Genetski algoritam i hibridni genetski algoritam primenjeni na problem bojenja grafova

Seminarski rad u okviru kursa
Računarska inteligencija
Matematički fakultet

Selena Vukadinović, Aleksandar Jakovljević,
vukadinovic.selena@gmail.com, a.jakovljevic96@gmail.com

15. juni 2019

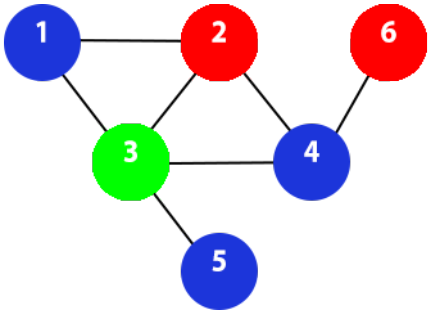
Sažetak

U radu su prikazana dva algoritma za rešavanje problema bojenja grafa. Prvi algoritam, genetski algoritam, koristi operatore uobičajene za genetske algoritme prilagođene problemu bojenja grafova, dok hibridni genetski algoritam dodatno koristi i lokalnu pretragu. Algoritmi su nad relevantnim DIMACS test primerima dali zadovoljavajuće rezultate. Sprovedenom analizom nad rezultatima može se uočiti mesto za poboljšanje oba algoritma.

Sadržaj

1 Uvod

Problem bojenja grafa predstavlja NP-kompletan problem [8] i odnosi se na problem dodele boja čvorovima grafa. Za dati graf $G(V, E)$, gde je V skup čvorova, a E skup grana, potrebno je dodeliti čvorovima boju iz skupa od k boja, gde je k prirodan broj, tako da za dva susedna čvora, odnosno dva čvora povezana granom, važi da su različtih boja. Ispravno bojenje grafa se naziva valjano bojenje grafa 1. Broj boja koje su dodeljene čvorovima grafa naziva se hromatski broj grafa u oznaci $\chi(G)$. Broj boja je ograničen tako da važi da je gornja granica $\chi(G) \leq 1 + \Delta(G)$, gde je $\Delta(G)$ stepen grafa, dok se donja granica naziva minimalni hromatski broj grafa [7]. Pronalaženje minimalnog hromatskog broja grafa predstavlja NP-težak problem [8].



Slika 1: Valjano bojenje grafa

1.1 Prethodni radovi

Veliki broj radova je napisan na temu rešavanja problema bojenja grafova koji uključuju korišćenje genetskih algoritama [5, 1, 3]. Kao efikasne algoritme ističe se hibridni evolutivni algoritam koji koristi specijalizovano ukrštanje i tabu pretragu [2], kao i hibridni genetski algoritam koji koristi veći broj operatora selekcije i mutacije u kombinaciji sa mudrošću veštačke gomile (engl. *wisdom of artificial crowd*) i lokalne pretrage [4].

2 Primenjeni algoritmi

U ovom poglavlju opisani su algoritmi koji su korišćeni prilikom rešavanja zadatog problema. Korišćeni su genetski algoritam i njegova modifikacija koja podrazumeva korišćenje lokalne pretrage i drugačijeg operatora mutacije, odnosno hibridni genetski algoritam.

2.1 Genetski algoritam

Genetski algoritmi (GA) poseduju univerzalnu strukturu koja podrazumeva proces selekcije i skup operatora: ukrštanje i mutaciju 1.

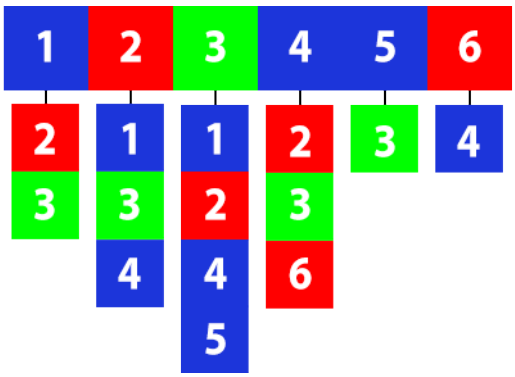
```
1000        while not self.stop_condition():
1002            new_generation =
                heapq.nlargest(self._elite_size, chromosomes)
1004
```

```
            for_reproduction =
1006        self.selection_tournament(chromosomes)

1008        chromosomes =
            self.create_generation(for_reproduction,
1010        new_generation)

1012        self._top_chromosome = Chromosome(
            min(chromosomes, key=attrgetter('fitness')),
1014        min(chromosomes, key=attrgetter('fitness')).fitness)

1016        self._current_iteration += 1
```

Listing 1: Struktura genetskog algoritma

Populaciju čine jedinke čiji je genetski kod lista u kojoj jedan gen predstavlja boju gde indeks u listi odgovara čvoru grafa 2. Korišćena je lista povezanosti za čuvanje podataka o susedima svakog od čvorova 3.

