Maximum clique korišćenjem Tabu pretrage

Projekat iz Računarske inteligencije
Matematički fakultet
Univerzitet u Beogradu

Jovan Đorđević
mi17164@alas.matf.bg.ac.rs
Tamara Tomić
mi17122@alas.matf.bg.ac.rs

Maj 2022

Sadržaj

1 Opis problema

Neka je G = (V, E) neusmeren graf, gde je $V = \{1, ..., n\}$ skup čvorova i $E \subset V$ x V skup grana grafa. Klik C grafa G je podskup V takav da su svaka dva čvora povezana , odnosno da važi $\forall u, v \in C, \{u, v\} \in E$. Klik C je maksimalan ako nije sadržan ni u jednom drugom kliku, odnosno ako je njegova kardinalnost veća od kardinalnosti svih drugih klikova. Pronalaženje maksimalnog klika je NP-težak problem, koji ima primenu u rešavanju mnogih svakodnevnih problema - teorija klasifikacije, dijagnostika kvara, biološke analize, analize klastera, izbor projekta, itd. Problem pronalaska maksimalnog klika je ekvivalentan problemu maksimalnog nezavisnog skupa i usko povezan sa problemom bojenja grafa. Lep pregled raznih metoda za rešavanje ovog problema i eksperimentalnih rezultata može se naći u [2].

2 Ulazni podaci i struktura Graf

Graf smo implementirali preko matrice povezanosti. Matrica je dimenzije N x N gde je N broj čvorova + 1. Ako u grafu postoji grana izmedju čvorova 'i' i 'j', i,j $\in V$, vrednost matrice m[i][j] ce biti jednaka 1, u suprotnom 0, tj:

$$adjacencyMatrix[i][j] = \begin{cases} 1, & \text{if } (i,j) \in E \\ 0, & \text{if } (i,j) \notin E \end{cases}$$

Test instance na kojima će biti testiran naš algoritam mogu se preuzeti sa sledeće adrese: https://networkrepository.com/dimacs.php.
Kompletan naš kod, kao i rezultati mogu se pogledati na sledećem linku:
https://github.com/JovanDjordjevic/RI$_m aximum_c lique_t abu_s earch$

Zadati grafovi se tretiraju kao neusmereni grafovi bez težina. U prvom redu svakog fajla navedene su informacije o broju čvorova u grafu, a u svakom narednom redu navedena je jedna grana. Najmanji graf u ovoj kolekciji ima 28 čvorova i 420 grana, dok najveći imaju oko 3 do 4 hiljade čvorova i oko 4 do 6 miliona grana. Ovaj set test instanci je izabran kako bi što je moguće bliže uporedili naše rezultate sa rezultatom iz rada [1]. Bitno je prilikom analize rezultata imati na umu razlike u implementaciji.

3 Tabu pretraga - uopšteno

Tabu pretraga je metaheuristika koja spada u algoritme lokalne pretrage i koristi se za rešavanje problema kombinatorne optimizacije. Cilj ove vrste pretrage je da izbegne zaglavljivanje u lokalnim minimumima, što se postiže korišćenjem tabua (zabranjenih stanja). Održava se tzv. tabu lista, tj. lista stanja u kojima je pretraga već bila i u koja se više neće vraćati. Dozvoljavaju se potezi koji pogoršavaju vrednost funkcije, pod uslovom da nema boljih dozvoljenih poteza. Ova dva pristupa omogućavaju da se napusti lokalni mininum. Detaljnije o tabu pretrazi i njenim primenama može se naći u radu [3].

4 Brute force algoritam

4.1 Implementacija

Za implemntaciju brute force algoritma koristili smo rekurzivni pristup pronalaska maksimalne veličine klike. Funkcija *isClique* proverava da li je trenutni podskup koji se nalazi u *store* klik veličine cliqueSize (cliqueSize je argument funkcije). U dvostrukoj for petlji proveravamo da li je bilo koja vrednost u matrici povezanosti grafa jednaka 0 za čvorove koji trenutno čine podskup. Kada naiđemo na prvu nulu, možemo da prekinemo i vratimo *False*, jer ukoliko bilo koja dva čvora nisu povezana, podskup sigurno nije klik. Ukoliko izađemo iz petlje i nismo vratili False, znači da su svi čvorovi povezani i da podskup jeste klik dužine cliqueSize, pa vraćamo *True*.

