# 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. Naloga ima odprt tudi svoj repozitorij na GitHubu, na naslovu https://github.com/matickremzar/Graphs-of-type-SB-and-domination-on-their-cartesian-products-/tree/main. Tam so zbrane tudi vse datoteke, ki sva jih uporabila med izdelavo projekta.

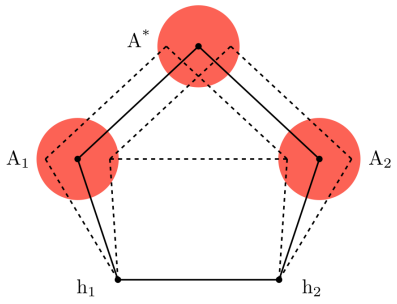
Cilj je razumeti, kako prepoznati, konstruirati in spreminjati grafe tipa (SB), ter analizirati njihove lastnosti, zlasti dominacijsko število njihovih kartezičnih produktov.

## 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

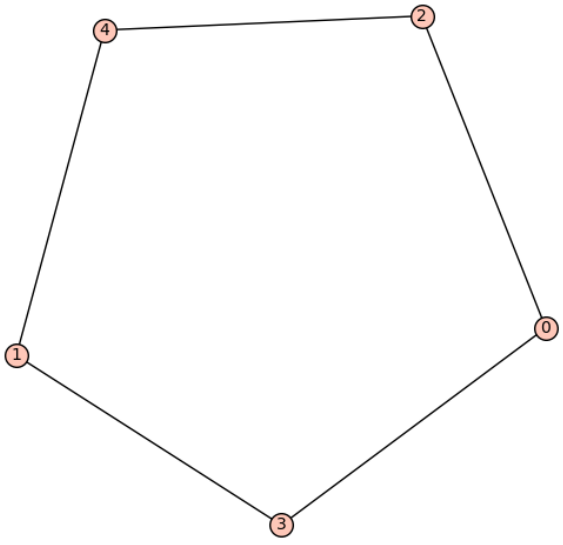
## 3.1 Preverjanje, ali je graf tipa (SB)

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 *graph*6 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)

```
1: function IsSB(G: Graph)                                  ▷ Funkcija za preverjanje grafa tipa (SB)
2:     if G.diameter() ≠ 2 then return False                ▷ Premer mora biti 2
3:     adj ← G.adjacency_matrix()                           ▷ Pridobimo matriko sosednosti
4:     for i ← 1 to adj.nrows() do
5:         for j ← i + 1 to adj.nrows() do                  ▷ Samo zgornji trikotnik
6:             if adj[i, j] = 1 then                        ▷ Če sta i in j soseda
7:                 common_neighbour ← False
8:                 for k ← 1 to adj.nrows() do
9:                     if adj[i, k] = 1 and adj[j, k] = 1 then   ▷ Preverimo skupne sosede
10:                        common_neighbour ← True
11:                        break
12:                    end if
13:                end for
14:                if common_neighbour then
15:                    continue
16:                end if
17:                nonadj_vertex_exists ← False
18:                for k ← 1 to adj.nrows() do
19:                    if adj[i, k] = 0 and adj[j, k] = 0 and k ≠ i and k ≠ j then
20:                        nonadj_vertex_exists ← True
21:                        break
22:                    end if
23:                end for
24:                if nonadj_vertex_exists then
25:                    return True
26:                end if
27:            end if
28:        end for
29:    end for
30:    return False                                         ▷ Graf ni tipa (SB)
31: end function
```

