# Minimalna suma bojenja

Seminarski rad u okviru kursa
Računarska Inteligencija
Matematički fakultet

Jovan Mirkov, Dušica Golubović

12. februar 2021.

**Sažetak**

U ovom radu će biti obrađen grafovski problem minimalne sume bojenja kao i metaheuristike za rešavanje opštih problema koje su prilagođene za dati problem.

# Sadržaj

# 1 Uvod

Neka je dat graf $G = (V, E)$ gde je $V$ skup čvorova, a $E$ skup grana. Definišemo problem: Za graf G pronaći ispravno bojenje čvorova, koristeći prirodne brojeve, tako da je suma tog bojenja najmanja od svih mogućih ispravnih bojenja. Ova minimalna suma se često u literaturi naziva *hromatska suma* (eng. *chromatic sum*). Ispravno bojenje je takvo bojenje da ako $\exists(u, v) \in E$ onda čvorovi $u$ i $v$ ne smeju biti obojeni istom bojom. [1]

# 2 Algoritam grube sile

Algoritam iscrpne pretrage ispituje sve varijante rešenja i time obezbeđuje egzaktno rešenje, ali zbog velike složenosti $O(n^n)$ koja raste eksponencijalno sa rastom ulaza je praktično neupotrebljiv za veće ulaze. Pseudokod algoritma je dat u nastavku.

```
function brute_force(graph):
  moguce_boje = [1,..,length(graph)]
  globalni_minimum = -inf
    for(svako moguce bojenje grafa od mogucih boja):
      if(ispravno_bojenje(bojenje grafa)):
        azuriraj globalni minimum
        break

return globalni_minimum
```

Listing 1: Algoritam Grube sile

# 3 Pohlepni algoritam

Pohlepni algoritmi u svakom koraku uzimaju lokalno optimalno rešenje u nadi da će nizom takvih odluka doći do globalno optimalnog rešenja. Ovom strategijom zaobilazimo različita dodatna ispitivanja rešenja i time dobijamo algoritam koji je vremenski efikasan. U konkretnoj implementaciji za posmatrani problem, za metod pretrage grafa je odabrana pretraga u širinu.

```
function pohlepni_algoritam(graf, koren):
    red = {koren}
    obidjeni = {koren}
    boje_cvorova = {niz duzine broja cvorova u grafu
     inicijalizovani na -1}
  while (red nije prazan):
      tekuci = red.pop()
      nije_zadovoljeno = true
      boja = 1
      susedi = uzmi_susedne(tekuci)
          while (nije_zadovoljeno):
              if (boja nije u susedi):
                  boje_cvorova[tekuci] = boja
              else:
                  boja++
    for (susedi od tekuceg):
      if (sused nije u obidjeni):
        obidjeni.dodaj(sused)
        red.dodaj(sused)

    return boje_cvorova
```

Listing 2: Pohlepni algoritam

# 4 Linearno programiranje

Linearno programiranje je varijanta optimizacije sa ograničenjima u kojoj pokušavamo da minimizujemo funkciju oblika

$$f(x) = c_1x_1 + ... + c_nx_n \tag{1}$$

gde je $x = [x_1, ..., x_n]^T$ vektor promenljivih a $c = [c_1, ..., c_n]$ vektor koeficijenata funkcije cilja. Ograničenja su data u obliku funkcija $g_i(x) = a_i1x_1 + ... + a_inx_n$ $i \in \{1, .., m\}$ za koje važi $g_i(x) \leq b_i$.

Funkcija cilja koju treba minimizovati za naš problem je

$$f(x) = \sum_{u=1}^{n} \sum_{k=1}^{K} kx_{uk} \tag{2}$$

, gde je $x_uk$ binarna promenljiva za koju važi $x_uk = 1$ ako je čvor $u$ obojen bojom $k$ i $k \in \{1, ..., K\}$

Ograničenja koja važe

- $\sum_{k=1}^{K} x_{uk} = 1, u \in \{1, ..n\}$
- $x_{uk} + x_{vk} \leq 1, \forall (u, v) \in E, k \in \{1, .., K\}$
- $x_{uk} \in \{0, 1\}$

[2] Prvi uslov postoji kako bismo se osigurali da svaki čvor poseduje tačno jednu boju, dok drugi kako bi se ispoštovalo ispravno bojenje, odnosno da nijedan susedni čvor ne budu obojeni istim bojama.

