

Finančni praktikum

$k$ -total rainbow domination number vs.  
domination number

Tim Resnik  
Lana Herman  
Univerza v Ljubljani

Fakulteta za matematiko in fiziko

November, 2019

# Kazalo

<b>1</b>	<b>Problem naloge</b>	<b>2</b>
<b>2</b>	<b>Razlaga pojmov</b>	<b>2</b>
<b>3</b>	<b>Reševanje problema</b>	<b>3</b>
3.1	Majhni grafi . . . . .	3
3.2	Veliki grafi . . . . .	7
3.3	Drevesa . . . . .	10
3.4	Dvodelni grafi . . . . .	10

# 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 projektne naloge je najti tak graf, za katerega ta neenakost 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  $G \square H$  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*' ( $k$ RDF) grafa  $G$  funkcija  $f$ , ki slika iz  $V(G)$  v množico  $\{1, \dots, k\}$ . Zanja 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

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

Graf  $K_k$  predstavlja polni graf na  $k$  vozliščih. Nazadnje definirajmo še '*k-rainbow total domination function*' ( $k$ RTDF), 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

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

## 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* predstavlja 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  $n$  predstavlja 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  $k$  predstavlja 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  $k$ .

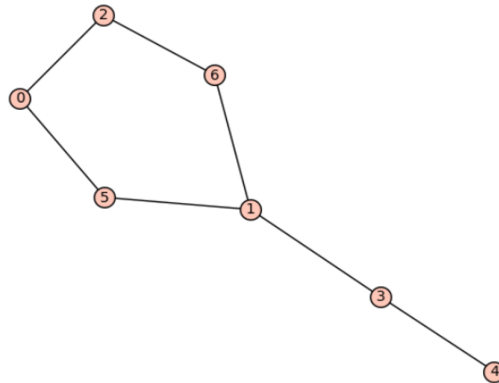
Funkcija *cartesian\_product()* porabi  $\mathcal{O}(nm)$  časa, saj vsakemu vozlišču iz grafa  $G$  ( $n$  vozlišč) priredi vsako vozlišče iz  $k$ -polnega grafa ( $m$  vozlišč).

Torej je časovna zahtevnost te pseudokode enaka  $\mathcal{O}(n^2m^2)$ .

To kodo sva uporabila za  $n \in \{5, \dots, 20\}$  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.:

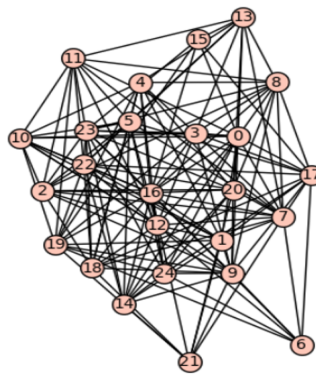
(1.6666666666666667, 13)

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



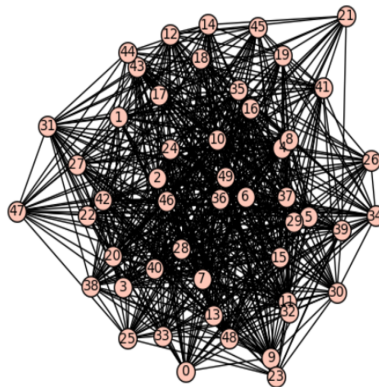
(1.3333333333333333, 498)

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

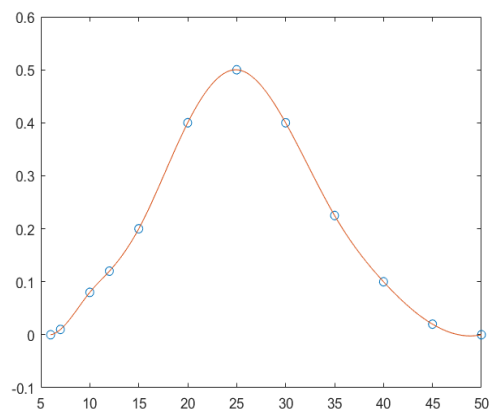


(1.6666666666666667, 11)

