On the Complexity of Frequent Subtree Mining in Very Simple Structures

Pascal Welke[1], Tamás Horváth[1,2], and Stefan Wrobel[2,1]

[1] Dept. of Computer Science, University of Bonn, Germany
[2] Fraunhofer IAIS, Schloss Birlinghoven, Sankt Augustin, Germany

Abstract. We study the complexity of frequent subtree mining in very simple graphs beyond forests. We show for d-tenuous outerplanar graphs that frequent subtrees can be listed with polynomial delay if the *cycle degree*, i.e., the maximum number of blocks that share a common vertex, is bounded by some *constant*. The crucial step in the proof of this positive result is a polynomial time algorithm deciding subgraph isomorphism from trees into d-tenuous outerplanar graphs of *bounded* cycle degree. We obtain this algorithm by generalizing the algorithm of Shamir and Tsur that decides subgraph isomorphism between trees. Our results may also be of some interest to algorithmic graph theory, as they indicate that even for very simple structures, the cycle degree is a crucial parameter for the tractability of subgraph isomorphism. We also discuss some interesting problems towards generalizing the positive result of this work.

1 Introduction

One of the central problems of graph mining is the *frequent connected subgraph mining* (FCSM) problem defined as follows: *Given* a set D of graphs, called *transaction graphs*, from some graph class $\mathcal{G}$ and some integral frequency threshold $t > 0$, *generate* all connected graphs that are subgraph isomorphic to at least t graphs in D. We require the patterns in the output to be unique up to isomorphism.

This problem is of high relevance to Inductive Logic Programming. Indeed, each transaction graph $G \in D$ can be regarded as an intensional database D_G over a relational vocabulary consisting of a set of constants, a set of unary, and a set of binary predicate symbols corresponding to the vertices, the vertex labels, and the edge labels of the graphs in D, respectively. Furthermore, all patterns P can be viewed as goal clauses, or equivalently, as Boolean conjunctive queries over the same vocabulary. Since for any such P and D_G, the first-order logical implication $P \models D_G$ becomes equivalent to subsumption between the clauses corresponding to P and D_G [7], which, in turn, is equivalent to relational homomorphism (cf. [14]), the FCSM problem can be regarded as a special case of the problem of generating all frequent Boolean conjunctive queries in the learning from interpretations setting with the additional constraint that the logical consequence (or pattern matching) operator is defined by subsumption under object identity (see, e.g., [4, 6]).

The FCSM problem has been intensively studied over the past two decades by the data mining community, with the primary goal of designing practical algorithms for different problem settings. Despite the intensive research effort, its complexity aspects are still not well understood. For example, there are only a few non-trivial positive results about special cases of the FCSM problem when frequent connected subgraphs are required to be generated with polynomial delay. Polynomial delay pattern enumeration means that the delay between generating two consecutive patterns is bounded by a polynomial in the size of D. One of the well-known such cases is that the transaction graphs are restricted to forests [3]. The goal of this work is to generalize this positive result. In particular, we concentrate on the following question that, to the best of our knowledge, has not been investigated so far: *How far can we go beyond forests when insisting on polynomial delay frequent pattern generation?* We study this question by looking at very simple transaction and pattern classes.

In particular, we focus on a subclass of *d-tenuous outerplanar* graphs. A graph is *outerplanar* if it can be drawn in the plane in a such way that edges are allowed to intersect each other only in their endpoints and all vertices can be reached from outside without crossing any edge. An outerplanar graph is *d-tenuous* if it has at most d diagonals per block. This later definition uses the fact that all blocks (i.e., biconnected components with more than one edge) of an outerplanar graph can be drawn in the plane as a convex polygon containing some non-crossing diagonals. We note that d-tenuous outerplanar graphs form a practically relevant graph class. For example, the molecular graphs of most pharmacological compounds are d-tenuous outerplanar graphs for some small d [13].

The main result of this paper is that frequent subtrees can be listed with polynomial delay from d-tenuous outerplanar graphs of bounded *cycle degree*. The cycle degree of a graph is the maximum number of blocks a vertex can belong to. Our mining algorithm is an adaptation of a generic levelwise search algorithm to d-tenuous outerplanar graphs of bounded cycle degree. The most complex part of the algorithm is the pattern matching subroutine. It utilizes that any block of a d-tenuous outerplanar graph has polynomially many spanning trees only. This property, combined with bounded cycle degree, allows for an efficient algorithm deciding subgraph isomorphism. Though a d-tenuous outerplanar graph of bounded cycle degree may have exponentially many spanning trees, we show that any instance of our more general problem can be decomposed with a divide-and-conquer strategy into polynomially many instances of deciding subgraph isomorphism between trees, which can be solved in polynomial time [18]. Our technique requires an efficient combination of the blocks' spanning trees, as well as a careful assembly of certain partial subgraph isomorphisms.

