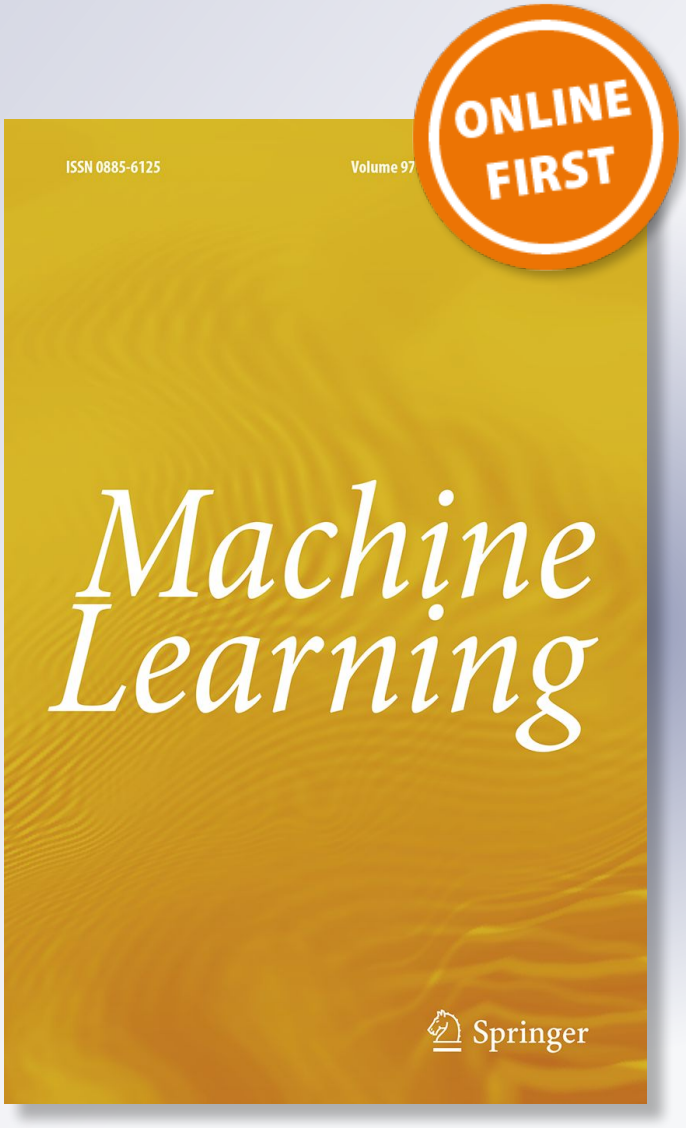
Probabilistic and exact frequent subtree mining in graphs beyond forests

Pascal Welke, Tamás Horváth & Stefan Wrobel

Machine Learning

ISSN 0885-6125

Mach Learn
DOI 10.1007/s10994-019-05779-1



Springer

Machine Learning
https://doi.org/10.1007/s10994-019-05779-1

Probabilistic and exact frequent subtree mining in graphs beyond forests

Pascal Welke[1] · **Tamás Horváth**[1,2,3] · **Stefan Wrobel**[1,2,3]

Received: 14 December 2017 / Accepted: 9 January 2019

Abstract
Motivated by the impressive predictive power of simple patterns, we consider the problem of mining frequent subtrees in arbitrary graphs. Although the restriction of the pattern language to trees does not resolve the computational complexity of frequent subgraph mining, in a recent work we have shown that it gives rise to an algorithm generating probabilistic frequent subtrees, a random subset of all frequent subtrees, from arbitrary graphs with polynomial delay. It is based on replacing each transaction graph in the input database with a forest formed by a random subset of its spanning trees. This simple technique turned out to be quite powerful on molecule classification tasks. It has, however, the drawback that the number of sampled spanning trees must be bounded by a polynomial of the size of the transaction graphs, resulting in less impressive recall even for slightly more complex structures beyond molecular graphs. To overcome this limitation, in this work we propose an algorithm mining probabilistic frequent subtrees also with polynomial delay, but by replacing each graph with a forest formed by an exponentially large implicit subset of its spanning trees. We demonstrate the superiority of our algorithm over the simple one on threshold graphs used e.g. in spectral clustering. In addition, providing sufficient conditions for the completeness and efficiency of our algorithm, we obtain a positive complexity result on exact frequent subtree mining for a novel, practically and theoretically relevant graph class that is orthogonal to all graph classes defined by some constant bound on monotone graph properties.

Keywords Pattern mining · Frequent subgraph mining · Frequent subtree mining · Probabilistic patterns

Editors: Fabrizio Riguzzi, Nicolas Lachiche, Christel Vrain, Elena Bellodi, Riccardo Zese.

✉ Tamás Horváth
horvath@cs.uni-bonn.de

[1] University of Bonn, Bonn, Germany

[2] Fraunhofer IAIS, Schloss Birlinghoven, Sankt Augustin, Germany

[3] Fraunhofer Center for Machine Learning, Sankt Augustin, Germany

Published online: 15 February 2019

1 Introduction

A common approach of extending traditional machine learning algorithms to structured data such as graphs and other relational structures is to embed the instances into some space spanned by some appropriately chosen feature set. This method, also known as *propositionalization* is a common technique in Inductive Logic Programming (ILP) (see, e.g. Kramer et al. 2001, for a survey on this topic). In contrast to kernel methods, where the feature space can be of infinite dimension and the corresponding embedding function is not explicitly given or even unknown together with the feature space, propositionalization in ILP always assumes finite feature sets and explicitly specified embedding functions. One of the main advantages of this technique is that learning/mining in first-order logic can be reduced to some traditional problem setting concerned with learning/mining in a *single* table of *fixed* width. That is, the columns of the table correspond to features and the rows to images of the instances in the underlying feature space. This technique allows also for an effective control of decidability and complexity issues raised by first-order logic. We note that pattern based propositionalization for graph structured data belongs also to this branch of ILP technique. Indeed, since graphs are special relational structures, our approach can be considered as propositionalization in ILP for a restricted type of relational structures. The difference is that for function-free ILP learning settings (i.e., all function symbols are of arity 0 or equivalently, constants) when the patterns are Boolean conjunctive queries (i.e., first-order function-free goal clauses), the embedding function is typically defined by relational homomorphisms. In contrast, regarding graph patterns as first-order function-free goal clauses, for graphs homomorphisms are typically required to be injective (i.e., subgraph isomorphisms).

Since the first application of *frequent subgraphs* as features to molecule classification (Deshpande et al. 2005), many further studies have empirically demonstrated a remarkable predictive performance of frequent patterns on various real-world datasets. In fact, as shown for instance by Bringmann et al. (2006) in the context of correlated pattern mining, even very *simple* patterns, such as paths or trees, often suffice to obtain considerable predictive accuracy. Our empirical results in Welke et al. (2018) on various benchmark chemical graph datasets also confirm that frequent subtrees as features do result only in a marginal loss in predictive performance compared to frequent connected subgraphs.

Despite the structural simplicity of trees, frequent subtrees *cannot* be generated in output polynomial time for *arbitrary* transaction graphs (unless P = NP). The reduction used in the proof of the computational intractability of frequent connected subgraph mining (Horváth et al. 2007) applies to tree patterns as well. This complexity limitation appears to prohibit frequent pattern mining in practically feasible time even for relatively simple graph structures. In particular, the empirical results in our recent paper (Welke et al. 2018) indicate that all the most popular state-of-the-art frequent tree (and subgraph) mining algorithms seem to be limited to databases of small molecular graphs in practice, as their runtime consistently explodes even for slightly more complex graph structures. To overcome this problem, in Welke et al. (2018) we proposed to generate only a *random* subset of frequent subtrees, called *probabilistic* frequent subtrees, and empirically demonstrated that their predictive performance is very close to that of the complete set of frequent subtrees. This, in turn, is a tight approximation of the predictive performance of the complete set of frequent connected subgraphs.

Utilizing that a tree is subgraph isomorphic to a graph if and only if it is subtree isomorphic to one of the graph's spanning trees, our algorithm in Welke et al. (2018) generates probabilistic frequent subtrees in the following simple way: It replaces each transaction graph in

the input database by a forest formed by the vertex disjoint union of a *random* subset of its spanning trees and, using e.g. the levelwise search algorithm (Mannila and Toivonen 1997), generates the set of frequent connected subgraphs (i.e., subtrees) for the forest database obtained. Since spanning trees can be sampled in polynomial time (Wilson 1996) and subgraph isomorphism from a tree into a forest can be decided in polynomial time (Matula 1968), the results in Horváth and Ramon (2010) immediately imply that probabilistic frequent subtrees can be enumerated with polynomial delay in this way if for each transaction graph, the number of spanning trees in the sample is bounded by a polynomial of the graph's size. Note that the output is sound (i.e., all patterns printed are frequent subtrees), but not necessarily complete (i.e., some frequent subtrees may not be enumerated). If, however, the number of spanning trees of the transaction graphs is bounded by a polynomial of their size and all transaction graphs are replaced with their *complete* set of spanning trees, then the algorithm sketched above solves the *exact* frequent subtree mining problem correctly (i.e., soundly and completely) and with polynomial delay. The efficiency follows, together with the remarks above, from the positive result that the set of all spanning trees of a graph can be listed with polynomial delay (Read and Tarjan 1975).

In this work we go beyond the limitation of processing polynomially many spanning trees only and present an algorithm able to generate (probabilistic) frequent subtrees from *arbitrary* graphs with *polynomial delay* by considering a potentially *exponentially* large implicit subset of the spanning trees for each graph in the database. The core of our mining algorithm is a pattern matching algorithm that, for a tree pattern H and transaction graph G, (i) partitions G into a certain set of induced subgraphs, (ii) considers a (random) subset of *local* spanning trees for each induced graph, and (iii) decides whether H is subtree isomorphic to one of the *global* spanning trees of G obtained by combining its local spanning trees in an appropriate way. It is inspired by the paradigms developed by Matoušek and Thomas (1992) and by Shamir and Tsur (1999) for solving subgraph isomorphism for other graph classes.

In a nutshell, we decide the arising pattern matching problem by a dynamic programming algorithm traversing a rooted tree generated for G in a bottom-up manner and computing the final solution from partial ones previously calculated. In our case, the nodes of the rooted tree controlling the evaluation are constructed from the *articulation vertices* of G. Each node v of such a tree is associated with the set of spanning trees of certain blocks of G containing v. For all such local spanning trees τ, we solve the partial subtree isomorphism problem corresponding to v by carefully extending the partial subtree isomorphisms already computed for τ. Iterating over all spanning trees and for all nodes, we can correctly decide subtree isomorphism for the part of G that is "below" v in the rooted tree associated with G. We prove that our algorithm decides subgraph isomorphism from H to $\mathfrak{S}$ correctly, where $\mathfrak{S}$ is the set of spanning trees of G that can be obtained from the combinations of the local spanning trees. Furthermore, our algorithm runs in time *polynomial* in the combined size of H, G, and the number of local spanning trees selected. The significance of this result is that the number of global spanning trees in $\mathfrak{S}$ can be *exponential* in the number of local spanning trees. This property has immediate consequences to probabilistic and exact frequent subtree mining.

Regarding *probabilistic* frequent subtree mining, by considering exponentially many (implicit) global spanning trees instead of polynomially many ones, our technique has an improved performance in terms of *recall* over the simple algorithm in Welke et al. (2018). On the one hand, this improvement is only marginal on molecular graph datasets, due to the relatively simple graph structure of pharmacological compounds (cf. Horváth et al. 2010; Horváth and Ramon 2010). On the other hand, however, on *threshold graphs*, which have applications among others in *spectral clustering* (see, e.g., von Luxburg 2007), the algorithm

presented in this work results in a much higher recall compared to the simple one in Welke et al. (2018). It is important to note that the threshold graphs used in our experiments had a structural complexity *beyond* that of the molecular graphs of pharmacological compounds. Somewhat surprisingly, none of the state-of-the-art frequent subgraph mining algorithms, which are very effective otherwise on molecular graphs, were able to produce any output for threshold graphs in practically feasible time.[1] A robust mining algorithm whose runtime depends *not* on certain, typically unknown implicit characteristics of the data, but on some user specified parameters is of high value for practical problems where the transaction graphs have no (known) specific structural properties that could be utilized by the mining algorithm. Our algorithm is *robust* in this sense because its delay depends only on the number and size of the input graphs and on the sample size parameter. This latter parameter can thus be used to control the trade-off between recall and time complexity. In contrast, most state-of-the-art frequent subgraph mining tools are explicitly or implicitly engineered towards certain structural properties and are therefore not applicable in such a scenario.

Regarding *exact* frequent subtree mining, we first note that despite more than two decades of research there are only a few non-trivial theoretical results concerning the complexity of frequent subgraph mining. In particular, if the transaction graphs are restricted to forests then frequent connected subgraphs (i.e., trees) can be generated with polynomial delay (see, e.g., Chi et al. 2005). Beyond forests, frequent connected subgraphs can be listed in incremental polynomial time for graphs of bounded tree-width (Horváth and Ramon 2010). Using the positive result of Matoušek and Thomas (1992), one can show that for graphs of bounded tree-width (Robertson and Seymour 1986) *and* bounded degree, frequent connected subgraphs can actually be generated with polynomial delay. As a byproduct of our approach, we extend the known positive complexity results on frequent subgraph mining by a new one formulated for a graph class that is not only of theoretical, but also of practical interest.

