Psevdokanonična predstavitev grafov na podlagi sosednosti

Luka Fürst

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko, Večna pot 113, SI-1000 Ljubljana
E-pošta: luka.fuerst@fri.uni-lj.si

Neighborhood-based Pseudo-canonical Representation of Graphs

A canonical representation of a graph is a string that is invariant to the numbering of the graph vertices. Consequently, two graphs are isomorphic if and only if they have equal canonical representations. The problem of computing a canonical representation is thus at least as hard as the graph isomorphism problem. In this paper, we present a polynomial-time algorithm for finding a pseudo-canonical graph representation with the property that a pair of equal representations implies isomorphism but the opposite is not necessarily true. Nevertheless, for a given graph the algorithm iteratively builds representations that are increasingly more likely to be numbering-invariant. To validate our approach, we applied the proposed algorithm to the complete set of connected simple undirected graphs with up to 9 vertices and to a set of real-world graphs.

1 Uvod

Takšna in drugačna omrežja igrajo v današnjem svetu čedalje pomembnejšo vlogo. Za učinkovito preučevanje omrežij pa potrebujemo učinkovite grafne algoritme. V številnih postopkih za delo z grafi oz. omrežji nam pride prav *kanonični zapis* [1]. Gre za zapis, ki enolično določa strukturo grafa, od oštevilčenja vozlišč pa je povsem neodvisen; če številke vozlišč med seboj premešamo, se kanonični zapis ne sme spremeniti. Grafa imata enak kanonični zapis natanko tedaj, ko sta *izomorfna*, torej ko imata enako strukturo.

Eden od temeljnih problemov na področju odkrivanja vzorcev v grafih (angl. graph data mining) [2] je problem naštevanja neizomorfnih podgrafov v danem gostiteljskem grafu [3]. Pri reševanju tega problema nam kanonični zapisi koristijo takrat, ko preverjamo, ali smo izomorfno kopijo pravkar odkritega podgrafa že odkrili: namesto da bi iskali izomorfizme, med seboj preprosto primerjamo kanonične zapise. Kanonični zapisi se uporabljajo tudi na področju kemije, saj omogočajo učinkovito iskanje spojin in funkcionalnih skupin glede na njihovo strukturo [4].

Ker se preverjanje izomorfizma prevede na preverjanje enakosti kanoničnih zapisov, je problem določitve kanoničnega zapisa vsaj tako težak kot problem grafnega izomorfizma. Ta problem pa je GI-poln, zato ne vemo, ali zanj obstaja algoritem s polinomsko časovno zahtevnostjo.[1] Najboljši znani algoritem teče v kvazipolinomskem času [5].

V nekaterih aplikacijah lahko zahteve glede kanoničnega zapisa nekoliko omilimo in lahko, denimo, izomorfna grafa »občasno« proglasimo za neizomorfna [6]. V takih primerih zadošča *psevdokanonični* zapis, ki ga bomo opredelili s sledečo zahtevo:

> Če sta psevdokanonična zapisa grafov med seboj enaka, sta grafa izomorfna.

Seveda je zaželeno, da lastnost čim večkrat drži tudi v obratni smeri, vendar pa tega formalno ne zahtevamo.

V članku predstavljamo algoritem za tvorbo psevdokanoničnega zapisa grafov. Algoritem se izvaja v polinomskem času in postopoma gradi čedalje boljše predstavitve; čim dlje ga pustimo teči, tem večja je verjetnost, da bo zapis, ki ga bo zgradil za podani graf, dejansko kanoničen (*popolnoma* neobčutljiv na oštevilčenje vozlišč), čeprav, kot bomo videli, obstajajo grafi, pri katerih algoritem ne more zgraditi takšnega zapisa.[2] Algoritem smo preizkusili na popolnem naboru povezanih neusmerjenih enostavnih grafov z največ 9 vozlišči in na nekaterih grafih iz realnega sveta. Po nam dosegljivih podatkih lahko algoritem in njegovo teoretično in praktično analizo obravnavamo kot prispevek k znanosti.

