ELEKTROTEHNIŠKI VESTNIK 85(4): 162–168, 2018
IZVIRNI ZNANSTVENI ČLANEK

Iskanje vzorčnih grafov s pomočjo iskalnega načrta ob prisotnosti avtomorfizmov

Luka Fürst, Uroš Čibej, Jurij Mihelič

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

Povzetek. Problem podgrafnega izomorfizma, ki se ukvarja z iskanjem pojavitev podanega vzorčnega grafa v podanem gostiteljskem grafu, postaja zaradi vseprisotnosti velikih omrežij čedalje pomembnejši. Problem je NP-poln, na učinkovitost iskanja pa lahko negativno vplivajo tudi simetrije v vzorčnem grafu. V članku predstavljamo algoritem za reševanje problema podgrafnega izomorfizma s pomočjo iskalnega načrta, seznama navodil za sistematičen prebor vzorčnega grafa. Predstavljeni algoritem upošteva morebitne simetrije vzorčnega grafa in tako deluje učinkoviteje kot premočrtna različica, ki zgolj na vse mogoče načine izpolnjuje navodila iskalnega načrta. S preizkušanjem algoritma na umetnih in realnih grafih smo empirično potrdili njegovo prednost pred naivnim pristopom in odgovorili na več raziskovalnih vprašanj.

Ključne besede: graf, podgrafni izomorfizem, avtomorfizem, simetrija, iskalni načrt, preiskovalna ekvivalenca

Searching for pattern graphs using a search plan in the presence of automorphisms

The subgraph isomorphism problem, the goal of which is to find the occurrences of a given pattern graph in a given host graph, is, owing to the pervasiveness of large networks, becoming increasingly important. However, the problem is NP-complete, and the search efficiency may also be negatively affected by symmetries in the pattern graph. In this paper, we present an algorithm for solving the subgraph isomorphism problem using a search plan, a sequence of instructions for a systematic traversal of the pattern graph. The presented algorithm pays attention to the symmetries in the pattern graph and thus performs more efficiently than its straightforward counterpart which merely follows the search plan instructions in all possible ways. By testing our algorithm on artificial and real-world graphs, we empirically confirm its advantage over the naive approach and answer several research questions.

1 UVOD

Veliki grafi igrajo v današnjem svetu čedalje pomembnejšo vlogo. Socialna omrežja z milijoni ali celo milijardami vozlišč (uporabnikov) in še precej več povezavami (»prijateljstvi«) so nemara najbolj razvpit primer velikih grafov, seveda pa še zdaleč niso edini. Pomislimo na transportna in energetska omrežja, omrežja e-poštnih sporočil, omrežja citiranja znanstvenih objav, lingvistična omrežja itd. [1], [2], [3]

Zaradi obsega in pomembnosti velikih omrežij je nujno, da jih znamo učinkovito preiskovati. Pogosto nas zanima prisotnost določenih podgrafov v podanem omrežju. Na primer, štetje trojic vzajemnih prijateljstev v socialnem omrežju lahko prevedemo na štetje pojavitev 3-ciklov v grafu omrežja. Žal pa je *problem podgrafnega izomorfizma* (»ugotovi, ali podani gostiteljski graf H vsebuje pojavitev podanega vzorčnega grafa G«) NP-poln [4], pripadajoči preštevalni problem (»poišči vse pojavitve grafa G v grafu H«) pa $\#P$-poln [5]. To pomeni, da najverjetneje ne obstaja algoritem, ki bi tovrstne probleme reševal v polinomskem času. Tudi najučinkovitejši znani algoritmi v najslabšem primeru potrebujejo eksponentno veliko časa v odvisnosti od velikosti vhoda, četudi se lahko v tipičnih realnih primerih kar dobro obnesejo [6], [7], [8].

Problem podgrafnega izomorfizma je mogoče reševati na različne načine [9]. Med bolj znanimi je Ullmannov pristop [10], ki s pomočjo sestopanja gradi in popravlja dvojiško matriko, ki predstavlja vse mogoče monomorfizme* med vzorčnim in gostiteljskim grafom. V našem članku pa bomo primerke vzorčnih grafov iskali s pomočjo *iskalnega načrta* [11]. Iskalni načrt je zaporedje navodil za sistematični prebor vozlišč in povezav vzorčnega grafa, zato ga lahko uporabimo za iskanje njegovih pojavitev v poljubnem gostiteljskem grafu. Prvo navodilo v zaporedju obišče eno od vozlišč vzorčnega grafa, vsako od nadaljnjih pa obišče bodisi še ne obiskanega soseda že obiskanega vozlišča (in obenem tudi povezavo med vozliščema) bodisi še ne obiskano povezavo med že obiskanima vozliščema. Algoritma za iskanje pojavitev grafov s pomočjo iskalnega načrta ni težko zasnovati, saj moramo zgolj po vrsti slediti iskalnemu načrtu in sestopiti, ko ne moremo več naprej ali pa ko želimo poiskati naslednjo pojavitev vzorčnega grafa.

Poseben izziv pri reševanju problema podgrafnega

Prejet 12. junij, 2018
Odobren 23. avgust, 2018

*Monomorfizem je injektivna preslikava med množico elementov vzorčnega grafa in množico elementov gostiteljskega grafa, ki določa eno od pojavitev vzorčnega grafa v gostiteljskem.

izomorfizma so simetrije v obliki *avtomorfizmov* vzorčnega grafa. Če ima vzorčni graf k avtomorfizmov, bo iskalni postopek brez dodatnih ukrepov vsako njegovo pojavitev v gostiteljskem grafu odkril po k-krat. Tudi če gostiteljski graf ne vsebuje nobene pojavitve, bo iskalni algoritem zaradi avtomorfizmov odkril več kandidatnih delnih monomorfizmov, kot bi jih dejansko moral. Obstaja več pristopov za rokovanje s simetrijami [12], [13], [14], v tem članku pa si bomo pomagali s *preiskovalno ekvivalenco* vozlišč vzorčnega grafa [15], [16]. Če ima vzorčni graf simetrije, potem lahko na podlagi preiskovalne ekvivalence postopek iskanja izboljšamo tako, da ta tvori manj odvečnih delnih in popolnih monomorfizmov med vzorčnim in gostiteljskim grafom in s tem porabi manj časa. Kot bomo videli, je na podlagi particije množice vozlišč grafa, ki jo določa preiskovalna ekvivalenca, mogoče neposredno definirati nabor omejitev, ki zmanjša redundanco iskalnega postopka.

