UNIVERZA V LJUBLJANI
FAKULTETA ZA MATEMATIKO IN FIZIKO

Min-Graph Equipartition Problem with Simulated Annealing

Poročilo

Ana Marija Kravanja, Urška Jeranko, Oskar Kregar

Ljubljana, 2019

Kazalo

1. OPIS PROBLEMA

V naši projektni nalogi smo reševali problem deljenja grafa z uporabo dveh predpisanih metod. Poljuben graf smo morali razdeliti na dva skoraj enaka dela tako, da je bilo med tema nastalima deloma čim manj povezav. Projekta smo se lotili tako, da smo napisali algoritma požrešne metode (Greedy Method) in metode simuliranega ohlajanja (Simulated Annealing Heuristic), napisali pa smo tudi funkcijo, ki že razdeljen graf nariše in z barvami prikaže optimalno delitev vozlišč. Algoritma smo preizkusili na več različnih grafih in analizirali, kako se metodi obneseta na gostih, redkih ter poljubnih grafih. Pri metodi simuliranega ohlajanja pa smo spreminjali tudi začetno temperaturo in nato primerjali dobljene rešitve med seboj. Da bi bilo naše preizkušanje algoritmov preprostejše smo še napisali funkcijo, ki vrača naključne matrike sosednosti, poljubnih velikosti.

Grafe, ki smo jih preučevali, smo v oba algoritma vstavljali v obliki matrike sosednosti, zato smo na začetku definirali: Naj bo $G = (V, E)$ enostaven graf, pri čemer je V množica vozlišč in E množica povezav. Naj bo število vozlišč enako n. Definirali smo delitev grafa na dve množici X in Y, pri čemer je $|X| = \lfloor \frac{n}{2} \rfloor$. Iskali smo minimalno širino bisekcije, to je najmanjše število povezav med X in Y med vsemi možnimi delitvami.

Definicija 1.1. Naj bo $G = (V, E)$ enostaven graf in $X, Y \subseteq V$, tako da je $X \cap Y = \emptyset$ in $X \cup Y = V$.

- Za $x \in X$ označimo z $I(x)$ notranjo vrednost, to je število povezav $(x, z) \in E; z \in X \backslash \{x\}$. Analogno definiramo $I(y)$ za $y \in Y$.
- Za $x \in X$ označimo z $O(x)$ zunanjo vrednost, to je število povezav $(x, z) \in E; z \in Y$. Analogno definiramo $O(y)$ za $y \in Y$.
- Za $x \in X, y \in Y$ naj bo $\omega(x, y) := \begin{cases} 1, & \text{če je } (x, y) \in E \\ 0, & \text{sicer} \end{cases}$.
- Za $x \in X, y \in Y$ naj bo $S(x, y) := O(x) - I(x) + O(y) - I(y) - 2\omega(x, y)$.

2. POŽREŠNA METODA

Požrešna metoda je strategija, ki na vsakem posameznem koraku izbere optimalno rešitev s ciljem, da nas to privede do globalno optimalne rešitve. To pomeni, da algoritem izbere rešitev, ki je trenutno najboljša, vendar se pri tem ne ozira na posledice - ne gleda celotne slike. Težava te metode je, da na vsakem koraku izbere le lokalno najboljšo rešitev, samo upamo pa lahko, da je to tudi prava pot do globalnega optimuma. Pogosto torej sploh ne najde najboljše rešitve, povsem mogoče je celo, da nas pripelje do najslabše možne.

2.1. **Algoritem požrešne metode.** Začnemo z naključno delitvijo vozlišč grafa na dve skoraj enako veliki množici X in Y. Množicama X in Y recimo strani. Algoritem zamenja vozlišči na različnih straneh (pri čemer se vse povezave ohranijo), če s tem dobimo boljšo bisekcijo (manjše število povezav med stranema) in se ustavi, ko to ni več možno. Pri menjavi notranje vrednosti postanejo zunanje in obratno, zato dobimo izboljšano bisekcijo le, če je zgoraj definiran $S(x, y) > 0$.

