Finančni praktikum
Projektna naloga 23

Dominacije na grafih diametra 2

Avtorja: Jošt Plevel in Martin Iglič
Mentorja: Janoš Vidali in Riste Škrekovski

Ljubljana, december 2023

Kazalo

1 Opis problema in členitev reševanja

Za jasnejše uporabljanje pojmov iz najiine naloge bova najprej definirala pojma *diameter(premer)* in *dominacijsko število* grafa.

1.1 Definiciji

Definicija 1 (Diameter grafa)**.** *Naj bo graf $G = (V, E)$ podan z množico vozlišč V in množico povezav E. Diameter(premer) grafa $\delta(G)$ je maksimalna razdalja med vsemi najkrajšimi razdaljami med pari vozlišč $u, v \in V$.*
V najinem primeru obravnavava grafe $\delta(G) = 2$, torej grafe, kjer sta vsaki dve vozlišči $u, v \in G$ med seboj oddaljeni za največ dve povezavi.

Definicija 2 (Dominacijsko število)**.** *Naj bo graf $G = (V, E)$ podan z množico vozlišč V in množico povezav E. Podmnožica vozlišč $D \subset V$ grafa G imenujemo dominacijska množica, če je vsako vozlišče $v \in D$ sosed vozlišča $u \in V \backslash D$, torej vozlišča, ki ni v množici D.*
Dominacisko število grafa $\gamma(G)$ je moč najmanjše dominacijske množice D.

1.2 Opis problema

Najina naloga je poiskati graf na n vozliščih, ki je diametra dva in ima izmed vseh grafov s temi lastnostmi največje dominacijsko število med grafi velikosti n. Torej poiskati maksimum med dominacijskimi števili, ki pa so minimumi drugega problema.
Nalogo sva si razdelila tako, da na malih grafih narediva sistematično iskanje grafov in najprej poiščeva množico tistih, ki so diametra 2. Pri sistematičnem iskanju sva uporabila vgrajene funkcije, medtem ko sva na večjih grafih generirala naključne grafe in jim dodajala povezave, dokler niso imeli diametra 2. Nato sva spet s pomočjo vgrajenih funkcij poiskala dominacijsko število.

2 Majhni grafi

Pri iskanju največjega dominacijskega števila na majhnih grafih je prednost, da lahko pregledamo vse grafe na n vozliščih. Vendar pa je tak pristop generiranja spominsko zelo zahteven, zato sva uspela doseči le delovanje grafov, ki so imeli osem ogljišč ali manj. Kljub temu, pa so tukaj zajete vse možne rešitve.

2.1 Metoda iskanja

Iz množice vseh povezanih grafov velikosti n izločimo le tiste z diametrom 2. Na tem koraku imamo množico grafov M za, katere je vsak možen graf velikosti n, ki ima to lastnost. Torej velja $M = \{G; |V(G)| = n \wedge \delta(G) \leq 2\}$ Nato sva za vsak graf poiskala njegovo dominacijsko število. s preprosto zanko sva preverila, če ima kateri graf večje dominacijsko število kot njegov predhonik. V tem primeru, se je največje dominacijsko število dominacijsko število povečalo, sicer pa ostalo enako, kot pri predhodniku.

2.2 Delovanje programa

Ponavadi na začetku defeniramo n, ki nam pove, na kako velikih grafih naj se program izvaja.

Progaram je strukturiran iz več pomožnih funkcij.

2.2.1 Generiranje grafov velikosti n

Naslednja funkcija sprejme stopnjo grafa n in vrne seznam vseh povezanih grafov, ki so diametra 2 v zapisu *GRAPH*.

```
def seznam_grafov_velikosti_n_diametra_2(n):
    grafi_diameter_2_velikosti_i = list()
    vsi_grafi_velikost_i = list(graphs.nauty_geng(f"{n} -c"))
    for graf in vsi_grafi_velikost_i:
        if graf.diameter() <= 2:
            grafi_diameter_2_velikosti_i.append(graf)
    return(grafi_diameter_2_velikosti_i)
```

2.2.2 Iskanje največjega dominacijskega števila $\gamma(G)$

Zdaj, ko imamo množico ustreznih grafov, med vsemi poiščemo, največje dominacijsko število. To sva storila tako, da sva na začetku dominacijsko število nastavila na vrednost 0, nato pa sva za vsak graf vprašala, ali je dominacijsko števila trenutnega grafa večje od največjega od vseh prejšnjih. Če je bilo večje, se samo dominacijsko število poveča z novim. Na koncu program vrne iskano dominacijsko število.

```
def maksimalno_dom_st(seznam_grafov):
maksimalno_dominacijsko_stevilo = 0
for graf in seznam_grafov:
```

```
        if graf.dominating_set(value_only=True) > maksimalno_dominacijsko_stevilo:
            maksimalno_dominacijsko_stevilo = graf.dominating_set(value_only=True)
    return(maksimalno_dominacijsko_stevilo)
```

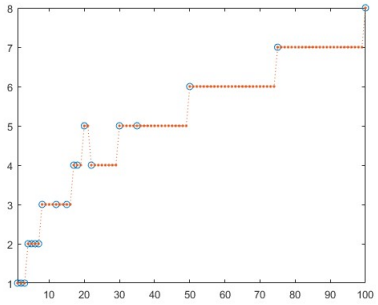
Zdaj imamo dobljeno iskano število, je pa definiranih še nekaj funkcij, ki pridejo morda prav pri nadaljni analizi.