Our result is based on a subgraph isomorphism algorithm that is always correct if *all* local spanning trees are considered.[2] Accordingly, a sufficient condition for our frequent pattern mining algorithm to be correct and efficient (i.e., polynomial delay) is that the input graphs are *locally easy*; a graph G of size n is locally easy if for all vertices v of G, the union of the blocks containing v has at most poly(n) spanning trees. This class of graphs is *orthogonal* to all graph classes (more precisely, nested hierarchies) that are defined by a constant upper bound on some *monotone* graph property (e.g., graphs of bounded tree-width). A graph property is called *monotone* if it is closed under taking subgraphs. By "orthogonality" we mean that it always contains an infinite number of graphs that are not contained in the other graph class. Furthermore, it turns out that the class of locally easy graphs includes a number of interesting and practically relevant graph classes. The most natural example is the class of *forests*. *Pseudoforests* (i.e., graphs in which every connected component has at most one cycle) and their generalizations, *cactus* graphs (i.e., in which all edges belong to at most one simple cycle) of bounded cyclic block degree (i.e., the maximum number of cyclic blocks[3] sharing a vertex is bounded by a constant) are some further straightforward subclasses of locally easy graphs. Other examples include the class of *d-tenuous outerplanar* graphs (Horváth et al. 2010) of bounded cyclic block degree and that of *k-easy* graphs of bounded cyclic block degree, where a graph is k-easy for some constant $k \geq 0$ integer if all

[1] We have observed a similar behavior on Erdős-Rényi graphs (Erdős and Rényi 1959), even for relatively small value of the edge probability (cf. Welke et al. 2018).

[2] We recall that the problem of deciding whether a tree is subgraph isomorphic to a graph G is NP-complete in general (see, e.g., Garey and Johnson 1979) and remains computationally intractable even for very simple graphs, e.g., when G is a cactus graph (Akutsu 1993).

[3] A cyclic block is a maximal biconnected subgraph with at least three vertices.

blocks have $O\left(n^k\right)$ spanning trees. Our positive result on mining locally easy graphs is thus another step towards exploring the border between tractable and intractable fragments of the frequent pattern mining problem. We conjecture that generalizing our positive result to any natural graph class beyond locally easy graphs is at least as difficult as solving the P versus NP problem.

A preliminary version of this paper appeared as Welke et al. (2015). In comparison to that conference article, in this work we present an application of our pattern mining algorithm to probabilistic frequent subtree mining and empirically evaluate its performance on threshold graphs. As another novel contribution compared to Welke et al. (2015), we generalize our results to a much broader graph class, provide all proofs as well as a tighter runtime analysis, and discuss some theoretically/practically interesting subclasses of this graph class.

Outline The rest of the paper is organized as follows. In Sect. 2 we provide the necessary background, describe a generic levelwise search algorithm mining frequent patterns, and state sufficient conditions for its efficiency. We present our subtree isomorphism algorithm in Sect. 3 and prove some of its algorithmic properties. Using this pattern matching algorithm, in Sect. 4 we describe our mining algorithm enumerating probabilistic frequent subtrees in arbitrary graph databases with polynomial delay and empirically compare its runtime and recall on threshold graphs with that of our algorithm from Welke et al. (2018). We discuss exact frequent subgraph mining for locally easy graphs in Sect. 5, together with some theoretical and practical properties of this graph class. Finally we conclude in Sect. 6 and mention some open problems for further research.

2 Preliminaries

In this section we collect all necessary preliminaries and fix the terminology and notation used in the paper. In particular, we recall some basic notions from graph theory (see, e.g., Diestel 2012), formally define the pattern mining problem considered in this work, give a generic mining algorithm for this problem, and formulate sufficient conditions for the efficiency of this algorithm.

Graphs An *undirected* (resp. *directed*) *graph* G consists of a finite set $V(G)$ of vertices and a set $E(G) \subseteq \{X \subseteq V(G) : |X| = 2\}$ (resp. $E(G) \subseteq V(G) \times V(G)$) of edges. We consider simple graphs, i.e., loops and parallel edges are not permitted. Unless otherwise stated, by *graphs* we mean undirected graphs. An edge $\{u, v\} \in E(G)$ will be denoted by uv and the set of neighbors of a vertex v by $\mathcal{N}(v)$. A *subgraph* of G is a graph G' with $V(G') \subseteq V(G)$ and $E(G') \subseteq E(G)$; G' is a subgraph of G *induced* by a set $V' \subseteq V(G)$ if $V(G') = V'$ and $uv \in E(G')$ if and only if $uv \in E(G)$ for all $u, v \in V'$. Such an induced subgraph is denoted by $G[V']$. A *labeled* graph is a graph G such that all vertices and all edges are labeled with the elements of some finite set. Examples of labeled graphs include molecular graphs, protein-protein interaction graphs, social networks, the Web graph etc. To keep the notation and description concise, we will state all results for *unlabeled* graphs by noting that all our arguments naturally apply to labeled graphs as well.

An *articulation vertex* $v \in V(G)$ is a vertex such that its removal increases the number of connected components of G. A graph is *biconnected* if it is connected and the removal of any vertex does not disconnect it. A *block* is a maximal subgraph of G that is biconnected (i.e., it contains no articulation vertex with respect to itself). A *cyclic block* is a block with

at least three vertices.[4] A *bridge* is an edge that does not lie on any cycle in G. Accordingly, a block is either cyclic or it is a bridge or an isolated vertex.

Two graphs G_1, G_2 are *isomorphic*, if there is a bijection $\varphi : V(G_1) \to V(G_2)$ such that $uv \in E(G_1)$ if and only if $\varphi(u)\varphi(v) \in E(G_2)$ for all $u, v \in V(G_1)$. G_1 is *subgraph isomorphic* to G_2, denoted $G_1 \preccurlyeq G_2$, if G_2 has a subgraph that is isomorphic to G_1. Finally, a *graph class* is a set of pairwise non-isomorphic graphs that share some common property (e.g., they have tree-width at most k for some integer $k > 0$).

Frequent Connected Subgraph Mining In this work, we study a special case of the following problem:

> FREQUENT CONNECTED SUBGRAPH MINING (FCSM) PROBLEM: *Given* a finite set $D \subseteq \mathcal{G}$ for some graph class $\mathcal{G}$ and an integer threshold $t > 0$, *list* all graphs $P \in \mathcal{P}$ for some graph class $\mathcal{P}$, called the *pattern class*, that are subgraph isomorphic to at least t graphs in D.

The patterns in the output must be pairwise non-isomorphic. In contrast to the standard problem definition (see, e.g., Horváth and Ramon 2010), we regard a more general problem *parameterized* by the pattern class $\mathcal{P}$ and focus on the special case of the problem above that $\mathcal{P}$ is the class of trees. This special case will be referred to as FREQUENT SUBTREE MINING (FTM) PROBLEM. Though in this paper we study the FTM problem, in this section we consider the more general FCSM problem. The reason is that below we give an algorithm for the generic FCSM problem and formulate sufficient conditions for $\mathcal{G}$ and $\mathcal{P}$ that guarantee the algorithm to generate frequent patterns with polynomial delay. These conditions may be of some independent interest for the study of other special cases of the FCSM problem.

Note that the mining problem above is a listing problem. For such problems, the following complexity classes are distinguished in the literature (see, e.g., Johnson et al. 1988). Suppose an algorithm $\mathfrak{A}$ for the FCSM problem gets D and t as input and outputs a sequence $O = [p_1, p_2, \cdots, p_n]$ of patterns.[5] Then $\mathfrak{A}$ generates O

- with *polynomial delay*, if the time before the output of p_1, between the output of any two consecutive elements of O, and between the output of p_n and the termination of $\mathfrak{A}$ is bounded by a polynomial of $size(D)$,
- in *incremental polynomial time*, if the algorithm outputs p_1 in time bounded by a polynomial of $size(D)$, the time between outputting p_i and p_{i+1} is bounded by a polynomial of $size(D) + \sum_{j=1}^{i} size(p_j)$, and the time between the output of p_n and termination is bounded by a polynomial of $size(D) + size(O)$,
- in *output polynomial time*, if the algorithm outputs O in time bounded by a polynomial of the combined size of D and O.

Clearly, polynomial delay implies incremental polynomial time, which, in turn, implies output polynomial time. It is an open problem whether the first two complexity classes are identical, or not. In frequent itemset mining, for example, the FP-Growth algorithm (Han et al. 2004) lists frequent patterns with polynomial delay, while the Apriori algorithm (Agrawal et al. 1996) in incremental polynomial time. We note, however, that the Apriori algorithm can easily be transformed into a polynomial delay algorithm by retaining the output of frequent patterns (cf. Horváth et al. 2010).

The FCSM problem and the FTM cannot be solved in output polynomial time; this follows directly from the related negative result in Horváth et al. (2007). One way to obtain positive

[4] This implies that each vertex in a cyclic block lies on at least one cycle in G.

[5] For sake of simplicity, we formulate the definition for $n > 0$.

results is to restrict the graph classes $\mathcal{G}$ and $\mathcal{P}$ in the FCSM problem. To follow this direction, below we first give a generic levelwise search pattern mining algorithm and establish sufficient conditions of polynomial delay pattern generation for this algorithm.

A Generic Mining Algorithm We obtain the main results of this paper by adapting a generic levelwise search algorithm to our problem setting. Levelwise search (Mannila and Toivonen 1997) is one of the most common techniques in pattern mining that can be used to efficiently mine frequent patterns for a broad range of problem settings. Its most popular application is the Apriori algorithm (Agrawal et al. 1996) for frequent itemset mining. In order to find a pattern in level $l+1$, it completely explores all levels up to l. On the one hand, this strategy is disadvantageous if one is interested in mining *long* frequent patterns, on the other hand, in frequent subgraph mining it allows for an incremental polynomial time pattern generation even for NP-complete pattern matching operators (Horváth and Ramon 2010).

Algorithm 1 A generic levelwise graph mining algorithm.

input: $D \subseteq \mathcal{G}$ for some graph class $\mathcal{G}$, a pattern class $\mathcal{P}$, and an integer $t > 0$
output: all frequent subgraphs of D that are in $\mathcal{P}$

```
1: let S_0 ⊆ P be the set of frequent pattern graphs consisting of a single vertex
2: for (l := 0; S_l ≠ ∅; l := l + 1) do
3:     set S_{l+1} := ∅ and C_{l+1} := ∅
4:     for all P ∈ S_l do
5:         print P
6:         for all H ∈ ρ(P) ∩ P satisfying H ∉ C_{l+1} do
7:             add H to C_{l+1}
8:             if SupportCount(H, D) ≥ t then
9:                 add H to S_{l+1}
```

Algorithm 1 is a generic levelwise search algorithm for the FCSM problem. It is a slight modification of the algorithm in Horváth and Ramon (2010); the only changes are in Lines 1 and 6. It calculates the set of candidate (resp. frequent) patterns of level l in the set variable $\mathcal{C}_l$ (resp. $\mathcal{S}_l$). In Line 6 it computes the set $\rho(P)$ of refinements of a pattern P obtained from P by extending it with an edge in all possible ways. That is, it either adds a new vertex w to P and connects it to any vertex in $V(P)$ by an edge, or it connects two vertices in $V(P)$ that have not been connected yet.[6] Clearly, $|\rho(P)|$ is bounded by $|V(P)|^2$. Subroutine SupportCount(H, D) in Line 8 returns the number of graphs $G \in D$ with $H \preccurlyeq G$.

It is shown in Horváth and Ramon (2010) that the original version of Algorithm 1 mines frequent patterns with polynomial delay if patterns and transactions satisfy certain conditions. However, these conditions have been formulated for the case that the pattern and transaction graph classes are the same. Below we give a theorem generalizing these conditions to the case that they can be different. Its proof is very similar to that in Horváth and Ramon (2010). We nevertheless give it for completeness.

Theorem 1 *Let $\mathcal{G}$ and $\mathcal{P}$ be the transaction and pattern graph classes satisfying the following conditions:*

1. *All graphs in $\mathcal{P}$ are connected. Furthermore, $\mathcal{P}$ is closed downwards under taking subgraphs, i.e., for all $H \in \mathcal{P}$ and for all connected graphs H' we have $H' \in \mathcal{P}$ whenever $H' \preccurlyeq H$.*

[6] For the case of tree pattern generation, the second type of extension can be omitted, as it always results in cycles. Hence, in this case $|\rho(P)| = |V(P)|$.

2. *The membership problem for $\mathcal{P}$ can be decided efficiently, i.e., for any graph H it can be decided in polynomial time if $H \in \mathcal{P}$.*
3. *Subgraph isomorphism in $\mathcal{P}$ can be decided efficiently, i.e., for all $H_1, H_2 \in \mathcal{P}$, it can be decided in polynomial time if $H_1 \preccurlyeq H_2$.*
4. *Subgraph isomorphism between patterns and transactions can be decided efficiently, i.e., for all $H \in \mathcal{P}$ and $G \in \mathcal{G}$, it can be decided in polynomial time if $H \preccurlyeq G$.*

Then the FCSM problem can be solved irredundantly with polynomial delay for $\mathcal{P}$ and for all finite subsets $D \subseteq \mathcal{G}$.