Vhodni podatki: Graf $G = (V, E), |V| = n$.

(1) Izberemo neključno delitev (X, Y).
(2) Izberemo $x \in X, y \in Y$, tako da je $S(x, y) > 0$.
(3) Zamenjamo vozlišči x in y.
(4) Ponavljamo 2. in 3. korak, dokler ne obstajata več $x \in X, y \in Y$, da je $S(x, y) > 0$.

Izhodni podatki: delitev (X, Y).

Algoritma smo se lotili tako, da smo najprej definirali število vozlišč n kot dolžino vhodnega enostavnega grafa G, ki ga v funkcijo vstavimo kot seznam seznamov, da dobimo matriko sosednosti. Nato smo morebitne enke na diagonali matrike G spremenili v ničle in s tem izbrisali povezave vozlišč s samimi seboj, ki na rezultat tako nimajo vpliva, s tem pa smo preprečili morebitne komplikacije. V naslednjem koraku smo izbrali delitvi X in Y tako, da je v množici X prva polovica vozlišč, v množici Y pa druga polovica, kar je tudi ena od možnih naključnih delitev. Ker je malo verjetno, da je to optimalna delitev, smo napisali pomožno funkcijo, ki preveri vse možne delitve na način, ki smo ga opisali zgoraj. Uporabili smo for zanki, ki tečeta po vseh elementih množice X in množice Y, da preverimo vse možne delitve. Notranjo in zunanjo vrednost teh dveh množic smo nastavili na 0. Nato smo uporabili še for zanki, ki štejeta povezave znotraj množic X in Y ter med njima in tako računata notranje in zunanje vrednosti. Na ta način dobimo vrednost $S(x, y)$, ki nam pove, ali vozlišči $x \in X$ in $y \in Y$ zamenjamo. Zanka se izvaja, dokler ne obstaja več pozitiven $S(x, y)$, takrat algoritem vrne množici vozlišč X in Y, ki predstavljata lokalno optimalno delitev vozlišč. Metodo lahko preizkusimo na poljubni začetni delitvi grafa na dve skoraj enako veliki množici.

2.2. **Psevdokoda požrešne metode.** Slika na naslednji strani prikazuje psevdokodo požrešne metode. Najprej je napisana pomožna funkcija, ki nam šteje povezave znotraj množic X in Y ter med njima, da nam pove, ali je $S(x, y)$ za izbrani vozlišči pozitiven, torej ali se vozlišči splača zamenjati. Druga funkcija pa je dejanska požrešna metoda, ki opravi začetno delitev, ponavlja while zanko, dokler ne najde več vozlišč za zamenjavo ter vrne novo delitev grafa.

```python
def pomozna(G,X,Y):
    n = len(G)
    for i in X:
        for j in Y:
            Ix = 0
            Ox = 0
            Iy = 0
            Oy = 0
            for k in X:
                if G[i][k] == 1:
                    Ix += 1
                if G[j][k] == 1:
                    Oy += 1
            for l in Y:
                if G[j][l] == 1:
                    Iy += 1
                if G[i][l] == 1:
                    Ox += 1
            S = Ox - Ix + Oy - Iy - 2 * G[i][j]
            if S > 0:
                x = X.index(i)
                y = Y.index(j)
                X[x] = j
                Y[y] = i
                return X, Y, False
    return X,Y,True

def pozresna_metoda(G): #G je matrika sosednosti
    n = len(G)
    for a in range(0,n):
        if G[a][a] ==1:
            G[a][a]=0
    d1=int(n/2)
    koncaj=False
    X=list(range(0,d1))
    Y=list(range(d1,n))
    while koncaj==False:
        [X,Y,koncaj]= pomozna(G,X,Y)
    return X,Y
```

