Naslov Skupina 19: Graffiti conjecture 232

Urban Merhar, Martin Kokošinek

1 Navodilo

Računalniško generirana domneva trdi: Če je G enostaven povezan graf, potem

$$2\gamma_t(G) \geq rad(G) + ecc(B).$$

Preveri domnevo na različne načine za male in velike grafe. Z uporabo populacijske metahevristike, preveri domnevo v upanju, da jo ovržeš.

Nekaj pripomb:

1. $ecc(v)$ je *ekscentričnost* od vozlišča v. Ekscentričnost od v je razdalja do najbolj oddaljenega vozlišča od vozlišča v, i.e., $max\{d(v,u) : u$ je vozlišče na grafu $\}$.
2. $rad(G)$ je *radij* grafa, t.j., minimum vseh ekscentričnosti vozlišč grafa G.
3. B je *obrobje* grafa G, t.j., množica vozlišč z maksimalno ekscentričnostjo.
4. $ecc(S)$ je *ekscentričnost* množice vozlišč S. Definirana je kot: Naj bo S podmnožica množice vozlišč V. Razdalja med vozliščem v in množico S, definirajmo kot razdaljo od v do najbližjega volišča v S. $ecc(S)$ je maksimum razadalj od vozlišča v $V \backslash S$ do množice S.

2 Kratek opis

Computer generated conjectures so računalniško ustvarjene domneve. $Graffiti$ je računalniški program, ki generira te matematične domneve oziroma odprte probleme. Računalniši program $Graffiti$ je ustvaril *Siemion Fajtlowicz*.

V najinem projektu pri predmetu Finančni praktikum si bova ogledala $Graffiti$ *conjecture* 232, ki jo bova testirala za majhne in velike grafe v upanju, da najdeva protiprimer. Ideja je, da enačbo zapiševa v programskem jeziku *Sage* in generirava naključne grafe. Na vsakem od teh grafov pa predpostavko testirava.

Že vgrajene funkcije, ki jih bova uporabila v programu:

1. $dominating\_set(total = True, value\_only = True)$ vrne najmanjšo dominirajočo množico na grafu G.
2. $radius()$ vrne radij grafa G.
3. $eccentricity()$ vrne ekscentričnost vozlišča v.
4. $periphery()$ vrne množico vozlišč iz obrobja grafa G.

2.1 Razlaga pojmov

- Dominirajoča Množica D: D je množica, kjer je vsako vozlišče iz $G \backslash D$ sosed nekega vozlišča iz D.
- Totalno Dominirajoča množica (TDM): Dominirajoči množici D dodamo pogoj, da so tudi vozlišča dominirajoče množice D sosedi vozlišč iz D.
- Totalno Dominirajoče Število (TDŠ): Moč totalno dominirajoče množice grafa G.
- $\gamma_t(G)$ je TDŠ grafa G.

2.1.1 Populacijska metahevristika

Hevristika (iz Grščine: 'najdem, odkrijem'): V računalništvu in matematični optimizaciji je visoko-nivojski način reševanja problemov, ko so klasični postopki prepočasni oziroma, ko klasične metode ne vrnejo točnih rezultatov. V zameno za polnost, optimalnost, natančnost, raje pridobimo na časovni zahtevnosti.
Meta-hevristika (meta iz Grščine: 'za, onstran') oziroma v prevodu Izčrpna-hevristika: Metahevristika vzame množico rešitev, ki je prevelika za analizo in s pomočjo določenih predpostavk glede optimizacije vrne zadovoljivo rešitev. Ta ni nujno globalno optimalna.
Populacijska metahevristika: Ohranjamo večje število kandidatov za rešitev in jih izboljšujemo s pomočjo populacijsih karakteristik. Primer je particle swarm optimization (PSO).

3 Plan dela

Zapisati učinkovit algoritem, ki bo za vsak generiran graf preveril lastnosti grafa in posledično domnevo. Za grafe, kjer domneva ne bi držala pa nam izpiše graf in vrne vrednosti lastnosti, ki so potrebovane v domnevi.

