ELEKTROTEHNIŠKI VESTNIK 85(5): 224–228, 2018
IZVIRNI ZNANSTVENI ČLANEK

Sosednost vozlišč v hipergrafih

Tomaž Hočevar[1], **Andrej Brodnik**[1,2], **J. Ian Munro**[3]

[1]*Univerza v Ljubljani, Fakulteta za računalništvo in informatiko, Večna pot 113, 1000 Ljubljana, Slovenija*
[2]*Univerza na Primorskem, Fakulteta za matematiko, naravoslovje in informacijsko tehnologijo, Glagoljaška 8, 6000 Koper, Slovenija*
[3]*David R. Cheriton School of Computer Science, Univerza v Waterlooju, Waterloo ON, Kanada N2L 3G1*
E-pošta: tomaz.hocevar@fri.uni-lj.si

Povzetek. V članku predstavimo algoritem za odgovarjanje na poizvedbe o sosednosti vozlišč v hipergrafu ali za zgraditev celotne matrike sosednosti. V primerjavi z naivno metodo preverjanja vseh hiperpovezav je algoritem hitrejši za logaritemski faktor. V hipergrafu z n vozlišči, m hiperpovezavami in vsoto velikosti hiperpovezav M odgovarja predstavljeni algoritem na posamezne poizvedbe o sosednosti v času $O(\frac{m}{\log m})$, s časovno zahtevnostjo predprocesiranja $O(M + \frac{m^{1+\varepsilon}}{\log m})$. Temelji na novem pristopu ekvivalenčnih razredov vozlišč. Predstavljen pristop je prenosljiv tudi na problem iskanja prič pri izračunu produkta dveh matrik z logičnimi oz. dvojiškimi vrednostmi.

Ključne besede: sosednost, hipergraf, množenje matrik, priča, ekvivalentnost

Node adjacency in hypergraphs

We present an algorithm for answering adjacency queries or building an adjacency matrix in a hypergraph. The algorithm exhibits a logarithmic speed-up compared to a naïve method of examining all hyperedges. In a hypergraph with n nodes, m hyperedges and sum of hyperedge sizes M, it answers individual adjacency queries in $O(\frac{m}{\log m})$, with a $O(M + \frac{m^{1+\varepsilon}}{\log m})$ preprocessing time. It represents a new approach based on equivalence classes of nodes. The results are also applicable to detecting witnesses in Boolean matrix products.

Keywords: adjacency, hypergraph, matrix product, witnesses, equivalency

1 UVOD

Hipergrafi so zelo splošen način predstavitve relacij med množicami objektov, še zlasti tedaj, ko relacije med objekti niso zgolj dvojiške, temveč večmestne. k-mestna relacija lahko na zgoščen način predstavlja množico dvojiških relacij med vsemi pari objektov (npr. predstavlja kliko v grafu brez naštevanja vseh $k(k-1)/2$ povezav med pari vozlišč). Take splošne predstavitve s hipergrafi se uporabljajo na področjih, kot so bioinformatika [1], klasifikacija slik [2], strojno učenje [3] in analiza (družbenih) omrežij [4].

Sosednost vozlišč v hipergrafih lahko definiramo na več načinov, to je odvisno od področja uporabe. Raziskave na področju analize omrežij (npr. [5], [6]) se večinoma osredotočajo na iskanje smiselnih in uporabnih definicij (sosednost lahko npr. predstavimo z utežjo, ki označuje število skupnih sosedov) in ne na učinkovitost izračuna definirane sosednosti.

Izračun matrike sosednosti (glej razdelek 1.2) lahko prevedemo na problem množenja dveh incidenčnih matrik. Za ugotavljanje sosednosti zgolj enega para vozlišč je optimalna naivna metoda. Po drugi strani pa so za izračun celotne matrike sosednosti asimptotično boljša izbira metode za hitro množenje matrik. Naša predlagana metoda se uvršča med ti skrajnosti. Pomeni nov pristop na podlagi skupin ekvivalentnih vozlišč, ki je dovolj splošen, da je lahko potencialno uporaben tudi pri reševanju kakšnega drugega algoritmičnega problema.