Proof Let $\mathcal{G}$ and $\mathcal{P}$ be two graph classes such that Conditions 1–4 hold. We first prove that Algorithm 1 is correct (i.e., sound and complete) and irredundant. The soundness is immediate from Lines 6 and 8. To show the completeness, let $H \in \mathcal{P}$ be frequent in D. We prove by induction on $|E(H)|$ that it will be generated by the algorithm. The proof of the base case that H consists of a single vertex is straightforward by Line 1. For the inductive step we have that H has a vertex with degree one or an edge that can be removed without disconnecting H. Let H' be the graph obtained from H by deleting such a vertex (and the edge adjacent to it) or such an edge. By construction, H' is connected and hence $H' \in \mathcal{P}$ follows from Condition 1. Since it is frequent, it will be generated by Algorithm 1 by the induction hypothesis. Furthermore, as $H \in \rho(H') \cap \mathcal{P}$, we have $H \in \mathcal{C}_{|E(H)|}$ by Lines 6 and 7. Therefore, H is added to $\mathcal{S}_{|E(H)|}$ because it is frequent (Line 9), completing the proof of completeness. Finally, the proof of irredundancy is immediate from the condition tested in Line 6 and the proof of polynomial delay is similar to that of the related Theorem 1 in Horváth and Ramon (2010). □

Note that the conditions above allow for mining frequent patterns that do *not* belong to $\mathcal{G}$. Furthermore, they enable the generation of restricted subsets of *all* frequent subgraphs. For example, we can mine frequent paths in transaction databases consisting of trees. We will utilize the latter property when restricting $\mathcal{P}$ to trees.

Regarding Condition 4, the complexity of subgraph isomorphism is typically disregarded by the pattern mining algorithms.[7] Efficient algorithms are restricted to tree databases (see Chi et al. 2005, for an overview), while general graph miners (e.g., Nijssen and Kok 2005; Kuramochi and Karypis 2004; Zhao and Yu 2008) use methods with exponential worst-case time bounds that are based on the classic algorithm by Ullmann (1976) or on some of its extensions (e.g., Cordella et al. 1999) to solve the subgraph isomorphism problem. The literature typically justifies this by showing experimental results on chemical graph databases, on which most mining systems are fast. Somewhat surprisingly, as we discuss in Sect. 4.2, most state-of-the-art graph mining algorithms are actually limited to molecular graph databases in practice (Welke et al. 2018). In fact, there is no clear way to predict whether the graph miners in the literature will be fast or inapplicable on a given dataset, which heavily restricts their usefulness e.g. in a data exploration setting.

Finally, although it is not required by Theorem 1, the complexity of deciding membership in the transaction class $\mathcal{G}$ is a crucial (practical) issue. For some well-defined graph classes, e.g., graphs of tree-width at most k, membership is computationally intractable if k is not a constant (Arnborg et al. 1987). Therefore deciding whether a given graph mining algorithm can be applied efficiently (i.e., whether $D \subseteq \mathcal{G}$) may already be intractable. In practice, the speed of many existing frequent subgraph mining systems (e.g., Kuramochi and Karypis

[7] In fact, most graph miners focus on efficient candidate enumeration, instead of embedding computation. Here, we go in the opposite direction, focusing on the embedding operator, and refer the reader to the related work (e.g. Chi et al. 2005) for a discussion of efficient candidate generation.

2004; Nijssen and Kok 2005) often depends on some graph properties that are not formally stated and hence not testable.

3 An efficient embedding operator for trees

This section is devoted to the support counting step (cf. SUPPORTCOUNT(H, D) in Line 8) of Algorithm 1. For the FTM problem setting this step reduces to the following decision problem:

> SUBTREE ISOMORPHISM (SUBTREEISO) PROBLEM: *Given* a tree H (the *pattern*) and a graph G (the *transaction* graph), *decide* whether or not $H \preccurlyeq G$.

In Theorem 2 below we first claim that SUBTREEISO can be decided in time polynomial in the number of *local* spanning trees of certain induced subgraphs of G. In Sect. 4 we then show that the algorithm used in the proof of this result can be modified in a natural way to decide SUBTREEISO with one-sided error in polynomial time by considering a potentially exponentially large subset of the spanning trees of G, for any arbitrary graph G. This modified algorithm will allow for efficient probabilistic frequent subtree mining. To state Theorem 2, our main result for this section, we first introduce the following notation: For a graph G and $v \in V(G)$, let $f_v(G)$ be the number of spanning trees in the union of the blocks containing v and define

$$f_{\max}(G) = \max_{v \in V(G)} f_v(G) \,. \tag{1}$$

Theorem 2 *The* SUBTREEISO *problem can be solved in time*

$$O\left(f_{\max}^2(G) \cdot |E(G)| \cdot |V(H)|^{1.5}\right) \,.$$

To put Theorem 2 into context, we note that SUBTREEISO is a well-known NP-complete problem (it generalizes e.g. the Hamiltonian path problem). If, however, the transaction graph is also a tree, the restricted problem belongs to P (see, e.g., Shamir and Tsur 1999). This positive result, together with that on generating the spanning trees of a graph with polynomial delay (Read and Tarjan 1975), implies that SUBTREEISO is in P if G has polynomially many spanning trees only; just list all spanning trees τ of G and check if H is subgraph isomorphic to τ. Theorem 2 generalizes this straightforward positive result to graphs that can have *exponentially* many spanning trees. To prove Theorem 2, we present Algorithm 2 and show that it decides the SUBTREEISO problem correctly (Lemma 4) and in time stated in the theorem (Lemma 5).

Algorithm 2 is inspired by the ideas in Matoušek and Thomas (1992) and Shamir and Tsur (1999). Analogously to tree decompositions of bounded tree-width graphs (see, e.g., Diestel 2012), our dynamic programming algorithm splits G into certain induced subgraphs and evaluates partial (non-induced) subgraph isomorphisms from subtrees of H to such subgraphs. The evaluation order of our algorithm is, however, controlled by a rooted tree *skeleton* defined on the articulation vertices of G. For all nodes v of the tree skeleton, the blocks that contain v and are "below" v in G are replaced by a (local) spanning tree τ in all possible ways. The subproblem corresponding to v is then solved by carefully combining τ with the spanning trees considered in the previous level. Iterating over all (local) spanning trees of the blocks, we can correctly decide SUBTREEISO for the part of G that is "below" v. We will now describe the algorithm and necessary notation.

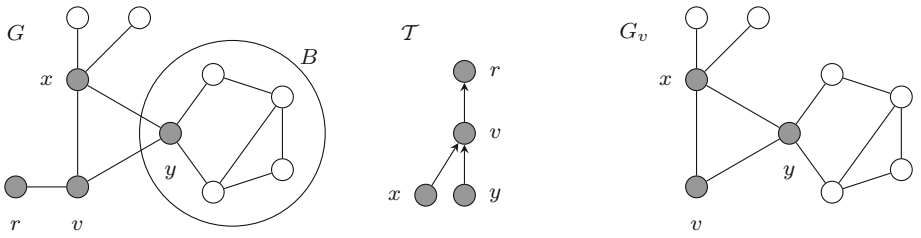


Fig. 1 **a** G, **b** tree skeleton $\mathcal{T}$, and **c** G_v for a small graph G (with respect to r). In (**a**), y is the root of the (cyclic) block B. Roots are shown in gray, while vertices that are not roots are shown in white

In what follows, H and G denote a tree and a graph, respectively. We assume w.l.o.g. that G is connected and that $2 \leq |V(H)| \leq |V(G)|$, implying that all blocks of G contain at least two vertices (i.e., a block is either cyclic or it is a bridge). We fix an arbitrary vertex $r \in V(G)$ and will implicitly also consider r, when talking about G. For a block B of G we define its *root* v to be the vertex of B with the smallest distance to r and will refer to B as a v-*rooted* block. Notice that the condition $|V(G)| \geq 2$ implies that r itself is also a root. For any $v \in V(G)$, the subgraph formed by the set of v-rooted blocks of G is denoted by $\mathcal{B}(v)$. Clearly, $\mathcal{B}(v)$ can be empty. On the set of roots of the blocks in G we define a directed graph $\mathcal{T}$ as follows (since G and r have been fixed, we omit them from the notation): For any $u, v \in V(\mathcal{T})$ with $u \neq v$, $(u, v) \in E(\mathcal{T})$ if and only if there exists a block B with root v such that $u \in V(B)$. We call $\mathcal{T}$ the *tree skeleton* of G (see, also, Fig. 1a, b). In the proposition below we show that $\mathcal{T}$ is indeed a rooted tree (i.e., a directed tree with edges directed towards the root). This tree will be used to direct our dynamic subgraph isomorphism algorithm.

Proposition 1 *$\mathcal{T}$ is a tree rooted at r.*

Proof It suffices to show that for all $u \in V(\mathcal{T})$ with $u \neq r$, u has outdegree at most 1; the claim then follows by noting that the outdegree of r is 0 and that $\mathcal{T}$ is connected, as G is connected. Suppose for contradiction that there exists $u \in V(\mathcal{T})$, $u \neq r$, with two different parents $v_1, v_2 \in V(\mathcal{T})$. Then there are $B_i \in \mathcal{B}(v_i)$ for $i = 1, 2$ that contain u. By maximality, B_1 and B_2 are edge disjoint. Furthermore, there is a path P_1 (resp. P_2) in G connecting r and v_1 (resp. v_2) that is edge disjoint with B_1 (resp. B_2). The union of P_1 and P_2 together with the paths connecting u with v_1 in B_1 and u with v_2 in B_2 contains a cycle intersecting both B_1 and B_2. But then u, v_1, and v_2 all belong to the same (cyclic) block of G, contradicting the maximality of B_1 and B_2. □

We need some further concepts. Let $v, w \in V(G)$. Then w is *below* v if all paths connecting r and w in G contain v. A *rooted subgraph* G_v of G for v is the subgraph of G induced by the set of vertices below v (see Fig. 1c) for an example). The same notation will be used consistently for the pair consisting of the tree pattern H and some vertex $y \in V(H)$, i.e., for any $u, y \in V(H)$, H_u^y is the tree obtained from the tree H rooted at y by keeping the subtree rooted at u. The definitions and the connectivity of G imply that G_v is connected, $G_r = G$, and G_w is a single vertex if and only if $w \notin V(\mathcal{T})$. A vertex $w' \in V(G)$ is called a *child* of v, if $vw' \in E(G)$ and $w' \in V(\mathcal{B}(v))$.

A *guidance tree* of G is a pair $\mathbb{T} = (\mathcal{T}, \mathcal{S})$ such that $\mathcal{T}$ is a tree skeleton of G and $\mathcal{S}$ is a family of sets S_v for all $v \in V(\mathcal{T})$. That is, all nodes v of $\mathcal{T}$ are associated with a set S_v, called

Algorithm 2 Subgraph Isomorphism from a Tree into a Connected Graph

Input : tree H with $|V(H)| > 1$ and an arbitrary connected graph G with $|V(G)| \geq |V(H)|$
Output: TRUE if $H \preccurlyeq G$; o/w FALSE

MAIN(H, G):
1: set $\mathcal{C} := \emptyset$
2: pick a vertex $r \in V(G)$ and compute the complete guidance tree $\mathbb{T} = (\mathcal{T}, \mathcal{S})$ of G for the tree skeleton $\mathcal{T}$ rooted at r
3: **for all** $v \in V(\mathcal{T})$ in a postorder **do**
4: **for all** $\tau \in S_v$ **do** // $S_v \in \mathcal{S}$ is the bag of v in $\mathbb{T}$
5: **for all** $w \in V(\tau)$ in a postorder **do**
6: $\mathcal{C} := \mathcal{C} \cup$ CHARACTERISTICS(v, τ, w)
7: **if** $(H_u^u, \tau, w) \in \mathcal{C}$ **then return** TRUE
8: **return** FALSE

FUNCTION CHARACTERISTICS(v, τ, w):
1: $\mathcal{C}_\tau := \emptyset$
2: **for all** $\theta \in \Theta_{vw}(\tau)$ **do**
3: **for all** $u \in V(H)$ **do**
4: let τ' be the tree satisfying $\theta = \tau \cup \tau'$
5: let C_τ (resp. $C_{\tau'}$) be the set of children of w in τ (resp. τ') and $C_\theta := C_\tau \cup C_{\tau'}$
6: let $B = \left(C_\theta \dot{\cup} \mathcal{N}(u), E\right)$ be the bipartite graph with

$$cu' \in E \iff (c \in C_\tau \wedge (H_{u'}^u, \tau, c) \in \mathcal{C}) \vee (c \in C_{\tau'} \wedge (H_{u'}^u, \tau', c) \in \mathcal{C})$$

for all $cu' \in C_\theta \times \mathcal{N}(u)$
7: **if** B has a matching that covers $\mathcal{N}(u)$ **then**
8: add (H_u^u, τ, w) to $\mathcal{C}_\tau$
9: **for all** $y \in \mathcal{N}(u)$ **do**
10: **if** B has a matching covering $\mathcal{N}(u) \setminus \{y\}$ **then**
11: add $\left(H_u^y, \tau, w\right)$ to $\mathcal{C}_\tau$
12: **return** $\mathcal{C}_\tau$

the *bag* of v. Each S_v is a subset of the set of spanning trees of $\mathcal{B}(v)$, called *local* spanning trees, all rooted at v. If S_v contains *all* spanning trees of $\mathcal{B}(v)$ for every $v \in V(\mathcal{T})$, then $\mathbb{T}$ is referred to as a *complete* guidance tree of G. Figure 2 shows an incomplete guidance tree for the graph from Fig. 1a. For the remainder of this section, by guidance trees we always mean complete guidance trees. (Incomplete guidance trees will be considered in Sect. 4.)

