UNIVERZA V LJUBLJANI
FAKULTETA ZA MATEMATIKO IN FIZIKO

Finančna matematika – 1. stopnja

Martin Praček in Mela Malej

Graffiti Conjecture 194

Seminarska naloga

Mentor: Riste Škrekovski

Ljubljana, 2019

Kazalo

Graffiti Conjecture 194

Povzetek
Za predmet Finančni praktikum v študijskem letu 2018/19 Martin Praček in Mela Malej dobila nalogo predstaviti problem Grafittijeve konjekuture 194.

1. Predstavitev problema

Grafittijeva konjektura 194 nam zastavi vprašanje iz teorije grafov.Zanima nas, ali za vsak preprost, povezan graf, ki zadošča pogoju

$$\alpha(G) \leq 1 + \lambda_{avg}(G)$$

, velja, da obstaja hamiltonska pot. Gre za računalniško generirano trditev, pri kateri nas zanima, ali lahko najdemo protiprimer.

2. Pogoji našega problema

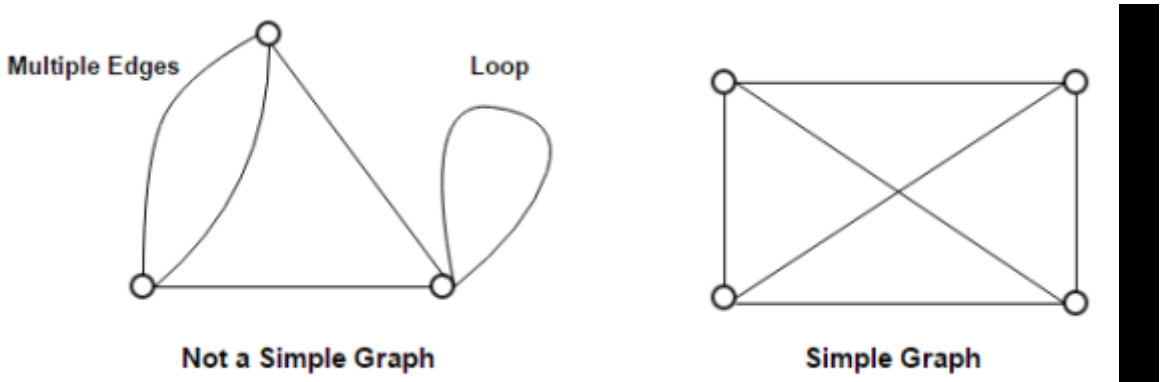
Naš pogoj je torej

$$\alpha(G) \leq 1 + \lambda_{avg}(G),$$

kjer je G naš graf, $\alpha(G)$ je neodvistnostno število, λ_{avg} pa je povprečna lokalna neodvistnost grafa.

Neodvistnostno število grafa $\alpha(G)$ nam pove moč največje množice, ki vsebuje vozlišča grafa G, od katerih nobeni dve niste sosednji.

Lokalna neodvistnost grafa $\lambda(G, v)$ nam pove neodvistnostno število podgrafa Gv grafa G, kjer je Gv definiran na sosedih vozlišča v. $\lambda_{avg}(G)$ nam pove povprečno lokalno neodvistnost.



Ves čas zahtevamo, da je graf **preprost in povezan**. Ta zahteva pomeni, da ne obstaja vozlišče, ki ne bi bilo sosednje vsaj enemu drugemu vozlišču, poleg tega pa med dvema povezavama obstaja kvečjemu ena povezava in ne obstaja povezave od vozlišča v do vozlišča v, kar pomeni da ne obstajajo zanke na enem vozlišču.

Graf ima **hamiltonsko pot**, če obstajata dve vozlišči, ki ju povezuje pot, ki natančno enkrat obišče vsako vozlišče grafa.

Najina naloga je torej na dovolj velikem vzorcu grafov pokazati da to velja, ali pa dobiti protiprimer, ki bo dokazal da to ne drži.

3. Delo

Z najinim delom sva pričela v programskem jeziku Python, vendar sva hitro ugotovila sizifovstvo najinega dela, in vse prestavila v programski jezik Sage, kjer sva v CoCalc spisala najino kodo.
V začetku sva zaradi slabega razumevanja enega izmed virov sicer mislila, da bova lahko najino kodo uporabila tudi za sorazmerno velike grafe, velike tudi do 20 vozlišč, a sva nato ugotovila, da to ne bo šlo in potrebujeva genetski algoritem.

4. Koda in ideja programa

Glavna ideja najine kode sledi glavni ideji najine naloge. Po vrsti sva razvrstila vse koščke neenakosti, in nato pogledala, če za primerne grafe le ta tudi velja. Tako sva dobila vrsto pomožnih ter eno glavno funkcijo.

