Graphs of type (SB) and domination on their cartesian products

Jan Hrastnik, Matic Kremžar

11. 12. 2024

1 Uvod

V projektni nalogi sva se ukvarjala z grafi tipa (SB) in dominacijo na njihovem kartezičnem produktu. Projektna naloga je bila izvedena v programu SageMath, obsežni izračuni pa s pomočjo spletne strani CoCalc, in predvsem lokalno na Linux-u. Naloga ima odprt tudi svoj repozitorij na GitHubu, na naslovu (tukaj). Tam so zbrane tudi vse datoteke, ki sva jih uporabila med izdelavo projekta.

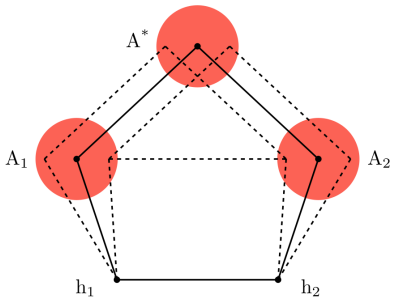
Cilj je razumeti, kako prepoznati, konstruirati in spreminjati grafe tipa (SB), ter analizirati njihove lastnosti, zlasti dominacijsko število njihovih kartezičnih produktov.

2 Osnovna ideja in definicije

Delala sva z usmerjenimi grafi $G = (V, E)$, kjer je V množica vozlišč, E pa množica povezav grafa.

Definicija 1. *Graf G je tipa (SB), če je njegov premer enak 2 in ima dve sosednji vozlišči $v_1, v_2 \in V$, da velja:*

- *v_1 in v_2 nimata skupnega soseda,*
- *G ima vozlišče $v^* \in V$, ki ni sosednje v_1 ali v_2; torej $v^* \not\sim v_1$ in $v^* \not\sim v_2$.*



Slika 1: Skica grafov tipa (SB)

Zapišemo lahko particijo vozlišč grafa G kot

$$V(G) = v_1, v_2 \cup A_1 \cup A_2 \cup A^*,$$

kjer je A_1 množica vozlišč, ki so sosednja v_1, A_2 množica vozlišč, ki so sosednja v_2 in A^* množica vozlišč, ki niso sosednja niti v_1 niti v_2.

Definicija 2. *Podmnožica vozlišč $D \subseteq V$ grafa $G = (V, E)$ se imenuje* dominacijska množica *grafa, če je vsako vozlišče $v \in V$ grafa v množici D, ali pa je kakšno njemu sosednje vozlišče v D.* Dominacijsko število $\gamma(G)$ *grafa je velikost najmanjše dominacijske množice grafa.*

Definicija 3. *Kartezični produkt $G\Box H$ grafov G in H je graf, za katerega velja:*

- *vozlišča grafa $G\Box H$ so kartezični produkt $V(G) \times V(H)$,*
- *dve vozlišči (u, v) in (u', v') sta sosednji v grafu $G\Box H$ natanko tedaj, ko je:*
  - *$u = u'$ in v je sosed v' v H,* ***ali***
  - *$v = v'$ in u je sosed u' v G.*

3 Naloga 1

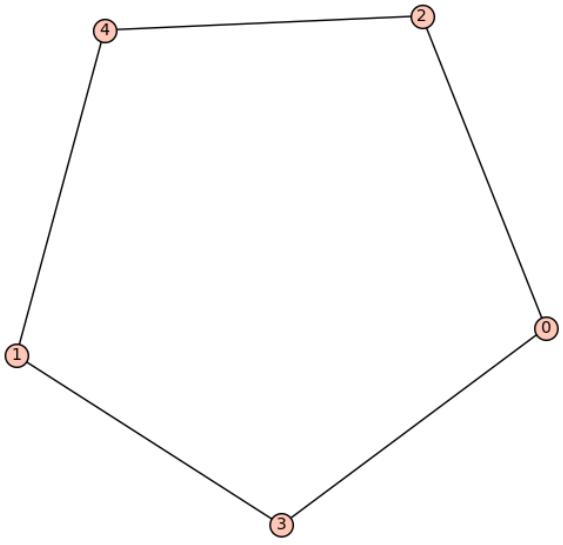
3.1 Preverjanje, ali je graf tipa (SB)

Sprva je potrebno zapisati kodo, ki preveri, ali je dan graf tipa (SB) (v datoteki *sb_detect_koncna.py*). Uporabimo matriko sosednosti. Dovolj je, da pregledamo zgolj zgornji trikotnik matrike, saj tako že pridobimo vse podatke o sosedih. Nato preverimo še veljavnost prej zapisanih lastnosti grafov tipa (SB). Najina metoda za preverjanje, ali je graf tipa (SB), ima časovno zahtevnost $O(|V|^3)$.