Slika 2: Genetski kod jedinke



Slika 3: Lista povezanosti

2.1.1 Funkcija prilagođenosti

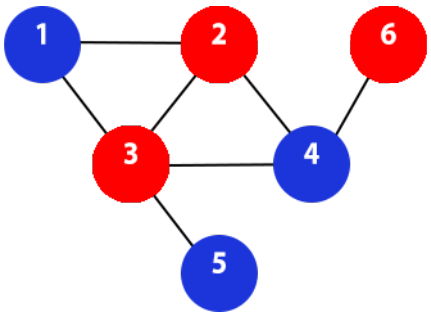
Prolaskom kroz listu povezanosti i upoređivanjem boja čvorova određuje se vrednost funkcije prilagođenosti 2. Ukoliko je boja suseda ista kao i boja posmatranog čvora, uvećava se vrednost funkcije prilagođenosti. Vrednost funkcije cilja je jednaka nuli, te je potrebno minimizovati vrednost funkcije prilagođenosti. Uslov zaustavljanja predstavlja prekoračenje dozvoljenog broja iteracija i jednakost funkcije prilagođenosti sa funkcijom cilja. U algoritmu je korišćen i elitizam kako bi se sačuvale jedinke sa najboljom funkcijom prilagođenosti.

```
1000        def fitness(self, chromosome):
                fitness_value = 0
1002            for i in range(self._num_vertices):
```

```
            for j in self._adjacency_list[i]:
1004            if chromosome[i] == chromosome[j]:
                    fitness_value += 1
1006
        return fitness_value
```

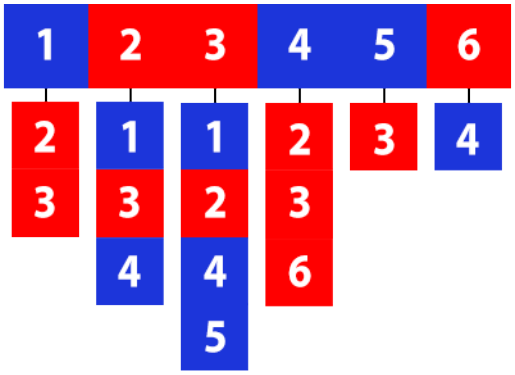
Listing 2: Funkcija prilagođenosti

Vrednost funkcije prilagođenosti jedinke grafa 4 čiji je kod dat na slici 5 i čija je lista povezanosti čvorova data na slici 6 iznosi 2. Naime, čvorovi čiji su indeksi 2 i 3 kao svoje susede imaju čvorove iste obojenosti, što prouzrokuje uvećavanje vrednosti funkcije prilagođenosti.



Slika 4: Neispravno bojenja grafa

Slika 5: Genetski kod jedinke neispravnog bojenja grafa



Slika 6: Lista povezanosti neispravnog bojenja grafa sa slike 4

2.1.2 Selekcija

Za proces selekcije korišćena je turnirska selekcija u kojoj se bira jedinka sa najboljom funkcijom prilagođenosti iz nasumično izabranog skupa jedinki populacije.

```
    def selection_tournament(self, chromosomes):
        return
        [self.selection_tournament_pick(chromosomes, self.
    _tournament_k)
        for i in range(self._reproduction_size - self._elite_size)
    ]

    def selection_tournament_pick(self, chromosomes, k):
        picked = []
        best_i = None
        for i in range(k):
            pick = random.randint(0, self._generation_size - 1)
            picked.append(chromosomes[pick])
            if best_i is None
            or picked[i].fitness < picked[best_i].fitness:
                best_i = i
        return picked[best_i]
```

Listing 3: Proces turnirske selekcije

2.1.3 Reprodukcija

U procesu reprodukcije korišćene su sve jedinke koje su izabrane u procesu selekcije kao i određen broj jedinki koji je sačuvan elitizmom. Kao operator ukrštanja korišćeno je jednopoziciono ukrštanje 4.

```
    def one_point_crossover(self, a, b):
        bp = random.randrange(1, self._num_vertices - 1)
        child1 = a[:bp] + b[bp:]
        child2 = b[:bp] + a[bp:]
        return child1, child2
```

