Univerza v Ljubljani

Fakulteta za matematiko in fiziko

Problem k-centrov

Filip Nose in Anja Plesec

Januar, 2021

Kazalo

1 Uvod

V poročilu bova predstavila problem k-centrov in kodo s katero sva ta problem rešila. Na začetku bova podrobneje opisala problem in kakšni so bili najini cilji, v nadaljevanju pa bova kodo podrobneje komentirala in predstavila rezultate do katerih sva prišla.

2 Naloga

Consider the k-center problem for graphs (using the distance for graphs). Use an ILP formulation an a good ILP solver to run experiments in simple graphs, like grids (with some deleted vertices or edges) or trees. Check the running time and the optimal value depending on the different parameters (k, number of vertices or edges being removed, the size of the grid, etc.) You can also consider 3-dimensional grids or other simple families of graphs.

3 Opis problema

Problem k-centrov (angl. *k-center problem*) govori o izbiri najboljše lokacije za k objektov na tak način, da minimiziramo največjo razdaljo med mesti, kjer je povpraševanje, ter najbljižjim krajem, kjer je objekt postavljen. Primeri teh objektov so gasilski in zdravstveni domovi, skladišča, šole itd.

Problem k-centrov lahko formuliramo z neusmerjenim grafom. Podan imamo graf $G = (V, E)$, kjer V predstavlja množico vozlišč danega grafa, E pa množico povezav. Poleg tega povezavam dodelimo pozitivne uteži d_{ij}, ki predstavljajo razdaljo med vozliščem i in vozliščem j. Cilj je najti množico $S \subseteq V$ in vozlišče $v \in V$, kjer je $|S| \leq$ k, tako da bo razdalja $\max_{v \in V} d(v, S)$ najmanjša.

S preprostejšimi besedami bi lahko rekli, da je cilj najti vozlišče, ki minimizira maksimalno razdaljo med vozliščem in centrom.

Za iskanje k-centrov bova uporabila sledeč CLP:

Vhodni podatki:

- d_{ij} ... razdalja med vozliščem i in centrom j
- k ... število centrov, ki jih moramo locirati

Spremenljivke:

- $y_i = 1$, če je v vozlišču i center
- $x_{ij} = 1$, če vozlišče i spada pod center j
- R ... maksimalna razdalja med vozliščem in najbližjim centrom

Iščemo torej

$$\max R$$

pri pogojih:

$$x_{ij} \in \{0,1\} \quad \forall i,j \in V$$

$$y_i \in \{0,1\} \quad \forall i \in V$$

$$\sum_{j \in V} x_{ij} \geq 1 \quad \forall i \in V$$

$$\sum_{j \in V} y_i = k$$

$$x_{ij} \leq y_j \quad \forall i,j \in V$$

$$\sum_{j \in n} x_{ij} d_{ij} \leq R \quad \forall i \in V$$

4 Cilji

Pri najinem projektu sva se osredotočila predvsem na mreže, katerim smo odstranili poljubna vozlišča in poljubne povezave. Napisala sva tudi kodo za drevesa in gozdove. Najin cilj je bil, da ugotoviva, kako se spreminja čas delovanja in optimalna vrednost ob spreminjanju:

- števila k,
- števila vozlišč ali povezav, ki sva jih odstranila in
- velikosti mreže.

5 Koda

V nadaljevanju so predstavljene funkcije za mreže. Podobne funkcije sva uporabljala za trodimenzionalne mreže in drevesa.

5.1 Mreža

V najini kodi sva najprej definirala funkcijo `mreza`, ki sprejme parametre m,n,a in b. S to funkcijo ustvarimo mrežo velikosti m×n, v kateri pa poljubno izbrišemo naključnih a vozlov in naključnih b povezav.

```
def mreza(m,n,a,b):
    mreza = graphs.Grid2dGraph(m,n)
    if a > mreza.order():
        print("Za ukaz je na voljo premalo vozlov.")
    else:
        i = 0
        while i < a:
            mreza.delete_vertex(mreza.random_vertex())
```

```
            i = i+1
        i = 0
    if b > mreza.size():
        print("Za ukaz je na voljo premalo povezav.")
    else:
        while i < b:
            mreza.delete_edge(mreza.random_edge())
            i = i+1
    return mreza
```

