Mine 'Em All: A Note on Mining All Graphs

Ondřej Kuželka, Jan Ramon

Mine 'Em All: A Note on Mining All Graphs

Ondřej Kuželka[1] and Jan Ramon[2,3]

[1] School of Computer Science & Informatics, Cardiff University, UK
KuzelkaO@cardiff.ac.uk
[2] Department of Computer Science, KU Leuven, Belgium
[3] INRIA, Lille, France
jan.ramon@cs.kuleuven.be

Abstract. We study the complexity of the problem of enumerating all graphs with frequency at least 1 and computing their support. We show that there are hereditary classes of graphs for which the complexity of this problem depends on the order in which the graphs should be enumerated (e.g. from frequent to infrequent or from small to large). For instance, the problem can be solved with polynomial delay for databases of planar graphs when the enumerated graphs should be output from large to small but it cannot be solved even in incremental-polynomial time when the enumerated graphs should be output from most frequent to least frequent (unless P=NP).

1 Introduction

In this paper we study graph mining problems from a nontraditional perspective. We are inspired by the question which properties of the problem make some graph mining problems solvable in incremental polynomial time or with polynomial delay. Here, we do not require the discovered graph patterns to be frequent and we want to output all patterns occurring in at least one database graph. However, we still want to also output their occurrences. In addition, we constrain the order in which the patterns should be printed, e.g. from most frequent patterns to least frequent patterns, which allows us to connect our results to results on (in)frequent graph mining. Surprisingly, for several graph classes, we show that different orders lead to very different computational complexities. For instance mining planar graphs cannot be done *in incremental-polynomial time* when the output graphs should be ordered by frequency but it can be done with *polynomial delay* when they should be ordered from largest to smallest.

2 Preliminaries

In this section we first briefly review some basic concepts and fix the notations used in this paper. We start with some standard definitions from graph theory.

Graphs. An *undirected graph* is a pair (V, E), where V is a finite set of *vertices* and $E \subseteq \{e \subseteq V : |e| = 2\}$ is a set of *edges*. Two vertices are said to be *adjacent* (in G) if they are connected by an edge (of the graph G). A *labeled undirected graph* is a triple (V, E, λ), where (V, E) is an undirected graph and $\lambda : V \cup E \to \Sigma$ is a function assigning a label from an alphabet Σ to every element of $V \cup E$. We will denote the set of vertices, the set of edges, and the labeling function of a graph G by $V(G)$, $E(G)$, and λ_G, respectively. We define $|G| = |V(G)| + |E(G)|$ and call this the order of G. Note that in graph theory, often other notions of 'order' are used, measuring only the number of edges or the number of vertices. A graph G' is a *subgraph* of a graph G, if $V(G') \subseteq V(G)$, $E(G') \subseteq E(G)$, and $\lambda_{G'}(x) = \lambda_G(x)$ for every $x \in V(G') \cup E(G')$; G' is an *induced subgraph* of G if it is a subgraph of G satisfying $\{u, v\} \in E(G')$ if and only if $\{u, v\} \in E(G)$ for every $u, v \in V(G')$. For a subset $S \subseteq V(G)$, $G[S]$ denotes the (unique) induced subgraph of G with vertex set S. A *contraction* of an edge $e = \{u, v\}$ in a graph G is an operation which produces a new graph by replacing u and v in $V(G)$ as well as in all $\{x, y\} \in E(G)$ by a new vertex w (pictorially, this can be imagined as shrinking the edge). A *subdivision* of an edge $e = \{u, v\}$ is an operation which produces a new graph by removing e and adding a path connecting u and v.

Tree decomposition, tree-width. The notion of tree-width was reintroduced in [11, 3]. It proved to be a useful parameter of graphs in algorithmic graph theory. A *tree-decomposition* of a graph G, denoted $TD(G)$, is a pair $(T, \mathcal{X})$, where T is a rooted unordered tree and $\mathcal{X} = (X_z)_{z \in V(T)}$ is a family of subsets of $V(G)$ satisfying

(i) $\cup_{z \in V(T)} X_z = V(G)$,
(ii) for every $\{u, v\} \in E(G)$, there is a $z \in V(T)$ such that $u, v \in X_z$, and
(iii) $X_{z_1} \cap X_{z_3} \subseteq X_{z_2}$ for every $z_1, z_2, z_3 \in V(T)$ such that z_2 is on the simple path connecting z_1 with z_3 in T.

The set X_z associated with a node z of T is called the *bag* of z. The nodes of T will often be referred to as the nodes of $TD(G)$. The tree-width of $TD(G)$ is $\max_{z \in V(T)} |X_z| - 1$, and the *tree-width* of G, denoted $\mathrm{tw}(G)$, is the minimum tree-width over all tree-decompositions of G. By graphs of bounded tree-width we mean graphs of tree-width at most k, where k is some constant.

A class of graphs $\mathcal{G}$ is called *hereditary* if for any graph $G \in \mathcal{G}$ all its subgraphs also belong to $\mathcal{G}$. The class of graphs of treewidth at most k is hereditary. The same also holds for planar graphs (as clearly any subgraph of a planar graph is still planar).

Graph isomorphism, Graph canonization. Graphs G and G' are *isomorphic* if there exists a bijection $\pi : V(G) \to V(G')$ such that $\{u, v\} \in E(G)$ if and only if $\{\pi(u), \pi(v)\} \in E(G')$. Graph canonization is a function from graphs to strings such that two graphs have the same canonization if and only if they are isomorphic.

