UNIVERZA V LJUBLJANI
FAKULTETA ZA MATEMATIKO IN FIZIKO

Finančna matematika

Marcel Blagotinšek, Peter Milivojević

Maximum number of edges in a connected graph with n vertices and diameter d

Skupinski projekt

Poročilo

Advisers: doc. dr. Janoš Vidali,
prof. dr. Riste Škrekovski

Ljubljana, 2023

1. Navodilo naloge

A connected graph with diameter d on n vertices with the minimal number of edges will be a tree and henceforth, it will have $n-1$ edges. It will be harder to answer which graphs on a fixed number of vertices n and fixed diameter d have the maximal number of edges. We want to analyse the structure of such graphs. So, for a fixed number of vertices n and a fixed diameter d, when these two values are small, apply an exhaustive search. Next, for larger n and d, apply some metaheuristic. Try to obtain some specific properties of these graphs. Verify for how large n and d your exhaustive search and your metaheuristic implementations are efficent.

2. Opis problema

Želiva poiskati povezane grafe na n vozliščih s premerom d, ki bodo imeli maksimalno število povezav. Najin cilj je, na podlagi testiranja oz. generiranja, pridobiti kar se da dober vpogled v strukturo teh grafov in posledično ugotoviti, če za njih veljajo kakšne posebne lastnosti. Za majhne vrednosti n in d, se bova problema lotila z generiranjem grafov, za večje pa bova uporabila metodo simulated annealing. Ugotavljala bova tudi efektivnost najinih metod v odvisnosti od vrednosti n in d.

3. Potek Dela

Ideja prve faze projekta t.i. exhaustive search-a je, da z generiranjem vseh možnih povezanih grafov na n vozliščih s premerom d, poiščeva tiste, ki imajo maksimalno število povezav. To bova počela za majhne vrednosti n in d. Kako majhne, bo odvisno od časovne zahtevnosti samega algoritma, kajti je pričakovati, da bo že pri ne malo od 5 večjih vrednostih n algoritem počasen. Na podlagi generiranja grafov za različne n in d bova poskušala ugotoviti kakšne lastnosti, tako strukturne kot vizualne, lahko pripiševa tem grafom. Naraščanje/padanje števila povezav v odvisnosti od števila vozlišč oz. premetra bova prikazala tudi s pomočjo grafa, ki se bo morda obnašal podobno kot kakšna znana funkcija, kar bo vsekakor pomagalo pri oceni števila povezav za večje vrednosti n in d. Kot omenjeno bova poskusila najti kakšno formulo za maksimalno število povezav pri številu vozlišč n in premeru d. Tako pridobljene formule, četudi bo morda držala, ne bova dokazovala in jo bova posledično uporabila kot oceno v primeru generiranja grafov. Na koncu bova poleg ugotovitev glede lastnosti grafov v poročilu zapisala tudi pri kako velikih vrednostih n in d je najin algoritem prenehal učinkovito delovati. V drugi fazi projekta se bova problema lotila z metahevrističnо metodo simulated annealing. Začela bova z nekim začetnim povezanim grafom G, ki bo ustrezal pogojem n in d, nato pa bova dodala povezavo iz množice povezav komplementa grafa G. V kolikor bo premer grafa $G+e$ ostal isti, imamo nov graf, ki ima isti premer vendar povezavo več. Če bo premer novega grafa manjši od d, pa bova poiskala vozlišči u in v na maksimalni razdalji in odstranjevala povezave iz poti med u in v toliko časa, dokler ne bo premer spet d. Seveda se lahko zgodi, da bo premer večji od d, takrat pa bova spet poiskala vozlišči u in v na maksimalni razdalji in dodajala neke povezave na poti med u in v toliko časa, dokler ne bo premer spet d. Povezave bova morala dodajati med ustreznimi vozlišči. Torej, če bo nov premer $d-1$, bova dodala povezavo med vozliščema na oddaljenosti 2. Pri tem se zavedava, da z neko verjetnostjo v nekem koraku vzameva graf z manj povezavami, ki pa je morda boljše izhodišče za naprej. Tudi tukaj bova začela na manjših vrednostih, in s tem preveriva, če najin algoritem deluje, nato

pa n in d povečujeva. Tudi v drugi fazi projekta bova pozorna na efektivnost oz. časovno zahtevnost, ter bova ugotovitve glede tega zapisala v poročilu.
Algoritme in programe bova v obeh fazah pisala v CoCalc Jupyter notebook-u.

