Retrakt za hiperkocke

Tomaž Pustotnik, Patrik Gorše

Februar 2017

1 Motivacija in opis problema

1.1 Definicije

Definicija 1.1. ***Homomorfizem*** *je preslikava med dvema grafoma, ki ohranja njuno strukturo. Preslikava f iz grafa $G = (V, E)$ v graf $G' = (V', E')$ je torej homomorfizem, če za $f : V \to V'$ velja, da slika iz vozlišč iz grafa G v vozlišča iz grafa G' tako, da za povezavi u, v vsebovani v E, velja f(u), f(v) vsebovani v E'.*

Definicija 1.2. *Graf H, ki ga dobimo iz grafa G tako, da odstranimo nekaj vozlišč in povezav je* ***podgraf*** *grafa G.*

Definicija 1.3. ***Retrakt*** *grafa G je njegov podgraf H, pri čemer obstaja homomorfizem $f : G \to H$.*

Definicija 1.4. ***Hiperkocka*** *Q_n je n-dimenzionalni analog kvadratu (n=2) in kocki (n=3). Iz oglišča n-dimenzionalne hiperkocke poteka n povezav, zato lahko n-dimenzionalno hiperkocko opišemo kot graf $G = (V, E)$, kjer naključno vozlišče postavimo v izhodšče n-dimenzionalnega koordinatnega sistema. Brez škode za splošnost lahko predpostavimo, da je dolžina vseh povezav v G enaka 1, zato lahko vsakemu vozlišču smiselno priredimo koordinate, ki naj bi mu pripadale glede na njegovo lokacijo v koordinatnem sistemu. Ker smo predpostavili dolžino vseh povezav 1, lahko vsaka koordinata zavzame zgolj vrednosti 0 ali 1.*
Primer: Kvadrat opišejo točke (0,0), (0,1), (1,0), (1,1).

1.2 Opis problema

Najina naloga bo napisati celoštevilski linearni program s katerim bo mogoče ugotoviti ali obstaja preslikava, ki je homomorfizem za dano hiperkocko Q_n in nek njen podgraf. Za najino nalogo sva napisala tri različne programe. V prvem primeru sva napisala program, ki preveri ali je nek določen podgraf hiperkocke njen retrakt. V drugem primeru pa sva napisala program, ki za dano hiperkocko preveri kolikšen delež vseh njenih podgrafov je homomorfnih. Toda ker število vozlišč v hiperkocki Q_n znaša 2^n, torej njihovo število narašča eksponentno, sva napisala še program, ki za velike n naključno generira podgrafe in na koncu vrne približen delež podgrafov hiperkocke, ki so njeni retrakti.

2 ILP formulacija

Rešujemo problem pri katerem bo program preveril, če je nek določen podgraf hiperkocke njen retrakt. Podana sta grafa $G = (V, E)$ (hiperkocka) in njegov podgraf $H = (V', E')$. Definirajmo spremenljivko

$$y_{u,v} = \begin{cases} 1 & ; \text{ vozlišče u iz grafa G se homomorfno preslika v vozlišče v iz grafa H} \\ 0 & ; \text{ sicer} \end{cases}$$

kjer bo ILP maksimiziral. Kaj točno maksimiziramo ne rabimo zapisati, saj ugotavljamo le dopustnost problema. Če je problem dopusten, je podgraf H retrakt hiperkocke, v nasprotnem primeru pa ni. Sedaj zapišimo naš problem kot celoštevilski linearni programiram.

p.p.

(1) $\sum_{v\in[H]} y_{u,v} = 1$ za vsako vozlišče u iz grafa G.

(2) $y_{v,v} = 1$ za vsako vozlišče v iz grafa H.

(3) $y_{u,w} + y_{v,z} \leq 1$ za vsak par vozlišč (u, v) iz E in vsak par vozlišč (w,z) iz E'^c.

(4) $y_{u,z} + y_{v,w} \leq 1$ za vsak par vozlišč (u, v) iz E in vsak par vozlišč (w,z) iz E'^c.

