Minimum bandwidth problem

Projekat u okviru kursa Računarska inteligencija
Univerzitet u Beogradu
Matematički fakultet

Luka Radanović
mi19169@alas.math.rs

Avgust 2022

Sadržaj

1 Uvod

1.1 Opis problema

Problem minimalnog bandwidth-a grafa je problem obeležavanja čvorova različitim prirodnim brojevima tako da je najveća apsolutna razlika dva susedna čvora minimalna.

Formalno, neka je $G = (V, E)$ graf (V je skup čvorova, a E skup grana grafa) takav da je $|V| = n$ i $|E| = m$. Obeležavanje f grafa G je bijektivno preslikavanje $f : V \to \{1, 2, \ldots, n\}$. Bandwidth grafa G u odnosu na f, u oznaci $B_f(G)$, je $B_f(G) = \max\{|f(u) - f(v)| : (u, v) \in E\}$. Problem minimalnog bandwidth-a predstavlja problem nalaženja obeležavanja f^* takvog da je odgovarajući bandwidth najmanji mogući: $B_{f^*}(G) = \min_f\{B_f(G)\}$.

Ako posmatramo odgovarajuću matricu susedstva $A = (a_{ij})$ grafa G, ovaj problem predstavlja pronalaženje permutacije vrsta i kolona matrice takve da je rastojanje najdaljeg ne-nula elementa od glavne dijagonale minimalno.

1.2 Neki od dosadašnjih pristupa

Pokazano je da je ovaj problem NP-kompletan, tako da se egzaktni algoritmi mogu koristiti samo za manje dimenzije problema. Postoji veliki broj pohlepnih algoritama, kao i metaheuristika koje su razvijene za rešavanje ovog problema [2].

Od pohlepnih algoritama istorijski najznačajniji je Cuthill-McKee algoritam. Ovi algoritmi su uglavnom zasnovani na pretrazi grafa u širinu (BFS), i veoma su efikasni. Mana im je što daju znatno lošija rešenja od metaheurističkih algoritama.

Neke od metaheuristika koje su dale najbolje rezultate u minimizaciji bandwidth-a su pristup zasnovan na metodi promenljivih okolina (VNS) [4], kao i simulirano kaljenje sa dualnom reprezentacijom rešenja (DRSA) [5]. DRSA je dao nešto bolje rezultate u minimizaciji od VNS-a, ali je zato sporiji [3].

2 Simulirano kaljenje

U ovom radu je korišćen pristup na kom je zasnovan DRSA algoritam. Ovaj algoritam koristi simulirano kaljenje, ali uz to koristi dve reprezentacije rešenja i kompleksniju funkciju susedstva, što mu je omogućilo da postigne bolje rezultate od ranijih algoritama simuliranog kaljenja.

Simulirano kaljenje je inspirisano procesom zagrevanja i hlađenja metala u cilju dobijanja stabilne kristalne strukture. Proces se sastoji od povećavanja temperature do maksimalne vrednosti na kojoj se metal topi, a zatim njenog smanjivanja sve dok se ne dođe u stanje u kom su čestice metala uređene tako da je energija celog sistema minimalna.

2.1 DRSA algoritam

Algoritam koristi tri parametra koji se odnose na temperaturu: početnu temperaturu T_0, krajnju temperaturu T_f i stopu hlađenja $\alpha (0 < \alpha < 1)$. Temperatura se tokom izvršavanja smanjuje po formuli $T = \alpha T$ dok ne dostigne vrednost

T_f što označava kraj izvršavanja algoritma. Pored toga, algoritam koristi i parametar L (dužinu Markovljevog lanca) koji određuje koliko će se puta menjati rešenje na istoj temperaturi T i L_f (završnu dužinu Markovljevog lanca) koja se dostiže kada temperatura dostigne T_f. Za ove parametre se koriste vrednosti koje su korišćene i u originalnom radu: $T_0 = 1000$, $\alpha = 0.99$, $L_f = 10nm$, $T_f = 10^-7$, $L = 40$. U ovom radu se koristi još jedan dodatni parametar, a to je maksimalno vreme izvršavanja t_{max}. Ako vreme izvršavanja algoritma pređe ovu vrednost, algoritam se prekida i vraća se najbolje rešenje dobijeno do tog trenutka. Takođe postoji i parametar koji određuje kako će se dobiti početno rešenje: nasumično ili pohlepnim algoritmom.