Algorithm 1 Funkcija isClique

Require: Broj $cliqueSize \geq 0$
Ensure: True, ako je podskup klik veličine cliqueSize, False inače
  for $i \leftarrow 1$ to $cliqueSize$ **do**
    for $j \leftarrow i + 1$ to $cliqueSize$ **do**
      if *ne postoji grana izmedju čvora store[i] i store[j]* **then**
        return False
  return True

Funkcija *maxCliquesBF* prima kao argumente graf, čvor od kog počinjemo proveru i trenutno poznatu maksimalnu dužinu klika. U for petlji proveramo da li možemo da dodamo bilo koji drugi čvor, počevši od početnog + 1 (newNode), a da podskup i dalje bude klik. Ukoliko možemo upoređujemo da li je taj klik veći od trenutno poznatog najvećeg klika (dužine newSize). Zatim, rekurzivno pozivamo funkciju *maxCliquesBF* za argumente *newNode* i *newSize + 1* i upoređujemo vrednost koju vrati sa trenutnom maksimalnom dužinom.

Algorithm 2 Funkcija maxCliquesBF

Require: Čvor od kog krećemo proveru, trenutna maksimalna dužina klika
Ensure: Nova maksimalna dužina klika
  $maxcliquegets0$
  for $newNode \leftarrow startNode + 1$ to $ukupanbrojčvorova$ **do**
    $store[newSize] \leftarrow newNode$
    if *jeste klik duzine newSize+1* **then**
      $maxclique \leftarrow max(maxclique, newSize)$
      $maxclique \leftarrow max(maxclique,$ *maxCliquesBF(newNode, newSize+1))*
  return maxclique

5 AMTS algoritam

Bruteforce algoritam postaje neupotrebljiv za veće instance (videti rezultate u 5.6), pa se do rezultata mora doći na drugačiji način. Mi smo se opredelili za pristup koji koristi tabu pretragu: AMTS - skraćeno od *Adaptive multistart tabu search*. Ideja je da se pokreće tabu pretraga više puta iz različitih početnih stanja i da se restartuje kada se pretraga zaglavi.

5.1 Funkcija evaluacije

Pomoću funkcije evaluacije f(S) proveravamo da li je S validan kandidat.

$$f(S) = \sum_{u,v \in S} e_{u,v}$$

$$e_{u,v} = \begin{cases} 1, & \text{if } (u,v) \in E \\ 0, & \text{if } (u,v) \notin E \end{cases}$$

Ukoliko je

$$f(S) = k * (k-1)/2 \tag{1}$$

to znači da su svaka dva čvora iz S povezana i da je S validan k-klik. U suprotnom, S nije validan kandidat. Cilj nam je da pronađemo skup S koji dostiže maksimalnu vrednost funkcije f takvu da važi jednakost (1).

