Rešavanje problema minimalnog particionisanja grafa na klike genetskim algoritmom

Seminarski rad u okviru kursa
Računarska inteligencija
Matematički fakultet

Rudinac Katarina, Nićković Teodora
rudinackatarina@gmail.com
nickovic97@gmail.com

Sažetak

U ovom radu izvršen je opis i poređenje tri varijacije genetskog algoritma čiji je cilj bio rešavanje problema minimalnog particionisanja grafa na klike. Rešavanje problema svedeno je na problem bojenja grafa zarad eksperimentalnog testiranja performansi nad DIMACS skupom.

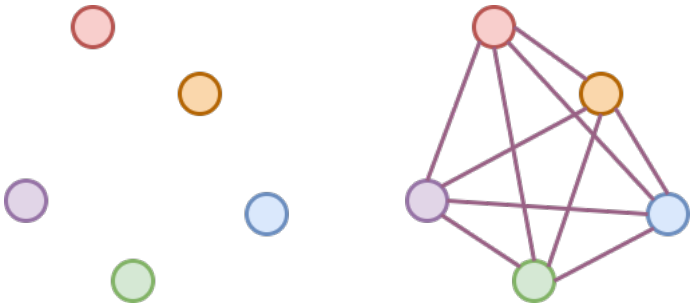
Sadržaj

1 Uvod

Prema Merriam Webster rečniku, klika je definisana kao *"uzak krug ljudi sa zajedničkim interesovanjima, stavovima ili verovanjima"* [1]. Iz same definicije, može se naslutiti zašto bi ovakve grupe bile predmet mnogobrojnih psiholoških, sociolоških i marketinških istraživanja - u jako povezanim grupama, postoji visok nivo poverenja i informacije se brzo šire [3]. Iz tog razloga, nije iznenađujuće ni to što se u teoriji grafova, termin 'klike' javlja upravo kod istraživača koji su se bavili društvenim mrežama (ne misli se na Internet aplikacije, već generalne mreže poznanstava u društvu). Luk i Peri [22] su klikama nazivali kompletne podgrafove - podgrafove u kojima su svaka dva čvora povezana, koji su predstavljali grupe ljudi u kojima se svake dve osobe međusobno poznaju. Formalnije, korisne osobine ovakvih grupa, kao što su bliskost i dostupnost, mogu se opisati i u terminima grafova. Bliskost se može opisati kao visok stepen čvorova u podgrafu, a dostupnost kao činjenica da je prečnik takvog podgrafa jednaka 1. Pre razmatranja problema particionisanja na klike, potrebno je navesti definicije pojmova i notaciju koja će se koristiti u nastavku.

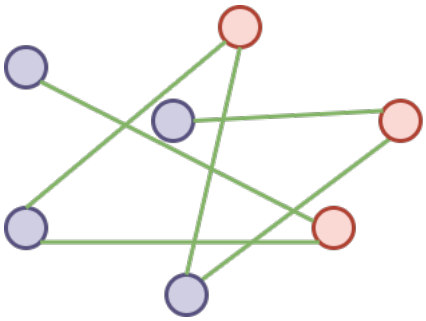
1.1 Osnovni pojmovi

$G = (V, E)$ predstavlja neusmereni graf sa skupom čvorova $V = \{1, ..., n\}$ i skupom ivica $E \subseteq V \times V$. **Komplement grafa** označavamo sa $\overline{G}$. Važi $\overline{G} = (V, \overline{E})$, gde je $\overline{E} = \{(i, j) \,|\, i, j \in V \wedge (i, j) \notin E\}$. $G(S)$ je podgraf indukovan skupom S, gde je $S \subseteq V$. Skup čvorova C je **klika** ako je podgraf $G(S)$ potpun graf (svi čvorovi su medjusobno povezani). Skup čvorova S u grafu G je **nezavistan skup** ako nikoja dva čvora iz S nisu povezana.



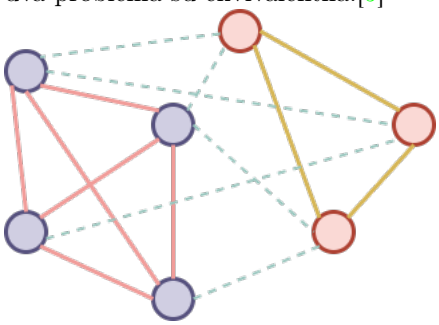
Ovde možemo videti nezavistan skup čvorova, kao i kliku sa istim skupom čvorova.