3.2 Iskanje vseh grafov tipa (SB) na do 10 vozliščih

Nalogo sva si razdelila na 2 dela: prvi je iskanje grafov s premerom 2, in drugi je iskanje grafov tipa (SB) znotraj najdenih grafov s premerom 2. Najprej sva v datoteko *diam_two_graphs.txt* zbrala vse grafe s premerom 2 na do 10 vozliščih. To sva storila zato, ker s SageMath lahko dosti hitreje preverimo premer grafa, kot pa če je ta graf tipa (SB). S tem sva si zmanjšala množico kandidatov za grafe tipa (SB).

Nato sva te grafe prefiltrirala s spodnjo kodo in podatke sprva zbrala v novi datoteki, ki je bila velika približno 400 MB. Grafe sva shranjevala v formatu slovarja. Kasneje sva spoznala, da lahko grafe bolj kompaktno shranimo v tako imenovan *graph6* format. S tem sva zmanjšala velikost na le 20MB.



Slika 2: Graf tipa (SB) na 5 vozliščih

Algorithm 1 Preverjanje, ali je graf tipa (SB)

```
function IsSB(G: Graph)                                   ▷ Funkcija za preverjanje grafa tipa (SB)
    if G.diameter() ≠ 2 then return False                 ▷ Premer mora biti 2
    adj ← G.adjacency_matrix()                            ▷ Pridobimo matriko sosednosti
    for i ← 1 to adj.nrows() do
        for j ← i + 1 to adj.nrows() do                   ▷ Samo zgornji trikotnik
            if adj[i, j] = 1 then                         ▷ Če sta i in j soseda
                common_neighbour ← False
                for k ← 1 to adj.nrows() do
                    if adj[i, k] = 1 and adj[j, k] = 1 then   ▷ Preverimo skupne sosede
                        common_neighbour ← True
                        break
                    end if
                end for
                if common_neighbour then
                    continue
                end if
                nonadj_vertex_exists ← False
                for k ← 1 to adj.nrows() do
                    if adj[i, k] = 0 and adj[j, k] = 0 and k ≠ i and k ≠ j then
                        nonadj_vertex_exists ← True
                        break
                    end if
                end for
                if nonadj_vertex_exists then
                    return True
                end if
            end if
        end for
    end for
    return False                                          ▷ Graf ni tipa (SB)
end function
```

Algorithm 2 Filtriranje grafov tipa (SB)

```
Odpri datoteko 'sb_graphs.txt' v načinu dodajanja ("a").
Odpri datoteko "diam_two_graphs.txt" v načinu branja ("r").
for vsako vrstico line v datoteki "diam_two_graphs.txt" do
    g ← pretvori_v_graf(line)          ▷ Ustvari graf iz podatkov na trenutni vrstici.
    if is_sb(g) then                   ▷ Preveri, če je graf g tipa (SB).
        Zapiši line v datoteko 'sb_graphs.txt'.
    end if
end for
Zapri obe datoteki.
```

Z zadnjo kodo še preverimo, koliko je grafov tipa (SB) za posamezen n (število vozlišč), manjši ali enak 10. Rezultati so prikazani spodaj.

Algorithm 3 Preštevanje grafov tipa (SB) z različnim številom vozlišč

```
counter ← [0, 0, 0, ..., 0] ▷ (seznam s 11 ničlami, za štetje grafov z različnim številom vozlišč)
Odpri datoteko 'sb_graphs_g6.txt'
for vsako vrstico line v datoteki do
    g ← pretvori_v_graf(line)                                   ▷ (ustvarimo graf iz vrstice)
    counter[len(g.vertices())] ← counter[len(g.vertices())] + 1
end for
Izpis rezultatov:
for vsak i od 0 do 10 do
    izpiši "Število grafov tipa (SB) z i vozlišči: counter[i] "
end for
```

n (število vozlišč)	Število grafov tipa (SB)
0	0
1	0
2	0
3	0
4	0
5	2
6	11
7	116
8	1688
9	43420
10	2079097

Tabela 1: Tabela števila grafov tipa (SB) glede na število vozlišč

4 Naloga 2

Namen te naloge je naključno konstruirati grafe tipa (SB) za večje n. Ideja je, da to storimo preko konstrukcije prej opisanih množic iz particije vozlišč. Funkcija *generate_random_sb* vzame dva argumenta; prvi je željeno število vozlišč, drugi pa utež, ki določi kolikšna je verjetnost povezav. Nato za začetek v vsako izmed petih množic iz particije razporedi eno vozlišče, ostale pa dodaja v množice A_1, A_2 in A^* tako, da so ustrezno povezani z ostalimi skupinami za premer 2.