# 5 Simulirano kaljenje

Simulirano kaljenje(eng. *simulated annealing*) je optimizacioni algoritam zasnovan na unapređenju jednog rešenja. To je optimizacioni proces zasnovan na fizičkom procesu zagrevanja supstanci i analize njihovog ponašanja dok se temperatura smanjuje i one se hlade. Ona koristi nasumičnu strategiju pretrage koja ne prihava samo rešenje koje smanjuje funkciju cilja, već može da prihvati i rešenje koje povećava funkciju cilja.[3]

Najpre se generiše početno rešenje koje je iz dopustivog prostora rešenja. Zatim sve dok se ne dostigne kriterijum zaustavljanja(maksimalni broj iteracija) traži se rešenje u okolini trenutnog. U našem slučaju rešenje je predstavljeno nizom celobrojnih vrednosti koje predstavljaju boju čvora grafa, gde se na $i$-toj poziciji u nizu nalazi vrednost boje za $i$-ti čvor. Pronalazak rešenja u okolini se postiže tako što nasumičnom čvoru promenimo boju. Ako smo dobili nedopustivo rešenje, prelazimo u narednu iteraciju. Ako je novo rešenje bolje od trenutno najboljeg onda novodobijeno rešenje postaje trenutno najbolje. Ako nije proveravamo da li je nasumično odabran broj iz $U(0, 1)$ manji od verovatnoće pripadnosti. Verovatnoću pripadnosti dobijamo kao vrednost funkcije koja simulira opadanje temperature kroz vreme. I za to je uzeta funkcija $p(x) = 1/\sqrt{i}$ gde je $i$ trenutna iteracija.

```
function simulated_annealing(graph):
  trenutno_resenje = inicijalizuj(graph)
  najbolje_resenje = trenutno_resenje
  for (not kriterijum zaustavljanaj zadovoljen):
    novo_resenje = nadji_susedno(trenutno_resenje)
      if (not ispravno_bojenje(novo_resenje)):
```

```
        nastavi;
      if (funckija_cilja(novo_resenje) < funkcija_cilja(
    trenutno_resenje)):
            trenutno_resenje = novo_resenje
            else:
                q = uniform(0,1)
                if (q < verovatnoca_prihvatanja):
                    trenutno_resenje = novo_resenje
        if(funkcija_cilja(novo_resenje) < funkicija_cilja(
    najbolje_resenje)):
            najbolje_resenje = trenutno_resenje

return najbolje_resenje
```

Listing 3: Algoritam Simuliranog Kaljenja

# 6 Genetski algoritam

Genetski algoritam je algoritam globalne optimizacije koji pripada grupi P-metaheuristika. Za razliku od simuliranog kaljenja, one se baziraju na populaciji rešenja koja se menja kroz iteracije. Genetski algoritam koristi pojave inspirisane prirodom i zasniva se na evoluciji u kojoj samo najjače jedinke opstaju. Populaciju čine jedinke koje se drugačije nazivaju *hromozomima*. U našem problemu jedinke su, kao i u slučaju simuliranog kaljenja, predstavljene nizom vrednosti boja za dati graf. Funkcija *prilagođenosti* za datu jedinku u našem slučaju je jednaka funkciji *cilja* (funkciji koju optimizujemo tj tražimo minimum) što znači da je jedinka bolja ako joj je manja funkcija cilja. Operatori koji su specifični za ovaj algoritam su operator selekcije, operator ukrštanja i operator mutacije.