4. Koda

4.1. 1. faza - koda:

```python
from sage.graphs.graph_generators import graphs

def najdi_graf_z_premerom(n, d):
    # Najvecje stevilo povezav in graf z najvec povezavami.
    max_povezave = 0
    graf_z_max_povezav = None

    # Zanka po vseh povezanih grafih z n vozlisci, ki jih generiramo z uporabo nauty_geng().
    for G in graphs.nauty_geng(str(n) + " -c"):

        # Premer grafa.
        premer = G.premer()

        # Ce srecamo graf katerega premer je enak nasemu premeru d
        if premer == d:
            # zabelezimo stevilo povezav
            stevilo_povezav = G.size()

            # Ce je stevilo povezav vecje od trenutnega maksimuma, posodobi maksimum.
            if stevilo_povezav > max_povezave:
                max_povezave = stevilo_povezav
                graf_z_max_povezav = G.copy()

    return graf_z_max_povezav, max_povezave

# Primer za neko stevilo vozlisc n in premer d.
n = 8
d = 3

# Poiscemo povezan graf z dolocenim stevilom vozlisc in premerom, ki bo imel maksimalno stevilo povezav.
graf_z_max_povezav, max_povezave = najdi_graf_z_premerom(n, d)


```

```
# Ce je graf najden ga prikazemo
if graf_z_max_povezav:
    print(f"Povezan␣graf␣z␣{n}␣vozlsci␣in␣premerom␣{d}␣s␣{max_povezave}␣povezavami:")
    print(graf_z_max_povezav)

    graf_z_max_povezav.show()
else:
    print(f"Graf␣z␣{n}␣vozlisci␣in␣premerom␣{d}␣ni␣bil␣najden␣.")


import pandas as pd
import matplotlib.pyplot as plt

rezultati = []

# Zanka za preiskovanje razlicnih kombinacij n in d, grafe z maksimalnim stevilom povezav shranjujemo v slovar
for n in range(1, 10):
    for d in range(1, n):
        graf_z_max_povezav, max_povezave = najdi_graf_z_premerom(n, d)
        rezultat_slovar = {
            'n': n,
            'd': d,
            'max_povezave': max_povezave
        }
        rezultati.append(rezultat_slovar)

# Prikazemo rezultate s tabelo
df = pd.DataFrame(rezultati)

print(df)

# Prikazemo tudi graf, ki predstavlja maksimalno stevilo povezav v odvisnosti od d za razlicne n
plt.figure(figsize=(10, 6))
for n in range(1, 9):
    podskupina = df[df['n'] == n]
    plt.plot(podskupina['d'], podskupina['max_povezave'], label=f'n={n}')
plt.xlabel('premer␣(d)')
plt.ylabel('maksimalno␣stevilo␣povezav')
plt.legend()
plt.title('max_povezave(d)␣za␣razlicne␣n')
plt.show()
```

```
import networkx as nx
import matplotlib.pyplot as plt
import random
from itertools import combinations
import math


def spodnja_meja(n, d):
if d >= n:
    return 'Izbrani␣premer␣je␣prevelik'
elif d < 1:
    return 'Izbrani␣premer␣je␣premajhen'
elif d == 1:
    G = nx.complete_graph(n)
    return G
else:
    G = nx.complete_graph(n - d + 1)
    new_node = n - d + 1
    for i in range(n - d):
        existing_node = i
        G.add_edge(new_node, existing_node)
    if d > 2:
        for i in range(n - d + 2, n):
            new_nodes = i
            G.add_edge(new_nodes, new_nodes - 1)
    return G


# Presteje stevilo povezav v grafu.
def ciljna_funkcija(graf):
    return len(graf.edges)


# Najde najkrajso mozno pot v grafu med zacetnim in
    koncnim vozliscem.
def najdi_pot(graf, zacetek, konec):
    try:
        pot = nx.shortest_path(graf, source=zacetek,
            target=konec)
        return pot
    except nx.NetworkXNoPath:
        return None


