Rešavanje optimizacionog problema nalaženja minimalnog Štajnerovog stabla

Računarska inteligencija

Ognjen Stamenković

64/2017

Matematički fakultet Univerziteta u Beogradu

Sadržaj

1. Uvod

Problem pronalaženja minimalnog Štajnerovog stabla u grafovima je poznat NP-težak optimizacioni problem koji je primenjivan na dizajn VLSI, dizajn komunikacionih mreža, biologiju sistema i menadžment podataka.

Neka je G(V, E, c) povezan neusmeren graf, gde je V skup čvorova, E skup grana i c funkcija koja slika svaku granu iz E u pozitivan ceo broj koji se naziva težina grane. Neka je T podskup od V koji se naziva skup terminala. Problem minimalnog Štajnerovog stabla ima za cilj da pronađe povezani podgraf G' ⊆ G koji sadrži sve čvorove iz skupa terminala za koji je suma težina grana iz G' minimalna. Podgraf G', koji je optimalno rešenje ovog problema naziva se minimalno Štajnerovo stablo grafa G za skup terminala T.

Zbog NP-težine ovog problema, vreme koje je potrebno da se nađe minimalnog Štajnerovo stablo se može povećavati eksponencijalno sa povećanjem veličine grafa. Međutim, puno instanci problema u realnom svetu uključuje velike grafove od više hiljada ili čak desetina hiljada čvorova. Prema tome, potrebno je razviti algoritme koji imaju dobre performanse za velike grafove (1).

U ovom projektu biće predstavljeni sledeći algoritmi za rešavanje ovog optimizacionog problema:

- Algoritam grube sile
- Genetski algoritam
- Diskretni algoritam za optimizaciju rojem čestica
- Algoritam simuliranog kaljenja

Dodatno, ovi algoritmi će biti napisani u programskom jeziku *Python* korišećem okruženja *Jupyter Notebook* i primenjeni na skup instanci problema pronalaženja minimalnog Štajnerovog stabla.

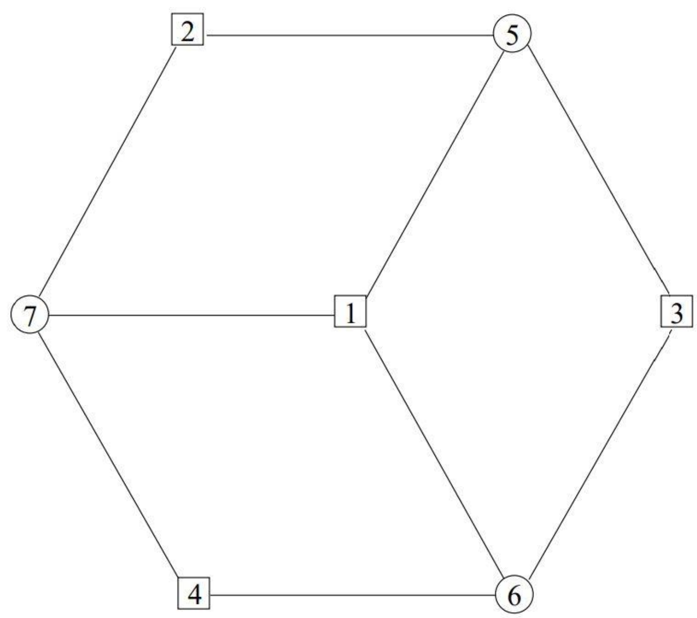
2. Skup podataka

Za potrebe testiranja optimizacionih algoritama korišćena je biblioteka *SteinLib Testdata Library* čiji je cilj prikupljanje instanci problema Štajnerovih stabla u grafovima i njegove varijante i informacija o njihovim izvorima, stepenu rešivosti i karakteristikama.

Biblioteka se može naći na stranici http://steinlib.zib.de/steinlib.php. Test podaci su čuvani u *.stp* formatu. Za potrebe projekta implementirano je učitavanje podataka ovog formata.