Poglejmo si naslednji motivacijski problem. Recimo, da je bilo v preteklem letu organiziranih m konferenc. Za vsako konferenco poznamo seznam udeležencev, ki so neka podmnožica vseh n aktivnih raziskovalcev. Predpostavimo, da se med konferenco vsi prisotni raziskovalci spoznajo med seboj. Kako lahko sedaj na učinkovit način odgovarjamo na vprašanja, ali se dva raziskovalca poznata? In če se, na kateri konferenci sta se spoznala?

1.1 Hipergrafi

Hipergraf je posplošitev grafa in je sestavljen iz množice vozlišč V ($n = |V|$) in množice hiperpovezav E ($m = |E|$). V primerjavi s povezavami v navadnih grafih so hiperpovezave $e \in E$ predstavljene s podmnožico vozlišč ($e \subseteq V$) in ne vedno s točno dvema vozliščema, kot velja v navadnih grafih. Naj bo $m_i = |E_i|$ velikost i-te hiperpovezave in $M = \sum m_i$ velikost problema oz. vhodnih podatkov. Za podrobnejšo predstavitev hipergrafov bralcu svetujemo knjigo Hypergraph Theory [7].

Na hipergraf $G(V, E)$ lahko gledamo na dva načina. Lahko ga obravnavamo kot družino množic ali pa kot

Prejet 8. avgust, 2018
Odobren 28. september, 2018

dvodelni graf. V drugem primeru prva skupina vozlišč dvodelnega grafa ustreza vozliščem V hipergrafa G, druga skupina pa povezavam E. Vozlišči dvodelnega grafa $v \in V$ in $e \in E$ sta povezani natanko takrat, ko je vozlišče v element množice oz. hiperpovezave e.

1.2 Sosednost vozlišč

Vozlišči $a, b \in V$ hipergrafa sta *sosedni*, če si delita katero od hiperpovezav ($\exists e \in E\colon a \in e \wedge b \in e$). V skladu s to definicijo bi lahko hipergraf predstavili z grafom, v katerem bi vsako hiperpovezavo modelirali s kliko vozlišč.

Naivni način za ugotavljanje sosednosti je, da preverimo vse hiperpovezave in ugotovimo, ali sta v kateri prisotni obe vozlišči, ki nas zanimata (a in b). Množico vozlišč, ki sestavljajo hiperpovezavo, lahko hranimo v razpršeni množici (angl. *hash set*), ki omogoča poizvedbe o prisotnosti nekega elementa v množici v pričakovanem konstantnem času. Za ta problem obstajajo tudi naprednejše tehnike, ki odgovarjajo na poizvedbe v konstantnem času [8], [9]. Časovna zahtevnost posamezne poizvedbe o sosednosti je torej $O(m)$. Za gradnjo matrike sosednosti, ki vsebuje podatke o sosednosti vseh parov vozlišč, pa s tem načinom potrebujemo $O(n^2 m)$ časa. Cilj je izboljšati ta naivni pristop.

2 SORODNA DELA

V matriki sosednosti lahko namesto podatka o sosednosti hranimo število hiperpovezav, v katerih hkrati nastopata obe vozlišči. S tem pravzaprav dobimo še podrobnejšo sliko sosednosti. Matriko števila skupnih hiperpovezav M lahko izračunamo kot produkt incidenčne matrike A in njene transponirane matrike A^T ($M = AA^T$). V incidenčni matriki A ustrezajo vrstice vozliščem, stolpci pa hiperpovezavam. Incidenčna matrika vsebuje vrednost 1 v j-tem stolpcu i-te vrstice, če je i-to vozlišče del j-te hiperpovezave, sicer je na tem mestu vrednost 0. Produkt matrik lahko izračunamo s katerim od algoritmov za hitro množenje matrik, ki ima časovno zahtevnost $O(n^\omega), \omega = 2.373$ [10].