Listing 4: Jednopoziciono ukrštanje

2.1.4 Mutacija

Nakon procesa ukrštanja, primenjuje se operator mutacije 5. Zaglavljivanje u lokalnom optimumu predstavlja veliki problem genetskog algoritma primenjenog nad problemom bojenja grafa, te pojedini autori preporučuju veliku verovatnoću mutacije [4]. Iako velikom verovatnoćom mutacije pretraga počinje da liči na slučajnu pretragu, strategijom elitizma obezbeđuje se čuvanje najkvalitetnijih jedinki i zadržava se usmeravanje pretrage.

```
    def mutation(self, chromosome):
        t = random.random()
        if t < self._mutation_rate:
            i = random.randrange(0, self._num_vertices - 1)
            chromosome[i] = random.choice(self.
    _allowed_gene_values)
        return chromosome
```

Listing 5: Mutacija

O vrednostima parametra algoritama više u nastavku rada 3.

2.2 Hibridni genetski algoritam

Hibridni genetski algoritam (HGA) pored genetskog algoritma koristi dodatne tehnike pronalaženja optimalnog rešenja. U našem slučaju, kodiranje gena, funkcija prilagođenosti 2.1.1 i selekcija 2.1.2, reprodukcija 2.1.3 su iste kao u prethodnom algoritmu 2.1. Koristi se drugačiji operator mutacije 6. Razlika u strukturi je upotreba lokalne pretrage nakon stvaranja nove generacije 7.

2.2.1 Mutacija

Za razliku od prethodnog algoritma gde je mutacija bila neusmerena 5, odnosno birala se boja iz skupa svih boja, ovde je ograničen izbor boja. Ukoliko postoji skup boja, takav da boje iz skupa ne boje nijedan susedan čvor čvoru koji odgovara mutirajućem genu, mutiramo gen u boju iz pomenutog skupa, inače biramo nasumično boju 6.

```
1000    def mutation(self, chromosome):
            t = random.random()
1002        if t < self._mutation_rate:
                i = random.randrange(0, self._num_vertices - 1)
1004            set_of_colors =
                set([chromosome[vertex]
1006                for vertex in self._adjacency_list[i]])
                available_colors =
1008            list(set(self._allowed_gene_values) - set_of_colors)
                if len(available_colors) > 0:
1010                chromosome[i] = random.choice(available_colors)
                else:
1012                chromosome[i] =
                    random.choice(self._allowed_gene_values)
1014        return chromosome
```

Listing 6: Mutacija

2.2.2 Lokalna pretraga

Lokalna pretraga se izvršava nad jedinkama čija je funkcija prilagođenosti ispod unapred zadate granice. Ovo rešenje je nastalo kao modifikacija već postojećeg rešenja [4]. Razlika je u tome što se lokalna pretraga primenjuje nad svim jedinkama koje imaju vrednost ispod granice u svakoj iteraciji, dok je u [4] upotrebljena kao drugi operator mutacije nad najkvalitetnijom jednikom ukoliko je njena funkcija prilagođenosti ispod unapred zadatke granice. Ukoliko se posmatra nesipravno obojen graf sa slike 4 i njegova lista povezanosti 6, prilikom lokalne pretrage genu sa indeksom 3 dodeliće se boja iz skupa dozvoljenih boja, što je u ovom slučaju zelena i dobiće se valjano bojenje grafa 1 čiji je kodiranje 2, a lista povezanosti 3. Na ovaj način pronađeno je rešenje i postignut je uslov zaustavljanja algoritma.

```
1000    def local_search(self,
                         old_chromosome,
1002                     old_chromosome_fitness):
            for k in range(self._local_search_iters):
1004            chromosome = old_chromosome
                for i in old_chromosome:
1006                for j in self._adjacency_list[i]:
                        if old_chromosome[i] == old_chromosome[j]:
1008                        # Pravimo skup boja suseda cvora
                            set_of_colors =
1010                        set([old_chromosome[vertex]
                            for vertex in self._adjacency_list[i]])
1012
```

```
                        set_of_all_colors =
1014                    set(self._allowed_gene_values)
                        # Razlika skupa svih boja i boja suseda
1016                    available_colors =
                        list(set_of_all_colors - set_of_colors)
1018                    # Ukoliko je skup nije prazan,
                        # dodeljujemo boju iz skupa
1020                    # dozvoljenih boja
                        if len(available_colors) > 0:
1022                        chromosome[i] =
                            random.choice(available_colors)
1024            # Racunamo funkciju prilagodjenosti nove jedinke
                chromo_fitness = self.fitness(chromosome)
1026
                # Ukoliko je novo resenje bolje, menjamo staro
1028            # resenje njime
                if old_chromosome_fitness > chromo_fitness:
1030                old_chromosome, old_chromosome_fitness =
                    chromosome, chromo_fitness
1032
        return old_chromosome, old_chromosome_fitness
```