V pričujočem članku bomo pokazali, kako lahko algoritem za reševanje problema podgrafnega izomorfizma, ki temelji na iskalnem načrtu, izboljšamo tako, da upošteva preiskovalno ekvivalenco vzorčnega grafa. Poleg tega bomo predstavili rezultate eksperimentov, ki smo jih izvedli na umetnih in realnih grafih. Oboje lahko obravnavamo kot izvirni prispevek k znanosti.

V razdelku 2 bomo definirali pojme, ki jih bomo uporabili pri opisu problema in iskalnega algoritma. V razdelku 3 bomo natančno formulirali problem. V razdelku 4 bomo predstavili naivni in izboljšani algoritem za reševanje problema podgrafnega izomorfizma s pomočjo iskalnega načrta. V razdelku 5 bomo rezultate eksperimentov prikazali in o njih razpravljali, z razdelkom 6 pa bomo članek zaključili.

2 OZADJE

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*. Kadar bo jasno (ali pa nepomembno), kateremu grafu pripada množica vozlišč oziroma povezav, bomo namesto V_G in E_G pisali kar V in E. V članku se bomo brez pretirane izgube splošnosti omejili na neusmerjene grafe brez zank: $E \subseteq \{uv \mid u < v\}$.* Prav tako bomo predpostavili, da velja $V = \{1, 2, \ldots, |V|\}$.

Graf H je *podgraf* grafa G ($H \sqsubseteq G$), če velja $V_H \subseteq V_G$ in $E_H \subseteq E_G$. *Homomorfizem* med grafoma G in H je funkcija $h\colon G \to H$, ki ohranja sosednosti: $\forall (u, v) \in E_G\colon h(u)\,h(v) \in E_H$. Injektivni homomorfizem se imenuje *monomorfizem* ali *podgrafni izomorfizem*. Če je $h\colon G \to H$ monomorfizem, potem je graf $h(G) \sqsubseteq H$ *pojavitev* grafa G v grafu H. Pojavitvi grafa G v grafu H, določeni z monomorfizmoma $h_1\colon G \to H$ in $h_2\colon G \to H$, obravnavamo kot istovetni (kot eno in isto pojavitev), če pokrivata isto množico vozlišč in povezav grafa H, torej če velja $\{h_1(v) \mid v \in V_G\} = \{h_2(v) \mid v \in V_G\}$ in $\{h_1(e) \mid e \in E_G\} = \{h_2(e) \mid e \in E_G\}$.

Bijektivni homomorfizem se imenuje *izomorfizem*. Grafa G in H sta *izomorfna*, če obstaja izomorfizem $h\colon G \to H$. Izomorfizem v istem grafu ($h\colon G \to G$) se imenuje *avtomorfizem*. Množico vseh avtomorfizmov grafa G označimo z $\mathrm{Aut}(G)$.

Monomorfizem $\{1 \mapsto v_1,\ 2 \mapsto v_2,\ \ldots,\ |V| \mapsto v_{|V|}\}$ bomo zapisali kar kot $v_1 v_2 \ldots v_{|V|}$. Graf C_4 (4-cikel), prikazan na sliki 2, ima 8 avtomorfizmov: $\mathrm{Aut}(C_4) = \{1234,\ 2341,\ 3412,\ 4123,\ 4321,\ 3214,\ 2143,\ 1432\}$. Eden od številnih monomorfizmov med grafom C_4 in grafom na sliki 3 je $\{1 \mapsto 7,\ 2 \mapsto 11,\ 3 \mapsto 10,\ 4 \mapsto 6\}$ oziroma 7 11 10 6.

Iskalni načrt [11], [17] je zaporedje navodil za prebor celotnega vzorčnega grafa G, ki nam omogoča iskanje pojavitev grafa v poljubnem gostiteljskem grafu H. Prvo navodilo je vedno oblike $[u]$; razumemo ga kot »obišči poljubno vozlišče w v grafu H in ga priredi vozlišču u v grafu G (tj. vzpostavi $h(u) = w$)«. Vsako od nadaljnjih navodil pa ima bodisi obliko $[u \to v]$ (»prični v vozlišču, prirejenem vozlišču u, obišči enega od njegovih še ne obiskanih sosedov in izbranega soseda priredi vozlišču v) bodisi obliko $[u ? v]$ (»preveri, ali sta vozlišči grafa H, prirejeni vozliščema u in v, povezani«). Navodilo oblike $[u \to v]$ obišče novo vozlišče in povezavo, navodilo oblike $[u\,?\,v]$ pa samo novo povezavo. Na primer, eden od iskalnih načrtov za graf C_4 se glasi $\langle [1],\ [1 \to 2],\ [2 \to 3],\ [3 \to 4],\ [1\ ?\ 4] \rangle$.

Če v gostiteljskem grafu izpolnimo iskalni načrt za podani vzorčni graf na vse mogoče načine (tj. v vsakem koraku preizkusimo vse veljavne možnosti), bomo zanesljivo našli vse pojavitve vzorčnega grafa v gostiteljskem. V tem smislu so vsi iskalni načrti za podani graf med seboj enakovredni. Kot bomo videli, pa niso nujno enakovredni po učinkovitosti iskanja.