Problem minimalnog particionisanja na klike je zapravo problem podele čvorova grafa G na minimalan broj podskupova čvorova koji međusobno čine klike. Ovaj problem je NP-težak a njegov problem odlučivanja je NP-kompletan, i rešava se pretragom u dubinu, koja je uvek eksponencijalna. To je jedan od dvadeset jednog NP-kompletnog problema koje je Ričard Karp opisao u svom radu "Redukovanje kombinatornih problema"iz 1972.[20] Međutim, ispostavlja se da je ovaj problem moguće svesti na problem bojenja grafa, koji će biti opisan u daljem tekstu.



Ovde je prikazan graf obojen sa 2 različite boje tako da čvorovi na krajevima nikoje dve ivice nisu iste boje.

Bojenje grafa G zapravo znači dodeljivanje boja čvorovima grafa, tako da nikoja dva čvora koja su povezana ivicom budu iste boje. Graf G je k-obojiv ukoliko postoji bojenje tog grafa koje ne koristi više od k različitih boja da oboji čvorove pomenutog grafa. Minimalno k za koje se G može obojiti se zove **hromatski broj** i oznaka za to je $\chi(G)$ i nalaženje ovog broja, kao i čvorova koji spadaju u skupove određenih boja, spada u problem bojenja grafa. Ispostavlja se da je i ovaj problem NP-težak kao i problem particionisanja i da spada u probleme kombinatorne optimizacije. Bojenjem grafa, skup čvorova tog grafa se particioniše na nezavisne skupove. Nezavistan skup u G zapravo je klika u njegovom komplementu $\overline{G}$. Dakle, ako je dat početni graf G i uradi se bojenje za komplement $\overline{G}$, dobija se particionisanje skupa čvorova koje je za $\overline{G}$ particionisanje na nezavisne skupove, a za G particionisanje na klike. Prema tome, ova dva problema su ekvivalentna.[5]



Ovde se vidi komplement od goreobojenog grafa, sa čvorovima podeljenim u dva skupa koji čine klike.

Problem minimalnog particionisanja na klike ćemo u nastavku rešavati svođenjem na problem bojenja. Ovo će dosta olakšati kasnije eksperimentalno testiranje, zbog toga što za problem bojenja postoje javno testirani skupovi [8][2]. Kako su nam značajne samo primene problema particionisanja na klike, ali ne i bojenja grafa, nećemo se mnogo zadržavati na analizi radova koje drugi problem primenjuju. Problem bojenja značajan nam je isključivo kao dobro proučen, pa time i efikasno rešavan problem, koji je ekvivalentan sa našim polaznim problemom. Navešćemo tahnike koje su do sada korišćene za njegovo rešavanje i porediti njihovu efikasnost (deo 1.2). Nakom toga ćemo izložiti implementaciju našeg rešenja, koje je kombinacija metoda koje su se pokazale dobro u radovima

drugih autora (deo 2). Testiraćemo performanse implementacije nad DIMACS skupom (deo 3). Na kraju, kratko ćemo se osvrnuti na primenu problema minimalnog particionisanja na klike, i kako se problemi iz stvarnog života mogu modelirati na ovaj način (deo 4).

2 Istorijat rešavanja problema bojenja

Pre analize konkretnih radova, korisno je dati grub pregled tehnika koje se koriste za rešavanje problema kombinatorne optimizacije[19][17]. Postoje četiri glavna pristupa:

Konstruktivne metode. Rešenje se konstruiše korak po korak, u skladu sa statički ili dinamički definisanim redosledom. Takve metode su na primer pohlepne metode (engl. *greedy methods*) i grananje sa ograničavanjem (engl. *branch and bound*).

Lokalna pretraga. Kreće se od jedne kompletne konfiguracije (jednog rešenja) koje se iterativno modifikuje, tako da (generalno) bira lokalno optimalnu modifikaciju. Takve metode su spuštanje nizbrdo (engl. *descent*), simulirano kaljenje (engl. *simulated annealing*) i tabu pretraga (engl. *tabu search*).

