UNIVERZA V LJUBLJANI
FAKULTETA ZA MATEMATIKO IN FIZIKO

Finančna matematika

Aljaž Kump, Mateo Vrtunski

Dominacijsko število in popolno dominacijsko število

Skupinski projekt pri Finančnem Praktikumu

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

Ljubljana, 26.1.2024

1. Navodilo naloge

State the problems of determining the domination number $\gamma(G)$ and the total domination number $\gamma_t(G)$ of a graph as ILPs. Also consider their LP relaxations. Experimentally study the differences between these four numbers for various classes of graphs. Try to determine some bounds on these numbers for trees, hypercubes, Kneser graphs,... with a given number of vertices n. If your programming environment already has implementations of these two invariants, then compare the speed of your program with the ones provided. Report your results.

2. Definicije

Za začetek samo par definicij, da vemo kaj dominacijsko število sploh je.

Definicija 2.1. *Množica $S \subseteq V$ vozlišč grafa $G = (V, E)$ je **dominacijska množica**, če je vsako vozlišče $v \in V$ element množice S ali pa je sosed (imata skupno povezavo) vsaj enemu vozlišču iz množice S.*

Definicija 2.2. ***Dominacijsko število grafa*** $\gamma(G)$ *je enako velikosti najmanjše dominacijske množice grafa G.*

Definicija 2.3. *Množica S vozlišč grafa G je **popolna dominacijska množica**, če je vsako vozlišče $v \in V$ sosed vsaj enemu vozlišču iz množice S (vključno z vozlišči množice S).*

Definicija 2.4. ***Popolno dominacijsko število grafa*** $\gamma_t(G)$ *je enako velikosti najmanjše popolne dominacijske množice grafa G.*

3. Opis dela

Prvo sva se lotila CLP in LP algoritma za iskanje $\gamma(G)$ in $\gamma_t(G)$. V najinem primeru nam CLP pove ali je vozlišče v S, torej vrednost 0 ali 1. LP pa nam pove s kolikšno verjetnostjo je vozlišče v S, torej pričakujemo manjše vrednosti, z decimalnimi števili. Ko sva preverila, da vse skupaj deluje pravilno sva poganjala algoritem na različnih grafih. S krajšim premislekom sva prišla do tega, da bi moralo biti $\gamma(G) \leq \frac{n}{2}$ in $\gamma(G) \leq \gamma_t(G) \leq 2\gamma(G)$ za povezane grafe. Naj bosta to najini splošni hipotezi za $\gamma(G)$ in $\gamma_t(G)$. Pri nepovezanih grafih pa v resnici gledamo dominacijsko število povezanega dela in prištejemo število izoliranih vozlišč. Popolno dominacijsko število pa ne obstaja za nepovezane grafe že iz definicije.
V prvem delu naju je samo zanimalo ali obstaja kakšna zveza med številom vozlišč in dominacijskim ter popolno dominacijskim številom. Nato sva pogledala še primerjavo z LP relaksacijo na nekaterih grafih. V teoriji bolj učinkovita metoda, ki CLP problem reši kot navaden LP. Na koncu sva pa še primerjala najin algoritem z vgrajenim. Osredotočila sva se na hitrost izvedbe, kajti izkazalo se je, da algoritem deluje pravilno in je vračal enake vrednosti za $\gamma(G)$ in $\gamma_t(G)$ kot vgrajena funkcija.

4. CLP in LP programi

Tukaj so opisane funkcije za CLP in LP za $\gamma(G)$ in $\gamma_t(G)$. Uporabljene bodo v prvih testiranjih na grafih:
CLP za dominacijsko število

```
def CLP_dominating_number(G):
p = MixedIntegerLinearProgram(maximization = False)
b = p.new_variable(binary = True)
p.set_objective(sum([b[v] for v in G]))

for u in G:
    p.add_constraint(b[u] + sum([b[v] for v in G.neighbors
       (u)]) >= 1)

return p.solve()
```

LP za dominacijsko število

```
def LP_dominating_number(G):
p = MixedIntegerLinearProgram(maximization = False)
b = p.new_variable(real = True, nonnegative = True)
p.set_min(b, 0)
p.set_max(b, 1)
p.set_objective(sum([b[v] for v in G]))

for u in G:
    p.add_constraint(b[u] + sum([b[v] for v in G.neighbors
       (u)]) >= 1)

return p.solve()
```