Operator *selekcije* služi kako bismo prenosili dobre osobine jedne generacije u novu generaciju. Pomoću ovog operatora mi biramo jedinke iz populacije koje će učestvovati u reprodukciji. Dve najkorišćenije strategije su turnirska i ruletska selekcija. Kod turnirske selekcije jedinke igraju turnire. Prvo se određuje veličina turnira $k$, koja predstavlja parametar genetskog algoritma. Zatim se nasumično bira $k$ jedinki iz populacije i uzima se ona jedinka koja ima najmanju funkciju cilja tj najbolju prilagođenost. U našem problemu opredelili smo se za ovaj tip selekcije.

Operator *ukrštanja* predstavlja reprodukciju jedinki. U ovom procesu učestvuju jedinke koje su odabrane u procesu selekcije, njih nazivamo roditeljima. Ukrštanje vršimo tako što nasumično odaberemo poziciju i vršimo kombinovanje. Jedno dete nasleđuje do tačke prekida jednog roditelja, a od nje drugog roditelja, a drugo dete obrnuto. Tako dobijamo dve nove jedinke koje potencijalno imaju bolju funkciju prilagođenosti od svojih roditelja i one dolaze u populaciji na mesto svojih roditelja.

Operator *mutacije* služi da sa (obično) nekom malom verovatnoćom izmenimo jedinku tako da sprečimo da one postanu suvišne slične. Ovde je operator mutacije implementiran tako da na nasumičan način bira čvor i nasumično mu menja boju. [4]

Tokom celog algoritma mi zapravo održavamo dopustivost tako što na početku inicijalizujemo populaciju sa rešenjima iz dopustivog prostora i staramo se da nakon izmena jedinki(što operatorom ukrštanja što operatorom mutacije) primenimo operator koji transformiše nedopustive jedinke u dopustive jedinke.

Kako bismo sačuvali najbolje jedinke iz populacije primenjuje se *elitizam* koji podrazumeva da jedinke sa najboljom prilagođenošću odmah prebacimo u novu populaciju.

Osim opšteg genetskog algoritma razmotrena je i jedna modifikacija tj urađen je hibridni genetski. U ovoj modifikaciji je u svakoj iteraciji na jedinku koja ima najbolju prilagođenost u populaciji primenjen algoritam simuliranog kaljenja i to u cilju bržeg konvergiranja ka optimalnom rešenju.

Opšti genetski algoritam je prikazan u nastavku.

```
function ga(graph):
  populacija = inicijalizuj_populaciju()
  nova_populacija = populacija
  while (not dostignut maksimalan broj iteracija):
      izvrsi elitizam i najbolje jedinke iz populacije prebaci u
     novu populaciju;
      roditelj1 = selekcija(populacija);
      roditelj2 = selekcija(populacija);
      dete1, dete2 = ukrstanje(roditelj1,roditelj2);
      mutacija(dete1);
      mutacija(dete2);
      stavi decu u novu populaciju;
      populacija = nova_populacija
  return najbolja_jedinka_iz_populacija;
```

Listing 4: Genetski algoritam

# 7 Optimizacija rojem čestica

Ovo je algoritam koji takođe spada u P-metaheuristike tj algoritme koji su zasnovani na populaciji i zasnovan na prirodnom procesu ponašanja ptica u svom jatu. Ono što je u genetskom jedinka ovde je čestica, dok ono što je u genetskom populacija ovde je to roj. Svaka čestica predstavlja tačku u n-dimenzionom prostoru, gde je n broj čvorova u grafu. Ažuiriranje pozicije vršimo tako što na svaku česticu dodamo vektor brzine. Ažuriranje vektora brzine vršimo na osnovu formule

$$v_i(t+1) = c_v * v_i(t) + c_p * r_p(p_i - x_i) + c_s * r_s * (g - x_i) \qquad (3)$$

