Efficient Frequent Subgraph Mining in Transactional Databases

Pascal Welke
University of Bonn
welke@cs.uni-bonn.de

***Abstract*—Frequent connected subgraph mining (FCSM) has been an active area of research over the last twenty years. This review shall focus on the practical and theoretical issues arising in the *transactional setting*, where we are given a finite list of small to medium sized graphs and must find all graphs that are *subgraph isomorphic* to some user-defined number of graphs in the list. In particular, we present the generic approach to FCSM and investigate sufficient conditions for its computational tractability and intractability. Interestingly, it turns out that these are dependent on the complexity of the HamiltonianPath problem. This implies that FCSM is computationally tractable only for very restricted transaction graph classes. We subsequently review existing exact FCSM algorithms with a focus on applicability to arbitrary graph databases and present recent approximative FCSM algorithms that remain computationally tractable for all transactional databases.**

***Index Terms*—Frequent Subgraph Mining, Subgraph Isomorphism, Computational Complexity**

I. Introduction

We present an overview on classical and recent results in the area of frequent connected subgraph mining (FCSM). Here, we are given a finite list $\mathcal{D}$ of small to medium sized graphs and a pattern graph H is considered to be frequent if it is *subgraph isomorphic* to at least $t \in [|\mathcal{D}|]$ graphs in $\mathcal{D}$. Frequent subgraphs can be used in many applications e.g. directly for exploratory data analysis, association rule mining, or indirectly for similarity assessment of graphs, classification, among many other possibilities.

Over the last twenty years, many practical implementations of frequent subgraph mining systems for the transactional setting have been proposed. All algorithms employ some depth-first or breadth-first search strategy over candidate patterns, counting support, and pruning infrequent patterns and their extensions. Most published work focuses on the efficient enumeration of candidate patterns and on canonicalization of the patterns to avoid duplicates [7, 17]. They address the support counting step in a less detailed manner, either citing some off the shelf subgraph isomorphism algorithm, or roughly sketching ways to keep track of *all* possible embeddings of patterns into the transaction database (compare Section II-B). Jiang et al. [17] suggest that this is due to the fact that the subgraph isomorphism problem is seen as "harder to address". Hence more work is spent to reduce the number of calls to the subgraph isomorphism subroutine as much as possible. While this is an important issue, the main computational effort for medium to large graph databases is to evaluate the embedding operator for candidate patterns on transaction graphs [37].

It seems, that around the year 2008 the general interest in novel algorithms faded and many people moved on to parallelizing existing algorithms [cf. 17, 27] or solving different problem formulations using variations of the existing algorithms. Jiang et al. [17] concluded in 2013 that this is due to the maturity of the field. We disagree; our review shows that existing exact algorithms are either restricted to very simple graph classes or have exponential delay in common cases. This in practice restricts such graph mining algorithms almost exclusively to chemical graph databases. Welke et al. [35] have shown that the state-of-the-art graph mining algorithms (which all have exponential delay in the worst case) are inapplicable on several non-chemical datasets. 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. We will hence review the related work with special regard to the computational complexity of the mining algorithms. We start by investigating sufficient conditions for computational tractability and intractability of FCSM. In particular, we will have a close look at the techniques employed to solve the subgraph isomorphism problem during the mining. For reviews focusing more on other aspects, we refer to [7, 17].

Recently, however, several novel approaches have emerged that approximate the set of frequent subgraphs. To avoid the computational complexity issues mentioned above, they (i) restrict the pattern class and (ii) give up on the correctness of the set of "frequent" patterns found. As a result of this trade-off, they instead guarantee polynomial delay for arbitrary transaction graph databases. This complements the exact but inefficient algorithms that were proposed in the first decade of this millennium and allows us to conclude with a simple practical guideline which algorithms to use and how to engineer hybrid algorithms that combine the best of both worlds.

a) Outline: This survey is organized as follows: In Sect. II we discuss the generic approach to frequent subgraph mining and its computational complexity issues. Sect. II-A presents efficient embedding operators for the generic approach, while Sect. II-B presents embedding list based operators, which are most common in the early practical mining systems. We present exact algorithms in Sect. III, approximative algorithms in Sect. IV, and conclude in Sect. V.

II. The Complexity of FCSM

In its most general form, the FCSM problem can be formulated as follows:

Definition 1 (Frequent Connected Subgraph Mining Problem): Given a finite list $\mathcal{D} \subseteq \mathcal{G}$ (called *graph database*) for some graph class $\mathcal{G}$ and an integer threshold $t \in [|\mathcal{D}|]$, *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 $\mathcal{D}$. The patterns in the output must be pairwise non-isomorphic.

An equivalent definition of this problem uses a *relative* (instead of an absolute) frequency threshold. We parametrize the problem by the transaction class $\mathcal{G}$ *and the pattern class* $\mathcal{P}$. These two graph classes are typically given implicitly, by hardcoding them into the algorithms.[1] In particular, if $\mathcal{P}$ is the class of trees, then we call this the Frequent Subtree Mining (FTM) problem.