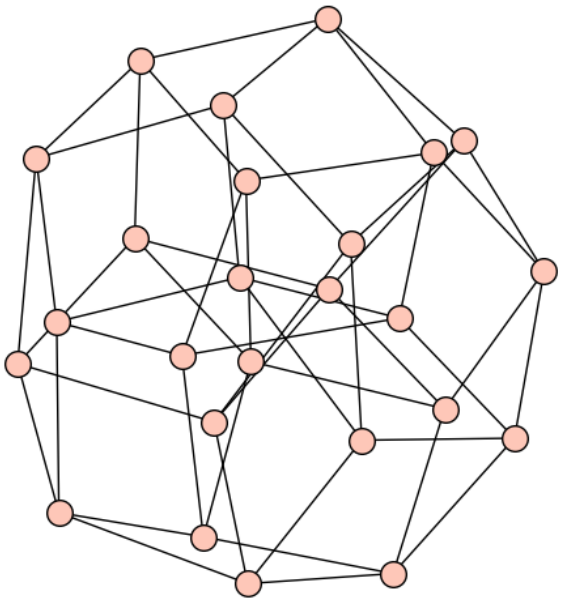
Koda je dostopna v datoteki *sb_random_alternate.py*.

5 Naloga 3

Želimo pridobiti tudi nov graf tipa (SB) iz danega grafa tipa (SB) tako, da naredimo nekaj manjših naključnih modifikacij (npr. dodajanje/odvzemanje vozlišč/povezav).
Koda je dostopna v datoteki *random_modify_SB_graph.py* na GitHubu, zaradi dolžine pa je zapisana le glavna ideja programa.
Funkcija *random_modify_sb_graph* kot argument vzame graf tipa (SB) in ga najprej shrani (kot varovalo za kasneje, če naključne spremembe ne dajo grafa tipa (SB)). Zapiše particijo vozlišč in se odloči za naključno število naključnih sprememb iz nabora. Te spremembe so same po sebi lahke za sprogramirati, paziti je treba zgolj, da ne prihaja do kakšnih neželenih struktur (samozanke ...).

Število sprememb je na začetku kolikor toliko majhno, da ne tvegamo, da bi graf preveč spremenili iz tipa (SB). Program izvede te spremembe in preveri ali nov graf ustreza tipu (SB). V kolikor ne, se postopek na takšnem grafu nadaljuje do tisočkrat, kasneje pa spet začnemo z originalnim grafom, ker bi bil novi graf lahko že precej degeneriran.
Funkcijo sva stestirala na približno 5000 primerih, pri čemer je za n blizu 30 postopek v okoli 95% primerov vrnil graf tipa (SB) preden je bilo izvedenih 1000 poskusov, torej preden smo spet vzeli prvotni graf. Skoraj vedno, ko je bilo generiranje grafa uspešno v teh prvih 1000 poskusih, se je to zgodilo v prvih 5 poskusih, kar potrjuje smiselnost zaustavitvenega pogoja z *while* zanko. Sklepava, da so namreč bili grafi potem že zelo 'oddaljeni' od tipa (SB). Za velike n je bil pri teh neuspešnih poskusih verjetno največkrat problem, da se je povečal premer grafa.

6 Naloga 4



Slika 3: Kartezični produkt grafa s slike 2 s samim seboj

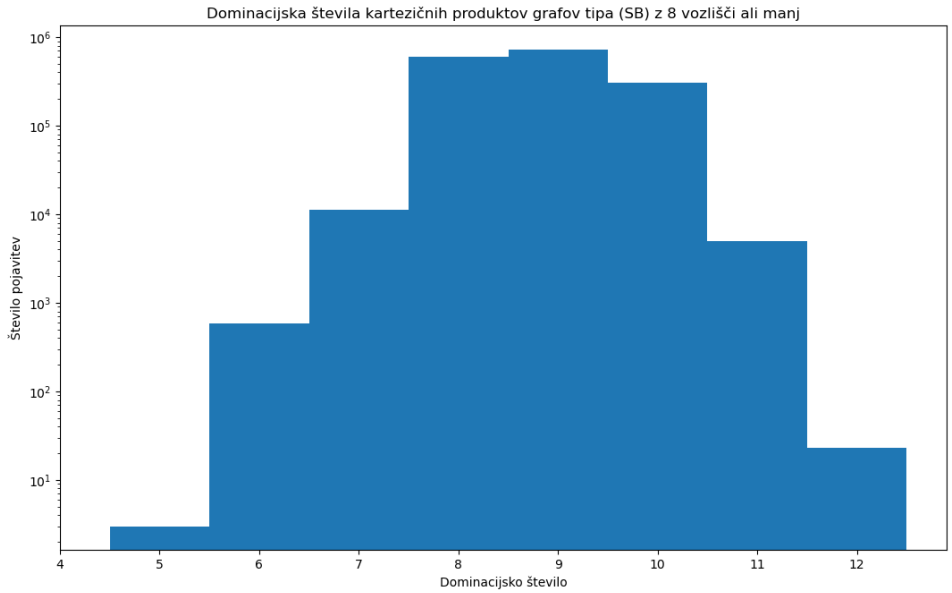
V datoteki *sb_filter_n.py* sva najprej prefiltrirala vse grafe tipa (SB) na do 10 vozliščih v dve skupini: tiste z 8 vozlišči ali manj in na tiste z 9 ali 10 vozlišči. Razlog za to je, da brute-force metoda za računanje dominacijskih števil vseh kartezičnih produktov grafov na do 8 vozliščih vzame približno 11 ur, z grafi na 9 in 10 vozliščih pa bi to že postalo časovno neizvedljivo za najine možnosti. Za izračun dominacijskih števil vseh kartezičnih produktov grafov na do 8 vozliščih sva uporabila kodo *sb_domination_brute_force_lower*. Vsa ta dominacijska števila sva shranila v datoteko *sb_cartesian_products_dominating_numbers_below_9.txt* v mapi *Podatki*. Datoteka za vsako dominacijsko število shranjuje tudi izvorna (SB) grafa. Predvidevava, da bi bila z vključitvijo grafov na več vozliščih dominacijska števila še večja, zato sklepava, da je najnižja vrednost 5, ki jo zavzamejo 3 kartezični produkti dveh grafov tipa (SB). Spodnja tabela prikazuje razporeditev števila kartezičnih produktov glede na dominacijsko število.