Za izračun, ali obstaja skupna hiperpovezava, namesto točnega števila skupnih povezav lahko uporabimo katerikoli algoritem za množenje dvojiških matrik oz. matrik logičnih vrednosti. Npr. Yu je v [11] predstavil algoritem s časovno zahtevnostjo $O(n^3/\log^4 n \cdot (\log\log n)^6)$.

Iskanje skupne hiperpovezave oz. predstavnika skupnih hiperpovezav ustreza iskanju *prič* v produktu dveh matrik logičnih vrednosti. Algoritmi za iskanje takih prič so bili obravnavani v okviru problema najkrajših poti med vsemi pari vozlišč [12] in pri iskanju najnižjega skupnega prednika (angl. *lowest common ancestor*) vseh parov vozlišč v usmerjenih acikličnih grafih [13].

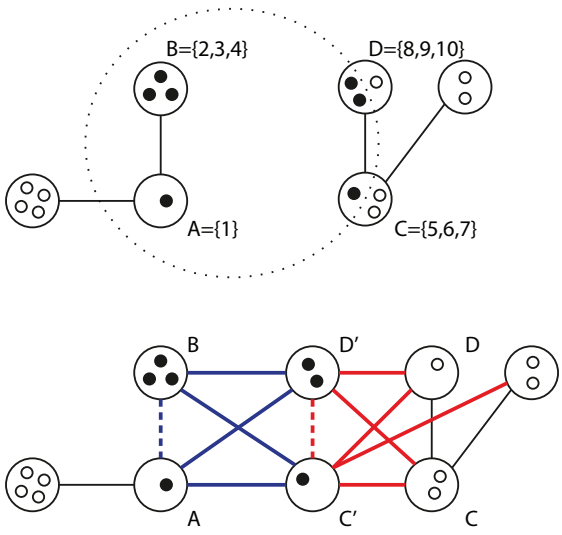
Za iskanje prič obstaja preprost algoritem z uporabo naključnosti (naključni algoritem tipa Las Vegas) s pričakovano časovno zahtevnostjo $O(n^\omega \log n)$ [12]. Obstajajo tudi derandomizirane (deterministične) različice tega pristopa s polilogaritemsko časovno zahtevnostjo $O(n^\omega \log^c n)$, vendar z veliko konstanto c.

Galil in Margalit [14] sta predstavila drugačen deterministične algoritem za iskanje prič pri množenju matrik logičnih vrednosti. Njun pristop temelji na razbitju matrik v bloke, ki jih lahko zmnožimo z algoritmom za hitro množenje matrik. Tako ugotovimo, v katerem bloku se nahaja pozitivno število prič, ki jih lahko nato s podobnim postopkom rekurzivno poiščemo samo v tem bloku. Časovna zahtevnost njune rešitev je $O(n^{\omega+\log^{-1/3} n})$. Kowaluk s sodelavci [13] je prilagodil njun pristop za iskanje *največje priče* s časovno zahtevnostjo $O(n^{2+\frac{1}{4-\omega}}) = O(n^{2.613})$, kar je pozneje Czumaj [15] izboljšal na $O(n^{2.575})$.

3 PODATKOVNA STRUKTURA

Podan je hipergraf $G(V, E)$. Naša metoda temelji na postopnem izboljševanju razbitja množice [16] s konstrukcijo in vzdrževanjem pomožnega grafa $H(T, P)$, ki ga imenujemo *graf razredov*. Vsako vozlišče $t \in T$ ustreza ekvivalenčnemu razredu vozlišč $t \subseteq V$, kjer je ekvivalenčna relacija definirana kot sosednost vozlišč v G. Postopek vzdržuje naslednji invarianti. Vsa vozlišča v razredu $t \in T$ so sosedna med seboj in za vsak par vozlišč $u, v \in t$ velja, da sta u in v nekem tretjem vozlišču $w \in V$ bodisi obe sosedni bodisi nobeno sosedno. Sosednost med vozliščema u in v v G, kjer $u \in t_1$ in $v \in t_2$, predstavimo s povezavo $(t_1, t_2) \in P$ v grafu razredov H. S tako strukturo lahko odgovorimo na poizvedbo o sosednosti dveh vozlišč v G preprosto tako, da ugotovimo, kateremu razredu pripada vsako od vozlišč. Vozlišči sta sosednji, če pripadata istemu razredu, ali pa sta pripadajoča razreda sosednja v H.