Subgraph isomorphism and induced subgraph isomorphism. We say that a graph G_1 is *subgraph isomorphic* to a graph G_2 if G_1 is isomorphic to a subgraph of G_2. We say that a graph G_1 is *induced subgraph isomorphic* to a graph G_2 if G_1 is isomorphic to an induced subgraph of G_2. Deciding whether a graph is (induced) subgraph isomorphic to another graph is NP-complete and it remains NP-complete even for bounded-treewidth graphs [10]. Note that there are graph classes, e.g. bounded-treewidth graphs or planar graphs, for which isomorphism can be decided in polynomial time but for which subgraph isomorphism is NP-complete. We are mostly interested in such classes because for them it is not obvious whether *fast* graph mining algorithms exist.

Homeomorphism and induced homeomorphism. We say that a graph G_1 is *homeomorphic* to a graph G_2 if there is a graph G_1' which can be obtained from G_1 by subdividing its edges and G_1' is subgraph isomorphic to G_2. We say that a graph G_1 is *induced homeomorphic* to a graph G_2 if there is a graph G_1' which can be obtained from G_1 by subdividing its edges and G_1' is induced subgraph isomorphic to G_2. Deciding if a graph is homeomorphic to another graph is NP-complete even for graphs of bounded treewidth and unbounded maximum degree [10].

Minor embedding and induced minor embedding. We say that a graph G_1 is *minor-embeddable* to a graph G_2 if there is a graph G_1' isomorphic to G_1 which can be obtained from a subgraph of G_2 by contracting edges and deleting loops and multiple-edges thus produced. We say that a graph G_1 is *induced minor-embeddable* to a graph G_2 if there is a graph G_1' isomorphic to G_1 which can be obtained from an induced subgraph of G_2 by contracting edges and deleting loops and multiple-edges thus produced. Deciding if a graph is minor-embeddable to another graph is NP-complete even for graphs of bounded treewidth and unbounded maximum degree [10].

Fixed-parameter tractability. Formally, a parameterized decision problem is a language $L \subseteq \Sigma^* \times N$ where Σ is a finite alphabet and N is the set of natural numbers [1]. An instance of a parametrized problem is a pair (x, k) where $k \in N$ is called *parameter* of the problem. A problem is *fixed-parameter tractable* (abbreviated *FPT*) if there exists an algorithm for solving instances of it which runs in time $|x|^{O(1)} \cdot f(k)$ where f is a computable function. Notice that whether a problem is fixed-parameter tractable depends on the selected parameterization. For instance, when the parameter of the problem is $|x|$, i.e. the actual size of the problem, then any problem e.g. from classes such as e.g. NP, EXP, NEXP is fixed-parameter tractable with such a parameterization. On the other hand, it is widely believed that e.g. the clique problem is not fixed-parameter tractable with the parameter being size of the clique. To capture a conjectured intractability hierarchy, the W-hierarchy is used which consists of an infinite number of increasingly more intractable classes W[1], W[2], etc. The W-hierarchy is based on *fixed-parameter reductions*. A problem L_A is fixed-parameter reducible to a

problem L_B if there exists an algorithm for transforming instances $(x,k)_A$ of the problem L_A to instances $(x',k')_B$ of the problem B such that:

(i) the transformation algorithm runs in time $|x|^{O(1)} \cdot f(k)$ where f is a computable function,
(ii) $k' \leq g(k)$ where g is a computable function,
(iii) $(x,k)_A \in L_A$ if and only if $(x',k')_B \in L_B$ (informally, $(x,k)_A$ has a 'yes' solution if and only if $(x',k')_B$ has a 'yes' solution).

3 Graph Mining Problems

In this section, we define the mining problems studied in this paper and describe their basic properties. We start with the definition of the classical frequent connected graph mining problem.

A *transaction database* is a a multiset of graphs from a given class $\mathcal{G}$. Given a pattern matching operator $\preccurlyeq$ (subgraph isomorphism or induced subgraph isomorphism), the frequency of a graph G in a transaction database DB, denoted by $\mathsf{freq}(G,DB)$, is given as $\mathsf{freq}(G,DB) = |\{G' \in DB | G \preccurlyeq G'\}|$. Given a threshold t, G is said to be frequent if $\mathsf{freq}(G,DB) \geq t$. The elements of the multiset $\{G' \in DB | G \preccurlyeq G'\}$ are called *occurrences* of the graph G in the database DB. We will often represent the set of occurrences also just by names or IDs of the graphs contained in it (e.g. G_1 will be represented just by "1").

Definition 1 (The Frequent Connected Graph Mining (FCGM) Problem). Given *a class $\mathcal{G}$ of graphs, a transaction database DB of graphs from $\mathcal{G}$, a pattern matching operator $\preccurlyeq$, and frequency threshold, list the set of frequent connected graphs $G \in \mathcal{G}$ and their occurrences.*

In this paper, we are interested in another closely related type of problem which is to mine all graphs with frequency at least one *in certain order*.

Definition 2 (The Ordered Mining Problems). Given *a class $\mathcal{G}$ of graphs, a transaction database DB of graphs from $\mathcal{G}$ and a pattern matching operator $\preccurlyeq$, list the set of connected graphs $G \in \mathcal{G}$ with $\mathsf{freq}(G,DB) \geq 1$ and their occurrences in the transactions in the given order*[4]*:*

- *from most frequent to least frequent ($\mathsf{ALL}_{F\to I}$ problem),*
- *from least frequent to most frequent ($\mathsf{ALL}_{I\to F}$ problem),*
- *from smallest size to largest size ($\mathsf{ALL}_{S\to L}$ problem),*
- *from largest size to smallest size ($\mathsf{ALL}_{L\to S}$ problem).*

Here size *of a graph G refers to $|E(G)|$ when $\preccurlyeq$ is subgraph isomorphism and to $|V(G)|$ when $\preccurlyeq$ is induced subgraph isomorphism.*

[4] $\mathsf{ALL}_{F\to I}$ stands for 'frequent to infrequent', $\mathsf{ALL}_{I\to F}$ stands for 'infrequent to frequent', $\mathsf{ALL}_{S\to L}$ stands for 'small to large' and $\mathsf{ALL}_{L\to S}$ stands for 'large to small'.