Let $\mathbb{T} = (\mathcal{T}, \mathcal{S})$ be a guidance tree of G, $v \in V(\mathcal{T})$, and H be a tree. An *iso-triple*[8] ξ of H relative to v is a triple (H_u^y, τ, w) such that $u \in V(H)$, $y \in \mathcal{N}(u) \cup \{u\}$, $\tau \in S_v$, and $w \in V(\tau)$. Let G' be an induced subgraph of G and τ be a spanning tree of G'. Then $G\{G'/\tau\}$ denotes the graph obtained from G by removing all edges of G' that are not in τ (i.e., by substituting G' with τ). Now we are able to define the partial subgraph isomorphisms we are interested in. A *v-characteristic* is an iso-triple $\xi = (H_u^y, \tau, w)$ relative to v such that there exists a subgraph isomorphism φ from H_u^y to $(G\{\mathcal{B}(v)/\tau\})_w$ with $\varphi(u) = w$. In the lemma below we provide a characterization of subgraph isomorphisms from H to G in terms of v-characteristics. Its proof follows directly from the definitions. (Recall that by guidance trees we mean complete guidance trees in this section.)

[8] Though our terminology is similar to that in Hajiaghayi and Nishimura (2007), which in turn is based on the concepts in Matoušek and Thomas (1992), the definitions of iso-triples and characteristics in this paper are semantically different from their definitions.

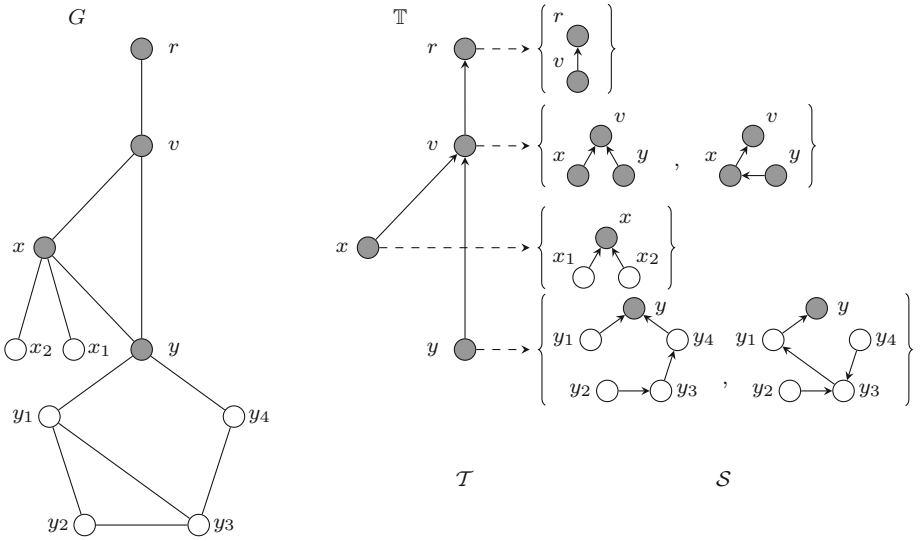


Fig. 2 A guidance tree $\mathbb{T} = (\mathcal{T}, \mathcal{S})$ for the graph G from Fig. 1a. $\mathbb{T}$ is incomplete; each bag contains at most two local spanning trees. Roots are shown in gray, while vertices that are not roots are shown in white

Lemma 1 *Let H be a tree, G be a graph with root r, and $\mathbb{T} = (\mathcal{T}, \mathcal{S})$ be a guidance tree of G such that $\mathcal{T}$ is rooted at r. Then $H \preccurlyeq G$ if and only if there exists a v-characteristic (H_u^u, τ, w) for some $v \in V(\mathcal{T})$, $u \in V(H)$, $\tau \in S_v$, and $w \in V(\tau)$.*

Notice that the number of v-characteristics (H_u^y, τ, w) is bounded by a polynomial in the number of local spanning trees τ. More precisely, there are $O(|V(H)| \cdot |V(\mathcal{B}(v))|)$ v-characteristics for each local spanning tree $\tau \in S_v$. We will show how these characteristics can be computed recursively by a post-order traversal of the tree skeleton $\mathcal{T}$. In order to recover all v-characteristics, the spanning trees of the w-rooted blocks must carefully be combined with τ when w itself is also a root (i.e., $w \in V(\mathcal{T})$). To formalize these considerations, we introduce the following notation. For any $v \in V(\mathcal{T})$, $\tau \in S_v$, and $w \in V(\tau)$ we define $\Theta_{vw}(\tau)$ by

$$\Theta_{vw}(\tau) := \begin{cases} \{\tau \cup \tau' : \tau' \in S_w\} & \text{if } w \in V(\mathcal{T}) \setminus \{v\} \\ \{\tau\} & \text{o/w (i.e., if } w \notin V(\mathcal{T}) \text{ or } v = w), \end{cases}$$

where $\tau \cup \tau'$ is the graph with vertex set $V(\tau) \cup V(\tau')$ and edge set $E(\tau) \cup E(\tau')$. That is, for the case that $w \in V(\mathcal{T}) \setminus \{v\}$, $\Theta_{vw}(\tau)$ is the set of trees obtained by "gluing" the local spanning tree τ and τ' at vertex w, for all local spanning trees $\tau' \in S_w$. The definition is correct, as $V(\tau) \cap V(\tau') = \{w\}$ for this case. Note that if w is a root vertex different from v then it always has at least one child in $\mathcal{B}(w)$, i.e., τ' is always a tree with at least one edge. As an example, the combination of the blue and the red tree in Fig. 3 denotes an element of $\Theta_{vw}(\tau)$. In Lemma 2 below we first provide a characterization of v-characteristics for subtrees H_u^y with $y \in \mathcal{N}(u)$.

Lemma 2 *Let H be a tree, G be a graph, and $\mathbb{T} = (\mathcal{T}, \mathcal{S})$ be a guidance tree of G. An iso-triple (H_u^y, τ, w) of H is a v-characteristic for some $v \in V(\mathcal{T})$ and $y \in \mathcal{N}(u)$ if and only if there exists a $\theta \in \Theta_{vw}(\tau)$ and an injective function ψ from $\mathcal{N}(u) \setminus \{y\}$ to the children*

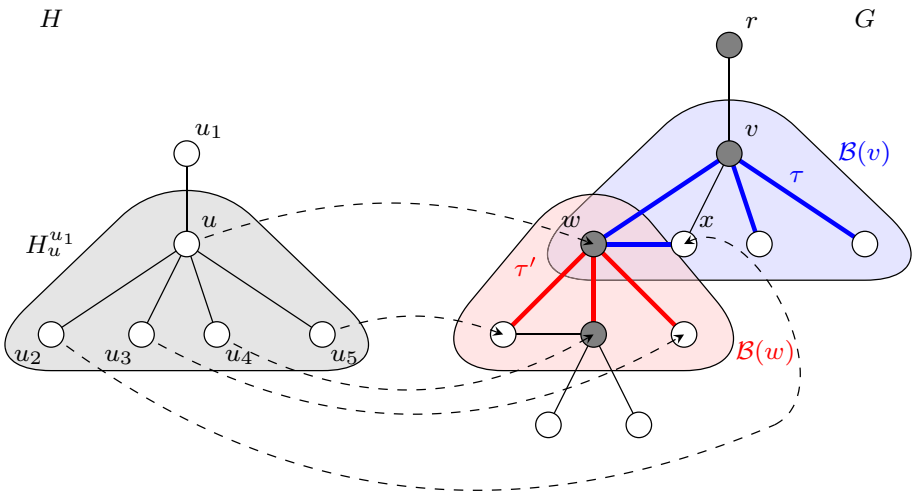


Fig. 3 This figure shows a small graph G with its subgraphs $\mathcal{B}(v)$ and $\mathcal{B}(w)$ (depicted by the rounded triangles) on the right. One spanning tree τ of $\mathcal{B}(v)$ and τ' of $\mathcal{B}(w)$ are shown in red and blue, respectively. The tree pattern H depicted on the left is subgraph isomorphic to G. The iso-triple $(H_u^{u_1}, \tau, w)$ is a v-characteristic, as there exists a subgraph isomorphism (depicted by the dashed lines) from $H_u^{u_1}$ to $\big(G\{\mathcal{B}(v) \cup \mathcal{B}(w)/\tau \cup \tau'\}\big)_w$. Note that the iso-triple $(H_u^{u_1}, \tau, w)$ is *not* a w-characteristic, as $x \notin V(\mathcal{B}(w))$ (Color figure online)

of w in θ such that for all $u' \in \mathcal{N}(u) \setminus \{y\}$ there is a subgraph isomorphism $\varphi_{u'}$ from $H_{u'}^{u}$ to $(G\{\mathcal{B}(v) \cup \mathcal{B}(w)/\theta\})_{\psi(u')}$ mapping u' to $\psi(u')$.

Proof "$\Rightarrow$" Suppose (H_u^y, τ, w) is a v-characteristic for some $v \in V(\mathcal{T})$ and $y \in \mathcal{N}(u)$. Then, by definition, there is a subgraph isomorphism φ from H_u^y to $(G\{\mathcal{B}(v)/\tau\})_w$ with $\varphi(u) = w$. Let R be an arbitrary spanning tree of $(G\{\mathcal{B}(v)/\tau\})_v$ containing the image $\varphi(H_u^y)$ as a subtree. Then $R[V(\mathcal{B}(w))] \in S_w$ and $R[V(\mathcal{B}(v))] = \tau$ and hence $\theta = R[V(\mathcal{B}(v))] \cup R[V(\mathcal{B}(w))] \in \Theta_{vw}(\tau)$ implying that for all $u' \in \mathcal{N}(u) \setminus \{y\}$, φ maps $H_{u'}^u$ to $(G\{\mathcal{B}(v) \cup \mathcal{B}(w)/\theta\})_{\varphi(u')}$. As φ is injective we can set ψ to be the restriction of φ to $\mathcal{N}(u) \setminus \{y\}$. As φ is a subgraph isomorphism, we can set $\varphi_{u'}$ to be the restriction of φ to $(G\{\mathcal{B}(v) \cup \mathcal{B}(w)/\theta\})_{\varphi(u')}$ for all $u' \in \mathcal{N}(u) \setminus \{y\}$.

"$\Leftarrow$" Let $\varphi : V(H_u^y) \to V((G\{\mathcal{B}(v)/\tau\})_w)$ with $\varphi : u \mapsto w$ and $x' \mapsto \varphi_{u'}(x')$ for all $u' \in \mathcal{N}(u) \setminus \{y\}$ and $x' \in V(H_{u'}^u)$. Since for all u', $\varphi_{u'}$ is at the same time a subgraph isomorphism from $H_{u'}^u$ to $(G\{\mathcal{B}(v)/\tau\})_w$, it holds that $\varphi_{u'}(u') = \psi(u')$. But then, as ψ is injective, φ is a subgraph isomorphism, implying the claim. □

In Lemma 3 we formulate an analogous characterization for the entire pattern H (i.e., for $y = u$). The proof of this lemma is similar to that of Lemma 2.

Lemma 3 *Let H, G, and $\mathbb{T} = (\mathcal{T}, \mathcal{S})$ be as in Lemma 2. An iso-triple (H_u^u, τ, w) of H is a v-characteristic for some $v \in V(\mathcal{T})$ if and only if there exists a $\theta \in \Theta_{vw}(\tau)$ and an injective function ψ from $\mathcal{N}(u)$ to the children of w in θ such that for all $u' \in \mathcal{N}(u)$ there is a subgraph isomorphism $\varphi_{u'}$ from $H_{u'}^u$ to $(G\{\mathcal{B}(v) \cup \mathcal{B}(w)/\theta\})_{\psi(u')}$ mapping u' to $\psi(u')$.*

Lemma 4 below is concerned with the correctness of Algorithm 2 deciding subtree isomorphism from a tree into an arbitrary text graph G. We assume without loss of generality that G is connected.

Lemma 4 (Correctness) *Algorithm 2 is correct, i.e., for all trees H and connected graphs G with $2 \leq |V(H)| \leq |V(G)|$, it returns* TRUE *if and only if $H \preccurlyeq G$.*

Proof Algorithm 2 first fixes a root r of G (Line 2 of MAIN) and computes the complete guidance tree $\mathbb{T} = (\mathcal{T}, \mathcal{S})$ of G, where $\mathcal{T}$ is rooted at r. By traversing $\mathcal{T}$ in a postorder manner (Line 3), it calculates the set $\mathcal{C}$ of v-characteristics for all $v \in V(\mathcal{T})$ (Lines 4–6). We only need to show that $\mathcal{C}$ is correct (i.e., complete and sound); the correctness of the algorithm then follows directly from Line 7 by Lemma 1.

The completeness of $\mathcal{C}$ holds by the fact that all possible iso-triples $\xi = (H_u^y, \tau, w)$ relative to v are tested for being v-characteristics (Lines 3, 4, and 5 of MAIN together with Lines 3, 7, and 9 of CHARACTERISTICS). Thus, it remains to show that it is decided correctly whether or not $\xi = (H_u^y, \tau, w)$ is a v-characteristic. We prove this by double induction on the height $h_{\mathcal{T}}(v)$ of v in $\mathcal{T}$ and on the height $h_\tau(w)$ of w in τ. Depending on whether or not $h_\tau(w) = 0$ and $h_{\mathcal{T}}(v) = 0$, four cases can be distinguished. We only show the base case $h_{\mathcal{T}}(v) = h_\tau(w) = 0$, denoted ($\alpha$), and the most general case $h_{\mathcal{T}}(v) > 0$ and $h_\tau(w) > 0$, denoted (β), by noting that the proofs of the other two cases can be shown by an argumentation similar to the one used for the most general case.