4 Potek dela

Za preverjanje dane domneve sva napisala program, ki jo testira. Samo domnevo pa sva nekoliko obrnila na način, da lahko učinkovito gledava tudi njeno razliko. To sva storila z namenom da opazujeva kateri grafi so bližje temu, da bi bili lahko protiprimer domneve. Kot je vidno v spodnjem programu sva tako na grafih preverjala $2\gamma_t(G) - rad(G) - ecc(B) \geq 0$. Na ta način sva s pomočjo razlike $2\gamma_t(G) - rad(G) - ecc(B)$ ocenjevala kako blizu pride nek graf k temu, da bi bil protiprimer.

Funkcija z imenom $domneva(G)$ sprejme dan graf G in na njem testira domnevo ter vrne razliko $2\gamma_t(G) - rad(G) - ecc(B)$, če predpostavka velja, sicer pa pove, da predpostavka ne velja.

Program je predstavljen na spodnjih slikah:

Program za preverjanje domneve

```
def domneva(G):
    '''Funkcija, ki testira domneva Graffiti conjecture 232.'''
    if G.is_connected() == False:
        return('Graf ni povezan!')
    B = set(G.periphery())
    d = G.distance_all_pairs()
    razdalja = []
    for v in G:
        if v not in B:
            sezv = []
            for u in B:
                sezv.append(d[v][u])
            razdalja.append(min(sezv))
    razlika = 2 * G.dominating_set(total=True, value_only=True) - G.radius() - (max(razdalja) if razdalja else 0)
    if razlika >= 0:
        print('Razlika med levo in desno stranjo domneve je:')
        print(razlika)
        return('Domneva velja!')
    else:
        return('Domneva NE velja!')
```

Posamezne funkcije v programu

```
# total = True da bo total dominating set
# value_only = True -> ker nas zanima samo vrednost in ne seznam vozlisc
G.dominating_set(total=True, value_only=True)
```

```
# radij grafa
G.radius()
```

```
# B je obrobje grafa
B = set(G.periphery())
d = G.distance_all_pairs()
razdalja = []
for v in G:
    if v not in B:
        sezv = []
        for u in B:
            sezv.append(d[v][u])
        razdalja.append(min(sezv))

# razdalja je ekscentricnost obrobja grafa G

(max(razdalja) if razdalja else 0)
```

```
# Preizkusimo funkcijo na grafu G:
domneva(G)
```

5 Majhni grafi

Majhne grafe sva v *sagu* generirala s pomočjo funkcije $graphs.nauty\_geng('v\ -\ c')$. Kjer je v število vozlišč grafa, $-c$ pa pomeni, da so grafi povezani. Na dovolj majhnih grafih sva generirala vse možne enostavne in povezane grafe ter na njih testirala predpostavko.

Generator malih grafov

```
# GENERIRANJE MAJHNIH GRAFOV
# ker grafe generiras z nauty_geng so vsi preprosti (po dokumentaciji)
seznam = [G for G in graphs.nauty_geng('n -c')] # povezani grafi na "n" vozliscih, "-c" pomeni da so grafi povezani
```

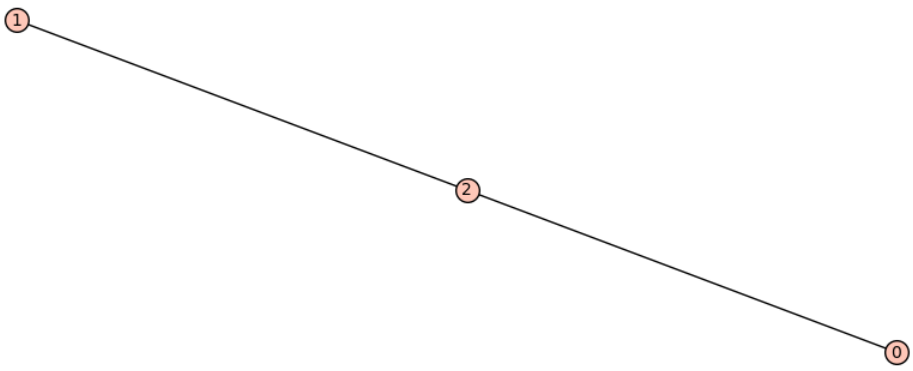
```
protiprimeri = []
for graf in seznam:
    if domneva(graf) == 'Domneva NE velja!':
        graf.show()
        protiprimeri.append(graf)
```

```
# Ali obstaja protiprimer?
protiprimeri
```