5.2 CLP

V nadaljevanju sva definirala funkcijo `najkrajsa_razdalja`, ki reši naš CLP problem. Vrne nam najkrajšo možno najdaljšo razdaljo v našem grafu, tako da razdalja od poljubnega vozlišča do najbližjega centra manjša ali enaka le tej. Funkciji moramo podati število centrov K in graf, ki ga ustvarimo s pomočjo funkcije `mreza`.

```
def najkrajsa_razdalja(G, st_centrov):
    K = st_centrov
    razdalje = G.distance_all_pairs()

    p = MixedIntegerLinearProgram(maximization=False)
    x = p.new_variable(binary=True)
    y = p.new_variable(binary=True)

    p.set_objective(p['R'])

    for u in G:
        p.add_constraint(sum(x[u, v] for v in G) == 1)

    p.add_constraint(sum(y[v] for v in G) == K)

    for u in G:
        for v in G:
            p.add_constraint(x[u, v] <= y[v])

    for u in G:
        for v in G:
            if v in razdalje[u]:
                p.add_constraint(razdalje[u][v] * x[u, v]
                    <= p['R'])
            else:
                p.add_constraint(x[u, v] == 0)
    max_razdalja = p.solve()
    skladisca = [k for k, v in p.get_values(y).items() if
        v == 1]
    return round(max_razdalja)
```

Od tu naprej so definirane funkcije, ki sva jih uporabila za izvajanje poskusov.

5.3 Optimalna vrednost v odvisnosti od k in velikosti kvadratne mreže

Zanima nas kako se spreminja povprečna optimalna vrednost R, ko spreminjamo velikost mreže in število centrov k.
Funkcija nam vrne matriko, kjer element a_{ij} predstavlja povprečno optimalno vrednost R (za n ponovitev) pri mreži $i \times i$, ki ima j centrov. Lahko povemo tudi koliko vozlišč in/ali povezav naj bo pri tem izbrisanih izbrisanih (za vse velikosti matrik enako št. izbrisanih vozlišč a in število izbrisanih povezav b).

Velikost mreže narašča od 3×3 do n×n.
K-ji naraščajo od 1 do max_k.

Koda je podobna tudi za mreže, kjer spreminjamo samo eno dimenzijo: m×3 do m×n, ki pa ni vključena v poročilo.

```
def R_v_odvisnosti_od_velikosti_kvadratne_mreze_in_k(n,a,
    b,max_k,stevilo_ponovitev):
     seznam = [] # i-ti element tega seznama pove
         povprecne R-je za razlicne k-je za ixi matriko
     for i in range(6, n+1):
          seznam1 = [] #seznam povprecnega R za
              stevilo_centrov = j
          for j in range(1, max_k + 1):
               seznam2 = []
               for p in range(stevilo_ponovitev):
                    G = mreza(i,i,a,b)
                    stevilo_komponent =
                        connected_components_number(G)
                    if stevilo_komponent <= j: #st. komponent
                         <= k (st.centrov) (j)
                         razdalja = round(najkrajsa_razdalja(G
                             , j))
                         seznam2.append(razdalja)
                    else:
                         seznam2.append(None)

               vsota = 0
               stevec = 0
               povprecje = 0
               for v in range(len(seznam2)):
                    if seznam2[v] != None:
                         vsota += seznam2[v]
                         stevec += 1
               if stevec == 0:
                    povprecje = None
               else:
                    povprecje = vsota/stevec

               seznam1.append(povprecje)
```

```
        seznam.append(seznam1)

    return seznam
```

5.4 Optimalni čas v odvisnosti od k in velikosti mreže

Za računanje časov prejšnji kodi pri izračunu R-a dodamo:

```
zacetni = time.time()
razdalja = round(najkrajsa_razdalja(G, j))
koncni = time.time() - zacetni
seznam2.append(koncni)
```

Vrne matriko: element a_{ij} pove povprecni čas izvajanja (za n ponovitev) pri mreži $i \times i$, ki ima j centrov.