The FCSM problem is a *listing problem*, where the output is a list of objects. For listing problems, the following *output sensitive complexity measures* are distinguished in the literature [18]. Suppose an algorithm $\mathfrak{A}$ for some listing problem gets an instance x as input and outputs some sequence $y = [p_1, p_2, \cdots, p_n]$ of patterns. Then $\mathfrak{A}$ generates y

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

In general, the FCSM problem and the FTM problem can *not* be solved in output polynomial time unless **P** $\neq$ **NP** [15, 35]. One way to obtain positive results, however, is to restrict the transaction and/or pattern graph classes in the FCSM problem. This allows to achieve some tractability results, in particular for the FTM problem.

Theorem 1 ([35]): 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 $H \in \mathcal{P}$.

[1] An example of an implicit transaction graph class $\mathcal{G}$ would be to write an algorithm that expects all graphs in the database $\mathcal{D}$ to be connected, omitting checks for multiple connected components and repetition of work on the later components. Restricting the pattern class $\mathcal{P}$ e.g. to trees allows to generate only candidate patterns that are trees and more intricate methods to avoid duplicate candidates [see, e.g., 7].

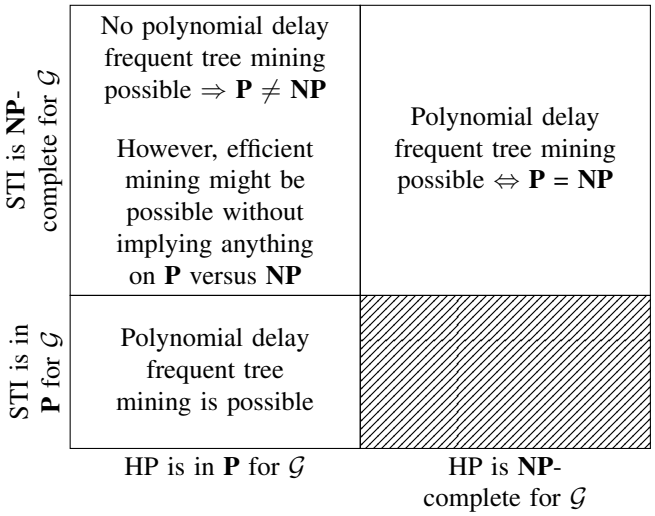

Fig. 1. Relationship between complexities of the HamiltonianPath (HP) and SubtreeIsomorphism (STI) problems for transaction graphs from $\mathcal{G}$ and the complexity of the FTM problem.

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 in the size of $\mathcal{D}$ for $\mathcal{P}$ and $\mathcal{G}$.

Theorem 1 states that polynomial delay mining of frequent trees is possible if the SubtreeIsomorphism problem can be decided in polynomial time for a given transaction graph class $\mathcal{G}$ (Conditions 1–3 hold if $\mathcal{P}$ is the class of trees). A negative result in [15], however, implies that the FTM problem cannot be solved in output polynomial time for arbitrary transaction graphs, unless **P** = **NP**. Together, these two results draw a intricate picture of the complexity of the SubtreeIsomorphism problem: A few transaction graph classes (most notably, if $\mathcal{G}$ is the class of trees) are easy to mine. On the other hand, the question whether frequent tree mining is possible in output polynomial time is equivalent to the question whether **P** = **NP** for a broad range of transaction graph classes. For the remaining transaction graph classes, interestingly, frequent subtree mining is connected to the complexity of the HamiltonianPath problem. An overview of these results is shown in Fig. 1.

Three main research areas have evolved from this theoretical and practical limitation: (i) Practical algorithms that ignore the complexity issues and aim to be fast on certain (typically not well defined) types of transaction databases, (ii) algorithms that restrict either pattern or transaction graph class and are efficient on well defined data, and (iii) algorithms that relax the problem to avoid the complexity of the exact mining.

A. Efficient Algorithms for the SubgraphIsomorphism Problem

How to efficiently address Cond. 4 of Thm. 1 is usually left out by graph mining papers. In fact, most graph miners focus

on efficient candidate enumeration, instead of embedding computation. The literature typically justifies this by showing experimental results on chemical graph databases, where the mining systems are fast. We refer to [7] for a discussion of efficient candidate generation for tree patterns and now turn to efficient embedding computation algorithms.

Various efficient algorithms exist for restricted versions of the SubgraphIsomorphism problem; possibly even more special cases have been shown to remain **NP**-complete. There is no way to give a complete overview on the work that was done in this area; we present a few results that relate to mining. In particular, we focus on the complexity of the SubtreeIsomorphism problem as this is relevant for both exact mining and for several efficient relaxations.

