UNIVERZA V LJUBLJANI
FAKULTETA ZA MATEMATIKO IN FIZIKO

Finančna matematika

Blaž Povh, Luka Houška

Dominant Edge Metric Dimension

Skupinski projekt

Poročilo

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

Ljubljana, januar 2024

1. Navodilo naloge

Implement an ILP for computing dominant edge metric dimension and answer the following questions.

(1) Find trees on n vertices which have minimum/maximum dominant edge metric dimension.
(2) Find trees T on n vertices for which $Dedim(T) - edim(T)$ is maximum/minimum possible.
(3) Find trees T on n vertices for which $Dedim(T)/edim(T)$ is maximum/minimum possible.
(4) Determine $Dedim(G)$ of a grid graph $Pk \square Pt$.

For small graphs, apply a systematic search; for larger ones, apply some stochastic search. Report your results.

2. Načrt dela

Najina projektna naloga se nanaša dominantne metrične dimenzije grafov. Projekt sva razdelila na več manjših delov. Slednje sva si potem enakomerno razdelila. Natančnejši pregled najinega dela je opisan v nadaljevanju, tukaj pa je le kratek opis postopkov najinega dela. Najprej sva napisala ustrezen CLP, ki izračuna dominantno metrično dimenzijo povezav za dani graf. V nadaljevanju sva za vsako podnalogo ustvarila nove funkcije, ki so za izbran graf izračunale v navodilih podane karakteristike in generirala različne grafe, ter na njih testirala prej napisano kodo. V nadaljevanju sva napisala kodo, ki sva jo testirala tudi na večjih grafih. Temeljila pa je na metahevrističnem algoritmu imenovanem *local search*. Najin cilj oz. motivacija je bila ugotoviti, ali za grafe, ki ustrezajo določenim pogojem, veljajo kakšne posebne lastnosti. Za majhne grafe, natančneje drevesa, sva se problema lotila s testiranjem na vseh različnih drevesih. Za večje grafe pa sva kodo algoritem testirala le na naključnih grafih, saj je ta algoritem časovno zelo potraten. Za reševanje najine naloge sva uporabila okolje Sage (SageMath) znotraj spletne platforme CoCalc. V nadaljevanju bova razložila, kako sva se lotila vsake podnaloge, ter dodala tudi kodo s komentarji, ki se nahaja na najinem GitHub repozitoriju.

3. Definicije

Za lažje razumevanje najinega problema, si najprej poglejmo par pojmov ter definicij, ki sva jih uporabljala v sklopu najine projekne naloge.

Definicija 3.1. *V grafu G, množici vozlišč S pravimo, da razrešuje povezave, če za vsak par povezav $e, f \in E(G)$ obstaja $s \in S$, tako da $d(s, e) \neq d(s, f)$.*

Definicija 3.2. *Metrična dimenzija povezav na povezanem grafu G, označili jo bomo z $edim(G)$, je moč najmanjše množice vozlišč $S \subseteq V(G)$, ki razlikuje vse pare povezav, t.j. za vsak par povezav $e, f \in E(G)$, obstaja vozlišče $s \in S$, tako da $d(s, e) \neq d(s, f)$. Tu upoštevamo, da je za povezavo $e = uv$, $d(s, e) = min\{d(s, u), d(s, v)\}$.*

Definicija 3.3. *Množica vozlišč C je* vozliščno pokritje, *če ima vsaka povezava iz grafa G vsaj eno krajišče v C.*

Definicija 3.4. *Dominantna množica, ki razrešuje povezave S za graf G, je množica S, ki je hkrati vozliščno pokritje in razrešuje povezave. Dominantna metrična dimenzija povezav grafa G je moč najmanjše množice S, ki razrešuje povezave. Označimo jo z $Dedim(G)$.*