gde su redom $r_p$ i $r_g$ iz $U(0,1)$ raspodele, $c_p$, $c_s$ i $c_v$ unapred zadati parametri algoritma, $p_i \in R^n$ najbolja pozicija za česticu $i$, $x_i \in R^n$ trenutna pozicija, $g \in R^n$ globalna najbolja pozicija. Opšti algoritam je dat u nastavku.

```
    /* inicijalizacija vektora i brzine */
    for svaku cesticu x_i iz roja X:
        izaberi koordinate vektora x_i;
        najbolja_pozicija_i = x_i;
        v_i = 0;
        if f(x_i) < f(najbolja_u_roju):
            najbolja_u_roju = x_i;

    while(not kriterujem zaustavljanja ispunjen):
        izaberi r_p i r_g iz U(0,1);
        v_i = c_v * v_i + c_p*r_p (p_i - x_i) + c_s*r_s * (g - x_i
    );
        x_i = x_i + v_i;
        if f(x_i) < f(p_i):
            p_i = x_i;
        if f(x_i) < f(g):


    return g;
```

Listing 5: Optimizacija rojem čestica

Takođe, i u ovom algoritmu održava se dopustivost rešenja pomoću operatora koji rešenje i prostora nedopustivih rešenja prebacuje u prostor dopustivih rešenja.

# 8 Rezultati

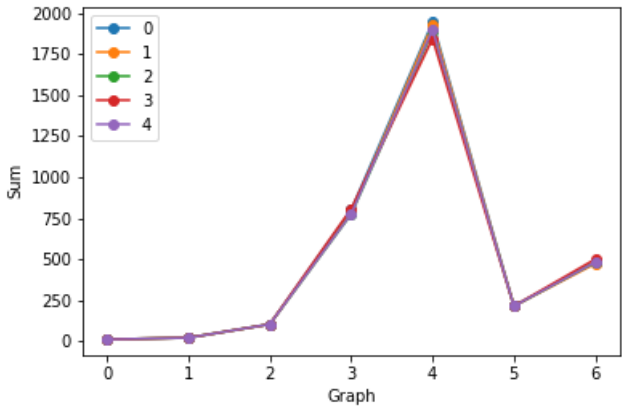
Za testiranje algoritama korišćeni su grafovi predstavljeni u DIMACS formatu preuzeti sa sajta Carnegie Mellon univerziteta u Pitsburgu.[5] Svaki graf za testiranje je odabran da ima određeno svojstvo:

1. Graf napravljen tako da sa minimalnom sumom nema minimalni hromatski broj
2. Graf sa malim brojem čvorova
3. Graf sa srednjim brojem čvorova
4. Graf sa manjim hromatskim brojem u odnosu na broj čvorova
5. Graf sa velikim brojem čvorova i velikim hromatskim brojem
6. Graf sa velikim brojem čvorova i malim hromatskim brojem
7. Graf sa velikim brojem čvorova i malim brojem grana

## 8.1 Pohlepni algoritam

Nakon završenog konkretnog algoritma rešenje je poboljšano zamenom vrednosti boja tako da čvorovima iste boje kojih ima najviše pridužimo boju najmanje vrednosti, i tako redom za cvorove manje brojnosti, tj. bojama k-tog ranga pridruzimo cvorovima k-te mnogobrojnosti.

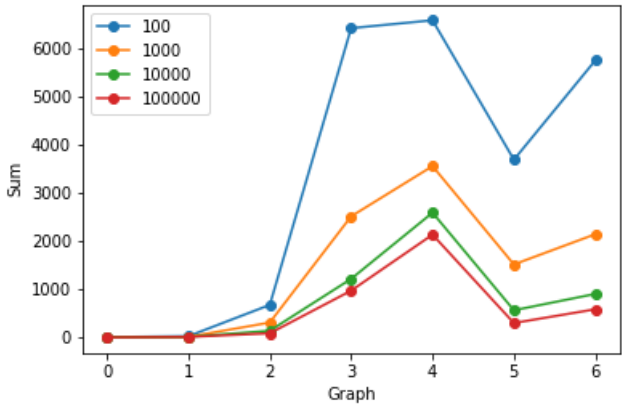
Jedini parametar u ovom algoritmu je početni čvor pretrage. Gledajući rezultate možemo videti da ćemo dobiti različite sume i bojanja u zavisnosti od kog čvora započnemo pretragu i da su te razlike primetne samo u većim grafovima.