Množica $A \subseteq \mathrm{Aut}(G)$ *pokriva* množico $P = \{v_1,\ v_2,\ \ldots,\ v_k\} \subseteq V$ (oznaka: $\mathrm{cover}(A, P)$), če za vsako permutacijo $\sigma\colon P \to P$ obstaja avtomorfizem $a \in A$, tako da velja $a(v_1) = \sigma(v_1)$, $a(v_2) = \sigma(v_2)$, $\ldots$, $a(v_k) = \sigma(v_k)$. Na primer, množica $\mathrm{Aut}(C_4)$ pokriva množico $\{1, 3\}$, saj avtomorfizem **123**4 zajema permutacijo $\{1 \mapsto 1,\ 3 \mapsto 3\}$, avtomorfizem **321**4 pa permutacijo $\{1 \mapsto 3,\ 3 \mapsto 1\}$. *Stabilizator* množice $A \subseteq \mathrm{Aut}(G)$ glede na množico $P \subseteq V$ je množica vseh avtomorfizmov v A, ki fiksirajo vse elemente množice P: $\mathrm{Stab}(A, P) = \{a \in A \mid \forall v \in P\colon a(v) = v\}$. Na primer, $\mathrm{Stab}(\mathrm{Aut}(C_4),\ \{1, 3\}) = \{\mathbf{12}3\mathbf{4},\ 1\mathbf{43}2\}$.

Zaporedje $\mathcal{P} = \langle P_1, P_2, \ldots, P_s \rangle$ je *urejena particija* množice V, če velja $\bigcup_{i=1}^{s} P_i = V$ in $P_i \cap P_j = \emptyset$ za vse $i \neq j$. Urejena particija $\langle P_1, P_2, \ldots, P_s \rangle$ je *preiskovalno ekvivalentna*, če za vse $i \in \{1, \ldots, s\}$ velja $\mathrm{cover}(A_{i-1}, P_i)$ in $A_i = \mathrm{Stab}(A_{i-1}, P_i)$, pri čemer je $A_0 = \mathrm{Aut}(G)$. Na primer, pri grafu C_4 je urejena particija $\langle \{1, 3\},\ \{2, 4\} \rangle$ preiskovalno ekvivalentna. To velja tudi za particijo $\langle \{1, 2\},\ \{3\},\ \{4\} \rangle$, ne pa za particijo

*Vrstni red krajišč povezave ni pomemben: zapisa uv in vu oba pomenita povezavo med vozliščema u in v.

$\langle\{1,2\},\ \{3,4\}\rangle$, saj množica $\mathrm{Stab}(\mathrm{Aut}(G),\{1,2\}) = \{1234\}$ ne pokriva množice $\{3,4\}$.

Množica $\mathcal{P} = \{P_1, P_2, \ldots, P_s\}$ je *particija* množice V, če velja $\bigcup_{i=1}^{s} P_i = V$ in $P_i \cap P_j = \emptyset$ za vse $i \neq j$. Particija $\{P_1, P_2, \ldots, P_s\}$ je *preiskovalno ekvivalentna*, če obstaja permutacija $\sigma\colon \{1,\ldots,s\} \to \{1,\ldots,s\}$, tako da je urejena particija $\langle P_{\sigma(1)}, P_{\sigma(2)}, \ldots, P_{\sigma(s)}\rangle$ preiskovalno ekvivalentna.

Kot bomo pokazali v razdelku 4, lahko pri reševanju problema podgrafnega izomorfizma na podlagi preiskovalno ekvivalentne particije $\{P_1, P_2, \ldots, P_s\}$ vpeljemo omejitve, ki število odkritih monomorfizmov med vzorčnim in gostiteljskim grafom zmanjšajo za faktor $F = |P_1|!\,|P_2|!\,\ldots\,|P_s|!$, zato med različnimi kandidatnimi preiskovalno ekvivalentnimi particijami izberemo *optimalno* — tisto, pri kateri je faktor (*ocena*) F največji. Pri grafu C_4 je optimalna particija $\{\{1,3\},\ \{2,4\}\}$, pri grafu C_6 (slika 2) pa particija $\{\{1,3,5\},\ \{2\},\ \{4\},\ \{6\}\}$ z oceno $3!\,1!\,1!\,1! = 6$; alternativa $\{\{1,3\},\ \{2,5\},\ \{4\},\ \{6\}\}$ je z oceno $2!\,2!\,1!\,1! = 4$ slabša. Oceno optimalne preiskovalno ekvivalentne particije grafa G bomo označili s $F^*(G)$.

3 OPREDELITEV PROBLEMA

Vhod v problem, ki ga rešujemo v članku, sestavljajo

- vzorčni graf G,
- gostiteljski graf H,
- iskalni načrt za graf G in
- preiskovalno ekvivalentna particija grafa G,

pričakovani izhod pa je seznam vseh pojavitev grafa G v grafu H, pri čemer ne zahtevamo, da se vsaka pojavitev izpiše samo po enkrat.

4 NAŠ PRISTOP

Najprej pokažimo naslednjo trditev in njeno posledico:

Trditev 1. Naj bo $\langle P_1, P_2, \ldots, P_s\rangle$ preiskovalno ekvivalentna urejena particija množice V, pri čemer je $P_i = \{v_{i,1}, v_{i,2}, \ldots, v_{i,k_i}\}$ in $v_{i,1} < v_{i,2} < \ldots < v_{i,k_i}$ za vse $i \in \{1,\ldots,s\}$. Naj bo H_G pojavitev grafa G v grafu H. Potem obstaja izomorfizem $h\colon G \to H_G$, za katerega pri vseh $i \in \{1,\ldots,s\}$ velja $h(v_{i,1}) < h(v_{i,2}) < \ldots < h(v_{i,k_i})$.