5.2 Konstrukcija inicijalnog resenja

Prvi korak pri implementaciji algoritma je generisanje inicijalnog rešenja od kog krećemo pretragu. Polazimo od praznog skupa S i u svakom koraku dodajemo čvor *nodeToAdd*, koji se nalazi u skupu *notInS*, a ne nalazi se u skupu *S*. U strukturi podataka *frequencyGroups*, koju čuvamo kao heš mapu, ključevi su frekvencije, a vrednosti su skup čvorova koji imaju tu frekvenciju. Za prvi čvor koji ubacujemo u *S* uzima se nasumičan čvor sa najmanjom frekvencijom pojavljivanja. Kada ga ubacimo u skup *S*, izbacujemo ga iz skupa *notInS* i povećavamo mu frekvenciju pojavljivanja za 1. U strukturi podataka *degreeSets*, koju takođe čuvamo kao heš mapu, ključevi su stepeni povezanosti sa *S*, a vrednosti su skup čvorova koji ima taj stepen. Iz skupa čvorova sa najvećim stepenom biraju se naredni kandidati za ubacivanje u S. Prolaskom kroz sve čvorove u vektor *degreesTowards* na pozijicu *i* upisujemo stepen povezanosti čvora *i* (koji se ne nalazi u S) ka S. Sada kada imamo ove podatke, ideja je da biramo kandidate tako da imaju najviše grana ka S i da imaju najmanju frekvenciju pojavljivanja u S. Svaki put kada naiđemo na manju frekvenciju resetujemo skup kandidata. Svaki put kada nekog kandidata (*nodeToAdd*) ubacimo u skup *S*, izbacujemo ga iz skupa *notInS* i uvećavamo frekvenciju pojavljivanja za 1. Takođe, stepen povezanosti sa S čvora *nodeToAdd* sada treba postaviti na 0 i izbaciti ga iz skupa čvorova koji su imali najveći stepen povezanosti. Ukoliko je ovaj kandidat bio jedini sa *maxDegree* stepenom povezanosti, treba naći novi najveći stepen povezanosti. Nakon toga, potrebno je obići sve čvorove koji nisu u S i ažurirati njihov stepen povezanosti sa novim skupom *S*. Ukoliko su bili povezani sa kandidatom koji smo dodali u skup *S* pomeraju se u drugi skup stepeni. Po potrebi treba ažurirati i promenljivu *maxDegree*.

5.3 Ograničena okolina

Ideja je da izlistamo sve moguće kombinacije poteza, pri čemu potez predstavlja zamenu čvora $u \in S$, čvorovom $v \notin S$. U skupu A nalaze se čvorovi koji su kandidati za izbacivanje iz skupa S. Želimo da izbacimo čvor sa najmanjim stepenom povezanosti sa ostalim čvorovima iz S (minInS), a koji je dozvoljeno izbaciti zbog tabu uslova (poglavlje 5.4). U skupu B nalaze se čvorovi koji su kandidati za ubacivanje u skup S. Želimo da ubacimo čvor koji ima najveći stepen povezanosti ka S (maxOutS), a koji je dozvoljeno ubaciti zbog tabu uslova. Da bismo od kandidata S dobili kandidat S' vršimo zamenu jednog čvora iz A sa jednim čvorom iz B. Sve moguće ovakve zamene cine našu ograničenu okolinu.
Vektor koji sadrži podatke o ograničenoj okolini pored čvorova i i j koje treba izbaciti odnosno ubaciti ima i parametar *delta*, koje predstvalja promenu vrednosti funkcije evaluacije i jednak je razlici stepena povezanosti čvora koji izbacujemo i čvora koji ubacujemo ($maxOutS$ - $minInS$).

5.4 Tabu lista i zabrana

Tabu lista je struktura podataka koja nam služi za čuvanje nedozvoljenih (tabu) poteza. Potez zamene (u, v) o kome smo pričali u poglavlju 5.3 je tabu ako se barem jedan od čvorova u i v nalazi u tabu listi. Kada neki čvor u izbacimo iz kandidata S on postaje tabu potez za narednih Tu iteracija (zvanih tabu zabrana, eng. *tabu tenure*), za vreme kojih on ne može biti ponovo dodat u S. Slično, kada neki čvor v ubacimo u S on postaje tabu potez za narednih Tv iteracija tokom koji ne može biti uklonjen iz rešenja. Po uzoru na [1] vrednosti Tu i Tv, koje zavise od funkcije evaluacije f, računamo na sledeći nacin:

$$Tu = l + Random(C) \tag{2}$$

$$Tv = 0.6 + l1 + Random(0.6 * C) \tag{3}$$

gde je:

$$l1 = k * (k-1)/2 - f(S), l = min\{l1, 10\}, C = max\{\lfloor k/40 \rfloor, 6\} \tag{4}$$

Prvi deo tabu zabrane Tu proizilazi iz toga da rešenja sa manjom funkcijom evaluacije treba da imaju dužu zabranu kako bismo pobegli iz zamke lokalnog optimuma. S obzirom na to da tačna vrednost tabu zabrane nije poznata ostali delovi su random pretpostavke. Primećujemo takođe da je