Definirala sva tudi funkcije za izračun časov in R-jev, kjer pri fiksni velikosti spreminjamo samo k-je ali obratno. Te so uporabne za večje mreže, kjer so časi računanja precej daljši. Te funkcije ne vrnejo matrike, ampak seznam.
V nadaljevanju sva definirala podobne funkcije, ki nam vrnejo sezname časov izvajanja ali njihovih optimalnih vrednosti in njihova povprečja, v odvisnosti od velikosti mreže in od števila izbrisanih vozlišč in/ali izbrisanih povezav. Tu je primer za brisanje vozlišč, za brisanje povezav je funkcija podobna.

5.5 Optimalne razdalje R v odvisnosti od spreminjanja števila izbrisanih vozlišč

Funkcija `R_v_odvisnosti_od_spreminjanje_stevila_vozlisc` za poljubno velikost mreže m×n vrne povprečne optimalne vrednosti R pri različnem številu izbrisanih vozlišč. Podobna je funkcija za izračun časov izvajanja.
Število izbrisanih vozlišč gre od 0 do max_st_izbrisanih_vozlisc.
k predstavlja število centrov, b pa je število izbrisanih povezav, ki je konstantno.

V `R_v_odvisnosti_od_spreminjanje_stevila_vozlisc` uporabimo podfunkcijo `opt_vrednost_vozli`, ki izvede eno ponovitev in vrne seznam, katerega i-ti element pove R za i izbrisanih vozlišč mreže m×n.

```
def opt_vrednost_vozli(m,n,max_st_izbrisanih_vozlisc,b,k)
    :
    seznam_vrednosti = []
    for i in range(0,max_st_izbrisanih_vozlisc + 1):
        G = mreza(m,n,i,b)
        stevilo_komponent = connected_components_number(G
            )
        if stevilo_komponent <= k:
            razdalja = round(najkrajsa_razdalja(G, k))
            seznam_vrednosti.append((razdalja))
        else:
            seznam_vrednosti.append(None)
```

```
    return seznam_vrednosti
```

fukncija vrne opt. razdalje R glede na število odstranjenih vozlišč FIKSNO:velikost mreze, K, st. izbrisanih povezav SPREMINJAM: st_izbrisanih vozlišč

```
def R_v_odvisnosti_od_spreminjanje_stevila_vozlisc(m,n,
    max_st_izbrisanih_vozlisc,b,k,stevilo_ponovitev):
    seznam = []
    for i in range(stevilo_ponovitev):
        razdalje = opt_vrednost_vozli(m,n,
            max_st_izbrisanih_vozlisc,b,k)
        seznam.append(razdalje)

    povprecja = []
    for j in range(max_st_izbrisanih_vozlisc + 1):
        vsota = 0
        stevec = 0
        for i in range(stevilo_ponovitev):
            if seznam[i][j] != None:
                vsota += seznam[i][j]
                stevec += 1
        if stevec == 0:
            povprecja.append(None)
        else:
            povprecja.append(vsota/stevec)
    return seznam, povprecja
```

Koda za brisanje povezav je precej podobna tej, zato v poročilo ni vključena.

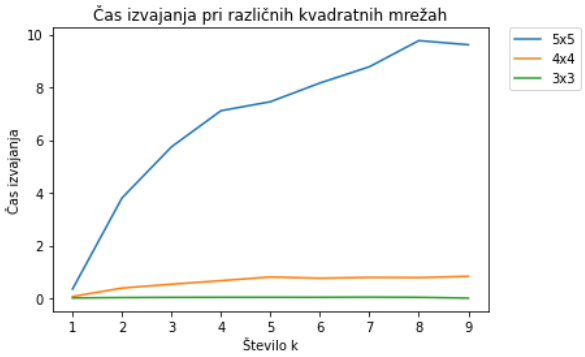
6 Poskusi

6.1 Čas izvajanja

Za na konec sva izvedla poskuse na že omenjenih funkcijah, torej kako se čas izvajanja in optimalna vrednost R spreminjata glede na večanje števila centrov(k), večanje mreže in glede na število izbrisanih vozlišč in/ali robov.

