Finančni praktikum

k-total rainbow domination numer vs. domination number

Tim Resnik
Lana Herman
Univerza v Ljubljani

Fakulteta za matematiko in fiziko

November, 2019

Kazalo

1 Problem naloge

V projektni nalogi se bova ukvarjala z domeno, ki se ukvarja s povezavo med 'k-rainbow total domination number' (označimo z $\gamma_{krt}(G)$) in 'domination number' (označimo z $\gamma(G)$). Definiciji za $\gamma_{krt}(G)$ in $\gamma(G)$ sta v razdelku **Razlaga pojmov**.
Domneva pravi, da za graf G in $k \geq 4$ obstaja tesna povezava $\gamma_{krt}(G) \geq 2\gamma(G)$. Cilj najine projetkne naloge je najti tak graf, za katerega ta neenkost ne drži. To sva na majhnih grafih preverila na konkretnih primerih, kjer sva število vozlišč in število k vnesla ročno. Za večje grafe sva uporabila metodo *Simulated Annealing*.
Poiskala sva tudi primere, za katere velja enakost $\gamma_{krt}(G) = 2\gamma(G)$.

2 Razlaga pojmov

Graf G ima množico vozlišč $V(G)$ in množico povezav $E(G)$. Za množico $N_G(v)$ velja, da vsebuje vsa sosednja vozlišča v, v grafu G. Za grafa G in H, je kartezični produkt $G \Box H$ graf z množico vozlišč $V(G) \times V(H)$.
Dominirana množica grafa G je $D \subseteq V(G)$, taka da za vsako vozlišče $v \in V(G)$ in $v \notin D$ velja, da je sosed nekemu vozlišču iz D. *Dominirano število*, $\gamma(G)$, je velikost najmanjše dominirane množice. Če za $\forall v \in V(G)$ velja, da je sosed vozlišču iz D, za D rečemo, da je *totalno dominirana množica* grafa G. *Totalno dominirano število*, $\gamma_t(G)$, je velikost najmanjše totalno dominirane množice.
Za pozitivno celo število k, je *'k-rainbow domination function'* (kRDF) grafa G funkcija f, ki slika iz $V(G)$ v množico $\{1, \cdots, k\}$. Zanjo velja, da za katerikoli $v \in V(G)$ in $f(v) = \emptyset$ velja $\cup_{u \in N_G(v)} f(u) = [k]$. Definiramo $\|f\| = \sum_{v \in V(G)} |f(v)|$. $\|f\|$ rečemo *teža* f-a. *'k-rainbow domination number'*, $\gamma_{kr}(G)$, grafa G je minimalna vrednost $\|f\|$ za vse 'k-rainbow domination functions'. Po definiciji vemo, da za vse $k \geq 1$ velja

$$\gamma_{kr}(G) = \gamma(G \Box K_k).$$

Graf K_k predstavlja polni graf na k vozliščih. Nazadnje definirajmo še *'k-rainbow total domination function'* (kRTDF), katera se od 'k-rainbow domination function' razlikuje v dodatnem pogoju, ki zagotavlja, da če za $\forall v \in V(G)$ velja $f(v) = \{i\}$, potem obstaja tak $u \in N_G(v)$, da je $i \in f(u)$. *'k-rainbow total domination number'*, $\gamma_{krt}(G)$, grafa G je minimalna vrednost $\|f\|$ za vse 'k-rainbow total domination functions'. Tudi tu za vse $k \geq 1$ velja