Dokaz: Naj bo $h_0\colon G \to H_G$ poljuben izomorfizem in naj bodo $\sigma_1^*\colon P_1 \to P_1$, …, $\sigma_s^*\colon P_s \to P_s$ takšne permutacije, da za vse $i \in \{1,\ldots,s\}$ velja $h_0(\sigma_i^*(v_{i,1})) < h_0(\sigma_i^*(v_{i,2})) < \ldots < h_0(\sigma_i^*(v_{i,k_i}))$. Naj bo permutacija $\sigma^*\colon V \to V$ definirana kot $\sigma^*(v_{i,j}) = \sigma_i^*(v_{i,j})$ za vse $i \in \{1,\ldots,s\}$ in $j \in \{1,\ldots,k_i\}$.

Naj bo σ_1 permutacija množice P_1, σ_2 permutacija množice P_2 itd. in naj $A(\sigma_1,\sigma_2,\ldots,\sigma_r)$ označuje množico avtomorfizmov $a \in \mathrm{Aut}(G)$ z lastnostjo $a(v_{i,j}) = \sigma_i(v_{i,j})$ za vse $i \in \{1,\ldots,r\}$ in $j \in \{1,\ldots,k_i\}$. Ker po definiciji preiskovalne ekvivalence množica $\mathrm{Aut}(G)$ pokriva množico P_1, je množica $A(\sigma_1)$ neprazna za pojubno permutacijo σ_1 množice P_1. Ker množica $\mathrm{Stab}(\mathrm{Aut}(G), P_1)$ pokriva množico P_2, je neprazna tudi množica $A(\sigma_1,\sigma_2)$, in to za poljubni par permutacij σ_1 in σ_2. Če razmislek nadaljujemo, ugotovimo, da za poljubno množico permutacij $\sigma_1\colon P_1 \to P_1$, …, $\sigma_s\colon P_s \to P_s$ množica $A(\sigma_1,\sigma_2,\ldots,\sigma_s)$ vsebuje vsaj en avtomorfizem. Vsak avtomorfizem iz množice $A(\sigma_1^*,\sigma_2^*,\ldots,\sigma_s^*)$ premeša vozlišča v vsaki posamezni množici $P_1, \ldots, P_s$ tako, da $h = h_0 \circ \sigma^*$ postane izomorfizem z iskano lastnostjo: po definiciji iz prejšnjega odstavka velja namreč $h(v_{i,1}) < h(v_{i,2}) < \ldots < h(v_{i,k_i})$ za vsak $i \in \{1,\ldots,s\}$. ■

Posledica 2. Pri uporabi preiskovalno ekvivalentne particije $\{P_1, P_2, \ldots, P_s\}$ je število izomorfizmov $h\colon G \to H_G$, ki izpolnjujejo pogoje iz trditve 1, enako $|\,\mathrm{Aut}(G)|\ /\ F^*(G)$.

Dokaz: Število vseh izomorfizmov $G \to H_G$ je enako številu avtomorfizmov na grafu G. Vendar pa v vsaki množici $|P_i|!$ izomorfizmov, ki v grafu H_G zajamejo vseh $|P_i|!$ permutacij slike množice P_i, le eden izpolnjuje pogoj $h(v_{i,1}) < h(v_{i,2}) < \ldots < h(v_{i,k_i})$. Ker to velja za vse $i \in \{1,\ldots,s\}$, se število izomorfizmov zmanjša za faktor $F^*(G) = |P_1|!\,|P_2|!\,\ldots\,|P_s|!$. ■

Iz trditve 1 neposredno sledi, da se lahko pri iskanju pojavitev vzorčnega grafa G v gostiteljskem grafu H omejimo samo na tiste (delne in popolne) monomorfizme $G \to H$, za katere veljajo navedene omejitve. Z drugimi besedami: če vozlišči u in v z lastnostjo $u < v$ pripadata istemu ekvivalenčnemu razredu v preiskovalno ekvivalentni particiji grafa G, potem ohranimo (in razvijamo) zgolj tiste delne in popolne monomorfizme $h\colon G \to H$, za katere velja $h(u) < h(v)$, vse druge pa zavržemo.

Algoritem za iskanje pojavitev s pomočjo iskalnega načrta je prikazan kot algoritem 1. Postopek se sproži s klicem procedure IZVRSI, ki sprejme vzorčni graf G, iskalni načrt zanj (zaporedje navodil $\langle D_1, D_2, \ldots, D_r\rangle$, pri čemer je navodilo D_1 vedno oblike $[v_0]$), preiskovalno ekvivalentno particijo ($\mathcal{P}$) grafa G in gostiteljski graf H. Algoritem po korakih — z zaporednim izvrševanjem navodil iskalnega načrta — gradi monomorfizem $h\colon G \to H$. Navodilo $[v_0]$ seveda izpolnjujejo vsa vozlišča grafa H. Navodilo $[v_1 \to v_2]$ pa algoritem izpolni tako, da izbere enega od sosedov w_2 vozlišča $h(v_1)$ v grafu H, ki v trenutnem poskusu iskanja pojavitve grafa G še ni bil obiskan in ki izpolnjuje pogoj

$$\forall v \in h^{-1}(V_H)\colon (\exists P \in \mathcal{P}\colon \{v, v_2\} \subseteq P) \implies (v < v_2 \iff h(v) < w_2), \quad (1)$$

pri čemer $h^{-1}(V_H)$ označuje množico vseh vozlišč $v \in V_G$, za katera po trenutnem delnem monomorfizmu h že obstaja slika $h(v) \in V_H$. Ko algoritem izpolni vsa navodila iskalnega načrta in tako zgradi popoln monomorfizem, izpiše pojavitev, ki jo določa monomorfizem, nato pa sestopi, da poišče še druge monomorfizme.

Sestopimo tudi tedaj, ko nobeno vozlišče grafa H ne izpolnjuje trenutnega navodila oziroma pogoja (1).