Evolutivne metode. Kreće se od celokupne populacije rešenja, koja simulacijom evolutivnih procesa, tj. korišćenjem operatora selekcije, ukrštanja i mutacije teži ka boljim rešenjima.

Hibridne metode. Bilo kakve kombinacije prethodnih tehnika.

Za instance male veličine, moguće je koristiti i egzaktne algoritme, ali već za grafove sa više od 100 čvorova, one postaju neupotrebljive zbog jako loše efikasnosti [12].

Prve heuristike koje su se koristile za rešavanje problema bojenja zasnovane su na pohlepnim metodama. Ovakve metode su efikasne, ali njihova rešenja su daleko od zadovoljavajućih [6][21]. Međutim, kako ipak daju rešenje mnogo bolje od slučajno generisanog, a dosta su efikasne, ovakve metode su korisne za generisanje početnog rešenja za druge metaheuristike, pogotovo za metode lokalne pretrage.

Što se metoda lokalne pretrage tiče, jedno od prvih predloženih rešenja, koje je i danas popularno, bio je algoritam *Tabucol*, algoritam koji za bojenje grafa koristi tabu pretragu [18]. Objavljen je godinu dana nakon što je Fred Glover formulisao tabu pretragu i pokazao njene dobre performanse na širokom spektru problema [15]. Ekstenzivan pregled algoritama lokalne pretrage koji rešavaju problem bojenja, zainteresovani čitalac može naći u [12]. Kako su trenutno aktuelne metode za rešavanje ovog algoritma dosta efikasnije od lokalnih pretraga, nećemo se previše zadržavati na pojedinostima ovih implementacija.

Genetski algoritam, iako primenljiv na dosta različitih klasa problema, bez mnogo prilagođavanja konkretnom problemu daje nekompetitivne rezultate [19]. Za problem bojenja to vidimo i u [9].

Sredinom devedesetih stvara se ideja o hibridnim metodama, pogotovo o kombinaciji evolutivnih i metoda lokalne pretrage. Intuitivno, ovo zvuči kao dobra ideja, jer koristi prednosti oba pristupa - evolutivnim algoritmima koji pretražuju ceo prostor rešenja smanjujemo verovatnoću da se zaglavimo u lokalnim minimumima, a lokalnom pretragom smo u stanju da višetruko poboljšamo svaku jedinku u populaciji. Ono što gubimo na efikasnosti zbog evolutivnih metoda koje pretražuju široko, kao i na tome što dodajemo operator lokalne pretrage, nadoknađujemo jakim jedinkama. Ovakva intuicija ispostaviće se kao dobra u [7](1995) [11](1996), što su ujedno i prvi radovi koji spajaju genetski algoritam sa lokalnom pretragom (eng. *genetic local search* - ***GLS***). Glavna razlika ovako formulisanog genetskog algoritma od običnog je u operatoru mutacije, koji je ovde zamenjen operatorom lokalne pretrage. U [7] taj operator je spuštanje nizbrdo, a u [11] to je upravo *Tabucol*. S obzirom da su autori drugog rada koristili malo izmenjenu verziju algoritma *Tabucol*, teško je reći da li je genetski algoritam tu doneo nešto više od dobrog načina divesifikacije. Rezultati jesu bolji, ali ne mnogo.

Nakon ovoga, većina radova o rešavanju problema bojenja koristi neki od sledeća tri pristupa, ili njihovu modifikaciju (navodimo i radove sa značajnim rezultatima za svaki od pristupa):

- GLS sa izmenjenim operatorima genetskog algoritma (najznačajnije su izmene na operatoru ukrštanja [10][14])
- Metoda promenljivih okolina (eng. *variable neighbourhood search*) [4]
- Algoritmi adaptivne memorije (eng. *adaptive memory algorithms*), koji umesto jednostavnog ukrštanja kao kod genetskih algoritama, čuvaju se određeni delovi rešenja, koji se onda koriste za izgradnju novih jedinki. [13]

Bitno je naglasiti i da se određene implementacije pokazuju sjajno za grafove sa nekim skupom karakteristika, ali su njihove performanse mnogo lošije za drugačije grafove. Na primer, dosta rešenja značajno opada u kvalitetu kada se testira na slučajno generisanim grafovima. U ovom radu, nismo se osvrtali na karakteristike grafova nad kojima testiramo, iako specijalizovani pristupi, ili bilo kakvo uzimanje ovoga u obzir, može dosta uticati na kvalitet rešenja. U [16], zainteresovani čitaoci mogu naći duboku analizu ove problematike.

