Problem maksimalne klike

Maximum clique

Seminarski rad u okviru kursa

Računarska inteligencija

Matematički fakultet

Luka Jovanović, 197/2016

Nevena Mesar, 107/2015

Sažetak

Optimizacija problema pronalaženja maksimalne klike u grafu genetskim algoritmom. Pronalaženje brzog i prihvatljivog rešenja za različite grafove.

Sadržaj

1 Uvod

Problem koji se rešava je NP-težak problem. Obzirom da je brute force algoritam vremenski zahtevan, primenjuju se različite metode optimizacije za rešavanje. Odlučili smo se da predstavimo brute force algoritam i rešenje dobijeno integracijom genetskog algoritma. Genetski algoritmi su razvijeni po ugledu na prirodne procese. Koriste se za optimizacije funkcija, obrade slika, rešavanje poznatih problema kao što je problem trgovačkog putnika i slično.

2 Brute force algoritam sa odsecanjem

Nadogradnja na brute force, efikasan za jednostavne grafove.

```
1000 def intersection(a: list, b: list):
         return [value for value in a if value in b]
1002
1004 def brute_force_clique_(G:dict, U:list, size, nodes:list):
         global max, clique_nodes
1006     if len(U) == 0:
             max = size
1008         clique_nodes = nodes
             return
1010
         while len(U) != 0:
1012         if size + len(U) <= max:
                 return
1014         i = U[0]
             nodes.append(i)
1016         U.remove(i)
             brute_force_clique_(G, intersection(U,G[i]), size+1,nodes[:])
1018
1020 def brute_force_clique(G:dict):
         brute_force_clique_(G, list(G.keys()), 0, [])
```

Listing 1: Glavni deo koda

3 Optimizacija genetskim algoritmom

3.1 Strukture

Za lakšu implementaciju koristili smo par klasa.

- `Node` – informacije o stepenu čvora i skupu suseda.
- `SortedListNode` – informacije o čvoru i dostupnosti ostalim čvorovima iz njega (reach)
- `Graph` – matrica susedstva, informacija o susednim čvorovima svakog čvora grafa, olakšava sortiranje po stepenu čvora. Karakteristične metode su:
  - `add_edge` : dodaje se u matricu susedstva, za svaki čvor se računa stepen i dodaje mu se sused u listu, svi čvorovi koji nisu već obrađeni smeštaju se u listu sortiranih čvorova
  - `sortList` : skup sortiranih čvorova se ažurira tako da bude opadajući prema stepenu čvora
  - `visualize` : korišćenjem networkx i matplotlib biblioteka
- `Clique` – klasa za cuvanje kombinacija klika, svaka ima skup kandidata koji mogu da se uključe u kliku. Karakteristične metode su:
  - `init` : za početni čvor se konstruiše klika, za svaki od čvorova grafa ako u matrici susedstva grafa postoji grana dodaje se u listu kandidata za kliku

- `add_vertex` : ako već nije u listi klike briše se iz liste kandidata i prebacuje u kliku, onda se ostali kandidati uklanjaju ako ne postoji veza(grana) sa novoubačenim čvorom
- `remove_vertex` : ako postoji u kliki brise se iz skupa kandidata, za svaki čvor grafa koji nije već u kliki ako postoji veza sa svim čvorovima unutar klike dodaje se u skup kandidata
- `compute_sorted_list` : svakom od kandidata se ažurira reach ka ostalim kandidatima, takvi Node-ovi se dodaju u listu koja se sortira opadajuće prema reach-u

3.2 Gradivni elementi genetskog algoritma

Populaciju čini lista klika dobijenih tako tako što je početni čvor odabran random i za njega pronađena najbolja klika. Ukupan broj klika u populaciji je 10. Selekcija je obično random izvlačenje dve klike iz populacije. Prioritizuju se čvorovi koji imaju najveće stepene i domete.

3.2.1 Ukrštanje

Za dve klike se nađe presek čvorova i uzme se prvi odatle, bez ikakvog sortiranja čvorova. Ako nema preseka vrača se klika dobijena pohlepnim ukrštanjem, tako da se konstruiše potpuno nova klika od najboljeg čvora iz spojene liste čvorova te dve klike.

Za svaki od čvorova preseka se proverava da li pripadaju listi kandidata i ako je to slučaj dodaju se u kliku. Nakon toga se za čvorove sortirane liste tako dobijene klike proverava da li među kandidatima klike i dodaju ako jesu.

Za dalje testiranje i razvijanje algoritma u ovom koraku neki od predloga bi bili da se sortiraju čvorovi i uvek uzima najbolji.

3.2.2 Mutacija

Bira se predefinisan broj čvorova (1) koji mutiraju i uklanjaju se iz skupa čvorova klike. Dalje se ili formira ponovo opadajuće sortirana lista čvorova za ovako izmenjenu kliku i na osnovu najboljeg čvora dopunjuje klika ili će se iz skupa mogućih čvorova dodavati u kliku dok je to moguće.