(α) For the base case $h_{\mathcal{T}}(v) = h_\tau(w) = 0$ we have $C_\theta = \emptyset$ and hence $B = (\emptyset \dot{\cup} \mathcal{N}(u), \emptyset)$ (Lines 5 and 6 of CHARACTERISTICS). Applying Lemma 2 to this case, ξ is a v-characteristic if and only if $\mathcal{N}(u) = \{y\}$, which, in turn, holds if and only if there is a matching covering $\mathcal{N}(u) \setminus \{y\}$ in B (Lines 10–11 of CHARACTERISTICS), as there are no edges in B.

(β) If $h_{\mathcal{T}}(v) > 0$ and $h_\tau(w) > 0$ then $C_\tau \neq \emptyset$. Two cases can be distinguished:

(i) If $w \notin V(\mathcal{T})$ then $C_{\tau'} = \emptyset$ and thus $C_\theta = C_\tau$. Applying Lemma 2 to this case, ξ is a v-characteristic if and only if there exists an injective function $\psi : \mathcal{N}(u) \setminus \{y\} \to C_\tau$ such that for all $u' \in \mathcal{N}(u) \setminus \{y\}$, there exist a child c of w in τ (i.e., $c \in C_\tau$) and a subgraph isomorphism $\varphi_{u'}$ from $H_{u'}^u$ to $(G\{\mathcal{B}(v)/\tau\})_c$ with $\varphi_{u'}(u') = \psi(u') = c$ (i.e., a v-characteristic $(H_{u'}^u, \tau, c)$). By the induction hypothesis, the bipartite graph B is constructed correctly in Line 6 of CHARACTERISTICS, and hence ψ exists if and only if there exists a matching in B covering $\mathcal{N}(u) \setminus \{y\}$.

(ii) If $w \in V(\mathcal{T})$ then $C_\theta = C_\tau \cup C_{\tau'}$ with $C_\tau, C_{\tau'} \neq \emptyset$. Then, by Lemma 2, ξ is a v-characteristic if and only if for all $u' \in \mathcal{N}(u) \setminus \{y\}$ there exist a child c of w in θ and an injective function $\psi : \mathcal{N}(u) \setminus \{y\} \to C_\tau \cup C_{\tau'}$ such that there is a subgraph isomorphism $\varphi_{u'}$ from $H_{u'}^u$ to $(G\{\mathcal{B}(v) \cup \mathcal{B}(w)/\tau \cup \tau'\})_c$ with $\varphi_{u'}(u') = \psi(u') = c$. Such a subgraph isomorphism either corresponds to a v-characteristic $(H_{u'}^u, \tau, c)$ for $c \in C_\tau$, which has already been computed by the induction hypothesis on $h_\tau(w)$, or to a w-characteristic $(H_{u'}^u, \tau', c)$ for $c \in C_{\tau'}$, which has already been computed by the induction hypothesis on $h_{\mathcal{T}}(v)$. Hence ψ exists if and only if a matching in B (constructed in Line 6 of CHARACTERISTICS) covering $\mathcal{N}(u) \setminus \{y\}$ exists (Lines 10–11 of CHARACTERISTICS).

The proof for the v-characteristics (H_u^u, τ, w) using Lemma 3 is analog for the test in Lines 7–8 of CHARACTERISTICS.

□

In Lemma 5 below we show that the runtime of Algorithm 2 is polynomial in the combined size of H, G, and

$$f(\mathbb{T}) = \max_{S_v \in \mathcal{S}} |S_v|, \tag{2}$$

where $\mathbb{T} = (\mathcal{T}, \mathcal{S})$ is the guidance tree of G computed in Line 2 of Algorithm 2. Together with Lemma 4 this implies Theorem 2 by noting that $f(\mathbb{T}) \leq f_{\max}(G)$, where $f_{\max}(G)$ is defined in (1). It is immediate from the definition, that $f(\mathbb{T})$ cannot be larger than $f_{\max}(G)$. It can, however, be strictly smaller: As an example, consider Fig. 3, where $\mathcal{B}(w)$ contains only a subset of the cyclic blocks containing w (i.e., it does not contain the block induced by the vertices w, x, v). Here, $f(\mathbb{T}) = 3$ and $f_{\max}(G) = 9$.

Lemma 5 (Runtime) *Algorithm 2 runs in* $O\left(f^2(\mathbb{T}) \cdot |E(G)| \cdot |V(H)|^{1.5}\right)$ *time.*

Proof Note that the edge sets of the v-rooted blocks of G form a partition of $E(G)$, i.e.,

$$E(G) = \dot{\bigcup_{v \in V(\mathcal{T})}} E(\mathcal{B}(v)) \,. \tag{3}$$

This partition and the tree skeleton $\mathcal{T}$ can be computed in linear time (Tarjan 1972). By definition, $|S_v| \leq f(\mathbb{T})$ for all $v \in V(\mathcal{T})$. Thus, as the spanning trees of a graph can be generated with linear delay (Read and Tarjan 1975), S_v can be computed in $O\left(|E(\mathcal{B}(v))| \cdot f(\mathbb{T})\right)$ time for each $v \in V(\mathcal{T})$. Hence, by (3), Main spends altogether

$$O\left(|E(G)| \cdot f(\mathbb{T})\right) \tag{4}$$

time for computing the guidance tree $\mathbb{T}$. Furthermore, it calls subroutine Characteristics only

$$O\left(\sum_{w \in V(G)} f(\mathbb{T})\right) \tag{5}$$

times because the number of pairs (v, w) considered in Lines 3 and 5 is $O\left(|V(G)|\right)$. Indeed, each vertex w can occur in at most two sets of rooted blocks: In $\mathcal{B}(v)$ for its unique parent v in $\mathcal{T}$ (unless $w = r$) and in $\mathcal{B}(w)$ if w is a root itself. Regarding the complexity of Characteristics, note that $|\Theta_{vw}(\tau)|$ is bounded by $f(\mathbb{T})$ for any $\tau \in S_v$ (see Line 2 of Characteristics) and that the bipartite graph B constructed in Line 6 has at most $|\mathcal{N}(u)| + |\mathcal{N}(w)|$ vertices for any $\theta \in \Theta_{vw}(\tau)$.

The edges of B can be constructed by membership queries to $\mathcal{C}$. We can implement the set $\mathcal{C}$ of characteristics found by the algorithm as a multidimensional array of polynomial size (in $f(\mathbb{T})$ and $|V(G)|$) such that each look-up and storage operation can be performed in constant time. A maximum matching of B can be found in $O\left(|\mathcal{N}(w)| \cdot |\mathcal{N}(u)|^{1.5}\right)$ time (Hopcroft and Karp 1973, Thm. 3). Applying the same trick as in Chung (1987) and Shamir and Tsur (1999) for ordinary subtree isomorphism, we can answer the matching queries for u and all of its neighbors in Lines 7 and 10 of Characteristics using a single bipartite matching computation and an additional operation that is linear in the size of B. Hence, for a vertex $w \in V(G)$, function Characteristics runs in time

$$O\left(f(\mathbb{T}) \cdot |\mathcal{N}(w)| \cdot \sum_{u \in V(H)} |\mathcal{N}(u)|^{1.5}\right) \subseteq O\left(f(\mathbb{T}) \cdot |\mathcal{N}(w)| \cdot |E(H)|^{1.5}\right) \tag{6}$$

$$= O\left(f(\mathbb{T}) \cdot |\mathcal{N}(w)| \cdot |V(H)|^{1.5}\right) , \tag{7}$$

where (6) follows from the handshaking lemma and (7) from the fact that H is a tree. Thus, by (5) and by another application of the handshaking lemma to G, together with (4) we obtain

an overall time complexity

$$O\left(f(\mathbb{T})\left(|E(G)| + f(\mathbb{T}) \cdot |E(G)| \cdot |V(H)|^{1.5}\right)\right)$$

which, in turn, is equal to

$$O\left(f^2(\mathbb{T}) \cdot |E(G)| \cdot |V(H)|^{1.5}\right) \tag{8}$$

as claimed. □

Note that in the case that H and G are both trees, $f(\mathbb{T}) = 1$ and hence (8) corresponds to the time complexity of the ordinary subtree isomorphism algorithms given in Chung (1987) and Matula (1968). We will address the implications of this algorithm for probabilistic and exact frequent tree mining in the next two sections.

4 Probabilistic frequent subtree mining

In Welke et al. (2018) we introduced the concept of *probabilistic frequent subtrees*, a random subset of frequent trees, and presented an algorithm enumerating this kind of sound, but incomplete pattern set with polynomial delay. It is based on replacing each graph in the input with a forest formed by the vertex disjoint union of a random subset of its spanning trees. On the one hand, the more spanning trees are considered by the algorithm, the higher the recall of its output is. On the other hand, however, its delay depends linearly on the number of spanning trees, implying that in order to guarantee polynomial delay it can consider at most *polynomially* many spanning trees per graph. In this section we show that the results from Sect. 3 allow us to go beyond this limitation. In particular, in Sect. 4.1 we propose a *boosted* probabilistic frequent subtree mining algorithm that, using a variant of Algorithm 2, implicitly considers *exponentially* many spanning trees for the transaction graphs and still guarantees polynomial delay. In Sect. 4.2 we empirically compare its performance to that of the simple algorithm in Welke et al. (2018).

4.1 The boosted algorithm

Recall that Algorithm 2 decides the SubTreeIso problem by splitting the input transaction graph G into certain induced subgraphs and by considering the set of *all* local spanning trees for all such induced subgraphs. In case it takes *not* all, but only some subsets of the local spanning trees, its output becomes correct only with respect to the subset of global spanning trees of G that can be constructed by "gluing" together the local spanning trees considered in all possible ways. In Theorem 3 below we formulate a straightforward extension of Theorem 2 to this more general setting of the SubTreeIso problem.

To state this result, we need the following notion. Let $\mathbb{T} = (\mathcal{T}, \mathcal{S})$ be an arbitrary (i.e, not necessarily complete) guidance tree of G with bag $S_v \in \mathcal{S}$ for all $v \in V(\mathcal{T})$ and consider the graph T with $V(T) = V(G)$ and $E(T) = \bigcup_{v \in V(\mathcal{T})} E(\tau_v)$, where $\tau_v \in S_v$ for all $v \in V(\mathcal{T})$. The definitions imply that T is a spanning tree of G. Hence, the disjoint union of all such spanning trees of G, i.e, which can be obtained by taking all possible combinations of the local spanning trees in the bags, forms a forest. We denote this forest by $\mathfrak{S}(\mathbb{T})$. We are ready to formulate a generalization of Theorem 2 to arbitrary (i.e., incomplete) guidance trees (see (2) for the definition of $f(\mathbb{T})$):

Algorithm 3 Subgraph Isomorphism from a Tree with One-Sided Error

Input : tree H with $|V(H)| > 1$ and guidance tree $\mathbb{T} = (\mathcal{T}, \mathcal{S})$ for some connected graph G with $|V(G)| \geq |V(H)|$
Output: TRUE if $H \preccurlyeq \mathfrak{S}(\mathbb{T})$; o/w FALSE

MAIN$(H, \mathbb{T})$:
1: set $\mathcal{C} := \emptyset$
2: **for all** $v \in V(\mathcal{T})$ in a postorder **do**
3: **for all** $\tau \in S_v$ **do** // $S_v \in \mathcal{S}$ is the bag of v in $\mathbb{T}$
4: **for all** $w \in V(\tau)$ in a postorder **do**
5: $\mathcal{C} := \mathcal{C} \cup$ CHARACTERISTICS(v, τ, w) // see Algorithm 2
6: **if** $(H^u_u, \tau, w) \in \mathcal{C}$ **then return** TRUE
7: **return** FALSE

Theorem 3 *Let H be a tree, G be a graph, and $\mathbb{T} = (\mathcal{T}, \mathcal{S})$ be a guidance tree of G. Then one can decide whether $H \preccurlyeq \mathfrak{S}(\mathbb{T})$ in time*

$$O\left(f^2(\mathbb{T}) \cdot |E(G)| \cdot |V(H)|^{1.5}\right) .$$

Proof Consider Algorithm 3 for the modified pseudocode of MAIN given in Algorithm 2, using the same subroutine CHARACTERISTICS. Its input includes $\mathbb{T} = (\mathcal{T}, \mathcal{S})$, instead of G. (Line 2 of MAIN in Algorithm 2 is accordingly removed.) The proofs of Lemma 4 and Lemma 5 immediately apply to the partial sets of local spanning trees as well, implying the correctness and the runtime with respect to $\mathfrak{S}(\mathbb{T})$. □