$$\gamma_{krt}(G) = \gamma_t(G \Box K_k).$$

3 Reševanje problema

3.1 Majhni grafi

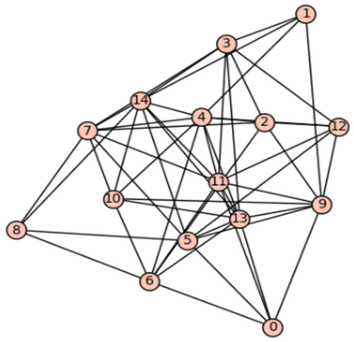
Za majhne grafe sva definirala naslednjo pseudokodo.

```
result=[]; #seznam vseh gamak/gama
gresult=[]; #seznam proti primerov
dvaresult=[]; #seznam grafov, ki imajo x=2
n=15; #število vozlišč
p=0.5; #verjetnost da posamezno povezavo dodaš v graf
for i in range(10): #preveri na 10ih random grafih
    for k in range(2,6):
        g=graphs.RandomGNP(n,p)
        gk=g.cartesian_product(graphs.CompleteGraph(k))
        gama=g.dominating_set(value_only=True)
        gamak=gk.dominating_set(value_only=True,total=True)
        x=float(gamak/gama);
        if x<2: gresult.append(g)
        if x==2: dvaresult.append(g)
        result.append(x)
print(min(result))
```

V kodi seznam *result* predstavlja seznam vseh koeficientov $\frac{\gamma_{krt}(G)}{\gamma(G)}$, seznam *gresult* predstvalja vse tiste grafe, za katere velja neenakost $\frac{\gamma_{krt}(G)}{\gamma(G)} < 2$, seznam *dvaresult* pa predstavlja grafe za katere velja enakost $\frac{\gamma_{krt}(G)}{\gamma(G)} = 2$.
Prva zanka preteče i naključnih grafov, generiranih z že vgrajeno funkcijo iz programa *sage*, imenovano *graphs.RandomGNP(n,p)*, kjer n predstvalja število vozlišč, p pa verjetnost, da posamezno povezavo doda v graf. Ta zanka je porabi $\mathcal{O}(n)$ časa.
Druga zanka se zapelje čez določeni število k-jev, kjer k predstvalja število vozlišč na polnem grafu. Po definiciji namreč vemo, da 'k-rainbow total domination number' izračunamo preko kartezičnega produkta med grafom G in polnim grafom s k vozlišči. Druga zanka porabi $\mathcal{O}(m)$ časa, kjer je m število grafov, ki jih preteče k.
Funkcija *cartesian_product()* porabi $\mathcal{O}(nm)$ časa, saj vsakemu vozlišču iz grafa G (n vozlišč) priredi vsako vozlišče iz k-polnega grafa (m vozlišč).
Torej je časovna zahtevnost te pseudokode enaka $\mathcal{O}(n^2m^2)$.

To kodo sva uporabila za $n \in \{5, \cdots, 20\}$ na različnem številu korakov in različnih k-jih. Za $k \geq 3$ sva minimum zmeraj dobila enak 2, za $k = 2$ pa sva dobila veliko protiprimerov, npr.:

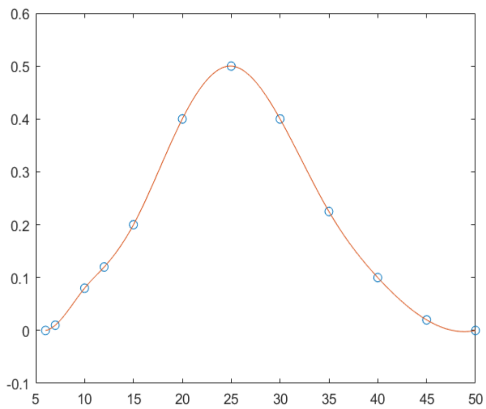
```
gresult[0].plot()
```



```
len(gresult)
```

```
205
```

Opazimo, da je takih grafov kar 205 (koda je preverila na 1000 grafih s 15 vozlišči). To število se seveda spreminja, ko kodo poženemo večkrat. Zato sva jo pognala večrat in na različnem številu vozlišč. Vse skupaj je koda pregledala približno 10^{100} grafov, izključno za $k = 2$. Podatke sva izpisala v program *Matlab*, jih za posamezno število vozlišč povprečila in jih aproksimirala z že vgrajeno funkcijo *interp1*. Dobila sva spodnji graf.