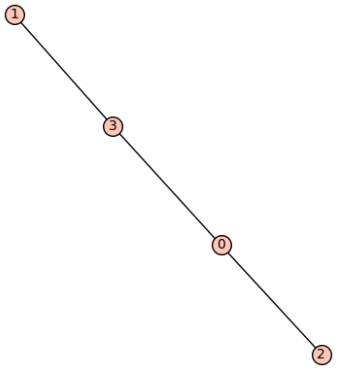
Vsi grafi so ji ustrezali, zato sva se odločila podrobneje pogledati grafe, kjer je bila razlika med levo in desno stranjo predpostavke najmanjša.

Graf na dveh vozliščih. Razlika v predpostavki je $2\gamma_t(G) - rad(G) - ecc(B) = 3$.

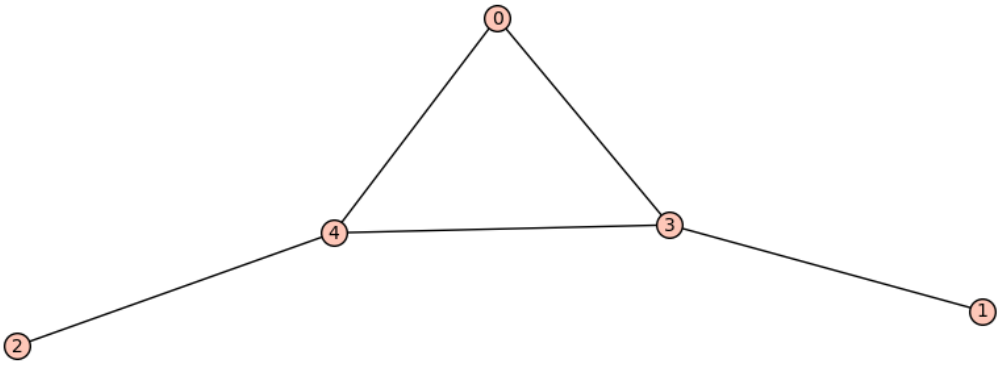
Graf na treh vozliščih. Razlika je 2.



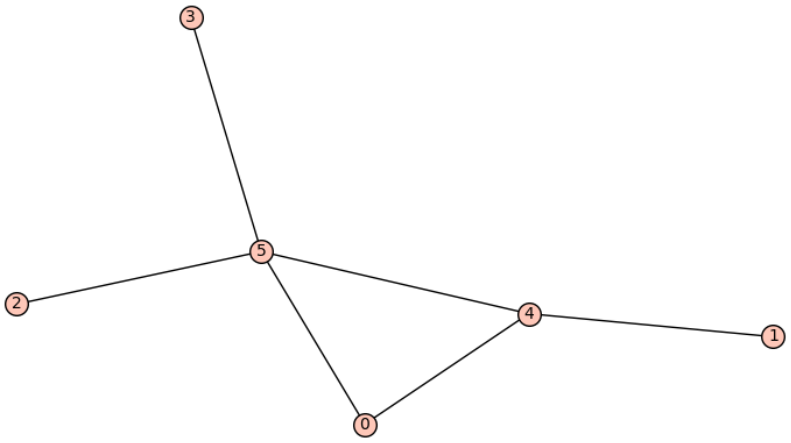
Grafi na štirih vozliščih imajo razliko predpostavke enako 1.