The SubtreeIsomorphism problem can be solved in polynomial time if the pattern graph H is a tree and the transaction graph G is a forest. Various efficient algorithms have been proposed in the last fifty years [8, 21, 24, 30, 33]. However, the positive result for the SubtreeIsomorphism problem does not hold if the pattern graph is allowed to be disconnected, i.e., the SubgraphIsomorphism problem is **NP**-complete even for forest transactions [12]. The currently fastest known algorithm for the SubtreeIsomorphism problem for a tree pattern H and forest transaction G requires $O\left(|V(G)| \cdot |V(H)|^{1.5} / \log |V(H)|\right)$ time [30]. Generally, the SubtreeIsomorphism problem is solved by bottom-up evaluation over a rooted version of one of the two trees. This is done by combining partial subgraph isomorphisms of the children of the current vertex by solving bipartite matching instances. As a result, frequent subtree mining in forest databases is computationally tractable (cf. Thm 1).

Marx and Pilipczuk [22] systematically investigate the complexity of the SubgraphIsomorphism problem for several combinations of pattern and transaction graph classes. They consider the tractability of the problem when bounding any combination of ten parameters, containing, e.g., number of vertices, number of connected components, maximum vertex degree, and tree-width, for pattern, or transaction. Their (maximal) positive results require either the number of vertices or the maximum vertex degree of the pattern to be constant. If the pattern has constant vertex degree, polynomial algorithms are known for transactions that are almost-k trees [1], or have constant tree-width [23], or log-bounded fragmentation [13]. On the other hand, it follows from Akutsu [1] that the SubtreeIsomorphism problem is **NP**-complete for unbounded pattern degree even for outerplanar graphs. As outerplanar graphs have tree-width at most two, we have a clear distinction of the complexity of the SubtreeIsomorphism problem based on the tree-width of the transaction graph class: It is in **P** for the class of graphs with tree-width at most one and is **NP**-complete for the class of all graphs with tree-width k for all $k \geq 2$, unless we restrict the vertex degree of the pattern tree to be a constant. However, the vertex degree a pattern tree can be unbounded if a less restrictive property, called *block degree* of the transaction graphs is restricted [35].

B. Embedding Lists and Exponential Algorithms

As positive complexity results for restrictions of the SubtreeIsomorphism problem are scarce, algorithms for the SubgraphIsomorphism problem that do not guarantee bounded worst-case runtime but are fast in practice on many types of graphs have been proposed. To this end, Ullmann [31] computes the set of all subgraph isomorphisms from a pattern graph H to a transaction graph.[2] An extension of Ullmann's algorithm is due to [9]. His algorithm fixes an order $[v_1, v_2, \ldots, v_{|V(H)|}]$ of the vertices of H and considers the sequence $[H_1, H_2, \ldots, H_{|V(H)|} = H]$ of induced subgraphs $H_i = H[\bigcup_{j=1}^{i} v_j]$. It computes the set $EL(H_{i+1}, G) := \{\varphi : \varphi : V(H_{i+1}) \to V(G) \text{ is a subgraph isomorphism}\}$ by extending each subgraph isomorphism $\varphi \in EL(H_i, G)$ to subgraph isomorphisms from H_{i+1} to G as follows: The algorithm checks whether the novel vertex v_{i+1} in H_{i+1} is compatible to φ. That is, whether there exists a vertex $w \in V(G)$ that is not yet part of the image of φ and is connected to all images of the neighbors of v_{i+1} in H_{i+1}. Hence each $\varphi \in EL(H_i, G)$ can be extended to up to $|V(G)| - i$ isomorphisms from H_{i+1} to G. The algorithm either terminates by finding a subgraph isomorphism from H to G or stops after finding a subgraph isomorphism from some H_i to G, but none from H_{i+1} to G.

This method works well for chemical graphs and some other workloads [26, 40]. Due to a moderate number of vertex and edge labels, high sparsity and (almost) planarity of chemical graphs the sizes of the sets $EL(H_i, G)$ tend to be small. The runtime of Ullmann's algorithm depends on the total number of subgraph isomorphisms from any H_i in $[H_1, H_2, \ldots, H_{|V(H)|} = H]$ to G. This number is at most

$$\sum_{i=1}^{|V(H)|} |EL(H_i, G)| \leq \sum_{i=1}^{|V(H)|} \binom{|V(G)|}{|V(H_i)|} \cdot |V(H_i)|! \ .$$

This bound is best possible without any further assumptions on G. Consider the case that G is an unlabeled complete graph: For each permutation of each k-sized subset of the vertices of G there exists a unique isomorphism from (any graph) H with $|V(H)| = k$. Hence, the number of subgraph isomorphisms that need to be computed may be exponential in the size of G and factorial in the size of H. There is no known way to compute or estimate the exact number of such embeddings (an exact solution in polynomial time would solve the SubgraphIsomorphism problem). Hence it is required to run the algorithm and to observe the required runtime and consumed memory for storing embeddings.

On the other hand, however, this method is easy to implement and particularly well-suited for the workload of frequent subgraph mining systems. For a breadth-first or depth-first mining algorithm all (resp. one) subgraphs of any pattern graph H were already enumerated and their support count has already been computed. Hence, we have (in the notation from above) already computed $EL(H_{|V(H)|-1}, G)$ for some suitable

[2] Checking whether this set is empty, or not, obviously solves the SubgraphIsomorphism problem.

direct predecessor[3] $H_{|V(H)|-1}$ of $H = H_{|V(H)|}$. If we store all embeddings for all patterns from the previous level, we can compute the set of embeddings of H into any graph G in the database, by reusing the embeddings of a predecessor.