(5) $y_{u,z} + y_{v,z} \leq 1$ za vsak par vozlišč (u, v) iz E in vsak z iz V'.

Zgoraj navedeni pogoji nam povedo naslednje:

(1) Vsako vozlišče iz grafa G se preslika v natanko eno vozlišče v grafu H.
(2) Če je vozlišče v vsebovano v grafu H, se mora iz grafa G preslikati samo vase.
(3) in (4) Če sta vozlišči u in v sosednji v grafu G ter vozlišči w in z nista sosednji v grafu H, potem se ne more vozlišče u preslikati v w ter vozlišče v v z in obratno. Torej homomorfna preslikava mora ohranjati sosednjost.
(5) Če sta vozlišči u in v sosednji v grafu G, potem se ne moreta preslikati v isto vozlišče v grafu H.

S pomočjo teh pogojev sva napisala tri programe, ki preverjajo homomorfnost podgrafov hiperkocke. Pri vseh programih sva podgrafe generirala tako, da sva iz grafa G odstranjevala vozlišča in ne povezav. Ko iz grafa G odstranimo vozlišče, seveda odstranimo tudi vse povezave, ki jih je to vozlišče imelo s svojimi sosedi. Nikoli pa nisva odstranila povezave med dvema vozliščema in istočasno obe vozlišči pustila v podgrafu H, saj avtomatično vemo, da takšni podgrafi niso retrakti hiperkocke.

Dokaz:
Imejmo neprazen graf G v katerem imamo povezani vozlišči u in v. Recimo, da generiramo podgraf H, ki tudi vsebuje vozlišči u in v, vendar v njem med vozliščema ne obstaja povezava. V tem primeru dobimo protislovje, saj smo v pogoju (2) zapisali, da se vsako vozlišče, ki je vsebovano v grafu G in H hkrati, preslika samo vase. Toda istočasno morata biti po definiciji homomorfizma sliki vozlišč v grafu H sosednji, če sta bila tudi njuna originala sosednji vozlišči v grafu G. Iz tega sledi, da so edini realistični kandidati za podgrafe grafi, ki jim odstranimo vozlišča.

3 Podgraf

Ta program kot 'input' dobi hiperkocko in nek njen ustrezen podgraf, kot 'output' pa nam pove ali je podgraf retrakt hiperkocke ali ne. Na začetku v spremenljivko a zapišemo koliko dimenzionalno hiperkocko želimo imeti. Nato ročno zapišemo podgraf H tako, da napišemo katera vozlišča iz hiperkocke bomo izbrisali. Program potem poskuša rešiti celoštevilski linearni program pri pogojih (1),(2),(3),(4) in (5).
Programirala sva v Sagemath-u. Problem sva zapisala kot celoštevilski linearni program, kjer sva preverjala zgolj dopustnost problema. Spodaj je primer za hiperkocko dimenzije $n = 3$ in podgraf brez vozlišč '000' in '010'.

```
n = 3    #povemo dimenzijo hiperkocke
G = graphs.CubeGraph(n)    #generiramo hiperkocko
H = G.subgraph(v for v in G if v not in ['000','010'])    #podgraf brez vozlišč '000' in '010'

p = MixedIntegerLinearProgram(maximization = True)    #definiramo linearni program
y = p.new_variable(binary = True)    #definiramo binarno spremenljivko y

for u in G.vertices():    #za vsako vozlišče u iz grafa G

    p.add_constraint(sum([y[u, v] for v in H.vertices()]) == 1)    #vsako vozlišče u iz grafa G
```

se lahko preslika v natanko eno vozlišče v grafu H

for v in H.vertices(): #za vsako vozlišče v iz grafa H

p.add_constraint(y[v,v]==1) #če je vozlišče vsebovano v grafu G in H hkrati, se mora preslikati samo vase

for u, v in G.edges(labels=False): #za vsaki dve sosednji vozlišči v grafu G

for w, z in H.complement().edges(labels=False): #za vsaki dve nesosednji vozlišči v grafu H