Algorithm 1: DRSA algoritam

```
Data: t_max, greedy_init
Result: Najbolje pronađeno rešenje w
T_0 ← 1000;
α ← 0.99;
T_f ← 10^−7;
L ← 40;
L_f ← 10 * n * m;
γ ← compute-increment-factor(T_0,T_f,L,L_f) ;
start_time ← time();
if greedy_init = true then
    x ← greedy-algorithm();
else
    x ← generate-random-solution();
end
w ← x;
while T > T_f do
    improvement ← false;
    for i ← 1 to L do
        y ← g(x);
        if f(y) < f(x) then
            x ← y;
        else if random(0,1) < e^−(f(y)−f(x))/T then
            x ← y;
        if f(x) < f(w) then
            w ← x;
            improvement ← true;
    end
    if improvement = false then
        T ← T * α;
        L ← L * γ;
    if time()−start_time > t_max then
        break
end
return w
```

Na početku se parametri inicijalizuju na odgovarajuće vrednosti, a trenutna temperatura T se postavlja na T_0. Zatim se izračunava faktor inkrementiranja

γ Markovljevog lanca tako da kada T dostigne T_f, L dostigne L_f. On se računa po formuli $\gamma = e^{(\frac{log(L_f)-log(L)}{r})}$, gde je r ukupan broj smanjivanja temperature koji se dobija kao $r = \frac{log(T_f)-log(T_0)}{log(\alpha)}$.

Zatim se pravi inicijalno rešenje. U originalnom algoritmu se generiše slučajno izabrano rešenje, dok je u ovom radu pored tog pristupa isprobano i korišćenje pohlepnog algoritma za dobijanje početnog rešenja.

Algoritam se sastoji od while petlje koja se izvršava dok god je temperatura T veća od završne T_f i unutrašnje for petlje koja na trenutnoj temperaturi L puta menja vrši menjanje rešenja. Od trenutnog rešenja x se dobija susedno y korišćenjem funkcije susedstva g i prihvata se ako ima manju vrednost funkcije evaluacije f. Ako nema, prihvata se sa verovatnoćom $e^{-(f(y)-f(x))/T}$. Ako u toku izvršavanja for petlje ne uspe da se nađe rešenje bolje od trenutno najboljeg, trenutna temperatura T se smanjuje sa faktorom α, a L povećava sa faktorom γ. Na kraju svake iteracije se proverava da li je prekoračeno maksimalno vreme izvršavanja t_{max}.

2.2 Interna reprezentacija rešenja

DRSA interno koristi dve reprezentacije rešenja i obe predstavljaju permutacije skupa $\{1, 2, \ldots, n\}$. Prva reprezentacija je $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$, gde π_i predstavlja broj (tj. oznaku) dodeljenu čvoru i. Druga reprezantacija rešenja $\rho = (\rho_1, \rho_2, \ldots, \rho_n)$ ima značenje da je čvoru ρ_i dodeljena oznaka i.

Razlog za korišćenje dve reprezentacije je u tome što je funkcija susedstva sačinjena od operatora koji rade nad π i operatora koji rade nad ρ reprezentaciji, što algoritmu omogućava bolje kretanje kroz prostor rešenja.

U ovom radu se primarno koristi π reprezentacija, a ρ se izračunava samo kada potrebno se koristi odgovarajući operator funkcije susedstva, za razliku od originalnog rada gde se oba rešenja održavaju sve vreme.

2.3 Funkcija susedstva

Funkcija susedstva se koristi da se od trenutnog rešenja dobije novo. U ovom algoritmu, funkcija susedstva se sastoji od tri operatora:

- Prvi operator je REX (Random Exchange Operator) koji radi nad π reprezentacijom. Ovaj operator nasumično bira dve pozicije i i j i razmenjuje njihov sadržaj, tj. međusobno razmenjuje oznake dva čvora grafa.

- Drugi operator je NEX (Neighbor Exchange Operator) koji takođe radi nad π reprezentacijom. Ovaj operator nasumično bira jedan čvor i i razmenjuje njegovu oznaku sa oznakom jednog od njegovih suseda j koji je takođe izabran nasumično. On dakle takođe menja sadržaj pozicija i i j u π, samo što su sada oni obavezno susedi.