4. Majhni grafi

Najin prvi korak je bil generiranje funkcije, ki s pomočjo ustreznega ter učinkovitega CLP izračuna dominantno metrično dimenzijo povezav za dani graf, čemur krajše rečemo kar Dedim. Prvo sve definirala funkcijo *razdalje_povezave_vozlisca(g)*, ki kot argument sprejme graf g in vrne matriko razdalj med povezami in vozlišči, torej element v vrstici i in stolpcu j predstavlja razdaljo med povezavo e_i in vozliščem v_j.

```
1     def razdalje_povezave_vozlisca(g):
2         razdalje = distances_all_pairs(g)
3         razdalje_povezave = {}
4
5         for i,j in g.edges(labels=False):
6             razdalje_povezave[i,j] = {}
7             for v in g:
8                 razdalje_povezave[i,j][v] = min(razdalje[i][v
                      ], razdalje[j][v])
9
10        return razdalje_povezave
```

To funkcijo sva potem uporabila v funkciji *CLP_Dedim(g)*, ki kot argument sprejme graf g in vrne vrednost $Dedim(g)$, ter vozlišča, ki sestavljajo rešitev.

```
1     import itertools
2     def CLP_Dedim(g):
3
4         razdalje = razdalje_povezave_vozlisca(g)
5
6         p = MixedIntegerLinearProgram(maximization = False)
7         x = p.new_variable(binary = True)
8         p.set_objective( sum([x[v] for v in g]) )
9
10        for u,v in g.edges(labels = False):
11            p.add_constraint(x[u] + x[v] >= 1)
12
13
14        edges = list(g.edges(labels=False))
15        for (i,j),(u,v) in itertools.combinations(edges, 2):
16            vsota = sum(abs(razdalje[(i,j)][k] - razdalje[(u,v
                  )][k])*x[k] for k in g.vertices())
17            p.add_constraint(vsota >= 1)
18
19
```

```
20          optimalna_resitev = p.solve()
21          vrednosti_za_S = p.get_values(x)
22
23          return optimalna_resitev, vrednosti_za_S
```

4.1. Prvi del naloge

Prvi del naloge je od naju zahteval, da najdeva drevesa T na n vozliščih, ki imajo minimalen/maksimalen $Dedim(T)$. V ta namen sva definirala funkciji *drevesa_ min(i,j)* in *drevesa_ max(i,j)*, ki kot argumenta sprejmeta števili i in j in za vsako število vozlišč med i in j narišeta drevo T, ki ima najmanjšo oz. največjo $Dedim$ in pod njim izpišeta vrednost $Dedim$. Pomagala sva si z vgrajeno funkcijo *TreeIterator*, ki je zelo uporabna za generiranje vseh dreves na n vozliščih.

4.2. Drugi del naloge

Drugi del naloge je od naju zahteval, da najdeva drevesa T na n vozliščih, ki imajo minimalno/maksimalno možno razliko $Dedim(T) - edim(T)$. Tu je $edim(T)$ metrična dimenzija grafa na povezavah, torej opustimo pogoj, da more biti iskana množica vozliščno pokritje. Najina prva naloga je bila, da definirava funkcijo *CLP_ edim(g)*, ki nam vrne metrično dimenzijo grafa. Nato sva definirala funkciji *drevesa_ min_ razlika(i,j)* in *drevesa_ max_ razlika(i,j)*, ki kot argumenta sprejmeta števili i in j in za vsako število vozlišč med i in j narišeta drevo T, ki ima najmanjšo oz. največjo razliko $Dedim(T) - edim(T)$ in pod njim izpišeta vrednost. Tudi tu sva si pomagala z vgrajeno funkcijo *TreeIterator*.

4.3. Tretji del naloge

Tretji del naloge je od naju zahteval, da najdeva drevesa T na n vozliščih, ki imajo minimalen/maksimalen možen količnik $Dedim(T)/edim(T)$. V ta namen sva definirala funkciji *drevesa_ min_ ulomek(i,j)* in *drevesa_ max_ ulomek(i,j)*, ki kot argumenta sprejmeta števili i in j, $i, j \geq 3$ in za vsako število vozlišč med i in j narišeta drevo T, ki ima najmanjši oz. največji količnik $Dedim(T)/edim(T)$ in pod njim izpišeta vrednost. Tudi tu sva si pomagala z vgrajeno funkcijo *TreeIterator*.