$Tu > Tv$. To se može objasniti time da je zabrana izbacivanja čvora iz trenutnog rešenja S mnogo restriktivnija od zabrane ubacivanja čvora koji nije u S, pošto generalno ima mnogo manje čvorova u S nego van S. Za implementiranje tabu liste korisimo vektor *tabulist*. Svaki element *tabulist[i]* čuva vrednost Ti + I, gde je I trenutni broj iteracija, a Ti je tabu zabrana za čvor i. Na ovaj način lako je odrediti da li je čvor i tabu potez u iteraciji j: ukoliko je $tabulist[i] > j$ i je zabranjeno pomerati, inače nije. Bitno je napomenuti da ove tabu zabrane ponovo računamo samo kada se promeni vrednost funkcije evaluacije f, pošto direktno zavisi od nje.

5.5 TS^0

Koristeći prethodno opisane funkcije funkcija TS^0 prvo vrši potrebne inicijalizacije i izračunavanja - funkciju evaluacije f(S), tabu zabrane (*tabutenure*), stepene povezanosti (inkrementalno). Potom, generiše se ograničena okolina i prelazi se na deo biranja sledećeg poteza. Ukoliko postoje bolji potezi (ogračena okolina nije prazna), nasumično se bira jedan od njih. Ukoliko ne postoji bolje rešenje sa verovatnoćom P biramo (mnogo) goru zamenu.

$$P = min\{(l+2)/|V|, 0.1\} \tag{5}$$

$$l = k * (k-1)/2 - f(s) \tag{6}$$

gde V predstavlja broj čvorova grafa. Ukoliko je neki od skupova A i B prazan ili je verovatnoća manja od p čvor u biramo nasumično, a čvor v biramo nasumično iz vektora $probV$, u kome se nalaze čvorovi koji nisu u S i zadovoljaju sledeći kriterijum:

$$degreesTowardsS[v] < \lfloor k * dens \rfloor \tag{7}$$

gde *dens* predstavlja gustinu grafa. U suprotnom na random način biramo po jedan čvor iz A i B za zamenu. Kada god izaberemo novi potez potrebno je ažurirati odgovarujeće podatke. Izbaciti čvor u iz S i ubaciti ga u *notInS*. Ubaciti čvor v u S i izbaciti ga iz *notInS*. Ažurirati funkciju evaluacije, odnosno uvećati je za delta koliko se promenila, kao i ponovo izračunati vrednosti u vektoru *tabulist* (uvećati *tabutenure* za broj iteracija). Takođe, potrebno je ažurirari vrednosti stepena povezanosti ka skupu S, što radimo inkrementalno. Na kraju, proveramo da li je skup čvorova koji se nalazi u S validna k-klika i ukoliko jeste vraćamo S i broj iteracija do kog smo stigli. Ukoliko

nije, uvećavamo broj iteracija i po potrebni ažuriramo funkciju evaluacije. Na kraju, ukoliko u while petlji nismo vratili rešenje, vraćamo najbolje S koje smo našli i broj iteracija.

5.6 AMTS

Funkcija AMTS prvo generiše inicijalno rešenje pozivajući funkciju opisanu u poglavlju 5.2. Zatim, dok god nije dostignut maksimalan broj iteracija (u našem slučaju je to 10^5), poziva se funkcija TS^0. Ukoliko je rezultat poziva te funkcije validna k-klika AMTS vraća dobijeni skup S i broj iteracija za koji je uspeo da ga pronađe. Ukoliko rezultat nije validna k-klika treba proveriti da li je potrebno resetovati frekvencije na 0 - u slučaju da su sve frekvencije manje od k (osim na nultom mestu, tu je uvek 0), to je potrebno uraditi. Potom, skup S se resetuje na prazan skup i konstruiše se novo inicijalno rešenje na isti način. Ukoliko se desi da prođe maksimalan broj iteracija, a nije pronađeno validno rešenje funkcija vraća prazan skup, kao signal neuspeha.

6 Testiranje i rezultati

6.1 Brute force algoritam

Algoritam grube sile pokrenut je na sedam grafova i rezultati su dati u tabeli. Za grafove C125-9, hamming-6-2, keller-4 (oni se ne nalaze u tabeli) izvršavanje je trajalo duže od devet sati, pa je prekinuto.