Psevdokanonično predstavitev gradimo na podlagi sosednosti vozlišč, načeloma pa bi lahko algoritem zasnovali na poljubni kombinaciji *grafnih invariant* — lastnosti, ki so odvisne samo od strukture grafa, ne pa od konkretnega oštevilčenja vozlišč.

V razdelku 2 bomo opredelili pojme, ki jih bomo potrebovali skozi celotni prispevek, v razdelku 3 pa bomo predstavili algoritem za izdelavo psevdokanoničnega zapisa podanega grafa. V razdelku 4 bomo delovanje metode eksperimentalno potrdili, z razdelkom 5 pa bomo prispevek zaključili.

2 Terminologija in notacija

Graf G je dvojica (V_G, E_G), kjer je V_G množica *vozlišč*, $E_G \subseteq V_G \times V_G$ pa množica *povezav*. Namesto

[1] Kratica GI se nanaša na problem grafnega izomorfizma (angl. graph isomorphism problem).

[2] Če razreda problemov GI in P nista istovetna, potem taki grafi obstajajo pri *poljubnem* polinomskem algoritmu.

V_G in E_G bomo pisali V in E, kadar bo jasno, kateremu grafu pripadata množici, ali pa ko to ne bo pomembno. Brez pretirane izgube splošnosti se bomo omejili na neusmerjene grafe brez zank: $(v, v) \notin E$, $(u, v) \in E \iff (v, u) \in E$. Prav tako bomo predpostavili, da velja $|V| = n$ in $V = \{1, 2, \ldots, n\}$. Rekli bomo, da je vozlišče u *manjše* (oz. *večje*) od vozlišča v, če velja $u < v$ (oz. $u > v$). Množico vseh sosedov vozlišča u bomo označili z $\mathcal{N}(u) = \{v \in V \mid (u, v) \in E\}$.

Grafa G in H sta *izomorfna* (oznaka: $G \simeq H$), če obstaja bijektivna preslikava $h\colon V_G \to V_H$, ki ohranja sosednosti in nesosednosti, torej če za vsak par vozlišč $u, v \in V_G$ velja $(u, v) \in E_G \iff (h(u), h(v)) \in E_H$. Takšni preslikavi pravimo *izomorfizem*.

Niz $z^*(G)$ je *kanonični zapis* grafa G, če za vsak par grafov G in H velja $z^*(G) = z^*(H) \iff G \simeq H$. Niz $z(G)$ je *psevdokanonični zapis* grafa G, če za vsak par grafov G in H velja $z(G) = z(H) \implies G \simeq H$.

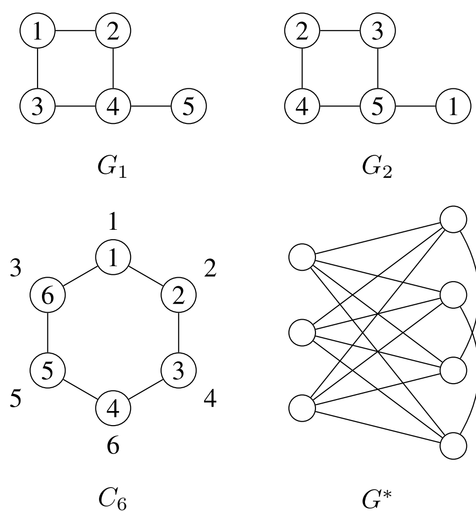
3 Algoritem

Za začetek pokažimo sledečo trditev:

Trditev 1. Niz $z(G) = S_1/S_2/\ldots/S_{n-1}$, kjer je $S_u = s_{u,u+1}s_{u,u+2}\ldots s_{u,n}$ za vsak $u \in \{1, \ldots, n-1\}$ in $s_{u,v} = 1$ (v primeru $(u, v) \in E_G$) oziroma 0 (v nasprotnem primeru) za vsak $v > u$, je psevdokanonični zapis grafa G.

Na primer, za grafa G_1 in G_2 na sliki 1 velja $z(G_1) = 1100/010/10/1$ in $z(G_2) = 0001/110/01/1$.