Postopek poleg delnega monomorfizma $h\colon G \to H$ vzdržuje tudi množico *Visited*, ki vsebuje vsa vozlišča grafa H, ki smo jih pri gradnji tekočega monomorfizma že obiskali. Preslikava $h\colon G \to H$ je, kot vemo, monomorfizem samo tedaj, ko se različna vozlišča grafa G preslikajo v različna vozlišča grafa H.

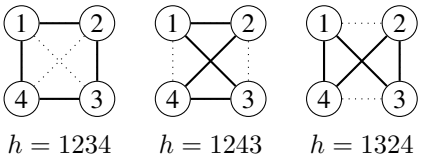
Množica $\mathrm{Neighbors}(u) = \{v \in V \mid uv \in E\}$ vsebuje vse sosede vozlišča u. Spremenljivka i v pomožni proceduri IZVRSIPOM podaja zaporedno številko trenutnega navodila iskalnega načrta.

V razdelku 5 bomo Algoritem 1 primerjali z njegovo naivno različico. Ta se od Algoritma 1 razlikuje le po funkciji PREVERIPE:

function PREVERIPE(G, $\mathcal{P}$, h, v_2, w_2)
  return true
end function

Naivni algoritem preiskovalne ekvivalence sploh ne upošteva in zgolj premočrtno na vse mogoče načine izpolnjuje navodila iskalnega načrta.

Oglejmo si razliko med delovanjem naivnega in izboljšanega algoritma na primeru iskanja pojavitev grafa C_4 v grafu K_4 (slika 2). Vse tri pojavitve so prikazane na sliki 1. Graf C_4 ima 8 avtomorfizmov ($|\operatorname{Aut}(C_4)|$), ocena njegove optimalne preiskovalno ekvivalentne particije ($F^*(C_4)$) pa znaša 4. Zato naivni algoritem vsako pojavitev grafa C_4 odkrije po 8-krat, izboljšani pa le po dvakrat ($= |\operatorname{Aut}(C_4)|/F^*(C_4)$). Na primer, za pojavitev, določeno z monomorfizmom 1324, bi naivni algoritem tvoril monomorfizme 1324, 3241, 2413, 4132, 4231, 2314, 3142 in 1423, izboljšani pa le monomorfizma 1324 in 3142, saj sta edina, ki izpolnjujeta zahtevi $h(1) < h(3)$ in $h(2) < h(4)$.



Slika 1: Vse pojavitve grafa C_4 v grafu K_4

5 EKSPERIMENTALNI REZULTATI IN RAZPRAVA

S pomočjo eksperimentov smo poskušali empirično odgovoriti na naslednja raziskovalna vprašanja:

- Kako se po učinkovitosti razlikujeta algoritem, ki upošteva preiskovalno ekvivalenco vzorčnega grafa, in algoritem, ki zgolj premočrtno na vse mogoče načine izpolnjuje navodila iskalnega načrta?
- Kako na delovanje algoritma vpliva ocena preiskovalno ekvivalentne particije in kako velikost vzorčnega grafa?
- Ali izbira drugačnega iskalnega načrta za isti vzorčni graf vpliva na učinkovitost algoritma?

Večino poskusov smo izvedli z vzorčnimi grafi, prikazanimi na sliki 2. Barve vozlišč označujejo posamezne ekvivalenčne razrede v optimalni preiskovalno ekvivalentni particiji. Vozlišča iste barve pripadajo istemu ekvivalenčnemu razredu, neobarvana vozlišča pa vsak svojemu. Na primer, optimalna preiskovalno ekvivalentna particija grafa C_6 je $\{\{1,3,5\}, \{2\}, \{4\}, \{6\}\}$. Števili ob oznaki grafa (npr. $(8\ /\ 4)$ pri grafu C_4) podajata število avtomorfizmov ($|\operatorname{Aut}(G)|$) in oceno optimalne preiskovalno ekvivalentne particije ($F^*(G)$). Kot smo že povedali, bo naivni algoritem vsako pojavitev grafa odkril po $|\operatorname{Aut}(G)|$-krat, izboljšani pa po $(|\operatorname{Aut}(G)|\ /\ F^*(G))$-krat.

V nadaljevanju bomo z N označevali število pojavitev podanega vzorčnega grafa G v podanem gostiteljskem grafu H, s t_0 oziroma t_+ pa čas v milisekundah, ki ga za podani par grafov G in H potrebuje naivni oziroma izboljšani algoritem. Algoritma smo implementirali v programskem jeziku C, poganjali pa na računalniku s 3,40-gigaherčnim 8-jedrnim procesorjem Intel Core i7-3770.

V prvem nizu poskusov smo Algoritem 1 in njegovo naivno različico preizkušali na umetnih gostiteljskih grafih, in sicer na grafih M_n z $(n+1)^2$ vozlišči in $2n(n+1)$ povezavami ($E = \{v,\ v+1 \mid v \not\equiv 0 \pmod{(n+1)}\} \cup \{(v,\ v+n+1) \mid v \le n(n+1)\}$) in na grafih K_n z n vozlišči in $n(n-1)\,/\,2$ povezavami ($E = \{uv \mid u < v\}$). Graf M_3 je prikazan na sliki 3.

Tabela 1 prikazuje rezultate izvajanja obeh algoritmov na vseh parih vzorčnih grafov in gostiteljskih grafov M_{100} in K_{15}. Kot bi lahko pričakovali, razmerje med časom delovanja naivnega in izboljšanega algoritma narašča z naraščajočo oceno optimalne preiskovalno ekvivalentne particije. Razlika je še posebej izrazita pri vzorčnih grafih K_n. Na primer, pri grafu K_9 se število odkritih pojavitev z uporabo preiskovalne ekvivalence zmanjša za faktor $F^*(K_9) = 9! = 362\,880$, poraba časa pa za faktor, večji od $50\,000$. Tudi v primerih, ko graf H ne vsebuje nobene pojavitve grafa G, je izboljšani algoritem hitrejši od naivnega, saj uporaba preiskovalne ekvivalence omeji tudi število vzpostavljenih delnih, ne samo popolnih monomorfizmov.