To put our result in context, we first note that frequent connected subgraph mining is possible with polynomial delay if the pattern matching is restricted to a constrained subgraph isomorphism that maps bridges to bridges and different blocks to different blocks [13]. This directly implies that frequent subtrees can be listed with polynomial delay if D consists of forests (see, also, [3]). For

bounded tree-width graphs, a proper generalization of forests, frequent subtrees can be generated in incremental polynomial time [12] for ordinary subgraph isomorphism. That is, a next new frequent subtree for this graph class can be computed in time polynomial in the combined size of the input and the set of patterns already computed. Beyond forests, however, it is an open question whether polynomial delay mining of frequent subtrees is possible even for graphs of bounded tree-width, if the vertex degree is not bounded by some constant. In fact, the question remains open even for the very simple case of 0-tenuous outerplanar graphs, also known as *cactus* graphs. An affirmative answer to this question would immediately solve the open problem whether polynomial delay pattern generation is possible for NP-hard pattern matching operators, as it is an NP-complete problem to decide whether a tree is subgraph isomorphic to a cactus graph [2]. On the other hand, as shown in Section 4.1, a negative answer would imply that P $\neq$ NP. Regarding the two cases above, we conjecture that polynomial delay frequent pattern generation is *not* possible for computationally intractable pattern matching operators; this is certainly the case for graph classes for which the Hamiltonian path problem is NP-complete. If this conjecture holds then the results of this paper imply that *even for very simple structures, the cycle degree of graphs is a crucial parameter* that must be taken into account when designing efficient frequent subgraph mining algorithms. Furthermore, polynomial delay generation of frequent acyclic Boolean conjunctive queries (i.e., Datalog goal clauses) is possible only for very simple relational structures in the learning from interpretation setting with subsumption under object identity [4, 6].

The rest of the paper is organized as follows. In Section 2 we collect the necessary background and then describe our mining algorithm in Section 3. In Section 4 we conclude and discuss some interesting problems towards generalizing our results. Due to space limitations we omit the proof of the correctness and only sketch the runtime of our pattern matching algorithm.

2 Notions

In this section we fix the terminology and notation used in the paper. We first recall some standard notions from graph theory (see, e.g., [5]) and then, in Section 2.1, formally define the mining problem considered in this work.

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 and denote an edge $\{u, v\} \in E(G)$ by uv. The set of neighbors of a vertex v is denoted 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 some symbols from a finite set. Examples of labeled graphs include

molecular graphs, protein-protein interaction graphs, the Web graph etc. To keep the notation and description concise, we will state all our results for unlabeled graphs by noting that all our arguments apply to labeled graphs as well.

Two graphs G_1, G_2 are *isomorphic*, if there is a bijection $\varphi : V(G_1) \to V(G_2)$ such that $uv \in E(G_1)$ iff $\varphi(u)\varphi(v) \in E(G_2)$ 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 isomorphic to G_1.

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 $k > 0$. In this paper we will concentrate on the class of outerplanar graphs; a graph is *outerplanar*, if (i) it can be drawn in the plane in such a way that edges do not cross each other except maybe in their endpoints and (ii) every vertex lies on the outer face. That is, each vertex can be reached from outside without crossing any edge. A graph G is outerplanar if and only if all its biconnected components are outerplanar [10]. A *biconnected component* B of an outerplanar graph is either a single edge not belonging to a cycle, called *bridge*, or it is a maximal induced subgraph of G composed of a single Hamiltonian cycle and possibly some diagonal edges [10]. Biconnected components of the later type will be referred to as *blocks*. A *d-tenuous* outerplanar graph is an outerplanar graph in which each block has at most d diagonals. Notice that forests are special outerplanar graphs.

2.1 The Frequent Connected Subgraph Mining Problem

In this work, we study special cases 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. [12]), we regard a more general problem *parameterized* by the pattern class. Though in this paper we are interested in the case that $\mathcal{P}$ is the class of trees, we use here this more general problem setting. The reason is that in Section 3.1 we give an algorithm for this generic problem and state conditions guaranteeing efficient pattern mining. These conditions may be of some independent interest.

Note that the problem above is a listing problem. For such problems, the following complexity classes are distinguished in the literature (see, e.g., [15]). Suppose an algorithm $\mathfrak{A}$ for the FCSM problem gets D and t as input and outputs a list $O = [p_1, p_2, \cdots, p_n]$ of patterns. Then $\mathfrak{A}$ generates O

- with *polynomial delay*, if the time before the output of p_1, between the output of 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 $size(D) + size(O)$.

Clearly, polynomial delay implies incremental polynomial time. It is an open problem whether the two classes are identical, or not. In frequent itemset mining, for example, the FP-Growth algorithm [9] lists frequent patterns with polynomial delay, while the Apriori algorithm [1] in incremental polynomial time. Our special focus in this work is to establish sufficient conditions for polynomial delay pattern mining.

3 The Mining Algorithm

In this section we present our algorithm mining frequent tree patterns in d-tenuous outerplanar graphs of bounded cycle degree with polynomial delay. We denote the transaction graph class to be considered by $\mathcal{O}_{d,c}$, where c is an upper bound on the cycle degree of the graphs it contains. More precisely, let G be a graph and $\mathcal{B}(G)$ the set of its blocks. Then the number of distinct blocks containing a vertex $v \in V(G)$, denoted by $\delta_C(v)$, is called the *cycle degree* of v. That is, $\delta_C(v) := |\{B \in \mathcal{B}(G) : v \in V(B)\}|$. We define the cycle degree of a graph to be the maximum cycle degree over all its vertices. Notice that the *vertex degree* of the graphs in $\mathcal{O}_{d,c}$ can be unbounded.

The class $\mathcal{O}_{d,c}$ is not only of theoretic, but also of practical interest e.g. in computational chemistry. For example, for the NCI2012 dataset[3] consisting of 265,242 molecular graphs we have the following cycle degree distribution:

cycle degree	#graphs
0	21,727
1	238,218
2	5,099
3	196
4	2

Additionally, most molecular graphs have an outerplanar graph structure with a small number of diagonals per block [13].