3 Opis rešenja

Na osnovu prethodnog osvrta na uspešnost različitih metoda rešavanja problema bojenja, implementiraćemo jednostavan genetski algoritam i dva hibridna - jedan koji koristi jednostavan operator spuštanja nizbrdo kao lokalnu pretragu, i drugi koji koristi tabu pretragu.

3.1 SimpleGA

Ovo nije previše efikasan algoritam, ali budući da implementacija kreće od njega, priložen je, kao i kako bi se na njemu najjednostavnije objasnio način kodiranja.

3.1.1 Kodiranje

Za hromozom je uzeta lista od n elemenata, gde n predstavlja broj čvorova u grafu. Indeksi u listi predstavljaju redne brojeve čvorova, a vrednosti na tim indeksima predstavljaju boje čvorova. Takođe, postoji i lista povezanosti preko koje je opisan graf, gde liste na svakom indeksu predstavljaju čvorove koji su susedni čvoru koji ima taj indeks.

3.1.2 Funkcija prilagođenosti i funkcija cilja

Prilagođenost hromozoma se računa kao broj čvorova koji su nepravilno obojeni, a nepravilna obojenost čvora podrazumeva da ima istu boju kao bilo koji od susednih čvorova. Funkcija prilagođenosti se može učiniti preciznijom ako se broje ne samo pogrešno obojeni čvorovi, već konflikti koje svaki pogrešno obojen čvor izaziva. Recimo, ukoliko je čvor pogrešno obojen, ali je samo jedan sused tako obojen, to predstavlja manji problem od drugog čvora koji ima veći broj identično obojenih suseda. Ciljana vrednost funkcije prilagođenosti je, prema tome, nula. Kriterijum zaustavljanja je ili dostizanje nule, ili prekoračenje maksimalnog broja iteracija.

```
def calculate_fitness(self, code):
    fitness = 0
    for i in range(self.num_vertices):
        for j in self.adjacency_list[i]:
            if code[i] == code[j]:
                fitness += 1
    return fitness
```

3.1.3 Selekcija

Za implementaciju selekcije, odabrana je turnirska selekcija. Ona podrazumeva odabir hromozoma koji funkcioniše po principu biranja određenog broja hromozoma, od kojih onaj sa najboljom funkcijom prilagođenosti potom učestvuje u novoj generaciji. To se ponavlja onoliko puta koliko je novih hromozoma potrebno, s tim da postoji i opcioni elitizam, gde je određeni broj hromozoma rezervisan za najbolje hromozome koji se biraju prostim sortiranjem na osnovu funkcije prilagođenosti.

```
def tournament_selection(self):
    selected = []
    for i in range(self.reproduction_size - self.elitism_rate):
        current_pool = []
        for i in range(self.tournament_size):
            current_chromosome = random.choice(self.population)
            current_pool.append(current_chromosome)
        best_chromosome = self.select_best_chromosome(current_pool)
        selected.append(best_chromosome)
    selected.extend(heapq.nsmallest(self.elitism_rate, self.population))
    return selected
```

3.1.4 Operator ukrštanja

Pri ukrštanju, mogu se koristiti dva operatora. Jedan je klasičan jednopozicioni, koji na nasumičan način bira tačku prekida i onda jednom detetu dodeljuje

