Resavanja problema presecnog podgrafa

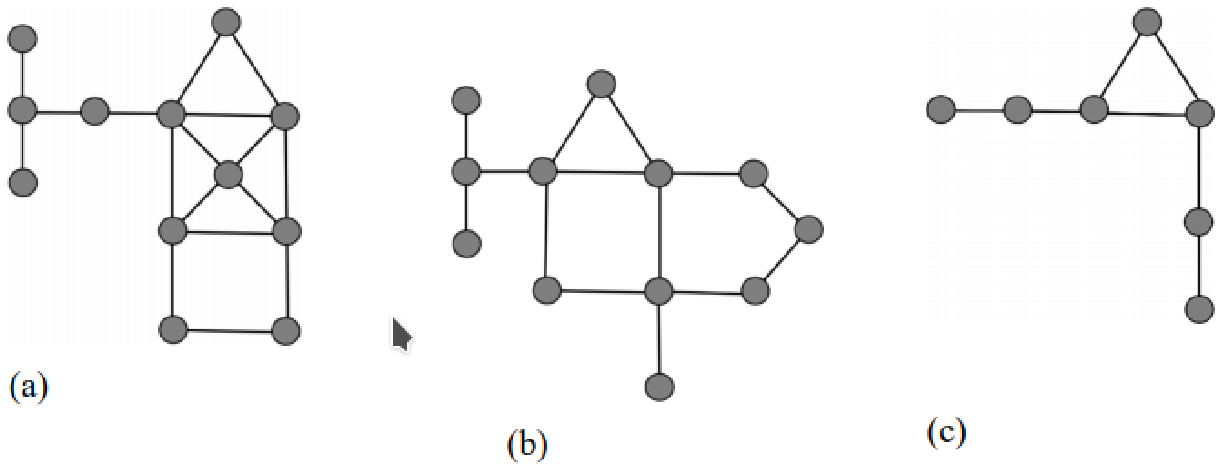
Mateja Marjanovic, 172/2015

Januar 2020

Sadrzaj

1 Uvod

Za graf G'=(V', E') kazemo da je podgraf grafa G=(V, E), ako vazi da je V' podskup od V I E' podskup od E, tj. skup grana I cvorova grafa G' je podskup skupa grana I cvorova grafa G.
Presecni podgraf dva grafa je graf koji je podgraf I od jednog I od drugog grafa. Uzmimo grafove G1 I G2 I neka su njihovi podgrafovi G1' I G2', presecni podgraf G1 I G2 bice G1' odnosno G2' ako su oni izomorfni. Za grafove kazemo da su izomorfni ako postoji '1-1' preslikavanje iz cvorova V1 u V2 I ivica E1 u E2.
Indukovani podgraf G' je podgraf grafa G u kome su sve ivice koje povezuju cvorove u V' prisutne.
Podgraf indukovan ivicama je slicno tako graf u kome za date ivice, svi cvorovi su ukljuceni.
Pokrice cvorovima (vertex cover) C je podskup cvorova takav da za sve grane (u, v) iz E, vazi u iz C ili v iz C. Slican pojam je I nezavisan skup koji predstavlja skup cvorova u kojoj nikoje dve nisu susedne. Cvorovi koji nisu iz pokrica cvorova (vertex cover) su iz nezavisnog skupa.



Na slici su prikazana tri grafa, prvi I drugi su nezavisni jedan od drugog, dok treci predstavlja njihov maskimalni presecni podgraf.