Listing 7: Lokalna pretraga

3 Eksperimentalni rezultati

U nastavku rada slede eksperimentalnim rezultati dobijeni nad odgovarajućim DIMACS test primerima koji se mogu pronaći na sledećoj veb lokaciji [6]. Eksperimenti su sprovedeni na računaru čije su specifikacije sledeće:

- Procesor: Intel(R) Core(TM) i5-4210U CPU @ 1.70GHz (4 CPUs)
- RAM: 8 GB DDR3 @1600MHz

Za izvršavanje algoritama, date su u tabeli 1 korišćene vrednosti parametra. Određeni su eksperimentalno i uz preporuku drugih autora. [4].

Tabela 1: Vrednosti parametra.

GA i HGA	
naziv parametra	vrednost
broj generacija	1000
veličina populacije	50
veličina selekcije	25
stepen mutacije	0.7
veličina turnira	4
Samo HGA	
granica aktivacije pretrage	5
broj iteracija pretrage	3

Od oba algoritma se zahtevalo po pedeset puta pronađu rešenje za date grafove kako bi se formirale statistike vremena izvršavanja i broja iteracija, čime smo dobili i tabelu uspešnosti 6 algoritama za date grafove. Tabele 2 i 3 sadrže statistike vremena izvršavanja algoritama, dok tabele 4 i 5 sadrže statistike broja iteracija za koje je pronađeno rešenje. Svi algoritmi su bar jednom obojili svaki graf minimalnim hromatskim brojem odgovarajućeg grafa.

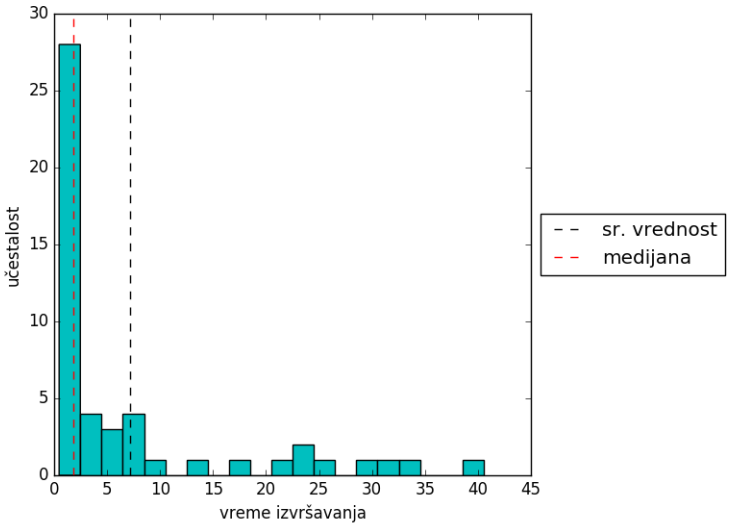
Iz naredne dve tabele 2 i 3 može se zaključiti da za date instance grafa, prosečno vreme izračunavanja genetskog algoritma je u većem broju slučajeva bolje nego prosečno vreme izračunavanja hibridnog genetskog algoritma. Hibridni genetski algoritam je pokazao bolje rezultate u slučajevima *huck.col*, *queen5_ 5.col* i *games120.col*. Može se uočiti da su najbolji rezultati u većem broju slučajeva dobijeni korišćenjem hibridnog genetskog algoritma, ali to je slučaj i sa najgorim rezultatima. Medijana je dobar pokazatelj problema koji algoritmi imaju sa lokalnim ekstremima, jer ukoliko dođe do zaglavljivanja, potrebno je dosta vremena da se algoritmi odglave 7. U obe tabele, vrednost medijane je uvek manja od srednje vrednosti.