Iz tabele 1 razberemo tudi to, da se razmerje med izvajalnim časom naivnega in izboljšanega algoritma pri večjih vzorčnih grafih približuje oceni optimalne preiskovalno ekvivalentne particije za vzorčni graf. S povečevanjem vzorčnega grafa se namreč zmanjšuje časovni delež programske kode, ki je neodvisna od velikosti vzorčnega grafa in je skupna obema algoritmoma. Postopno približevanje ciljnemu razmerju pride še bolj do izraza, če algoritma poženemo na družini grafov z različno velikostjo, a enako oceno optimalne preiskovalno ekvivalentne particije. Tabela 2 prikazuje rezultate iskanja grafa L_n v grafu M_{100} za $n \in \{2, \ldots, 14\}$. Vidimo, da se razmerje v porabi časa približuje vrednosti

Algoritem 1 Iskanje pojavitev grafa s pomočjo iskalnega načrta in preiskovalne ekvivalence

```
1: function IZVRSI(G, ⟨[v_0], D_2, ..., D_r⟩, P, H)
2:     for all v ∈ V_G do
3:         h(v) := ⊥                          ▷ to pomeni, da v (še) nima svoje slike v grafu H
4:     end for
5:     for all w_0 ∈ V_H do
6:         h(v_0) := w_0                      ▷ izpolnimo prvo navodilo iskalnega načrta
7:         Visited := {w_0}
8:         IZVRSIPOM(G, ⟨D_2, ..., D_r⟩, 2, P, H, h, Visited)
9:     end for
10: end function
11:
12: function IZVRSIPOM(G, ⟨D_2, ..., D_r⟩, i, P, H, h, Visited)
13:     if i = r then
14:         izpiši h(G)                        ▷ izpolnili smo vsa navodila iskalnega načrta
15:     else
16:         if D_i = [v_1 → v_2] then
17:             for all w_2 ∈ Neighbors(h(v_1)) do   ▷ navodilo izpolnimo na vse mogoče načine
18:                 if w_2 ∉ Visited ∧ PREVERIPE(G, P, h, v_2, w_2) then
19:                     h(v_2) := w_2
20:                     Visited := Visited ∪ {w_2}
21:                     IZVRSIPOM(G, ⟨D_2, ..., D_r⟩, i + 1, P, H, h, Visited)
22:                     Visited := Visited \ {w_2}
23:                     h(v_2) := ⊥
24:                 end if
25:             end for
26:         else if D_i = [v_1 ? v_2] then
27:             if h(v_1) h(v_2) ∈ E_H then     ▷ preverimo obstoj povezave med že obiskanima vozliščema
28:                 IZVRSIPOM(G, ⟨D_2, ..., D_r⟩, i + 1, P, H, h, Visited)
29:             end if
30:         end if
31:     end if
32: end function
33:
34: function PREVERIPE(G, P, h, v_2, w_2)
35:     for all v ∈ V_G do          ▷ preverimo, ali lahko v monomorfizem h dodamo preslikavo v_2 ↦ w_2
36:         if h(v) ≠ ⊥ ∧ ∃P ∈ P: {v, v_2} ⊆ P then
37:             if (v < v_2 ∧ h(v) > w_2) ∨ (v > v_2 ∧ h(v) < w_2) then
38:                 return false
39:             end if
40:         end if
41:     end for
42:     return true
43: end function
```

$F^*(L_n) = 2$.

Poskusi so pokazali, da iskalni načrt praviloma nima skoraj nikakršnega vpliva na delovanje algoritma, kljub temu pa neugodni primeri obstajajo. Vsi dosedanji poskusi so bili izvedeni z iskalnimi načrti, ki vozlišča vzorčnih grafov obiskujejo po vrsti od vozlišča 1 do vozlišča $|V|$. Pojavitve grafa C_{11}, denimo, odkrivamo z iskalnim načrtom $\mathcal{N}_1 = \langle [1], [1 \to 2], [2 \to 3], \ldots, [9 \to 10], [10 \to 11], [11\ ?\ 1] \rangle$. V kombinaciji z optimalno preiskovalno ekvivalentno particijo $\{\{1, 2\}, \{3\}, \{4\}, \ldots, \{11\}\}$ je iskalni načrt $\mathcal{N}_1$ ugoden, saj moramo že pri izpolnjevanju drugega navodila upoštevati omejitev $h(1) < h(2)$. To pomeni, da že zgodaj v postopku iskanja pojavitve zatremo nekatere odvečne delne monomorfizme. Iskalni načrt $\mathcal{N}_2 = \langle [2], [2 \to 3], [3 \to 4], \ldots, [9 \to 10], [10 \to 11], [11 \to 1], [1\ ?\ 2] \rangle$ pa je v obravnavem primeru izrazito neugoden, saj omejitev $h(1) < h(2)$ učinkuje šele pri izpolnjevanju predzadnjega navodila. Preden bo algoritem neperspektivni monomorfizem zavrgel, ga bo zgradil skoraj v celoti.