CLP za popolno dominacijsko število

```
def CLP_total_dominating_number(G):
p = MixedIntegerLinearProgram(maximization = False)
b = p.new_variable(binary = True)
p.set_objective(sum([b[v] for v in G]))

for u in G:
    p.add_constraint(sum([b[v] for v in G.neighbors(u)])
       >= 1)

resitev = p.solve()
#return len([v for v in G.vertices() if resitev[x[i]] ==
   1])
return resitev
```

LP za popolno dominacijsko število

```
    def LP_total_dominating_number(G):
    p = MixedIntegerLinearProgram(maximization = False)
    b = p.new_variable(real = True, nonnegative = True)
    p.set_min(b, 0)
    p.set_max(b, 1)
    p.set_objective(sum([b[v] for v in G]))

    for u in G:
        p.add_constraint(sum([b[v] for v in G.neighbors(u)])
           >= 1)

    resitev = p.solve()
    #return len([v for v in G.vertices() if resitev[x[i]] ==
       1])
    return resitev
```

5. Delo na grafih

Uporabljene funkcije:

```
    def dominacije_na_drevesih_z_n_vozli i(n):
        LP = []
        CLP = []
        LPt = []
        CLPt = []
        drevesa = list(graphs.trees(n))
        stevilo_dreves = len(drevesa)
        for i in range(stevilo_dreves):
            t = drevesa[i]
            LP.append(LP_dominating_number(t))
            CLP.append(CLP_dominating_number(t))
            LPt.append(LP_total_dominating_number(t))
            CLPt.append(CLP_total_dominating_number(t))
        return LP, CLP, LPt, CLPt

    def dominacije_na_kneserjevih_grafih(n,k):
        if 2*k > n:
            return False
        else:
            t = graphs.KneserGraph(n,k)
            show(t)
            LP = LP_dominating_number(t)
            CLP = CLP_dominating_number(t)
            LPt = LP_total_dominating_number(t)
            CLPt = CLP_total_dominating_number(t)
        return LP, CLP, LPt, CLPt
```

```
    def dominacije_na_hiperkocke(n):
        t = graphs.CubeConnectedCycle(n)
        show(t)
        LP = LP_dominating_number(t)
        CLP = CLP_dominating_number(t)
        LPt = LP_total_dominating_number(t)
        CLPt = CLP_total_dominating_number(t)
        return LP, CLP, LPt, CLPt

    def dominacije_na_naklju en_graf(n,p):
        t = graphs.RandomGNP(n, p, seed=None, fast=True,
           algorithm='Sage')
        show(t)
        LP = LP_dominating_number(t)
        CLP = CLP_dominating_number(t)
        LPt = LP_total_dominating_number(t)
        CLPt = CLP_total_dominating_number(t)
        return LP, CLP, LPt, CLPt
```

5.1. Drevesa

Naredila sva funkcijo, ki zgenerira vsa drevesa z n vozlišči, na njih pa požene CLP in LP za dominacijsko in popolno dominacijsko število. Drevesa so preprosti grafi, za povezan graf imajo celo najmanj povezav. Torej bomo tu našli zgornjo mejo za dominacijska števila na povezanih grafih. Razlik med LP in CLP ni bilo, vsa števila so bila cela pri LP. Spet, zaradi preprostosti grafov. Oglejmo si rezultate:

št. vozlišč (n)	$\max(\gamma(G))$	$\min(\gamma(G))$	$\max(\gamma_t(G))$	$\min(\gamma_t(G))$
2	1	1	2	2
3	1	1	2	2
4	2	1	2	2
5	2	1	3	2
6	3	1	4	2
7	3	1	4	2
8	4	1	5	2
10	5	1	6	2
⋮	⋮	⋮	⋮	⋮
n	$\lfloor \frac{n}{2} \rfloor$	1	$\lfloor \frac{n}{2} \rfloor + 1$	2