In practice, it seems to be the case that a low average vertex degree in combination with a moderately-sized set of possible vertex and edge labels dramatically reduces the number of possible subgraph isomorphisms. Empirical evaluations of the existing frequent subgraph mining systems indicate that this approach works well on chemical graphs and some other databases. There is generally no guarantee that it is always the case. In fact, there are many practically relevant graph databases where the runtime and space requirements of Ullmann's algorithm explode for no apparent reason [35].

III. Algorithms for the FCSM Problem

We briefly discuss the most relevant exact algorithms with focus on their efficiency. As before, we distinguish between FTM systems (where efficient mining might be achievable) and general FCSM systems. An overview of the presented algorithms can be found in Table I.

A. Frequent Subtree Mining Algorithms

As we have seen, frequent subtrees can be enumerated efficiently (i.e., with polynomial delay) in forest transaction databases [15]. However, most existing systems do not use an efficient embedding operator and hence may result in exponential delay and memory consumption even in this case.

FreeTreeMiner [5] solves the FTM problem for tree databases. This work introduces tree mining as an area of research and develops the first[4] algorithm that uses canonical representations of trees for efficient pattern generation. The authors propose a canonical string representation for trees and a BFS algorithm to mine all frequent trees in a tree database. Chi et al. use the efficient algorithm of Chung [8] to compute the support of a candidate tree pattern in the tree database. They evaluate their algorithm on a chemical dataset, an IP multi-cast dataset that represents one-to-many streaming topologies on the Internet, and on synthetic datasets.

HybridTreeMiner [6] also solves the FTM problem for tree databases, and, in addition, the problem of mining rooted trees in databases of rooted trees. They use a DFS approach, instead of a BFS approach and propose a novel way of counting the support. Now, the authors resort to embedding lists but use them in a smart way that requires only one pass over the database. This, however, results in exponential worst case time and space in the size of the output. They show, however, that this approach is faster by an order of magnitude compared to FreeTreeMiner. Interestingly, however, the IP multi-cast dataset is not considered in this study.

[3]When mining graphs that may contain cycles, the notions are slightly modified to allow the extension to work edge-by-edge, not vertex-by-vertex. In the context of the FTM problem, however, both notions are equivalent.

[4]Zaki [39] introduced "tree mining" before, but considered rooted ordered trees and a different embedding operator.

FreeTreeMiner [28] solves the FTM problem in databases containing more general graphs. The algorithm explicitly computes the embedding lists for all extensions by a single edge while evaluating the frequency of a candidate pattern. These candidate patterns are only recursively extended if they have a canonical form. The authors do not prove the soundness, completeness, nor irredundancy of their algorithm and evaluate their algorithm on a chemical database.

F3TM [40] similarly solves the FTM problem in databases containing cyclic graphs using a depth-first search over the pattern space. They focus on the candidate generation step and employ an iterative version of [31] for the support counting step that is intertwined with the candidate generation step. They evaluate F3TM on chemical and artificial data [19].

B. Frequent Subgraph Mining Algorithms

FSG [19] is a BFS algorithm for mining all frequent subgraphs in graph transaction databases. To compute the support of a candidate pattern, FSG stores the support set of each frequent pattern and intersects the support sets of parent patterns to reduce the number of explicit subgraph isomorphism tests to be evaluated for any candidate pattern. The authors do not disclose the implementation details or a reference for their embedding operator. Neither do they mention additional storage requirements for storing embeddings explicitly, which might indicate that they use an algorithm that does not require such knowledge. The authors evaluate FSG on chemical and artificial graph datasets. The algorithm was used to first show the impressive predictive performance of frequent subgraph based learners on chemical graph datasets [11].

MoSS, is specifically targeted at chemical graph databases [2, 3]. Their algorithm implements special domain knowledge (e.g., handling of aromatic bonds) and is a depth-first search over a pattern space that can be "seeded" with a chemically meaningful core pattern that will be contained in all frequent patterns to be found. The authors use embedding lists to compute the support count; their approach, however, suffers from a missing graph canonicalization scheme. Hence patterns are enumerated multiple times (and their support is computed multiple times). The authors qualitatively analyze the patterns found using their approach on a chemical dataset.

gSpan [38] mines frequent subgraphs using a depth-first traversal of the pattern space. To avoid multiple enumeration of the same candidate pattern, it applies an inclusion-exclusion principle on frequent edges. That is, a pattern is extended with an ever shrinking set of frequent edges. To compute the support of a candidate pattern, the algorithm recursively works on the support sets of the patterns being extended, resulting in a reduced number of calls to the embedding operator. gSpan uses the subgraph isomorphism algorithm by [9]. They show experiments on the datasets used by [19] and show that their algorithm outperforms FSG.