Algoritem se začne s praznim grafom razredov



Slika 1: Sprememba grafa razredov zaradi nove hiperpovezave e (črna vozlišča), ki zajema vsa vozlišča iz razredov A in B ter del vozlišč iz razredov C in D

$H(T = \emptyset, P = \emptyset)$ in množico vozlišč U, ki v vsakem trenutku vsebuje vozlišča iz V, ki niso v nobenem od razredov v T, se pravi $U = V \setminus \bigcup T$. Nato zaporedoma obravnavamo hiperpovezave $e \in E$ v poljubnem vrstnem redu. Vsaka obravnavana hiperpovezava lahko doda nov razred vozlišč iz U v T in delno ali pa v celoti vsebuje katere od že vzpostavljenih razredov v T.

Za vsako hiperpovezavo $e \in E$ (gl. algoritem 1) obravnavamo njene delne preseke z razredi (vozlišči) $M_i \in T$, ki jih po potrebi razcepimo (ustvarimo novo vozlišče v H) tako, da ohranjamo zgoraj definirani invarianti. Odcepljeni razred razreda M_i povežemo z vsemi razredi v H, s katerimi je bil povezan razred M_i. V H ustvarimo tudi nov razred M', ki vsebuje vozlišča iz U; to so tista, ki se še ne pojavijo v T. Tako pridemo do stanja, kjer je hiperpovezava e sestavljena iz množice razredov $X \subseteq T$, pri čemer so nekateri od njih že povezani med seboj. Na koncu sosednost med razredi v X označimo tako, da jih povežemo v kliko v H.

Algoritem 1: Gradnja in poizvedba v grafu razredov

```
function GrafRazredov.Dodaj(e)
    M_i ← {x | x ∈ e, Razred(x) = i}
    M' ← {x | x ∈ e, x ∈ U}
    U ← U \ M'
    X ← {M'}        ▷ novi razred bo v kliki razredov
    for all M_i do                    ▷ razcepi razrede
        if |M_i| < VelikostRazreda(i) then
            X ← X ∪ { Razcepi(i) }
        else
            X ← X ∪ {M_i}
    for all x ∈ X do                  ▷ naredi kliko razredov
        for all y ∈ X do
            if not Sosednja(x, y) then
                Poveži(x, y)

function GrafRazredov.Poizvedba(a, b)
    A ← Razred(a)
    B ← Razred(b)
    return A = B ∨ Sosednja(A, B)
```

Postopek, opisan v algoritmu 1, ilustrira slika 1 na primeru dodajanja hiperpovezave $e = \{1, 2, 3, 4, 5, 8, 9\}$. Vozlišča hiperpovezave najprej razdelimo v množice M_i glede na pripadnost vozlišč hiperpovezave e obstoječim razredom. Tako v podanem primeru dobimo $M_A = \{1\}, M_B = \{2, 3, 4\}, M_C = \{5\}$ in $M_D = \{8, 9\}$. Razred M', ki vsebuje nova vozlišča, je prazen. Sedaj za vsako množico M_i, ki ne zajema celotnega razreda (v primeru sta to M_C in M_D), odcepimo del vozlišč iz razreda. Odcepljena dela razredov C' in D' podedujeta tudi povezave, ki so sosednje razredoma C in D, kar je označeno z rdečimi črtami. Elemente množice razredov $X = \{A, B, C', D'\}$, ki jih sedaj v celoti zajema hiperpovezava e, povežemo v H med seboj z modrimi povezavami v kliko. Pri tem naj opozorimo, da so nekatere povezave iz P obravnavane večkrat (na sliki so označene s črtkano črto). Povezava (A, B) je obstajala že pred dodatkom nove hiperpovezave, povezava (C', D') pa je bila dodana pri odcepitvi dveh povezanih razredov.