Tabela 2: Vreme izvršavanja i njegove statistike dobijene GA.

naziv grafa	suma	sr. vrednost	medijana	najbolje	najgore
myciel4	0.911	0.018	0.017	0.009	0.033
myciel3	0.117	0.002	0.002	0.001	0.006
david	94.947	1.899	1.741	0.358	4.076
myciel6	63.174	1.264	0.907	0.405	3.799
huck	15.782	0.316	0.235	0.118	1.487
queen5_5	22.224	0.444	0.325	0.059	1.911
myciel5	5.678	0.114	0.091	0.059	0.354
jean	29.751	0.595	0.465	0.121	2.697
games120	53.083	1.062	0.971	0.602	2.749

Tabela 3: Vreme izvršavanja i njegove statistike dobijene HGA.

naziv grafa	suma	sr. vrednost	medijana	najbolje	najgore
myciel4	0.919	0.018	0.017	0.004	0.044
myciel3	0.132	0.003	0.003	0.001	0.004
david	191.579	3.832	2.034	0.097	23.899
myciel6	359.574	7.191	1.843	0.470	40.540
huck	10.350	0.207	0.148	0.046	0.996
queen5_5	13.667	0.273	0.121	0.039	1.458
myciel5	14.830	0.297	0.173	0.064	3.538
jean	61.674	1.233	0.203	0.068	15.993
games120	48.885	0.978	0.431	0.258	9.206



Slika 7: Histogram HGA za *myciel6.col* graf

Što se tiče rezultata dobijenih nad brojem generacija iz tabele 4 i tabele 3, ukazuju da je hibridni genetski algoritam bolji u svakom segmentu od genetskog algoritma, što je posledica korišćenja lokalne pretrage u hibridnom genetskom algoritmu i drugačijeg operatora mutacije koji ubrzava konvergenciju (slika 11 i slika 10). Kao i u tabelama vremena izvršavanja, srednja vrednost broja iteracija je veća od medijane osim u slučaju grafa *myciel3* gde je kod oba algoritma srednja vrednost manja od medijane.

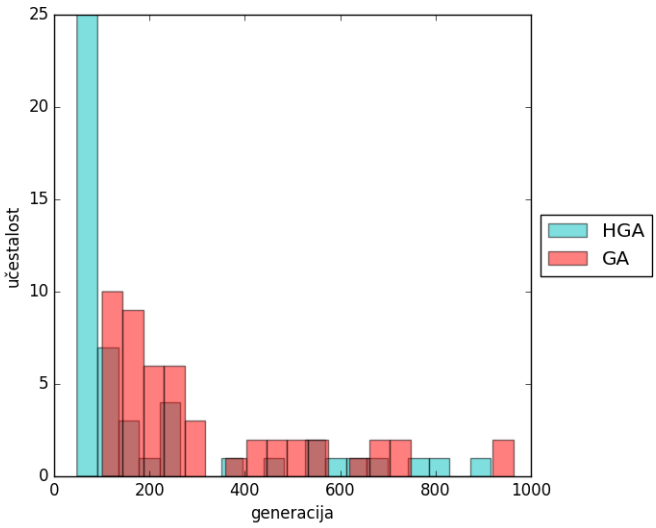
Tabela 4: Broj generacija i njegove statistike dobijene GA.

naziv grafa	suma	sr. vrednost	medijana	najbolje	najgore
myciel4	777	15.54	15.0	7	31
myciel3	84	1.68	2.0	0	4
david	22793	455.86	413.5	85	987
myciel6	16130	322.60	229.0	101	963
huck	4637	92.74	68.5	32	453
queen5_5	11628	232.56	166.0	28	965
myciel5	3049	60.98	49.5	29	171
jean	9796	195.92	152.5	38	911
games120	8752	175.04	161.0	98	459