Slika 1. Psevdokoda požrešne metode

3. METODA SIMULIRANEGA OHLAJANJA

Algoritem se je v osnovi razvil zaradi procesa toplotne obdelave kovin. Različne materiale segrevamo oz. ohlajamo, da bi spremenili fizikalne lastnosti njihove notranje strukture. Ko se kovina enkrat ohladi, postane njena nova struktura fiksna in kovina tako obdrži pridobljene lastnosti. Pri simuliranem ohlajanju je temperatura naša spremenljivka, ki ponazarja toplotni proces. Na začetku temperaturo nastavimo na visoko vrednost in jo nato sčasoma nižamo, ko se algoritem izvaja. Na začetku, ko je temperatura nastavljena na visoko vrednost, algoritem pogosteje sprejema rešitve, ki so slabše od naše trenutne rešitve zato, da se izogne lokalnim optimumom, ki nas ne bodo pripeljali do globalnega optimuma. Z nižanjem temperature pa se niža tudi verjetnost, da bo algoritem izbiral slabše rešitve; postopoma se torej osredotoči na iskalno območje, na katerem upamo, da je mogoče najti rešitev blizu optimalne. Algoritem je zelo učinkovit pri iskanju rešitev za probleme velikih velikosti, ki vsebujejo številne lokalne optimume.

3.1. **Algoritem metode simuliranega ohlajanja.**

Graf definiramo analogno kot pri požrešni metodi. Vozlišči pa zdaj zamenjamo z verjetnostjo

$$P(x, y, t) = e^{\frac{Q(R)-Q(S)}{t}}; t \geq 0.$$

S $Q(S)$ je označena funkcija, ki množici S priredi neko kvaliteto, torej če je $Q(S) > Q(R)$, pomeni, da je množica S bolj ugodna za nadaljnjo obravnavo. Za funkcijo Q smo si v našem primeru izbrali število povezav med množicama. Torej če je $Q(R)$ funkcija, kjer smo v množici R zamenjali vozlišča, in $Q(S)$ množica S, kjer ju nismo zamenjali, in velja, da je $Q(R) < Q(S)$, potem z določeno verjetnostjo zamenjamo R z S. V množici R je v tem primeru med podmnožicama namreč manj povezav kot v S.

(1) t = temperatura, ki jo nastavimo na visoko vrednost
(2) S = začetna delitev grafa na množici X in Y
(3) N = S (t.j. trenutna najboljša rešitev)
(4) Ponavljaj, dokler je N najboljša rešitev, nam je zmanjkalo časa ali pa je $t \leq 0$:
(5) R = delitev, kjer zamenjamo $x \in X$ in $y \in Y$
(6) če je $Q(R) < Q(S)$ ali naključno število med 0 in 1 manjše od $P(x, y, t)$, potem je $S = R$
(7) zmanjšamo t
(8) če je $Q(S) > Q(N)$:
(9) $N = S$
(10) vrni N

Algoritem simuliranega ohlajanja smo napisali kot funkcijo, ki sprejme kot vhodna podatka matriko sosednosti G (seznam seznamov) in temperaturo t, ki jo običajno nastavimo na visoko vrednost (korak 1). Tudi tokrat smo začeli tako, da smo definirali število vozlišč n kot dolžino vhodnega grafa G in morebitne enke na diagonali matrike sosednosti G spremenili v ničle. Ponovno smo definirali začetno delitev tako,