Sosednost dveh vozlišč $u, v \in V$ določimo tako, da ugotovimo, kateremu razredu v H pripada vsako od njiju. Če gre za isti razred, ali pa sta ta dva razreda povezana v H, sta vozlišči u in v sosednji.

Ocenimo časovno zahtevnost gradnje $H(T, P)$. Z $n_H(l)$ označimo število razredov $|T_l|$ po l obravnavanih hiperpovezavah. Velja $n_H(0) = 0$, za $l \geq 1$ pa $n_H(l) \leq 2n_H(l-1) + 1$ in zato $n_H(l) < 2^l$. Podobno definirajmo $m_H(l) < 2^{2l}$ kot število povezav med razredi, $|P_l|$. Naj bo $f(l)$ število operacij, ki jih zahteva dodatek l-te hiperpovezave. Za vsako od m_l vozlišč l-te hiperpovezave moramo ugotoviti, kateri skupini pripada. Poleg tega je morda treba pri odcepitvi dela skupine podvojiti vse obstoječe povezave ($2^{2(l-1)}$) in na koncu obravnavati (povezati) vse pare skupin, ki so del hiperpovezave ($n_H(l)^2$). Za število operacij torej velja $f(l) \leq m_l + 2^{2(l-1)} + 2^{2l}$ in $\sum_{l=1}^{m} f(l) = O(M + 2^{2m})$, kjer je M vsota velikosti hiperpovezav ($M = \sum_{l=1}^{m} m_l$).

Časovna zahtevnost predlaganega algoritma je eksponentna v odvisnosti od števila hiperpovezav, kar je učinkovito in smiselno samo za majhne vrednosti m v primerjavi z naivno metodo s časovno zahtevnostjo $O(n^2 m)$. Učinkovitost opisanega pristopa pri manjšem številu hiperpovezav pa lahko kljub temu izkoristimo tako, da hiperpovezave razdelimo v $\frac{m}{k}$ skupin velikosti k in za vsako skupino posebej zgradimo graf razredov, kar zahteva $O(M + \frac{m}{k} 2^{2k})$ časa. Za odgovor na poizvedbo o sosednosti dveh vozlišč moramo preveriti vseh $\frac{m}{k}$ skupin, v posamezni skupini pa lahko s pomočjo grafa razredov izračunamo odgovor na poizvedbo v konstantnem času. Algoritem 2 povzema opis procesa gradnje struktur in odgovarjanja na poizvedbe.

Algoritem 2: Izračun sosednosti vozlišč v hipergrafih

```
function Predobdelava(V, E, k)
    n, m ← |V|, |E|
    for i = 1 to m/k do               ▷ razdeli v skupine
        S_i = {E_{1+(i−1)k}, ..., E_{ik}}
    for i = 1 to m/k do               ▷ zgradi grafe razredov
        H_i ← GrafRazredov()
        for all e ∈ S_i do
            H_i.Dodaj(e)

function Poizvedba(a, b)
    for i = 1 to m/k do               ▷ poizveduj v vseh skupinah
        if H_i.Poizvedba(a, b) then
            return Da
    return Ne
```

Izbira velikosti skupine, k, je kompromis med hitrostjo gradnje grafov razredov in hitrostjo poizvedb. Večji

ko je k, hitrejše so poizvedbe, ker je treba obravnavati manj skupin hiperpovezav. Po drugi strani pa večji k močno vpliva na učinkovitost gradnje grafov razredov. Da je čas gradnje polinomski (in ne eksponenten), mora biti k logaritmičen. Primer take izbire je $k = \varepsilon \frac{1}{2} \log m$, ki vodi do naslednjega izreka.