4.4. Četrti del naloge

Četrti del naloge je od naju zahteval, da izračunava $Dedim(g)$ za kartezične produkte dveh poti $P_k \Box P_t$. V ta namen sva definirala funkcijo *grid_ graph(i,j)*, ki kot argument sprejme števili i in j, $i \leq j$ in za vse kombinacije števila vozlišč med i in j nariše graf $P_k \Box P_t$ in pod njim izpiše vrednost $Dedim(P_k \Box P_t)$.

5. Ugotovitve

5.1. Prvi del naloge

Hitro je bilo vidno, da so drevesa, na n vozliščih, ki imajo maksimalen $Dedim(g)$ zvezde S_n, njihov $Dedim(S_n)$ pa je $n-1$. Za drevesa z minimalnim $Dedim(T)$ nisva našla neke oblike grafa, ki bi se do potankosti ohranjala za vse n, sva pa ugotovila, da za število vozlišč $n = 3k, 3k+1, 3k+2$ kjer je $k \geq 2$ velja, da je največja stopnja vozlišča v drevesu $\Delta(T) = k+1$, vozlišče s takšno stopnjo je natanko eno, drevo ima $k+1$ listov, vsa ostala vozlišča v, ki niso listi pa imajo $deg_T(v) = 2$. To sva preverila za drevesa z $|V(T)| \leq 20$ in nimava razloga, da ne bi verjela, da to velja

za vse n, bi se pa zavoljo matematične korektnosti to spodobilo dokazati, kar pa bi bilo(v primeru, da je sploh možno) za to projektno nalogo preobsežna naloga in niti ni naloga tega projekta.

število vozlišč	**največja stopnja vozlišča**
6	3
7	3
8	3
9	4
10	4
11	4
12	5
13	5
14	5

TABELA 1. Največja stopnja vozlišča v drevesu z min(Dedim(T))

SLIKA 1. Drevesa z $6 \leq |V(T)| \leq 11$ z minimalnim $Dedim(T)$

5.2. DRUGI DEL NALOGE

Ponovno sva hitro ugotovila vzorec, da so drevesa T na n vozliščih, ki imajo minimalen možen $Dedim(T) - edim(T)$ zvezde S_n in da je $Dedim(T) - edim(T) = 1$. Za drevesa z maksimalno razliko $Dedim(T) - edim(T)$ nisva našla oblike grafa, ki bi se do potankosti ohranjala za vse n, sva pa ugotovila, da za število vozlišč $n = 2k, 2k + 1$ kjer je $k \geq 3$ velja, da je razlika enaka $k - 1$. To sva preverila za

drevesa $|V(T)| \leq 20$ in ponovno nimava razloga, da ne bi verjela, da to velja za vse n.

število vozlišč	**max(Dedim(T) - edim(T))**
6	2
7	2
8	3
9	3
10	4
11	4
12	5
13	5

Tabela 2. Maksimalna razlika Dedim(T) - edim(T)

5.3. Tretji del naloge

Ugotovila sva, da so drevesa na n vozliščih, ki imajo minimalen količnik $Dedim(T)/edim(T)$ zvezde S_n, količnik pa bo za $n \geq 3$ enak $\frac{n-1}{n-2}$.

število vozlišč	**min(Dedim(T)/edim(T))**
3	2
4	1.5
5	1.3333
6	1.25
7	1.2
8	1.6667
9	1.1429
10	1.125

Tabela 3. Minimalen količnik Dedim(T)/edim(T)

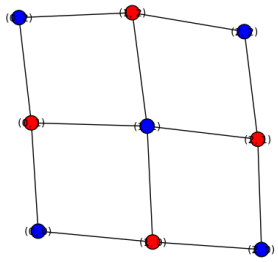
Ugotovila sva, da drevesom na n vozliščih z maksimalnim količnikom $Dedim(T)/edim(T)$ ustrezajo poti P_n, količnik pa je za $n \geq 4$ enak $\lfloor \frac{n}{2} \rfloor$.