6.2 Čas izvajanja pri kvadratnih matrikah

Graf prikazuje kako se čas izvajanja spreminja glede na število centrov. Vidimo lahko, da večje kot je število centrov, daljši je čas izvajanja. Na začetku strmo narašča, za velike k-je pa se ratuje bolj in bolj konstantna. Očitna je tudi precejšnja razlika časov izvajanja pri večanju velikosti mreže. Pri mreži 3×3 so časi nekaj stotink sekunde, 5×5 že nekaj sekund, pri 6×6 pa lahko čas merimo že v minutah.

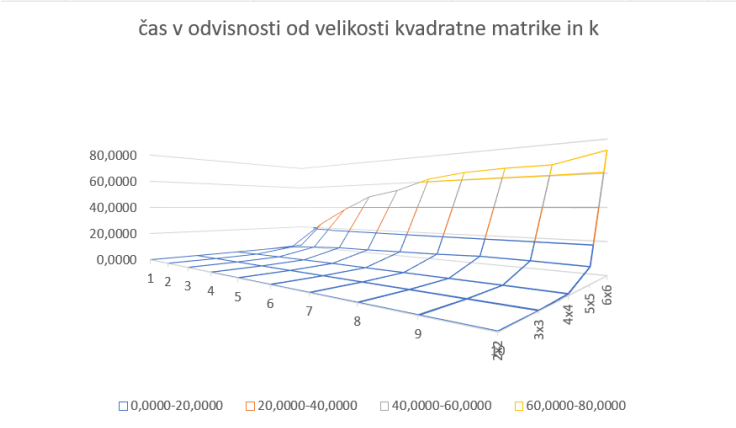


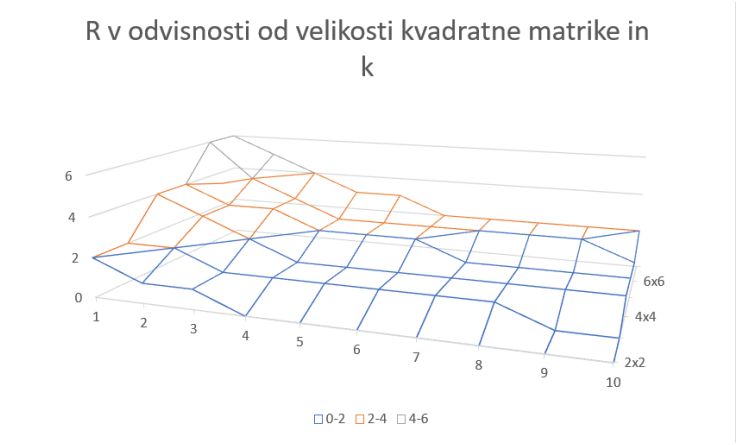
6.3 Čas in R v odvisnosti od velikost matrike in k

nxn\k	1	2	3	4	5	6	7	8	9	10
2x2	2	1	1	0						
3x3	2	2	1	1	1	1	1	1	0	
4x4	4	3	2	1	1	1	1	1	1	1
5x5	4	3	3	2	2	2	1	1	1	1
6x6	6	4	3	2	2	2	2	2	2	1
7x7	6	5	4	3	3	2	2	2	2	2

nxn\k	1	2	3	4	5	6	7	8	9	10
2x2	0,0019	0,0025	0,0029	0,0019						
3x3	0,0135	0,0295	0,0379	0,0404	0,0404	0,0397	0,0460	0,0395	0,0079	
4x4	0,0697	0,3894	0,5357	0,6695	0,8091	0,7631	0,7933	0,7858	0,8349	0,7739
5x5	0,3512	3,8047	5,7414	7,1104	7,4513	8,1603	8,7763	9,7675	9,6103	9,1586
6x6	1,3744	24,2125	37,6758	49,3276	54,0472	62,1392	65,5273	66,7933	67,0000	73,5362