Ugotovimo da je $min(\gamma(G)) = 1$, kar predstavlja drevo s samimi listi. Pri temu drevesu je tudi $min(\gamma_t(G)) = 2$. Za maksimum pa ugotovimo, da je $\lfloor \frac{n}{2} \rfloor = max(\gamma(G))$. To je pri drevesu z potjo dolžine n za lihe n in pri drevesu z potjo dolžine $n-1$ za sode n, kjer je zadnje vozlišče n povezano z vozliščem $n-3$. To nista edina drevesa z $max(\gamma(G))$, ampak za te smo lahko prepričani. Za popolno dominacijsko število pa je $max(\gamma_t(G)) = \lfloor \frac{n}{2} \rfloor + 1$, ampak samo za $n \geq 5$. To pomeni: za $n = |V|$ je $\gamma(G) \leq \frac{n}{2}$ za vse povezane grafe, saj velja za drevesa, ki so povezani grafi z najmanj povezavami, torej imajo največje dominacijsko število glede na število vozlišč.

5.2. Kneserjevi grafi

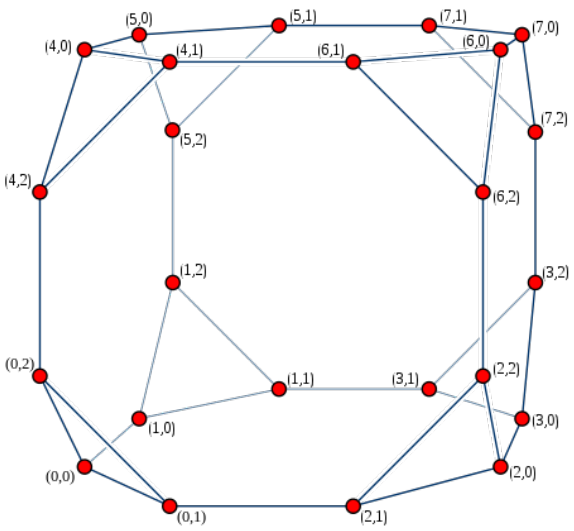
Kneserjev graf $K_{n,k}$ pri $n \geq 2k+1$ je graf, katerega točke ustrezajo podmnožici množice n elementov in kjer sta dve točki povezani, če in samo če sta odgovarjajoča seznama (množici) povezav disjunktna. Število vozlišč je enak $\binom{2n+k}{n}$, število povezav pa $\frac{1}{2}\binom{n+k}{n}\binom{2n+k}{n}$. Iz tu vidimo, da je vsako vozlišče stopnje $\binom{n+k}{n}$. Če je $k = 1$ imamo poln graf, $K_{5,2}$ pa je recimo petersonov graf. Poglejmo si tabelo:

$K(n,k)$	LP	CLP	LPp	CLPp
$K(5,2)$	2,50	3	3,33	4
$K(6,2)$	2,14	3	2,50	3
$K(7,2)$	1,91	3	2,10	3
$K(10,2)$	1,56	3	1,61	3
$K(11,2)$	1,49	3	1,52	3
$K(7,3)$	7	7	8,75	12
$K(8,3)$	5,09	7	5,60	8
$K(9,3)$	4	7	4,20	7

Ugotovimo, da za k = 2 je $\gamma(K(n,k)) = 3$, za k = 3 pa $\gamma(K(n,k)) = 7$. Za večje grafe ni uspel izračunati (sva pustila več ur). Nagiba se sicer, da je $\gamma(K(n,k)) = k^2 - k + 1$, ampak tega nisva mogla preveriti. Za popolno dominacijsko število $\gamma_t(K(n,k)) = 3$ za $k = 2$ in $n \geq 5$. Se pa LP vrednost vedno manjša, posledica več in več povezav. Lahko trdimo, da večji kot bo n manjši bo $\gamma_t(K(n,k))$ za fiksen k.