```

```
    # Kot argument sprejme graf ter pot iz katere zelimo odstranit povezavo, nato iz nje nakljucno odstrani povezavo.
    def odstrani_nakljucno_povezavo_iz_poti_v_grafu(graf, pot):
        nakljucni_indeks_povezave = random.randint(1, len(pot) - 1)
        povezava_za_odstranitev = (pot[nakljucni_indeks_povezave - 1], pot[nakljucni_indeks_povezave])
        graf.remove_edge(*povezava_za_odstranitev)
        return graf


    # METAHEVRISTIcNI ALGORITEM
    def simulirano_hlajenje_2_povezavi_razmaka_spodnja_meja(n, max_iteracij, zacetna_temperatura, stopnja_hlajenja, premer):
        trenutna_resitev = spodnja_meja(n, premer)
        najboljsa_resitev = trenutna_resitev.copy()
        temperatura = zacetna_temperatura

        for iteracija in range(max_iteracij):
            # Preverimo kaksen je premer, bodisi je vecji ali enak premeru trenutne resitve, bodisi pa je manjsi od 1. V zadnjem primeru je torej nepovezan graf. Dodamo povezavo.
            if premer <= nx.diameter(trenutna_resitev) or nx.diameter(trenutna_resitev) < 1:
                # Izbere 2 nakljucni vozlisci, ki nista povezani.
                vozlisce1 = random.choice(list(trenutna_resitev.nodes))
                vozlisca_2_razmaka = [vozlisce for vozlisce in trenutna_resitev.nodes - set([vozlisce1]) if nx.shortest_path_length(trenutna_resitev, source=vozlisce1, target=vozlisce) == 2]
                vozlisce2 = random.choice(vozlisca_2_razmaka)
                # Dodamo povezavo med izbranima vozliscema.
                nova_resitev = trenutna_resitev.copy()
                nova_resitev.add_edge(vozlisce1, vozlisce2)
                # Preverimo ali je nov graf tak, da ima vec povezav. V primeru da je to res, posodobimo najboljso resitev, sicer pa z verjetnostjo izberemo ali bomo posodobili trenutno resitev ali ne.
```

```
            # Opomba: Lahko pride do izbire "slabsega" grafa, upamo, da nas bo ta "slabsi" vseeno pripeljal do boljse resitve v nadaljevanju.
            delta = ciljna_funkcija(nova_resitev) - ciljna_funkcija(trenutna_resitev)
            if (delta > 0 and premer <= nx.diameter(nova_resitev)) or random.random() < math.exp(-delta / temperatura):
                trenutna_resitev = nova_resitev.copy()
                if ciljna_funkcija(nova_resitev) > ciljna_funkcija(najboljsa_resitev) and premer == nx.diameter(nova_resitev):
                    najboljsa_resitev = nova_resitev.copy()
        else:
            # Poiscemo kombinacije vozlisc z najvecjo ekscentricnostjo.
            ekscentricnosti = nx.eccentricity(trenutna_resitev)
            max_ekscentricnost = max(ekscentricnosti.values())
            vozlisca_z_max_ekscentricnostjo = [vozlisce for vozlisce, ekscentricnost in ekscentricnosti.items() if ekscentricnost == max_ekscentricnost]
            kombinacije_parov = list(combinations(vozlisca_z_max_ekscentricnostjo, 2))
            rezultat = []
            # Izberemo najbolj oddaljeni vozlisci.
            for i, j in kombinacije_parov:
                potencialen_rezultat = najdi_pot(trenutna_resitev, i, j)
                if potencialen_rezultat and len(potencialen_rezultat) > len(rezultat):
                    rezultat = potencialen_rezultat
            nova_resitev = trenutna_resitev.copy()
            nova_resitev = odstrani_nakljucno_povezavo_iz_poti_v_grafu(trenutna_resitev.copy(), rezultat)
            # Preverimo ali je nov graf tak, da ima vec povezav. V primeru da je to res, posodobimo najboljso resitev, sicer pa z verjetnostjo izberemo ali bomo posodobili trenutno resitev ali ne.
            # Opomba: Lahko pride do izbire "slabsega" grafa, upamo, da nas bo ta "slabsi" vseeno pripeljal do boljse resitve v nadaljevanju.
```

```
            delta = ciljna_funkcija(nova_resitev) - ciljna_funkcija(trenutna_resitev)
            # IF pogoj je vedno izpolnjen, pustimo ga zgolj za voljo testiranja.
            if ((delta > 0 and premer <= nx.diameter(nova_resitev)) or random.random() < math.exp(-delta / temperatura)) and nx.is_connected(nova_resitev):
                trenutna_resitev = nova_resitev.copy()
                if ciljna_funkcija(nova_resitev) > ciljna_funkcija(najboljsa_resitev) and premer == nx.diameter(nova_resitev):
                    najboljsa_resitev = nova_resitev.copy()
        # Znizamo(ohladimo) temperaturo po stopnji hlajenja.
        temperatura *= stopnja_hlajenja

    return najboljsa_resitev

# Prikaz delovanja algoritma na primeru.
st_vozlisc = 30
max_iteracij = 1000
zacetna_temperatura = 1.0
stopnja_hlajenja = 0.95
zeljen_premer = 7

najboljsi_graf_s_m = simulirano_hlajenje_2_povezavi_razmaka_spodnja_meja(st_vozlisc, max_iteracij, zacetna_temperatura, stopnja_hlajenja, zeljen_premer)
print(f"Stevilo povezav v najboljsem generiranem grafu: {ciljna_funkcija(najboljsi_graf_s_m)}")

# Graf se prikazemo.
plt.figure(figsize=(8, 8))
nx.draw(najboljsi_graf_s_m, with_labels=True, font_weight='bold', node_color='skyblue', node_size=800, font_size=10)
plt.show()
```