- Treći operator je ROT (Rotation Operator) koji radi nad ρ reprezentacijom. Ovaj operator nasumično bira dve pozicije i i j u ρ, takve da je $i < j$, a zatim se element na poziciji i pomera na poziciju j, a elementi od $i+1$ do j se pomeraju za jednu poziciju ulevo. ROT operator pravi glatku promenu među čvorovima, jer svi čvorovi od ρ_{i+1} do ρ_j menjaju svoju oznaku za jedan, dok se jedina velika promena dešava kod ρ_i koji menja

oznaku sa i na j. U ovom radu, kao i u originalnom, se kod operatora ROT i i j biraju tako da važi $1 \leq |i - j| \leq 5$

Pri svakom pozivu funkcije susedstva, jedan od tri operatora se bira nasumično i primenjuje na trenutno rešenje. Svakom od operatora je pridružena odgovarajuća verovatnoća. U ovom radu se koriste verovatnoće koje su dobijene kao optimalne u testiranjima u originalnom radu: verovatnoća za prvi je 0.6, za drugi 0.2 i za treći 0.2.

2.4 Funkcija evaluacije

Funkcija evaluacije koju DRSA koristi, pored samog bandwidth-a koji određeno rešenje daje, uzima u obzir i sve apsolutne razlike oznaka susednih čvorova što omogućuje da izdvoji bolja rešenja među rešenjima koja imaju isti bandwidth. Time, ova funkcija smanjuje šansu da se algoritam zaglavi u platou.

Za izračunavanje funkcije evaluacije koristi se vektor d dužine n, čiji i-ti element d_i predstavlja broj apsolutnih razlika jednakih i, određenih rešenjem koje se evaluira. Ovo se može posmatrati i kao broj ne-nula elemenata na dijagonali koja se nalazi na rastojanju i od glavne dijagonale u matrici susedstva. Za $i = 0, 1, \ldots, n-1$ broj apsolutnih razlika jednakih i je najviše $n - i$, pa je $0 \leq d_i \leq n - i$ za svako d_i, pa je broj mogućih vrednosti za svako d_i jednak $n - i + 1$. Neka je v vektor dužine n čiji je i-ti element broj mogućih vrednosti koje d_i može da ima, $v = (n + 1, n, \ldots, 2)$. Pomoću d i v može se definisati brojevni sistem sa promenljvom osnovom, čije su cifre elementi vektora d, a elementi v brojevi mogućih vrednosti za svaku poziciju.

Ideja je da se funkcija evaluacije sastoji od dva dela: celobrojnog (koji predstavlja bandwidth β) i razlomljenog (sa vrednošću od 0 do 1). Ovaj drugi deo omogućava razlikovanje rešenja sa istim bandwidth-om i računa se tako što se vektor d preslika u normalizovanu vrednost δ ($0 \leq \delta < 1$).

Algorithm 2: Preslikavanje vektora d u δ vrednost

Data: d, v, β
Result: δ
$\delta \leftarrow 0$;
for $i \leftarrow 0$ **to** β **do**
  $\delta \leftarrow (\delta + d_i)/v_i$;
end

Funkcija evaluacije f se računa kao $f = \beta + \delta$. Dodavanje vrednosti δ omogućava razlikovanje rešenja sa istim bandwidth-om.

Detaljnijim razmatranjem se može zaključiti da ako se računa na gore opisan način, δ ne mora biti manje od 1, jer se zapravo posmatrao samo broj ne-nula elemenata na gornjoj dijagonali koja je na datom rastojanju od glavne. Zato je broj mogućih vrednosti za svako i zapravo duplo veći. Ipak, pokazalo se da ovakvo računanje funkcije evaluacije dovodi do boljih rezultata algoritma, pa je tako implementirano i u ovom radu, s tim što se vodi računa da se uvek pamti tačan bandwidth.

3 Pohlepni algoritam

Jednostavan pohlepni algoritam koji se koristi u ovom radu je napravljen po ugledu na tradicionalne heuristike za rešavanje problema minimizacije bandwidth-a i zasnovan je na obilasku grafa u širinu (BFS).

BFS pretraga se pokreće iz čvora sa najmanjim stepenom (ako ih ima više, uzima se bilo koji). Zatim se čvorovima dodeljuju oznake onim redom na kojim se na čvorove nailazi. Ako graf ima više komponenti povezanosti, postupak se ponavlja redom za svaku od njih.

Na ovaj način, čvorovi se obeležavaju redom po nivoima, a svaka grana (u neusmerenom grafu) ima krajeve u istom ili susednim nivoima, što može znatno ograničiti bandwidth. Upravo zato se i polazi od čvora sa najmanjim stepenom, da bi širina nivoa bila što manja.

4 Algoritam grube sile

Ovaj algoritam isprobava redom sve permutacije skupa $\{1, 2, \ldots, n\}$, tj. sva moguća obeležavanja grafa i vraća ono koje daje minimalan bandwidth.

Složenost algoritma je $O(n!)$, pa je moguće primenjivati ga samo na veoma malim dimenzijama problema.