Dokaz. Če velja $z(G) = z(H)$, potem imata grafa G in H gotovo enako število vozlišč. Poleg tega za vsak par (u, v) z lastnostjo $u < v$ velja $(u, v) \in E_G \iff (u, v) \in E_H$. Ker sta grafa neusmerjena, velja ta lastnost za vse pare (u, v), od tod pa že sledi, da sta grafa izomorfna. □



Slika 1: Primeri grafov.

Žal pa opisana predstavitev sama po sebi ni nujno odporna na spremembe v oštevilčenju vozlišč. Grafa G_1 in G_2 na sliki 1 sta izomorfna, vendar pa, kot smo videli, nimata enakih psevdokanoničnih zapisov. Zato se nam ponuja sledeča ideja: vozliščem grafa dodelimo takšne številke, da bo zapis, izdelan po metodi iz trditve 1, enak pri čimveč različnih začetnih oštevilčenjih vozlišč. Problem (psevdo)kanonične predstavitve grafa tako prevedemo na problem (psevdo)kanoničnega oštevilčenja vozlišč grafa.

Pri iskanju (psevdo)kanoničnega oštevilčenja se moramo opreti na lastnosti vozlišč, ki so neodvisne od začetnega oštevilčenja. Algoritem, ki ga bomo predstavili, obravnava sosešččine vozlišč. Na primer, pri grafih G_1 in G_2 na sliki 1 ima eno vozlišče enega soseda, eno vozlišče tri sosede, tri vozlišča po dva soseda. Vozlišču z enim sosedom se splača dodeliti (denimo) številko 1, vozlišču s tremi sosedi (denimo) številko 5, ostalim vozliščem pa številke 2, 3 in 4. Številki 1 in 5 bosta tako vedno, ne glede na začetno oštevilčenje grafa, dodeljeni vozliščema z enim oziroma tremi sosedi. Če poleg neposrednih sosedov upoštevamo tudi sosede sosedov, sosede sosedov sosedov itd., pa dobimo še robustnejše oštevilčenje.

Postopek za določanje psevdokanoničnega oštevilčenja vozlišč je prikazan kot algoritem 1. Algoritem se prične s klicem funkcije IZVRSI na podanem grafu. Funkcija na začetku vsem vozliščem dodeli številko $id = 1$, nato pa v zanki tako dolgo kliče funkcijo NAPREJ in po potrebi še funkcijo RAZPLETI, dokler nima vsako vozlišče svoje unikatne številke.

Funkcija NAPREJ v k-ti iteraciji obravnava množico W, ki jo tvorijo vozlišča s številko k, a le v primeru, če velja $|W| \geq 2$. (Ko namreč neko vozlišče dobi unikatno številko, ga ne obravnavamo več; njegova številka postane dokončna.) Za vsako vozlišče w iz množice W se zgradi zaporedje številk vozlišč njegovih sosedov, nato se to zaporedje leksikografsko uredi (funkcija UREDI), da dobimo zaporedje $T(w)$, nazadnje pa se vozliščem v množici W dodelijo številke glede na položaje njihovih zaporedij $T(w)$ v leksikografsko urejenem zaporedju zaporedij $T(w_1), T(w_2), \ldots, T(w_{|W|})$. (Operator $\prec$ v vrstici 24 preberemo kot »je leksikografsko manjši od«.) Vozlišča z enakim zaporedjem dobijo isto številko, zato se bo z njimi še treba ukvarjati.

Če klic funkcije NAPREJ v ničemer ne spremeni oštevilčenja, moramo spremeniti vsaj eno neunikatno številko, saj bi se sicer ujeli v neskončno zanko. V funkciji RAZPLETI poiščemo množico vozlišč z najmanjšo neunikatno številko in vsem vozliščem iz te množice (razen najmanjšemu) povečamo številko za 1.