da množica X vsebuje prvo polovico vozlišč, množica Y pa drugo polovico (korak 2). Nato smo napisali pomožno funkcijo $seznam\_stevila\_sosedov(G, X, Y)$, ki nam prešteje povezave znotraj množic X in Y ter med njima in poleg števila povezav vrne še začetno število sosedov za vsako vozlišče. Na ta način smo dobili notranje in zunanje vrednosti množic X in Y. V naslednjem koraku smo privzeli, da je trenutna rešitev, to je naša začetna delitev, kar optimalna oz. najboljša rešitev (korak 3). Potem smo napisali while zanko, ki se izvaja, dokler nismo prekoračili določeno število korakov (časovna omejitev), je temperatura manjša od 0 ali število povezav med množicama vozlišč enako 0 (korak 4). Potem za trenutni seznam števila sosedov in trenutno število povezav (te se bodo spreminjale, ko se zanka izvaja) nastavimo naše najboljše rešitve, ki so zaenkrat kar rešitve začetne delitve. Sledi računanje trenutnega števila povezav za naključno izbrana dva vozlišča iz različnih množic, za kar pa potrebujemo trenutni seznam števila sosedov. Seveda moramo upoštevati tudi, ali sta ta dva vozlišča povezana med seboj. Nato moramo preveriti, ali se vozlišča splača zamenjati. Pri tej metodi to storimo tako, da preverimo, ali je trenutno število povezav manjše od najboljšega števila povezav. Če to ne velja, lahko vozlišči vseeno zamenjamo, če je naključno število med 0 in 1 manjše od $P(x, y, t)$ (korak 6). Če je vsaj eden od zgornjih pogojev izpolnjen, se vozlišči zamenjata, torej se spremenijo tudi sosedi, notranje vrednosti pa postanejo zunanje in obratno. Na koncu še prištejemo korak, da ne prekoračimo časovne omejitve, in znižamo temperaturo, kot narekuje metoda simuliranega ohlajanja. Algoritem vrne optimalni množici vozlišč in število povezav med njima.

3.2. **Psevdokoda simuliranega ohlajanja.** Na prvi sliki na naslednji strani je prikazana pomožna funkcija,ki vrne seznam seznamov in število povezav med začetnima množicama delitve X in Y. Prva številka v notranjem seznamu za vsako vozlišče pove, koliko sosedov ima v svoji množici in koliko jih ima v drugi množici. Ta števila bomo potrebovali pri računanju trenutnega števila povezav pri simuliranem ohlajanju, kar pa je bistvenega pomena. Na naslednji sliki je funkcija simuliranega ohlajanja, ki je odvisna tudi od parametra t, torej temperature. Na enak način kot pri požrešni metodi funkcija opravi začetno delitev na množici X in Y. Računanje števila povezav se izvaja v while zanki, ki teče, dokler je $t > 0$ oz. ne presežemo določenega števila korakov. V zanki vozlišča izbiramo naključno in na poseben način preverjamo, ali se ju splača zamenjati. Zadnja funkcija pa je funkcija, ki generira simetrične matrike, katerih elementa sta lahko samo 0 in 1. Te matrike predstavljajo naše naključne matrike sosednosti oz. enostavne grafe.

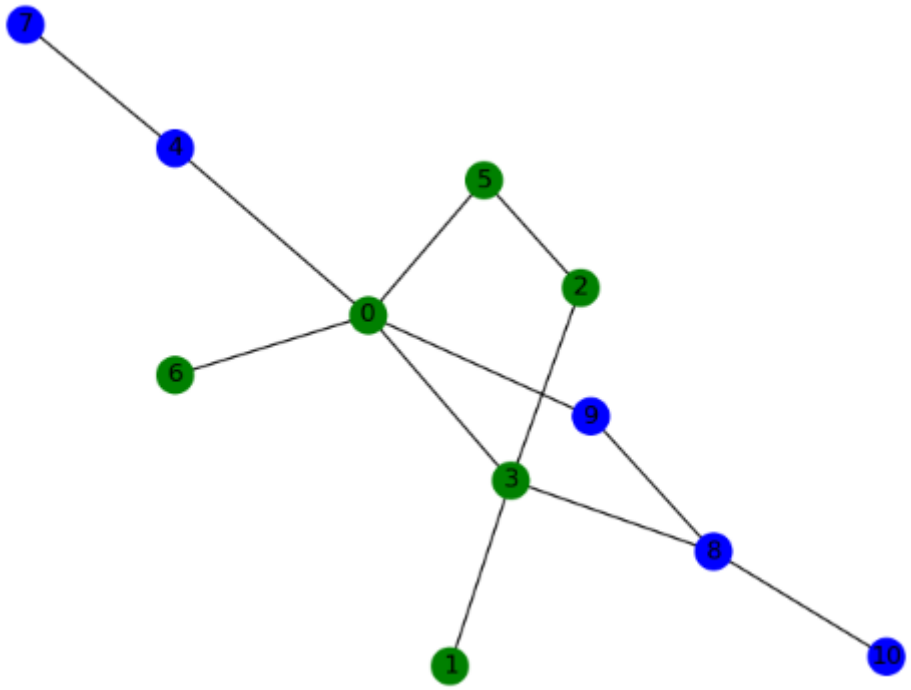
```
61  def seznam_stevila_sosedov(G,C,D):    #izracuna zacetno stevilo sosedov za v
62      n = len(G)
63      #prva stevilka pove stevilo, s kolikimi je povezan v svoji mnozici, drug
64      seznam_stevila_sosedov = [[0, 0]] * n
65      stevilo_povezav = 0
66      for nahajanje, i in enumerate(C, 0):
67          Ix = 0
68          Ox = 0
69          for m in C:
70              if G[i][m] == 1:
71                  Ix += 1
72          for j in D:
73              if G[i][j] == 1:
74                  Ox += 1
75          seznam_stevila_sosedov[nahajanje] = [Ix, Ox]
76          stevilo_povezav += Ox
77      for nahajanje2, j in enumerate(D,0):
78          Iy = 0
79          Oy = 0
80          for l in C:
81              if G[j][l] == 1:
82                  Oy += 1
83          for o in D:
84              if G[j][o] == 1:
85                  Iy += 1
86          seznam_stevila_sosedov[nahajanje2 + len(C)] = [Iy, Oy]
87      return seznam_stevila_sosedov, stevilo_povezav
```