Opazimo, da protiprimerov ne dobimo za grafe z manj kot 7 vozlišč, za večje grafe pa velja, da gre število protiprimerov proti nič, ko se veča število vozlišč. Največ pa jih je pri grafih s približno 25 vozlišči.

3.2 Veliki grafi

Za velike grafe sva definirala naslednjo pseudokodo.

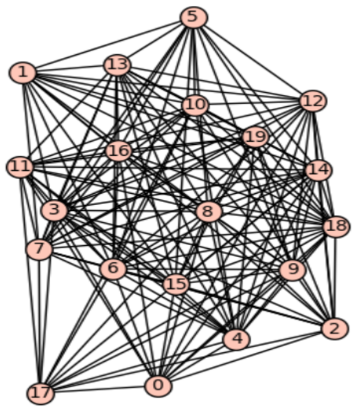
```
k = 4
def spremeni_graf(G):
    for i in range(5):
        H = Graph(G)
        if random() < 0.5:
            H.delete_edge(H.random_edge())
            if not H.is_connected():
                H = Graph(G)
        else:
            if H.complement().size() == 0:
                H.delete_edge(H.random_edge())
            else:
                H.add_edge(H.complement().random_edge())
    return H
```

```
def krit(G):
    gama=G.dominating_set(value_only=True)
    gamak=(G.cartesian_product(graphs.CompleteGraph(k))).dominating_set(value_only=True, total=True)
    return gamak/gama
```

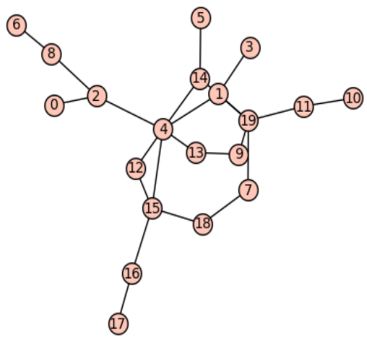
```
p = 0
stevilo_korakov = 2000
najboljsi = G
kk = k_min = krit(G)
for p in range(0, stevilo_korakov):
    Temp = stevilo_korakov / (p+1)
    novi = spremeni_graf(G)
    kn = krit(novi)
    razlika = kk - kn
    if razlika > 0:
        G = novi
        kk = kn
        if kn < k_min:
            najboljsi = novi
            k_min = kn
    elif exp((razlika * 100)/Temp) > random():
        G = novi
        kk = kn
    p += 1
print(najboljsi.show(), k_min)
```

Funkcija *spremeni_graf()* nam ne nepovezan graf tako spremeni, da mu naključno odsrani povezavo in naključno doda povezavo. To funkcijo sva definirala zato, da sva jo lahko uporabila v naslenji kodi. Njena časovna zahtevnost je $\mathcal{O}(n^2)$, saj gre čez dve for zanki, vsaka ima zahtevnost $\mathcal{O}(n)$.
Zadnja koda pa predstavlja metodo *Simulated Annealing*. To je metoda za približevanje h globalnemu optimumu dane funkcije. Tako sva s spreminjanjem danega grafa iskala tak graf, za katerega bo veljalo $\frac{\gamma_{krt}(G)}{\gamma(G)} < 2$. To kodo sva pognala na grafih z vsaj 20 vozlišči. Na potlagi najinih poiskusov sva ugotovila, da za grafe s sodim številom vozlišč je koeficient enak 2, za grafe z lihim številom vozlišč pa velja formula $\frac{n}{\lceil \frac{n}{2} \rceil}$. Opazimo torej, če pošljemo $n \to \infty$, gre koeficient proti 2.

Prvi graf ima 20 vozlišč, $k = 4$ in $p = 0.5$, drugi pa ima $k = 5$ in $p = 0.1$.



(None, 2)



(None, 2)