5.3. Hiperkocke

Nisva našla vgrajenega grafa za hiperkocko, zato sva vzela Čiklične kvadre: To je graf kjer oglišče kocke zamenjamo s ciklom dolžine n. Primer za $n = 3$:



n	$\gamma(G)$	$\gamma_t(G)$
2	6	8
3	16	24
4	46	56

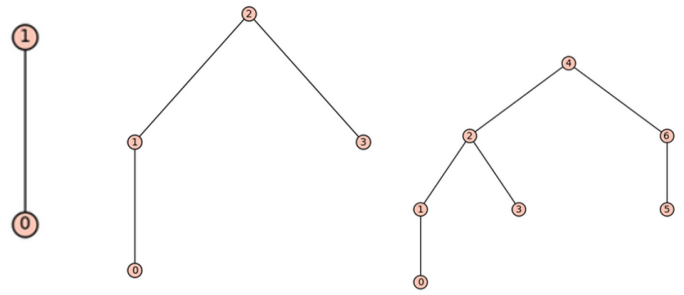
Za večje n sva pustila več ur in ni izračunal sva pa našla, da je za $\gamma(G)$ zaporedje tako: 6, 16, 46, 96, 224, 512, ...

6. Uporaba relaksacije

V tem delu bova še videla, kaj nama lahko prinese LP relaksacija. Za nadalje, $D(n)$ je dominacijsko v odvisnosti od n in $T(n)$ popolno dominacijsko število, lp je oznaka za LP relaksacijo.

6.1. Fibonaccijevo drevo

Fibonaccijevo drevo F_i je rekurzivno definirano. Na levi strani začetnega vozlišča je F_{i-1}, na desni pa F_{i-2}.



Uporabljena koda:

```
def Fib(a,b,n):
i = 0
while i < n:
    a,b = b, a+b
    i += 1
return a

def D_for_FibonacciTree(n):
seed = [0,1,1,2,3]
if n < len(seed):
    return seed[n]
else:
    for i in range(len(seed), n+1):
        seed.append(Fib(0,1,i) + seed[0])
        seed.pop(0)
return seed[-1]

def T_for_FibonacciTree(n):
seed = [0,0,2,2,4,6]
```

```
    if n < len(seed):
        return seed[n]
    else:
        for i in range(len(seed), n+1):
            seed.append(Fib(2,2,i-2) + seed[0])
            seed.pop(0)
    return seed[-1]
```

Oglejmo si tabelo:

n	D(n)	D(n)lp	t(n)	T(n)lp
2	1	1	2	2
3	2	2	2	2
4	3	3	4	4
5	5	5	6	6
6	9	9	10	10
7	14	14	17	17
8	23	23	28	28
9	37	37	44	44
10	60	60	72	72

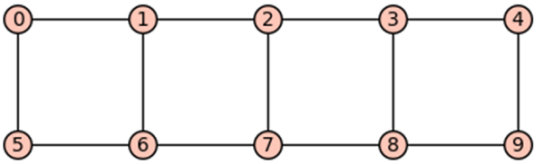
Opazimo, da s pomočjo linearne relaksacije dobimo isti rezultat. $D(G)$ ima podobno rast kot Fibonaccijevo zaporedje z začetnima členoma 0,1. Bolj natančno se da ugotoviti, da velja formula $D(n) = Fib(0, 1, n) + D(n - 5)$. V tem primeru namesto G pišemo število vozlišč n. Za $T(n)$ pa je najboljši približek $T(n) = Fib(2, 2, n - 2) + T(n - 6)$. Ta nam da pravilen odgovor skoraj za vse n. Izjema je, ko $(n - 1)$ deli 6. V tem primeru se pravilen rezultat zamakne za 1, -1.

6.2. Ciklični grafi

Tukaj smo opazili da, $D(n)$ je zaporedje $1, 1, 1, 2, 2, 2, 3, 3, 3, ... = \left\lfloor \frac{n+2}{3} \right\rfloor$. $D(n)lp$ je vedno manjši od D(n) in linearno narašča za $n > 2$ in je definiran kot D(n)lp $= \frac{n}{3}$. Z njim lahko tudi izračunamo $D(n)$. Če je ostanek deljenja n s 3 enak 1 prištejemo $\frac{2}{3}$, če je 2 prištejemo $\frac{1}{3}$ drugače 0. Zaporedje $T(n)$ ni enostavno zapisati v kompaktni obliki. S pomočjo Wolfram Alphe zaporedje sledi predpisu $\frac{1}{4}(2(n - 1) + i(-i)^{n-1} - i(i)^{n-1} + 4)$, i je imaginarno število. $T(n)lp$ je veliko bolj enostavno in je definirano kot $T(n)lp = \frac{n}{2}$. Odmik je -1 za $n = 4k + 2$, -0.5 za $n = 4k + 1$ ali $n = 4k + 3$ in 0 za $n = 4k$.