The *parameter* of the above problems is the size of DB. One can easily construct examples for which the number of frequent connected subgraphs is exponential in this parameter. Thus, in general, the set of all frequent connected subgraphs cannot be computed in time polynomial only in the size of DB. Since this is a common feature of many listing problems, the following problem classes are usually considered in the literature (see, e.g., [6]). For some input I, let O be the output set of some finite cardinality N. Then the elements of O, say $o_1, \ldots, o_N$, are listed with:

- *polynomial delay* if the time before printing o_1, the time between printing o_i and o_{i+1} for every $i = 1, \ldots, N-1$, and the time between printing o_N and the termination is bounded by a polynomial of the size of I,
- *incremental polynomial time* if o_1 is printed with polynomial delay, the time between printing o_i and o_{i+1} for every $i = 1, \ldots, N-1$ (resp. the time between printing o_N and the termination) is bounded by a polynomial of the combined size of I and the set $\{o_1, \ldots, o_i\}$ (resp. O),
- *output polynomial time* (or *polynomial total time*) if O is printed in time polynomial in the combined size of I and the *entire* output O.

Clearly, polynomial delay implies incremental polynomial time, which, in turn, implies output polynomial time. Furthermore, in contrast to incremental polynomial time, the delay of an output polynomial time algorithm may be exponential in the size of the input even before printing the first element of the output.

Example 1. Let us have graphs $G_1 = (\{1,2,3\}, \{\{1,2\},\{2,3\},\{3,1\}\})$ and $G_2 = (\{1,2,3,4\}, \{\{1,2\},\{2,3\},\{3,4\}\})$, $DB = \{G_1, G_2\}$ and let $t = 2$. A solution of the FCGM problem is

$$H_1 = (\{1\}, \{\}), OCC_1 = \{1,2\}$$
$$H_2 = (\{1,2\}, \{\{1,2\}\}), OCC_2 = \{1,2\}$$
$$H_3 = (\{1,2,3\}, \{\{1,2\},\{2,3\}\}), OCC_3 = \{1,2\}$$

where OCC_i denotes the occurrences of the graph H_i. A solution of the problem $\mathsf{ALL}_{F \to I}$ is

$$H_1 = (\{1\}, \{\}), OCC_1 = \{1,2\}$$
$$H_2 = (\{1,2\}, \{\{1,2\}\}), OCC_2 = \{1,2\}$$
$$H_3 = (\{1,2,3\}, \{\{1,2\},\{2,3\}\}), OCC_3 = \{1,2\}$$
$$H_4 = (\{1,2,3\}, \{\{1,2\},\{2,3\},\{3,1\}\}), OCC_4 = \{1\}$$
$$H_5 = (\{1,2,3,4\}, \{\{1,2\},\{2,3\},\{3,4\}\}), OCC_4 = \{2\}$$

Remark 1. There is an incremental-polynomial-time algorithm for the FCGM (FCIGM) problem if and only if there is an incremental-polynomial time algorithm for $\mathsf{ALL}_{F \to I}$ with (induced) subgraph isomorphism as a pattern matching operator.

In general, we will see in the next sections that the different ordered mining problems possess different computational complexities under standard complexity theoretic assumptions.

4 Mining All (Induced) Subgraphs

4.1 Negative Results

In this section, we provide several negative results regarding complexity of some of the enumeration problems considered in this paper. The first theorem connects the hardness of the frequent subgraph enumeration problem to fixed-parameter tractability of the pattern matching operator (subgraph isomorphism or induced subgraph isomorphism).

Theorem 1. *Let $\mathcal{G}$ be a class of graphs. Let $\preccurlyeq$ be either subgraph isomorphism or induced subgraph isomorphism. If deciding $H \preccurlyeq G$ where $G, H \in \mathcal{G}$ is not fixed-parameter tractable with the parameter $|H|$ then there is no output-polynomial-time algorithm for enumerating frequent connected graphs from databases consisting of graphs from $\mathcal{G}$.*

Proof. Let us suppose that there is an algorithm for mining frequent connected graphs from databases of graphs from $\mathcal{G}$ which runs in output-polynomial time. Let $G, H \in \mathcal{G}$. We show that then it is always possible to decide whether $H \preccurlyeq G$ holds in time $f(|H|) \cdot |G|^{O(1)}$. If $|H| > |G|$ then $H \not\preccurlyeq G$ and we can finish. If $|H| \leq |G|$, we set $DB = \{H, G\}$ and we let the pattern mining algorithm run on DB with minimum frequency $t = 2$. Since there are at most $2^{|E(H)|}$ connected subgraphs of H, the mining algorithm will produce an output of length at most $|H| \cdot 2^{|E(H)|}$ in time $\mathsf{poly}\left(|H| + |G|, |H| \cdot 2^{|E(H)|}\right)$. If the output contains a frequent graph F such that $|V(F)| = |V(H)|$ and $|E(F)| = |E(H)|$ (such a graph F must be isomorphic to H), we return *true*. Otherwise, we return *false*. Now, if we return true then $H \preccurlyeq G$ must hold because if a graph F isomorphic to H is frequent, it must hold $F \preccurlyeq G$ and therefore also $H \preccurlyeq G$. Similarly, if we return false then there is no frequent graph F isomorphic to H and therefore $H \not\preccurlyeq G$. This all put together runs in time $f(|H|) \cdot |G|^{O(1)}$ where f is a computable function. However, then the just described procedure would give us an algorithm for the pattern matching operator which would be fixed-parameter tractable with the parameter $|H|$ which is a contradiction. □

From the proof of the above theorem, we can obtain the following corollary[5].