5. Ugotovitve

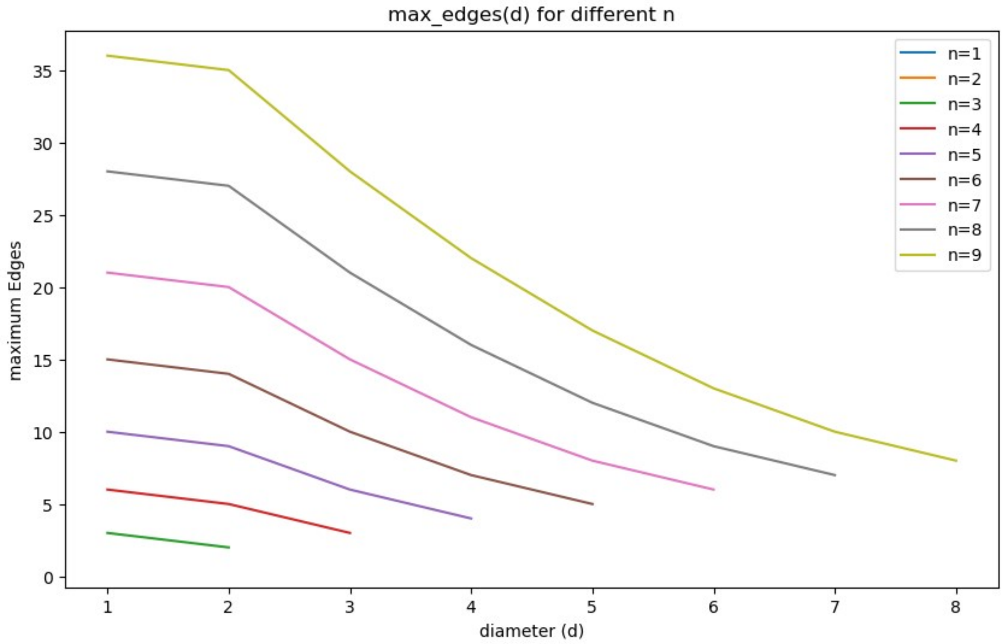
5.1. 1. faza - ugotovitve:

Za $d = 1$ ugotovimo, da je ne glede na izbiro števila vozlišč n, iskani graf ravno polni graf in ima posledično $\frac{n(n-1)}{2}$ povezav. V nasledjem koraku hitro ugotovimo, da se pri $d = 2$ število povezav zmanjša le za 1, saj se z odstranitvijo katere koli poljubne povezave v polnem grafu premer poveča na $d = 2$ in ker smo za to potrebovali odstraniti le eno samo povezavo je največje možno število povezav v grafu z n točkami in premerom $d = 2$ enako $\frac{n(n-1)}{2} - 1$. Podobno opazimo, da so grafi za premere $d = n - 1$ ravno drevesa s stopnjo 2 in je zato število povezav enako $n - 1$. Tako nas pri dani nalogi v resnici zanimajo predvsem grafi z $d \in \{3, \ldots, n-2\}$.
V prvi fazi sva pričela reševati z opazovanjem in računanjem grafov z manjšim številom vozlišč n, pri tem sva si pomagala tudi s kodo iz 4.1.

Napisani algoritem je z uporabo funkcije nauty geng generiral vse povezane grafe na n vozliščih, izločeval tiste, katerih premer ni bil enak d, ter posodabljal spremenljivko z maksimalnim številom povezav ter tem povezavam ustreznemu grafu. Podatke o največjem možnem številu povezav za grafe do 10 vozlišč sva zbrala v tabeli 1 in vrednosti prikazala na spodnjem grafu.

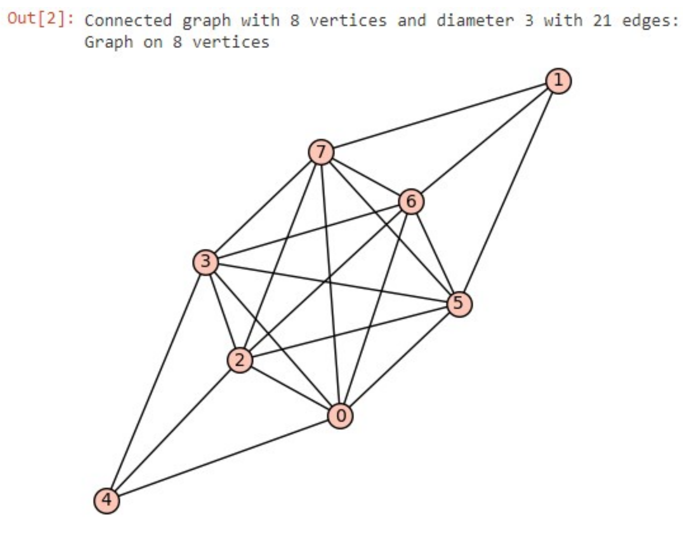
n\d	**Število povezav**								
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
2	1								
3	3	2							
4	6	5	3						
5	10	9	6	4					
6	15	14	10	7	5				
7	21	20	15	11	8	6			
8	28	27	21	16	12	9	7		
9	36	35	28	22	17	13	10	8	
10	45	44	36	29	23	18	14	11	9

Tabela 1. Število povezav glede na število vozlišč in premer.



Slika 1. Maksimalno število povezav v odvisnosti od d pri različnih n.

Z opazovanjem tabele sva na podlagi vzorca uspela za grafe z $d > 1$ zapisati formulo, ki nama pove maksimalno število povezav v grafu z n vozlišči in premerom d: $\frac{(n-d+1)(n-d)}{2} + n - 2$. Te formule ne bova dokazovala in jo bova v nadaljnem raziskovanju uporabljala kot oceno, saj vanjo brez dokaza ne moreva biti popolnoma prepričana.
Z nadaljnim opazovanjem generiranih grafov sva opazila, da vsi grafi vsebujejo poln podgraf velikosti $n - d + 1$. Na grafu prikazanem spodaj se to lepo vidi.



Slika 2. Graf z 8 vozlišči in premerom 3, ki vsebuje poln podgraf velikosti 6.

Glede učinkovitosti sva ugotovila, da je najin algoritem učinkovit za grafe z številom vozlišč do 9. Od tod naprej traja enostavno preveč časa. Že pri številu vozlišč enako 8, je povezanih grafov kar 251548592.