6.3. Ladder graph

Tako kot samo ime pove ima graf obliko lestve. Za vhod n vrne dve poti dolžine n povezani med sabo. Tabela:

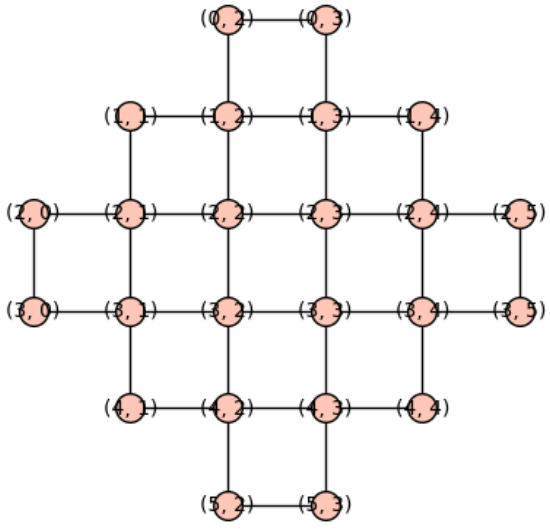


n	D(n)	D(n)lp	t(n)	T(n)lp
2	2	1,3	2	2
3	2	2	2	2
4	3	2,4	4	4
5	3	3,0	4	4
6	4	3,4	4	4
7	4	4	6	6
8	5	4,4	6	6
9	5	5,0	6	6
10	6	5,5	8	8

$D(n)$ ima zelo enostaven vzorec. Enak je $D(n) = 1 + \lfloor \frac{n}{2} \rfloor$. Najboljši približek za $D(n)lp$ je $\frac{n+1}{2}$. S tem lahko tudi določimo $D(n)$. Če je n liho je odmik 0, drugače pa $-0,5$. $T(n) = 2 \lfloor \frac{n+2}{3} \rfloor = T(n)lp$

6.4. Aztec Diamond Graph

Aztec Diamond Graph od n ima obliko enako območju, ko želimo izračunati $\left|x - \frac{1}{2}\right| + \left|y - \frac{1}{2}\right| \leq n$.



Uporabljena koda:

```
    def T_for_Aztec(n):
        seed = 0
        for i in range(1, n+1):
            seed += (1/2)*(2*i + (-1)**(i+1) + 1)
    return seed

    def D_for_Aztec(n):
    result = []
    current_number = 2

    for i in range(1, n + 1):
        repetition = 3 if current_number % 4 == 0 else 2
        result.extend([current_number] * repetition)
        current_number += 2

    return sum(result[:n+1])
```

Tabela:

n	D(n)	D(n)lp	t(n)	T(n)lp
1	2	1,3	2	2
2	4	4	4	4
3	8	6	8	8
4	12	10	12	12
5	16	14	18	18
6	22	19,2	24	24
7	28	25,3	32	32
8	36	31,7	40	40
9	44	39,8	50	50
10	52	47,65	60	60

Pri $D(n)$ gledamo razlike med $D(n)$ in $D(n-1)$. Te sledijo vzorcu 2,2,4,4,4,6,6,8,8,8,... Vsako sodo število zapišemo dvakrat, če deli 4 pa trikrat. Za $D(n)lp$ ne obstaja nobena enostavna formula. Vedno je manjši od D(n). Po naključju se izkaže, da je $D(n) = D(n)lp + 2\sqrt{D(n)lp}$ zelo dobra aproksimacija. Za $n = 1,\ldots,40$ je največja absolutna razlika 1,19. Pri $T(n)$ gledamo razlike med $T(n)$ in $T(n-1)$. Te sledijo vzorcu 4,4,6,6,8,8,10,10,...