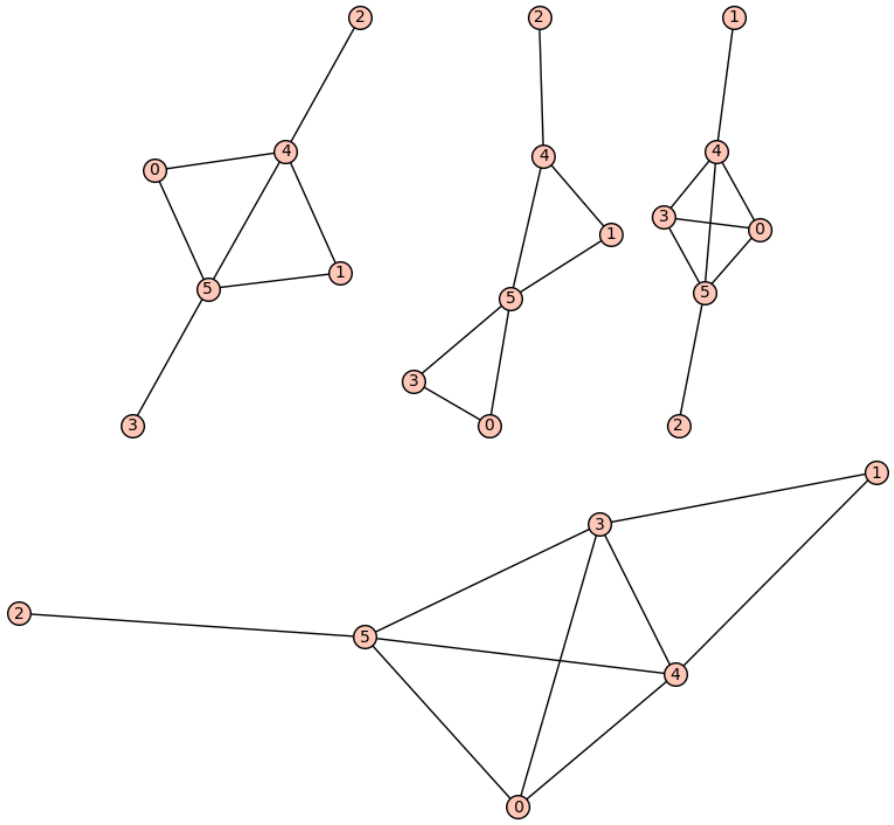


Pri grafih generiranih na petih vozliščih dobiš graf katerega razlika predpostavke je 0. Generiranih je skupno 21 grafov na pet vozliščih. Program testira vse te grafe v manj kot 0.2 sekunde.