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

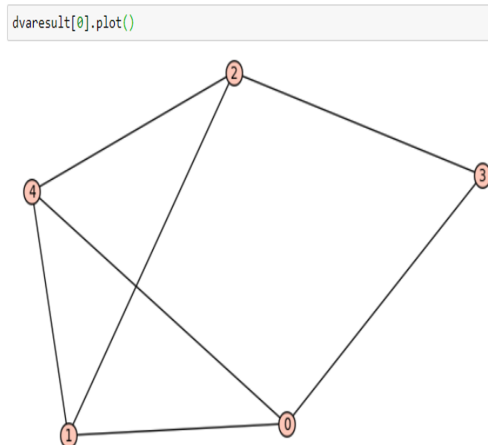


Koda je za vse zgornje grafe preverila na 1000 naključnih grafi. Pognala sva jo večkrat in na različnem številu vozlišč. Vse skupaj je koda pregledala približno  $10^{100}$  grafov, izključno za  $k = 2$ . Podatke sva izpisala v program *Matlab*, jih za posamezno število vozlišč povprečila in jih aproksimirala z že vgrajeno funkcijo *interp1*. Dobila sva spodnji graf.

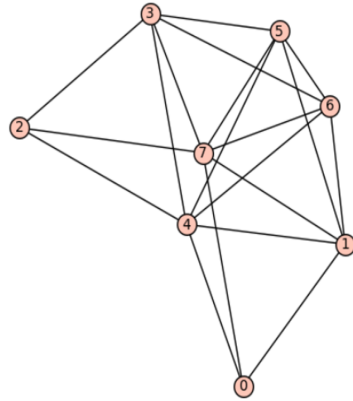


Z grafa je vidno, da protiprimerov ne dobimo za grafe z manj kot 7 vozlišč, za večje grafe pa velja, da gre število protiprimerov proti nič, ko se večja število vozlišč, oziroma od 50 vozlišč naprej ostaja število zelo majhno. Največ pa jih je pri grafi s približno 25 vozlišči.

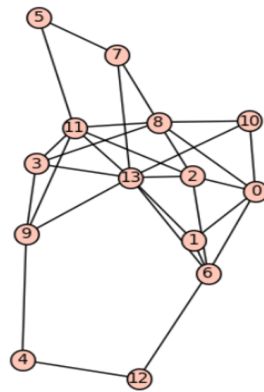
Spodaj je še nekaj primerov majhnih grafov za  $k \geq 4$ .



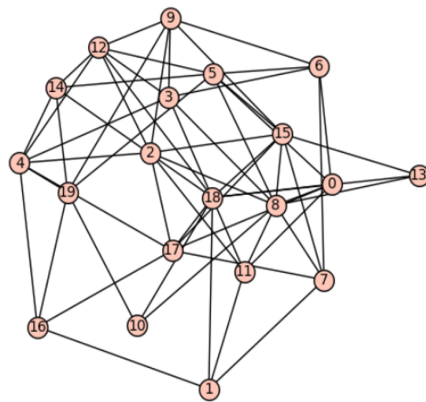
```
dvaresult[0].plot()
```



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



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



## 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

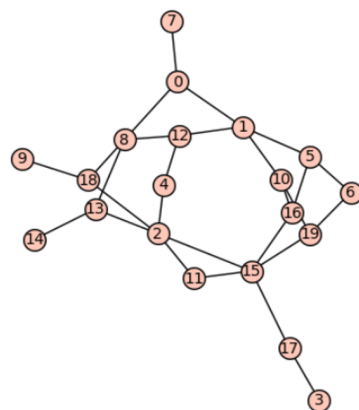
p = 0
stevilo_korakov = 2000
najboljsi = G
kk = k_min = krit(G)
for p in range(0, stevilo_korakov):
    Temp = stevilo_korakov / (p+1)
    novi = spremeni_graf(G)
    kn = krit(novi)
    razlika = kk - kn
    if razlika > 0:
        G = novi
        kk = kn
        if kn < k_min:
            najboljsi = 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 odstrani 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 podlagi najinih poizkusov sva ugotovila, da za grafe s sodim številom vozlišč, za vsak  $k \geq 4$ , je koeficient enak 2, za grafe z lihim številom vozlišč pa sva domnevala, po nekaterih poizkusih iz manjših grafov, da velja formula  $\frac{n}{\lfloor \frac{n}{2} \rfloor}$ . Potem sva pognala kodo na grafu z neparnim številom vozlišč, kjer sva dobila koeficient 2, se pravi protiprimer najine domneve. Sploh pri velikih grafih se tudi opazi, kako število povezav v grafu vpliva na čas, ki ga algoritem potrebuje za izračun željenega koeficienta. Namreč pri grafih z manj povezavami je koeficient res enak 2, a je čas računanja neprimerljivo daljši, najverjetneje zaradi funkcije *spremeni\_graf*, ki se mora v zelo redkem grafu velikokrat ponoviti, saj pogosto vrne graf, ki ni povezan, kar jo občutno upočasni.