Tabela 1 prikazuje delovanje algoritma na primeru grafa C_6 s slike 1. (Na sliki podajajo številke v vozliščih začetno, zunanje številke pa dobljeno končno oštevilčenje.) Na začetku vsem vozliščem dodelimo številko 1. Ker imajo vsa vozlišča enako zaporedje T ($\langle 1, 1\rangle$), se številke vozlišč ne spremenijo, zato pokličemo funkcijo RAZPLETI. Vozlišče 1 dobi unikatno številko 1 (z njim se ne bomo več ukvarjali), ostala pa številko 2. V naslednjem klicu funkcije NAPREJ velja $T(2) = T(6) = \langle 1, 2\rangle$ in $T(3) = T(4) = T(5) = \langle 2, 2\rangle$, zato vozlišči 2 in 6 ohranita številko 2, vozlišča 3, 4 in 5 pa dobijo številko 4. Postopek nadaljujemo, dokler nima vsako vo-

Algoritem 1 Psevdokanonično oštevilčenje vozlišč.

1: **function** IZVRSI($G = (V, E)$)
2: **for all** $v \in V$ **do**
3: $id(v) := 1$
4: **end for**
5: **repeat**
6: $sprememba :=$ NAPREJ(G, id)
7: $konec := (\forall u, v \in V : id(u) \neq id(v))$
8: **if** $\neg konec \wedge \neg sprememba$ **then**
9: RAZPLETI(G, id)
10: **end if**
11: **until** $konec$
12: **return** id
13: **end function**
14: **function** NAPREJ($G = (V, E)$, id)
15: $id' := id$
16: **for all** $k \in \{1, 2, \ldots, |V|\}$ **do**
17: $W :=$ VRSTNIKI(V, id', k)
18: **if** $|W| > 1$ **then**
19: **for all** $w \in W$ **do**
20: $T_0 := \langle id'(w') \mid w' \in \mathcal{N}(w) \rangle$
21: $T(w) :=$ UREDI(T_0)
22: **end for**
23: **for all** $w \in W$ **do**
24: $P := \{w' \in W \mid T(w') \prec T(w)\}$
25: $id(w) := |P| + 1$
26: **end for**
27: **end if**
28: **end for**
29: **return** $(id \neq id')$
30: **end function**
31: **function** RAZPLETI($G = (V, E)$, id)
32: $k^* := \min\{k \mid |\text{VRSTNIKI}(V, id, k)| > 1\}$
33: $W :=$ VRSTNIKI(V, id, k^*)
34: **for all** $w \in W \setminus \{\min(W)\}$ **do**
35: $id(w) := id(w) + 1$
36: **end for**
37: **end function**
38: **function** VRSTNIKI(V, id, k)
39: **return** $\{v \in V \mid id(v) = k\}$
40: **end function**

Tabela 1: Delovanje algoritma na grafu C_6.

	1	2	3	4	5	6
id	1	1	1	1	1	1
T	$\langle 1,1 \rangle$	$\langle 1,1 \rangle$	$\langle 1,1 \rangle$	$\langle 1,1 \rangle$	$\langle 1,1 \rangle$	$\langle 1,1 \rangle$
id	1	1	1	1	1	1
id	**1**	2	2	2	2	2
T		$\langle 1,2 \rangle$	$\langle 2,2 \rangle$	$\langle 2,2 \rangle$	$\langle 2,2 \rangle$	$\langle 1,2 \rangle$
id		2	4	4	4	2
T		$\langle 1,4 \rangle$	$\langle 2,4 \rangle$	$\langle 4,4 \rangle$	$\langle 2,4 \rangle$	$\langle 1,4 \rangle$
id		2	4	**6**	4	2
id		**2**	4		4	**3**
T			$\langle 2,6 \rangle$		$\langle 3,6 \rangle$	
id			**4**		**5**	

zlišče svoje unikatne številke. Vidimo, kako algoritem na začetku upošteva samo neposredne sosede, nato pa spremembe številk vplivajo na vse bolj oddaljena vozlišča.