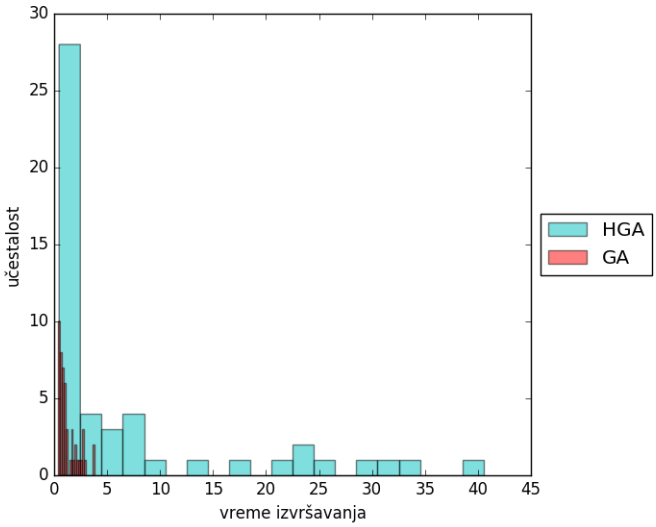
Tabela 5: Broj generacija i njegove statistike dobijene HGA.

naziv grafa	suma	sr. vrednost	medijana	najbolje	najgore
myciel4	311	6.22	6.0	2	12
myciel3	43	0.86	1.0	0	1
david	8170	163.4	96.5	20	976
myciel6	10476	209.58	91.0	48	916
huck	1117	22.34	20.0	11	59
queen5_5	6815	136.3	57.5	16	751
myciel5	1656	33.12	25.0	16	223
jean	3805	76.1	24.0	14	860
games120	2729	54.58	41.0	33	244

Na histogramima 9 i 8 mogu se videti prethodno navedene činjenice koja ukazuju na brže izvršavanje GA i većoj konvergenciji HGA.



Slika 8: Histogram GA i HGA za *myciel6* graf

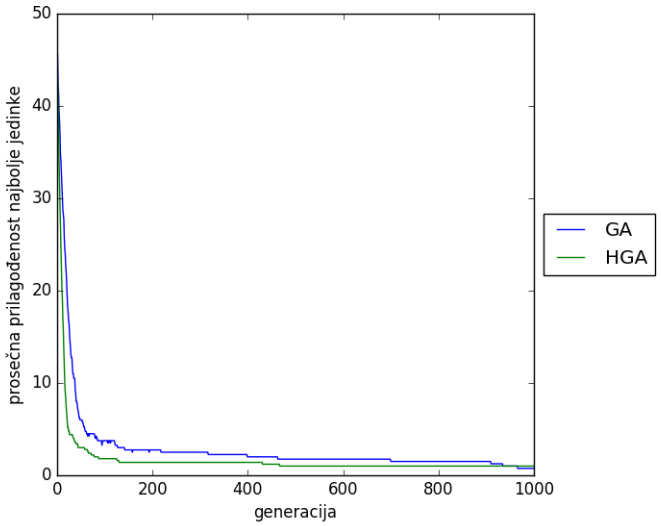


Slika 9: Histogram GA i HGA za *myciel6* graf

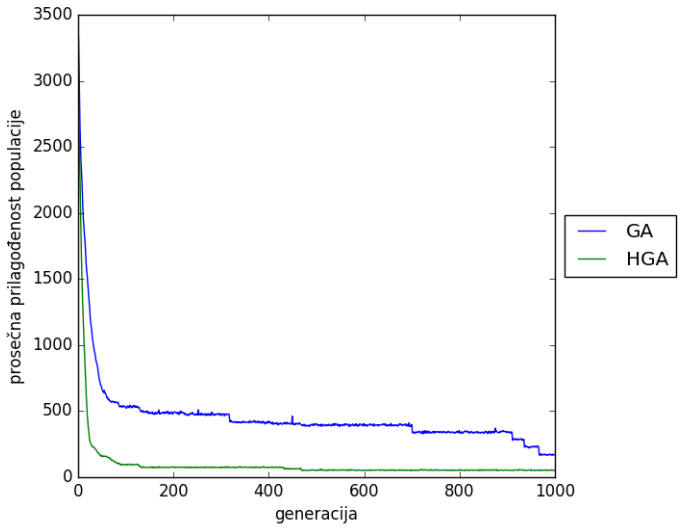
U tabeli 6 posmatra se odnos ukupnog broja pokušaja i zahtevanog broja uspeha. Može se uočiti da genetski algoritam ima veću uspešnost od hibridnog genetskog algoritma.

