Istraživanje i optimizacija minimalnog nezavisnog dominantnog skupa u grafovima

Sara Kalinić 387/2021

September 27, 2024

1 Uvod

Minimalni nezavisni dominantni skup čvorova je važan problem u teoriji grafova i računarstvu. Ovaj problem je NP-težak, što znači da ne postoji poznat polinomni algoritam koji može rešiti sve instance ovog problema. U ovom radu istražujemo različite heurističke tehnike za optimizaciju rešenja, uključujući varijantna okruženja pretraživanja (VNS), genetske algoritme i optimizaciju mravima.

2 Teorijski Okvir

2.1 Definicija

Nezavisni dominantni skup čvorova u grafu $G = (V, E)$ je podskup čvorova $D \subseteq V$ tako da su svi čvorovi u V ili u D ili su susedi nekog čvora iz D.

3 Implementacija Rešenja

3.1 Grubom Silom

Grubom silom je metoda koja se koristi za pronalaženje minimalnog nezavisnog dominantnog skupa čvorova u grafu pretraživanjem svih mogućih podskupova čvorova. U ovoj sekciji objašnjava se implementacija ove metode koristeći Python i biblioteku NetworkX.

3.1.1 Opis Koda

Za implementaciju grube sile razvijen je Python kod koji se oslanja na kombinacije čvorova u grafu. Ključni delovi koda uključuju:

- **Definisanje funkcija:**
  - `is_independent_set(graph, subset)`: Ova funkcija proverava da li je dati `subset` nezavisni skup. Prolazi kroz sve moguće kombinacije čvorova u `subset` i proverava da li postoji ivica izmeu bilo koja dva čvora. Ako postoji ivica, skup nije nezavistan.

- `is_dominating_set(graph, subset)`: Ova funkcija proverava da li `subset` dominira ceo graf. Ona prikuplja sve čvorove iz `subset` i njihove susede, i proverava da li je ukupan broj čvorova u `dominating_nodes` jednak broju čvorova u grafu.
- `brute_force_mids(graph)`: Ova funkcija implementira glavnu logiku grube sile. Ona generiše sve moguće podskupove čvorova koristeći funkciju `itertools.combinatior` i proverava svaki podskup da li ispunjava uslove za nezavisni i dominantni skup. Ako se nae manji skup koji zadovoljava uslove, on se čuva kao najbolji rezultat.

- **Vizualizacija grafa:**
  - Funkcija `draw_graph(graph, min_set)` koristi biblioteku Matplotlib za vizualizaciju grafa. Čvorovi koji čine minimalni nezavisni dominantni skup se prikazuju u crvenoj boji, dok su ostali čvorovi u svetloplavoj boji. Ova vizualizacija pomaže u boljem razumevanju strukture grafa i pozicije odabranih čvorova.

3.1.2 Rezultati

Na grafu koji se učitava iz datoteke `tests/test_30_0.3.in`, poziv funkcije `brute_force_mids(G)` vraća minimalni nezavisni dominantni skup. Rezultat se štampa u konzoli i vizualizuje pomoću funkcije `draw_graph`.

U sledećoj tabeli prikazani su rezultati dobijeni primenom grube sile na grafovima sa različitim brojem čvorova i verovatnoćom veze.

Test Name	Number of nodes	Iterations	Time (s)
test_10_0.3.in	2	1023	0.0010
test_10_0.5.in	3	1023	0.0010
test_10_0.7.in	2	1023	0.0010
test_15_0.3.in	4	32767	0.0380
test_15_0.5.in	2	32767	0.0260
test_20_0.3.in	5	1048575	1.2748
test_20_0.5.in	3	1048575	0.6900
test_25_0.3.in	4	33554431	27.9059
test_25_0.5.in	3	33554431	26.8291
test_25_0.7.in	2	33554431	23.3268
test_30_0.3.in	0	0	0.0
test_30_0.5.in	0	0	0.0
test_35_0.3.in	0	0	0.0
test_35_0.5.in	0	0	0.0
test_40_0.3.in	0	0	0.0
test_40_0.5.in	0	0	0.0
test_40_0.7.in	0	0	0.0

Table 1: Rezultati testiranja za minimalni nezavisni dominantni skup Grubom silom

Ova metoda, iako jednostavna i intuitivna, može biti vrlo spora za velike grafove zbog eksponencijalne kompleksnosti. Ipak, koristi se kao osnova za poreenje s drugim heurističkim metodama u daljoj analizi.

3.2 Variable Neighborhood Search (VNS)

Variable Neighborhood Search (VNS) je heuristička metoda optimizacije koja istražuje više susednih prostora rešenja kako bi pronašla bolje rešenje. U ovoj sekciji objašnjavamo implementaciju VNS algoritma za pronalaženje minimalnog nezavisnog dominantnog skupa čvorova u grafu.