Izrek 1: *Na poizvedbe o skupni povezavi para vozlišč v hipergrafu z n vozlišči in m hiperpovezavami z vsoto velikosti M lahko z $O(M + \frac{m^{1+\varepsilon}}{\log m})$ predprocesiranja odgovarjamo v času $O(\frac{m}{\log m})$.*

Če želimo zgraditi celotno matriko sosednosti, torej odgovoriti na vseh n^2 mogočih poizvedb, morata biti čas predprocesiranja in čas, namenjen poizvedbam, uravnotežena. Optimalna izbira za k je v tem primeru $k = \log n$.

Iskanje skupnih hiperpovezav je povezano s problemom iskanja prič produkta AA^T, kjer je A matrika logičnih vrednosti. Predstavljeni algoritem lahko uporabimo za izračun poljubnega produkta AB dveh matrik logičnih vrednosti, in sicer tako, da iz njiju sestavimo pomožno matriko C (glej enačbo 1). Produkt matrike C s transponirano vrednostjo pa v rezultatu vsebuje iskano vrednost AB kot podmatriko. Izbira $k = \log n$ v izreku 1) nas vodi do naslednje posledice:

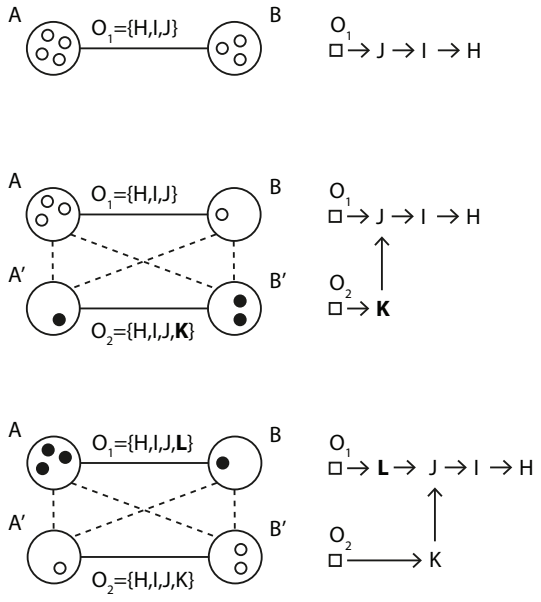
$$C = \begin{bmatrix} A \\ B^T \end{bmatrix}, \qquad C^T = \begin{bmatrix} A^T & B \end{bmatrix}, \qquad CC^T = \begin{bmatrix} AA^T & AB \\ B^T A^T & B^T B \end{bmatrix}. \tag{1}$$

Posledica 1: *Priče produkta matrik logičnih vrednosti A in B dimenzije $n \times n$ lahko izračunamo v času $O(\frac{n^3}{\log n})$.*

4 RAZŠIRITVE

Opisana podatkovna struktura se osredotoča zgolj na obstoj skupne hiperpovezave, lahko pa bi jo preprosto prilagodili tudi za iskanje primera hiperpovezave ali priče, ki je lahko poljubna ali pa največja. Za to bi morali v grafu razredov H hraniti oznake na vozliščih in povezavah, ki bi nam povedale, katera hiperpovezava je bila vzrok za vzpostavitev razreda ali povezave med razredoma. Če vozlišči a in b pripadata razredoma A in B, odgovorimo na poizvedbo o njuni skupni hiperpovezavi, če je $A = B$, kar z oznako njunega razreda, sicer pa z oznako povezave med A in B v grafu razredov.

Podatkovna struktura je dovolj splošna, da dovoljuje tudi poročanje seznama vseh hiperpovezav, ki so skupne dvema vozliščema v G. Sprememba je podobna kot v prejšnjem primeru, le da namesto posamezne oznake na vozliščih in povezavah v grafu razredov, hranimo sezname oznak. Sedaj je v primeru odcepitve (dela) razreda treba podvojiti (kopirati) celoten seznam oznak. Pri tem uporabimo tehniko *COW* (angl. *copy on write*), ki dejansko kopiranje izvede šele, če je prišlo v podatkih, ki jih kopiramo, do spremembe. Ker pa nikoli ne pride več do pisanja (spreminjanja) v seznamu, ki ga želimo kopirati, lahko vzdržujemo seznam v času $O(1)$ na kopiranje.