Note that Theorem 2 in the previous section is the special case of the theorem above that $\mathbb{T}$ is a *complete* guidance tree. Furthermore, Theorem 3 is formulated for deciding subgraph isomorphism from trees into arbitrary graphs with one-sided error. That is, if Algorithm 3 returns "YES", then the answer is always correct; o/w it may happen that there exists a spanning tree T of G such that $H \preccurlyeq T$, but $H \not\preccurlyeq \mathfrak{S}(\mathbb{T})$. This property holds also for the algorithm in Welke et al. (2018), which guarantees efficiency by explicitly considering a *polynomial* number of (random) *global* spanning trees of G. The importance of the result in Theorem 3 above is that it guarantees polynomial time already for the case that the number of *local* spanning trees in the bags of $\mathbb{T}$ is bounded by a *polynomial* of G which, in turn, may however implicitly represent *exponentially* many *global* spanning trees in $\mathfrak{S}(\mathbb{T})$ (cf. Sect. 5 for a straightforward example of this case). This result may be of some independent interest. Theorem 3 gives rise to the following positive result on efficient mining of frequent subtrees (without loss of generality, we formulate it for connected transaction graphs):

Theorem 4 *Let D be a finite set of connected graphs, $\mathbb{T}_G = (\mathcal{T}_G, \mathcal{S}_G)$ be a guidance tree of G for all $G \in D$, and let D' be the set of forests defined by $D' = \{\mathfrak{S}(\mathbb{T}_G) : G \in D\}$. Then for any positive frequency threshold, the set of frequent subtrees of D' can be generated with delay polynomial in the combined size of the original dataset D and $f(D') = \max_{G \in D} f(\mathbb{T}_G)$.*

Proof The correctness and irredundancy are immediate from Theorem 1. Regarding the complexity, we can apply the arguments used in the proof of Theorem 1 in Horváth and Ramon (2010) to the setting considered in the theorem. They imply that the dominating term in the delay is the complexity of the pattern matching operator. By Theorem 3, it is polynomial in $f(D')$ and the size of D, as claimed. □

Clearly, for all positive frequency thresholds, any frequent subtree of D' is at the same time a frequent subtree of D as well. (The reverse direction does not hold for potential

incompleteness.) For the particular case, which is in the focus of this section, that the bags in $\mathbb{T}_G$ are some *random* subsets of the corresponding sets of all local spanning trees, frequent subtrees of D' will be referred to as *probabilistic* frequent subtrees. We note that this definition is different from the one introduced in Welke et al. (2018). Applying Theorem 4 to this case we have that probabilistic frequent subtrees can be listed with polynomial delay in the size of D, whenever $f(D')$ is bounded by a polynomial in the size of D. We now discuss some algorithmic and implementation issues concerning the generation of such random bags.

For the algorithmic aspects of sampling local spanning trees we note that a spanning tree of any graph B can be generated uniformly at random in *expected* time $O\left(|V(B)|^3\right)$ using the algorithm of Wilson (1996). We can improve on this time and achieve a deterministic algorithm with $O\left(|E(B)| \cdot \log(|V(B)|)\right)$ runtime if the spanning trees are not required to be drawn uniformly at random. Indeed, just pick a random permutation of the edge set and apply Kruskal's minimum spanning tree algorithm using this edge order (for a detailed discussion see Welke et al. 2018).

Regarding the practical implementation of this algorithm, we note that sampling the spanning trees is actually never the dominating term. Following the idea of Theorem 4, instead of sampling local spanning trees anew for each invocation of the embedding operator, we select a root for all $G \in D$ in a preprocessing step and consider the corresponding tree skeleton $\mathcal{T}$ of G. For each $v \in V(\mathcal{T})$ we sample, with replacement, l spanning trees of the v-rooted blocks, where $l \in \mathbb{N}$ is some user specified parameter. In case of sampling multiple identical local spanning trees for v-rooted blocks, we keep only one copy to speed up the algorithm. In particular, if all v-rooted blocks are bridges for some $v \in \mathcal{T}$, then the graph induced by the v-rooted blocks is a tree. In this case, we can safely just use this tree once, instead of sampling l identical spanning trees without changing the set of computed v-characteristics. We call such a root *trivial*.

The *global* spanning trees in $\mathfrak{S}(\mathbb{T})$ above, considered implicitly by our algorithm, are *random*. They are generated neither uniformly nor independently from the set of all spanning trees of G, even if we sample the local spanning trees uniformly and independently at random. This is due to the fact that any random local spanning tree picked for a non-trivial root contributes to at least two spanning trees in $\mathfrak{S}_G$, whenever G (with respect to the fixed root r) has at least two non-trivial roots. Our experimental results in Sect. 4.2 below however show that despite this kind of dependency, the recall increases by increasing values of l.

4.2 Experimental evaluation

In this section we experimentally demonstrate the advantage of using *local* spanning trees instead of *global* ones in probabilistic frequent subtree mining. In what follows we will refer to the former technique as *boosted probabilistic subtree* (BPS) and to the latter one as *probabilistic subtree* (PS) mining. In particular, we show for different values of t that within time t, BPS considers a dramatically larger number of spanning trees per graph on average compared to PS, resulting in an improvement in terms of recall of frequent subtrees.

Our experiments clearly indicate that the amount of improvement strongly depends on the structural properties of the transaction graphs at hand. In particular, the improvement obtained for molecular graphs of small pharmacological compounds is negligible; we observed this consistently on several such benchmark graph datasets. As already mentioned, most *exact* frequent pattern mining algorithms have an excellent performance on this kind of graphs, with GASTON (Nijssen and Kok 2005) being notably the fastest. However, all these exact methods seem to be limited to this particular graph class, as they were unable to produce any

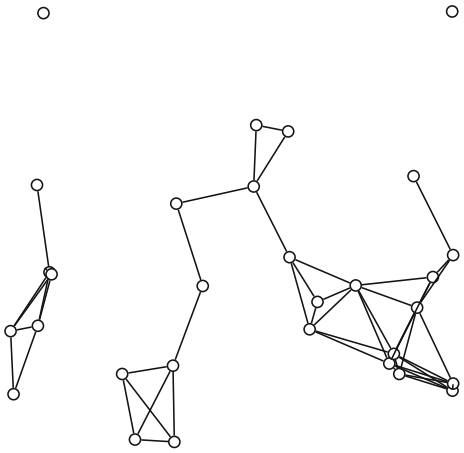


Fig. 4 A threshold graph on 30 points in the 2D Euclidean unit square for $d = 0.2$

frequent patterns in feasible time, even for slightly more complex structures beyond molecular graphs. In particular, for small neighborhood graphs extracted from social networks, none of the existing implementations were able to return any frequent patterns. In contrast, already PS could consistently produce an output having such a high recall[9] of frequent patterns that make BPS unnecessary for this other kind of graphs. This is due to the fact that such neighborhood graphs contain typically a single block only and hence, BPS and PS behave similarly on them.

If, however, the transaction graphs have exponentially many spanning trees *and* several cyclic blocks at the same time, then PS is able to consider only a small fraction of all spanning trees, implying a negative impact on the recall. Such situations occur, for example, in case of *threshold graphs*, which are defined by local neighborhood relationships between objects in a metric space. Two vertices representing two objects are connected by an edge if and only if the distance of the corresponding objects is smaller than some given threshold (see Fig. 4 for a threshold graph on 30 two-dimensional points). This kind of graphs have different practical applications, for example in spectral clustering (cf. von Luxburg 2007). While in that application field there are only rules of thumb on how to choose a suitable threshold for a particular metric and clustering task, one is interested in threshold graphs having a high edge density within each cluster and a low one among the clusters. This requirement typically results in threshold graphs having multiple cyclic blocks that are connected by a few bridges only and hence, in a large number of spanning trees. To demonstrate the advantage of BPS over PS, we have therefore considered threshold graphs in our experiments.

In particular, we evaluate our methods on artificial graph data sets that simulate the two-dimensional Brownian motion over time. To construct such a graph database, we first draw n points from the unit square $\{(x, y) : 0 \leq x, y \leq 1\}$ independently and uniformly at random and label them with c different labels[10] at random for some $c > 0$ integer. Given a parameter $d \in (0, \sqrt{2}]$, we construct a *threshold graph* by connecting two points if and only if their Euclidean distance is at most d. Subsequent graphs in the database are obtained by (i) moving

[9] More precisely, we could calculate only a lower bound using Sloane (2016) and found that already it was very close to 1. We omit the details and refer the reader to Welke (2019) for a detailed overview and discussion of these results.

[10] The number of vertex labels has a non-trivial influence on the number of (non-isomorphic) spanning trees of graphs and also on the number of frequent patterns in a graph database.

each point randomly according to a normal distribution with standard deviation μ centered at its former position and (ii) constructing a threshold graph on the resulting set of points with respect to the same threshold d as above. If a point would leave the unit square due to its random move, it is reflected back inside. Hence, a database constructed in this way depends on the parameters n, c, d, μ, and N, where N is the number of time steps (or equivalently, the number of graphs in the database). To obtain the dataset, we generated $N = 200$ graphs with $n = 30$ vertices with $c \in \{2, 5, 10, 30\}$ random colors (i.e., vertex labels). We set $d = 0.2$ and $\mu = 0.02$, as the threshold graphs induced by these numbers fulfilled the desirable structural properties discussed above.

We first compare the average number of spanning trees and non-isomorphic spanning trees considered by the PS and BPS methods. As the resulting graphs may be disconnected (see, e.g., Fig. 4), we extend our algorithms to this case as follows: We compute the number of *different*[11] spanning trees considered for each connected component separately, sum them up, and normalize the result by the number of connected components. To obtain the number of non-isomorphic spanning trees for each graph, we compute a canonical string for each tree in the above two sets, count the number of different strings, and again normalize by the number of connected components of the graph. Notice that the average number of non-isomorphic spanning trees calculated in this way can be smaller than 1 (e.g. when G has many singleton vertices with the same label).

Table 1 shows the average number of sampled spanning trees and that of non-isomorphic spanning trees for the threshold graph dataset defined above for PS and BPS. One can see that for all $k \in \{2, 5, 10, 30\}$, both the average number of sampled spanning trees and the resulting non-isomorphic spanning trees is much higher for BPS. For example, for 30 labels and $k = 10$ we get on average only 4.51 different spanning trees and 4.06 non-isomorphic spanning trees for PS. On the other hand, BPS considers on average 2606.08 different and 349.90 non-isomorphic spanning trees, when sampling $k = 10$ local spanning trees. In order to obtain a similar number of non-isomorphic spanning trees on average with PS, one would need to sample at least 350 global spanning trees per graph.

An interesting observation is that the fraction of non-isomorphic spanning trees to different trees considered is rather different for PS and BPS. While for PS almost all sampled trees are non-isomorphic, this fraction drops to below 20% for BPS and larger values of k. We do not know whether this is because the overall number of non-isomorphic spanning trees is rather small or because of the fact that the combination of local spanning trees results in many "similar" global spanning trees due to the dependency. We assume the latter by stressing that the average number of non-isomorphic spanning trees is still much higher than what can be achieved with a reasonable parameter k for PS.

Finally, we investigate the recall of frequent subtree patterns that can be obtained in a given time budget. That is, we fix a (low) frequency threshold $\theta = 2\%$ (corresponding to the absolute frequency threshold of 4) and mine (probabilistic) frequent subtrees on the threshold graph database for increasing values of k until the algorithm exceeds a runtime budget of 200 s. For a given value of k and for both methods PS and BPS, we repeat the mining algorithm ten times and average runtime and recall to mitigate for the effects of the random samples. Figure 5 shows the number of frequent patterns found (y-axis) per time (x-axis) for increasing values of the sampling parameter k. BPS obtains a significantly higher number of frequent patterns per time than PS for all time budgets up to 200 s.[12] For example, for 10 vertex colors (i.e., $c = 10$), we obtain on average 73,396 patterns in 195 s for $k = 59$ using PS and 101,503

[11] Here, two spanning trees T, T' are identical if and only if $E(T) = E(T')$, *not* if they are isomorphic.

[12] Note that according to this definition, the plots in Fig. 5 can end before $x = 200$.

Table 1 Average number of spanning trees considered by PS (Welke et al. 2018) and BPS

k	2 labels		5 labels		10 labels		30 labels	
	PS	BPS	PS	BPS	PS	BPS	PS	BPS
2	1.44	4.89	1.44	4.98	1.43	4.81	1.44	4.74
	0.94	2.57	0.98	2.59	0.98	2.65	0.99	2.62
3	1.87	29.91	1.87	32.44	1.86	28.13	1.86	31.66
	1.33	7.89	1.40	9.15	1.41	8.63	1.41	9.68
4	2.27	118.78	2.27	102.48	2.27	112.12	2.26	100.03
	1.70	21.44	1.78	23.82	1.80	29.69	1.81	24.81
5	2.66	243.88	2.66	243.44	2.66	230.08	2.65	265.21
	2.04	38.58	2.16	44.67	2.19	53.97	2.20	53.69
6	3.04	510.26	3.03	465.82	3.04	490.62	3.05	498.13
	2.40	63.13	2.53	77.55	2.56	91.49	2.60	99.82
7	3.42	865.82	3.43	880.51	3.42	789.28	3.41	883.88
	2.73	99.16	2.91	117.07	2.93	129.55	2.96	141.80
8	3.79	1364.81	3.77	1382.15	3.80	1306.57	3.77	1231.39
	3.06	147.61	3.24	161.30	3.31	183.91	3.32	187.95
9	4.16	1996.29	4.17	1979.02	4.14	1888.02	4.15	1816.81
	3.38	200.06	3.62	251.41	3.65	257.12	3.70	277.92
10	4.51	2717.93	4.51	2744.65	4.51	2868.81	4.51	2606.08
	3.79	260.56	3.97	326.28	4.02	364.06	4.06	349.90

For each number k of sampled global (resp. local) spanning trees for PS (resp. BPS) we report the average number of sampled spanning trees per connected component in the first row and the resulting average number of *non-isomorphic* spanning trees per connected component in the second row

patterns for $k = 36$ for BPS, a 38.3% increase. Comparing the runtimes necessary to obtain a given amount of frequent patterns, this difference gets even more concrete. To obtain at least the same number of frequent patterns returned by PS in at most 200 s, BPS needs only 148.77 s, 106.74 s, 109.52 s, and 124.38 s, for 2, 5, 10, and 30 vertex colors, respectively. Thus, on transaction graphs consisting of several dense cyclic blocks, such as, for example, threshold graphs, BPS clearly has a superior performance over PS.