[5] The hardness for the $\mathsf{ALL}_{F \to I}$ problem follows from Theorem 1 together with Remark 1, whereas hardness of the $\mathsf{ALL}_{S \to L}$ problem follows from a simple modification of the proof of Theorem 1 where we use a hypothetic incr.-poly.-time algorithm for solving the $\mathsf{ALL}_{S \to L}$ problem and stop it after printing the first graph with more edges than H (or more vertices than H in the case of induced subgraph mining).

Corollary 1. *Let $\mathcal{G}$ be a class of graphs. Let $\preccurlyeq$ be either subgraph isomorphism or induced subgraph isomorphism. If deciding $H \preccurlyeq G$ where $G, H \in \mathcal{G}$ is not fixed-parameter tractable with the parameter $|H|$ then $\mathsf{ALL}_{F\to I}$ and $\mathsf{ALL}_{S\to L}$ cannot be solved in incremental polynomial time.*

However, there are also graph classes with FPT subgraph isomorphism, e.g. planar graphs [9], for which $\mathsf{ALL}_{F\to I}$ cannot be solved in incr.-poly. time[6].

Theorem 2. *The problem $\mathsf{ALL}_{F\to I}$ cannot be solved in incremental polynomial time for the class $\mathcal{G}$ of planar graphs and subgraph isomorphism as pattern matching operator.*

Proof. We can use NP-hardness of Hamiltonian cycle problem [2], similarly as [5]. We construct a database consisting of a given graph G and a cycle C on $|V(G)|$ vertices. A graph isomorphic to C with frequency 2 (recall that frequency is given implicitly by the printed occurrences) is output by the mining algorithm among the first $|V(G)| + 1$ graphs if and only if G contains a Hamiltonian cycle. It is easy to see that we could then use an incremental-polynomial time algorithm for the $\mathsf{ALL}_{F\to I}$ problem to solve the Hamitonian cycle problem. Therefore there is no incremental-polynomial time algorithm for the $\mathsf{ALL}_{F\to I}$ problem (unless P=NP). □

This theorem is interesting because in Section 4.2, we will see that the problem $\mathsf{ALL}_{L\to S}$ can be solved with polynomial delay for planar graphs.

Even stronger negative result can be obtained for the problem $\mathsf{ALL}_{I\to F}$.

Theorem 3. *Let $\mathcal{G}$ be a class of graphs. Let $\preccurlyeq$ be either subgraph isomorphism or induced subgraph isomorphism. If deciding $H \preccurlyeq G$ where $G, H \in \mathcal{G}$ is NP-hard then $\mathsf{ALL}_{I\to F}$ cannot be solved in incremental polynomial time (unless $\mathsf{P} = \mathsf{NP}$).*

Proof. Let H and G be graphs from $\mathcal{G}$. We will show how to use an incremental-polynomial-time algorithm for the problem $\mathsf{ALL}_{I\to F}$ to decide whether $H \preccurlyeq G$ in polynomial time. We construct a database of graphs $DB = \{H, G, G\}$ and let the algorithm for the problem $\mathsf{ALL}_{I\to F}$ run until it outputs a graph and its occurrences (implicitly giving us also the frequency) and then we stop it (it follows from definition of incremental-polynomial time that this will run only for time polynomial in the sizes of H and G). It is easy to see that the output graph has frequency 1 if and only if $H \not\preccurlyeq G$. Thus, if the frequency of the output graph is 1 we return 'not (induced) subgraph isomorphic' and if the frequency of the output graph is greater than 1 then we return '(induced) subgraph isomorphic'. Therefore if deciding $H \preccurlyeq G$ is NP-hard for graphs from $\mathcal{G}$ there cannot be an incremental-polynomial-time algorithm for the problem $\mathsf{ALL}_{I\to F}$ (unless $\mathsf{P} = \mathsf{NP}$). □

Using the fact that (induced) subgraph isomorphism is NP-complete even for bounded-treewidth graphs [10] and planar graphs [9], we can obtain the following.

[6] The complexity of the $\mathsf{ALL}_{S\to L}$ problem in these cases remains an interesting open problem.

Corollary 2. *The problem* $\mathsf{ALL}_{I\to F}$ *cannot be solved in incremental polynomial time for the class of planar graphs and for the class of bounded-treewidth graphs.*

Note that Theorem 1 cannot be made as strong as Theorem 3 (i.e. showing that $\mathsf{ALL}_{F\to I}$ cannot be solved in incremental-polynomial time if the pattern matching operator is NP-hard) because the results of Horváth and Ramon from [5] demonstrate that even if the pattern matching operator is NP-hard there can be an incremental-polynomial-time algorithm for mining frequent subgraphs. Theorem 3 shows that we cannot expect such a result for mining *infrequent* subgraphs (i.e. subgraphs with frequency below a threshold).

4.2 Positive Results for $\mathsf{ALL}_{F\to I}$ and $\mathsf{ALL}_{S\to L}$

Before presenting our new results for $\mathsf{ALL}_{L\to S}$ in the next section, we note that there exists the following positive result for frequent graph mining from bounded-treewidth graphs, which was presented in [4, 5].

Theorem 4 (Horváth and Ramon [5], Horváth, Otaki and Ramon [4]). *The FCGM and FCIGM problems can be solved in incremental-polynomial time for the class of bounded-treewidth graphs.*

This result directly translates to a positive result for the problem $\mathsf{ALL}_{F\to I}$ summarized in the following corollary (recall that we have shown in the previous section that $\mathsf{ALL}_{I\to F}$ cannot be solved in incremental-polynomial time for bounded-tree-width graphs) and to a result for the problem $\mathsf{ALL}_{S\to L}$ (this other result follows from the fact that the respective algorithms are level-wise).

