VNS metoda za nalaženje maksimalne klike

Projekat u okviru kursa Računarska inteligencija
Matematički fakultet
Univerzitet u Beogradu

Todor Todorović 241/2019
Nevena Radulović 407/2016

28. jun 2023.

Sadržaj

1 Opis problema i matematička formulacija

Neka je $G = (V, E)$ neusmeren graf, gde je $V = \{1, ..n\}$ skup čvorova i $E \subset V \times V$ skup grana grafa. Klika C grafa G je podskup skupa V u kome su svaka dva čvora međusobno spojena granom $e \in E$. Maksimalna klika je klika čija je kardinalnost veća od kardinalnosti svih drugih klika. Ponalaženje maksimalne klike (engl. Maximum Clique Problem) je NP-težak problem što znači da se egzaktnim metodama ne može rešiti u polinomijalnom vremenu. Sa druge strane problemi iz realnog života uglavnom iziskuju da se do rešenja dođe u razumnom vremenu, što je egzaktnim metodama neizvodljivo, naročito za probleme većih dimenzija. Iz tog razloga se za rešavanje NP-teških problema najčešće koriste metaheurističke metode [3].

Zbog svojih mnogobrojnih primena problem maksimalne klike je naširoko izučavan. Neke od oblasti u kojima se primenjuje su socijalne strukture, ekonomija, obrasci u telekomunikacionom saobraćaju, dizajn kodova za ispravljanje grešaka, dijagnostika kvarova na velikim multiprocesorskim sistemima, prepoznavanje obrazaca, biološke i hemijske analize itd. Problem pronalaska maksimalne klike je ekvivalentan problemu maksimalnog nezavisnog skupa i usko je povezan sa problemom bojenja grafa.

Da bismo problem pronalaženja maksimalne klike formulisali kao problem celobrojnog programiranja koristićemo binarne promenljive koje se definišu na sledeći način

$$x_i = \begin{cases} 1, & \text{ako čvor } i \in C, \\ 0, & \text{inače.} \end{cases}$$

Sada matematičku formulaciju možemo napisati na sldeći način:

$$\max \sum_{i=1}^{n} x_i \tag{1}$$

$$x_i + x_j \leq 1, \quad (i, j) \in E^C \tag{2}$$

$$x_i \in \{0, 1\}, \quad i = 1, 2, ..., n. \tag{3}$$

Celobrojni zbir (1) predstavlja veličinu klike i cilj je njegova maksimizacija. Uslovi (2) obezbedjuju da svaki par čvorova skupa C zaista jeste susedan, dok je uslovima (3) iskazana binarna priroda promenljivih koje su krišćene u modelu.

2 Reprezentacija grafa i ulazni podaci

U ovom radu, graf je predstavljen matricom povezanosti. Matrica je dimenzija $N \times N$, gde je N broj čvorova grafa. Ukoliko u grafu postoji grana između čvora u i čvora v vrednost matrice povezanosti ($adjacencyMatrix$) je 1, u suprotnom je 0, odnosno

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

Test instance na kojima smo testirali algoritam mogu se preuzeti sa sledeće adrese: https://networkrepository.com/dimacs.php.

3 Brute force algoritam

3.1 Implementacija

Brute force algoritam je implementiran rekurzivnom metodom uz korišćenje pomoćne funkcije *isClique* koja kao argument dobija skup čvorova *clique* i ako je taj skup klika, funkcija vraća $True$ a u subrotom vraća $False$.

Algorithm 1 Pomoćna funkcija *isClique*

Require: Skup čvorova *clique*
Ensure: True ako je skup *clique* klika, False u suprotnom
  n = broj čvorova u clique
  for $i = 1$ to n **do**
    for $j = i + 1$ to n **do**
      if ne postoji grana između čvorova *clique*[i] i *clique*[j] **then**
        return False
      end if
    end for
  end for
  return True

Argumenti $brute\_force$ algoritma su matrica susedstva grafa G, čvor od koga se započinje pretraga $startNode$ i trenutna dužina klike $size$. Promenljiva $maxClique$ se koristi da čuva veličinu trenutne maksimalne klike i inicijalno je jednaka 0. Promenljiva n označava broj čvorova grafa G. Ideja je da se u for petlji pokušava dodavanje bilo kog narednog čvora, počevši od $(startNode+1)$-og, ali tako da trenutni skup i dalje ostane klika. Ukoliko je moguće dodati čvor, upoređuje se veličina te klike sa dosadašnjom maksimalnom veličinom i ona se ažurira ukoliko je nova klika veća. Zatim u rekurzivnom koraku pozivamo algoritam sa onom vrednošću koju je uzeo $startNode$ i uvećanom vrednošću $size$ za 1. Povratna vrednost tog poziva ažurira po potrebi trenutnu vrednost veličine maksimalne klike.