FFSM [16] also uses a depth-first traversal of the pattern space. They use a novel canonical representation of arbitrary graphs that has size $O\left(n^2\right)$ for a graph on n vertices and

Name	Reference	Transactions	Embedding Algorithm	Delay	Comment
		Frequent Subtree Mining Algorithms			
FreeTreeMiner	Chi et al. [5]	Forests	Chung [8]	polynomial	
HybridTreeMiner	Chi et al. [6]	Forests	Embedding lists	exponential	
FreeTreeMiner	Rückert and Kramer [28]	Graphs	support sets	exponential	
F3TM	Zhao and Yu [40]	Graphs	Ullmann [31]	exponential	
		Frequent Subgraph Mining Algorithms			
FSG	Kuramochi and Karypis [19]	Graphs	Embedding lists	exponential	Mines all frequent subgraphs
MoSS	Borgelt and Berthold [2] Borgelt et al. [3]	Chemical Graphs	Embedding lists	exponential	Mines all frequent subgraphs
gSpan	Yan and Han [38]	Graphs	Cordella et al. [9]	exponential	Mines all frequent subgraphs
FFSM	Huan et al. [16]	Graphs	Embedding lists	exponential	Mines all frequent subgraphs
Gaston	Nijssen and Kok [25, 26]	Graphs	Embedding lists	exponential	Can mine paths, trees, and cyclic patterns
–	Horváth and Ramon [14]	Bounded Tree-Width	Horváth and Ramon [14]	incr. pol. time	Mines all frequent subgraphs
		Algorithms for Relaxed Problems			
SUMMARIZE-MINE	Chen et al. [4]	Graphs	Embedding lists	exponential	Mines a random subset of all frequent subgraphs
PS	Welke et al. [34, 35, 36]	Graphs	Shamir and Tsur [30] Welke et al. [35, 36]	pol. delay	Mines a random subset of all frequent subtrees
MUSE	Zou et al. [41]	Uncertain Graphs	Embedding lists	exponential	
REAFUM	Li and Wang [20]	Graphs	Embedding lists	exponential	β subgraph isomorphism
–	Schulz et al. [29]	Graphs	Dalmau et al. [10]	pol. delay	Mines a superset of all frequent subtrees

TABLE I
AN OVERVIEW OF FREQUENT SUBTREE AND SUBGRAPH MINING SYSTEMS FOR FOREST AND GRAPH TRANSACTION DATABASES.

propose extension and join operators that generate all frequent patterns. However, these operators may generate patterns multiple times, not necessarily in a canonical form. They use embedding lists and show how their extension and join operators can use them to only output frequent patterns.

Gaston [26] is the fastest frequent subgraph mining system on chemical graph databases [37]. Their algorithm mines frequent patterns in three stages: First, all frequent paths are generated. In the second stage, tree candidates are grown from the frequent paths. Finally frequent cyclic graphs are grown from the frequent trees and frequent paths by adding edges between existing vertices. Hence Gaston can be seen as both a specialized frequent subtree mining algorithm and as a frequent subgraph mining algorithm. There are two variants of Gaston that differ in their support counting subroutine. The first variant uses embedding lists, the second computes the subgraph isomorphisms "from scratch" for each candidate pattern. The authors are not very specific on the details of the latter. They describe it as a backtracking algorithm that has exponential worst-case running time in the size of the pattern and the transaction graphs involved. They evaluate their algorithm on an artificial tree dataset and on three large molecular datasets.

Horváth and Ramon [14] propose an algorithm that mines all frequent connected subgraphs in transaction databases consisting of graphs of bounded tree-width. Their algorithm runs in incremental polynomial time, while the embedding operator by itself is **NP**-complete (compare Section II-A). That is, the SubgraphIsomorphism problem is **NP**-complete for transaction graphs with tree-width at most some constant k if the vertex degree of the pattern is not bounded by a constant, as well. This result is, to the best of our knowledge, the only existing result that describes an efficient algorithm for a problem in the upper left quadrant of Figure 1. Their algorithm identifies a polynomially sized subset of *non-redundant iso-quadruples* that are stored for each frequent subgraph and each transaction. Such iso-quadruples represent partial subgraph isomorphisms but – in comparison to explicitly storing all possible embeddings from the patterns to the transaction graphs – may represent multiple embeddings of the pattern that are in some sense equivalent. Their embedding operator extends ideas from [13] to the case that the vertex degree of the pattern is unbounded. Interestingly, the approach of Horváth and Ramon requires a breadth-first traversal of the pattern space to be efficient. They show that almost all ($> 99.9\%$) of the graphs in a large chemical graph database have tree-width at most 3, and hence that their result is practically relevant but do not give any empirical evaluation of their algorithm.

IV. ALGORITHMS FOR RELAXED PROBLEMS

As we have seen, there has not been much progress in terms of efficient systems for exact frequent subtree mining or exact

frequent subgraph mining. Recently, researchers have turned to approximations of the FCSM and FTM problem. There has also been some interest in dealing with graph databases that contain noisy data. While this setting is different from the scope of this survey, some of the resulting algorithms can be applied to exact transactional graph databases and yield approximations of the set of frequent subgraphs.