Slika 1: Rezultati pohlepnog algoritma

## 8.2 Simulirano kaljenje

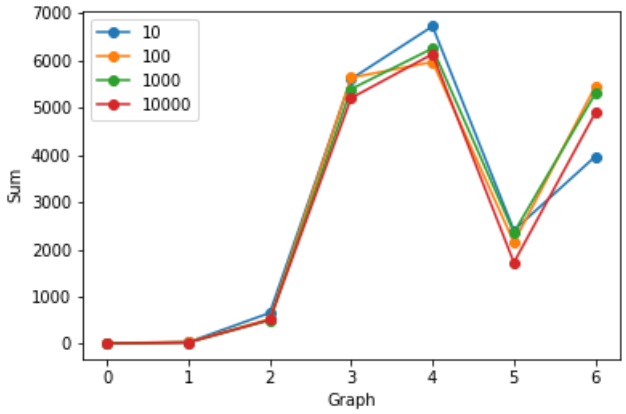
Parametar ovog algoritma je samo broj iteracija. Grubim odabirom vrednosti ovog parametra uočavamo da se suma minimalno menja između vrednosti 10000 i 100000 iteracija i time pretpostavljamo da se daljim povećanjem broja iteracija suma neće značajno smanjivati. Grubim oda-




Slika 2: Rezultati simuliranog kaljenja

birom vrednosti iteracija kao parametra primećujemo da sume ne opadaju sa rastom broja iteracija i da su uglavnom u bliskim okolinama. Za testiranje po broju čestica izabraćemo slučaj od 100 iteracija radi udobnosti.

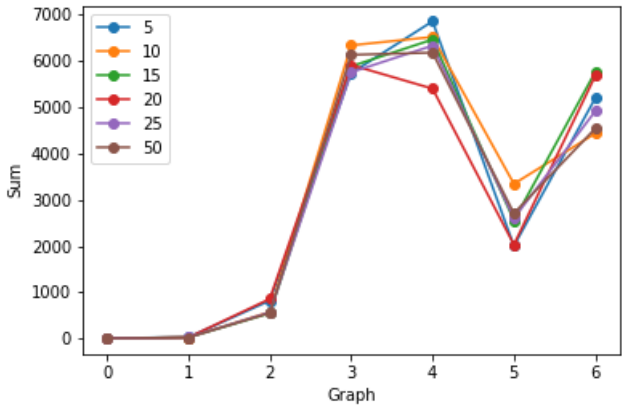
## 8.3 Optimizacija rojem čestica




Slika 3: Rezultati PSO prema broju iteracija

Za odabrani broj iteracija iz prethodnog testiranja najbolje se pokazala varijanta algoritma sa 20 čestica. Primetimo da je tako odabrana

kombinacija parametara znatno uticala na četvrti graf u odnosu na ostale.




Slika 4: Rezultati PSO prema broju čestica

## 8.4 Genetski algoritam

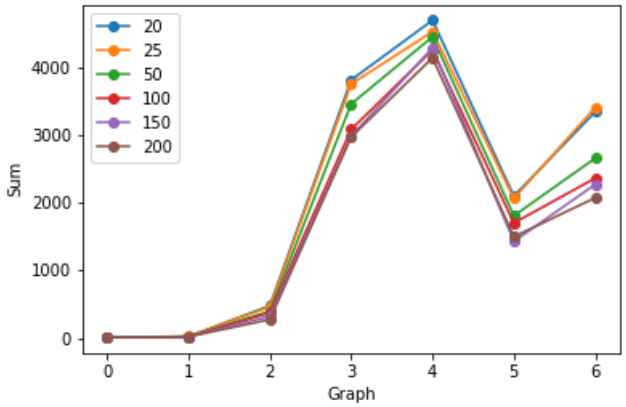
Primenom genetskog algoritma na različite veličine populacije između 10 i 250, razlika u sumama izmedju te dve vrednosti je na svim grafovima približna redu veličine 100. U slučaju odsustva elitizma hibridna varijanta algoritma dostiže znatno bolje rezultate. Ova zapažanja su istaknuta bez vizualizacije radi bolje preglednosti.