**Algorithm 2** Filtriranje grafov tipa (SB)

```
1: Odpri datoteko 'sb_graphs.txt' v načinu dodajanja ("a").
2: Odpri datoteko "diam_two_graphs.txt" v načinu branja ("r").
3: for vsako vrstico line v datoteki "diam_two_graphs.txt" do
4:     g ← pretvori_v_graf(line)          ▷ Ustvari graf iz podatkov na trenutni vrstici.
5:     if is_sb(g) then                   ▷ Preveri, če je graf g tipa (SB).
6:         Zapiši line v datoteko 'sb_graphs.txt'.
7:     end if
8: end for
9: 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 $n$. Vzamemo nek graf na $n$ vozliščih in ga s pomočjo smiselnega odvzemanja in dodajanja povezav (poskušamo doseči na particijo vozlišč, da pridobimo pogoje za (SB)) poskušamo spremeniti v graf tipa (SB). Pri tem moramo paziti na premer.

Tukaj je več možnih idej, kako implementirati metodo, ki bi zgenerirala naključen graf tipa (SB). Midva sva se tega lotila na sledeč način: s SageMath zgeneriramo naključen graf, s pomočjo funkcije *randomGNP(n, p)*, ki sprejme not število željenih vozlišč in verjetnost, ali se naj doda povezava med dvema vozliščema. Nato izberemo naključno povezavo, in vozlišči v tej povezavi označimo z $v_1$ in $v_2$. Nato konstruiramo množice $A_1, A_2, A^*$, kot je navedeno v definiciji grafa tipa (SB). Če je kakšna od teh množic morda prazna, v to prazno množico dodamo novo vozlišče in ustrezne povezave zanj. Nato poskrbimo, da so vsa vozlišča v $A_1$ povezana z vsemi vozlišči v $A^*$, torej tista, ki niso, jih povežemo med seboj. Podobno storimo za vozlišča v $A_2$. S tem je postopek zaključen in smo pridobili graf tipa (SB). Časovna zahtevnost tega algoritma je $O(|V|^2)$.

# 5 Naloga 3

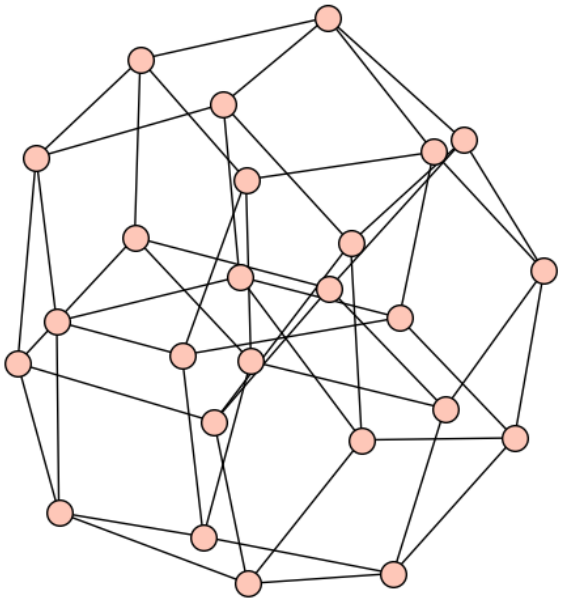
Želimo pridobiti tudi nov graf tipa (SB) iz danega grafa tipa (SB) tako, da naredimo nekaj manjših naključnih modifikacij (npr. dodajanje/odvzemanje vozlišč/povezav).

**Algorithm 4** Generiraj naključen SB graf - naloga 2

**Vhod:** Celo število $n$ (število vozlišč)
**Izhod:** Graf $g$ tipa SB z premerom 2
$g \leftarrow$ Naključen graf z $n$ vozlišči in verjetnostjo povezave 0.5
$(h1, h2) \leftarrow$ prvi rob grafa $g$
$a1 \leftarrow$ sosedi vozlišča $h1$
$a2 \leftarrow$ sosedi vozlišča $h2$
Odstrani $h2$ iz seznama $a1$
Odstrani $h1$ iz seznama $a2$
$a\_star \leftarrow$ prazen seznam
**for** vsako vozlišče $v$ v grafu $g$ **do**
    **if** $v \notin a1$ in $v \notin a2$ in $v \neq h1$ in $v \neq h2$ **then**
        Dodaj $v$ v seznam $a\_star$
    **end if**
**end for**
$vertices\_to\_remove \leftarrow$ prazen seznam
**for** vsako vozlišče $n1$ v $a1$ **do**
    **for** vsako vozlišče $n2$ v $a2$ **do**
        **if** $n1 == n2$ **then**
            Odstrani rob $(h1, n1)$
            Dodaj $n1$ v seznam $vertices\_to\_remove$
        **end if**
    **end for**
**end for**
**for** vsako vozlišče $v$ v seznamu $vertices\_to\_remove$ **do**
    Odstrani $v$ iz seznama $a1$
**end for**
**if** dolžina seznama $a1$ je 0 **then**
    $u \leftarrow$ dodaj novo vozlišče
    Dodaj $u$ v seznam $a1$
    Dodaj rob $(h1, u)$
**end if**
**if** dolžina seznama $a2$ je 0 **then**
    $u \leftarrow$ dodaj novo vozlišče
    Dodaj $u$ v seznam $a2$
    Dodaj rob $(h2, u)$
**end if**
**if** dolžina seznama $a\_star$ je 0 **then**
    $u \leftarrow$ dodaj novo vozlišče
    Dodaj $u$ v seznam $a\_star$
    $v_{a1} \leftarrow$ prvi element seznama $a1$
    $v_{a2} \leftarrow$ prvi element seznama $a2$
    Dodaj rob $(u, v_{a1})$
    Dodaj rob $(u, v_{a2})$
**end if**
**for** vsako vozlišče $u$ v seznamu $a1$ **do**
    **for** vsako vozlišče $v$ v seznamu $a\_star$ **do**
        Dodaj rob $(u, v)$
    **end for**
**end for**
**for** vsako vozlišče $u$ v seznamu $a2$ **do**
    **for** vsako vozlišče $v$ v seznamu $a\_star$ **do**
        Dodaj rob $(u, v)$
    **end for**
**end for**
$diam \leftarrow$ premer grafa $g$
**if** $diam == 2$ in $g$ je tipa SB **then**
    **Vrni** graf $g$
**end if**

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