3.1 A Generic Levelwise Search Mining Algorithm

We obtain our positive result by adapting a generic levelwise search mining algorithm to our problem setting. Levelwise search [1] is one of the most popular techniques in pattern mining that can be used to efficiently mine frequent patterns in a broad range of problem settings. In order to find a pattern in level $l+1$, it completely explores all levels up to l. On the one hand, this is disadvantageous if one is interested in frequent patterns only from level l for some large l, on the other hand, in frequent subgraph mining it allows for an incremental polynomial time pattern generation even for NP-complete pattern matching operators [12].

[3] cactus.nci.nih.gov/download/nci, Release 4 File Series

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: A generic levelwise graph mining algorithm.

Algorithm 1 is a generic levelwise search algorithm for the FCSM problem. It is a slight modification of a related algorithm in [12]; 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. Clearly, $|\rho(P)| \in O\left(|V(P)|^2\right)$. Subroutine supportCount(H, D) in Line 8 returns the number of graphs $G \in D$ with $H \preccurlyeq G$.

It is shown in [12] that the original version of Algorithm 1 mines frequent patterns with polynomial delay if patterns and transactions satisfy certain conditions. These conditions have however been formulated for the case that the pattern and transaction graph classes are the same. In the theorem below we generalize these conditions to the case that they can be different. Due to space limitation, we only sketch the proof.

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 $H' \preccurlyeq H$.*
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 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. Let P be a graph. If $P \notin \mathcal{P}$ or P is not frequent in D then it cannot be part of the output of Alg. 1 (see Lines 6 and 8, respectively). Now let $P \in \mathcal{P}$ be frequent in D. If P has more than one vertex, there is a vertex with degree one or there is an edge that can be removed without disconnecting P. Inductively, we get a sequence of edge (and isolated vertex) removals that in reverse order is a sequence of refinements generating P from a single vertex. By Condition 1, every connected subgraph of P is in $\mathcal{P}$ and is also frequent. Thus, by induction, Algorithm 1 generates and outputs P. The condition in Line 6 ensures that the algorithm outputs any pattern P at most once up to isomorphism and that P will not be processed if $P \notin \mathcal{P}$. Thus the LevelwiseGraphMining algorithm is correct.

Regarding its runtime, the time between starting and outputting the first pattern (or termination if there is no frequent single vertex) is linear in the size of the database. We can count the frequency of single vertices by a single scan over the database. We now show that the time needed for Lines 6–9 is polynomial in the size of D. Conditions 1–3 imply that there is a canonical string representation for all graphs in $\mathcal{P}$ that can be computed in polynomial time. We can store $\mathcal{S}_i$ and $\mathcal{C}_i$ as prefix trees of canonical strings of patterns. In this way, we can add and look up patterns in $\mathcal{S}_i$ or $\mathcal{C}_i$ in time linear in the size of the canonical string of a pattern. $|\rho(P)|$ is polynomial in the size of P and thus, in the size of D. Therefore, by Condition 2, $\rho(P) \cap \mathcal{P}$ can be computed in polynomial time. $H \notin \mathcal{C}_{l+1}$ can be checked in time linear in the size of the canonical string representation of H. supportCount can be implemented by iterating over D, checking for each graph $G \in D$ if $H \preccurlyeq G$, and maintaining a counter; by Condition 4 it runs in polynomial time. Overall, the time between printing consecutive patterns and the time between printing the last pattern and termination is polynomial in the size of D. □

Notice that the conditions above allow for generating 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 later property when restricting $\mathcal{P}$ to trees. For this case, one can check that $\mathcal{P}$ satisfies the first three conditions (Conditions 1 and 2 are straightforward for trees and Condition 3 follows e.g. from [18]). Thus, to show that tree patterns can be mined with polynomial delay in any database of graphs from some graph class $\mathcal{G}$, we only need to show that Condition 4 holds for trees and for the graphs in $\mathcal{G}$. We state this for $\mathcal{G} = \mathcal{O}_{d,c}$ in Theorem 3 in Section 3.2, immediately implying our main result for this work:

Theorem 2. *All frequent subtrees can be generated with polynomial delay in transaction databases from $\mathcal{O}_{d,c}$ for constant d and c.*

3.2 The Algorithm Deciding Subgraph Isomorphism

The main result of this section is formulated in Theorem 3. Due to space limitations, we omit all proofs and describe only the algorithm implying Theorem 3.

Theorem 3. *For any tree H and $G \in \mathcal{O}_{d,c}$, it can be decided in polynomial time whether $H \preccurlyeq G$.*

For the rest of this section, let $G \in \mathcal{O}_{d,c}$ be a graph with n vertices and H be a tree with $k > 1$ vertices. We can assume without loss of generality that $k \leq n$, as there is no subgraph isomorphism from H to G whenever $|V(H)| > |V(G)|$, and that G is connected, as any subgraph isomorphism maps a connected graph into a connected component. We fix an arbitrary vertex $r \in V(G)$ and will denote the choice of r by G^r. For a biconnected component B we define its *root*, denoted $\rho_r(B)$, to be the vertex of B with the smallest distance to r, where by distance between two vertices we mean the length of a shortest path connecting them. Since B is a maximal biconnected subgraph, $\rho_r(B)$ is unique. Furthermore, r is always among the roots of biconnected components (recall that bridges are also biconnected components). B will also be referred to as a *v-rooted* biconnected component, where $v = \rho_r(B)$. The subgraph formed by the set of v-rooted biconnected components of G^r is denoted by $\mathcal{B}_v^r$ (we omit G from the notation as it will always be clear from the context). On the set of roots of all biconnected components in G^r we define a directed graph T_G^r as follows: For any $u, v \in V(T_G^r)$ with $u \neq v$ we have $(u, v) \in E(T_G^r)$ if and only if there exists a biconnected component B in $\mathcal{B}_v^r$ with $u \in V(B)$. T_G^r is a tree (rooted at r) that will be traversed by the algorithm deciding subgraph isomorphism. Figure 1 a) and b) show an example of the structures G^r and T_G^r for a small graph G.