3.2.1 Opis Koda

Implementacija VNS algoritma obuhvata sledeće ključne funkcije:

- **Inicijalizacija:**
  - Funkcija `initialize(graph)` generiše nasumično rešenje, gde svaki čvor u grafu ima 30% verovatnoće da bude uključen u rešenje. Ovo se postiže korišćenjem `random.random()` funkcije.
- **Fitness funkcija:**
  - Funkcija `calc_fitness(solution, graph)` izračunava fitness vrednost datog rešenja. Ako je rešenje nezavisni i dominantni skup, vraća se recipročna vrednost broja čvorova u skupu. U suprotnom, vraća se negativna beskonačnost.
- **Lokalna pretraga:**
  - Funkcija `local_search_invert_best_improvement(solution, value, graph)` primenjuje lokalnu pretragu da poboljša trenutno rešenje. Ona invertuje svaki bit u rešenju i proverava da li dobijeno novo rešenje ima bolju fitness vrednost. Ako se pronae bolje rešenje, ono se čuva i proces se nastavlja.
- **Shaking:**
  - Funkcija `shaking(solution, k)` kreira novo rešenje tako što nasumično menja `k` bitova u trenutnom rešenju. Ova funkcija pomaže u istraživanju novih područja rešenja.
- **VNS glavni algoritam:**
  - Funkcija `vns(graph, vns_params)` implementira VNS algoritam. Za svaku vrednost `k` (od `k_min` do `k_max`), generiše se novo rešenje koristeći funkciju `shaking`, koje se zatim poboljšava lokalnom pretragom. Ako novo rešenje ima bolju fitness vrednost, ono postaje novo trenutno rešenje.

3.2.2 Rezultati

Funkcija `vns_main(graph)` poziva glavni VNS algoritam sa unapred definisanim parametrima. Na kraju, najbolji rezultat se prikazuje u konzoli, a graf se vizualizuje koristeći funkciju `draw_graph`.

```
Najbolji fitnes: (rezultati)
Minimalni Nezavisni Dominirajući Skup: (rezultati)
```

Ova metoda pokazuje značajnu efikasnost u pronalaženju rešenja za problem minimalnog nezavisnog dominantnog skupa, uz potencijal za dodatna unapreenja kroz fino podešavanje parametara.

Test Name	Number of Nodes	Fitness Value	Number of Iterations	Time (s)
test_10_0.3.in	2	0.5	10000	0.93
test_10_0.5.in	3	0.33	10000	1.39
test_10_0.7.in	2	0.5	10000	0.93
test_15_0.3.in	4	0.25	10000	2.42
test_15_0.5.in	2	0.5	10000	1.91
test_20_0.3.in	5	0.2	10000	4.40
test_20_0.5.in	3	0.33	10000	3.19
test_25_0.3.in	4	0.25	10000	5.06
test_25_0.5.in	3	0.33	10000	4.37
test_25_0.7.in	2	0.5	10000	3.74
test_30_0.3.in	4	0.25	10000	9.27
test_30_0.5.in	3	0.33	10000	6.36
test_35_0.3.in	6	0.17	10000	10.00
test_35_0.5.in	3	0.33	10000	7.49
test_40_0.3.in	5	0.2	10000	10.00
test_40_0.5.in	3	0.33	10000	10.00
test_40_0.7.in	2	0.5	10000	7.92

Table 2: Rezultati testiranja za minimalni nezavisni dominantni skup VNS metodom

3.3 Genetski Algoritam (GA)

Genetski algoritam (GA) je optimizacijska tehnika inspirisana prirodnom selekcijom, koja koristi procese evolucije za rešavanje složenih problema. U ovoj sekciji detaljno opisujemo implementaciju GA za pronalaženje minimalnog nezavisnog dominantnog skupa čvorova u grafu.

3.3.1 Opis Koda

Implementacija genetskog algoritma se sastoji iz nekoliko ključnih klasa i funkcija:

- **Klasa Individual:**
  - Klasa `Individual` predstavlja jedno rešenje problema. Svaki `Individual` sadrži kod (niz binarnih vrednosti) koji predstavlja uključene čvorove, i izračunava svoju fitness vrednost kroz metodu `calc_fitness`.
  - Funkcije `is_independent_set` i `is_dominating_set` proveravaju da li je skup čvorova nezavistan ili dominantan.
- **Selekcija:**
  - Funkcija `selection(population, tournament_size)` koristi turnirsku selekciju da odabere roditelje za kreiranje novog naraštaja. Izvodi se nasumični uzorak iz populacije, a najbolji pojedinac se bira kao roditelj.
- **Kros-over:**
  - Funkcija `crossover(parent1, parent2, child1, child2)` kombinuje kodove dva roditelja u dva nova potomka na osnovu nasumične tačke preseka.