5 Eksperimentalni rezultati

Algoritmi u radu su implementirani u jeziku C++, kompilirani koristeći G++ 10.3.0, i pokretani na AMD Athlon X4 840 procesoru na 3.0 GHz. DRSA u originalnom radu je implementiran u C-u i pokretan na nešto bržem procesoru (sudeći po upoređenom vremenu izvršavanja VNS algoritma). Iako je teško odrediti tačno koliko zbog razlike u hardveru, implementacija iz originalnog rada je svakako dosta efikasnija i znatno brža od one iz ovog rada.

Implementacija DRSA sa i bez korišćenja pohlepnog rešenja za inicijalizaciju, kao i sam pohlepni algoritam prvo su upoređeni sa algoritmom grube sile na dovoljno malim instancama koje su slučajno generisane za potrebe testiranja. Algoritmi su pokretani po tri puta i izračunata je srednja vrednost za bandwidth i vreme nalaženja bandwidth-a.

Rezultati su u sledećoj tabeli:

ime	n	Gruba sila		DRSA		DRSA_BFS		BFS	
		β	t (s)	β	t (s)	β	t (s)	β	t (s)
graph_9_2	9	2	0.0199893	2	0.0111269	2	0.0157975	3	2.9521e-05
graph_8_1	8	3	0.00167633	3	1.33463e-05	3	2.75613e-05	4	2.1336e-05
graph_13_1	13	3	447.559	3	0.0215134	3	2.2192e-05	3	4.6908e-05
graph_12_1	12	2	31.4245	2	0.0248881	2	0.0240763	3	3.5489e-05
graph_11_1	11	3	2.26534	3	0.0026155	3	0.00252803	4	3.3563e-05
graph_10_1	10	2	0.172391	2	0.0040499	2	0.00437372	3	2.7731e-05
graph_9_1	9	1	0.0150303	1	0.00629719	1	1.54993e-05	1	2.6432e-05

Tabela 1: Rezultati na slučajno generisanim matricama

Simulirano kaljenje sa i bez korišćenja pohlepnog algoritma uspeva da nađe optimalna rešenja na manjim matricama, i to brže od algoritma grube sile. Razlika u vremenu izvršavanja se posebno vidi sa povećavanjem dimenzija matrice. Pohlepni algoritam ima najbrže izvršavanje ali čak ni na malim instancama ne mora uvek dati optimalna rešenja.

Za poređenje sa algoritmom iz originalnog rada, odabrano je i iskorišćeno 15 matrica iz Harwell-Boeing skupa retkih matrica [1], koji se već tradicionalno koristi za poređenje rezultata rešavanja ovog problema.

Zbog dugog izvršavanja, algoritam nije puštan da se izvršava do kraja, već se posmatra najbolje dobijeno rešenje koje se dobija za 300 sekundi izvršavanja i poredi se sa najboljim dobijenim rešenjem iz originalnog rada. Za svaku instancu je algoritam pokretan po tri puta i beležen prosečan dobijen bandwidth kao i prosečno vreme njegovog dobijanja. Postupak je ponovljen za običan DRSA, DRSA sa početnim pohlepnim rešenjem i za sam pohlepni algoritam.

ime	n	Originalni DRSA	DRSA		DRSA_BFS		BFS	
		β	β	t (s)	β	t (s)	β	t (s)
will199	199	64	66	14.3757	66.3333	12.7677	195	0.0106505
pores_1	30	7	7.66667	0.18529	7.66667	0.184772	14	0.00010385
can_161	161	18	21.6667	11.0212	32	3.17277	33	0.00386305
sherman1	1000	46	99.3333	295.163	68	0.14067	68	0.139427
can_292	292	38	42.3333	119.004	44.6667	76.7704	77	0.0132052
can_144	144	13	13	14.6074	13	12.0245	20	0.00332474
lund_a	147	23	25	17.2851	23	0.00435125	23	0.00342371
arc130	130	63	63	188.655	64	98.1438	128	0.00249965
fs_760_1	760	37	79.3333	294.541	77.6667	295.994	206	0.0765773
gent113	113	27	27.3333	5.27794	28.3333	12.2223	110	0.00226914
ash85	85	9	10.3333	2.2452	10	2.55389	15	0.00125526
nos3	960	44	181	297.752	81	0.125799	81	0.117394
ash292	292	19	22.3333	76.4161	22	148.146	33	0.0136006
bcsstk05	153	20	20.6667	23.3754	21.3333	19.2524	26	0.00421739
ibm32	32	11	11	0.19647	11	0.177752	28	0.000122744

Tabela 2: Rezultati na odabranim matricama iz skupa Harwell-Boeing

Pošto je vreme izvršavanja algoritma simuliranog kaljenja ograničavano na 300 sekundi, za dosta instanci algoritam ne strigne da se izvrši do kraja (tj. temperatura T ne dostigne T_f), pa se samim tim često i ne dobijaju vrednosti za bandwidth koje su dostignute u originalnom radu (tamo se algoritam izvršavao znatno kraće zbog efikasnije implementacije i malo boljeg hardvera).