A vertex w is *below* v with respect to r if all paths in G from r to w contain v. If r is clear from the context we simply say w is below v. The definitions imply that every vertex is below itself and that all vertices are below r. A *rooted subgraph* G_v^r of G for a vertex $v \in V(G)$ is the subgraph of G induced by the set of vertices below v, i.e.

$$G_v^r = G\left[\{w : w \text{ is below } v\}\right].$$

See Figure 1 c) for an example of G_v^r. It follows from the definitions and from the connectivity of G that G_v^r is a connected graph. Furthermore, the definitions above imply that $G_r^r = G$ and that G_w^r is a single vertex if and only if $w \notin V(T_G^r)$.[4] A vertex $w \in V(G)$ is called a *child* of v, if $vw \in E(G)$ and w belongs to a v-rooted biconnected component.

Using the above notions, we now describe our subgraph isomorphism algorithm given in Algorithm 2. It utilizes the fact that for any free tree H and connected graph G, $H \preccurlyeq G$ if and only if H is subgraph isomorphic to a spanning tree of G. Since G can have exponentially many spanning trees, we cannot directly apply the subtree isomorphism algorithm given in [18] to *all* spanning trees of G. For every vertex v, however, the graph induced by all biconnected components containing v has only a polynomial number of spanning trees in the size of G, since both d and c are bounded by constants. Our algorithm decides subgraph isomorphism by computing and combining (partial) subgraph isomorphisms from subtrees of the pattern to such spanning trees rooted at v in a bottom up way.

[4] We recall that by condition, G contains at least one edge.

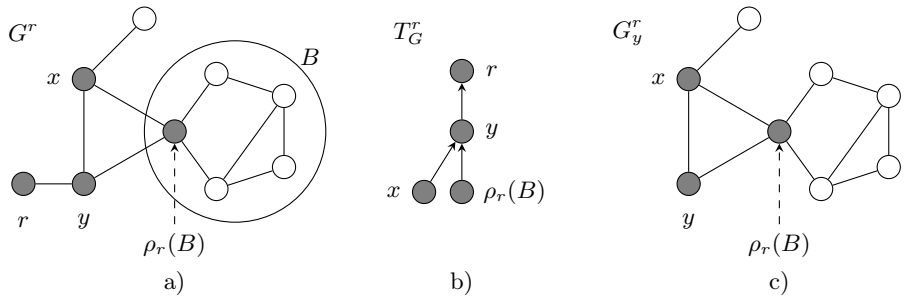

Fig. 1. G^r, T^r_G and G^r_y for a small graph G. $\rho_r(B)$ is the root of the block B. Roots are shown in gray, while vertices that are not roots are shown in white.

We now formally define the type of partial subgraph isomorphisms used by our algorithm.[5] Let G' be an induced subgraph of G and τ be a spanning tree of G'. Then $G|_\tau$ denotes the graph obtained from G by removing all edges of G' that are not in τ. We will often choose $G' = \mathcal{B}^r_v$ for some root v and spanning tree τ of $\mathcal{B}^r_v$. For this case $G^r_v|_\tau$ denotes the graph obtained from G^r_v by removing all edges from $\mathcal{B}^r_v$ that are not in τ. As an example, $G^r_v|_\tau$ in Fig. 2 arises from G by removing the edges rv, vx and the vertex r. Finally, an *iso-triple* ξ of H relative to a root $v \in V(T^r_G)$ is a triple (H^y_u, τ, w), where $u \in V(H)$, $y \in \mathcal{N}(u) \cup \{u\}$, τ is a spanning tree of $\mathcal{B}^r_v$, and $w \in V(\tau)$.

Definition 4. *An iso-triple ξ is a v-characteristic if and only if there exists a subgraph isomorphism φ from H^y_u to $(G^r_v|_\tau)^v_w$ with $\varphi(u) = w$.*

In Lemma 5 below we first provide a characterization of subgraph isomorphisms from trees into outerplanar graphs in terms of v-characteristics. Its proof follows directly from the definitions.

Lemma 5. *Let H be a tree, G be an outerplanar graph, and r be an arbitrary vertex of G. Then $H \preccurlyeq G$ if and only if there exists a v-characteristic (H^u_u, τ, w) for some $v \in T^r_G$, $u \in V(H)$, spanning tree τ of $\mathcal{B}^r_v$, and $w \in V(\tau)$.*

As an example, consider the star graph H with center u and neighbors u_1, u_2, u_3, u_4, u_5 and the graph G^r in Fig. 2. Obviously, there is a subgraph isomorphism from H to G mapping u to w. For each leaf u_i of H ($u_i = u_1$ in Fig. 2), we would find a v-characteristic $(H^{u_i}_u, \tau, w)$, as we can injectively map the four remaining neighbors of u to the children of w in τ and τ', i.e. to the neighbors of w in G, except for v. When we compute the r-characteristics for v later on, we will then find a characteristic $(H^{u'}_{u'}, \tau'', v)$ for each leaf u' of H and

[5] Though the terminology below follows that of [8] (which, in turn, is based on the concepts in [16]), the definitions of iso-triples and characteristics in this paper are entirely different from those in [8].

