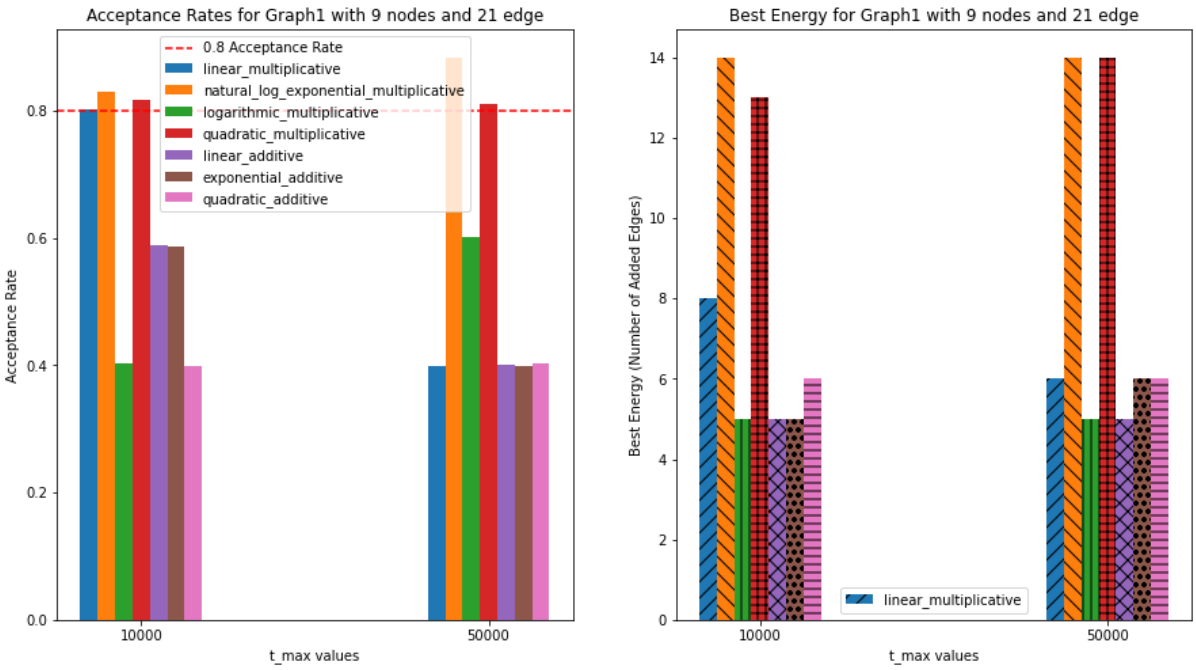
Tabela 4.4: Prosečno vreme izvršavanja

Dokumentacija sadrži pregledne grafike koji prikazuju rezultate svakog algoritma. Na osnovu podataka iz tabela i analize detaljnih izveštaja, zaključeno je da se genetski algoritam najviše ističe u pogledu balansa između minimalnog broja dodatih grana i vremena izvršavanja. Za ostale algoritme, efikasnost je u velikoj meri zavisila od karakteristika konkretnog ulaznog grafa, ali se među njima izdvojio i VNS po kriterijumu minimalnog broja dodatih grana. Primetno je da se najbolji rezultati najčešće postižu unutar najkraćeg vremenskog ograničenja od 15 sekundi, što je jedan od ključnih parametara ovog algoritma. Tehnike hlađenja korišćene u algoritmu simuliranog kaljenja dale su različite rezultate, s tim da se najčešće povoljni rezultati dobijaju pri višim početnim temperaturama (tmax=50000).

U nastavku slede grafici sa uporednim rezultatima različitih ulaznih parametara za genetski algoritam, kao i za algoritam simuliranog kaljenja sa dodatkom hlađenja.



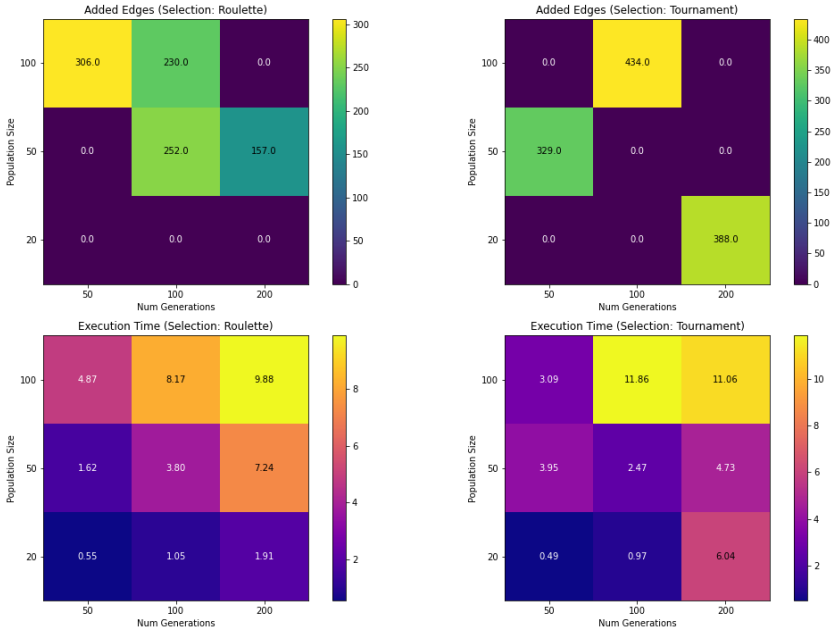
Slika 4.2: Rezultati genetskog algoritma