Časovno zahtevnost algoritma lahko določimo na podlagi opažanja, da v vsakem obhodu zanke vsaj eno vozlišče grafa pridobi svojo končno številko. Če funkcija NAPREJ vrne **false**, potem funkcija RAZPLETI nekemu vozlišču dodeli unikatno številko, unikatna številka pa je samodejno tudi dokončna. V nasprotnem primeru pa funkcija NAPREJ bodisi pridela vsaj eno unikatno številko ali pa poveča številke vsaj eni skupini vozlišč. Številke vozlišč v tej skupini sicer niso unikatne, toda vsaj eno vozlišče iz skupine bo ohranilo svojo številko do konca izvajanja algoritma. Od tod sledi, da imamo največ $(n-1)$ iteracij postopka, vsaka iteracija pa zahteva čas $O(n^2 \log n)$: za vsako od n vozlišč uredimo zaporedje številk vseh njegovih sosedov v času $O(n \log n)$, nato pa za celoten graf uredimo zaporedje zaporedij številk v času $O(n^2 \log n^2) = O(n^2 \log n)$. Skupna časovna zahtevnost algoritma potemtakem znaša $O(n^3 \log n)$.

4 Eksperimentalni rezultati

Psevdokanonično predstavitev lahko obravnavamo kot kanonično, če je povsem neobčutljiva na začetno oštevilčenje vozlišč. To lahko preverimo tako, da zgradimo psevdokanonični zapis za vsako od $n!$ možnih oštevilčenj. Če so vsi zapisi med seboj enaki, gre za pravo kanonično predstavitev.

Algoritem 1 smo izvedli na celotni množici povezanih enostavnih neusmerjenih grafov z največ 9 vozlišči. Tabela 2 za vsak $n \in \{2, \ldots, 9\}$ in $i \in \{1, \ldots, n-1\}$ prikazuje, pri koliko grafih z n vozlišči dobimo po i iteracijah zanke v funkciji IZVRSI zapis, ki ni povsem neobčutljiv na začetno oštevilčenje vozlišč. Graf G^* na sliki 1 je najmanjši graf (in edini tak graf s sedmimi vozlišči), čigar končni psevdokanonični zapis ni kanoničen. Kot vidimo, imajo vsa vozlišča grafa enako število sosedov. Takšni grafi povzročajo našemu algoritmu največ preglavic.

V drugi skupini poskusov smo algoritem preizkusili na nekaterih realnih grafih iz zbirk KONECT [7] in SNAP [8]. V fazi predobdelave smo v vseh grafih odstranili morebitne oznake vozlišč in povezav, usmerjene povezave spremenili v neusmerjene in odstranili morebitne večkratne povezave in zanke. Pri poskusih smo algoritem 1 primerjali z algoritmom za računanje pravih kanoničnih zapisov, ki je vgrajen v sistem Sage.[3] Algoritem 1 smo implementirali v programskem jeziku C, vse poskuse pa smo izvršili na računalniku z 8-jedrnim procesorjem Intel Core i7-3770 s taktom 3,40 GHz.

Rezultati izvajanja poskusov so zbrani v tabeli 3. Poleg porabe časa v milisekundah (t_{PK} za algoritem 1 in t_{K} za algoritem v sistemu Sage) podajamo tudi število iteracij (N) zanke v funkciji IZVRSI. V skladu s pričakovanji se psevdokanonični zapisi v skoraj vseh primerih izračunajo hitreje od kanoničnih. (Odgovora na vprašanje, zakaj je graf Advogato izjema, zaenkrat še nimamo.) Ker je število permutacij vozlišč preveliko, ne moremo zanesljivo trditi, ali so dobljeni psevdokanonični zapisi

[3] http://www.sagemath.org/

Tabela 2: Število nekanoničnih psevdokanoničnih zapisov za grafe v množici vseh povezanih enostavnih neusmerjenih grafov z n vozlišči ($\mathcal{G}_n$).

		i							
n	$\|\mathcal{G}_n\|$	1	2	3	4	5	6	7	8
2	1	0							
3	2	0	0						
4	6	2	1	0					
5	21	12	7	3	0				
6	112	88	52	27	7	0			
7	853	781	427	235	89	20	1		
8	11 117	10 873	6032	2780	1164	319	52	19	
9	262 180	260 105	145 392	44 952	16 295	4691	899	201	113

Tabela 3: Rezultati na realnih grafih.