2.1. Opis formata podataka

U ovoj sekciji se nalazi kratak opis formata podataka koji se koristi SteinLib biblioteci. Za ilustraciju koristi se graf prikazan na slici 1 sa terminalnim čvorovima 1, 2, 3 i 4.



Slika 1: Primer formata podataka

U nastavku su linije koje opisuju ovaj primer u formatu SteinLib biblioteke.

```
33D32945 STP File, STP Format Version 1.0
```

```
SECTION  Comment
Name     “Odd wheel”
Creator  “T. Koch, A. Martin and S. Voss”
Remark   “Example used to describe the STP data format”
END

SECTION  Graph
Nodes 7
Edges 9
E 1 5 1
E 1 6 1
E 1 7 1
E 2 5 1
E 2 7 1
E 3 5 1
E 3 6 1
E 4 6 1
E 4 7 1
END

SECTION Terminals
Terminals 4
T 1
T 2
```

```
T 3
T 4
END

SECTION Coordinates
DD 1 80 50
DD 2 55 5
DD 3 130 50
DD 4 55 95
DD 5 105 5
DD 6 105 95
DD 7 30 50
END
```

Datoteka je podeljena u sekcije. Sekcija počinje ključnom rečju SECTION koju prati naziv skecije. Sekcija se završava ključnom rečju END. Svaki red u sekciji počinje ključnom rečju koja ukazuje na tip reda.

3. Algoritmi

3.1. Algoritam grube sile

Algoritam grube sile implementiran je tako što se generišu sve kombinacije izbora ne-terminalnih čvorova. Za svaku kombinaciju pravi se graf koji uključuje čvorove iz kombinacije i terminalne čvorove. Za ovaj graf se proveri da li predstavlja Štajnerovo stablo. U slučaju da predstavlja, njegova vrednost se pamti ako je ona manja od minimalne vrednosti do tog trenutka.

3.2. Genetski algoritam

Genetski algoritmi implementiraju se kao računarska simulacija u kojoj populacija apstraktno reprezentovanih jedinki koje su kandidati za rešenje problema, treba da se približava boljim rešenjima. Reprezentacija jedinke naziva se hromozomom. Cilj je naći vrednost za koju zadata funkcija cilja dostiže svoj ekstremum. Početna rešenja, tj. jedinke su obično reprezentovane nizom nula i jedinica. Postupak se odvija kroz generacije. Funkcija koja pridružuje vrednost jedinkama se naziva funkcija prilagođenosti. Iz jedne generacije, se na osnovi vrednosti funkcije prilagođenosti, kroz proces selekcije biraju jedinke koje će biti iskorišćene za stvaranje novih jedinki. Kvalitetnije se biraju sa većom verovatnoćom. Operatorom ukrštanja se od izabranih jedinki dobijaju nove jedinke, a operatorom mutacije dolazi do modifikacije jedinke.

U predloženom genetskom algoritmu, jedinka je predstavljena nizom nula i jedinica dužine $|V| - |T|$. Jedinka prestavlja su koji čvorovi uključeni u formiranje Štajnerovog stabla iz skupa ne-terminalnih čvorova.

Početna populacija jedinki je generisana slučajno.

Evaluacija jedinke populacije vrši se rekonstrukcijom matrice povezanosti grafa koja uključuje samo odabrane čvorove iz jedinke i terminalne čvorove. Za ovaj novodobijeni graf se proverava da li je povezan, kako bi bilo moguće da se dobije stablo. Funkcija prilagođenosti jedinke predstavlja vrednost minimalnog razapinjućeg stabla kreiranog od novodobijenog grafa.

Za ukrštanje koristi se jednopoziciono ukrštanje sa slučajno izabranom pozicijom. Za svaku jedinku slučajno se bira par za ukrštanje iz ostatka populacije.

Mutacija se vrši sa verovatnoćom 0.2 nad svakim članom populacije. Mutacija predstavlja inverziju slučajno odabranog gena jedinke.