Slika 2: Primer podvajanja seznama oznak brez kopiranja

Slika 2 prikazuje opisani postopek na primeru kopiranja oznake O_1 na povezavi med razredoma A in B. Zgornja slika prikazuje začetno stanje, kjer so v seznamu O_1 shranjene hiperpovezave H, I in J. Če dodamo hiperpovezavo K, ki vključuje črno pobarvana vozlišča, moramo odcepiti razreda A' in B', kar je prikazano na srednji sliki. Tako med novimi povezavami, ki so nakazane s črtkanimi črtami, nastane tudi povezava med razredoma A' in B' z oznako $O_2 = O_1 \cup \{K\}$. Sedaj lahko brez škode dodamo tudi nove hiperpovezave v O_1, kar je ilustrirano na spodnji sliki. Nova hiperpovezava L (označena s črnimi vozlišči), ki zajema celotna razreda A in B, doda oznako L v seznam O_1. Vanj jo lahko vključimo brez vpliva na O_2, ki se sicer sklicuje na O_1.

Ker smo množico hiperpovezav E razdelili na $\frac{m}{k}$ skupin velikosti $k = \log m$, na poizvedbo odgovorimo tako, da združimo $\frac{m}{k}$ disjunktnih seznamov, ki jih dobimo s poizvedbami v vsakem od grafov razredov. To zahteva $O(\frac{m}{\log m} + occ)$ časa, pri čemer je occ dolžina iskanega seznama skupnih hiperpovezav.

5 SKLEP

S preprostim algoritmom smo demonstrirali, kako lahko pri ugotavljanju sosednosti vozlišč v hipergrafu pridemo do pohitritve za logaritemski faktor. Pristop, ki temelji na skupinah ekvivalentnih vozlišč, je dovolj splošen, da dovoljuje tudi številne prilagoditve. Lahko z njim morda dosežemo še večje pohitritve? Čeprav število skupin v pomožnem grafu narašča eksponentno, pa nikoli ne preseže n, ker nobena skupina ne more biti prazna.

Taka visoka segmentacija se običajno pojavi skupaj s številnimi povezavami med skupinami. Morda bi lahko z dobro strategijo združevanja razredov grafu razredov hevristično ohranjali manjše število skupin in povezav. Razreda sta ekvivalentna in ju lahko združimo, če sta povezana in imata enako množico sosedov. Zaznavanje ekvivalentnih skupin bi lahko izvajali učinkovito npr. z uporabo prstnih odtisov množice sosedov. Kljub temu pa ni očitno, kakšen bi bil vpliv vmesnega združevanja na časovno zahtevnost algoritma.

ZAHVALA

Raziskavo je omogočila Javna agencija za raziskovalno dejavnost Republike Slovenije (ARRS) v okviru programov *P2-0209* in *P2-0359* ter projekta *N2-0053*.

Poleg tega sta raziskavo omogočila še *Canada Research Chairs Programme* in *Natural Science and Engineering Council of Canada*.

LITERATURA

[1] S. Klamt, U.-U. Haus, and F. Theis, "Hypergraphs and Cellular Networks", *PLoS Computational Biology*, vol. 5, no. 5, p. e1000385, may 2009.

[2] Jun Yu, Dacheng Tao, and Meng Wang, "Adaptive Hypergraph Learning and its Application in Image Classification", *IEEE Transactions on Image Processing*, vol. 21, no. 7, pp. 3262–3272, jul 2012.

[3] D. Zhou, J. Huang, and B. Schölkopf, "Learning with Hypergraphs: Clustering, Classification, and Embedding", *Advances in Neural Information Processing Systems 19*, vol. 19, pp. 1601–1608, 2007.