Kako bi pretražio sve moguće kombinacije i našao globalno najveću kliku, algoritam se poziva za početni čvor $startNode$=0, i početnu veličinu klike $size$=1.

Algorithm 2 $brute\_force$ *za MCP*

Require: Matrica susedstva grafa G, polazni čvor, veličina trenutne maksimalne klike
Ensure: Veličina maksimalne klike grafa G
  n = broj čvorova grafa G
  $maxClique = 0$
  for $node = startNode + 1$ *to* n **do**
    $clique[size] = node$
    if $isClique(clique)$ **then**
      $maxClique = max(maxClique, size)$
      $maxClique = max(maxClique, brute\_force(node, size + 1))$
    end if
  end for
  return $maxClique$

Pošto Bruteforce algoritam postaje neupotrebljiv za veće instance, pokušali smo da implementiramo algoritam zasnovan na VNS pretrazi i na taj način dođemo do rešenja.

4 VNS algoritam

Metoda promenljivih okolina(engl. Variable Neighborhood search - VNS), koju su predložili Mladenović i Hansen, je metaheuristički metod koji se koristi za rešavanje problema kombinatorne i globalne optimizacije. Kao i sve metaheurističke metode i VNS se sastoji iz uopštenih koraka koji se izvršavaju sve dok nije ispunjen neki kriterijum zaustavljanja, a ti koraci se moraju prilagoditi konkretnom problemu koji se rešava. Osnovna ideja je sistematična promena okolina prilikom lokalne pretrage. Lokalna pretraga polazi od početnog rešenja i u njegovoj okolini nalazi bolje rešenje. Postupak se nastavlja od tog rešenja dok se ne dođe do lokalnog optimuma. Ovaj pristup koristi empirijski utvrđenu činjenicu da su za mnoge probleme lokalni optimumi, u odnosu na različite okline, relativno blizu jedan drugog. Zbog toga ima smisla istraživati okoline lokalnih optimuma u potrazi za globalnim.

U ovom radu implementirana je osnovna metoda promenljivih okolina (engl. Basic VNS) koju karakteriše stohastički korak koji se naziva razmrdavanje rešenja.

4.1 Reprezentacija rešenja i konstrukcija inicijalnog rešenja

Dopustivo rešenje problema maksimalne klike predstavljeno je skupom prirodnih brojeva koji predstavljaju čvorove koji čine kliku. Početno rešenje, iz koga ćemo započeti VNS pretragu, sadrži jedan nasumično odabrani čvor $v \in V$. Da bi se utvrdilo da li je generisano rešenje dopustivo i koliko odstupa od dopustivog koristi se funkcija prilagođenosti.

4.2 Funkcija prilagođenosti

Funkcija prilagođenosti definisana je na sledeći način:

Neka je n broj čvorova posmatranog rešenja S i neka je m broj grana koje su određene čvorovima rešenja S. Funkciju prilagođenosti (engl. $Fitness$ funkciju) definisali smo na sledeći način

$$fit(S) = \begin{cases} \frac{n(n-1)}{2} - m, & \text{ako } \frac{n(n-1)}{2} - m > 0 \\ \frac{1}{n}, & \text{ako } \frac{n(n-1)}{2} - m = 0. \end{cases}$$

Ideja je usmeriti pretragu ka rešenju sa što manjom vrednosti fitness funkcije. Za rešenja S koja nisu klika ova vrednost će biti broj grana koje nedostaju da bi to rešenje bilo klika, dok za rešenja S koja jesu klika, tj. broj grana koje nedostaju je 0, vrednost funkcije prilagođenosti se definiše kao $\frac{1}{n}$.