Corollary 3. *The problems* $\mathsf{ALL}_{F\to I}$ *and* $\mathsf{ALL}_{S\to L}$ *can be solved in incremental-polynomial time for the class of bounded-treewidth graphs.*

4.3 Positive Results for $\mathsf{ALL}_{L\to S}$

In this section, we describe an algorithm called LargerToSmaller (Alg. 1) which, when given a class of graphs $\mathcal{G}$ in which isomorphism can be decided in polynomial time, solves the problem $\mathsf{ALL}_{L\to S}$ in incremental-polynomial time, or with polynomial delay if $\mathcal{G}$ also admits a polynomial-time canonization. The main employed trick is the observation that for the problem $\mathsf{ALL}_{L\to S}$ it is not necessary to use subgraph isomorphism explicitly for computing occurrences.

The algorithm maintains a data structure ALL storing key-value pairs where keys are graphs and values are sets of IDs[7] of graphs in which the given key graph is contained either as a subgraph or as an induced subgraph (depending on whether we are mining subgraphs or induced subgraphs). The data structure provides four functions/procedures: $\mathsf{ADD}(K, OCC, ALL)$, $\mathsf{GET}(K, ALL)$, $\mathsf{KEYS}(n, ALL)$, and $\mathsf{DELETE}(n, ALL)$.

[7] Here, IDs are just some identifiers given to the database graphs.

The procedure $\mathsf{ADD}(K, OCC, ALL)$ adds the IDs contained in OCC to the set associated with a key contained in ALL which is isomorphic to K or, if no such key is contained in ALL, the procedure stores K in ALL and associates OCC with it. If we restrict attention to graphs from a class $\mathcal{G}$ for which a polynomial-time canonization function running in $O(p(|H|))$ exists (where p is a polynomial) then the procedure $\mathsf{ADD}(K, OCC, ALL)$ can be implemented to run in time $O(p(|K|))$ (we can just store the key graphs as canonical strings, therefore a hashtable with constant-time methods for finding and adding values by their keys can be used). If a polynomial-time canonization function does not exist but graph isomorphism can be decided in time $O(p_{iso}(|K|))$ where p_{iso} is a polynomial then the procedure $\mathsf{ADD}(K, OCC, ALL)$ can be implemented to run in time $O(|\{K' \in KEYS(ALL)\colon |V(K')| = |V(K)| \text{ and } |E(K')| = |E(K)|\}| \cdot p_{iso}(|K|))$.

The function $\mathsf{GET}(K, ALL)$ returns all IDs associated with a key isomorphic to K. The exactly same considerations as for the procedure ADD apply also for this function.

The function $\mathsf{KEYS}(n, ALL)$ returns a pointer to a linked list containing all key graphs stored in ALL which have size n. Since the data structure ALL does not allow deletion of individual keys, it is easy to maintain such a linked list[8].

Finally, the procedure $\mathsf{DELETE}(n, ALL)$ removes the pointer[9] to the linked list containing all key graphs of order n stored in ALL.

The algorithm LargerToSmaller fills in the data structure ALL, starting with the largest graphs and proceeding to the smaller ones. When it processes a graph H, it first prints it and the IDs of the graphs associated to it in the data structure ALL, and then it calls the function REFINE which returns all connected subgraphs H' of H which can be obtained from H by removing an edge or an edge and its incedent vertex of degree one, in the case of subgraph mining, or just by removing a vertex and all its incident edges, in the case of induced subgraph mining. It then associates all occurrences of the graph H with the graphs H' in the datastructure ALL using the procedure ADD. Since the same graph H' may be produced from different graphs H, the occurrences of H' accumulate and we can prove that when a graph H is printed, the data structure ALL already contains all IDs of graphs in which H is contained.

Theorem 5. *Let $\mathcal{G}$ be a hereditary class of graphs with isomorphism decidable in polynomial time. Given a database DB of graphs from $\mathcal{G}$, the algorithm* LargerToSmaller *solves the problem $\mathsf{ALL}_{L\to S}$ in incremental polynomial time. If the graphs from $\mathcal{G}$ also admit a poly-time canonization then the algorithm* LargerToSmaller *solves the problem $\mathsf{ALL}_{L\to S}$ with polynomial delay.*

[8] The reason why the function KEYS does not just return all the key graphs but rather a pointer to the linked list is that if it did otherwise, Algorithm 1 could never run with polynomial delay

[9] Note that we just remove the pointer and do not actually "free" the memory occupied by the graphs. For the practical implementation, we used a programming language with a garbage collector.

Algorithm 1 LargerToSmaller

Require: database DB of transaction graphs
Ensure: all connected (induced) subgraphs and their occurrences

```
1: let ALL be a data structure for storing graphs and their occurrences (as described
   in the main text).
2: for G ∈ DB do
3:    ADD(G, {ID(G)}, ALL)
4: endfor
5: let m := max_{G∈DB} |E(G)| (m := max_{G∈DB} |V(G)| for induced subgraph mining).
6: for (l := m; l > 0; l := l − 1) do
7:    for H ∈ KEYS(l, ALL) do
8:       OCC ← GET(H, ALL)
9:       PRINT(H, OCC)
10:      for H' ∈ REFINE(H) do
11:         if H' is connected then
12:            ADD(H', OCC, ALL)
13:         endif
14:      endfor
15:   endfor
16:   DELETE(l, ALL)
17: endfor
```

Proof. First, we show that the algorithm prints every (induced) subgraph of the graphs in DB. Let us assume, for contradiction, that this is not the case and let G^* be a maximal connected graph which is a (induced) subgraph of a graph in the database and such that no graph isomorphic to it is printed by the algorithm. It is easy to verify that such a graph G^* cannot be isomorphic to any graph in DB. Since G^* is not isomorphic to a graph from DB and since it is a maximal graph not printed, there must be a supergraph G' of G^* such that $|E(G^*)|+1 = |E(G')|$ in the case of subgraph isomorphism ($|V(G^*)|+1 = |V(G')|$ in the case of induced subgraph isomorphism, respectively) and such that a graph isomorphic to it is printed by the algorithm. However, if such a graph was printed then a graph isomorphic to G^* would have to be in $\mathsf{REFINE}(G')$ and would have to be printed eventually, which is a contradiction.