Selekcija jedinke se vrši sa verovatnoćom koja je proporcijalna prilagođenosti jedinke. U svakoj iteraciji čuva se trenutno najbolja jedinka za sledeću iteraciju.

Algoritam 1: Predloženi genetski algoritam za rešavanje problema pronalaženja minimalnog Štajnerovog stabla

```
Input: Graf G(V, E, c), terminalni skup T, maksimalni broj iteracija M
Output: Štajnerovo stablo G’ ⊆ G
Inicijalizovati početnu populaciju
Evaluirati početnu populaciju i upamtiti najbolje rešenje
for i = 0 to M do
    Izvršiti ukrštanje nad tekućom populacijom kako bi se dobila nova populacija
    Izvršiti mutaciju nad novom populacijom
    Evaluirati novu populaciju
    Ažurirati najbolje rešenje ako je dobijeno bolje
    Izvršiti selekciju jedinki iz nove populacije
    Zameniti najlošije rešenje populacije sa sačuvanim najboljim
    Nova populacija postaje tekuća populacija
end
```

3.3. Diskretni algoritam za optimizaciju rojem čestica

Algoritam za optimizaciju rojem čestica je računarska metoda koja optimizuje problem iterativno, pokušavajući da poboljša kandidate za rešenje vodeći se merom kvaliteta. Algoritam optimizuje problem pomoću populacije kandidata za rešenje koji se nazivaju čestice. One se kreću u prostoru rešenja pomoću jednostavne matematičke formule koja zavisi od njihovog najboljeg položaja i globalno najboljeg položaja. Kako se otkrivaju bolje pozicije čestica, biće korišćene za usmeravanje njihove kretnje.

U predloženom diskretnom algoritmu za optimizaciju rojem čestica, jednu česticu predstavlja niz nula i jedinica koje označavaju da li je ne-terminalni čvor uključen u formiranje Štajnerovog stabla (kao kod genetskog algoritma).

Generisanje inicijalne populacije vrši se slučajno. Međutim, postavljeno je da bude duplo verovatnije da proizvoljan čvor bude uključen kako bi se povećala šansa za generisanje povezanog stabla.

Početna brzina čestice je niz float vrednosti jednake dužine kao veličina čestice. Svaka vrednost niza brzine proporcijalna je verovatnoći da će odgovarajući gen čestice promeniti vrednost.

Ažuriranje vrednosti brzine vrši se po formuli:

$$V_{ij}(t+1) = \begin{cases} \omega_t \times V_{ij}(t) + c_1 r_1 + c_2 r_2 & \text{if } gB_j = pB_{ij} = 1 \\ \omega_t \times V_{ij}(t) - c_1 r_1 - c_2 r_2 & \text{if } gB_j = pB_{ij} = 0 \\ \omega_t \times V_{ij}(t) & \text{otherwise} \end{cases}$$

Gde su c_1 i c_2 konstante, r_1 i r_2 slučajne vrednosti iz opsega (0, 1), a ω_t faktor inercije dobijen formulom:

$$w_t = w_{t=max} - \frac{w_{t=max} - w_{t=min}}{t = max} \times t$$

Ažuriranje pozicije vrši se po formuli:

$$X_{ij}(t+1) = \begin{cases} 1 & \text{if } r_{ij} < sig(V_{ij}(t+1)) \\ 0 & \text{otherwise.} \end{cases}$$

Evaluacija čestice se radi na isti način kao kod genetskog algoritma, koristeći minimalno razapinjuće stablo.