Chen et al. [4] try to address the drawbacks of the exact systems described in Section III-B on larger graph transactions, i.e., the large number of embeddings of a pattern that need to be processed by the embedding operators described in Section II-B. To this end, they propose to replace each graph in the database by a *summarized graph* and to mine frequent patterns in this novel graph database using the gSpan algorithm [38]. A summarized graph G' is created from a labeled transaction graph G by choosing a random partition $V(G) = V_1 \dot\cup V_2 \dot\cup \ldots \dot\cup V_k$ of the vertex set of G such that for all $i \in [k]$, all vertices in V_i have the same label. Now, the vertices of G' are the partitions V_i and there exists an edge (V_i, V_j) with label l if and only if there exists an edge $(v_i, v_j) \in E(G)$ with $v_i \in V_i$, $v_j \in V_j$, and label l. Hence, the summarized graph G' is not simple, i.e., it may contain self-loops and multiple edges (with different labels) between any two vertices. This construction results in a two-sided error, i.e., for two graphs H and G and a summarized graph G' of G there may be (i) *false negatives:* that is $H \preccurlyeq G$ but $H \not\preccurlyeq G'$, or (ii) *false positives:* that is $H \not\preccurlyeq G$ but $H \preccurlyeq G'$. These effects obviously translate to the set of frequent patterns found by the algorithm of Chen et al.. To deal with false negatives, the authors propose to lower the frequency threshold in the mining phase and give a probabilistic guarantee on its effectiveness. To further increase the recall of frequent patterns their algorithm repeats the summarization independently several times. To address the false positives, they propose to simply retest the patterns found to be frequent in the summarized graph database on the original graph database. Together, this yields the SUMMARIZE-MINE algorithm that guarantees to find a subset of all frequent subgraphs in a given database. Chen et al. [4] find frequent subgraphs instead of frequent subtrees and use an exponential worst-case time embedding algorithm, thus they are not able to guarantee any delay bounds. Their algorithm requires to retest all patterns found in the summarized graph database on the original database to ensure that they are indeed frequent. Hence, no real structural simplification of the FCSM or SubgraphIsomorphism problems takes place.

Welke et al. [34] propose to mine frequent subtrees in arbitrary transaction graph databases with one-sided error. That is, their probabilistic subtree mining algorithm (PS) guarantees that each tree in the output is indeed frequent, but does not guarantee that all frequent trees are found. This is done by sampling a polynomial number of spanning trees for each graph in the transaction database and mining frequent subtrees in the resulting forest database. Hence, they transform an intractable FCSM problem to a tractable FTM problem which can be solved with polynomial delay (cf. Sect. III-A). In subsequent work, they present an extension that allows to implicitly consider exponentially many spanning trees per transaction graph, while maintaining the computational efficiency of the mining algorithm [35]. A recent paper proposes a faster embedding operator [36].

Zou et al. [41] propose MUSE to mine patterns in databases of *uncertain graphs*. An uncertain graph is a labeled graph G together with a probability function $p : E(G) \to [0, 1]$ on its edges and represents the probability distribution P over all graphs $(V(G), E')$ for $E' \subseteq E(G)$, with $P((V(G), E')) := \prod_{e \in E'} p(e)$. Now, for a pattern graph H and an uncertain graph G, the probability of H matching G is defined as

$$P_{\preccurlyeq}(H, G) = \sum_{E' \subseteq E(G)} P((V(G), E'))I(H, (V(G), E')) \ ,$$

where $I(H, (V(G), E')) = 1$ if $H \preccurlyeq (V(G), E')$ and $I(H, (V(G), E')) = 0$ otherwise. Given a graph database $\mathcal{D}$ and a frequency threshold $\theta \in (0, 1]$, MUSE approximates the set of all pattern graphs H with $\frac{1}{|\mathcal{D}|} \sum_{G \in \mathcal{D}} P_{\preccurlyeq}(H, G) \geq \theta$, i.e., where the average probability of H matching the graphs is at least θ. They show that counting the number of such patterns for a given database is #P-complete [32] and their algorithm approximates the set of such patterns. MUSE works by generating patterns on the database with probabilities removed from the edges by explicitly storing and extending the embeddings as described in Section II-B. Based on these explicit embeddings they propose an exponential time algorithm to compute the matching probabilities and an approximate algorithm that computes an interval of matching probabilities. They show how to obtain an algorithm that guarantees with high probability for some $\epsilon \in (0, 1]$ and a frequency threshold θ that all patterns with support at least θ are output, all patterns with support less than $(1-\epsilon)\theta$ will not be output, and decisions for remaining patterns are arbitrary. However, due to the necessity of evaluating a function over all embeddings of a pattern, the method does not run in output polynomial time. In a way, this can be seen as the opposite approach to Welke et al. [34]: They consider some probability distributions on the set of spanning trees given by the database graphs and obtain frequent subtrees from certain samples directly, instead of mining patterns on the underlying graphs.