Slika 2. Psevdokoda funkcije za računanje števila povezav

```
def simulirano_ohlajanje(G,t): #temperaturo običajno nastavimo na visoko vrednost
    n = len(G)
    for a in range(0, n): #spremenimo diagonalo, da ima same 0
        if G[a][a] == 1:
            G[a][a] = 0
    d1 = int(n / 2)
    X = list(range(0, d1)) #vektorja indeksov vozlišč
    Y = list(range(d1, n))
    A = list(range(0, d1)) #vektorja indeksov vozlišč
    B = list(range(d1, n))
    k = 1
    (najboljsi_seznam_stevila_sosedov, najboljse_stevilo_povezav) = seznam_stevila_sosedov(G,X,Y) #tisti, ki je pri p
    while (t>0) and (k<10000):
        (trenutni_seznam_stevila_sosedov, trenutno_stevilo_povezav) = (najboljsi_seznam_stevila_sosedov, najboljse_st
        mesto = random.randrange(len(A))
        a = A[mesto]
        mesto2 = random.randrange(len(B))
        b = B[mesto2]
        #prvi indeks ti vedno pove s kolikimi je povezan v svoji množici, drugi  pa s kolikimi v drugi množici
        trenutno_stevilo_povezav += trenutni_seznam_stevila_sosedov[a][0] + trenutni_seznam_stevila_sosedov[b][0]
        trenutno_stevilo_povezav -= trenutni_seznam_stevila_sosedov[a][1] + trenutni_seznam_stevila_sosedov[b][1]
        if G[a][b] == 1:
            trenutno_stevilo_povezav += 2
        print(trenutno_stevilo_povezav,trenutni_seznam_stevila_sosedov,a,b)
        Q = najboljse_stevilo_povezav-trenutno_stevilo_povezav #naša kvaliteta
        if (trenutno_stevilo_povezav < najboljse_stevilo_povezav) or (random.uniform(0, 1) < math.exp(Q/t)):
            najboljse_stevilo_povezav = trenutno_stevilo_povezav
            #pogledamo, kako se spremenijo sosedi
            for i in A:
                if G[i][a] == 1:
                    trenutni_seznam_stevila_sosedov[i][0] -= 1
                    trenutni_seznam_stevila_sosedov[i][1] += 1
                if G[i][b] ==1:
                    trenutni_seznam_stevila_sosedov[i][0] += 1
                    trenutni_seznam_stevila_sosedov[i][1] -= 1
            for j in B:
                if G[j][b] == 1:
                    trenutni_seznam_stevila_sosedov[j][0] -= 1
                    trenutni_seznam_stevila_sosedov[j][1] += 1
                if G[j][a] ==1:
                    trenutni_seznam_stevila_sosedov[j][0] += 1
                    trenutni_seznam_stevila_sosedov[j][1] -= 1
            #pri vozliščih, ki sta se zamnjali, notranje povezave postanejo zunanje in obratno
            obrat = trenutni_seznam_stevila_sosedov[a][0]
            trenutni_seznam_stevila_sosedov[a][0] = trenutni_seznam_stevila_sosedov[a][1]
            trenutni_seznam_stevila_sosedov[a][1] = obrat
            obrat2 = trenutni_seznam_stevila_sosedov[b][0]
            trenutni_seznam_stevila_sosedov[b][0] = trenutni_seznam_stevila_sosedov[b][1]
            trenutni_seznam_stevila_sosedov[b][1] = obrat2
            najboljsi_seznam_stevila_sosedov = trenutni_seznam_stevila_sosedov
            vmesni= A[mesto] #zamenjamo ju še v vektorju A in B
            A[mesto] = B[mesto2]
            B[mesto2] = vmesni
        k += 1
        t -= 0.1
    return A, B, trenutno_stevilo_povezav
```

```
112  def nakljucna_matrika(velikost): #funkcija ki vrne simetrične matrike
113      A = np.random.randint(2, size=(velikost,velikost))
114      m= np.tril(A) + np.tril(A,-1).T
115      return np.array(m).tolist()
```