gene prvog roditelja od nulte pozicije do tačke prekida, i od tačke prekida do kraja gene drugog roditelja, dok je kod drugog deteta obrnuto. Drugi operator je uniformni sa preferencom ka genima boljeg roditelja. On prvo pronalazi boljeg roditelja na osnovu funkcije prilagođenosti, i potom za svaki pojedinačni gen nasumično bira vrednost u opsegu od nula do jedan. Ukoliko je ta vrednost manja od neke unapred zadate verovatnoće, oba deteta uzimaju gen na i-toj poziciji od boljeg roditelja, dok u suprotnom samo jedno dete uzima gen na toj poziciji od lošijeg roditelja.

```
def crossover_chunk(self, parent1, parent2):
    cross_point = random.randint(1, self.num_vertices - 1)
    child1_code = parent1.code[:cross_point] + parent2.code[cross_point:]
    child2_code = parent2.code[:cross_point] + parent1.code[cross_point:]

    child1 = Chromosome(child1_code, self.calculate_fitness(child1_code))
    child2 = Chromosome(child2_code, self.calculate_fitness(child2_code))

    return child1, child2
```

```
def uniform_crossover(self, parent1, parent2):
    prob = 0.5
    better_parent = None
    if parent1.fitness < parent2.fitness:
        better_parent = parent1
        worse_parent = parent2
    else:
        better_parent = parent2
        worse_parent = parent1

    child1_code = []
    child2_code = []
    for i in range(self.num_vertices):
        r = random.random()
        if r < prob:
            child1_code.append(better_parent.code[i])
            child2_code.append(better_parent.code[i])
        else:
            child1_code.append(better_parent.code[i])
            child2_code.append(worse_parent.code[i])

    child1 = Chromosome(child1_code, self.calculate_fitness(child1_code))
    child2 = Chromosome(child2_code, self.calculate_fitness(child2_code))

    return child1, child2
```

3.1.5 Mutacija

Umesto da mutacija bude potpuno nasumična na malom procentu jedinki, ona je promišljena, na većem procentu. Naime, ukoliko postoji validno bojenje za neki gen, on će se obojiti tom bojom. Ukoliko ne postoji, tek tada se boja bira nasumično.

```
def mutate(self, chromosome):
    p = random.random()
    if p < self.mutation_rate:
        i = random.randrange(0, len(chromosome) - 1)
        available_colors = self.get_available_colors(chromosome, i)
        if len(available_colors) > 0:
            chromosome[i] = random.choice(available_colors)
        else:
            chromosome[i] = random.choice(self.possible_gene_values)
```

```
    return chromosome
```

3.2 TabuGA

TabuGA je genetski algoritam koji kao operator lokalne pretrage primenjuje tabu pretragu. Kriterijum za pokretanje pretrage je određena granica u kvalitetu jedinke. Ukoliko je prilagođenost lošija od predodređene granice, na jedinku će se određeni broj puta primeniti tabu pretraga, kako bi se njen kvalitet poboljšao.

```
def tabu_search(self, chromosome):
    current_solution = chromosome
    best_solution = chromosome
    tabu_list = []
    for i in range(self.max_tabu_iters):
        if current_solution.fitness < best_solution.fitness:
            best_solution = current_solution
        nbhd, moves = self.generate_neighbourhood(chromosome, tabu_list)
        current_solution = self.select_best_chromosome(nbhd)
        index_of_current_solution = nbhd.index(current_solution)
        tabu_list.append(moves[index_of_current_solution])
    return best_solution
```

3.3 DescentGA

DescentGA je hibridni genetski algoritam koji kao operator lokalne pretrage koristi jednostavan operator spuštanja nizbrdo. To je jedina razlika između TabuGA i DescentGA, kao i jedini dodatak u odnosu na SimpleGA. Kriterijum za upotrebu je isti kao i kod tabu pretrage - ukoliko je prilagođenost jedinke lošija od ogređene granice, potrebno je primeniti lokalnu pretragu.

```
def descent(self, chromosome):
    for k in range(self.max_descent_iters):
        current_chromosome = chromosome.code
        for i in chromosome.code:
            for j in self.adjacency_list[i]:
                if chromosome.code[i] == chromosome.code[j]:
                    available_colors=self.get_available_colors(chromosome,i)
                    if len(available_colors) > 0:
                        chromosome[i] = random.choice(available_colors)
        if chromosome.fitness > self.calculate_fitness(chromosome):
            chromosome = Chromosome(chromosome,
                                    self.calculate_fitness(chromosome))

    return chromosome
```