3.2.3 Lokalno poboljšanje

Ideja je da se pruži prilika čvorovima klike da budu zamenjeni za neke od ostalih kandidata koji bi ušli u kliku umesto njih. Ovo je operacija koja sme da se izvrši predefinisan broj puta, u nasem slučaju je to 10.

Kada se obrade svi kandidati modifikovane klike, poredi se da li je njena veličina bolja od početne i vraća se izmenjena klika ukoliko je to slučaj.

3.3 Koraci genetskog algoritma

- Učitavanje grafa u globalnu promenljivu iz nekog od test fajlova
- Generisanje random populacije, njeno sortiranje po veličini klike opadajuće tako da je najbolja klika prva u listi
- Pamćenje globalne najbolje klike i njene veličine
- Kroz iteracije(generacije)
  - Ako je veličina prethodne globalne ista kao trenutna najbolja generiše se nova random populacija i resetuje seed za RNG, moguće je ponoviti ovaj postupak dok se ne dostigne SHUFFLE_TOLERANCE(10), inače se ažurira veličina prethodno najbolje klike

- Trenutna populacija se sortira opadajuće i njena prva klika se bira za najbolju lokalnu, ako je ona bolja od globalne, globalna se ažurira
- Radi se lokalno pobiljšanje globalne najbolje i tako ažurirana se dodaje u novu populaciju. Ovo je korak u kome se jasno vidi elitizam, propagira se u naredne generacije
- Popunjavamo ostatak mesta u novoj populaciji dok možemo tako što koristimo pomenute metode za selekciju i ukrštanje, radimo lokano poboljšanje za kliku dobijenu ukrštanjem, i ako kao takva nije bolja od makar jedne početne klike mutira se
- Zamenjujemo populaciju sa novom

4 Podaci i učitavanje

Podaci su preuzeti sa dimacs benchmark set, izdvojili smo par instanci za test. Skup koji je u ponudi je dat u tabeli 1. Istaknute su instance korišćene za masovno testiranje i za koje je program najčešće pokretan prilikom testiranja.

Za više informacija o instancama i njihovm familijama pogledati literaturu

Implementirana je funkcija koja direktno učitava podatke iz fajlova u strukturu grafa za koju smo se opredelili.

4.1 Pokretanje programa

Na graficima su plavom bojom označene grane koje pripadaju pronađenoj klili, a crnom postojeće grane u grafu. Imati na umu da grafovi imaju veliki broj grana i da zbog toga deluje da je obojena pozadina.

Program je napravljen tako da moze da primi i parametre na tri načina. Prvi je da se program pokrene bez argumenata, onda se pokreće masovna obrada za sve predefinisane grafove redom. Treći način je da se navede koji fajl treba da se učita i za koliko se iteracija(generacija) pokreće obrada. Može se uneti indeks grafa koji treba da se učita i za koji se obrada pokreće. Trenutno je predefinisano 5 grafova koji su označeni u tabeli 1.

Moguće je čuvati generisane klike u `.png` formatu ukoliko je kreiran folder na putanji sa koje se program pokreće. Naziv foldera je moguće promeniti u kodu. Ukoliko ne postoji folder, izvoz slika će se preskočiti.

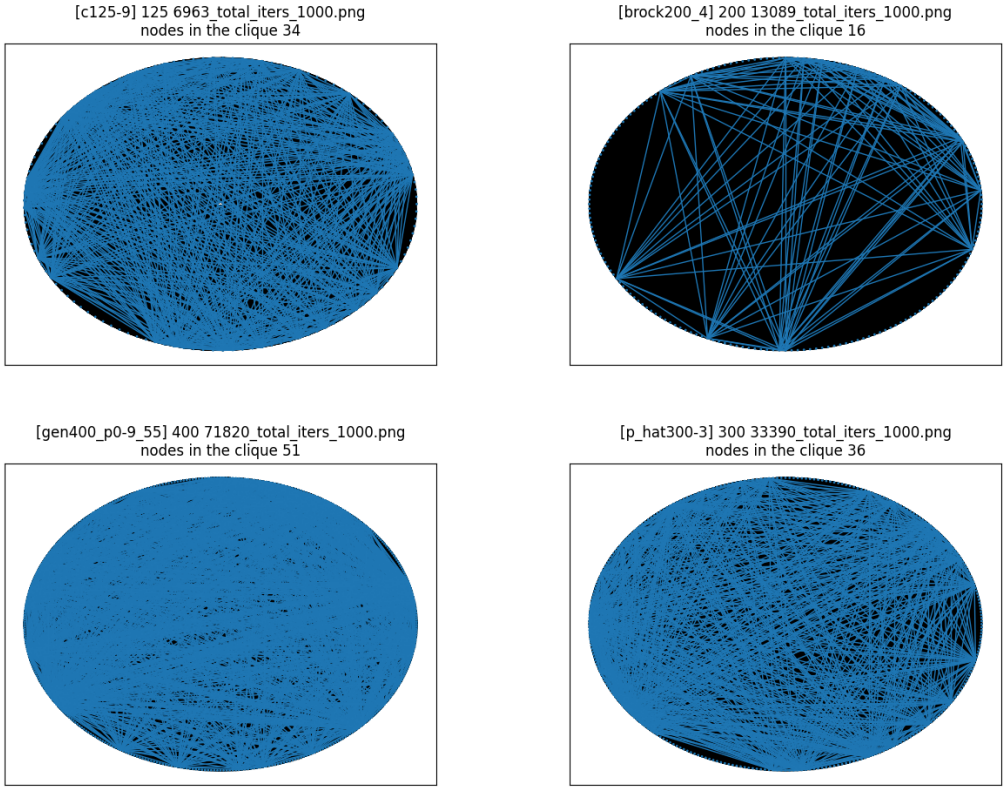
Primer jedne masove obrade predstavljen je u 4.2