Algoritam 2: Predloženi diskretni algoritam za optimizaciju rojem čestica

```
Input: Graf G(V, E, c), terminalni skup T, maksimalni broj iteracija M
Output: Štajnerovo stablo G’ ⊆ G
Inicijalizovati početnu populaciju
Evaluirati početnu populaciju i upamtiti najbolje rešenje
Upamtiti globalno najbolje rešenje
for i = 0 to M do
    Izračunati faktor inercije ω_t
    Za svaku česticu populacije ažurirati brzinu
    Za svaku česticu populacije ažurirati poziciju
    Evaluirati pozicije čestica
    Ažurirati najbolju poziciju svake čestice
    Ažurirati globalno najbolju poziciju rešenja
end
```

3.4. Algoritam simuliranog kaljenja

Algoritam simuliranog kaljenja iterativno poredi funkcije prilagođenosti tekućeg rešenja i susednog rešenja. Ako susedno rešenje daje bolji rezultat, ono postaje tekuće rešenje. Inače, ako susedno rešenje ne daje bolji rezultat, ono može postati tekuće rešenje sa verovatnoćom koja je zavisi od razlike vrednosti funkcije prilagođenosti ova dva rešenja i od vrednosti parametra temperature sistema. Ovaj parametar služi kao imitacija temperature u kaljenju.

U predloženom algoritmu simuliranog kaljenja rešenje sistema predstavlja podstablo koje sadrži sve terminalne čvorove. Inicijalno rešenje dobija se određivanjem minimalnog razapinjućeg stabla početnog grafa, i zatim brisanjem jednostepenih ne-terminalnog čvorova.

Susedno rešenje se generiše od trenutnog rešenja, brisanjem proizvoljne grane iz stabla. Zatim se slučajno bira po jedan čvor iz dve novonastale komponente povezanosti. Korišćenjem svih grana iz polaznog grafa, pronalazi se najkraći put između odabranih čvorova, i grane ovog puta se ubacuju u graf susednog rešenja. Treba osigurati da su svi terminali dobijenog rešenja u istoj komponenti povezanosti.

Za vrednost funkcije prilagođenosti uzima se suma težina grana stabla rešenja.

Parametar temperature predstavlja razliku maksimalnog broja iteracija i vrednost trenutnog broja iteracije.

Verovatnoća prihvatanja susednog rešenja, u slučaju da je ono lošije računa se sledećom formulom:

$$p = e^{-\frac{|f_{new} - f_{curr}|}{T_i}}$$

Algoritam 3: Predloženi algoritam simuliranog kaljenja

```
Input: Graf G(V, E, c), terminalni skup T, maksimalni broj iteracija M

Output: Štajnerovo stablo G' ⊆ G

1:   Naći minimalno razapinjuće stablo od G
2:   Izbaciti jednostepene ne-terminalne čvorove
3:   Evaluirati početno rešenje
4:   for i = 0 to M do
5:       Odrediti susedno rešenje tekućem
6:       Evaluirati susedno rešenje
7:       Ako je kvalitetnije susedno rešenje, ono postaje
         tekuće, inače postaje tekuće sa verovatnoćom p
8:   end
```