Second, we show that the occurrences printed with each graph are correct. First, if a printed graph G does not have any strict supergraph in the database then it must be equivalent to one or more database graphs. However, it is easy to check by simple inspection of the algorithm that the occurrences must be correct in this case. For the rest of the printed graphs (i.e. graphs which have strict supergraphs in the databse), let us assume, for contradiction, that G^* is a maximal graph printed by the algorithm for which the printed occurrences are not correct, i.e. either there is an ID of a database graph printed for G^* of which G^* is not an (induced) subgraph or there is an ID of a database graph not printed for G^* of which G^* is actually an (induced) subgraph. *(False occurrence:)* If there is an ID of a database graph of which G^* is not a (induced) subgraph then at

least one of the graphs from which G^* can be obtained by refinement must have an ID associated which it should not have. But then G^* could not be maximal graph with this property, which is a contradiction. *(Missing occurrence:)* If there is a missing ID of a database graph of which G^* is a (induced) subgraph then one of the following must be true: (i) G^* is isomorphic to the database graph but then it is easily seen by inspection of the algorithm that the ID of this graph cannot be missing from the occurrences of G^*, (ii) there is a strict supergraph G' of G^* which is (induced) subgraph isomorphic to the database graph and which is not printed (and therefore its occurrences are not added to the data structure ALL), but this is not possible as the first part of the proof shows, (iii) there is a strict supergraph G' of G^* which is (induced) subgraph isomorphic to the database graph and the respective ID was not associated to it, but then G^* could not be a maximal graph with this property. Thus, we have a contradiction.

Third, we show that if there is a polynomial-time isomorphism algorithm then the algorithm LargerToSmaller runs in incremental-polynomial time. First, notice that the first for-loop takes only polynomial time in the size of the database. We can see easily that the time before printing the first graph is also bounded by a polynomial in the size of the database. Next, the for-loop on line 10 is repeated at most $|H|$-times for any graph H. Adding a graph H' to the data structure ALL or getting occurrences of a graph H' from the data structure ALL takes time polynomial in the number of graphs already stored in it and the size of the graph being stored (which is discussed in the main text). The number of graphs already stored in the data structure ALL is bounded by $P \cdot M$ where P is the number of already printed graphs and M is the maximum size of a graph in the database. Thus, we have that the time between printing two consecutive graphs is bounded by a polynomial in the size of the database and in the number of already printed graphs, i.e. the algorithm runs in incremental polynomial time.

Fourth, we can show using essentially the same reasoning that if there is a polynomial-time graph canonization algorithm then the algorithm runs with polynomial delay. □

Using the results on complexity of graph canonization for planar [12] and bounded-treewidth graphs [7], we can get the following corollary.

Corollary 4. *The problem* $\mathsf{ALL}_{L \to S}$ *can be solved with polynomial delay for the classes of planar and bounded-treewidth graphs.*

In fact, there are many other classes of graphs for which graph isomorphism is known to be decidable in polynomial time, e.g. graphs of bounded degree [8]. One can easily use the theorems presented in this section for such classes too as long as they are hereditary, as is the case for bounded-degree graphs.

4.4 Other Negative Results

The following theorem asserts that the results for the problem $\mathsf{ALL}_{L \to S}$ are essentially optimal in the sense that existence of a polynomial-time algorithm for

graph isomorphism is both sufficient and necessary for existence of an incremental-polynomial-time algorithm for the problem $\mathsf{ALL}_{L\to S}$.

Theorem 6. *The problem $\mathsf{ALL}_{L\to S}$ can be solved in incremental-polynomial time for graphs from a hereditary class $\mathcal{G}$ if and only if graph isomorphism can be decided in polynomial-time for graphs from $\mathcal{G}$.*

Proof. ($\Rightarrow$) The proof idea is similar to the idea of the proof of Theorem 3. Let H and G be graphs from $\mathcal{G}$. We will show how to use an incremental-polynomial-time algorithm for the problem $\mathsf{ALL}_{L\to S}$ to decide whether H and G are isomorphic in polynomial time. If $|V(H)| \neq |V(G)|$ or $|E(H) \neq |E(G)|$, we return 'not isomorphic'. Otherwise, we construct a database of graphs $DB = \{H, G\}$ and let the algorithm for the problem $\mathsf{ALL}_{L\to S}$ run until it outputs a graph and its occurrences (implicitly giving us also the frequency) and then we stop it (it follows from definition of incremental-polynomial time that this will run only for time polynomial in the sizes of H and G). Then the output graph has frequency 1 if and only if H and G are not isomorphic. So, if the frequency of the output graph is 1 we return 'not isomorphic' and if the frequency of the output graph is greater than 1 then we return 'isomorphic'. This gives us an algorithm for deciding isomorphism of graphs which runs in polynomial time.

($\Leftarrow$) This direction is explicitly shown in Theorem 5. □

The next theorem indicates that $\mathsf{ALL}_{L\to S}$ is the simplest (complexity-wise) from the enumeration problems considered in this paper because e.g. the problems $\mathsf{ALL}_{F\to I}$ and $\mathsf{ALL}_{I\to F}$ may be unsolvable in incremental-polynomial time even if a polynomial-time graph isomorphism algorithm existed.

