Rešavanje problema maksimalnog balansiranog particionisanja

Jovana Bošković, Marina Pilipović
jboskovic97@gmail.com, beka.pilipovic@gmail.com

September 2020

Sažetak

U ovom seminarskom radu biće predstavljen grafovski problem maksimalnog balansiranog particionisanja, kao i neke prilagođene metaheuristike široke upotrebe sa kojima su se autori ovog teksta susreli na fakultetu u okviru kursa Računarksa inteligencija. Na kraju će biti prikazani komparativni rezultati svake metode dobijeni eksperimentalnim putem.
Ključne reči: mnbp, heuristika, optimizacija

1 Uvod

Mnogi stvarni problemi mogu se rešiti korišćenjem matematičkih struktura i njihovih svojstva. Da bi se kreirao matematički model, svaki element problema mora biti predstavljen kao neka matematička struktura i odnosi između elemenata moraju da odgovaraju relacijama između tih matematičkih struktura.

Za mnoge probleme kombinatorne prirode često se koriste grafovi i razne tehnike uvedene u teoriju grafova. Ovi problemi zahtevaju rad sa složenim i velikim grafovima, a korišćen pristup je podela grafa na manje podgrafe. Ako je problem predstavljen kao povezan graf, particioniranje prirodno zahteva da su i podgrafi povezani. U situacijama kada težinski povezan graf mora biti razdvojen na dve manje komponenente, logičan zahtev je tražiti da oba podgrafa jesu povezana i da zbir težina u obe komponenete bude što je moguće sličniji.

Problem opisan u prethodnom pasusu naziva se **problem maksimalno balansiranog particionisanja** (engl. maximally balanced connected partition problem, skraćeno MBCP). Dokazano je da je problem NP težak, tako da ne postoji alogritam koji ga rešava u polinomnom vremenu.

U nastavku rada predstavljeni su algoritmi koji rešavaju problem maksimalnog balansiranog particionisanja pomoću određenih metaheuristika.

2 Gruba sila

Algoritam grube sile (engl. brute force) se obično primenjuje na optimizacione probleme. Postupak za rešavanje ovih problema uključuje pre-

gledanje svih varijanti. Algoritmi grube sile su korisni za vrlo male ulaze, za velike ulaze mora se naći efikasnije rešenje.

Za rešavanje našeg problema, algoritam kreira i izračunava vrednost svakog mogućeg rešenja, tražeći najbolje. Ovaj način resavanja nam garantuje da ce optimalno rešenje biti pronađeno kad algoritam završi sa radom. Problem kod ovog pristupa je što zahteva dosta memorije, pa za velike ulaze nije upotrebljiv.

Za dati graf $G = (V, E)$, potencijalno rešenje maksimalnog balansiranog particionisanja može biti predstavljeno binarnim nizom dužine $|V|$. Elementi niza odgovaraju čvorovima, a vrednosti određuju kom podgrafu dati čvorovi pripadaju. Važi uslov: *ako* $x_i = 1$ *onda* $x_i \in V_1$, *ako* $x_i = 0$ *onda* $x_i \in V_2$. Broj mogućih rešenja predstavljenih na ovaj način je $2^{|V|}$. Zbog velike memorijske složenosti, ovaj algoritam nije upotrebljiv za velike grafove.

```
// best             - trenutno najbolje re enje (zbir te ina dva
      podgrafa najsli niji)
// V1_best, V2_best - najbolja podela grafa na dva podgrafa (
     podgrafovi predstavljeni u obliku skupa  vorova )
// d                - apsolutna razlika u te inama dve komponente
// ulaz: graf sa datim te inama  vorova
// izlaz: dve povezane komponente sa najsli nijim te inama
      vorova
// Inicijalizacija: best=w(V), V1_best=V, V2_best=()

Konstruisi sve particije (V1, V2) za cvorove V
do:
    if (oba podgrafa povezana):
        (1) izra unaj te ine w(V1) i w(V2)
        (2) d = |V1- V2|
        (3) if (d < best):
                best = d
                V1_best = V1
                V2_best = V2
while (dok nisu obra eni svi (V1,V2))

return  best, V1_best, V2_best
```

Listing 1: Pseudokod algoritma grube sile

Na početku algoritma se *trenutno najbolje resenje* inicijalizuje na maksimalnu moguću vrednost u ovom slućaju w(V), a podgrafovi se postavljaju na skup koji sadrži sve čvorove i prazan skup. Zatim se za svaku moguću particiju proverava da li su podgrafovi povezani i u slučaju da jesu računa se da li je nova vrednost manja od trenutne najbolje, ukoliko jeste ažurira se trenutno najbolje rešenje. Algoritam se završava kada se ispitaju sva moguća rešenja i promenljiva best sadrži najbolje rešenje.