Znano je, da je iskanje dominacijskega števila za grafe NP-težek problem. Želela bi najti, kakšna so dominacijska števila za kartezične produkte grafov tipa (SB) do 10 vozlišč. Izkaže se, da ne moremo dobiti točnega rezultata, saj je problem prezahteven. Zato bi se namesto tega vprašali, kakšen je možen nabor vrednosti, ki jih lahko zavzame dominacijsko število kartezičnega produkta dveh grafov tipa (SB)?

Poglejmo za začetek dominacijsko število na običajnih grafih tipa (SB). Intuitivno je jasno, da imajo vsi grafi tipa (SB) dominacijsko število najmanj 2. Tudi dokaz je jasen: Zapišimo particijo vozlišč nekega grafa tipa (SB), kot je predlagano v osnovni ideji problema. Če je v najmanjši dominacijski množici vozlišče $v_1$, imamo zraven pokrito še $v_2$ in $A_1$. Vemo pa, da je $A^*$ neprazna, saj obstaja vozlišče $v^*$, ki ni sosednje $v_1$ ali $v_2$. Torej $v_1$ ni zadosten in potrebujemo vsaj še eno

vozlišče v dominacijski množici. Analogno sledi za $v_2$. Če je v dominacijski množici neko vozlišče iz $A^*$ ali $A_2$ ($A_1$), rabimo še kakšno vozlišče, ki je sosedno $v_1$ ($v_2$).
Nato lahko vpeljemo Vizingovo domnevo: Za grafa G in H velja

$$\gamma(G) * \gamma(H) \leq \gamma(G \Box H).$$

Ta sicer ni dokazana v splošnem, je pa preverjena za zadosten nabor grafov, da jo je smiselno uporabiti v tej nalogi. Iz tega lahko trdimo, da je najnižje možno dominacijsko število na kartezičnem produktu dveh grafov tipa (SB) enako 4. Da je spodnja meja potencialno lahko dosežena, potrjujejo tudi rezultati, ki jih da program *sb_get_all_domination_numbers.py*. Ti so shranjeni v datoteki *sb_graphs_domination_numbers.txt*. V praksi se je izkazalo, da ima velika večina grafov tipa (SB) na do 10 vozliščih dominacijsko število enako 2, preostali pa 3.
Zanimiva variacija problema bi bila, v kolikor bi bil kartezični produkt nekih dveh grafov tipa (SB) spet graf tipa (SB). Posledično bi takoj sledilo, da obstajajo grafi tipa (SB) z dominacijskim številom vsaj 4. Ko sva pregledala grafe na do 10 vozliščih, takšnega primera nisva našla. Problem je bil seveda premer kartezičnih produktov grafov. Vseh 10000 grafov, ki sva jih tako generirala je imelo premer 4. Ta problem je bil reševan z datoteko *sb_cartesian.py*.
Iskanje zgornjih mej za dominacijsko število kartezičnega produkta dveh grafov tipa (SB) je zahtevnejša. Najboljša zgornja meja, ki sva jo uspela najti je še ena Vizingova izpeljava:

$$\gamma(G \Box H) \leq min\{\gamma(G)|H|, \gamma(H)|G|\}$$

Če uporabimo sedaj še to neenačbo, dobimo, da je zgornja meja dominacijskega števila 30, ko se omejimo na kartezične produkte grafov (SB) do 10 vozlišč. A ni jasno, ali je ta zgornja meja sploh zelo natančna, saj se je izkazalo, da je točen izračun dominacijskega števila grafa s 100 vozlišči zelo zahteven proces, zato tukaj nimava nabora izračunanih dominacijskih števil za te velike grafe, s katerimi bi lahko dobila občutek ali je ta zgornja meja 'dobra' ali ne.

Lahko pa si pogledamo kartezične produkte najmanjših grafov tipa (SB). Za prvih 20 najmanjših (SB) grafov sva poiskala dominacijska števila z Brute-Force metodo (*sb_domination_brute_force.py*). Koda kartezično pomnoži med seboj prvih 20 (SB) grafov in pregleda vse dominacijske množice. Spodnja meja iz Vizingove domneve (4) ni bila dosežena. Le prvi trije kartezični produkti so imeli dominacijsko število 5, ostali za njimi pa 6, 7, 8 ali 9.

| Dominacijsko število | Število grafov |
|---|---|
| 5 | 3 |
| 6 | 75 |
| 7 | 74 |
| 8 | 43 |
| 9 | 15 |

Tabela 2: Razporeditev kartezičnih produktov dveh med najmanjšimi 20 (SB) grafi glede na dominacijsko število.

Tudi program *sb_domination_sampling_method.py* se ukvarja s spodnjimi mejami za dominacijska števila za naključnih 10000 (SB) grafov (tudi višjih redov). Pomaga si z Vizingovo domnevo. Uporablja *cached_results*, da za posamezen graf izračuna dominacijsko število le enkrat. Namesto da gleda 20 najmanjših grafov, je bila ideja da se izbere naključen nabor vseh pridobljenih grafov.

Glede zgornje meje za dominacijsko število sva ugotovila, da gre po vsej verjetnosti z večanjem grafov v neskončnost, če bi gledali tudi kartezične produkte grafov tipa (SB) z več kot 10 vozlišči.