2.2.3 Dodatne funkcije

Funkcija **povej_podatke_do_velikosti_n** nam v obliki pisnega poročila pove, za vsako velikost grafa, koliko grafaov diametra 2 te velikosti je, koliko je največje dominacijsko število pri tej velikosti, ter koliko grafo v tej množico ima največje dominacijsko število.
Pri tem sva uporabila funkcijo **seznam_grafov_z_maks_dom_st(seznam_grafov_2_diameter)**, ki izračuna dominacijsko število in ustvari seznam vseh grafov, ki imajo največje dominacijsko število. Poudarila bi, da vsak element seznam je tipa *GRAPH*, kar pomeni, da je možno do njegove strukture možno dostopati.
Zadnja pomožna funkcija nam prav to nudi. Funkcija **podaj_t_primere_grafov_z_maks_dom_st(seznam_gr**
t = 3) nam iz seznam vrne naključnih t primerov grafov kot sliko v 2D.

2.3 Rezultati

Pri majhnih grafih je bilo možno pognati za grafe do velikosti 8. Pri večjih je server *CoCalc* vrnil napako, da je zmanjkalo prostora. Ker pri sistematičnem iskanju potrebujemo vse ustrezne grafe, naju je to prisillo v nekoliko ukleščeno analizo. Dominacijsko število pri grafih do velikosti do 7 je bilo dva, pri grafih na 8 vozliščih pa je bilo največje dominacijsko število 3. Kar pomeni, da pogoj, da je graf dimaetra dva zagoto zelo goste grafe posledično pa to prinese, da z malo vozlišči dominiramo celoten graf. Škoda je, da sva bila omejena s strojno opremo, Tako upava, da bi morda zmoglivejši računalnik uspel sistematično poiskati rešitve, do velikosti $n = 25$.
Največje dominacijsko število raste z velikostjo grafov, vendar pa je rast zelo počasna. V prihodnosti bi lahko problem modelirarali, tako da bi gledali za različne vrste dominacije, kar bi pomenilo, da bi lahko eno vozlišče še lažje dominiralo, več ostalih (sosedi sosedov).



Slika 1: Interpolacija podatkov po metodi *previous*

3 Iskanje velikega dominacijskega števila na velikih grafih

3.1 Metoda iskanja

Pri velikih grafih ne moremo uvesti sistematičnega iskanja dominacijskega števila, s katerim bi pregledali vse možne grafe diametra 2 za dan n, ker je grafov preveč in ker bi bil proces zamuden, tudi če bi lahko našli vse ustrezne grafe. Izkazalo se je, da je v izbranem orodju že za $n \geq 9$ preveč ustreznih, da bi lahko preverila vse. Zato sva uvedla naključno iskanje grafov z idejo, da ob velikem številu pregledanih grafov vendarle lahko dobimo reprezentativen rezultat. Torej je bilo treba narediti program, ki generira naključne grafe (seveda z ustreznimi lastnosti), na katerih potem preveri dominacijsko število.

3.2 Delovanje programa

Sledeče je okviren opis delovanja programa, izpostavljene so nekatere zanimivosti; za bolj podroben pregled programa prilagava povezavo do GitHub repozitorija: `github.com/MartinIglic/Domination-of-diameter-two-graphs`, kjer se nahaja celotna koda s komentarji, ki precej podrobno razložijo tudi delovanje pomožnih funkcij. Spisala sva program, ki najprej generira naključen nepoln povezan graf na n vozliščih z verjetnostjo p, da se med vsakima dvema vozliščema ustvari povezava. Zatem program preveri, če je ustvarjeni graf že diametra 2; če ni, stori sledeče:

- pregleda vse najkrajše poti med vsakim parom vozlišč,
- izbere najdaljšo,
- doda povezavo med še nepovezanima vozliščema na tej poti.