3 Metaheuristike

Metaheuristike možemo definisati kao skup algoritamskih koncepata koji koristimo za definisanje heurističkih metoda primenljivih na širok skup problema.To su metode koje nisu dizajnirane za rešavanje konkretnog problema, već za rešavanje čitave klase problema, ali iako su metaheuristike opšte metode, mogu biti prilagođene specifičnom problemu.

U nastavku teksta biće prikazano rešavanje problema maksimalnog balansiranog particionisanja na pohlepnom, genetskom i algoritmu simuliranog kaljanja. Algoritmi za predstavljanje rešenja koriste niz nula i jedinica, kao što je objašnjeno u delu Gruba sila. Dodatno, za svako rešenje se čuva fitnes, mera koja određuje koliko je rešenje dobro. Ova mera uzima u obrzir razliku suma čvorova podgrafova i da li su podgrafovi povezani. Radi veće optimizacije mere fitnes za obrađena rešenja se čuvaju, tako da ukoliko je vrednost bila izračunata, neće se opet računati. Više o meri fitnes u daljem tekstu.

3.1 Pohlepni algoritam

Algoritam kod koga se u svakom koraku uzima *lokalno optimalno* rešenje i koji garantuje da će takvi izbori na kraju dovesti do *globalno optimalnog* rešenja naziva se pohlepni ili gramziv algoritam (engl. greedy algorithm). Pohlepni algoritam ne vrše ispitivanje različitih slučajeva niti iscrpnu pretragu i stoga je po pravilu veoma efikasan.

U slucaju MBCP, predstavljeno rešenje pomoću pohlepnog algoritma se bazira na block-balance algoritmu.

Neka je $G = (V, E)$ dati graf. Algoritam započinjemo dvema particijama (V_1, V_2) takve da V_1 sadrži jedan čvor najveće težine, a V_2 ostale čvorove. Tokom iteracionog procesa algoritam traži čvor iz skupa V_2 takav da su obe particije $V_1 \cup \{u\}$ i $V_2 \setminus \{u\}$ povezane. Od svih mogućih čvorova, bira se čvor sa minimalnom težinom. Algoritam se zaustavlja kada dodavanjem čvora skupu V_1 ne smanjuje razliku izmedju suma čvorova particija.

Ovako implementiran algoritam ne vraća uvek najoptimalnije rešenje, ali je dosta efikasan.

```
1  Sortirati cvorove skupa V opadajuce po tezinama $w(v_1)  \geq  w(
       v_2)  \geq  ...  \geq  w(v_n) $
2
3  Postavi $V_1=\{v_1\}, V_2 \setminus \{u_1\}$
4
5  Ako je $w(V_1) \geq \frac{1}{2} w(V)$ onda skoci na korak 7, inace
       skoci na korak 4
6
7  Iz skupa $V_0=\{u \in V_2 | (V_1 \cup \{u\}, V_2 \setminus \{u\})$
       je povezana particija graha G \}
8
9  Izabrati $v \in V_0$ takav da  $w(u)=min_{v \in V_0}w(u)$, skoci
       na korak 3
10
11 Ako je $w(u) < w(V) - 2w(V_1)$ \\
12     onda $V_1 = V_1 \cup \{u\}, V_2 = V_2 \setminus \{u\} $ skoci
       na korak 3
13     inace skoci na korak 7
14
15 return $(V_1,V_2)$
```

Listing 2: Pseudokod pohlepnog algoritma

3.2 Simulirano kaljenje

Algoritam simuliranog kaljenja (engl. simulated annealing, skraćeno SA) je zasnovan na procesu kaljenja čelika, čiji je cilj oplemenjivanje me-