1: **function** MAIN(H, G)
2: pick a vertex $r \in V(G)$ and set $\mathcal{C} := \emptyset$
3: **for all** roots v of T_G^r in a postorder **do**
4: let $\mathcal{S}_v^r$ be the set of spanning trees of $\mathcal{B}_v^r$, each rooted at v
5: **for all** $\tau \in \mathcal{S}_v^r$ **do**
6: **for all** $w \in V(\tau)$ in a postorder **do**
7: **for all** $u \in V(H)$ **do**
8: **if** $w \in V(\tau) \setminus \{v\}$ or $w = r$ **then**
9: $\mathcal{C} := \mathcal{C} \cup$ CHARACTERISTICS(u, v, τ, w)
10: **if** $(H_u^u, \tau, w) \in \mathcal{C}$ **then return** TRUE
11: **return** FALSE

12: **function** CHARACTERISTICS(u, v, τ, w)
13: $\mathcal{C}_\tau := \emptyset$
14: **for all** $\theta \in \Theta_{vw}(\tau)$ **do**
15: let τ' be the tree satisfying $\theta = \tau \cup \tau'$
16: let C_τ (resp. $C_{\tau'}$) be the children of w in τ (resp. τ') and $C_\theta := C_\tau \cup C_{\tau'}$
17: let $B = (C_\theta \dot{\cup} \mathcal{N}(u), E)$ 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 $cu' \in C_\theta \times \mathcal{N}(u)$
18: **if** B has a matching that covers $\mathcal{N}(u)$ **then return** $\{(H_u^u, \tau, w)\}$
19: **else**
20: **for all** $u' \in \mathcal{N}(u)$ **do**
21: **if** B has a matching covering $\mathcal{N}(u) \setminus \{u'\}$ **then**
22: add $\left(H_u^{u'}, \tau, w\right)$ to $\mathcal{C}_\tau$
23: **return** $\mathcal{C}_\tau$

Algorithm 2: The Subtree Isomorphism Algorithm

unique spanning tree τ'' of B_r^r with the single edge vr. Note that it is crucial in this example to combine the spanning trees of $\mathcal{B}_v^r$ with those of $\mathcal{B}_w^r$ or $\mathcal{B}_r^r$, as we would otherwise not be able to find a subgraph isomorphism from H to G.

Our algorithm, depicted in Algorithm 2, utilizes the condition stated in Lemma 5 above. It selects an arbitrary vertex $r \in V(G)$ (Line 2) and calculates the characteristics for all root vertices by traversing T_G^r in a postorder manner (Line 3). For any $v \in V(T_G^r)$, the set of v-characteristics can be computed, as we discuss below, from the sets of characteristics of the children of v in T_G^r, by considering all spanning trees of $\mathcal{B}_v^r$. Since G is a d-tenuous outerplanar graph of bounded cycle degree, the number of spanning trees of $\mathcal{B}_v^r$ is bounded by a polynomial of the size of G. Thus, for all $v \in V(T_G^r)$ visited, Algorithm 2 first computes the set $\mathcal{B}_v^r$ of v-rooted biconnected components and then calculates the set $\mathcal{S}_v^r$ of all spanning trees of $\mathcal{B}_v^r$ (Line 4). Each spanning tree is regarded as a tree rooted at v. The root v itself, if it is different from r, is ignored at the stage when the algorithm visits it, as it will be processed when visiting its

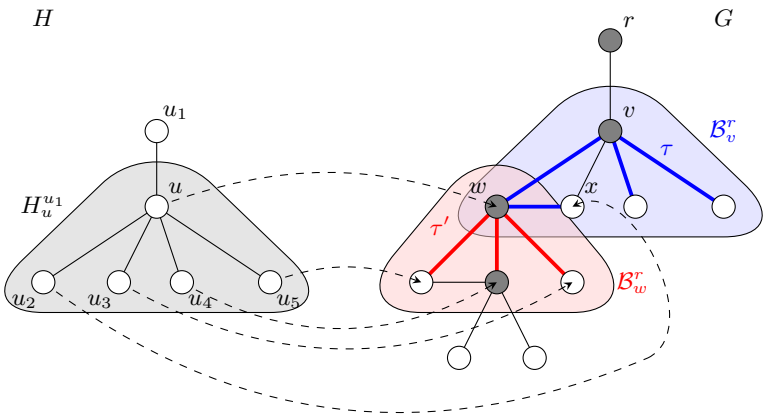

Fig. 2. This figure shows a small graph G^r with its subgraphs $\mathcal{B}_v^r$ and $\mathcal{B}_w^r$ (depicted by the rounded triangles) and the star graph H. One spanning tree τ of $\mathcal{B}_v^r$ and τ' of $\mathcal{B}_w^r$ are shown in red and blue, respectively.

parent vertex in T_G^r (Lines 6 and 8). Otherwise, i.e., when $v = r$, we need to compute the r-characteristics at the end of the postorder traversal of τ, i.e., when $r = v = w$, as r has no parent root.

In Loop 3–10 we calculate in variable $\mathcal{C}$ the set of v-characteristics for all combinations of $v \in V(T_G^r)$, spanning tree τ of $\mathcal{B}_v^r$, $u \in V(H)$, and $w \in V(\tau)$. By Lemma 5, there exists a subgraph isomorphism from H to G if and only if there are v, τ, u, and w such that the iso-triple (H_u^u, τ, w) is a v-characteristic. This condition is tested in Line 10. If such a v-characteristic exists the algorithm terminates by returning the answer TRUE. Thus, the algorithm reaches Line 11 and returns FALSE only if there is no subgraph isomorphism from H to G.