4. Rezultati i diskusija

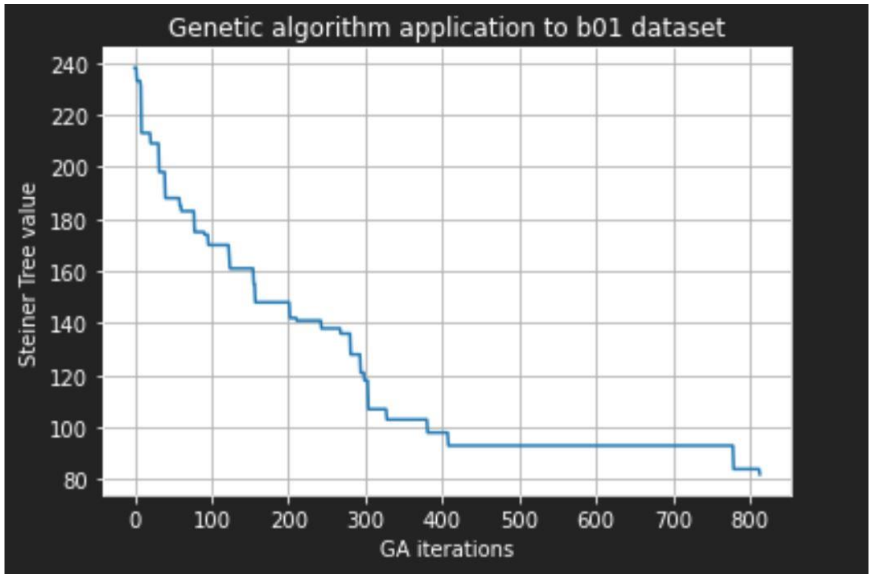
4.1. Algoritam grube sile

Algoritam grube sile daje korektno rešenje prilikom testiranja na grafu od 4 čvora. Kod testiranja na b01.stp primeru, algoritam ne nalazi optimalno rešenje. Algoritam se ne završava u razumnom vremenu. Iako je graf iz primera b01 jedan od grafova za manjim brojem čvorova iz skupa podataka B, broj mogućih rešenja za Štajnerovo stablo je reda veličine oko 2^{41}, što čini algoritam grube sile neprimenjivim na primere iz skupa podataka.

4.2. Genetski algoritam

Tokom razvoja genetskog algoritma, testiranje koda i debagovanje je bilo rađeno na primeru b01.stp iz B skupa podataka SteinLib biblioteke. U svim testovima primene algoritma na instance Štajnerovog problema, maksimalan broj iteracija bio je ograničen na 10000.

Na primeru b01.stp, genetski algoritam je konvergirao ka optimalnom rešenju u 814 iteracija. Na slici 2 prikazan je grafik zavisnosti tekuće najbolje vrednosti minimalnog Štajnerovog stabla od iteracije genetskog algoritma.



Slika 2: Grafik zavisnosti tekuće najbolje vrednosti minimalnog Štajnerovog stabla od iteracije genetskog algoritma

Na grafiku se vidi da vrednost minimalnog Štajnerovog stabla ravnomerno opada kroz iteracije, upadajući u lokalni minimum oko iteracije 400, pre nego da stigne do optimalne vrednosti.

U nastavku, genetski algoritam je testiran na prvih deset instanci problema iz skupa podataka B SteinLib biblioteke. U tabeli 1 prikazane su informacije o ovim instancama problema kao i performanse genetskog algoritma pri njegovoj primeni.

Graph name	Nodes	Edges	Terminals	Goal	Solution	Iterations	Error(%)
b01	50	63	9	82	82	1789	0
b02	50	63	13	83	91	10000	8.79
b03	50	63	25	138	138	66	0
b04	50	100	9	59	64	10000	7.81
b05	50	100	13	61	61	9172	0
b06	50	100	25	122	122	4191	0
b07	75	94	13	111	111	2557	0
b08	75	94	19	104	110	10000	5.45
b09	75	94	38	220	220	735	0
b10	75	150	13	86	87	10000	1.15

Tabela 1: Informacije o grafovima i performanse primene genetskog algoritma

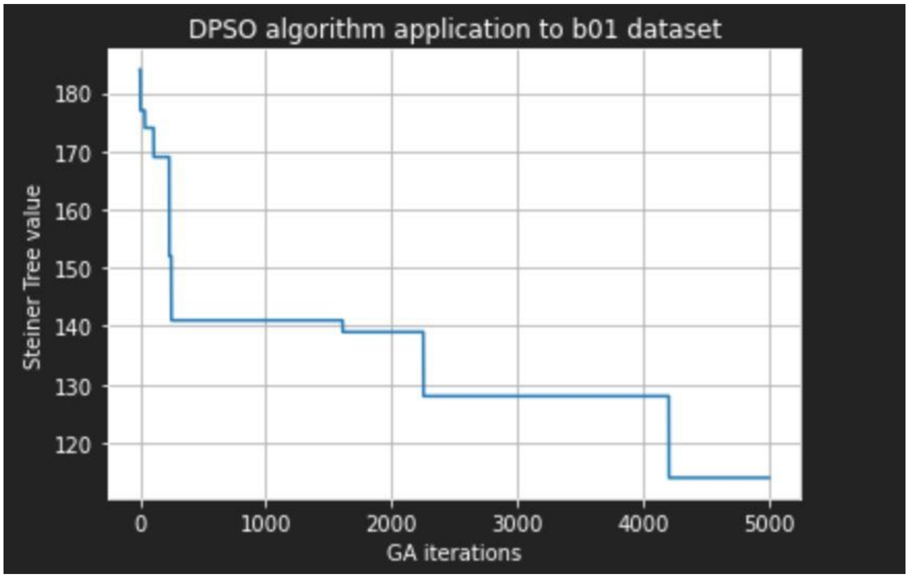
Može se primetiti da genetski algoritam dostiže optimalno rešenje u većini slučaja, dok u ostalim slučajevima, greška ne biva veća od 8.79%. Ovi rezultati su uporedivi sa rezultatima genetskog algoritma primenjenog u [3].