število vozlišč	**min(Dedim(T)/edim(T))**
4	2
5	2
6	3
7	3
8	4
9	4
10	5
11	5

Tabela 4. Maksimalen količnik Dedim(T)/edim(T)

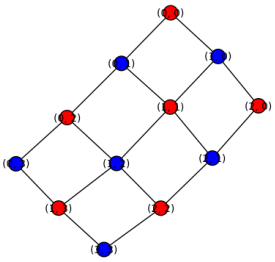
5.4. Četrti del naloge

Ugotovila sva, da je $Dedim(P_k \square P_t) = \left\lfloor \frac{k \cdot t}{2} \right\rfloor$, če je $k + t \geq 5$. Odkrila sva tudi katera vozlišča nastopajo v iskani množici. Če je $k \cdot t$ liho število, torej če sta k, t lihi potem so izbrana vozlišča (pobarvana z rdečo) kot sledi: v prvi vrstici izberemo drugo vozlišče po vrsti in potem vsakega drugega v tej vrstici, v naslednji vrstici začnemo s prvim vozliščem in potem izberemo vsakega drugega, v tretji vrstici ponovno začnemo z drugim vozliščem po vrsti itd.



Slika 2. Izbrana vozlišča, če je število vozlišč $P_k \square P_t$ liho.

Če je $k \cdot t$ sodo število, torej, če je vsaj eden izmed k, t sod potem so izbrana vozlišča kot sledi: v prvi vstici izberemo prvo vozlišče po vrsti in potem vsakega drugega v tej vrstici, v drugi vrstici izberemo drugo vozlišče po vrsti in potem vsekga drugega, v tretji vrstici spet začnemo s prvim vozliščem itd.



Slika 3. Izbrana vozlišča, če je število vozlišč $P_k \square P_t$ sodo.

6. Local search

Metodo local search sva dodala kot alternativo za velike grafe, saj najde približek rešitve dosti hitreje kot računanje prek CLP. Napisani algoritem deluje tako, da za podani graf najprej kreira poljubno dopustno rešitev, ki pa v večini primerov ni optimalna. Torej za začetno rešitev, si izbere poljubno množico vozlišč danega grafa, za katero velja, da razrešuje povezave ter je vozliščno pokritje. Ko imamo to začetno rešitev se lotimo iskanja optimalne rešitve. Pogledamo vsako vozlišče v začetni rešitvi in pogledamo, če imamo dopustno rešitev tudi, če začetni rešitvi odstranimo to vozlišče. V primeru, da imamo optimalno rešitev, ko odstranimo vozlišče gremo potem naprej do drugih vozlišč in tako preverimo za vsako vozlišče. To ni sicer najbolj "tipičen"local search algoritem. Primerneje bi sicer bilo, da bi za začetno rešitev na neki okolici pogledali, ali dobimo boljšo rešitev, če poljubno vozlišče iz začetne rešitve zamenjamo z kakšnim sosedom v grafu. Vendar za najin

problem to ni bilo izvedljivo, saj je bilo časovno preveč potratno. Ker je pa slabost najinega algoritma ta, da je končna rešitev močno odvisna od začetne rešitve, sva še malce izboljšala algoritem. Torej, za dani graf ustvarimo k začetnih rešitev in za vsako začetno rešitev izvedemo potem local search. Za končno rešitev pa potem vzamemo najbolj optimalno od vseh rešitev. Torej rešitev, kjer je množica vozlišč v rešitvi najmanjša. Ob testiranju algoritma na poljubnih grafih lahko ugotovimo, da algoritem ni pogosto različen od optimalne rešitve, je pa dosti hitrejši od CLP pri velikih grafih. Glede na preizuse sva ugotovila, da se za grafe do nekje 18 vozlišč splača uporabiti CLP, za večje grafe pa algoritem local search.

Vse funkcije, ki so omenjene v poročilu so napisane na Github-u v datoteki *clp.py*. Za pomoč pri nalogi sva uporabila spodnjo literaturo.

Literatura

[1] M. Tavakoli, M. Korivand, A. Erfanian, G. Abrishami, E. T. Baskoro, *The dominant edge metric dimension of graphs*, Electronic Journal of Graph Theory and Applications 11(1) (2023) 197–208.