5 Exact frequent subtree mining

Theorems 1 and 2 give rise to the characterization of a new *non-trivial* graph class beyond forests for which the FTM problem can be solved with polynomial delay. We will now formally define this graph class and investigate some of its properties. Recall that frequent subtrees can be mined efficiently in forest databases, or more generally, in graphs having polynomially many spanning trees; this follows from the results e.g. in Chi et al. (2005) and Horváth and Ramon (2010). Such graphs will be referred to as *easy* graphs. Except for forests, the class of easy graphs is typically uninteresting from a practical viewpoint, as even for relatively simple graphs beyond forests, the number of spanning trees usually grows *exponentially* with the number of vertices. Our positive result extends to this practically and theoretically more interesting situation by requiring easiness *not* for the entire graph, but only for *local* surroundings of the vertices. Formally, a graph G with n vertices is *locally*

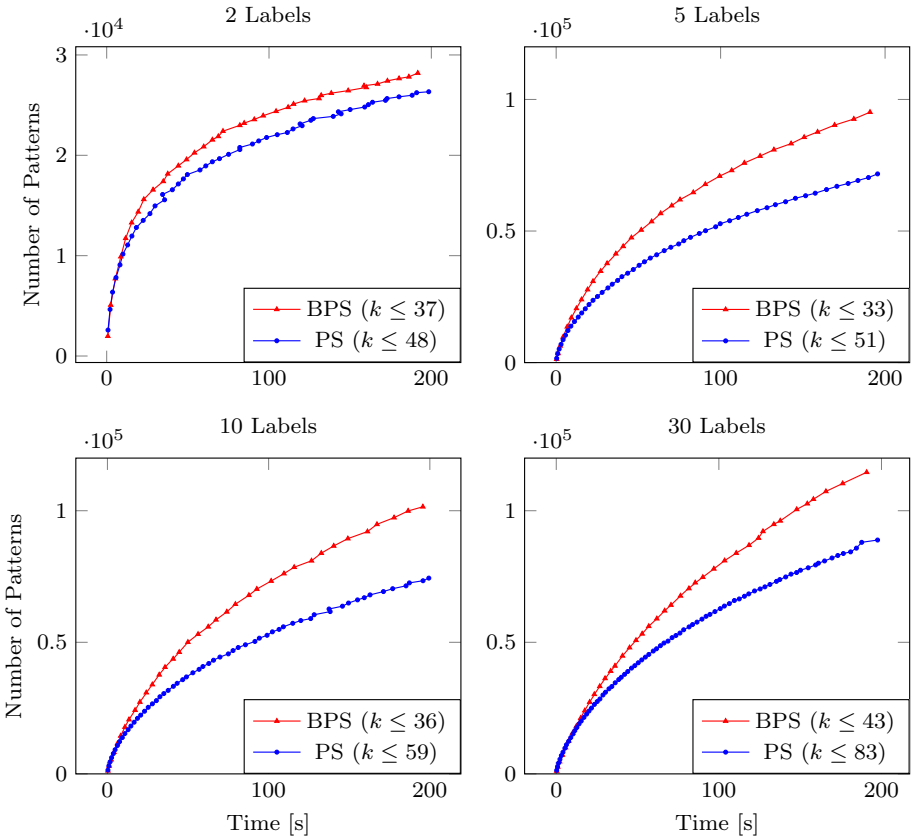


Fig. 5 Recall curves for 2%-frequent subtrees on the threshold graph database for PS and BPS for different numbers of vertex colors ranging from 2 to 30. Each dot corresponds to the average of 10 runs of the respective algorithms for some given value of k, ranging from 1 to the number indicated in the legends

easy if for all $v \in V(G)$, the number of spanning trees of the union formed by the blocks of G containing v is bounded by a polynomial of n, i.e., $f_{\max}(G) = O(\text{poly}(n))$ (cf. (1) in Sect. 3 for the definition of $f_{\max}(G)$). In particular, for the case that it is bounded by $p(n)$ for some polynomial p (resp. by n) we will speak of *locally p-easy* (resp. *locally linearly easy*) graphs. Clearly, all easy graphs are locally easy, but a locally easy graph may contain exponentially many spanning trees (see Fig. 6 for an example). We have the following result:

Theorem 5 *The FTM problem can be solved with polynomial delay for locally easy transaction graphs.*

Proof By Theorem 1, Algorithm 1 solves the FTM problem for locally easy transaction graphs with polynomial delay whenever all conditions required are fulfilled. Conditions 1 and 2 of Theorem 1 are straightforward when $\mathcal{P}$ is restricted to trees and Condition 3 follows e.g. from the results of Shamir and Tsur (1999). Finally, Theorem 2 immediately implies Condition 4 from tree patterns into locally easy graphs. □

Below we discuss some important properties of locally easy graphs implying the theoretical and practical importance of Theorem 5 above.

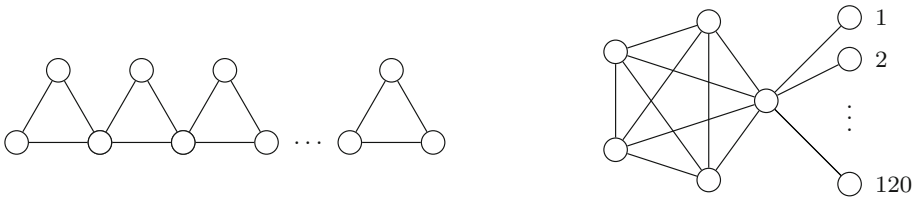


Fig. 6 A locally easy graph with exponentially many spanning trees on the left and a locally linearly easy graph of tree-width 4 on the right

(P1) The *membership* problem for locally easy graphs (i.e., whether a graph is locally easy or not) can be decided in cubic time, implying that it can be checked in polynomial time for any graph database D whether or not Theorem 5 is applicable to D. More precisely, let G be a graph with $|V(G)| = n$ and p be some polynomial. One can decide in cubic time whether G is locally p-easy by performing the following steps: (i) Compute first the set of all blocks of G, (ii) calculate the number of spanning trees for all blocks of G separately, and (iii) check for all $v \in V(G)$ whether the product of these values for all blocks sharing v is at most $p(n)$. The claim above then follows by noting that (i) can be solved in linear (Tarjan 1972) and (ii) in cubic time using Kirchhoff's theorem (see, e.g., Chap. 5.6 in Stanley and Fomin 1999).

(P2) Locally easy graphs may contain *exponentially* many spanning trees. As an example, consider the graph G given in the left-hand side of Fig. 6. It is locally linearly easy (for all $v \in V(G)$ there are at most 9 spanning trees in the union of the (cyclic) blocks containing v), still it has altogether $3^{O(n)}$ spanning trees. This and other examples show that our result formulated in Theorem 5 is non-trivial, as any brute-force pattern matching algorithm that decides whether a tree is subgraph isomorphic to a locally easy graph G by testing subtree isomorphism for *all* spanning trees of G becomes infeasible for such cases.

(P3) The class of locally easy graphs contains some interesting graph classes for which the FTM problem is computationally tractable. As an example, we mention the class of almost k-trees of bounded degree, where a graph G is an *almost k-tree* for some integer $k \geq 0$ if $|E(B)| \leq |V(B)|+k$ for all blocks B of G. One can decide in polynomial time whether a tree is subgraph isomorphic to an almost k-tree of bounded degree (Akutsu 1993). Combining this result with Theorem 1 we have that the FTM problem can be solved with polynomial delay for almost k-trees of bounded degree. We can obtain this result directly by Theorem 5 as well because the class of locally easy graphs properly contains that of almost k-trees of bounded degree. The strength of Theorem 5 is that it generalizes the positive mining result above also to almost k-trees of *unbounded* degree that are locally easy.

(P4) The class of locally easy graphs is *"orthogonal"* to all graph classes that are defined by a constant upper bound on some monotone graph property. To formalize this statement, we need some further definitions: A *nested hierarchy* of the class of all finite graphs is a family of graph classes $\mathcal{H} = \{\mathcal{G}_i : i \geq 0\}$ such that for all finite graphs G there exists a non-negative integer i with $G \in \mathcal{G}_i$ and $\mathcal{G}_j \subsetneq \mathcal{G}_{j+1}$ for all $j \geq 0$. The smallest integer i satisfying $G \in \mathcal{G}_i$ is denoted by $I_{\mathcal{H}}(G)$. A nested hierarchy $\mathcal{H}$ is *monotone* if $I_{\mathcal{H}}(G_1) \leq I_{\mathcal{H}}(G_2)$ whenever $G_1 \preccurlyeq G_2$, for all graphs G_1, G_2. The graph parameters size, order, maximum vertex degree, tree-width, number of spanning trees of a graph are some straightforward examples inducing monotone nested hierarchies; an example for some $I_{\mathcal{H}}$ resulting in a *non-monotone* nested hierarchy would be the number of connected components or blocks. Using the above concepts, we are ready to formulate the following claim:

Claim *For any monotone nested hierarchy* $\mathcal{H} = \{\mathcal{G}_i : i \geq 0\}$ *and for any integer* $k \geq 0$ *there are infinitely many locally linearly easy graphs that are not in* $\mathcal{G}_k$.

Proof Let $\mathcal{H}$ be a monotone nested hierarchy and let $G_1, G_2, G_3, \ldots$ be a sequence of graphs with $I_{\mathcal{H}}(G_i) = i$. Such a sequence exists by definition. For any $i \in \mathbb{N}$, adding new leafs to G_i does not decrease $I_{\mathcal{H}}(G_i)$, as $\mathcal{H}$ is monotone. However, it will eventually decrease the local easiness of the resulting graph: Recall from (1) in Sect. 3 that for all $i \geq 0$, $f_{\max}(G_i)$ is the maximum number of spanning trees in the union of the blocks of G_i containing v, over all $v \in V(G_i)$. By adding $\max(0, f_{\max}(G_i) - |V(G_i)|)$ new leafs (i.e., vertices of degree 1) to G_i in an arbitrary way, we obtain a locally linearly easy graph G'_i with $I_{\mathcal{H}}(G'_i) \geq i$ and $f_{\max}(G'_i) = f_{\max}(G_i)$ for all $i \geq 0$. Thus, for any $k \in \mathbb{N}$, $G'_{k+j} \notin \mathcal{G}_k$ for all $j \geq 1$, implying the claim. □

We illustrate the idea in the proof above on the monotone nested hierarchy induced by the tree-width (see, e.g., Diestel 2012, for the definition of tree-width). Consider the graph G obtained from the complete graph K_k on k vertices for some $k \geq 3$ by adding $k^{k-2} - k$ leafs to some vertex of K_k (see the right-hand side of Fig. 6 for an example with $k = 5$). On the one hand, the construction does not increase the tree-width, i.e., the tree-width of G is equal to that of K_k. On the other hand, as K_k has exactly k^{k-2} spanning trees by Cayley's formula, G is a locally linearly easy graph. Since the construction in this example holds for any $k \geq 3$, local easiness implies no constant upper bound on the tree-width.

The choice of tree-width in the example above is especially interesting because frequent subtrees of *bounded degree* can be generated with polynomial delay from graphs of bounded tree-width. This follows from Theorem 1 together with the positive result of Matoušek and Thomas (1992) on subgraph isomorphism between bounded tree-width graphs. This and other examples provide evidence that our main result formulated in Theorem 5 extends (or complements) several results on the (fixed parameter) tractability of the FTM problem for various monotone nested hierarchies for which subgraph isomorphism from a tree can be decided in polynomial time. We note, for example, that in the systematic overview of the parameterized complexity of subgraph isomorphism by Marx and Pilipczuk (2014), 9 out of the 10 parameters considered result in monotone nested hierarchies. Hence, our result extends the positive results in their work to the case that the patterns are restricted to trees.

(P5) A large fraction of the molecular graphs considered in chemoinformatics are actually locally easy. To confirm this observation, we first provide a sufficient condition for local easiness. Let G be a graph of size n and let $c, k \geq 0$ be integers. Then G is *degree-k easy* if each block of G has at most $O(n^k)$ spanning trees and it is of *cyclic block degree-c* if each vertex v of G belongs to at most c distinct cyclic blocks.[13] Clearly, if G is degree-k easy and of cyclic block degree-c for some constants k and c, then G is locally easy.

Many of the chemical graphs of pharmacological compounds are d-tenuous outerplanar graphs for $d \leq 5$ (Horváth et al. 2010). Informally, each cyclic block of such a graph is a planar graph composed of a single Hamiltonian cycle and at most d non-crossing diagonals. Clearly, d-tenuous outerplanar graphs are degree-$(d + 1)$ easy. Furthermore, chemical graphs have typically some very small cyclic block degree because they have small vertex degree. Thus, most chemical graphs are locally easy. To support this claim experimentally, we investigated local easiness for the graphs in the ZINC dataset.[14] Our version of the database contains 8, 946, 757 "lead-like" compounds. We have the following distribution of the molecules with respect to $f_{\max}(G)$ defined in (1):

[13] Note that the vertex degree is an upper bound on the cyclic block degree.