Li and Wang [20] consider transactional graph databases that contain graphs where some vertices, edges, or labels may be "wrong". They propose to relax the notion of isomorphism and subgraph isomorphism. To this end they introduce β (subgraph) isomorphism, where a graph H is β subgraph isomorphic to a graph G if there exists a sequence of vertex and edge additions or deletions and relabeling operations of length at most β that transforms H into a graph that is (subgraph) isomorphic to G. If applied in a generic mining algorithm, such an embedding operator would result in finding a *superset* of the frequent patterns with respect to subgraph isomorphism: β subgraph isomorphism is equivalent to subgraph isomorphism for $\beta = 0$ and for any $\beta > 0$ the existence of a subgraph isomorphism from H to G implies the existence of a β subgraph isomorphism from H to G. Their proposed tool REAFUM, however, first selects a small subset

of "representative" graphs for a given graph database and considers only those patterns as candidates that can be found in the set of representative graphs. Frequency counting takes place on the full dataset and is based on storing all embeddings of all approximate matches of the patterns (i.e., an extension of the ideas described in Section II-B). In their experimental evaluation they show that they are able to find more patterns than the exact Gaston algorithm on a small molecular dataset. Due to the candidate selection process, the resulting pattern set is not guaranteed to be a superset of all frequent patterns with respect to normal subgraph isomorphism.

Schulz et al. [29] propose a frequent tree mining algorithm that allows to mine a superset of the frequent subtrees in an arbitrary graph database. Subgraph isomorphisms are injective graph homomorphisms; Schulz et al. propose to add *some* injectivity constraints to graph homomorphisms, while maintaining computational efficiency of the embedding operators. Graph homomorphism can be decided in polynomial time for patterns of bounded tree-width and *arbitrary transaction graphs* [10]. Hence, for tree patterns, which have tree-width one, one can add a number of binary injectivity constraints between vertices (implemented by new edges with a new label) to the pattern as long as the resulting graph remains of bounded tree-width. Deciding homomorphism between such an extended pattern and a transaction graph that is extended with all possible edges with that new label ensures that the injectivity constraints are fulfilled. As a result, tree patterns that are frequent with respect to subgraph isomorphism are frequent with respect to partially injective homomorphism, as well. Schulz et al. propose a mining strategy for this embedding operator that finds a superset of all frequent tree patterns (with respect to subgraph isomorphism). This is done by mining "maximally" constrained tree patterns that are defined by *k-trees*. Hence this method can be seen as approximating the set of frequent subtrees "from above".

V. Conclusion

We have presented sufficient conditions for tractable frequent connected subgraph mining and have surveyed exact FTM and FCSM systems. As it turns out, only [5, 14] use efficient embedding operators to solve the FCSM in incremental polynomial time. [14] is mainly a theoretical result as it was not implemented. Thus the only existing implementation with guaranteed worst-case delay (by Chi et al.) can only mine trees in forest transactions. This leaves us with the practical problem that all other algorithms may or may not work on any given graph database, and there is no way of knowing for sure. Even a single graph with a ridiculous amount of possible embeddings can hold up the mining process almost indefinitely. Hence we conclude that additional work is required to obtain efficient algorithms that are applicable in broader settings; most likely this task involves focusing more on efficient embedding operators. The recent development of efficient algorithms for approximations of the FTM problem, however, may spark new interest in frequent subgraph mining as a field of research and as a tool for data analysis, as it mitigates the stability issues of the exact, inefficient mining algorithms.

It remains an open problem to decide which kind of algorithm should be chosen for a given graph database, or possibly even individually for each graph in a database. While it was shown that the approximative methods are faster than exact frequent subgraph miners on complex graph databases, exact graph mining algorithms are typically faster on chemical graphs (and probably on some other very simple graph databases as well). It is possible to combine the potentially inefficient embedding computation with the efficient methods to get the best of both worlds: Introducing a new parameter, we can allow a mining algorithm to store at most a certain number of embedding lists per graph. If a candidate pattern results in too many embeddings for a given graph, we discard them and switch to our probabilistic embedding operator for this graph. As the number of embeddings of a pattern is polynomial in the number of patterns of its predecessor for any given graph, this can be implemented efficiently. This adaptive algorithm could preserve the speed of e.g. Gaston on chemical graph databases, respectively that of our algorithm on other graph databases (with a small overhead).

As long as this algorithm is not implemented, though, you may follow this guideline: If you have a tree database, FreeTreeMiner [5] or PS [34]. If you have chemical graphs, use Gaston [26]. For all other graph databases, you may try Gaston-re [26], but should use PS [34].

Acknowledgments

This research was partially funded by the Federal Ministry of Education and Research of Germany as part of the competence center for machine learning ML2R (01—S18038CB) and was partially funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC 2070 – 390732324.

References

[1] T. Akutsu, "A polynomial time algorithm for finding a largest common subgraph of almost trees of bounded degree," *IEICE TFECCS*, vol. 76, no. 9, pp. 1488–1493, 1993.