[4] E. Estrada and J. A. Rodríguez-Velázquez, "Subgraph centrality and clustering in complex hyper-networks", *Physica A: Statistical Mechanics and its Applications*, vol. 364, pp. 581–594, 2006.

[5] K.-I. Goh, M. E. Cusick, D. Valle, B. Childs, M. Vidal, and A.-L. Barabási, "The human disease network", *Proceedings of the National Academy of Sciences of the United States of America*, vol. 104, no. 21, pp. 8685–8690, 2007.

[6] K. A. Zweig and M. Kaufmann, *A systematic approach to the one-mode projection of bipartite graphs*, 2011, vol. 1, no. 3.

[7] A. Bretto, *Hypergraph Theory*, ser. Mathematical Engineering. Heidelberg: Springer International Publishing, 2013.

[8] A. Brodnik and J. I. Munro, "Membership in constant time and minimum space", in *Lecture Notes in Computer Science*. Springer-Verlag, 1994, pp. 72–81.

[9] A. Brodnik and J. I. Munro, "Membership in constant time and almost-minimum space", *SIAM Journal on Computing*, 1999.

[10] F. Le Gall, "Powers of tensors and fast matrix multiplication", in *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation - ISSAC '14*. New York, New York, USA: ACM Press, 2014, pp. 296–303.

[11] H. Yu, "An Improved Combinatorial Algorithm for Boolean Matrix Multiplication", in *International Colloquium on Automata, Languages, and Programming*, 2015, pp. 1094–1105.

[12] R. Seidel, "On the All-Pairs-Shortest-Path Problem in Unweighted Undirected Graphs", *Journal of Computer and System Sciences*, vol. 51, no. 3, pp. 400–403, dec 1995.

[13] M. Kowaluk and A. Lingas, "LCA Queries in Directed Acyclic Graphs", in *Proceedings of the 32nd international conference on Automata, Languages and Programming*, 2005, pp. 241–248.

[14] Z. Galil and O. Margalit, "Witnesses for Boolean Matrix Multiplication and for Transitive Closure", *Journal of Complexity*, vol. 9, no. 2, pp. 201–221, jun 1993.

[15] A. Czumaj, M. Kowaluk, and A. Lingas, "Faster algorithms for finding lowest common ancestors in directed acyclic graphs", *Theoretical Computer Science*, vol. 380, no. 1-2, pp. 37–46, jun 2007.

[16] M. Habib, C. Paul, and L. Viennot, "Partition Refinement Techniques: An Interesting Algorithmic Tool Kit", *International Journal of Foundations of Computer Science*, vol. 10, no. 02, pp. 147–170, jun 1999.

Tomaž Hočevar je diplomiral in leta 2018 doktoriral s področja računalništva in informatike na Univerzi v Ljubljani. Na Fakulteti za računalništvo in informatiko je zaposlen kot asistent. Njegovo raziskovalno področje leži na preseku analize omrežij in algoritmov ter podatkovnih struktur.

Andrej Brodnik je redni profesor računalništva in informatike na Univerzi na Primorskem in predava tudi na Univerzi v Ljubljani. Njegovo osnovno raziskovalno področje so podatkovne strukture in algoritmi. Posebno pozornost namenja vplivom spoznanj teoretičnih raziskav na praktične rešitve. Objavlja še s področja kombinatorične optimizacije, še posebej v povezavi z bioinformatiko. Veliko pozornosti namenja razvoju poučevanja računalništva v osnovnih in srednjih šolah.

J. Ian Munro je redni profesor in *Canada Research Chair* za načrtovanje algoritmov na Univerzi v Waterlooju, kjer deluje že od leta 1971. Poleg tega je član osrednjega svetovnega združenja za računalništvo in informatiko *Association for Computing Machinery* in *Royal Society of Canada* ter prejemnik nagrade za življenjsko delo *CS-Can*.