- **Mutacija:**
  - Funkcija `mutation(individual, mutation_prob)` nasumično menja kodove pojedinaca kako bi se uvela genetska raznolikost, čime se sprečava stagnacija algoritma.
- **Genetski algoritam:**
  - Funkcija `ga(population_size, num_generations, tournament_size, elitism_size, mutation_prob, graph)` implementira glavni proces genetskog algoritma. U svakoj generaciji, populacija se sortira, elitizam se primenjuje, roditelji se biraju, kros-over i mutacija se primenjuju kako bi se stvorili novi pojedinci.

3.3.2 Rezultati

Funkcija `ga_main(graph)` pokreće genetski algoritam sa unapred definisanim parametrima. Na kraju, najbolji rezultat se prikazuje u konzoli, a graf se vizualizuje koristeći funkciju `draw_graph`.

```
--- %s seconds ---
Najbolji fitnes: (rezultati)
```

Test Name	Number of Nodes	Fitness Value	Number of Iterations	Time (s)
test_10_0.3.in	2	0.5	10000	0.10
test_10_0.5.in	3	0.33	10000	0.16
test_10_0.7.in	2	0.5	10000	0.09
test_15_0.3.in	5	0.2	10000	0.11
test_15_0.5.in	2	0.5	10000	0.14
test_20_0.3.in	6	0.17	10000	0.25
test_20_0.5.in	4	0.25	10000	0.17
test_25_0.3.in	6	0.17	10000	0.25
test_25_0.5.in	4	0.25	10000	0.22
test_25_0.7.in	3	0.33	10000	0.15
test_30_0.3.in	6	0.17	10000	0.20
test_30_0.5.in	4	0.25	10000	0.17
test_35_0.3.in	7	0.14	10000	0.38
test_35_0.5.in	4	0.25	10000	0.18
test_40_0.3.in	6	0.17	10000	0.32
test_40_0.5.in	5	0.2	10000	0.25
test_40_0.7.in	3	0.33	10000	0.28

Table 3: Rezultati testiranja za minimalni nezavisni dominantni skup Genetskim algoritmom

Ova metoda pokazuje efikasnost u pronalaženju minimalnog nezavisnog dominantnog skupa, sa sposobnošću da istraži široku oblast rešenja kroz evolucijske procese.

3.4 Optimizacija pomoću Metode Mravlje Kolonije (ACO)

Metoda Mravlje Kolonije (ACO) je bioinspirisana optimizacijska tehnika koja koristi ponašanje mrava u potrazi za hranom kako bi rešila složene probleme. U ovoj sekciji opisujemo implementaciju ACO za pronalaženje minimalnog nezavisnog dominantnog skupa čvorova u grafu.

3.4.1 Opis Koda

Implementacija ACO sadrži nekoliko ključnih funkcija:

- **is_independent_set:**
  - Ova funkcija proverava da li je dati skup čvorova nezavistan. Ako postoji ivica izmeu bilo koja dva čvora u skupu, funkcija vraća `False`.
- **is_dominating_set:**
  - Ova funkcija proverava da li je dati skup čvorova dominantan. Svi čvorovi u grafu moraju biti ili u skupu ili susedni čvorovima u skupu.
- **fitness:**
  - Funkcija `fitness` izračunava fitness vrednost rešenja. Ako je rešenje nezavisno i dominantno, fitness se izračunava kao inverz vrednosti broja čvorova u skupu, u suprotnom se vraća negativna vrednost.
- **aco_minimum_independent_dominating_set:**
  - Ova funkcija implementira ACO. U svakom iteraciji, svaki mrav gradi rešenje na osnovu feromona i preostalih čvorova. Vrednost feromona se ažurira u zavisnosti od fitness vrednosti rešenja.

3.4.2 Rezultati

Funkcija `aco_minimum_independent_dominating_set(graph, num_ants, num_iterations, alpha, beta, evaporation_rate)` pokreće ACO sa unapred definisanim parametrima. Na kraju, najbolji rezultat se prikazuje u konzoli, a graf se vizualizuje koristeći funkciju `draw_graph`.

```
Best Independent Dominating Set: (rezultati)
Fitness (1 / length of the set): (rezultati)
```

Ova metoda se pokazala kao efikasna u pronalaženju minimalnog nezavisnog dominantnog skupa, omogućavajući istraživanje rešenja kroz iteracije i feromone.