Graf	Broj čvorova	Veličina klike	Rezultat	Vreme
jonhnson8-2-4	28	4	4	0.0080034
jonhnson8-4-4	70	14	14	203.557
hamming6-4	64	4	4	0.188635
MANN-a9	45	9	9	2103.68

6.2 AMTS

U originalnom radu [1], AMTS algoritam implementiran je u programskom jeziku C, kompajliran pomoću GNU GCC i pokrenut na racunaru sa 2GB RAM memorije i 2.61 GHz procesorom. Naš algoritma implementiran je u programskom jeziku C++, ali pokrenut na dosta bržem računaru (32GB

RAM, 5.0 GHZ CPU).
Testiranje je izvršeno za 76/80 DIMACS test instanci. Dve instance (p-hat300-1 i p-hat-300-2) nismo uspeli da pronađemo u izvorima, a dve instance (MANN-a45 i MANN-a81) nismo testirali zato što smo poređenjem naših do tada dobijenih rezultata i rezultata iz rada [1] procenili da bi vreme izvršavanja bilo previše drugačko.
Parametri algoritma su sledeći:
maxIters = 10^5 - maksimalan broj iteracija koji sme da se potroši na traženje klike veličine k
L = $|V|$ - maksimalni dozvoljeni broj iteracija u kojima tabu pretraga ne nalazi poboljšanje trenutnog rešenja (detektovanje stagnacije)
Način na koji smo testirali se razlikuje od testiranja u originalnom radu, ali je prema našoj proceni više fer. U [1] veličina klike k je fiksirana i algoritam se zaustavalja čim nađe tu veličinu, bez da se uveri da li postoji klika veličine k+1 ili pokušaja da je pronađe. Naša verzija pokušava redom da nađe klike veličine 3 pa nadalje do neuspeha. Posledica ovoga je da je ukupno vreme izvršavanja dosta veće nego u [1] zato što uvek dopuštamo algoritmu da odradi maxIters iteracija da pokuša da nađe kliku veličine k+1, stoga rezultati nisu direktno uporedivi. Takođe, mi dozvoljavamo maksimalno 10^5 iteracija po pozivu AMTS funkcije dok se u [1] dozvoljava 10^8 iteracija. Za svaku test instancu, algoritam je pokrenut 5 puta. Rezultati su beleženi i prosleđeni python skripti koja za njih pravi grafikone (svi grafikoni mogu se pronaći u okviru github repozitorijuma), a prikazani su u tabelarnom obliku ispod. Za 54/76 grafova, izvršavanje je uspešno. Nalazi se klika koja je ili dokazano optimalna ili je jednako velika kao u [1]. Za 22/76 grafova izvršavanje je neuspešno (preciznije, za 3 od tih 22, algoritam je ponekad uspeo da nađe adekvatnu kliku). U poređenju sa rezultatima iz [1], naš algoritam se pokazao lošije u pogledu pronađenih rešenja. Najinteresantniji rezultat dobili smo za instancu p-hat1000-3 gde nije uspešno pronađena ni klika veličine 3 pa algoritam vraća vrednost 0.