To prikazuje tudi naslednji histogram. Skala je logaritemska.

Kot sva že omenila, nisva mogla poiskati vseh dominacijskih števil, zato sva za ugotavljanje zgornje meje vzela 10000 naključnih kartezičnih produktov grafov tipa (SB) na 10 vozliščih brez ponavljanja. Koda za to se nahaja v datoteki *sb_domination_sampling_upper.py*, rezultati pa v

Dominacijsko število	Število grafov
5	3
6	586
7	11339
8	605061
9	722506
10	307143
11	4992
12	23

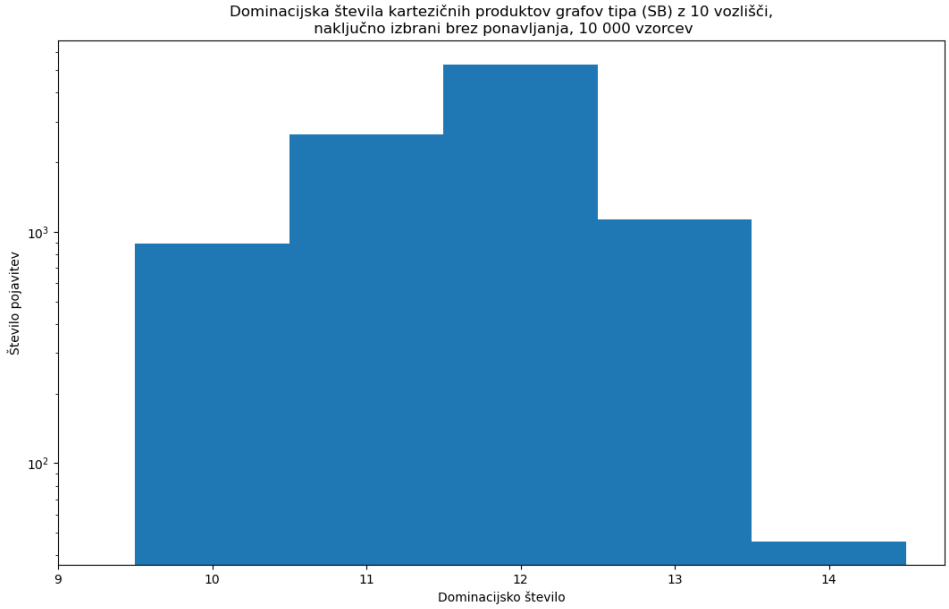
Tabela 2: Razporeditev omenjenih kartezičnih produktov glede na dominacijsko število.



sb_cartesian_products_dominating_numbers_upper_sampled.txt. Sklepava, da je najvišje dominacijsko število kartezičnega produkta dveh grafov tipa (SB) na do 10 vozliščih enako 14.

Dominacijsko število	Število grafov
10	889
11	2646
12	5289
13	1129
14	46

Tabela 3: Razporeditev omenjenih večjih kartezičnih produktov glede na dominacijsko število.


Dominacijska števila kartezičnih produktov grafov tipa (SB) z 10 vozlišči,
naključno izbrani brez ponavljanja, 10 000 vzorcev
Število pojavitev
10^3
10^2
9
10
11
12
13
14
Dominacijsko število