Slika 3. Psevdokoda simuliranega ohlajanja in naključnih matrik



Slika 4. Primer grafa z optimalno delitvijo

```
Požrešna metoda vrne sledečo delitev:
([7, 1, 2, 3, 4], [0, 6, 5, 8, 9, 10]), število povezav:4
Simulirano ohlajanje:
1.) t=120 , t-= 0.1     delitev: [5, 3, 1, 0, 2], [4, 6, 10, 8, 7, 9], število povezav: 5
                        delitev: [1, 5, 2, 7, 4], [6, 9, 8, 10, 0, 3], število povezav: 4
                        delitev: [3, 8, 9, 1, 10], [5, 4, 7, 0, 6, 2], število povezav: 5
                        ...
2.) t=500, t-= 0.1      delitev: [7, 0, 6, 4, 5], [8, 2, 3, 9, 1, 10], število povezav: 4
                        ...
3.) t=120, t-=1         ko 4. poženemo funkcijo dobimo optimalno delitev:
                        [1, 10, 6, 8, 9], [7, 2, 5, 4, 3, 0], število povezav: 3
```

Slika 5. Uporaba metod na grafu iz slike 4

4. SKLEP IN ZAKLJUČEK

Na koncu smo napisana algoritma preizkusili na konkretnih podatkih in preverjali nekaj predpostavk, ki smo jih zasledili med viri. Požrešna metoda bi naj bila manj zanesljiva od metode simuliranega ohlajanja, saj na vsakem koraku izbere optimalno rešitev, pri čemer se ne ozira na posledice. Ta metoda nekako predpostavlja, da je pot po lokalnih optimumih tudi pot do globalnega optimuma, kar pa ni vedno res, zaradi česar pogosto ne vrne optimalnega rezultata. Na drugi strani pa metoda simuliranega ohlajanja na začetku lahko pogosteje vrača v tistem trenutku slabše rezultate, vendar to počne ravno zaradi tega, da se izogne lokalnim optimumom, ki nas ne bi pripeljali do globalnega. Ko se temperatura niža, pa se niža tudi verjetnost izbiranja slabših rešitev, kar nas potem skoraj zagotovo pripelje vsaj v okolico globalnega optimuma, če že ne do njega.

Ko smo pognali obe metodi na istih grafih, smo ugotovili, da zgornja predpostavka v večini primerov drži. Razlika je še posebej opazna pri matrikah sosednosti velikih velikosti. Zanimivo je, da tudi na grafih z malo vozlišči, nobeni optimalni množici vozlišč nista popolnoma enaki. Seveda velja, da je možnih pravilnih rešitev lahko več. Različna delitev grafa ima namreč lahko enako število povezav med množicama, a kljub temu v veliko primerih ne prinese optimalnega rezultata.