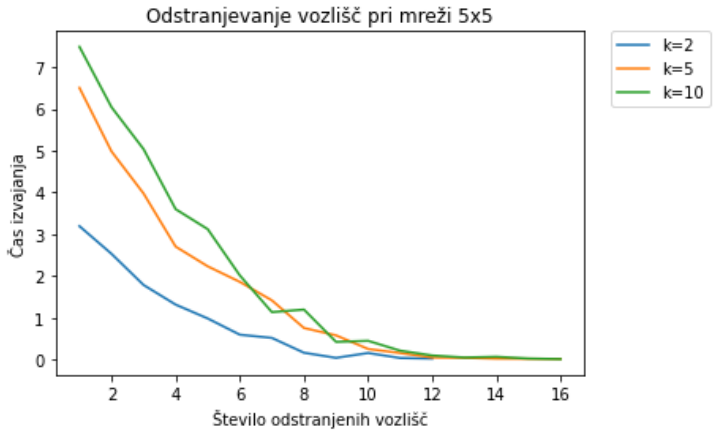
Spodnja grafa prikazujeta rezultate iz zgornje tabele, torej kako se čas izvajanja oz. optimalna vrednost spreminja, ko povečujemo velikost mreže in število centrov. Iz prvega grafa vidimo, da čas za velike mreže zelo hitro narašča, iz drugega pa da velikost nima tolkšnega vpliva na optimalno vrednost, kot pa število centrov.



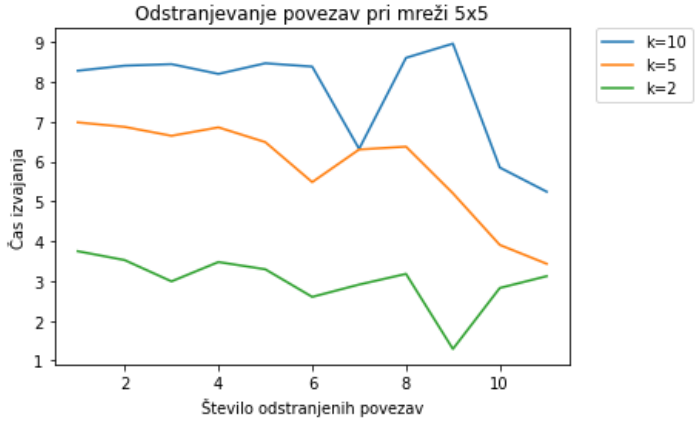


6.4 Čas izvajanja pri odstranjevanju vozlišč in povezav

Iz spodnjega grafa lahko vidimo, kako čas izvajanja pada, ko odstranjujemo vozlišča. Kar je seveda smiselno, saj manj vozlišč kot ima graf, hitreje se izvede CLP koda. Prav tako če pri brisanju vozlišč graf razpade na več delov, to še toliko bolj pripomore k manjšim časom izvajanja, saj CLP občutno hitreje izračuna več manjših mrež kot eno veliko. Poskus sva izvedla na 5×5 mreži, za tri različne k-je in sicer, 2,5 in 10.

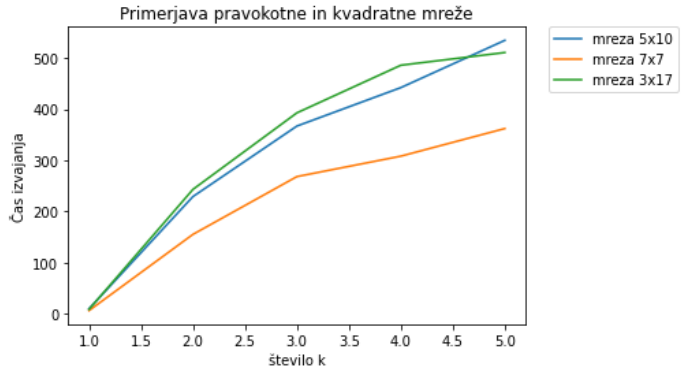


Graf prikazuje, kako se čas izvajanja spreminja glede na število odstranjenih povezav pri 5×5 mreži za k= {1,5,10}. Opaziti je padanje časov, vendar precej počasneje kot pri brisanju vozlišč. To zato, ker poleg tega da je vozlišč vedno manj, vsakič ko odstranimo vozlišče odstranimo tudi vse njegove povezave, kar lahko pomeni do 4 povezav. Pri brisanju povezav so spremembe manjše, hkrati pa smo jih tu izbrisali do 10, vseh povezav pa je 40. Do skokov na grafu pride tudi zato, ker so bilo za vsako izbrisano povezavo izvedeni le 3 poskusi in bi padanje verjetno bilo bolj enakomerno, če bi teh poskusov bilo več.