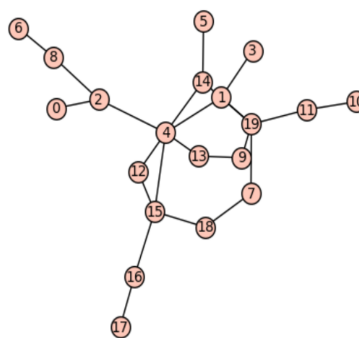


$k = 4$  in  $p = 0.1$



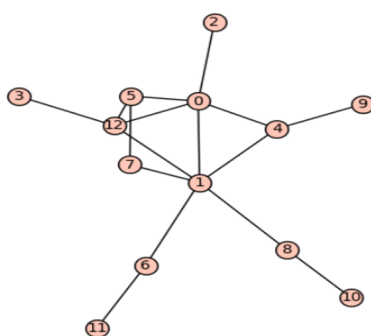
(None, 2)

$k = 5$  in  $p = 0.1$



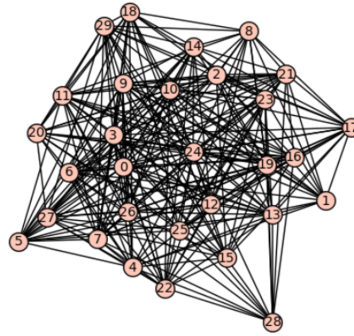
(None, 2)

$k = 6$  in  $p = 0.1$



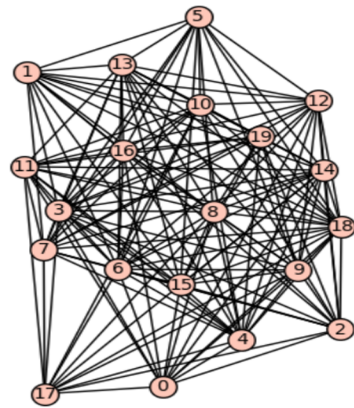
(None, 2)

$k = 4$  in  $p = 0.5$



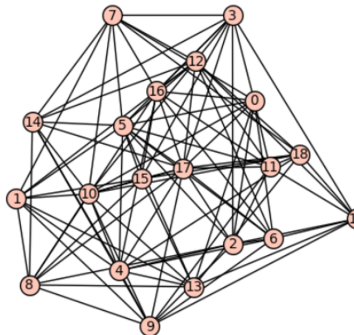
(None, 7/3)

$k = 4$  in  $p = 0.5$



(None, 2)

$k = 6$  in  $p = 0.5$

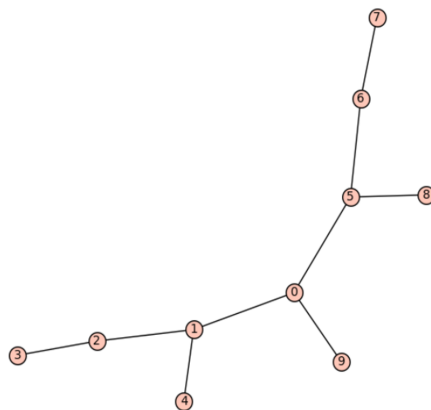


(None, 3)

### 3.3 Drevesa

Poizkuse sva izvajala tudi na povezanih drevesih, katere sva lahko preverila v celoti na večjem številu vozlišč, saj jih je samih po sebi manj, kot pa npr. vseh možnih povezanih grafov na istem številu vozlišč. Testirala sva za  $k$  med 4 in 12, ter število vozlišč do 20. Tu sva opazila, podobne stvari kot pri majhnih grafih, iz česar sklepava, da manj ko ima graf povezav (je podoben drevesu), večja je verjetnost, da se koeficient približa 2.

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



### 3.4 Dvodelni grafi

Posebej sva kodo izvajala še na dvodelnih drevesih z vozlišči  $n \in \{1, \dots, 20\}$ , kjer sva ugotovila, da formula  $\frac{n}{\lfloor \frac{n}{2} \rfloor}$  velja za vsa vozlišča od 1 do vključno s 3, nato pa je koeficient zmeraj enak 2.

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