Test Name	Number of Nodes	Fitness Value	Number of Iterations	Time (s)
test_10_0.3.in	3	0.33	1000	3.70
test_10_0.5.in	3	0.33	1000	2.03
test_10_0.7.in	2	0.5	1000	1.84
test_15_0.3.in	5	0.2	1000	3.72
test_15_0.5.in	2	0.5	1000	2.35
test_20_0.3.in	5	0.2	1000	4.19
test_20_0.5.in	3	0.33	1000	2.52
test_25_0.3.in	5	0.2	1000	4.12
test_25_0.5.in	3	0.33	1000	3.02
test_25_0.7.in	2	0.5	1000	2.96
test_30_0.3.in	4	0.25	1000	3.86
test_30_0.5.in	4	0.25	1000	4.54
test_35_0.3.in	8	0.125	1000	9.62
test_35_0.5.in	4	0.25	1000	5.25
test_40_0.3.in	6	0.17	1000	4.31
test_40_0.5.in	4	0.25	1000	4.93
test_40_0.7.in	3	0.33	1000	3.95

Table 4: Rezultati testiranja za minimalni nezavisni dominantni skup koristeći Ant Colony Optimization

4 Uporedna Analiza

U ovoj sekciji uporeujemo rezultate svih implementacija. Tabela 5 prikazuje performanse različitih pristupa.

Test Name	**Brute Force**	**VNS**	**Genetski**	**ACO**
test_10_0.3.in	2	2	2	3
test_10_0.5.in	3	3	3	3
test_10_0.7.in	2	2	2	2
test_15_0.3.in	4	4	5	5
test_15_0.5.in	2	2	2	2
test_20_0.3.in	5	5	6	5
test_20_0.5.in	3	3	4	3
test_25_0.3.in	4	4	6	5
test_25_0.5.in	3	3	4	3
test_25_0.7.in	2	2	3	2
test_30_0.3.in	/	4	6	4
test_30_0.5.in	/	3	4	4
test_35_0.3.in	/	6	7	8
test_35_0.5.in	/	3	4	4
test_40_0.3.in	/	5	6	6
test_40_0.5.in	/	3	5	4
test_40_0.7.in	/	2	3	3

Table 5: Broj čvorova za svaki test i algoritam

Test Name	**Brute Force (s)**	**VNS (s)**	**Genetski (s)**	**ACO (s)**
test_10_0.3.in	0.0010	0.9300	0.1030	3.7007
test_10_0.5.in	0.0010	1.3901	0.1578	2.0337
test_10_0.7.in	0.0010	0.9280	0.0890	1.8351
test_15_0.3.in	0.0380	2.4229	0.1060	3.7234
test_15_0.5.in	0.0260	1.9116	0.1385	2.3490
test_20_0.3.in	1.2748	4.3955	0.2510	4.1910
test_20_0.5.in	0.6900	3.1933	0.1736	2.5222
test_25_0.3.in	27.9059	5.0600	0.2548	4.1194
test_25_0.5.in	26.8291	4.3726	0.2200	3.0172
test_25_0.7.in	23.3268	3.7416	0.1476	2.9572
test_30_0.3.in	0.0	9.2687	0.2040	3.8563
test_30_0.5.in	0.0	6.3634	0.1698	4.5444
test_35_0.3.in	0.0	10.0042	0.3790	9.6152
test_35_0.5.in	0.0	7.4882	0.1820	5.2517
test_40_0.3.in	0.0	10.0047	0.3160	4.3098
test_40_0.5.in	0.0	10.0019	0.2481	4.9279
test_40_0.7.in	0.0	7.9168	0.2840	3.9456

Table 6: Vreme izvršenja za svaki test i algoritam

5 Zaključak

U ovom istraživanju, analizirali smo različite heurističke metode za pronalaženje minimalnog nezavisnog dominantnog skupa čvorova u grafu, uključujući VNS, GA, BRUT i ACO. Na osnovu dobijenih rezultata možemo izvući nekoliko značajnih zaključaka:

Brzina: Metoda Brut Force je imala izuzetno dugačko vreme izvršavanja, posebno za veće grafove, dok su VNS i Genetski algoritam bili brži od ACO. Ovo sugeriše da VNS može biti bolji izbor kada je brzina važnija od tačnosti rešenja.

Fleksibilnost: Heurističke metode, kao što su GA i VNS, pružaju veću fleksibilnost i bolje se prilagoavaju različitim tipovima grafova, što može biti ključno u primenama gde je potrebno prilagoditi pristup specifičnim zahtevima.

Praktična primena: Naši rezultati sugerišu da se heuristički pristupi mogu uspešno koristiti u praksi za rešavanje problema minimalnog nezavisnog dominantnog skupa čvorova, posebno u situacijama gde su klasične metode previše zahtevne ili neefikasne.

Preporuke za budući rad: Istraživanje može biti prošireno uključivanjem drugih heurističkih metoda i optimizacija, kao i ispitivanjem njihovih kombinacija za još bolje rezultate. Takoe, preporučuje se dalje istraživanje kako bi se unapredila tačnost trenutnih metoda.