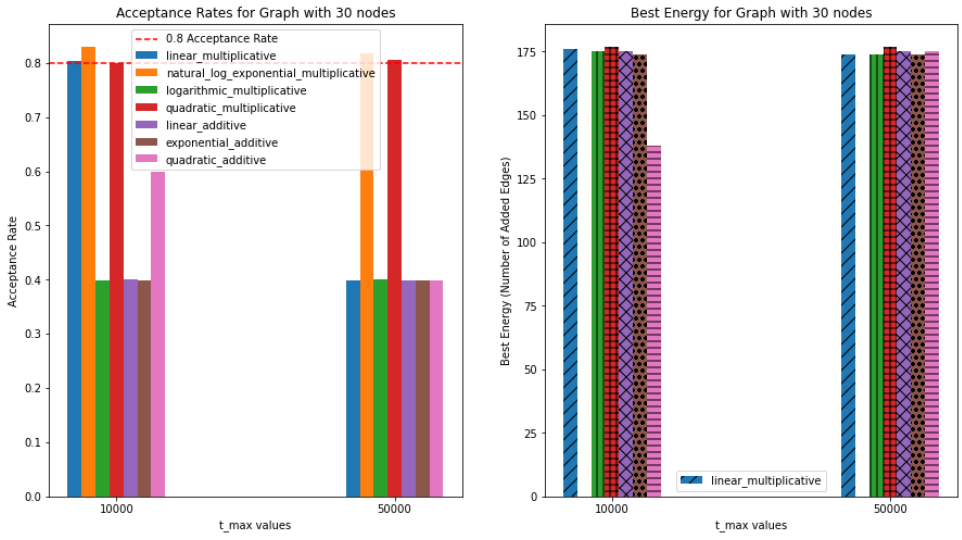
Slika 4.3: Rezultati algoritma simulirano kaljenje sa hlađenjem

4.3 Rezultati izvršavanja algoritama na velikim grafovima

Zbog velike složenosti samog problema, algoritmi su testirani za tri reprezentativna velika grafa sa po 20, 30 i 50 čvorova. Testiranjem različitih algoritama na ovim primerima, zapažamo prihvatljive rezultate korišćenjem genetskog algoritma i algoritma simuliranog kaljenja sa hlađenjem. Algoritmi VNS i standardno simulirano kaljenje nisu dali prihvatljive rezultate i pored modifikacije ulaznih parametara.



Slika 4.4: Rezultati genetskog algoritma - graf sa 30 čvorova



Slika 4.5: Rezultati algoritma simulirano kaljenje sa hlađenjem - graf sa 30 čvorova

Graph	Algorithm	Best Value	Execution Time
Graph with 20 nodes	Simulated Annealing	inf	40.284965
Graph with 20 nodes	Simulated Annealing - Cooling	43.0	54.108013
Graph with 20 nodes	VNS	inf	NaN
Graph with 20 nodes	Genetic Algorithm	43.0	7.323347

Slika 4.6: Rezultati različitih algoritama za graf sa 20 čvorova

Graph	Algorithm	Best Value	Execution Time
Graph with 30 nodes	Simulated Annealing	inf	73.846219
Graph with 30 nodes	Simulated Annealing - Cooling	138.0	109.957590
Graph with 30 nodes	VNS	inf	NaN
Graph with 30 nodes	Genetic Algorithm	157.0	7.235888

Slika 4.7: Rezultati različitih algoritama za graf sa 30 čvorova

Graph	Algorithm	Best Value	Execution Time
Graph with 50 nodes	Simulated Annealing	inf	215.980289
Graph with 50 nodes	Simulated Annealing - Cooling	403.0	299.023929
Graph with 50 nodes	VNS	inf	NaN
Graph with 50 nodes	Genetic Algorithm	878.0	36.100909

Slika 4.8: Rezultati različitih algoritama za graf sa 50 čvorova

Na osnovu preglednih rezultata izvršavanja, primećujemo drastičnu razliku u vremenu izvršavanja genetskog algoritma u odnosu na sve ostale algoritme. Ipak, algoritam simuliranog kaljenja daje bolje rezultate, iako je ukupno vreme ispitivanja svih kombinacija ulaznih parametara drastično veće u odnosu na genetski algoritam.

5 Zaključak

Problem pronalaženja minimalnog intervalnog grafa je vremenski zahtevan, naročito za veće grafove, zbog čega je potrebno primeniti različite algoritme i tehnike optimizacije. Na manjim grafovima, genetski algoritam i algoritam grube sile daju najbolje rezultate, kako po broju dodatih grana, tako i po vremenu izvršavanja. Međutim, na većim i nasumično generisanim grafovima, algoritmi poput VNS i simuliranog kaljenja pokazuju bolju efikasnost, iako rezultati zavise od specifičnih parametara poput vremena izvršavanja i početne temperature kod tehnika hlađenja. Genetski algoritam se izdvojio kao najefikasniji, postavljajući dobar balans između minimalnog broja dodatih grana i vremena izvršavanja.

Literatura

[1] https://www.csc.kth.se/~viggo/wwwcompendium/node50.html

[2] https://en.wikipedia.org/wiki/Interval_graph#/media/File:Interval_graph.svg

[3] Rao, S., and Richa, A. W. *New approximation techniques for some ordering problems*. Proc. 9th Ann. ACM-SIAM Symp. on Discrete Algorithms, ACM-SIAM, 1998, pp. 211–218.

[4] Michel Habib, Ross McConnell, Christophe Paul, and Laurent Viennot, *Lex-BFS and Partition Refinement, with Applications to Transitive Orientation, Interval Graph Recognition and Consecutive Ones Testing*, Discrete Mathematics, vol. 167, no. 1-3, pp. 87–102, 1997. Received November 1996; revised September 1997.

[5] https://doc.sagemath.org/

[6] https://nathanrooy.github.io/posts/2020-05-14/simulated-annealing-with-python/