p.add_constraint(y[u, w]+y[v, z]<=1) #ne more se hkrati preslikati vozliče u v w in vozlišče v v z

p.add_constraint(y[u, z]+y[v, w]<=1) #ne more se hkrati preslikati vozliče u v z in vozlišče v v w

for u, v in G.edges(labels=False):

for z in H.vertices():

p.add_constraint(y[u,z]+y[v,z]<=1) #ne moreta se dve sosednji vozlišči preslikati v isto vozlišče

try:

p = p.solve()

print("Ta podgraf je retrakt hiperkocke") #če se da rešiti ILP, povej da sta grafa homomorfna
except:

print("Ta podgraf ni retrakt hiperkocke") #sicer povej, da grafa nista homomorfna

4 Delež podgrafov

Pri tem programu naju je zanimalo kolikšen delež ustreznih podgrafov dane hiperkocke je homomorfnih. Kot 'input' ustavimo spremenljivko a s pomočjo katere generiramo n-dimenzionalno hiperkocko. Program potem sam zgenerira seznam vseh ustreznih podgrafov in jih shrani v spremenljivko S. Nato za vsak podgraf posebej enako kot pri programu 'Podgraf' preverimo ali je homomorfen dani hiperkocki.

```
n = 3    #definiramo n-dimenzionalno hiperkocko
G = graphs.CubeGraph(n)    # generiramo hiperkocko
S = Subsets(G.vertices())    #naredimo seznam vseh podgrafov naše hiperkocke

b = 0
c = 0

for s in S:
if len(s) == 0:
continue
Gs = G.subgraph(s)    #definiramo podgraf za vsak s iz S
b = b + 1    #povečamo spremenljivko b za ena, da preštejemo število vseh podgrafov
```

```
p = MixedIntegerLinearProgram(maximization = True)
y = p.new_variable(binary = True)

for u in G.vertices():
p.add_constraint(sum([y[u, v] for v in Gs.vertices()]) == 1)    #pogoj (1)

for v in Gs.vertices():    #pogoj (2)
p.add_constraint(y[v,v]==1)

for u, v in G.edges(labels=False):    #pogoja (3) in (4)
for w, z in Gs.complement().edges(labels=False):
p.add_constraint(y[u, w]+y[v, z]<=1)
p.add_constraint(y[u, z]+y[v, w]<=1)

for u, v in G.edges(labels=False):    #pogoj (5)
for z in Gs.vertices():
p.add_constraint(y[u,z]+y[v,z]<=1)

 try:
p = p.solve()
Gs.vertices()    #če je problem dopusten, izpiši vozlišča podgrafa
c = c + 1    #če je problem dopusten, povečaj c za ena
except:
pass

c/b    #na koncu izračunamo delež podgrafov, ki so homomorfni
```

5 Približni delež podgrafov

Kar naju je najbolj zanimalo je to, kako se spreminja delež podgrafov n-dimenzionalne hiperkocke, ko se n veča. Problem pa je, da se število ogljišč veča eksponentno, saj znaša 2^n za n-dimenzionalno hiperkocko. Zaradi tega lahko pri programu 'Delež podgrafov' računamo samo do n=3. Ker pa bi rada izračunala deleže tudi za večje n, sva napisala še tretji program pri katerem generirava naključne podgrafe in nato postopek nekajkrat ponoviva, da dobiva približen rezultat.
Vendar je treba podgrafe generirati na poseben način. Če bi najprej pustili cel graf, potem vzeli eno naključno vozlišče, nato vzeli dve naključni vozlišči in tako naprej dokler nebi pustili samo podgrafa z enim vozliščem, bi dobili napačen rezultat. Število podgrafov, ki jih naključno zgeneriramo, mora biti enako porazdeljeno kot dejansko število podgrafov hiperkocke in njihovo število je porazdeljeno binomsko. Razlog je v tem, da če hočemo iz hiperkocke z m vozlišči izbrisati n vozlišč, kjer je $m > n$, je število vseh podgrafov enako $\binom{m}{n}$.