Graf	Broj čvorova	Veličina klike	Pr. veličina	Pr. vreme	Pr. iteracija	Uspešnost
brock200-1	200	21*	21	1.503392	101733.8	Pass
brock200-2	200	12*	11.2	1.619086	112010.6	Mixed
brock200-3	200	15*	15	1.527296	102599.4	Pass
brock200-4	200	17*	16	1.494418	100570	Failed
brock400-1	400	27*	25	39.93268	106514.8	Failed
brock400-2	400	29*	25.4	40.28754	118757.8	Failed
brock400-3	400	31*	25	35.14646	104105.4	Failed
brock400-4	400	33*	33	113.85356	3363357.4	Pass
brock800-1	800	23*	21	206.525	128160.6	Pass
brock800-2	800	24*	21	206.10320	127728.4	Failed
brock800-3	800	25*	21.8	255.6374	158602.8	Failed
brock800-4	800	26*	21	192.3916	119597.2	Failed
C125.9	125	34*	34	0.9890952	100340.2	Pass
C250.9	250	44*	44	2.39492	101609.4	Pass
C500.9	500	57	57	8.132836	134294.4	Pass
C1000.9	1000	68	67.2	473.1254	184777.4	Mixed
C2000.5	2000	16	16	1324.712	123060.2	Pass
C2000.9	2000	80	75.8	1838.714	172683	Failed
C4000.5	4000	18	17.2	5639.228	125212.2	Mixed
DSJC500.5	5000	13*	13	6.138002	101772.6	Pass
DSJC1000.5	1000	15*	15	25.6607	123190	Pass
keller4	171	11*	11	1.173838	100224.2	Pass
keller5	776	27	27	202.2396	132377.8	Pass
keller6	3361	59	51.6	5243.37	169418	Failed
MANN_a9	45	16*	16	0.466452	100037	Pass
MANN_a27	378	126*	125	6.560738	133487.4	Failed
hamming6-2	64	32*	32	0.6927406	100100.8	Pass
hamming6-4	64	4*	4	0.4214128	100045	Pass
hamming8-2	256	128*	128	3.25035	101923.2	Pass
hamming8-4	256	16*	16	2.363712	100169	Pass
hamming10-2	1024	512*	512	44.6079	145105	Pass
hamming10-4	1024	40	40	23.18476	108133	Pass

Graf	Broj čvorova	Veličina klike	Pr. veličina	Pr. vreme	Pr. iteracija	Uspešnost
gen200-p0.9_44	200	44	44	1.658236	103976.8	Pass
gen200-p0.9_55	200	55	55	1.70394	103436.2	Pass
gen400-p0.9_55	400	55	55	6.539454	147807.8	Pass
gen200-p0.9_65	400	65	65	5.145824	113068	Pass
gen200-p0.9_75	40 0	75	75	5.660898	119447.8	Pass
c-fat200-1	200	12*	12	1.350972	100156	Pass
c-fat200-2	200	24*	24	1.40014	100392.4	Pass
c-fat200-5	200	58*	58	1.679924	100825.6	Pass
c-fat500-1	500	14*	14	5.581794	100571.8	Pass
c-fat500-2	500	26*	26	5.626896	100579.6	Pass
c-fat500-5	500	64*	64	6.099516	103476	Pass
c-fat500-10	500	126*	126	6.864328	100236.4	Pass
johnson8-2-4	28	4	4	0.2640802	10002	Pass
johnson8-4-4	70	14*	14	0.5944746	100062	Pass
johnson16-2-4	120	8*	8	0.9433682	100088	Pass
johnson32-2-4	496	16*	16	61.0864	100208	Pass
p-hat300-3	300	36*	36	2.973544	100817.6	Pass
p-hat500-1	500	9*	9	5.829276	100362	Pass
p-hat500-2	500	36*	36	5.890508	100655.6	Pass
p-hat500-3	500	50	50	5.973236	101391.2	Pass
p-hat700-1	700	11*	11	124.1382	100878.6	Pass
p-hat700-2	700	44*	44	123.69	100911.2	Pass
p-hat700-3	700	62	62	122.9916	101596.2	Pass
p-hat1000-1	1000	10	10	256.5618	100397	Pass
p-hat1000-2	1000	46	46	258.7654	101701.2	Pass
p-hat1000-3	1000	68	0	256.01	100507.2	Failed
p-hat1500-1	1500	12*	11.8	777.0168	129637.6	Pass
p-hat1500-2	1500	65	65	613.768	102711.2	Pass
p-hat1500-3	1500	94	94	635.215	107342.8	Pass

Graf	Broj čvorova	Veličina klike	Pr. veličina	Pr. vreme	Pr. iteracija	Uspešnost
san200_0.7_1	200	30*	26	6.11427	345034.2	Failed
san200_0.7_2	200	18*	18	2.368336	144151.6	Pass
san200_0.9_1	200	70*	70	5.07715	253636.6	Pass
san200_0.9_2	200	60*	60	2.293954	124871.4	Pass
san200_0.9_3	200	44*	44	1.783728	111387	Pass
san400_0.5_1	400	13*	9.14	7.774674	172995.8	Failed
san400_0.7_1	400	40*	22.8	6.535212	139283	Failed
san400_0.7_2	400	30*	19.8	9.89827	213682.2	Failed
san400_0.7_3	400	22*	18.8	6.455114	142725.8	Failed
san400_0.9_1	400	100*	84.2	27.93182	509987.2	Failed
san1000	1000	15*	9.4	321.8234	122924.8	Failed
sanr200_0.7	200	18*	18	1.45	100684	Pass
sanr200_0.9	200	42*	42	1.614368	101391.6	Pass
sanr400_0.5	400	13*	13	4.40858	103222.8	Pass
sanr400_0.7	400	21	21	4.348136	101518.4	Pass