tala tako da on postane čvršći. Zasniva se na poboljšanju vrednosti jednog rešenja. Na početku se proizvoljno ili na neki drugi način generiše početno rešenje i izračunava vrednost njegove funkcije cilja. Vrednost najboljeg rešenja se najpre inicijalizuje na vrednost početnog. Zatim se algoritam ponavlja kroz nekoliko iteracija. U svakom koraku se razmatra rešenje u okolini trenutnog. Ukoliko vrednost funkcije cilja bolja od prethodnog onda trenutno rešenje ažurira, u suprotnom se upoređuju vrednosti unapred definisane funkcije p i proizvoljno izabrane vrednosti q iz intervala (0,1). Ako je $p>q$, trenutno rešenje se ažurira novoizabranim. Algoritam se se ponavlja dok nije ispunjen kriterijum zaustavljanja.

```
1. Generisati pocetno resenje s

2. Inicijalizovati vrednost najboljeg resenja $f^{*} = f(s)$

3. Dok nije ispunjen kriterijum zaustavljanja ponavaljti sledece
    korake:

    Izabrati proizvoljno resenje $s'$ u okolini resenja s

    Ako je $f(s') < f(s)$ onda $s = s'$

    Ako je $f(s') \geq f(s)$ onda:

        Azurirati vrednost opadajuce funkcije p

        Izabrati proizvoljno $q \in [0,1]$

        Ako je $p>q$ onda $s= s^'$

    Ako je $f( s')<f^*$ onda $f^*=f(s')$

Ispisati vrednost resenja $f^*$
```

Početno rešenje se generiše slučajnim rasporedom nula i jedinica. Za meru kvaliteta rešenja se koristi funckija fitnes, dok se za traženje okoline rešenja koristi slučajan odabir elementa niza koji će konvertovati vrednost

3.3 Genetski algoritam

Genetski algoritmi su bazirani na Darvinovoj teoriji evolucije, u kojoj unutar jedne populacije najčešće opstaju najbolje prilagođene jedinke. Nova generacija jedinki se dobija ukrštanjem jedinki iz prethodne generacije, pri čemu se s vremena na vreme kod nekih jedinki mogu pojaviti mutacije na nekim genima. Očekivanje je da će se smenom generacija dobijati populacija jedinki koja je bolje prilagođena okruženju. Nakon inicijalizacije početne populacije, nove generacije jedinki se dobijaju uzastopnom primenom genetskih operatora. Smena generacija se vrši sve dok ne bude ispunjen kriterijum zaustavljanja. Početna generacija može biti generisana proizvoljno ili primenom neke heuristike. Najčešći genetski operatori su ukrštanje i mutacija.

Jedinke su predstavljene nizom nula i jedinica i svaka jedinka ima dodeljen fitnes. Fitnes se sastoji iz dva dela, razlike suma čvorova dva podgrafa i kaznenih poena ukoliko podgrafovi nisu povezani. Kazneni poeni se računaju formulom $Penalty(ind) = (ncc_2(ind) - 1) * max_1 + (ncc_1(ind) -$

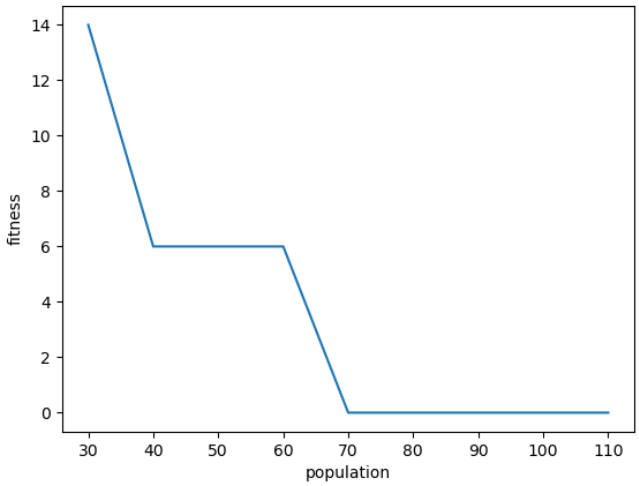
1) $* max_2$. Gde nnc(ind) predstavlja broj komponenti povezanosti pografova, a max najveću vrednost čvor.

U algoritmu je korišćena turnirska selekcija, ukrštanje sa jednom tačkom prekida i prosta mutacija, a prilikom smene generacija, čuvaju se elitne jedinke. Uslov zaustavljanja je maksimalna broj iteracija ili ukoliko se pronađe rešenje sa najmanjim fitnesom nula.

4 Rezultati istraživanja

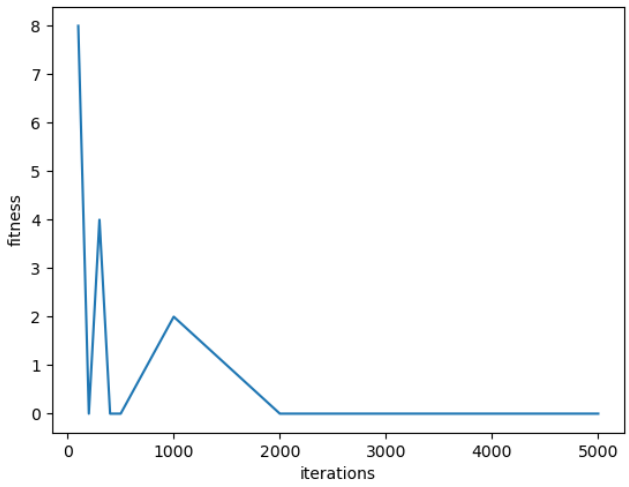
U ovom odeljku ćemo se osvrnuti na sve prethodno implementirane algoritme, kao i njihova poredjenja međusobno, menjanje njihovih ulaznih parametara i odnos između njih.