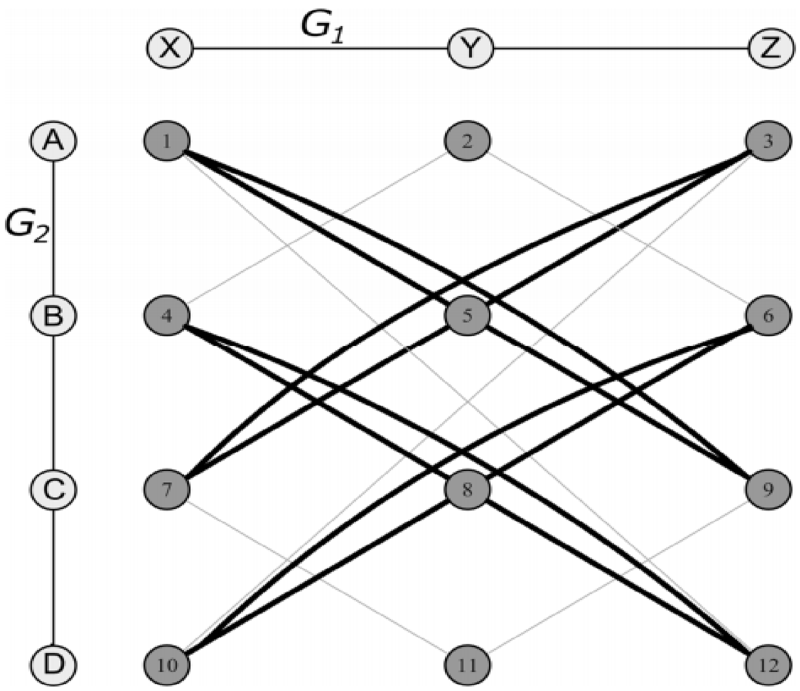
2 Opis resenja zadatog problema

Klika (clique) je skup cvorova grafa takvih da su svaka dva cvora koja pripadaju toj kliki medjusobno povezani. Razlikujemo maximal I maximum klike. Maximal klika predstavlja kliku kojoj ako dodamo neki cvor iz grafa, vise nece biti klika. Maximum klika je klika koja ima maksimalan broj cvorova (klika moze imati I dva povezana cvora), tj. u tom grafu se ne moze pronaci klika sa vise elemenata.

Modularni proizvod je ‘proizvod‘ dva grafa I kao rezultat daje graf ciji je skup cvorova V1xV2, grafova G1 I G2. Ivica izmedju dva cvora (u1, v1) I (u2, v2) postoji ako je (u1, u2) iz E1, a (v1, v2) iz E2 ili ne postoji (u1, u2) u E1 I ne postoji (v1, v2) u E2.
Maksimalna klika modularnog proizvoda predstavlja presecni podgraf.

2.1 Modularni proizvod

Kao sto je gore napisano modularni proizvod grafa G1=(V1, E1) I grafa G2=(V2,E2) je graf ciji su cvorovi parovi iz V1xV2. Dva cvora su povezana ako su prvi elementi susedni u G1 I drugi elementi susedni u G2, ili ako ni prvi elementi nisu susedi u V1, ni drugi elementi nisu susedi u V2.

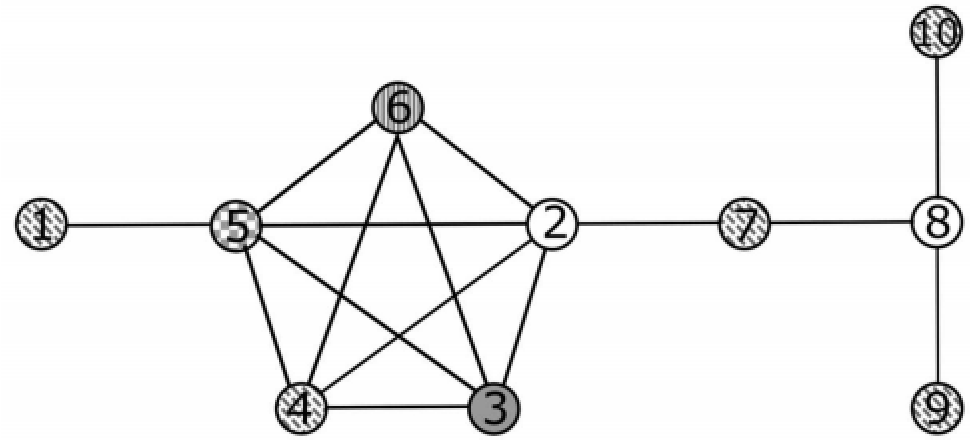


Iz tog razloga su 1 I 5 susedi u modularnom proizvodu, zato sto je A sused od B u G2, a X sused od Y u G1. Isto tako 1 I 9, ni (X, Z) ne pripada E1, ni (A, C) ne pripada E2.
Crne linije su grane klike, a sive nisu.
Na slici vidimo 4 klike, to su (1, 5, 9), (4, 8, 12), (7, 5, 3) I (10, 8, 6).
Problem sa modularnim proizvodom je sto ume da bude jako velik, ali jos vaznije, da ima veliku gustinu grana. To je jedan od razloga zasto resavanje ovog problema trosi puno vremena.

2.2 Bojenje grafa

Kada kazemo bojenje grafa, mislimo na dodeljivanje boje svakom cvoru, tako da nikoja dva susedna cvora nemaju istu boju. Ovo je NP-kompletan problem koji se izrodio zeljom da se oboje drzave na mapi, tako da nikoje dve nemaju istu boju.
Najmanji broj boja potrebnih da se oboji graf naziva se hromatski broj.

Ovo je bitno iz razloga sto klika ce imati isti broj boja I cvorova koji joj pripadaju, tj. nijedan cvor nece imati istu boju kao neki drugi cvor iz klike. Iz tog razlog lako se primeti klika kada svi susedi od nekog cvora imaju razlicite boje. To je jedna od osnova za implementiranje nekih od algoritama za detekciju klike.