Napomena: * u koloni veličina klike označava da je ta vrednost potvrđena kao optimalna. Poslednja kolona uspešnost ima sledeće značenje:
Pass: U svim iteracijama se pronalazi tražena veličina.
Mixed: U nekim iteracijama se pronalazi tražena veličina.
Failed: Ne pronalazi se tražena veličina ni u jednoj iteraciji.

Od 22 neuspešne instance, za 11 je ponovo pokrenut algoritam sa maxIters = 10^8. Rezultati su prikazani u tabeli ispod. Za 7/11 instanci ovaj put je pronađena adekvatna maksimalna klika. Za 3 instance algoritam uspeva da u barem 3/5 pokretanja nađe očekivanu kliku (detaljnije se može videti na graficima u repozitorijumu), dok samo za 1 povećan broj iteracija nije bio dovoljan da se postigne ikakav bolji rezultat. Algoritam se izvršavao dosta sporije za svih 11 instanci.

Graf	Broj čvorova	Veličina klike	Pr. veličina	Pr. vreme	Pr. iteracija	Uspešnost
MANN_a27	378	126*	126	591.9964	12598168.6.4	Pass
brock200-2	200	12*	12	154.1796	10729337.8	Pass
brock200-4	200	17*	17	196.4466	13515367.4	Pass
brock400-1	400	27*	25.2	418.5732	10116237.2	Failed
brock400-2	400	29*	29	492.5542	11877334	Pass
brock400-3	400	31*	31	743.518	17843127.8	Pass
san200_0.7_1	200	30*	30	177.4416	10495062.2	Pass
san400_0.5_1	400	13*	12.4	682.2252	16061443.8	Mixed
san400_0.7_1	400	40*	40	1250.508	20485776.6	Pass
san400_0.7_2	400	30*	28.8	1249.1758	27505364.6	Mixed
san400_0.7_3	400	22*	21.8	606.899	14968358.6	Mixed

Kada uporedimo pristup sa tabu pretragom sa brutreforce algoritmom, vidimo da je bruteforce brži za 2 najmanje instance (johnson8-2-4 i hamming6-4), ali počinje da zaostaje za metaheurističkim pristupom vec za naredne 2 po veličini (johnson8-4-4 i MANN-a9), dok se za malo veće instance bruteforce pokazao potpuno nepraktičan (traje više od 9 sati, dok tabu pretraga pronalazi rešenje za manje od jedan sekund).

7 Zaključak

Tokom istraživanja literature videli smo više različitih pristupa za rešavanje problema traženja maksimalne klike. Tabu pretraga se pokazala kao interesantan izbor. Rad [1], koji nam je bio primarni oslonac, je u trenutku kada je nastao pokazao solidne rezultate i bilo nam je interesantno da pokušamo da ih repliciramo. Iako naši rezultati izgledaju lošije u pogledu vremena izvršavanja, mislimo da prikazuju realniji način upotrebe ovog algoritma. Izmene parametara su poboljšale veličine nađenih kliki ali u zamenu za mnogo veće vreme izvršavanja, što za instance na kojima je testirano predstavlja dodatno udaljavanje od traženih rezultata. Moguća poboljšanja se potencijalno mogu naći u korišćenju nekakvih pametnijih struktura podataka i rado ćemo ih isprobati u budućnosti.

Literatura

[1] Qinghua Wu, Jin-Kao Hao - An adaptive multistart tabu search approach to solve the maximum clique problem

[2] D.-Z. Du, P.M. Pardalos - Handbook of Combinatorial Optimization, The Maximum Clique Problem

[3] Fred Glover, Manuel Laguna - Tabu search