Treba napomenuti da ponovljeno testiranje ne donosi identične rezultate, ali razlike nisu drastične.

4.3. Diskretni algoritam za optimizaciju rojem čestica

Diskretni algoritam za optimizaciju rojem šestica je takođe razvijan na primeru b01 SteinLib biblioteke. Maksimalan broj iteracija bio je ograničen na 5000.

Na slici 3 prikazan je grafik zavisnosti tekuće najbolje vrednosti minimalnog Štajnerovog stabla od iteracije ovog algoritma.



Slika 3: Grafik zavisnosti tekuće najbolje vrednosti minimalnog Štajnerovog stabla od iteracije agloritma sa rojem čestica

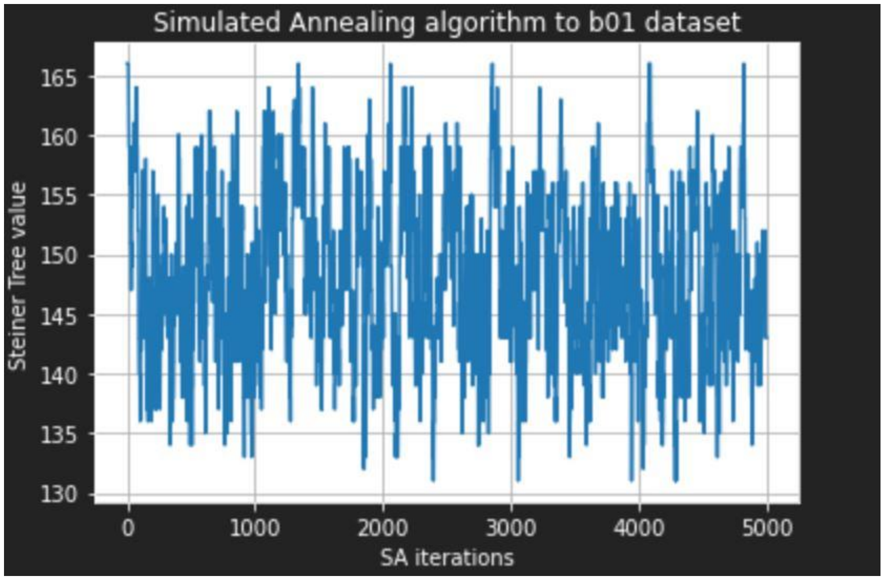
Može se primetiti da vrednost najboljeg rešenja opada tokom toka algoritma, ali on nije konvergirao u optimalno rešenje posle maksimalnog broja iteracija.

Pojedinačna iteracija ovog algoritma se previše dugo izvršava, što čini algoritam praktično neprimenjivim. Ovo se dešava zbog provere da li novodobijena vrednost pozicije čestice može predstavljati rešenje problema. Verovatno je da se često dobija čestica koja ne može predstavljati rešenje i tada se dosta vremena ulaže u pronalaženje nove pozicije koja bi mogla da se iskoristi u daljem algoritmu.