Na svim implementiranim algoritmima smo pokernuli četiri različita test primer, koja su različitih obima. Broj čvorova je u velikoj meri uticao na specifikacije i efikasnost algoritama. Prilikomo pokretanja primera sa najvećim brojem čvorova na algoritmu grube sile došlo je do pucanja programa zbog ograničenog vremena i memorije. Nije moguće izvršiti algoritam grube sile za velike ulaze.



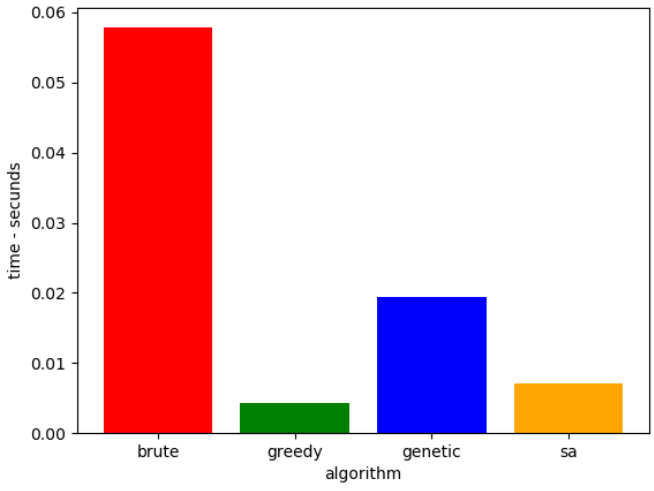
Slika 1: Poređenje veličine populacija u odnosu na rezultat

Genetski algoritam zahteva dosta ulazih parametara, kao što su veličina populacije, veroratnoća mutacije, verovatnoća ukrštanja itd. Slika 1 prikazuje kako veličina populacije utiče na rezultat odnosno na promenljivu *fitness* koja predstavlja balans između dva podgrafa. Što je vrednost promenljive *fitness* manja to su čvorovi u podrafima bolje rapoređeni. Vidimo da rezultat lošiji kako se veličina populacije smanjuje. Najveće promene se dešavaju kad broj pupulacije padne ispod 70.



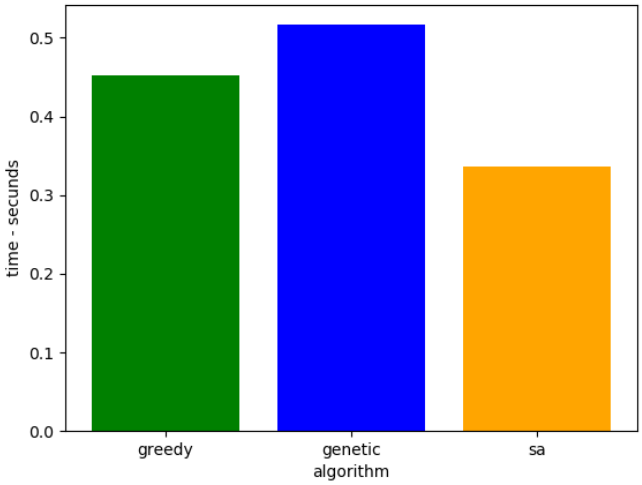
Slika 2: Poređenje broja iteracija sa rezultatom algorutmu simulirano kaljanje

Broj iteracija u velikom broj algoritama utiče na rezultat. Na slici 2 možemo videti da algoritam simulano kaljanje ne daje zasigurno najbolji rezultat ukoliko ga pozovemo za broj iteracija manji od 2000.



Slika 3: Dužina izvršavanja algoritama u sekundama za primer 3

Slika 3 nam prikazuje vreme izvršavanja u sekundama u odnosu na algoritme. Algoritam grube sile iziskuje mnogo više vremena u odnosu na sve ostale algoritme, što je i logično pošto on ispituje sve moguće opcije. Primer 4 koji ima najveći broj čvorova nam daje približne rezultate za sva tri algoritama koji su mogli da se izvrše u realnom vremenu [Slika 4].



Slika 4: Dužina izvršavanja algoritama u sekundama za primer 4