4.3 Razmrdavanje rešenja

Osnovna varijanta metode promenljivih okolina uključuje i komponentu koja se naziva razmrdavanje (engl. *shaking*). Cilj ove komponente je da spreči da se pretraga zaglavi u nekom lokalnom optimumu, koji nije globalni. U ovom koraku se iz unapred definisane okoline trenutnog rešenja na slučajan način

bira element od koga se nastavlja dalja pretraga. Okoline za fazu razmrdavanja mogu, a ne moraju biti definisane na isti način kao u fazi lokalne pretrage. Izbor okolina u ovim koracima značajno utiče na performanse VNS algoritma. U ovom radu razmrdavanje je postignuto na sledeći način: element S_{new} iz k-te okoline trenutnog rešenja S, dobija se zamenom k nasumično odabranih čvorova iz rešenja S sa k nasumično odabranih čvorova iz skupa S^c, tj. komplementa skupa S. Na taj način formira se novi skup S_{new}. Zatim skupu S_{new} dodajemo jedan nasumično odabran čvor iz njegovog komplementarnog skupa S^c_{new}. Ovako dobijeno rešenje S_{new} koristi se kao inicijalno rešenje za lokalnu pretragu.

Algorithm 3 *Shaking*

Require: Skup S veličine n, Broj k
Ensure: Skup S_{new} veličine $n+1$
$S_{new} = k\_swap(S, S^c)$
$S_{new} = S_{new} + 1\_add(S^c_{new})$

4.4 Lokalna pretraga

Za lokalnu pretragu korišćena je okolina N_1 čiji se elementi od trenutnog rešenja razlikuju u jednom čvoru. Pretraga se započinje od rešenja S_{new} i ono se iterativno poboljšava dokle god je to moguće. Prelazak u novo rešenje smo implementirali po best improvement strategiji. To podrazumeva pretragu cele okoline trenutnog rešenja, u cilju nalaženja lokalnog optimuma u toj okolini, a zatim se pretraga nastavlja od tog optimuma.

Algorithm 4 *local_search za MCP*

Require: Skup S
Ensure: Skup S_{best} koji je loklani optimum
Inicijalizacija: $best\_fit = fitenss(S), improvement = 1$
while $improvement$ **do**
  $improvement = 0$
  for $x \in S$ **do**
    for $y \in S^C$ **do**
      $S_{new} = (S \cup y) \setminus x$
      $new\_fit = fitness(S_{new})$
      if $new\_fit < best\_fit$ **then**
        $best\_fit = new\_fit$
        $S_{best} = S_{new}$
        $improvement = 1$
      end if
    end for
  end for
end while

4.5 VNS

U okviru VNS algoritma prvo se generiše inicijalno rešenje S na način opisan u poglavlju 4.1. Ovo rešenje je na početku algoritma ujedno i najbolje rešenje.

Zatim se smenjuju faze razmrdavanja i lokalne pretrage, sve do dok nije zadovoljen izabrani kriterijum zaustavljanja. U ovom radu je kao kriterijum zaustavljanja korišćen maksimalan broj iteracija. Vrednost parametra k inicijalno se postavlja na 1. U okviru *shaking* faze generiše se novo rešenje S_{new} iz k-te okoline rešenja S . Od rešenja S_{new} se zatim pokreće lokalna pretraga koja nalazi lokalni optimum. Ukoliko je vrednost $fitness$ funkcije nađenog lokalnog optimuma manja od vrednosti $fitness$ funkcije do tada nađenog najboljeg rešenja S, prelazi se u novi lokalni optimum i algoritam nastavlja sa izvršavanjem od tog rešenja. U suprotnom se povećava veličina k za *shaking* funkciju. Ukoliko je, pak, $k > \frac{n}{2}$ (gde je n broj čvorova trenutnog rešenja) resetuje se vrednost k na 1 i cela funkcija prelazi u sledeću iteraciju.

Algorithm 5 $VNS\ za\ MCP$

Require: Matrica susedstva grafa G
Ensure: Skup C koji je maksimalna klika u grafu G
Inicijalizacija: generisati inicijalno rešenje S, definisati okoline $N_1, N_2, ...N_{k_{max}}$ za fazu razmrdavanja, kao i vrednost parametra k_{max}. Definisati okolinu za fazu lokalne pretrage. Postaviti vrednost parametra *iter* na 1.
$k = 1$
while $iter < iter\_max$ **do**
  $S_{new} = shaking(S, k)$
  $iter = iter + 1$
  $S'' = local\_search(S_{new})$
  if $fitness(S'') \leq fitness(S)$ **then**
    $S = S'', k = 1$
  else
    $k = k + 1$
    if $k > n/2$ **then**
      $k = 1$
    end if
  end if
end while