Theorem 7. *If $\mathsf{GI} \in \mathsf{P}$ and $\mathsf{P} \neq \mathsf{NP}$ was true then there would be an incremental-polynomial-time algorithm for the problem $\mathsf{ALL}_{L\to S}$ for the class $\mathcal{G}$ of all graphs but no incremental-polynomial-time algorithm for the problems $\mathsf{ALL}_{I\to F}$ and $\mathsf{ALL}_{F\to I}$ for the class of all graphs.*

Proof. The positive result for the problem $\mathsf{ALL}_{L\to S}$ follows from Theorem 5. The hardness of FCGM and FCIGM has been shown in [5] and in [4] using reductions from Hamiltonian cycle problem (for mining under subgraph isomorphism) and from maximum clique problem (for mining under induced subgraph isomorphism), from which the hardness result for the $\mathsf{ALL}_{F\to I}$ problem follows. The hardness of the $\mathsf{ALL}_{I\to F}$ problem follows from Theorem 3. □

For now, we leave open the question of complexity of the $\mathsf{ALL}_{S\to L}$ problem conditioned only on the pattern matching operator not being in P. Theorem 1 asserts that solving this problem in incremental polynomial time is not possible if the pattern matching operator is not fixed-parameter tractable with the parameter being the size of the pattern graph, which is a widely believed conjecture.

5 Mining under Homeomorphism and Minor Embedding

Many of the results presented in this paper may be generalized to mining with other important pattern matching operators: (induced) homeomorphism and (induced) minor embedding. In this section, we briefly discuss these generalizations. Here, we only consider mining from unlabeled graphs because there is no generally agreed-upon definition of homeomorphism or minor embedding of labeled graphs for pattern mining[10].

The ideas from Theorem 1 are not relevant for mining under minor embedding or homeomorphism because minor embedding and homeomorphism are fixed-parameter tractable with the parameter being the size of the pattern graph as shown in [11, 3]. However, it can be used together with the following theorem to show hardness of the $\mathsf{ALL}_{S\to L}$ problem under induced homeomorphism and induced minor embedding.

Theorem 8. *Deciding induced homeomorphism or induced minor embedding $G_1 \preccurlyeq G_2$ is not fixed-parameter tractable with the size of G_1 as the parameter (unless FPT = W[1]).*

Proof. This can be shown by reduction from the k-independent set problem parameterized by k which follows from the following simple observation. Let G be a graph. G contains an independent set of size k if and only if $H \preccurlyeq G$ where H is a graph consisting of k isolated vertices. Notice that this theorem holds also when we restrict G_1 and G_2 to be connected graphs. The basic idea of the proof is then the same. The reduction from k-independent set problem is then as follows. We create a new graph G' by taking the graph G from the proof, adding a new vertex and connecting it to all vertices of G. Instead of taking H as a set of k isolated vertices we let H be a star on $k+1$ vertices (i.e. a tree in which all vertices are connected to one vertex v). We then again have $H \preccurlyeq G'$ if and only if G contains an independent set of size k. □

Ideas analogical to those from Theorems 1, 2, 3 and 8 can be used to obtain the following negative results for mining under (induced) homeomorphism and (induced) minor embedding.

Theorem 9. *The problem $\mathsf{ALL}_{S\to L}$ under induced homeomorphism or induced minor embedding cannot be solved in incremental-polynomial time for the class of all graphs (unless FPT = W[1]). The problems $\mathsf{ALL}_{F\to I}$ under (induced) homeomorphism or (induced) minor embedding cannot be solved in incremental-polynomial time for the class of all graphs (unless P = NP). The problem $\mathsf{ALL}_{I\to F}$ under (induced) homeomorphism or (induced) minor embedding cannot be solved in incremental-polynomial time for the class of bounded-treewidth graphs (unless P = NP).*

[10] For homeomorphism, for instance, we could allow the subdivided edges to have different labels or, to the contrary, we could require them all to have the same label etc. Then another question could be how we should treat labels of vertices etc. While these considerations are interesting even for practice, they are out of the scope of this paper.

The positive results from Section 4.2 may be adapted for mining under (induced) homeomorphism and (induced) minor embedding as follows. We can essentially use the algorithm LargerToSmaller as is, but we need to modify the procedure $\mathsf{REFINE}(H)$ (the modified algorithm will be denoted as LargerToSmaller* to avoid confusion). For mining under minor embedding, when given a graph H, the procedure REFINE should return all graphs H' which can be obtained from H by removing an edge or by contracting an edge and removing loops and multiple edges thus produced. For mining under homeomorphism, it should return all graphs H' which can be obtained from H by removing an edge or by contracting an edge incident to a vertex of degree 2. For mining under induced minor embedding, it should return all graphs H' which can be obtained from H by removing a vertex and all its incident edges or by contracting an edge and removing loops and multiple edges thus produced. Finally, for mining under induced homeomorphism, it should return all graphs H' which can be obtained from H by removing a vertex and all its incident edges or by contracting an edge incident to a vertex of degree 2.