Tabela 1: Rezultati na umetnih gostiteljskih grafih

H	G	N	t_0	t_+
M_{100}	L_4	178 596	31,0	19,5
	C_4	10 000	20,3	14,0
	K_4	0	7,1	6,1
	L_6	1 387 104	244	136
	C_6	19 800	127	58
	K_6	0	7,0	6,4
	L_9	28 273 662	5430	2920
	C_9	0	2380	977
	K_9	0	7,1	6,9
K_{15}	L_4	16 380	5,4	2,9
	C_4	4095	7,2	3,0
	K_4	1365	10,1	0,75
	L_6	1 801 800	274	143
	C_6	300 300	361	61
	K_6	5005	660	2,54
	L_9	908 107 200	159 000	84 100
	C_9	100 900 800	202 000	34 200
	K_9	5005	527 000	10,3

Tabela 2: Iskanje pojavitev grafa L_n v grafu M_{100} za različne vrednosti n

G	N	t_0	t_+	$t_0 \,/\, t_+$
L_2	20 200	5,78	4,98	1,16
L_3	59 998	12,1	9,84	1,23
L_4	178 596	31,5	20,0	1,58
L_5	492 006	86,9	53,2	1,63
L_6	1 387 104	249	139	1,79
L_7	3 780 626	700	395	1,77
L_8	10 455 084	1970	1080	1,82
L_9	28 273 662	5430	2980	1,82
L_{10}	77 233 024	15 400	8230	1,87
L_{11}	207 943 998	42 400	22 700	1,87
L_{12}	563 572 700	118 000	63 000	1,87
L_{13}	1 512 373 042	334 000	176 000	1,90
L_{14}	4 077 286 312	919 000	483 000	1,90

Kot prikazujejo rezultati za iskanje pojavitev grafa C_{11} v grafu K_{11} (tabela 3), je razlika med iskalnima načrtoma $\mathcal{N}_1$ in $\mathcal{N}_2$ očitna. (Graf K_{11} vsebuje 1 814 400 pojavitev grafa C_{11}.)

Tabela 3: Iskanje pojavitev grafa C_{11} v grafu K_{11} s pomočjo ugodnega in neugodnega iskalnega načrta

Iskalni načrt	t_0	t_+
$\mathcal{N}_1$	8340	4210
$\mathcal{N}_2$	8460	6960

Nazadnje predstavljamo še rezultate izvajanja obeh algoritmov na gostiteljskih grafih, ki izvirajo iz realnega sveta. Algoritma smo preizkusili na šestih grafih iz baz KONECT* (grafi Les Misérables, US Power Grid, David Copperfield in Jazz Musicians) [1] in SNAP† (grafa ca-HepTh in ca-CondMat) [2]. Grafom smo odstranili morebitne oznake, zanke in večkratne povezave med istim parom vozlišč, morebitne usmerjene povezave pa smo spremenili v neusmerjene. Nato smo v vsakem od grafov z naivnim in izboljšanim algoritmom našteli vse pojavitve grafov L_4, C_4 in K_4 in dobili rezultate, zbrane v tabeli 4. Tudi tukaj vidimo, da izboljšani algoritem v vseh primerih prekaša naivnega, razmerje v porabi časa pa je marsikje blizu $|F^*(G)|$. Ni presenetljivo, da na porabo časa bolj kot sama velikost gostiteljskega grafa vpliva razmerje med številom povezav in vozlišč. Večje povprečno število povezav na vozlišče na splošno pomeni tudi več pojavitev vzorčnega grafa.

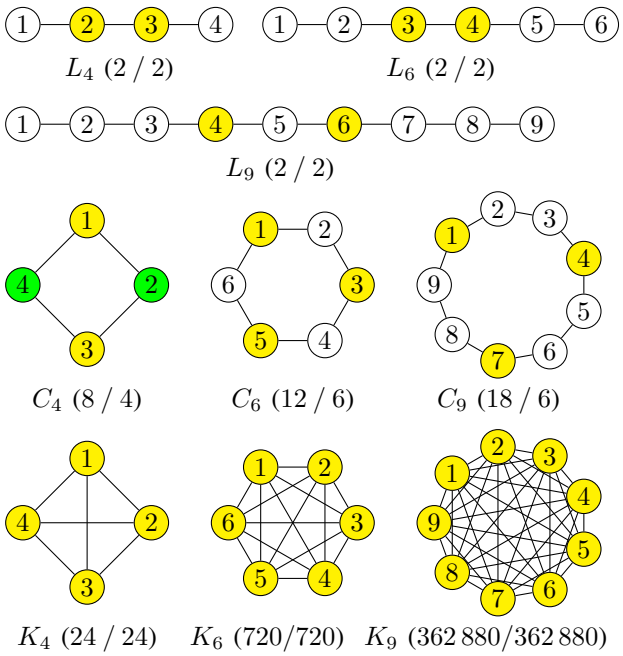
Tabela 4: Rezultati na realnih gostiteljskih grafih

H	G	N	t_0	t_+
Les Misérables	L_4	26 784	5,9	2,0
$\lvert V\rvert = 77$	C_4	2672	8,9	3,7
$\lvert E\rvert = 254$	K_4	639	7,6	0,8
US Power Grid	L_4	52 556	13	8,5
$\lvert V\rvert = 4941$	C_4	979	12	4,5
$\lvert E\rvert = 6594$	K_4	90	6,6	2,4
David Copperfield	L_4	61 254	15	4,5
$\lvert V\rvert = 112$	C_4	2579	9,4	2,9
$\lvert E\rvert = 425$	K_4	58	5,8	0,9
Jazz Musicians	L_4	3 850 915	480	260
$\lvert V\rvert = 198$	C_4	406 441	870	250
$\lvert E\rvert = 2742$	K_4	78 442	670	43
ca-HepTh	L_4	4 207 311	550	300
$\lvert V\rvert = 9877$	C_4	239 081	630	200
$\lvert E\rvert = 25\,973$	K_4	65 592	360	36
ca-CondMat	L_4	50 543 325	6300	3500
$\lvert V\rvert = 23\,133$	C_4	1 505 383	9900	2600
$\lvert E\rvert = 93\,439$	K_4	294 008	3700	270

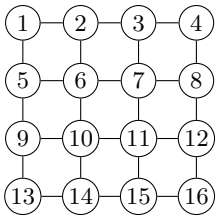
6 SKLEP

Predstavili smo algoritem za reševanje problema podgrafnega izomorfizma s pomočjo iskalnega načrta, ki upošteva avtomorfizme vzorčnega grafa in zato deluje učinkoviteje kot premočrtni iskalni algoritem. Algoritem s pomočjo poznavanja preiskovalno ekvivalentne particije vzorčnega grafa omeji množico obravnavanih monomorfizmov med vzorčnim in gostiteljskim grafom. Algoritem smo preizkusili na umetnih in realnih gostiteljskih grafih. Proučili smo vpliv ocene optimalne

*http://konect.uni-koblenz.de/
†http://snap.stanford.edu/data/



Slika 2: Grafi, ki smo jih uporabljali v večini poskusov



Slika 3: Graf M_3

preiskovalne ekvivalence, velikosti vzorčnega grafa in zaporedja navodil v iskalnem načrtu.