5 Eksperimentalni rezultati

Bruteforce i VNS algoritam testirani su na skupu od 11 instanci preuzetih sa [2]. Algoritmi su implementirani u programskom jeziku C++, a testiranje je rađeno na računaru sa intel procesorom I5-3320M, 4 jezgra, taktom 3.3ghz, ram memorijom od 16GB, pod operativnim sistemom Ubuntu. Bruteforce algoritam preuzet je iz rada [1]. Kako su instance korišćene u ovom radu većih dimenzija, algoritam se loše pokazao. Kako ni nakon 10 sati izvršavanja nisu dobijene tražene klike, algoritam je prekinut.

Za kriterijum zaustavljanja u VNS metodi korišćen je maksimalan broj iteracija $iter\_max$. Algoritam je pokretan 5 puta za svaku instancu sa različitim početnim rešenjima.

		VNS					VNS HMU	
Instanca	|V|	*best*	*worst*	*average*	*t*	*iter_max*	|clique|	*t*
C125_9	125	34	34	34	133.36	200	34	0.02
brock200_2	200	11	11	11	5000	5000	11-12	1.05
gen400_p0_9_55	400	55	55	55	62459	4000	54.8	34.28
hamming8_4	256	16	16	16	369	200	16	0.01
C250_9	250	40	40	40	17404	3000	44	0.22
keller4	171	11	11	11	66	200	11	0.01
gen200_p0_9_44	200	40	38	39.6	10209	3000	44	0.91
brock200_4	200	15	15	15	9036	2000	16-17	6.8
gen200_p0_9_55	200	55	55	55	18500	3000	55	0.24
keller5	776	25	25	25	111218	1000	27	0.38
p-hat300-3	300	36	33	34.6	22291	2000	36	0.07

Tabela 1: Poređenje predložene VNS metode i VNS metode iz literature

U Tabeli 1 prikazani su rezultati predloženog VNS algoritma, kao i rezultati VNS metode iz rada [3]. Prve dve kolone sadrže naziv instance, kao i broj čvorova grafa. Zatim su prikazane vrednosti dobijene VNS algoritmom:

- *best*-najbolje od 5 dobijenih rešenja
- *worst*-najgore od 5 dobijenih rešenja
- *average*-prosečno rešenje
- *t*-srednje ukupno vreme izvršavanja (u sekundama)
- *iter_max*-parametar metode za kriterijum zaustavljanja

Poslednje dve kolone sadrže rezultate dobijene u radu [3]-srednju veličinu klike, kao srednje ukupno vreme (u sekundama). Predloženi VNS našao je u 5 od 11 slučajeva isto rešenje kao VNS [3], dok je u 1 slučajeva dobijeno bolje rešenje nego u [3]. Na osnovu dobijenih rezultata može se uočiti robusnost predloženog VNS algoritma. Prsečne i najgore klike se ne razlikuju značajno od najboljih, što znači da je pri svakom pokretanju dobijeno kvalitetno rešenje, čak i kada nije najbolje.

6 Zaključak

U ovom radu predložen je VNS algoritam za problem nalaženja maksimalne klike u neusmerenom grafu, kao i Bruteforce algoritam. Oba algoritma testirana su na istom skupu instanci, s time da Bruteforce nije uspeo da se izbori sa zadatim instancama, dok su VNS algoritmom dobijena kvalitetna rešenja. Dobijena rešenja upoređena su sa rešenjima iz literature. Na osnovu analize može se zaključiti da je ova metoda pogodna za problem Maksimalne klike. Pravci daljeg unapređenja su dodatno testiranje za različite vrednosti parametara u metodi, primena neka druge varijante VNS algoritma (SVNS, RVNS,...) ili hibridizacija sa nekom drugom egzaktnom ili heurističkom metodom.

Literatura

[1] Bruteforce algorithm. https://github.com/JovanDjordjevic/RI_maximum_clique_tabu_search/blob/main/bruteforce.cpp.

[2] Dimacs networks. https://networkrepository.com/dimacs.php.

[3] Pierre Hansen, Nenad Mladenović, and Dragan Urošević. Variable neighborhood search for the maximum clique. *Discrete Applied Mathematics*, 145(1):117–125, 2004. Graph Optimization IV.