Theorem 10. *Let $\mathcal{G}$ be a class of graphs closed under formation of minors admitting a polynomial-time isomorphism algorithm and let the pattern matching operator $\preccurlyeq$ be either (induced) homeomorphism of (induced) minor embedding. Given a database DB of graphs from $\mathcal{G}$, the algorithm* LargerToSmaller* *solves the problem $\mathsf{ALL}_{L\to S}$ in incremental polynomial time. If the graphs from $\mathcal{G}$ also admit a poly-time canonization then the algorithm* LargerToSmaller* *solves the problem $\mathsf{ALL}_{L\to S}$ with polynomial delay.*

Proof (Sketch). We can essentially use the reasoning from the proof of Theorem 5. We only need to notice that $\preccurlyeq$ is transitive and that if $H \preccurlyeq G$ then a graph isomorphic to H can be obtained from G by a repeated application of the procedure REFINE. For instance, to show that all graphs homeomorphic to at least one database graph will be printed eventually, we can reason as follows. We can show using Theorem 5 that every graph which is subgraph isomorphic to at least one database graph must also be printed by the algorithm LargerToSmaller*. If a graph H is homeomorphic to a database graph G then G has a subgraph G', which must be printed at some point, which is isomorphic to a graph which can be obtained from H by subdividing its edges. But this also means that H can be obtained from G' by repeatedly contracting some of its edges incident to a vertex of degree 2 and removing the loops produced by this process. Thus, any graph homeomorphic to at least one of the database graphs will be printed eventually. It is also not difficult to see that all graphs produced by the refinement function REFINE must be homeomorphic to the graph being refined. Since homeomorphism is transitive any of the produced graphs will be homeomorphic to at least one of the database graphs. Finally, to show that the occurrences printed for every output graph are correct, we can repeat the reasoning from Theorem 5 (which we omit here due to space constraints). □

Using the fact that bounded-treewidth and planar graphs are closed under formation of graph minors and that they also admit a polynomial-time isomorphism algorithm, we can obtain the following corollary.

Corollary 5. *The problem* $\mathsf{ALL}_{L\to S}$ *under (induced) homeomorphism or (induced) minor embedding can be solved with polynomial delay for the classes of planar and bounded-treewidth graphs.*

6 Conclusions and Future Work

In this paper, we have shown how different orders in which graphs are enumerated affect computational complexity of the mining problem. We have presented several negative results. We have also described a positive result which shows that it is possible to mine all graphs from a database of bounded-treewidth or planar graphs with polynomial delay under any of the following six different pattern matching operators: (induced) isomorphism, (induced) homeomorphism and (induced) minor embedding. Here, by mining all graphs, we mean enumerating all graphs which have frequency at least one and printing their occurrences in the database. This result holds despite the fact that deciding any of the six pattern matching operators is NP-hard. However, since the positive result depends heavily on mining all graphs and not just the frequent ones, the question whether frequent graph mining is achievable with polynomial delay for an NP-hard pattern matching operator, remains open. In fact, as we have shown, e.g. for planar graphs the latter problem of mining frequent graphs cannot be done even in incremental polynomial time, whereas mining all graphs can be done with polynomial delay.

Lastly, it is worth noting that when we performed preliminary experiments with a simple implementation of the algorithm for the $\mathsf{ALL}_{L\to S}$ problem, we were able to mine completely about 70% molecules from NCI GI dataset consisting of approximately three and half thousand organic molecules. This suggests that the techniques presented in this paper might also lead to development of practical graph mining algorithms. For instance, it would not be difficult to obtain a graph mining algorithm, having similar positive complexity guarantees as the LargerToSmaller algorithm, for mining graphs of bounded radius (from database graphs of arbitrary radius) or with constraints on minimum vertex degree[11] etc. Exploring these and similar ideas is left for future work.

Acknowledgement. This work has been supported by ERC Starting Grant 240186 "MiGraNT: Mining Graphs and Networks, a Theory-based approach".

[11] The latter constraint on degrees would not probably be very relevant for bounded-treewidth graphs or planar graphs. This is because any graph of treewidth k must always have at least one vertex of degree at most k and because any planar graph must always have at least one vertex of degree at most 5. However, isomorphism algorithms are extremely fast in practice, despite not being polynomial in the worst case, so the algorithm that we would obtain for general graphs could still be practical.

The first author is supported by a grant from the Leverhulme Trust (RPG-2014-164).

References

1. J. Flum and M. Grohe. *Parameterized Complexity Theory*. Berlin: Springer Verlag, 2006.
2. M. R. Garey, D. S. Johnson, and R. E. Tarjan. The planar hamiltonian circuit problem is np-complete. *SIAM J. Comput.*, 5(4):704–714, 1976.
3. M. Grohe and D. Marx. Structure theorem and isomorphism test for graphs with excluded topological subgraphs. *CoRR*, abs/1111.1109, 2011.
4. T. Horváth, K. Otaki, and J. Ramon. Efficient frequent connected induced subgraph mining in graphs of bounded tree-width. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2013*, pages 622–637, 2013.
5. T. Horváth and J. Ramon. Efficient frequent connected subgraph mining in graphs of bounded tree-width. *Theor. Comput. Sci.*, 411(31-33):2784–2797, 2010.
6. D. S. Johnson, M. Yannakakis, and C. H. Papadimitriou. On generating all maximal independent sets. *Information Processing Letters*, 27(3):119 – 123, 1988.
7. D. Lokshtanov, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. Fixed-parameter tractable canonization and isomorphism test for graphs of bounded treewidth. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014*, pages 186–195, 2014.
8. E. M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *Journal of Computer and System Sciences*, 25(1):42 – 65, 1982.
9. D. Marx and M. Pilipczuk. Everything you always wanted to know about the parameterized complexity of subgraph isomorphism (but were afraid to ask). In *31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014), STACS 2014*, pages 542–553, 2014.
10. J. Matoušek and R. Thomas. On the complexity of finding iso- and other morphisms for partial k-trees. *Discrete Mathematics*, 108(1-3):343–364, 1992.
11. N. Robertson and P. D. Seymour. Graph minors .xiii. the disjoint paths problem. *J. Comb. Theory, Ser. B*, 63(1):65–110, 1995.
12. J. Torán and F. Wagner. The complexity of planar graph isomorphism. *Bulletin of the EATCS*, 97:60–82, 2009.