Graf	$\|V\|$	$\|E\|$	t_{PK}	t_{K}	N
Les Misérables	77	254	1,8	8,8	28
David Copperfield	112	425	0,7	5,6	5
Jazz	198	2742	4,5	16,9	10
US Power Grid	4941	6594	32 700	58 700	481
Advogato	6541	39 285	230 000	124 000	2013
ca-HepTh	9877	25 973	615 000	1 450 000	2266
ca-CondMat	23 133	93 439	9 220 000	23 700 000	5999

odporni na spremembe v oštevilčenju vozlišč, vendar pa se to po preizkušanju z naključnimi permutacijami začetnega oštevilčenja zdi dokaj verjetno.

5 Zaključek

Predstavili smo polinomski algoritem za izdelavo psevdokanoničnega zapisa grafa in ga preizkusili na umetnih in realnih grafih. Vsaj za grafe z največ devet vozlišči lahko trdimo, da je zapis, ki ga proizvede algoritem, v veliki večini primerov dejansko kanoničen, torej neobčutljiv na oštevilčenje vozlišč.

Pri gradnji psevdokanoničnih zapisov bi si lahko pomagali s simetrijami. Na primer, pri grafu K_6 so pravzaprav vsa oštevilčenja kanonična: ne glede na to, kako vozliščem dodelimo številke, bo dobljeni zapis vedno enak. Pri grafu C_6 imamo manj svobode; izkaže se, da lahko med seboj permutiramo le številke vozlišč 1, 3 in 5. Množice vozlišč z opisano lastnostjo tvorijo *preiskovalno ekvivalentno particijo* grafa [9, 10]. V prihodnosti bi bilo potemtakem smiselno raziskati, kako preiskovalna ekvivalenca vpliva na psevdokanonični zapis in ali jo lahko izkoristimo za dosego večje učinkovitosti algoritma ali robustnejših psevdokanoničnih predstavitev.

Literatura

[1] L. Babai, E. M. Luks, “Canonical labeling of graphs,” *Symposium on Theory of Computing (STOC) 1983, Boston, Massachusetts, ZDA*, str. 171–183, 1983.

[2] D. Cook, L. Holder, *Mining Graph Data*. John Wiley & Sons, 2006.

[3] M. Kuramochi, G. Karypis, “Finding frequent patterns in a large sparse graph,” *Data Mining and Knowledge Discovery*, vol. 11, št. 3, str. 243–271, 2005.

[4] D. Weininger, “SMILES, a chemical language and information system. 1. introduction to methodology and encoding rules,” *Journal of Chemical Information and Computer Sciences*, vol. 28, št. 1, str. 31–36, 1988.

[5] L. Babai, “Graph isomorphism in quasipolynomial time [extended abstract],” *Symposium on Theory of Computing (STOC) 2016, Cambridge, Massachussetts, ZDA*, str. 684–697, 2016.

[6] L. Fürst, M. Mernik, V. Mahnič, “Graph grammar induction as a parser-controlled heuristic search process,” *Applications of Graph Transformations with Industrial Relevance (AGTIVE) 2011, Budimpešta, Madžarska*, LNCS vol. 7233, str. 121–136, 2012.

[7] J. Kunegis, “KONECT: the Koblenz network collection,” *International Web Observatory Workshop (WWW) 2013, Rio de Janeiro, Brazilija*, str. 1343–1350, 2013. `http://konect.uni-koblenz.de/`

[8] J. Leskovec, A. Krevl, “SNAP Datasets: Stanford large network dataset collection.” `http://snap.stanford.edu/data`, 2014.

[9] J. Mihelič, L. Fürst, U. Čibej, “Exploratory equivalence in graphs: definition and algorithms,” in *Federated Conference on Computer Science and Information Systems (FedCSIS) 2014, Varšava, Poljska*, str. 447–456, 2014.

[10] L. Fürst, U. Čibej, J. Mihelič, “Maximum exploratory equivalence in trees,” in *Federated Conference on Computer Science and Information Systems (FedCSIS) 2015, Łódź, Poljska*, str. 507–518, 2015.