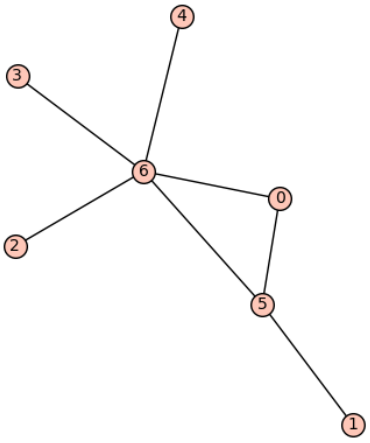


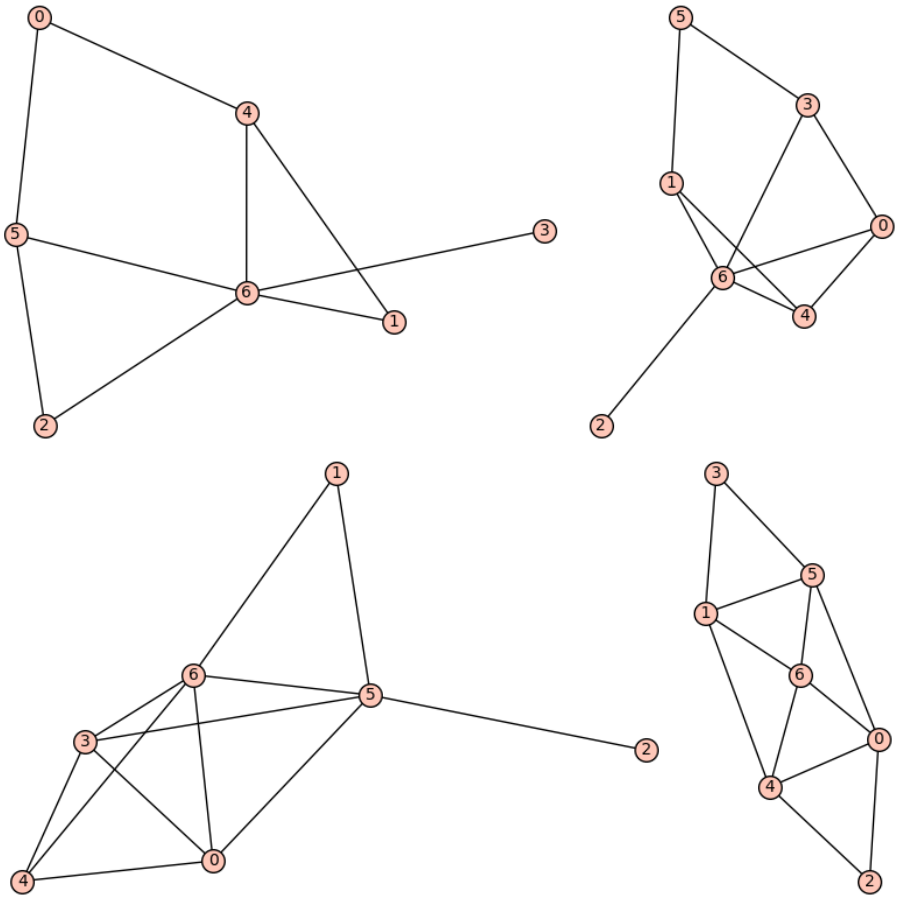
Pri grafih na šest vozliščih je razlika ponovno 0. Od skupno 112 vseh možnih grafov na šest vozliščih ima razliko 0 natanko 8 grafov. Program testira vse šestvozliščne grafe v približno 0.8 sekunde. Spodaj je prikazanih pet primerov.



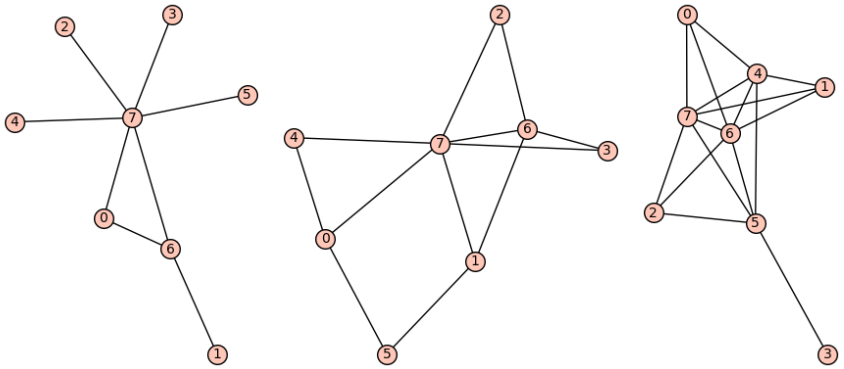


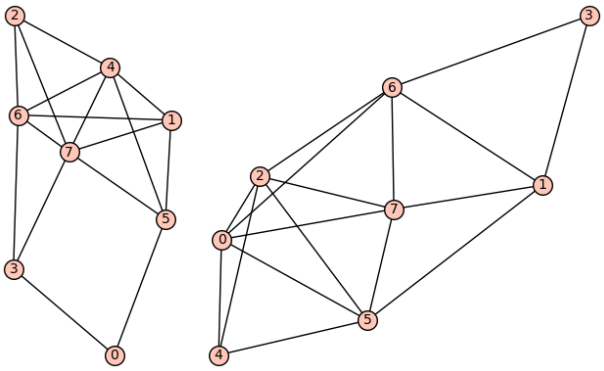
Za grafe na sedem vozliščih je minimalna razlika predpostavke ponovno 0. Sedaj je možno generirati 853 različnih grafov, razliko 0 pa jih ima 101. Program testira to predpostavko na vseh generiranih grafih na sedmih vozliščih v približno 3.7 sekunde. Ponovno si poglejmo nekaj primerov teh grafov.