Tabela 6: Procenat uspešnosti algoritma.

naziv grafa	GA	HGA
myciel4.col	100.00%	100.00%
myciel3.col	100.00%	100.00%
david.col	46.73%	46.73%
myciel6.col	96.15%	66.67%
huck.col	100%	100.00%
queen5_5.col	42.37%	31.45%
myciel5.col	100.00%	100%
jean.col	100.00%	98.04%
games120.col	100.00%	100.00%



Slika 10: Poređenje prosečne prilagođenosti najbolje jedinke za *david* graf



Slika 11: Poređenje prosečne prilagođenosti populacije za *david* graf

Ukoliko se uporede srednje vrednosti oba algoritma sa već postojećim srednjim vrednostima algoritma HPGAGCP iz [4], vidi se da su dobijeni rezultati višestruko lošiji. U tabeli 7 nalaze se upoređeni rezultati. Različitost u jačini procesora računara koji su korišćeni u ovom i navedenom eksperimentu ne predstavlja bitan faktor.

Tabela 7: Poređenje sa već postojećim algoritmom

naziv grafa	GA	HGA	HPGAGCP
myciel4.col	0.018	0.018	0.006
myciel3.col	0.003	0.003	0.003
david.col	1.899	3.832	0.019
huck.col	0.316	0.207	0.015
queen5_5.col	0.444	0.273	0.031
myciel5.col	0.114	0.297	0.014
jean.col	0.595	1.233	0.015
games120.col	1.062	0.978	0.027

4 Zaključak

Genetski algoritam i hibridni genetski algoritam predstavljaju neka od mnogobrojnih rešenja za problem bojenja grafa. Jedan od razloga zbog kojeg je hibridni genetski algoritam u proseku sporiji od genetskog jeste zbog lokalne pretrage. Rezultati koji su dobijeni su vremenski lošiji. Daljim usavršavanjem načina na koji se vrši pretraga nadomestio bi se problem brzine. Jedan od mogućih načina za poboljšanje hibridnog genetskog algoritma jeste paralelizovanje lokalne pretrage i veća upotreba procesora. Zbog problema koji nastaje usled zaglavljivanja na lokalnom esktremu, može se smanjiti broj generacija, što bi umanjilo procenat uspešnosti, ali bi omogućilo stvaranje novog genetskog materijala. Ovo bi predstavljalo problem za grafove koji zahtevaju veći broj generacija da bi se došlo do željenog rezultata. Proces selekcije se može unaprediti korišćenjem složenijih operatora koji su specijalizovani za problem bojenja grafa [2].

Literatura

[1] Fathelalem F Ali, Zensho Nakao, Richard B Tan, and Yen-Wei Chen. An evolutionary approach for graph coloring. In *IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No. 99CH37028)*, volume 5, pages 527–532. IEEE, 1999.

[2] Philippe Galinier and Jin-Kao Hao. Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization*, 3:379–397, 01 1999.

[3] Lixia Han and Zhanli Han. A novel bi-objective genetic algorithm for the graph coloring problem. In *2010 second international conference on computer modeling and simulation*, volume 4, pages 3–6. IEEE, 2010.

[4] Musa M Hindi and Roman Yampolskiy. Genetic algorithm applied to the graph coloring problem. *Midwest Artificial Intelligence and Cognitive Science Conference*, page 60, 01 2012.

[5] Kiyoharu Tagawa, Kenji Kanesige, Katsumi Inoue, and Hiromasa Haneda. Distance based hybrid genetic algorithm: an application for the graph coloring problem. In *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, volume 3, pages 2325–2332. IEEE, 1999.

[6] ThanhVu H. Nguyen (tnguyen at cs.unm.edu) and Thang Bui (tbui at psu.edu). Graph Coloring Benchmark Instances. on-line at: https://turing.cs.hbg.psu.edu/txn131/graphcoloring.html.

[7] Bojan M. Vučković. *Nove kombinatorne konstrukcije u vezi saproblemima iz hromatske teorije grafova,ekstremalne teorije skupova iteorije Bulovih matrica*. PhD thesis, Univerzitet u Beogradu, Matematički fakultet, 2017.

[8] David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory of Computing*, 3(6):103–128, 2007.

A Dodatak

Za pokretanje algoritama i analizu rezultata dovoljno je pokrenuti bash skript start.sh. Broj zahtevanih uspešnih testova je hard-kodovan, tako da ukoliko želimo da nam se skripta izvršava kraće, potrebno je smanjiti broj ručno. Da biste proverili da li je rešenje ispravno, u datoteku solution.txt iskopirati rešenje ispisano na standardni izlaz i kao argument komandne linije prilikom pokretanja test.py navesti i naziv grafa čije se rešenje nalazi u datoteci.