Graf sa slike ima kliku (2, 3, 4, 5, 6) I kao sto se moze primetiti, svaki od njih je obojen drugom bojom.

2.3 Algoritmi zasnovani na detekciji klike

2.3.1 Gruba sila

Sa ω(G) cemo oznacavati maksimum klike grafa G, sa $N_G(v)$ skup suseda cvora v u grafu G. Problem detekcije klike moze se resiti bektrekingom (backtracking), tako sto se zada relativno jednostavna rekurzivna formula:

$$\omega(G) = \max\{1 + \omega(NG(v)), \omega(G \backslash v)\}$$

Bira se maksimum dva broja, jedan je maksimum klike suseda od v uvecano za jedan (sam cvor v), dok je drugi maksimum klike grafa G, ako iz njega obrisemo cvor v.

Ulaz: graf, trenutna klika, maksimalna klika
Izlaz: maksimalna klika

dok je G neprazan
ponavljaj
    izaberi cvor iz grafa
    dodaj ga u trenutnu kliku
    rekurzivno pozovi funkciju za susede od v
    ako je kardinalnost trenutne klike veca od kardinalnosti maksimalne, maksimalnoj se dodeljuje trenutna
    izbaci v iz G

2.3.2 Bron-Kerbosch

Algoritam slican gruboj sili, sa par primetnih poboljsanja. Jedno od tih poboljsanja je sto cuva informaciju o tome koji je cvor ‘obidjen’, pa se ne gubi informacija I ne trosi toliko vremena. Drugo, pronalazi sve maximal klike, a ne samo jednu.
Modifikacija bira pivot iz unije P I X I bira takve elemente klike, tako da ne budu susedi od pivota.

Ulaz: cvorovi u kliki, trenutni cvorovi, korisceni cvorovi
Izlaz: maximal klika

```
ako je su I trenutni I korisceni cvorovi prazni
        nasli smo jednu maximal kliku
inace
        kreci se kroz trenutne cvorove
                u rekurzivnom pozivu dodajemo trenutni cvor u trenutnu kliku, a
                nad trenutnim I koriscenim cvorovima vrsimo presek sa susedima
                trenutnog cvora
                dodajemo ga u trenutni cvor
                dodajemo ga u koriscene cvorove
```

2.3.3 Branch-and-Bound pristup

U ovom algoritmu cemo koristiti grananje sa ogranicenjem (branch and bound) u kombinaciji sa minimalnim bojenjem grafa (minimalan broj boja) I naravno modularnim proizvodom. Pri bojenju grafa, umesto boja, cvorovima cemo dodeljivati cele brojeve.
Znamo da vazi $\omega \leq k$, gde je ω maksimum klike, a k najmanji broj boja kojim se moze obojiti graf.
Slicno kao I za kliku, trazenje minimalnog broja boja za bojenje grafa je NP-tezak problem, medjutim koriscenjem pohlepnih tehnika cemo napraviti pohlepno bojenje u polinomijalnom vremenu.

```
maxClique :: (Graph G) → Set
begin
        global Cmax ← ∅
        expand(G, ∅, V(G))
        return Cmax

expand :: (Graph G, Set C, Set P)
begin
        colourClasses ← colour(G, P)
        while colourClasses != ∅ do
                colourClass ← select(colourClasses)
                for v ∈ colourClass do
                        if |C| + |colourClasses| ≤ |Cmax| then return
                        P1 ← P ∩ N(G, v)
                        C ← C ∪ {v}
                        if |C| > |Cmax| then Cmax ← C
```

```
                if P1 != ∅ then expand(G, C, P1 )
                C ← C \ {v}
        colourClasses ← remove(colourClasses, colourClass)
```