Primer:
Imejmo dvodimenzionalno hiperkocko, torej je naš graf G enak kvadratu. Iz njega lahko zgeneriramo 4 podgrafe z enim vozliščem, 6 podgrafov z dvema vozliščema, 4 podgrafe s tremi vozlišči in enega, ki je enak grafu G. Če poženemo naš drugi program 'Delež podgrafov', ugotovimo da je od 15 podgrafov 9 homomorfnih hiperkocki.

$$(0 * 4 + 2/3 * 6 + 1 * 4 + 1 * 1)/15 = 3/5 = 9/15$$

Če bi predpostavljali enakomerno porazdelitev in ohranili vsa vozlišča, izbrisali eno naključno vozlišče, potem dve naključni vozlišči, nato tri, bi nam program podal približno število homomorfnih grafov dve tretjini.

$$(0 + 2/3 + 1 + 1)/4 = 2/3$$

Vidimo torej, da mora biti število podgrafov porazdeljeno binomsko, sicer nam bo program podal napačen rezultat.

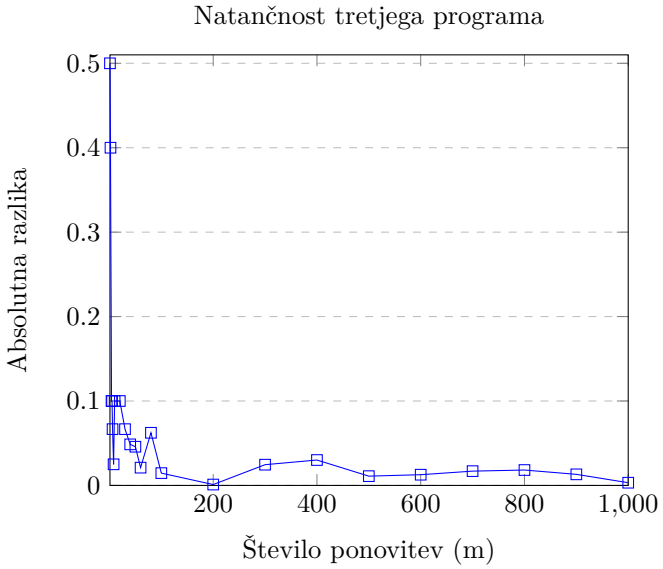
Najin program kot input dobi spremenljivko a, ki pove dimenzijo hiperkocke in kot output pove kolikšen delež podgrafov je retraktov hiperkocke.

```
n = 2
G = graphs.CubeGraph(n)    #generiramo hiperkocko
b = 0
c = 0
k = 2^n
d = scipy.stats.binom(k, 0.5)    #binomska verjetnostna porazdelitev

m=200    #postopek ponovimo dvestokrat
while m>0:
vozlisca = G.vertices()    #generiramo seznam vseh vozlišč grafa G
shuffle(vozlisca)    #premešamo vozlišča, da ne bi vedno izbrisali enih in istih
H = G.subgraph(vozlisca[:d.rvs()])    #izbrišemo prvih nekaj vozlišč iz seznama, kjer je d.rvs() porazdeljen binomsko
if H.order()>0:    #ne vpoštevamo primera, ki izbrišemo vsa vozlišča iz grafa G

for u in G.vertices():    #pogoj (1)
p.add_constraint(sum([y[u, v] for v in H.vertices()]) == 1)

for v in H.vertices():    #pogoj (2)
p.add_constraint(y[v,v]==1)

for u, v in G.edges(labels=False):    #pogoja (3) in (4)
for w, z in H.complement().edges(labels=False):
p.add_constraint(y[u, w]+y[v, z]<=1)
p.add_constraint(y[u, z]+y[v, w]<=1)

for u, v in G.edges(labels=False):    #pogoj (5)
for z in H.vertices():
p.add_constraint(y[u,z]+y[v,z]<=1)

try:
p = p.solve()
c = c + 1
except:
pass
m = m - 1

c/b    #na koncu izračunamo delež homomorfnih podgrafov
```