Ako se uporede rezultati algoritma sa nasumičnim i pohlepnim inicijalnim rešenjem, može se primetiti da simulirano kaljenje sa BFS pretragom na početku nekada daje bolje rezultate, zbog toga što ima početnu prednost, a vreme izvršavanja je ograničeno, pa običan DRSA ne uspeva da dostigne dovoljno dobro rešenje u zadatom roku. U nekim drugim slučajevima, BFS pretraga na početku ne pomaže i čak daje lošije rezultate. To je najverovatnije zbog toga što pohlepni algoritam daje rešenja takve strukture da je potrebno dosta perturbacija da bi se iz njih došlo do boljih rešenja, tj. rešenja dobijena pohlepnim algoritmom mogu predstavljati jak lokalni minimum iz kog je teško izaći. To se najbolje može videti na primeru matrice *nos*3. U ovom primeru DRSA sa BFS pretragom uspeva da da znatno bolje rešenje nego običan DRSA za to vreme, ali to

je zapravo rešenje koje je dao sam pohlepni algoritam, tj. DRSA za sve dato vreme nije uspeo da izadje iz lokalnog minimuma.

Pohlepni algoritam, naravno, u većini slučajeva daje lošija rešenja nego simulirano kaljenje, mada često uspeva da značajno smanji početni bandwidth. To dosta zavisi od strukture matrica; kod nekih asimetričnih matrica algoritam može i da pokvari početni bandwidth. U nekim slučajevima, pošto je izvršavanje DRSA bilo ograničeno, BFS pretraga je uspela da da bolje rešenje. Glavna prednost pohlepnog algoritma je njegova efikasnost: za sve matrice iz test primera, algoritam se izvršava za manje od jedne sekunde.

6 Zaključak

Tokom godina predloženi su mnogi pristupi za rešavanje problema minimalnog bandwidth-a grafa. Ovaj rad je pisan sa idejom da se implementira algoritam simuliranog kaljenja iz rada [5].

Takođe je isprobano korišćenje pohlepnog algoritma za inicijalno rešenje. Ovakav pristup možda može biti od koristi kada je važnije vreme izvršavanja od nalaženja optimalnog rešenja. Korišćenje pohlepnog algoritma pri inicijalizaciji može dati početnu prednost, a i omogućava da glavni algoritam u slučaju prerano prekida zbog isteka vremena može da ima rešenje koje je dovoljno dobro za neku primenu. Ako je važnije nalaženje najboljeg mogućeg rešenja, korišćenje pohlepnog algoritma može dovesti do jakog lokalnog minimuma i dodatno otežati rad algoritma, pa je verovatno bolje koristiti podrazumevanu verziju algoritma.

Ono što bi se dalje moglo unaprediti jeste efikasnost samog algoritma, kao i moguće dodavanje još nekog operatora za menjanje rešenja koji bi omogućavao lakše izlaženje iz lokalnih minimuma.

Literatura

[1] Matrix Market maintained by the Mathematical and Computational Sciences Division of the Information Technology Laboratory of the National Institute of Standards and Technology (NIST). http://math.nist.gov/MatrixMarket/data/Harwell-Boeing/.

[2] Guilherme Oliveira Chagas and Sanderson L. Gonzaga de Oliveira. Metaheuristic-based heuristics for symmetric-matrix bandwidth reduction: A systematic review. *Procedia Computer Science*, 2015.

[3] S. L. Gonzaga de Oliveira and L. M. Silva. An ant colony hyperheuristic approach for matrix bandwidth reduction. *Applied Soft Computing*, pages 2–4, 2020.

[4] N. Mladenovic, D. Urosevic, D. Perez-Brito, and C. G. Garcia-Gonzalez. Variable neighbourhood search for bandwidth reduction. *European Journal of Operational Research*, 2010.

[5] J. Torres-Jimenez, I. Izquierdo-Marquez, A. Garcia-Robledo, A. Gonzalez-Gomez, J. Bernal, and R.N. Kacker. A dual representation simulated annealing algorithm for the bandwidth minimization problem on graphs. *Information Sciences*, 2015.