4.1. **Pomožne funkcije.**

Kot že rečeno, sva v okvirju najinega dela uporabila vrsto pomožnih funkcij.

4.1.1. *neodvistnostno_stevilo(G).*

Ta funkcija nam vrne neodvistnostno število grafa G. Tukaj smo si pomagali z funkcijo *.independent_set*(), ki vrne največjo množico, kjer nobeni vozlišči nista sosednji.

```
def neodvistnostno_stevilo(G):
"""
Vrne neodvistno število grafa G
Za pomoc uporabimo independet_set() iz modula neusmerjenih grafov.
"""
neodvisno = G.independent_set()
dolzina = len(neodvisno)
return dolzina
```

4.1.2. *naredi_podgraf(G, seznam).*

Naredi podgraf grafa G na vozliščih, ki so del seznama G.

```
def naredi_podgraf(G, seznam):
"""
Funkcija, ki nam za dan seznam vozlisc vrne podgraf,
definiran na le teh.
Seznam je tukaj nabor vozlisc, ki nas zanimajo.
"""
nov_graf = dict()
for vozlisce in G:
    if vozlisce not in seznam:
        G.pop(vozlisce)
    else:
        for sosed in G[vozlisce]:
            sosedi = G[vozlisce]
            if sosed not in seznam:
                sosedi.remove(sosed)
        nov_graf[sosed] = sosedi
return nov_graf
```

4.1.3. *lokalna_neodvistnost(G, vozlisce).* Ta funkcija nam vrne lokalno neodvistnost grafa G.

```
def lokalna_neodvistnost(G, vozlisce):
"""
Vrne lokalno neodvistnost grafa G v vozliscu v.
Lokalna neodvistnost je neodvistnost podgrafa,
ki ga določajo sosedi vozlišča v,
za nek v iz množice vozlišč.
"""
mnozica = G[vozlisce]
novGraf = naredi_podgraf(G, mnozica)
lokalna_neodvistnost = neodvisnost(novGraf)
return lokalna
```

4.1.4. *povprecna(G).* Ta funkcija nam vrne povprečno lokalno neodvistnost.

```
def povprecna(G):
"""
Vrne povprečno lokalno neodvisnost grafa G
"""
povprecje =  0
for vozlisce in G:
    povprecje += lokalna_neodvisnost(G, vozlisce)
povprecna_vrednost  = povprecje/ len(G)
return povprecna_vrednost
```

4.2. **Glavna koda.** Za vsak slučaj sva si pripravila dve vrsti glavne kode, in sicer eno, ki je uporabna le na enem grafu in tisto, ki je uporabna za večje število grafov, namreč vse grafe enake velikosti.

```
def preverjanje_za_en_graf(G):
"""
Preveri, ali za en graf velja, če je ta graf izjema.
"""
if neodvisnostno_stevilo(G) <= 1 + povprecna(G):
   if hamiltonian.path(G) == None:
       graf = str(G)
       return "Ovrgli smo domnevo in izjema je" +  graf
   else:
       return "Izjeme nismo ovrgli"

def preverjanje(stevilo_vozlisc):
"""
Preveri veljavnost konjekture za vse grafe na določenem številu vozlišč.
"""
for G in nasi_grafi(stevilo_vozlisc):
    if neodvisnostno_stevilo(G) <= 1 + povprecna(G):
       if hamiltonian.path(G) == None:
          graf = str(G)
          return "Ovrgli smo domnevo in izjema je" +  graf
       else:
```

```
        return "Izjeme nismo ovrgli"
```

Na ta način smo oblikovali funkcije, s katerimi smo testirali grafe, za katere časovna zahtevnost ni bila prevelika, torej grafe do 8 vozlišč. Za večje grafe smo uporabili genetski algoritem.

4.3. **Genetski algoritem.** Genetski algoritem je vrsta algoritma, ki temelji na Darwinovi ideji o razvoju vrst in preživetju najmočnejših. Zelo pomembno pa je, kako določimo, kdo je tu najmočnejši, zato se moramo primerno odločiti za kriterij, po katerem bomo to delali. V našem primeru smo se odločili za čim manjše neodvisnostno število.

V našem primeru sva se odločila, da bo populacija rasla iz $zacetne\_populacija(maksimalna)$, kjer določimo začetno množico povezanih grafov. Z vrednostjo maksimalna smo določili največjo velikost generacije.
Nato sva te grafe razvrstila po določenem kriteriju in določila možnosti mutacije "populacije", torej vsakega izmed grafov. Grafu lahko dodamo ali odstranimo povezavo in dodamo ali odstranimo vozlišče, na koncu pa jih lahko še križamo.
Ko sva to naredila, sva najin program testirala.