4.4. Algoritam simuliranog kaljenja

Algoritam simuliranog kaljenja je takođe bio razvijan na primeru b01 SteinLib biblioteke. Maksimalan broj iteracija bio je ograničen na 5000.

Na slici 4 prikazan je grafik zavisnosti tekuće najbolje vrednosti minimalnog Štajnerovog stabla od iteracije na primeru b01.



Slika 4: Grafik zavisnosti tekuće najbolje vrednosti minimalnog Štajnerovog stabla od iteracije genetskog algoritma

Algoritam ne konvergira ka optimalnom rešenju posle 5000 iteracija. Ovo je verovatno izazvano istražujućom prirodom algoritma simuliranog kaljenja. Takođe, problem se potencijalno nalazi u načinu generisanje novog susednog rešenja. Pojavljivao se problem gde se u novogenerisanom susednom rešenju svi terminali ne nalaze u istoj komponenti povezanosti. Tada bi ovo rešenje moralo da bude odbačeno zbog stvaranja većih problema u daljem toku algoritma. Ovaj problem u teoriji ne bi trebalo da se pojavljuje, ali greška koja ga izaziva nije prodađena.

U nastavku je primenjen algoritam i na drugi primer iz B skupa podataka SteinLib biblioteke. U tabeli 2 su prikazane informacije o instancama b01 i b02 i performanse primene algoritma simuliranog kaljenja na iste.

Graph name	Nodes	Edges	Terminals	Goal	Solution	Iterations	Error(%)
b01	50	63	9	82	143	4999	42.66
b02	50	63	13	83	147	4999	43.54

Tabela 2: Informacije o grafovima i performanse primene algoritma simuliranog kaljenja

Greške pri primeni algoritma prelaze 40%, što je previsoko. Ovaj algoritam nije primenjiv.

5. Zaključak

U projektu su predstavljena četiri algoritma za rešavanje optimizacionog problema nalaženja minimalnog Štajnerovog stabla. Algoritam grube sile nije primenljiv na instance iz skupa za testiranje, zbog njihove previsoke zahtevnosti. Genetski algoritam je prikazao zadovoljavajuće performanse tokom tesitranja. Na instancama na kojima je primenjen, genetski algoritam je u većini slučajeva pronalazio optimalno rešenje, u ostalim slučajevima, pronađeno rešenje se nije razlikovalo od optimalnog za više od 8.79%. Diskretni algoritam za optimizaciju rojem čestica nije prikazao dobre performanse. On je previše sporo napredovao u nalaženju boljih rešenja, i pojedinačna iteracija ovog algoritma je bila previše spora da bi on bio primenjiv. Algoritam simuliranog kaljenja, takođe, nije prikazao dobre rezultate. On uopšte nije konvergirao ka boljim rešenjima. Ovo se najverovatnije dešavalo zbog nekih nepronađenih grešaka.

Zbog ograničenja u resursima, algoritmi nisu dovoljno široko istestirani. Poboljšanje projekta se može naći u dodatnom tesitranju genetskog algoritma, optimizaciji algoritma sa rojem čestica i rešavanjem navedenih problema algoritma simuliranog kaljenja.

6. Literatura

[1] A. Kartelj, *Računarska inteligencija.*

[2] P. Janičić i M. Nikolić, *Veštačka inteligencija.*

[3] Y. Sun, *Solving the Steiner Tree Problem in Graphs using Physarum-inspired Algorithms.*

[4] T. Koch, A. Martin, S. Voss, *SteinLib: An Updated Library on Steiner Tree Problems in Graphs.*

[5] M. Clerc, *Discrete Particle Swarm Optimization, illustrated by the Travelling Salesman Problem.*

[6] K. A. Dowsland, *Hill-Climbing, Simulated Annealing and the Steiner Problem in Graphs.*

[7] A. H. El-Maleh, A. T. Sheikh, S. M. Sait, *Binary particle swarm optimization based state assignment for area minimization of sequential circuits.*