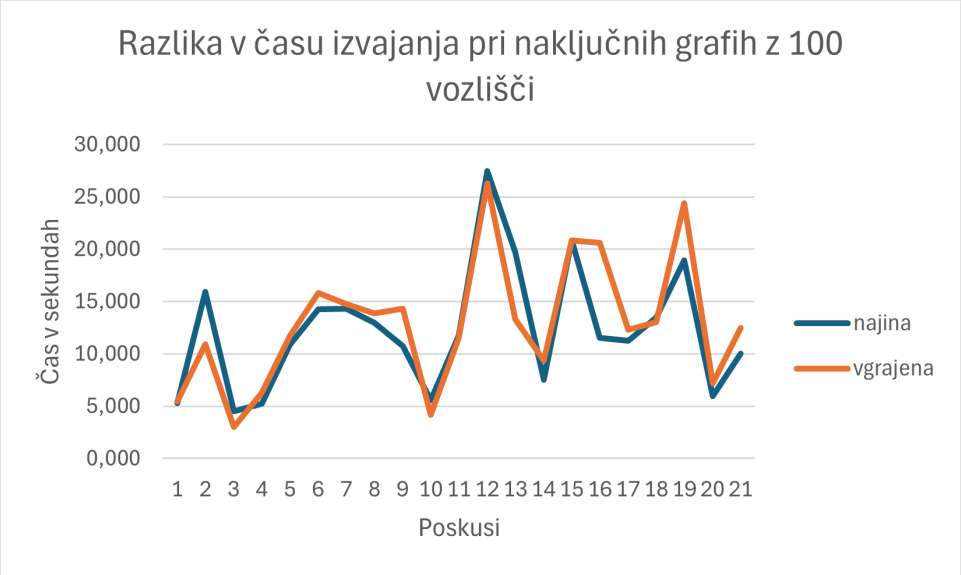
7. Primerjava z vgrajeno funkcijo

V sagu obstajata vgrajeni funkciji:

```
    G.dominating_set(value_only = True)
    G.dominating_set(total = True, value_only = True)
```

Prva vrne dominacijsko število, druga pa popolno dominacijsko število. Zanimalo naju je razmerje med hitrostjo izvedbe našega CLP-ja z vgrajenim. Na majhnih grafih kot so cikli in drevesa so bili časi milisekunde, razlike pa še manjše. Zato sva pognala 20x na naključnem grafu s 100 vozlišči in 0,2 verjetnost, da imata vozlišči

skupno povezavo (prejšnja skupina je ugotovila, da je to primerna vrednost). Rezultate sva ponazorila na grafu:



Rezultati so naju dokaj presenetili, saj sva pričakovala, da bo najina koda bistveno počasnejša. V resnici pa je za ta specifičen poskus, kar za 0,640854529 sekund hitrejša, v povprečju. Rada bi še izpostavila še to da so vrnjene vrednosti med našo in vgrajeno funkcijo vedno enake, še en dokaz, da funkcije delujejo pravilno.

8. Končne ugotovitve in zaključek

Ugotovila sva, da se na nekaterih tipih grafov lahko izpelje zaporedje oz. da je dominacijsko število odvisno od n. Zaporedja so lahko dokaj enostavna, lahko pa so precej kompleksna ali pa celo ne obstajajo. Skozi celotno delo so bile najine hipoteze pravilne, torej za povezane grafe velja:

$$\gamma(G) \leq \frac{n}{2} \text{ in } \gamma_t(G) \leq \gamma(G) \leq 2\gamma_t(G)$$

Pri LP Ugotovila sva tudi, da so relaksacije bistveno hitrejše. Tam smo tudi opazili vzorce med približkom in točno vrednostjo. Glede hitrosti smo prikazali, da je izračun popolno dominacijskega števila hitrejša kot dominacijsko število, kar je logično, saj ima popolno dominacijska množica manj pogojev za obstoj. Pozitivno presenečenje je seveda bilo pri primerjavi našega algoritma z že vgrajenim, nisva mogla pričakovati takšne uspešnosti. Vse skupaj se nama je zdelo zelo zanimivo in nisva imela težav pri iskanju motivacije za raziskovanje novih grafov. V programu se vidi, da sva še na drugih grafih delala ampak zaradi obsežnosti sva ohranila najbolj zanimive.