6.5 Čas in R pri pravokotnih in kvadratnih mrežah

Na grafu je razvidno, da za primerljivo število vozlišč CLP najhitreje izračuna kvadratne mreže, bolj kot pa je mreža podolgovata, več dela ima.



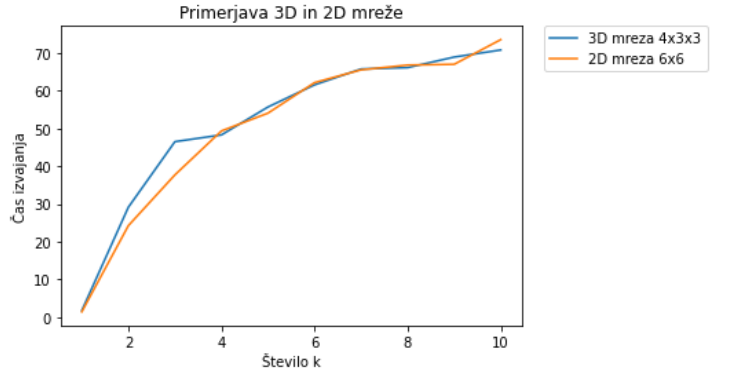
Za velike k-je vidimo da so optimalne razdalje enake, razlike pa se pojavijo pri manjših k. V splošnem ne moremo reči, da za vsak k velja, da bo za bolj podolgovate mreže optimalna razdalja večja. To pokaže tudi graf pri k=2, kjer optimalna razdalja pri kvadratni mreži ni najmanjša.Zgodi se lahko torej, da si nekateri k-ji pri podolgovatih mrežah lepše razdelijo prostor, kot pri kvadratnih mrežah.

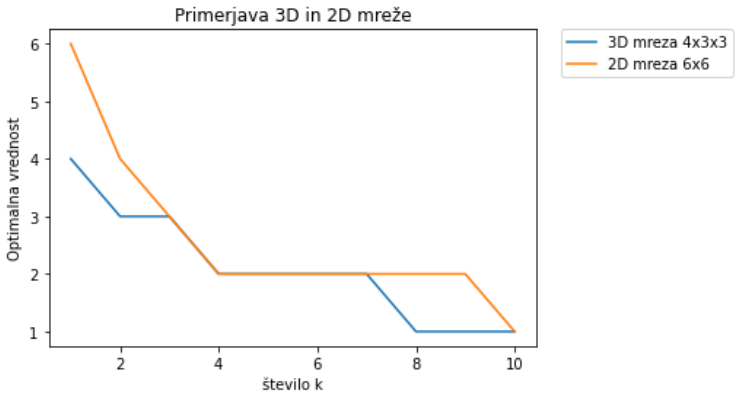
6.6 Optimalna vrednost pri drevesu in kvadratni mreži

Opazimo, da je optimalna vrednost pri drevesih, ki imajo isto število vozlišč kot mreža, večja ali enaka, ne glede na število centrov.

6.7 Primerjava tridimenzionalne in dvodimenzionalne mreže

Obe mreži imata 36 vozlišč. Na grafih vidimo, da so časi izvajanja pri obeh zelo podobni, razlikujeta pa se v optimalnih vrednostih. Trodimenzionalni grafi imajo pri istem k manjšo optimalno vrednost, kar je posledica boljše povezanosti vozlišč. Tu ima vozlišče lahko kar šest sosedov, v primerjavi z maksimalno štirimi sosedi pri dvodimenzionalnih mrežah.





7 Viri

- Rana, R., Garg, D. (2011). An Evaluation of K-Center Problem Solving Techniques,towards Optimality. Dostopno na: https://www.longdom.org/open-access/an-evaluation-of-kcenter-problem-solving-techniques-towards-optimality-0976-4860-2-206-214.pdf?fbclid=IwAR2KDCBmif7PnRmwStiAYKnHX4Gw6UFae0h4sAo-UPN7rUaT7uRH1Ys8unc