Dakle, algoritam krece tako sto imamo globalnu Cmax, koja nam govori o najboljoj kliki koju smo do sad nasli, na kraju izvrsavanja ce se u njoj nalaziti maximal klika.
Prvo obojimo graf G I to upisemo u listu listi colourClass, zatim odaberemo neku od listi iz te liste (odabreremo boju), a onda se krecemo kroz elemente te liste (kroz elemente iste boje). Ako je zbir kardinalnosti klike koju trenutno gradimo I ukupnog broja boja koje koristimo manji ili jednak od kardinalnosti do sad najbolje klike (Cmax), onda nema smisla ici dalje tim putem, jer cemo u najboljem slucaju dobiti kliku iste velicine, a mozda I manju.
Cvorove koje smo koristili redukujemo tako sto izvrsimo presek sa susedima od trenutnog cvora, a u kliku koju trenutno gradimo ubacujemo taj cvor.
Ako je ostalo jos neobradjenih susednih elemenata, rekurzivno pozivamo za kliku koju gradimo I za te susedne elemente.
Nakon toga izbacujemo trenutni cvor iz klike, kako bismo videli da li je bolje bez tog cvora (ako je sa tim cvorom optimalno, to ce se I pokazati u Cmax).
Nakon sto smo ispitali za sve elemente koji su obojeni jednom bojom, vise nema smisla za njih dalje ispitivati bilo sta, pa tu ‘klasu boja’ mozemo I da eliminisemo.

```
colour :: (Graph G, Set P) → List of Set
begin
    colourClasses ← ∅
    uncoloured ← copy(P)
    while uncoloured != ∅ do
        current ← ∅
        for v ∈ uncoloured do
            if current ∩ N(G, v) = ∅ then current ← current ∪ {v}
        uncoloured ← uncoloured \ current
        colourClasses ← append(colourClasses, current)
    return colourClasses
```

Inicijalno klasa boja je prazan skup I niko nije obojen, tj. svi su neobojeni. Krecemo se kroz niz neobojenih cvorova I biramo neki cvor I ubacujemo ga u niz current. U taj niz cemo u narednim iteracijama (u unutrasnjoj for petlji) ubaciti jos cvorova koji medjusobno nisu susedni jedan drugom. Sada imamo niz nesusednih elemenata I to znaci da njima moze da se dodeli boja, pa vise nema razloga da oni budu u nizu neobojenih. Dodeljujemo im boju. Tako radimo dokle god ima cvorova kojima nije dodeljenja nijedna boja.

3 Uporedjivanje sa drugim resenjima

Hardver na kome je izvrseno testiranje, Intel(R) Core(TM) i3-3217U CPU @ 1.80GHz, 3.7GB RAM na operativnom sistemu Ubuntu.

Iz datih tabela se moze primetiti da je Bron-Kerbosch mnogo sporiji od Branch and Bound-a. Tesitranje je vrseno na grafovima velicine 5 I 6 (to postaju grafovi velicine 25 I 36).

Input, n = 6	BK	BB
input_15	2s 286136us	0s 3479us
Input_16	3s 285679us	0s 1343us
Input_17	2s 292366us	0s 1278us
input_18	2s 287426us	0s 1273us
input_19	2s 290111us	0s 1275us
input_20	3s 297877us	0s 1294us
input_21	2s 289097us	0s 1277us
input_22	2s 305941us	0s 1271us
input_23	3s 297879us	0s 1302us
input_24	2s 275423us	0s 1270us

Input, n = 5	BK	BB
input_15	0s 80093us	0s 4709us
input_16	1s 86400us	0s 4934us
input_17	0s 97194us	0s 4495us
input_18	0s 77409us	0s 4605us
input_19	0s 83481us	0s 5015us
input_20	0s 70221us	0s 4543us
input_21	0s 80291us	0s 4619us
input_22	0s 77245us	0s 4799us
input_23	0s 75727us	0s 5517us
input_24	0s 112465us	0s 9125us

4 Zakljucak

U ovom radu bila su razmatrana dva pristupa problemu pronalazenja presecnog podgrafa, Bron-Kerbosch algoritam, koji je sporiji I jednostavniji I Branch and Bound, koji je osetno brzi, ali I malo komplikovaniji za razumevanje I implementaciju. Postoji jos nacina da se resi problem pronalazenja presecnog podgrafa I sigurno ima boljih I od Branch and Bound pristupa, ali on se pokazao kao dobro resenje I pored toga sto su grafovi koji se ispituju u ovom problemu imaju veoma gusto rasporedjene grane.

5 Literatura

[1] Maximum Common Subgraph Isomorphism Algorithms Edmund Duesbury1 , John D. Holliday1 , Peter Willett2 1UCB Pharma, Slough, United Kingdom 2 Information School, University of Sheffield, Sheffield, United Kingdom
[2] Solution of Maximum Clique Problem by Using Branch and Bound Method Mochamad Suyudi1 , Ismail Bin Mohd2 , Mustafa Mamat3, 6 , Sudrajat Sopiyan4 and Asep K. Supriatna5 1,4,5 Department of Mathematics, University of Padjadjaran, Indonesia. 2,3 Research Fellow, Department of Mathematics Universiti Malaysia Terengganu, Malaysia 6 Faculty of Informatics and Computing Universiti Sultan Zainal Abidin, Terengganu, Malaysia
[3] Reducing the Branching in a Branch and Bound Algorithm for the Maximum Clique Problem Ciaran McCreesh and Patrick Prosser University of Glasgow, Glasgow, Scotland