Prilikom porasta broja iteracija algoritam očekivano daje sve bolja rešenja. Hibridizacija algoritma simuliranim kaljenjem smanjuje sumu većih grafova za vrednosti približne broju reda veličine 800. Daljim pokretanjem algoritma (bez hibridne varijante) za do 5000 iteracija dolazimo do rešenja koji u slučajevima velikih grafova smanjuju sume za vrednosti reda veličine 1000.

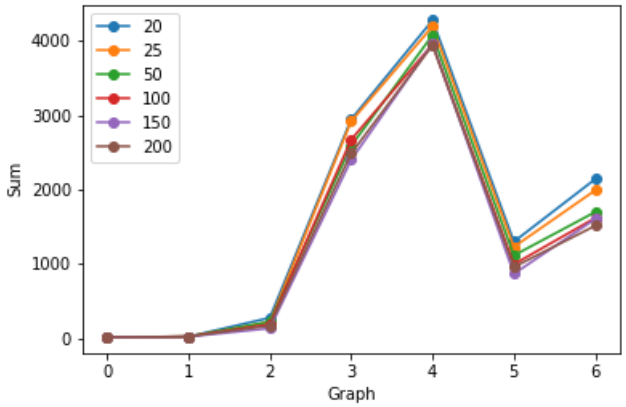
## 8.5 Objedinjeni rezultati

U sledećoj tabeli predstavljeni su objedinjeni rezultati sa najboljim dobijenim rešenjima za svaki algoritam i njihovim vremenom izvršavanja.

| Graf | Gr | SA | PSO | GA |
|---|---|---|---|---|
| 0 | 12\|0.00004s | 11\|0.69s | 14\|0.04s | 11\|19.9s |
| 1 | 22\|0.00004s | 21\|0.97s | 32\|0.09s | 21\|27.69s |
| 2 | 104\|0.0003s | 96\|5.37s | 862\|0.58s | 168\|258.5s |
| 3 | 800\|0.005s | 982\|40.2s | 5897\|3.09s | 1990\|4480.63s |
| 4 | 1944\|0.013s | 2145\|114.2s | 5386\|6.88s | 3523\|12520.1s |
| 5 | 215\|0.002s | 309\|12.7s | 2031\|2.04s | 778\|1245.6s |
| 6 | 483\|0.003s | 595\|19.3s | 5662\|2.59s | 1385\|2053.1s |




Slika 5: Rezultati genetskog algoritma prema broju iteracija




Slika 6: Rezultati hibridnog genetskog algoritma prema broju iteracija

# 9 Zaključak

Prema predstavljenim rezultatima može se zaključiti da se najbolje pokazao algoritam simuliranog kaljena i da se korišćenjem optimizacije rojem čestica verovatno dolazi do lokalnog minimuma. Sva izračunavanja su dobijena u skladu sa razumnim vremenom izvršavanja koje možemo priuštiti pri pokretanju, tako da se dobijeni rezultati mogu koristiti za proučavanje ponašanja konkretnog problema minimalne sume bojenja nad datim algoritmima i nikako u svrhu dobijanja uvida u optimalno rešenje algoritma za velike grafove.

# Literatura


[1] Corinne Lucet Yu Li Clément Lecat, Chu-Min Li. Exact methods for the minimum sum coloring problem. 1, 2016.

[2] Jin-Kao Hao Yang Wang. Solving the minimum sum coloring problem via quadratic programming.

[3] Andries P. Engelbrecht. *Computational Intelligence. An Introduction.* 2007.

[4] dr Mladen Nikolić dr Predrag Janičić. *Veštačka inteligencija.* 2007.

[5] Grafovi preuzeti sa sajta Carnegie Mellon univerziteta u Pitsburgu. dostupno na: https://mat.gsia.cmu.edu/COLOR/instances.html.