We now turn to the problem of computing the characteristics for the nodes of T_G^r. In Lemmas 6 and 7 below we formulate sufficient and necessary conditions for iso-triples being characteristics. These conditions, as we show below, allow for a recursive computation of the set of v-characteristics for any node $v \in V(T_G^r)$ from those of its children in T_G^r and with respect to the spanning trees of $\mathcal{B}_v^r$, justifying the post-order traversal of T_G^r (see Line 3 of Algorithm 2). To compute the v-characteristics for a root vertex v with respect to a spanning tree τ of $\mathcal{B}_v^r$, our algorithm traverses τ in a post-order manner and checks whether there exists a subgraph isomorphism from a certain subtree of H to $G_v^r|_\tau$, i.e., to the graph obtained from G_v^r by replacing $\mathcal{B}_v^r$ with its spanning tree τ. Since $V(\tau) = V(\mathcal{B}_v^r)$, this operation is invariant to the set of vertices of G_v^r. The difficulty of this step is, however, that $\mathcal{B}_v^r$ does not contain the complete neighborhood of w whenever $w \in V(\tau)$ with $w \neq r$ is a root vertex of another biconnected block in G^r (see Figure 2 for such a case). Therefore, we need to take into account also the

spanning trees of G^r_w. The crucial observation is, however, that we do not have to handle the potentially exponentially many spanning trees of G^r_w one by one. As we state below, we can recover all information needed to calculate the set of v-characteristics by combining τ with the set $\mathcal{S}^r_w$ of spanning trees of $\mathcal{B}^r_w$ only. Since G is a d-tenuous graph of bounded cycle degree, the cardinalities of $\mathcal{S}^r_v$ and $\mathcal{S}^r_w$ are polynomial in the size of G. Thus, we need to process only polynomially many combinations of spanning trees from $\mathcal{S}^r_v$ and $\mathcal{S}^r_w$, allowing for a polynomial time algorithm deciding subgraph isomorphism.

We start the formal description of the above considerations by introducing the following notation for the recovery of the subtrees we need from the spanning trees containing w. Let $v \in V(T^r_G)$. Then for any $\tau \in \mathcal{S}^r_v$ and $w \in V(\tau)$ we define $\Theta_{vw}(\tau)$ by

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

where $\tau \cup \tau'$ is the graph with vertex set $V(\tau) \cup V(\tau')$ and edge set $E(\tau) \cup E(\tau')$. Thus, when $w \in V(T^r_G) \setminus \{v\}$, $\Theta_{vw}(\tau)$ is the set of all trees obtained by "gluing" τ and τ' at vertex w, for all spanning trees τ' of $\mathcal{B}^r_w$, as $V(\tau) \cap V(\tau') = \{w\}$. As an example, the combination of the blue and the red tree in Fig. 2 denotes an element of $\Theta_{vw}(\tau)$. Note that if w is a root vertex different from v then it always has at least one child in $\mathcal{B}^r_w$, that is, τ' is always a tree with at least one edge. In this case, our algorithm needs to combine all pieces of information computed for the children of w in τ, as well as for the children of w in all spanning trees τ' of $\mathcal{B}^r_w$. Otherwise, there are two cases: w is either not a root vertex or it is equal to v. For the first case we have that w has no neighbor outside of $\mathcal{B}^r_v$; for the second case it holds that $\mathcal{B}^r_w = \mathcal{B}^r_v$.

Lemma 6 first considers the case of v-characteristics for subtrees H^y_u for some $y \in \mathcal{N}(u)$, i.e., which can be obtained from H^u by removing the subtree rooted at y (and thus, the edge uy as well).

Lemma 6. *An iso-triple (H^y_u, τ, w) of H is a v-characteristic for some $v \in V(T^r_G)$ 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 of w in θ such that for all $u' \in \mathcal{N}(u) \setminus \{y\}$ there is a subgraph isomorphism from $H^u_{u'}$ to $(G^r_v|_\theta)^v_{\psi(u')}$ mapping u' to $\psi(u')$.*

A function ψ corresponding to the v-characteristic $(H^{u_1}_u, \tau, w)$ in Lemma 6 is depicted by the dashed lines in Fig. 2. Lemma 7 formulates an analogous characterization for the entire pattern $H = H^u_u$.

Lemma 7. *An iso-triple (H^u_u, τ, w) of H is a v-characteristic for some $v \in V(T^r_G)$ and $y \in \mathcal{N}(u)$ if and only if there exists a $\theta \in \Theta_{vw}(\tau)$ and an injective function ψ from $\mathcal{N}(u)$ to the children of w in θ such that for all $u' \in \mathcal{N}(u)$ there is a subgraph isomorphism from $H^u_{u'}$ to $(G^r_v|_\theta)^v_{\psi(u')}$ mapping u' to $\psi(u')$.*

Lemmas 6 and 7 above are utilized by Subroutine CHARACTERISTICS (see Algorithm 2) in the computation of the set of v-characteristics for a vertex $v \in T_G^r$ with respect to a spanning tree τ of $\mathcal{B}_v^r$ and $w \in V(\tau)$. In particular, for all $\theta \in \Theta_{vw}(\tau)$, we determine the tree τ' satisfying $\theta = \tau \cup \tau'$ (Line 15) and then compute the sets $C_\tau, C_{\tau'}$ of children of w in τ and τ', respectively (Line 16). To decide whether the conditions formulated in Lemmas 6 and 7 for the characterization of v-characteristics hold, we construct a bipartite graph with vertex set $C_\tau \cup C_{\tau'} \dot\cup \mathcal{N}(u)$ and connect a vertex $u' \in \mathcal{N}(u)$ with a vertex $c \in C_\tau$ (resp. $c \in C_{\tau'}$) if and only if there is a subgraph isomorphism from the subtree of H^u rooted at u' to $(G_v^r|_\theta)_c^v$. Note that at this stage, all such partial subgraph isomorphisms have already been computed by the algorithm during the postorder traversal of τ (resp. τ') and stored for all children of w in θ in $\mathcal{C}_\tau$ (resp. in $\mathcal{C}_{\tau'}$). The algorithm then decides whether there exists a matching in the bipartite graph constructed that covers all elements of $\mathcal{N}(u)$ (cf. Lemma 7) or all elements of $\mathcal{N}(u) \setminus \{u'\}$, for all $u' \in \mathcal{N}(u)$ (cf. Lemma 6). The set of v-characteristics with respect to w will then be determined by the outcome of these tests.