Preiskovalna ekvivalenca ne vodi nujno do optimalnega nabora omejitev. Na primer, pri 4-ciklu dobimo na podlagi optimalne preiskovalno ekvivalentne particije nabor omejitev $\{h(1) < h(3),\ h(2) < h(4)\}$, toda še večje prihranke nam ponuja nabor $\{h(1) < h(2) < h(4)\}$. Tega nabora ne moremo izpeljati iz preiskovalne ekvivalence, kljub temu pa se lahko prepričamo, da z njegovo uporabo ne izpustimo nobene pojavitve 4-cikla, ne glede na oštevilčenje vozlišč v gostiteljskem grafu. Ena od zamisli za nadaljnje delo je torej iskanje tovrstnih omejitev, zanimiva pa bi bila tudi študija vpliva omejitev na različne iskalne algoritme.

LITERATURA

[1] J. Kunegis, "KONECT: the Koblenz network collection," *Proceedings of the International Web Observatory Workshop, Rio de Janeiro, Brazilija, maj 2013*, pp. 1343–1350, 2013.

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

[3] U. Čibej, "Graditev in analiza grafov slovenskih besed," *Elektrotehniški vestnik*, vol. 80, no. 4, pp. 141–146, 2013.

[4] S. A. Cook, "The complexity of theorem-proving procedures," *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing (STOC), Shaker Heights, Ohio, ZDA, maj 1971*, pp. 151–158, 1971.

[5] S. Arora, B. Barak, *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[6] L. P. Cordella, P. Foggia, C. Sansone, M. Vent, "A (sub)graph isomorphism algorithm for matching large graphs," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 10, pp. 1367–1372, 2004.

[7] Tomaž Hočevar, Janez Demšar, "A combinatorial approach to graphlet counting," *Bioinformatics*, vol. 30, no. 4, pp. 559–565, 2014.

[8] U. Čibej, J. Mihelič, "Improvements to Ullmann's algorithm for the subgraph isomorphism problem," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 29, no. 7, 2015.

[9] J. Lee, W.-S. Han, R. Kasperovics, J.-H. Lee, "An in-depth comparison of subgraph isomorphism algorithms in graph databases," *Proceedings of the VLDB Endowment*, vol. 6, no. 2, pp. 133–144, 2012.

[10] J. R. Ullmann, "An Algorithm for Subgraph Isomorphism," *Journal of the ACM*, vol. 23, pp. 31–42, 1976.

[11] J. Rekers, A. Schürr, "Defining and parsing visual languages with Layered Graph Grammars," *Journal of Visual Languages and Computing*, vol. 8, no. 1, pp. 27–55, 1997.

[12] M. O. Albertson, K. L. Collins, "Symmetry breaking in graphs," *The Electronic Journal of Combinatorics*, vol. 3, no. 1, 1996.

[13] M. G. Everett, S. P. Borgatti, "Computing regular equivalence: Practical and theoretical issues," *Metodološki zvezki*, vol. 17, 2002.

[14] I. P. Gent, K. E. Petrie, J.-F. Puget, "Symmetry in constraint programming," *Handbook of Constraint Programming*, vol. 10, pp. 329–376, 2006.

[15] J. Mihelič, L. Fürst, U. Čibej, "Exploratory equivalence in graphs: definition and algorithms," *Proceedings of the 2014 Federated Conference on Computer Science and Information Systems, FedCSIS 2014, Varšava, Poljska, september 2014*, pp. 447–456, 2014.

[16] L. Fürst, U. Čibej, J. Mihelič, "Maximum exploratory equivalence in trees," *Proceedings of the 2015 Federated Conference on Computer Science and Information Systems, FedCSIS 2015, Łódź, Poljska, september 2015*, pp. 507–518, 2015.

[17] L. Fürst, M. Mernik, V. Mahnič. Improving the graph grammar parser of Rekers and Schürr. *IET Software*, 5(2):246–261, 2011.

Luka Fürst je leta 2013 na Univerzi v Ljubljani doktoriral s področja računalništva in informatike. Zaposlen je kot asistent na Fakulteti za računalništvo in informatiko. Raziskovalno se ukvarja s sintaksno analizo in indukcijo grafnih gramatik, v novejšem času pa tudi z grafnimi algoritmi nasploh.

Uroš Čibej je leta 2007 doktoriral na Fakulteti za računalništvo in informatiko Univerze v Ljubljani iz podvajanja podatkov v porazdeljenih sistemih. Trenutno je asistent in raziskovalec na omenjeni fakulteti. Raziskovalno se ukvarja s porazdeljenimi sistemi, snovanjem in analizo algoritmov za kombinatorično optimizacijo (npr. za iskanje vzorcev v grafih), programskimi jeziki in simulacijami.

Jurij Mihelič deluje kot docent v okviru Laboratorija za Algoritmiko Fakultete za računalništvo in informatiko Univerze v Ljubljani, kjer je tudi leta 2006 doktoriral iz računalniških znanosti. Njegova raziskovalna področja zajemajo inženiring in optimizacijo algoritmov, kombinatorično optimizacijo ter sistemsko programsko opremo in virtualizacijo.