Nato spet preveri diameter grafa in ponovi zgornji postopek, če diameter ni dve. Ko graf doseže diameter dve, program s pomočjo vgrajenih funckij izračuna njegovo dominacijsko število. Če je le-to strogo večje od prejšnjega največjega, si zapomni tako število kot tudi sam graf. V vsakem primeru nato program odstrani *nekaj* povezav v grafu. Poglejmo si kodo za odstranitev povezav:

```
def odstrani_povezave():
global G
n = G.order()
k = random.randrange(5, 20)
for i in range(k):
    a = random.randrange(n)
    b = random.randrange(n)
    if G.has_edge(a, b) == True:
        if G.is_cut_edge(a, b) == False:
            G.delete_edge((a, b))
return None
```

Program naredi od 5 do 20 iteracij, kjer vsakič izbere dve vozlišči, preveri, če je med njima povezava, nato preveri, ali graf ostane povezan tudi brez te povezave. Izpolnjena morata biti oba pogoja, da je povezava res odstranjena. Lahko bi se zgodilo, da bi izbral same take pare vozlišč, kjer ne bi mogel odstraniti povezave oz. je sploh ne bi bilo. Program torej odstrani od 0 do 19 povezav. Za bolj napreden program bi bilo

smiselno z dodatnim testiranjem in premislekom interval prilagoditi, da bi bil ustrezen za različno velike grafe. V našem primeru, torej za $20 \leq n \leq 100$, se je taka izbira parametrov zdela ustrezna, ker graf diametra 2 vendarle ima precej povezav, hkrati pa se graf vendarle dovolj spreminja.

Podrobneje poglejmo še delovanje glavne funkcije, ki prejšnje pomožne poveže v eno.

```
def veliko_dom_st():
global G
maks = 0
i = 0
while True:
i += 1
    spremeni_v_premer_2()
    st = G.dominating_set(value_only = True)
    if st > maks:
        maks = st
        S = G
        S1 = G.dominating_set()
        print(maks, S.to_dictionary(), S1)
    if i % 1000 == 0:
        G = generiraj_povezan_graf(n, p)
    odstrani_povezave()
return None
```

Vnaprej generiran začetni graf funkcija spremeni v graf premera 2, nato vgrajena funckija poišče njegovo dominacijsko število; če je to strogo večje od doslej največjega, si funkcija zapomni tako število kot graf, poleg tega pa tudi že poišče najmanjše pokritje (s tem dobimo zagotovilo, da je program deloval pravilno in res našel, kar smo iskali).

Večinoma nato odstrani nekaj povezav in gre z novim grafom v novo ponovitev. Vsake toliko, v podanem primeru na vsakih 1000 ponovitev, pa program namesto odstranjevanja povezav ustvari nov graf neodvisno od prejšnjega. To je kot nekakšna varovalka, ki se je pri tstiranju izkazala za potrebno, sploh pri grafih recimo $n \leq 20$. V nekaj primerih je bil ustvarjeni graf zvezda, torej graf, ki mu ne moremo odstraniti nobene povezave, ne da bi razpadel na nepovezane komponente, hkrati pa je že diametra dve, torej mu program ne bo dodal nobene povezave. Lahko bi dodali v program tudi kontrolo, ki tak graf takoj identificira, ampak za ustrezen p se to ni pogosto dogajalo, program pa hitro predela tistih kvečjemu 1000 iteracij, če je graf zvezda.

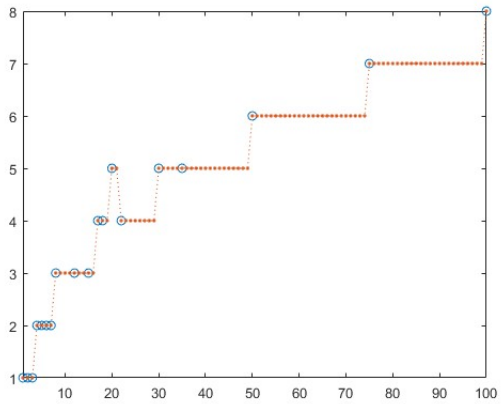
3.3 Rezultati

Za pridobitev rezultatov, ki so reprezentativni, je treba kodo poganjati dovolj časa, da program pregleda veliko grafov; upamo seveda, da tudi tistega, ki za dan n doseže maksimalno dominacijsko število. Program je tekel približno en dan za različne velikosti grafa. Npr. pri $n = 12$ je bilo pregledanih petkrat po 2,4 milijona grafov (verjetno sicer nekateri izmed njih kar precejkrat), pri $n = 100$ pa petkrat po slab oz. dober milijon. V tabeli so vidni rezultati.

Dominacijsko število z izjemo enega primera raste, ko se velikost grafa veča. Pri največjih treh testnih številih vozlišč je sicer vprašljivo, če se je program izvajal dovolj

n	$\gamma(n)$
12	3
15	4
16	4
17	4
18	4
20	5
22	4
30	5
38	5
50	6
75	7
100	8

dolgo, da je res našel maksimum. Pri tako veliki množici vozlišč je tudi fiksna verjetnost $p = 0.2$ morda preveč deterministična in bi morali poskušati še z drugimi vrednostmi parametra, da bi res prišli do grafa, ki ga iščemo.



Slika 2: Interpolacija podatkov po metodi *previous*