[14] Obtained from http://zinc.docking.org.

0	$\leq$	$f_{\max}(G)$	$<$	n	8,640,166	(96.57%)
n	$\leq$	$f_{\max}(G)$	$<$	n^2	302,541	(3.38%)
n^2	$\leq$	$f_{\max}(G)$	$<$	n^3	1864	(0.02%)
n^3	$\leq$	$f_{\max}(G)$			2186	(0.02%)

Thus, by our result in Theorem 5, all frequent trees can be generated from almost all such chemical graphs with polynomial delay. This complements the positive result of Horváth et al. (2010) on mining frequent connected subgraphs from d-tenuous outerplanar graphs with respect to a constrained subgraph isomorphism operator.

6 Concluding remarks

The results described in this paper raise several interesting practical and theoretical issues for further studies. In particular, as we demonstrated on threshold graphs in Sect. 4.2, the technique presented can be used to efficiently improve the recall of probabilistic frequent subtrees by considering exponentially many spanning trees. While the amount of improvement is impressive for threshold graphs and for other potential graph classes satisfying the structural properties discussed in Sect. 4, it is marginal e.g. for chemical or small neighborhood graphs extracted from social networks. This raises the practical question whether one can design an algorithm able to decide quickly for any transaction database whether probabilistic frequent subtrees should be generated by sampling global (cf. Welke et al. 2018) or rather local (cf. Sect. 4) spanning trees.

It would be interesting to understand how far the positive result of this work on exact frequent subtree mining can be generalized to other pattern classes beyond trees. Perhaps the first natural question towards this direction would be to ask whether it is possible to generate frequent *locally easy* subgraphs in *locally easy* transaction graphs with polynomial delay. In order to calculate the v-characteristics for a root vertex v with respect to a vertex u in the pattern, our algorithm combines at most two sets of local spanning trees at any time and assumes that neither u nor the vertices in its local environment are contained in a cycle. Therefore, in order to apply the algorithm to the more general patterns of locally easy graphs, we need to work with the spanning trees of certain local environments of u. However, in contrast to the transaction graphs, it may happen that such spanning trees are composed of the combination of the spanning trees of the blocks for a *non-constant* number of root vertices of the pattern graph. In such a case, an exponential number of spanning trees must be processed. This indicates that, if it is possible at all, such a generalization would require some more sophisticated approach.

An Open Problem Finally we give arguments clearly indicating the significance and difficulty of generalizing the positive result in Theorem 5 to transaction graphs beyond locally easy graphs. We suspect that obtaining such a generalization is at least as hard as solving the millennium problem P versus NP. In particular, it is natural to ask whether frequent subtrees can be generated with *polynomial delay* also from transaction graphs for which we only require the number of spanning trees per block to be bounded by a polynomial in the size of the whole graph (i.e., we do not assume any constant upper bound on the cyclic block degree). In contrast to locally easy graphs, subgraph isomorphism from trees into this type of more general graphs becomes NP-complete, even for the very simple class of *cactus* graphs (i.e., in which each cyclic block is a simple cycle, Akutsu 1993). We do not know the answer to the question above, not even to the case of cactus transaction graphs. We can, however, show the

importance and high difficulty of this open problem by discussing the potential two answers separately:

(i) Suppose the problem *can* be solved with polynomial delay. An important immediate consequence of this result would be that polynomial delay frequent pattern enumeration is possible even for NP-complete pattern matching operators, solving an open problem (cf. Horváth and Ramon 2010).

(ii) Suppose it *cannot* be solved with polynomial delay. Then, as the class of trees satisfies Conditions 1–3 of Theorem 1, by contraposition we have that Condition 4 of Theorem 1 does *not* hold, i.e., the corresponding subgraph isomorphism problem is *not* in P. But this would immediately imply that P $\neq$ NP, indicating the high difficulty of proving this case, as the subgraph isomorphism problem lies in NP for all pattern and text graph classes. Note that this consideration applies also to the particular case of cactus transaction graphs.

We conjecture that case (ii) holds, that is, polynomial delay pattern generation is impossible for computationally intractable pattern matching operators. This is certainly true for graph classes for which the Hamiltonian path problem is NP-complete (assuming P $\neq$ NP).[15] If our conjecture is true, then

(a) in case of intractable pattern matching operators, the primary question should be whether the pattern mining problem at hand can be solved in incremental polynomial time, rather to prove that polynomial delay pattern mining is not possible and

(b) even for very simple graph classes, *the cyclic block degree of the transaction graphs is a crucial parameter for polynomial delay frequent pattern mining*.

Acknowledgements Part of this work has been funded by the Ministry of Education and Research of Germany (BMBF) under project ML2R (grant number 01/S18038C).

References

Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., & Verkamo, A.I. (1996). Fast discovery of association rules. In Advances in Knowledge Discovery and Data Mining (pp. 307–328). AAAI/MIT Press.

Akutsu, T. (1993). A polynomial time algorithm for finding a largest common subgraph of almost trees of bounded degree. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, *76*(9), 1488–1493.

Arnborg, S., Corneil, D. G., & Proskurowski, A. (1987). Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic Discrete Methods*, *8*(2), 277–284. https://doi.org/10.1137/0608024.

Bringmann, B., Zimmermann, A., De Raedt, L., & Nijssen, S. (2006). Don't be afraid of simpler patterns. In J. Fürnkranz, T. Scheffer, & M. Spiliopoulou (Eds.), *European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD) Proceedings, Lecture Notes in Computer Science* (Vol. 4213, pp. 55–66). Springer. https://doi.org/10.1007/11871637_10.

Chi, Y., Muntz, R. R., Nijssen, S., & Kok, J. N. (2005). Frequent subtree mining—An overview. *Fundamenta Informaticae*, *66*(1–2), 161–198.

Chung, M. J. (1987). $O(n^{2.5})$ time algorithms for the subgraph homeomorphism problem on trees. *Journal of Algorithms*, *8*(1), 106–112. https://doi.org/10.1016/0196-6774(87)90030-7.

Cordella, L. P., Foggia, P., Sansone, C., & Vento, M. (1999). Performance evaluation of the VF graph matching algorithm. In *International Conference on Image Analysis and Processing (ICIAP)* (pp. 1172–1177). IEEE Computer Society. https://doi.org/10.1109/ICIAP.1999.797762.

Deshpande, M., Kuramochi, M., Wale, N., & Karypis, G. (2005). Frequent substructure-based approaches for classifying chemical compounds. *Transactions on Knowledge and Data Engineering*, *17*(8), 1036–1050. https://doi.org/10.1109/tkde.2005.127.

[15] We note that the Hamiltonian path problem is polynomial for the case of cactus graphs, making them an especially interesting candidate graph class.

Diestel, R. (2012). *Graph theory, Graduate texts in mathematics* (4th ed., Vol. 173). Berlin: Springer.
Erdős, P., & Rényi, A. (1959). On random graphs I. *Publicationes Mathematicae*, *6*, 290–297.
Garey, M. R., & Johnson, D. S. (1979). *Computers and intractability: A guide to the theory of NP-completeness*. New York: W. H. Freeman.
Hajiaghayi, M., & Nishimura, N. (2007). Subgraph isomorphism, log-bounded fragmentation, and graphs of (locally) bounded treewidth. *Journal of Computer and System Sciences*, *73*(5), 755–768. https://doi.org/10.1016/j.jcss.2007.01.003.
Han, J., Pei, J., Yin, Y., & Mao, R. (2004). Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Mining and Knowledge Discovery*, *8*(1), 53–87. https://doi.org/10.1023/b:dami.0000005258.31418.83.
Hopcroft, J. E., & Karp, R. M. (1973). An n^5/2 algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, *2*(4), 225–231. https://doi.org/10.1137/0202019.
Horváth, T., Bringmann, B., & Raedt, L. D. (2007). Frequent hypergraph mining. In S. Muggleton, R. P. Otero, & A. Tamaddoni-Nezhad (Eds.), *Inductive Logic Programming (ILP) Revised Selected Papers, Lecture Notes in Computer Science* (Vol. 4455, pp. 244–259). Berlin: Springer. https://doi.org/10.1007/978-3-540-73847-3_26.
Horváth, T., & Ramon, J. (2010). Efficient frequent connected subgraph mining in graphs of bounded tree-width. *Theoretical Computer Science*, *411*(31–33), 2784–2797. https://doi.org/10.1016/j.tcs.2010.03.030.
Horváth, T., Ramon, J., & Wrobel, S. (2010). Frequent subgraph mining in outerplanar graphs. *Data Mining and Knowledge Discovery*, *21*(3), 472–508. https://doi.org/10.1007/s10618-009-0162-1.
Johnson, D. S., Papadimitriou, C. H., & Yannakakis, M. (1988). On generating all maximal independent sets. *Information Processing Letters*, *27*(3), 119–123. https://doi.org/10.1016/0020-0190(88)90065-8.
Kramer, S., Lavrač, N., & Flach, P. (2001). Propositionalization approaches to relational data mining. In S. Džeroski & N. Lavrač (Eds.), *Relational data mining* (pp. 262–291). Berlin: Springer. https://doi.org/10.1007/978-3-662-04599-2_11.
Kuramochi, M., & Karypis, G. (2004). An efficient algorithm for discovering frequent subgraphs. *Transactions on Knowledge and Data Engineering*, *16*(9), 1038–1051. https://doi.org/10.1109/TKDE.2004.33.
Mannila, H., & Toivonen, H. (1997). Levelwise search and borders of theories in knowledge discovery. *Data Mining and Knowledge Discovery*, *1*(3), 241–258. https://doi.org/10.1023/a:1009796218281.
Marx, D., & Pilipczuk, M. (2014). Everything you always wanted to know about the parameterized complexity of Subgraph Isomorphism (but wereafraid to ask). In E. W. Mayr & N. Portier (Eds.), *International Symposium on Theoretical Aspects of Computer Science (STACS), Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, LIPIcs* (Vol. 25, pp. 542–553). https://doi.org/10.4230/LIPIcs.STACS.2014.542.
Matoušek, J., & Thomas, R. (1992). On the complexity of finding iso-and other morphisms for partial *k*-trees. *Discrete Mathematics*, *108*(1–3), 343–364. https://doi.org/10.1016/0012-365x(92)90687-b.
Matula, D. W. (1968). An algorithm for subtree identification. *Siam Review*, *10*, 273–274.
Nijssen, S., & Kok, J. N. (2005). The gaston tool for frequent subgraph mining. *Electronic Notes in Theoretical Computer Science*, *127*(1), 77–87. https://doi.org/10.1016/j.entcs.2004.12.039.
Read, R. C., & Tarjan, R. (1975). Bound on backtrack algorithms for listing cycles, paths, and spanning trees. *Networks*, *5*, 237–252.
Robertson, N., & Seymour, P. D. (1986). Graph minors. II. Algorithmic aspects of tree-width. *Journal of Algorithms*, *7*(3), 309–322. https://doi.org/10.1016/0196-6774(86)90023-4.
Shamir, R., & Tsur, D. (1999). Faster subtree isomorphism. *Journal of Algorithms*, *33*(2), 267–280. https://doi.org/10.1006/jagm.1999.1044.
Sloane, N. J. A. (2016). *The online encyclopedia of integer sequences. A000055: Number of trees with n unlabeled nodes*. http://oeis.org/A000055. Accessed 18 November 2016.
Stanley, R. P., & Fomin, S. (1999). *Enumerative combinatorics, Cambridge Studies in Advanced Mathematics* (Vol. 2). Cambridge: Cambridge University Press. https://doi.org/10.1017/CBO9780511609589.
Tarjan, R. (1972). Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, *1*(2), 146–160.
Ullmann, J. R. (1976). An algorithm for subgraph isomorphism. *Journal of the ACM*, *23*(1), 31–42. https://doi.org/10.1145/321921.321925.
von Luxburg, U. (2007). A tutorial on spectral clustering. *Statistics and Computing*, *17*(4), 395–416. https://doi.org/10.1007/s11222-007-9033-z.
Welke, P. (2019). *Efficient frequent subtree mining beyond forests*. Ph.D. thesis, University of Bonn.
Welke, P., Horváth, T., & Wrobel, S. (2015). On the complexity of frequent subtree mining in very simple structures. In J. Davis & J. Ramon (Eds.), *Inductive Logic Programming (ILP) Revised Selected Papers, Lecture Notes in Computer Science* (Vol. 9046, pp. 194–209). Berlin: Springer. https://doi.org/10.1007/978-3-319-23708-4_14.

Welke, P., Horváth, T., & Wrobel, S. (2018). Probabilistic frequent subtrees for efficient graph classification and retrieval. *Machine Learning*, *107*(11), 1847–1873. https://doi.org/10.1007/s10994-017-5688-7.

Wilson, D.B. (1996). Generating random spanning trees more quickly than the cover time. In: G.L. Miller (Ed.) *ACM Symposium on the Theory of Computing (STOC) Proceedings* (pp. 296–303). ACM. https://doi.org/10.1145/237814.237880.

Zhao, P., & Yu, J. X. (2008). Fast frequent free tree mining in graph databases. *World Wide Web*, *11*(1), 71–92. https://doi.org/10.1007/s11280-007-0031-z.