6 Natančnost tretjega programa

Najin zadnji program seveda približno izračuna delež retraktov hiperkocke, zato nas zanima njegova natančnost. Iz programa je jasno razvidno, da ko $m \to \infty$, bo tudi delež podgrafov konvergiral proti pravi vrednosti. Iz spodnjega grafa je razvidna natančnost programa. Na abscisi povečujemo število ponovitev programa (m) in na ordinati gledamo absolutno razliko med pravo vrednostjo izračunano s programom 'Delež podgrafov' ter približno vrednostjo izračunano s pomočjo tretjega programa 'Približni delež podgrafov'. Natančnost preizkušamo za dvodimenzionalno hiperkocko za katero vemo, da je njen natančen delež podgrafov 60%. Iz grafa lahko razberemo, da bo v povprečju absolutna napaka manjša,

ko povečujemo število ponovitev. Seveda pa to velja zgolj v povprečju, saj se lahko zgodi, da bo program pri manjšem m izračunal delež bolj natančno kot pri velikem, toda zakon velikih števil. Za $n > 150$ je absolutna napaka že konsistentno manjša od 0.05, kar je razlog, da



7 Analiza rezultatov

Zanimalo naju je, kako se spreminja delež homomorfnih podgrafov, ko se n-dimenzionalna kocka veča. Spodnja tabela prikazuje to odvisnost. Deleži označeni z * so izračunani s tretjim programom z 200 ponovitvami, torej predstavljajo približne vrednosti.

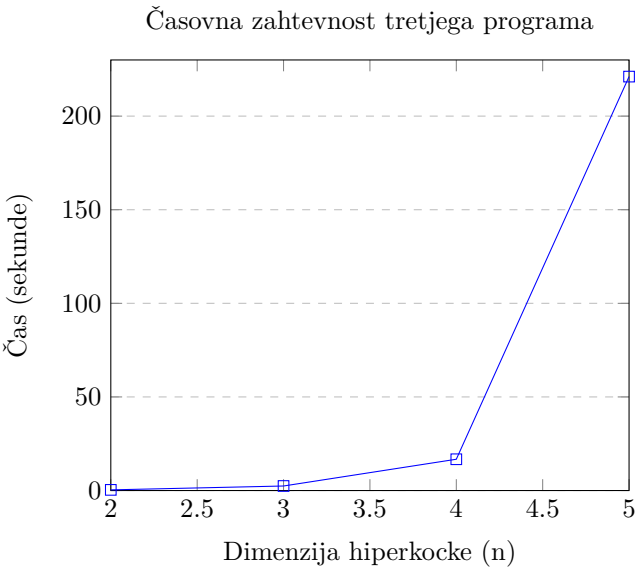
Tabela 1: Deleži homomorfnih podgrafov

n-dimenzionalna hiperkocka	delež homomorfnih podgrafov
2	0.6
3	0.4353
4	0.05*
5	0*

Iz tabele lahko vidimo, da postaja delež retraktov izjemno majhen, ko se dimenzija hiperkocke povečuje. Že pri $n = 5$ je delež tako majhen, da ga numerično zaokroževanje v računalniku zapiše kot ničlo.

8 Časovna zahtevnost

Pri najinem tretjem programu, ki edini lahko računa probleme za $n > 3$ sva računala koliko časa porabi za izračun deleža homomorfnih podgrafov hiperkocke. Spodnji graf prikazuje povezavo med dimenzijo hiperkocke in časom, ki je potreben za izračun. Vidimo lahko, da se čas potreben za rešitev problema povečuje eksponentno.


Časovna zahtevnost tretjega programa
Čas (sekunde)
0
50
100
150
200
2
2.5
3
3.5
4
4.5
5
Dimenzija hiperkocke (n)