Tabela 1: Spisak DIMACS grafova po familijama i instancama

instanca	poznato rešenje	broj čvorova	broj grana
C125.9	34	125	6 963
C250.9	44	250	27 984
C500.9	57	500	112 332
C1000.9	68	1 000	450 079
C2000.9	80	2 000	1 799 532
DSJC1000_5	15	1 000	499 652
DSJC500_5	13	500	125 248
C2000.5	16	2 000	999 836
C4000.5	18	4 000	4 000 268
MANN_a27	126	378	70 551
MANN_a45	345	1 035	533 115
MANN_a81	1 100	3 321	5 506 380
brock200_2	12	200	9 876
brock200_4	17	200	13 089
brock400_2	29	400	59 786
brock400_4	33	400	59 765
brock800_2	24	800	208 166
brock800_4	26	800	207 643
gen200_p0.9_44	44	200	17 910
gen200_p0.9_55	55	200	17 910
gen400_p0.9_55	55	400	71 820
gen400_p0.9_65	65	400	71 820
gen400_p0.9_75	75	400	71 820
hamming10-4	40	1 024	434 176
hamming8-4	16	256	20 864
keller4	11	171	9 435
keller5	27	776	225 990
keller6	59	3 361	4 619 898
p_hat300-1	8	300	10 933
p_hat300-2	25	300	21 928
p_hat300-3	36	300	33 390
p_hat700-1	11	700	60 999
p_hat700-2	44	700	121 728
p_hat700-3	62	700	183 010
p_hat1500-1	12	1 500	284 923
p_hat1500-2	65	1 500	568 960
p_hat1500-3	94	1 500	847 244

4.2 Primer ispisa programa za MUTATIONS = 1, DEFAULT_TOTAL_ITERATIONS = 500

```
--------------------------
Started working on: C:...[c125-9] 125 6963.txt

Generating initial population...
Found clique with 34 nodes in 8.62 sec:
[53, 113, 124, 98, 6, 10, 48, 79, 28, 8, 33, 95, 67, 103, 116, 30, 43, 51, 102,
109, 121, 0, 54, 24, 69, 4, 18, 39, 44, 65, 76, 78, 12, 97]

--------------------------
Started working on: C:...[brock200_4] 200 13089.txt

Generating initial population...
Found clique with 15 nodes in 12.20 sec:
[2, 71, 32, 39, 94, 90, 192, 52, 89, 7, 10, 30, 0, 19, 75]

--------------------------
Started working on: C:...[gen400_p0-9_55] 400 71820.txt

Generating initial population...
Found clique with 51 nodes in 59.27 sec:
[19, 68, 268, 17, 198, 10, 97, 104, 65, 105, 146, 320, 60, 338, 159, 18, 236, 155, 49,
239, 259, 25, 84, 215, 263, 108, 357, 89, 312, 366, 126, 187, 346, 387, 55, 133, 193,
234, 265, 385, 64, 72, 176, 209, 383, 394, 90, 185, 295, 389, 395]

--------------------------
Started working on: C:...[p_hat300-3] 300 33390.txt

Generating initial population...
Found clique with 35 nodes in 23.70 sec:
[75, 218, 204, 254, 267, 258, 48, 286, 289, 3, 238, 118, 32, 246, 298, 18, 198, 148,
20, 55, 97, 175, 221, 39, 19, 17, 137, 42, 88, 292, 47, 173, 189, 234, 251]
```



Slika 1: Grafici dobijeni za prethogdni output programa

5 Zaključak

Optimizacija genetskim algoritmom koju smo prikazali je efikasna u velikom broju slučajeva, dovoljno brza i za velike grafove.

Za visoke vrednosti iteracije nije pokazao poboljšanje, tako da u ovom obliku nije vredno testirati ostale instance. Moglo bi da se pronađe par načina za modifikaciju koja bi možda obogatila algoritam da bude efikasniji za više iteracija. S obzirom na to da daje zadovoljavajuće rezultate za potrebe testiranja ostavljamo tu temu čitaocima.

Za grafove iz `p-hat` i `dsjc` familija najbolje se pokazao za parametre sa oko 1000 iteracija i mutacije 3 čvora, dostignut je poznat broj čvorova, uz minimalno variranje vremena potrebnog za izvršenje.

Za ostale prikazane familije bilo je dovoljno 500 iteracija za najbolje rešenje koje može da se dostigne ovom verzijom algoritma.

Literatura

- DIMACS benchmark set
- instance families and generators
- NetworkX