Pri metodi simuliranega ohlajanja je zelo pomemben parameter tudi temperatura. Na rezultat vpliva tako njena začetna vrednost kot tudi vrednost zmanjševanja na vsakem koraku. Za optimalen rezultat, bi naj bila dobro izbrana začetna vrednost vsaj 500, o tem kako hitro bi jo naj zmanjševali, pa nismo našli podatka. Ta dva vpliva smo analizirali tako, da smo enega fiksirali, drugega pa spreminjali. Za začetne vrednosti smo si izbrali 50 (absolutno prenizko), 500 in 5000. Zmanjševali pa smo jo za 0.1, 1 in 5. Pričakovali smo, da bomo z višanjem začetne vrednosti in nižanjem vrednosti zmanjševanja dobivali vedno boljše rezultate. Najboljše rezultate bi tako naj dobili pri $t = 5000$ in $t- = 0.1$, najslabše pa pri $t = 50$ in $t- = 5$.

Ko smo spreminjali začetno vrednost in fiksirali zniževanje t, smo dobili najslabše rezultate za $t = 50$, kot smo pričakovali. Potem smo t nastavili na 500 in opazili, da se je večina rezultat izboljšala, nekateri so ostali enaki, nekaj pa se jih je tudi poslabšalo. Ko je bil $t = 5000$, se je večina rezultatov poslabšala, kar nas je presenetilo. Sklepamo lahko, da je res najboljše za začetno vrednost izbrati temperaturo okoli 500. Zanimivo je, da smo pri matrikah velikosti 20x20 dobili slabše rezultate, ko smo t povečali na 500, nato pa nekoliko boljše, ko smo t nastavili na 5000. Izpolnila pa se je naša predpostavka, da funkcija vrne boljši rezultat, če t na vsakem koraku le malo zmanjšamo. Tako smo v povprečju boljši rezultat dobili za $t- = 0.1$ kot za $t- = 1$ ali $t- = 5$. Prihajalo je tudi do odstopanj, kar pa je normalno, saj v funkciji operiramo tudi z verjetnostjo in naključji.

Na koncu smo še opazovali, kako se spreminjajo rešitve, če spreminjamo število korakov, torej k. Najprej smo k omejili na 100, nato pa še na 1000 in 10000. Po pričakovanjih smo boljše rezultate dobili, ko se je funkcija izvedla večkrat, v našem primeru torej 10000-krat, in najslabše, ko se je izvedla le 100-krat.
Če povzamem, smo v naši projektni nalogi ugotovili, da je metoda simuliranega

ohlajanja boljša izbira za iskanje optimalne delitve vozlišč. Požrešna metoda namreč vrača le lokalne optimume, medtem ko nas zanima globalni optimum. Poleg tega je zelo časovna zahtevna in tudi zaradi tega manj primerna. Čeprav tudi pri metodi simuliranega ohlajanja nimamo zagotovila, da je dobljena rešitev globalni optimum, metoda deluje tako, da nas pripelje vsaj v okolico iskanega optimuma, kar je velik napredek od požrešne metode. Poleg tega lahko spreminjamo parametra temperature in števila korakov ter ju nastavimo tako, da nas metoda pripelje čim bližje najboljši rešitvi. Za večjo natančnost in boljše utemeljitve je potrebno postopek čimvečkrat ponoviti pri različnih temperaturah, spreminjati je potrebno število korakov ter velikost matrik.

Literatura

[1] L. A. Wolsey, Integer Programming, Wiley Interscience,1998.

[2] C. Blum, A. Roli, Metaheuristics in Combinatorial Optimization: Overview and Conceptual Compariosn. [online]

[3] S. Luke, Essentials of Metaheuristics: a set of undergraduate lscture notes, online.

[4] S. Clinton, Generic algorithms with Python.

[5] K.L. Du, M.N.S. Swamy, Search and optimization by metaheuristics: tehniques and algorithms inspired by nature.

[6] El. G. Talbi, Metaheuristics: from design to implementation.