Grafi na osmih vozliščih imajo razliko leve in desne strani ponovno enako 0. Zdaj generiramo že 11117 različnih grafov in za njihovo testiranje porabimo že kar približno 40 sekund. Grafov z osmimi vozlišči, kjer je razlika predpostavke enaka 0 je 1662.





Z grafi na osmih vozliščih se konča naše preizkušanje majhnih grafov. Na tej točki računalnik ni več sposoben generirati vseh preprostih povezanih grafov na devetih vozliščih preko generatorja $graphs.nauty\_geng('v\ -c')$. Portal $CoCalc$ sam terminira generator, če je ta zagnan.

Zaključimo lahko, da dana domneva $Graffiti\ conjecture$ 232 vsekakor velja na majhnih grafih z osem ali manj vozlišči.

6 Veliki grafi

Za velike grafe ($n \leq 50$) sva uporabila vgrajeno funkcijo $graphs.RandomGNP(n, p)$. Ta funkcija generira graf z n vozlišči in ustvari povezavo med dvema vozliščema z verjetnostjo p. Funkcijo preverjanje domneve sva nekoliko modificirala in nato ustvarila generator grafov za vozlišča za $n = 50, 75, 100$ in verjetnosti $p = 0.33, 0.5, 0.66$. Vsako kombinacijo sva preverila 1000-krat. Tak program porabi približno 1 uro in 10 minut za izračun. Čas je bil izmerjen s pomočjo knjižnice *datetime*. Program je shranjeval najmanjšo razliko, ki jo je našel. Če je bila ta manjša ali enaka 2 je graf shranil v seznam $mala\_razlika$.

Preverjanje domneve za velike grafe

```
def domneva_veliki(G):
    if G.is_connected() == False:
        return None
    B = set(G.periphery())
    d = G.distance_all_pairs()
    razdalja = []
    for v in G:
        if v not in B:
            sezv = []
            for u in B:
                sezv.append(d[v][u])
            razdalja.append(min(sezv))
    razlika = 2 * G.dominating_set(total=True, value_only=True) - G.radius() - (max(razdalja) if razdalja else 0)
    if razlika < 0:
        return False
    else: return razlika
```

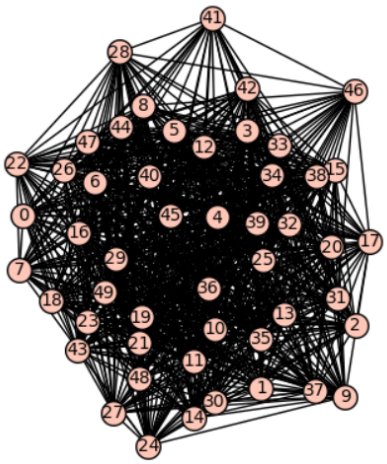
Generator velikih grafov

```
import datetime
print(datetime.datetime.now())

razlika50 = []
razlika75 = []
razlika100 = []

st_preverjanj = 1000

razlika = 10000
protiprimer = []
mala_razlika = []

for i in [50,75,100]:
    for j in [0.33,0.5,0.66]
        for k in range(st_preverjanj):
            G1 = graphs.RandomGNP(i, j)
            vrednost = domneva_veliki(G1)
            if vrednost == False:
                protiprimer.append(G1)
            elif vrednost == 0:
                if i == 50:
                    razlika50.append(G1)
                elif i == 75:
                    razlika75.append(G1)
                else:
                    razlika100.append(G1)
            if vrednost < razlika:
                razlika = vrednost
                if razlika <= 2:
                    mala_razlika.append(G1)

print(datetime.datetime.now())
```

Program sva pognala večkrat, v končnem sva preverila več kot 30000 grafov (ne nujno različnih). Nikoli se ni zgodilo, da bi obstajal protiprimer, še več razlika je bila vedno večja od 0. Pri grafih $n = 50$ in $p = 0.66$ se je pojavil primer, ko je bila razlika 2. To je najmanjša razlika, kar se jih je pojavilo pri generiranju.

```
print(razlika)
print(mala_razlika)
G = mala_razlika[0]
G.show()
```

```
2
[RandomGNP(50,0.660000000000000): Graph on 50 vertices]
```



Na koncu sva preverila še 3500 grafov z $n = 200$ in $p = 0.66$, kar je trajalo približno 5 ur. Razlika je bila 4, ta razlika je bila tudi najbolj pogosta pri vseh generiranjih.