3.3 Correctness and Runtime

The correctness of Algorithm 2 follows from Lemmas 6 and 7 by noting that for a leaf $u \in V(H)$ the algorithm finds a v-characteristic (H_u^y, τ, w) for the unique neighbor y of u and for all $\tau \in \mathcal{S}_v^r$, $w \in V(\mathcal{B}_v^r) \setminus \{v\}$, and $v \in V(T_G^r)$. Note that we do not require G to be a d-tenuous outerplanar graph of bounded cycle degree for the algorithm to work correctly. These properties are only used to obtain a polynomial bound on the runtime.

We now claim that Algorithm 2 runs in polynomial time if $G \in \mathcal{O}_{d,c}$. Suppose we call MAIN(H, G) in Alg. 2 for a tree H with $|V(H)| = k$ and $G \in \mathcal{O}_{d,c}$ with $|V(G)| = n$ for some $k \leq n$. Let s be the maximum cardinality of $\mathcal{S}_v^r$ for any $v \in V(T_G^r)$. Then $s \in O\left(n^{c(d+1)}\right)$. Indeed, as G is a d-tenuous outerplanar graph, there are at most $O\left(n^{d+1}\right)$ spanning trees in any block of G. Furthermore, there are at most c v-rooted blocks and an arbitrary number of v-rooted bridges. But bridges have exactly one spanning tree, the bridge itself. The claim then follows by noting that all spanning trees contain v. Since the spanning trees of a graph can be generated with polynomial delay [17], the sets $\mathcal{S}_v^r$ can be computed in polynomial time. Thus, MAIN calls subroutine CHARACTERISTICS a polynomial number of times.

Regarding the complexity of CHARACTERISTICS, we have already seen that there are at most $s \in O\left(n^{c(d+1)}\right)$ spanning trees τ' in $\mathcal{S}_w^r$. The bipartite graph B constructed in Line 17 has at most $|\mathcal{N}(w)| + |\mathcal{N}(u)| \in O(n)$ vertices for every such τ'. Its edges can be constructed by $O\left(n^2\right)$ 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 such that each lookup and storage operation can be performed in constant time. A maximum matching on B can be found in $O\left(n^{5/2}\right)$ time [11]. Overall, we compute $O(n)$ matchings for each spanning tree $\tau' \in \mathcal{S}_w^r$ in Lines 18 and 21. (Note, that we can check for the existence of a

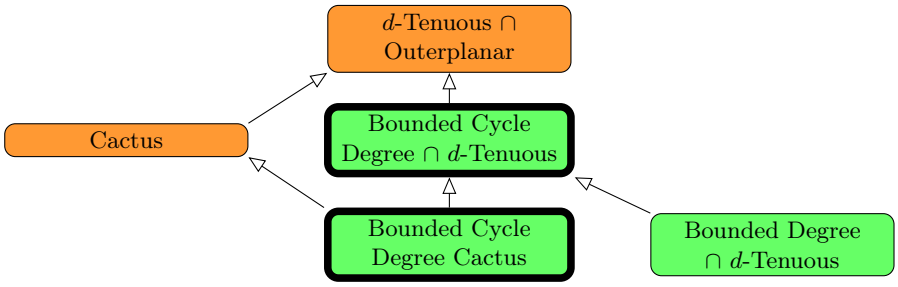

Fig. 3. Results on frequent subtree mining in outerplanar graphs. Green denotes polynomial delay and orange incremental polynomial time pattern mining. Thick borders indicate that the result is new up to our knowledge. Arrows depict subset relations between graph classes.

matching covering all but u' by deleting u' from B.) Thus, CHARACTERISTICS can be implemented to run in polynomial time and the overall runtime of Alg. 2 is bounded by a polynomial of the size of G.

4 Summary and Open Questions

The main contribution of this work is a polynomial delay algorithm generating frequent subtrees of d-tenuous outerplanar graphs of *bounded* cycle degree. It is based on a polynomial time subgraph isomorphism algorithm that may be of some independent interest. An overview of the results on mining frequent subtrees in different subclasses of outerplanar graphs is given in Fig. 3. Our results (marked with thick border) extend previous results on frequent subtree enumeration (as well as on subtree isomorphism [16]) in outerplanar graphs of bounded vertex degree to a more general class of outerplanar graphs, as bounded cycle degree is less restrictive than bounded vertex degree. Extending the picture with further non-trivial classes is an important challenge for graph mining, not only from theoretical, but also from application aspects.

The algorithm presented in this work is of theoretical interest. As future work, we are going to turn it into a practical algorithm. Our empirical studies on pharmacological molecules show that d and c are both very small numbers for the graphs in this application field.

Another question for future work is whether the positive result of this paper can be generalized to the case that the patterns are d-tenuous outerplanar graphs of bounded cycle degree. 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 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 pattern language above, we need 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. Therefore, if it is possible at all, such a generalization would require a more sophisticated approach.