4 Poređenje rezultata

5 Zaključak

A Dodatak

Rešenja za sve DIMACS grafove je teško naći. Ovde prilaženo link ka nezvaničnom skupu svih rešenja koja postoje: https://medium.com/@aditya.yadav/dimacs-

Ime grafa	Benchmark (s)	SimpleGA	DescentGA	TabuGA
myciel3.col	11	0.637598	.	.
myciel4.col	16	0.770418	.	.
myciel5.col	16	0.805036	.	.
myciel6.col				
huck.col	16	0.961352		
jean.col 90	6	1.001105		
games120.col	6	1.090587		

graph-coloring-chromatic-number-benchmarks-the-millennium-breakthrough-in-computing-eba6df44873c. Kako ovo nije zvaničan skup, primeri koji su korišćeni su oni koji se pojavljuju i na drugim mestima iz referenci.

Literatura

[1] https://www.merriam-webster.com/dictionary/clique.

[2] http://dimacs.rutgers.edu/archive/Challenges/.

[3] Robert Adlam and Peter Villiers. *Policing a Safe, Just and Tolerant Society: An International Model for Policing*. Waterside Press, 2004.

[4] C. Avanathay, A. Hertz, and N. Zufferey. A variable neighborhood search for graph coloring. *European Journal of Operational Research*, 151:379–388, 2003.

[5] J. Bhasker and Tariq Samad. The clique-partitioning problem. *Computers Mathematics with Applications*, 22(6):1–11, 1991.

[6] Daniel Brélaz. New methods to color the vertices of a graph. *Commun. ACM*, 22(4):251–256, 1979.

[7] D. Costa, A. Hertz, and O. Dubuis. Embedding of a sequential procedure within and evolutionary algorithm for coloring problem in graphs. *Journal of Heuristics*, 1:105–128, 1995.

[8] M. A. Trick D. S. Johnson. Cliques, coloring, and satisfiability: second dimacs implementation challenge. *DIMACS series in discrete mathematics and theoretical computer science*, 1996.

[9] L. Davis. Order-based genetic algorithms and the graph coloring problem. In *Handbook of genetic algorithms*, pages 72–90. Van Nostrand Reinhold, 1991.

[10] Raphael Dorne and Jin-Kao Hao. A new genetic local search algorithm for graph coloring. *Lecture notes in computer science*, 1498:745–754, 1998.

[11] C. Fleurent and J. A. Ferland. Genetic and hybrid algorithms for graph coloring. *Annals of Operations Research*, 63:437–61, 1996.

[12] P. Galinier and A. Hertz. A survey of local search methods for graph coloring. *Computers and Operations Research*, 33(9):2547–2562, 2006.

[13] P. Galinier, A. Hertz, and N. Zufferey. An adaptive memory algorithm for the k-colouring problem. *Les cahiers du GERAD*, 35, 2003.

[14] Philippe Galinier and Jin-Kao Hao. Hybrid Evolutionary Algorithms for Graph Coloring. *Journal of Combinatorial Optimization*, 3(4):379–397, 1999.

[15] Fred Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13(5):533–549, 1986.

[16] Jean-Philippe Hamiez and Jin-Kao Hao. An analysis of solution properties of the graph coloring problem. In M.G.C. Resende and J.P. de Sousa, editors, *Metaheuristics: Computer Decision-Making*, chapter 15, pages 325–346. Kluwer, 2003.

[17] Jin-Kao Hao. Metaheuristics for combinatorial optimization. `http://www.info.univ-angers.fr/pub/hao/papers/Tutorial.pdf`.

[18] Alain Hertz and D. Werra. Using tabu search techniques for graph coloring. *Computing*, 39(4):345–351, 12 1987.

[19] Predrag Janičić and Mladen Nikolić. *Veštačka inteligencija*. Matematički fakultet, 2019.

[20] R. Karp. Reductibility among combinatorial problems. *Proceedings of a Symposium on the Complexity of Computer Computations*, 35:85—-103, 1972.

[21] F.T. Leighton. A graph coloring algorithm for large scheduling problems. *Journal of Research of the National Bureau of Standards*, 84(6):489–506, 1979.

[22] R. Duncan Luce and Albert D. Perry. A method of matrix analysis of group structure. *Psychometrika*, 14(2):95–116, 1949.