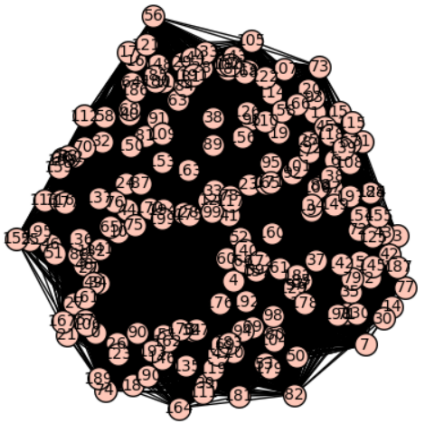
```
razlika
```

```
4
```

```
mala_razlika
```

```
[RandomGNP(200,0.666600000000000): Graph on 200 vertices]
```

```
mala_razlika[0].show()
```



6.1 Simulated-annealing

Po direktni analizi sva začela z metodo Simulated-annealing. Ideja je, da generiramo naključen graf z n vozlišči, preverimo domnevo, nakar izberemo naključni vozlišči in dodamo povezavo, če ne obstaja oziroma odstranimo povezavo, če obstaja. Potem še enkrat preverimo domnevo na spremenjenem grafu, če je rezultat enak ali boljši (neenakost bližje 0) potem nadaljujemo na novem grafu, sicer odstranimo spremembo in ponovno izberemo naključni vozlišči. To nardimo za k korakov nakar vrnemo graf in rezultat neenakosti domneve. V primeru, da bi našli protiprimer, program takoj konča in izpiše rezultat.

Simulated-anealing

```
import datetime
print(datetime.datetime.now())

def simulacija():
    rez = 100
    G = graphs.RandomGNP(100,0.4)
    k = 0
    while k < 500:
        if G.is_connected()== False:
            k+=1

        v1 = choice(list(range(n)))
        v2 = choice(list(range(n)))
        if v1 != v2:
            if v2 not in G.edges()[v1]:
                G.add_edge(v1,v2)
                razlika = domneva(G)
                if razlika == False:
                    return [G,razlika]
                elif razlika <= rez:
                    rez = razlika
                    k+=1
                else:
                    G.add_edge(v1,v2)
                    k+=1
            else:
                G.delete_edge(v1,v2)
                razlika = domneva(G)
                if razlika == None:
                    k+=1
                elif razlika == False:
                    return [G,razlika]
                elif razlika <= rez:
                    rez = razlika
                    k+=1
                else:
                    G.add_edge(v1,v2)
                    k+=1
        else:
            k+=1
    return [G,rez]
G,rez = simulacija()
print(datetime.datetime.now())
```

Preverjala sva grafe velikosti 10 (rezultat 2), 20 (rezultat 2) in 50 (rezultat 2), kjer sva uporabljala število korakov $k = 2000$. Izkazalo se je, da večje število korakov ni vračalo manjših rezultatov. Potem sva izvajala program še za 100 in 200 vozlišč. Že od prej vemo, da testiranje domneve na 200 vozliščih za 3500 grafov traja približno 5 ur. Zato sva število korakov zmanjšala na $k = 700$ in s tem čas na 1 uro. Za 100 vozlišč pa 500 korakov traja približno 20 minut, torej 1500 traja približno 1 uro, zato $k = 1500$. Verjetnost povezave med vozliščema pri generiranju grafa je bila $p = 0.8$, saj sva pri direktni analizi dobivala boljše rezultate, kot pri manjših verjetnostih. Rezultati so bili boljši kot pri direktni analizi, saj sva tudi za $n = 100, 200$ našla primere, kjer je bila razlika enaka 2. Spodaj so navedeni primeri grafov kjer so bile razlike majhne, pri $n = 10$ je 0, pri 15, 20, 100.