4.1 An Open Problem

It is natural to ask whether the positive result in Theorem 2 can be generalized to *arbitrary* d-tenuous outerplanar graphs. That is, is it possible to generate frequent subtrees in d-tenuous outerplanar graphs with polynomial delay if we do not assume any constant upper bound on the cycle degree? We do not know the answer and the question remains open even for the very simple class of 0-tenuous outerplanar graphs. Such diagonal-free outerplanar graphs are also referred to as *cactus* graphs. In order to discuss the two possible answers to the question above, we first state a lemma.

Lemma 8. *Let $\mathcal{P}, \mathcal{G}$ be graph classes satisfying Conditions 1–3 of Theorem 1, but not Condition 4, i.e., subgraph isomorphism from $\mathcal{P}$ to $\mathcal{G}$ is NP-complete. If the FCSM problem for $\mathcal{P}$ and $\mathcal{G}$ cannot be solved with polynomial delay then $P \neq NP$.*

Proof. Assume that frequent patterns from $\mathcal{P}$ can *not* be generated with polynomial delay in transaction databases from $\mathcal{G}$. If the subgraph isomorphism problem for $\mathcal{P}$ and $\mathcal{G}$ is not in **P**, we are done, as subgraph isomorphism is in **NP**.

Suppose for contradiction that there exists an algorithm $\mathfrak{A}$ that decides in polynomial time for all $P \in \mathcal{P}$ and $G \in \mathcal{G}$ whether P is subgraph isomorphic to G. But then, by using Algorithm 1 with $\mathfrak{A}$ as embedding operator, we get a polynomial delay mining algorithm, contradicting our assumption that such an algorithm does not exist. Thus, there is no polynomial time algorithm deciding whether P is subgraph isomorphic to G and hence $\mathbf{P} \neq \mathbf{NP}$. □

We are now ready to discuss the open problem formulated above:

(i) Suppose the problem *can* be solved with polynomial delay. Then this would imply that polynomial delay frequent pattern enumeration is possible for NP-complete pattern matching operators, thus solving an open problem.

(ii) Suppose the problem *can not* be solved with polynomial delay. Then, by Lemma 8 above, a proof of this negative result would immediately imply that $\mathbf{P} \neq \mathbf{NP}$. Somewhat surprisingly, deciding subgraph isomorphism from a tree into a cactus graph is an NP-complete problem [2]. Thus Lemma 8 applies to tree patterns and cactus transaction graphs. Therefore, it is likely very difficult to prove for this case that tree patterns can not be mined with polynomial delay.

Thus, both the positive or the negative answer would be a very interesting result. We conjecture that (ii) holds, that is, polynomial delay pattern generation is impossible for computationally intractable pattern matching operators; this is true for graph classes in which the Hamiltonian path problem is NP-complete. If our conjecture holds, then it implies that

(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, and not to show that polynomial delay pattern mining is not possible and
(b) *even for very simple graph classes, the cycle degree of the transaction graphs is a crucial parameter for polynomial delay frequent pattern generation.*

References

1. R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo. Fast discovery of association rules. In *Advances in Knowledge Discovery and Data Mining*, pages 307–328. AAAI/MIT Press, 1996.
2. T. Akutsu. 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, 1993.
3. Y. Chi, Y. Yang, and R. R. Muntz. Indexing and mining free trees. In *ICDM*, pages 509–512. IEEE Computer Society, 2003.
4. L. De Raedt. *Logical and relational learning*. Cognitive Technolog. Springer, 2008.
5. R. Diestel. *Graph Theory*, volume 173. Springer, 2012.
6. G. C. Garriga, R. Khardon, and L. D. Raedt. Mining closed patterns in relational, graph and network data. *Ann. Math. Artif. Intell.*, 69(4):315–342, 2013.
7. G. Gottlob. Subsumption and implication. *Inf. Process. Lett.*, 24(2):109–111, 1987.
8. M. Hajiaghayi and N. Nishimura. Subgraph isomorphism, log-bounded fragmentation, and graphs of (locally) bounded treewidth. *J. Comput. Syst. Sci.*, 73(5):755–768, 2007.
9. J. Han, J. Pei, Y. Yin, and R. Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Min. Knowl. Discov.*, 8(1):53–87, 2004.
10. F. Harary. *Graph Theory*. Addison-Wesley series in mathematics. Perseus Books, 1994.
11. J. E. Hopcroft and R. M. Karp. An n^5/2 algorithm for maximum matchings in bipartite graphs. *SIAM Journal on computing*, 2(4):225–231, 1973.
12. T. Horváth and J. Ramon. Efficient frequent connected subgraph mining in graphs of bounded tree-width. *Theor. Comput. Sci.*, 411(31-33):2784–2797, 2010.
13. T. Horváth, J. Ramon, and S. Wrobel. Frequent subgraph mining in outerplanar graphs. *Data Min. Knowl. Discov.*, 21(3):472–508, 2010.
14. T. Horváth and G. Turán. Learning logic programs with structured background knowledge. *Artif. Intell.*, 128(1-2):31–97, 2001.
15. D. S. Johnson, M. Yannakakis, and C. H. Papadimitriou. On generating all maximal independent sets. *Information Processing Letters*, 27(3):119–123, 1988.
16. J. Matoušek and R. Thomas. On the complexity of finding iso-and other morphisms for partial k-trees. *Discrete Mathematics*, 108(1):343–364, 1992.
17. R. C. Read and R. Tarjan. Bound on backtrack algorithms for listing cycles, paths, and spanning trees. *Networks*, 5:237–252, 1975.
18. R. Shamir and D. Tsur. Faster subtree isomorphism. In *Theory of Computing and Systems, 1997*, pages 126–131. IEEE, 1997.