[2] C. Borgelt and M. Berthold, "Mining molecular fragments: Finding relevant substructures of molecules," in *ICDM*, 2002.

[3] C. Borgelt, T. Meinl, and M. Berthold, "Moss: A program for molecular substructure mining," in *OSDM*, 2005.

[4] C. Chen, C. Lin, M. Fredrikson, M. Christodorescu, X. Yan, and J. Han, "Mining graph patterns efficiently via randomized summaries," *PVLDB*, vol. 2, no. 1, pp. 742–753, 2009. [Online]. Available: http://www.vldb.org/pvldb/2/vldb09-80.pdf

[5] Y. Chi, Y. Yang, and R. Muntz, "Indexing and mining free trees," in *ICDM*, 2003.

[6] ——, "HybridTreeMiner: an efficient algorithm for mining frequent rooted trees and free trees using canonical forms," in *SSDBM*, 2004.

[7] Y. Chi, R. Muntz, S. Nijssen, and J. Kok, "Frequent subtree mining - an overview," *Fund Inf*, vol. 66, no. 1–2, pp. 161–198, 2005.

[8] M. Chung, "$O(n^{2.5})$ time algorithms for the subgraph homeomorphism problem on trees," *J Alg*, vol. 8, no. 1, pp. 106–112, 1987.

[9] L. Cordella, P. Foggia, C. Sansone, and M. Vento, "Performance evaluation of the VF graph matching algorithm," in *ICIAP*, 1999.

[10] V. Dalmau, P. Kolaitis, and M. Vardi, "Constraint satisfaction, bounded treewidth, and finite-variable logics," in *CP*, 2002.

[11] M. Deshpande, M. Kuramochi, N. Wale, and G. Karypis, "Frequent substructure-based approaches for classifying chemical compounds," *TKDE*, vol. 17, no. 8, pp. 1036–1050, 2005.

[12] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, 1979.

[13] M. Hajiaghayi and N. Nishimura, "Subgraph isomorphism, log-bounded fragmentation, and graphs of (locally) bounded treewidth," *J Comp Syst Sci*, vol. 73, no. 5, pp. 755–768, 2007.

[14] T. Horváth and J. Ramon, "Efficient frequent connected subgraph mining in graphs of bounded tree-width," *Theor Comp Sci*, vol. 411, no. 31–33, pp. 2784–2797, 2010.

[15] T. Horváth, B. Bringmann, and L. D. Raedt, "Frequent hypergraph mining," in *ILP*. Springer, 2007.

[16] J. Huan, W. Wang, and J. Prins, "Efficient mining of frequent subgraphs in the presence of isomorphism," in *ICDM*, 2003.

[17] C. Jiang, F. Coenen, and M. Zito, "A survey of frequent subgraph mining algorithms," *Knowl Eng Rev*, vol. 28, no. 1, pp. 75–105, 2013.

[18] D. Johnson, C. Papadimitriou, and M. Yannakakis, "On generating all maximal independent sets," *Inf Proc Let*, vol. 27, no. 3, pp. 119–123, 1988.

[19] M. Kuramochi and G. Karypis, "An efficient algorithm for discovering frequent subgraphs," *TKDE*, vol. 16, no. 9, pp. 1038–1051, 2004.

[20] R. Li and W. Wang, "REAFUM: representative approximate frequent subgraph mining," in *SDM*, 2015.

[21] A. Lingas, "An application of maximum bipartite c-matching to subtree isomorphism," in *CAAP*, 1983.

[22] D. Marx and M. Pilipczuk, "Everything you always wanted to know about the parameterized complexity of Subgraph Isomorphism (but were afraid to ask)," in *STACS*, 2014.

[23] J. Matoušek and R. Thomas, "On the complexity of finding iso-and other morphisms for partial k-trees," *Disc Math*, vol. 108, no. 1–3, pp. 343–364, 1992.

[24] D. Matula, "Subtree isomorphism in o(n5/2)," *Ann Disc Math*, vol. 2, pp. 91–106, 1978.

[25] S. Nijssen and J. Kok, "A quickstart in frequent structure mining can make a difference," in *KDD*, 2004.

[26] ——, "The gaston tool for frequent subgraph mining," *Electronic Notes in Theor Comp Sci*, vol. 127, no. 1, pp. 77–87, 2005.

[27] A. Petermann, M. Junghanns, and E. Rahm, "Dimspan - transactional frequent subgraph mining with distributed in-memory dataflow systems," *CoRR*, vol. abs/1703.01910, 2017. [Online]. Available: http://arxiv.org/abs/1703.01910

[28] U. Rückert and S. Kramer, "Frequent free tree discovery in graph data," in *SAC*, 2004.

[29] T. Schulz, T. Horváth, P. Welke, and S. Wrobel, "Mining tree patterns with partially injective homomorphisms," in *ECMLPKDD*, 2018.

[30] R. Shamir and D. Tsur, "Faster subtree isomorphism," *J Alg*, vol. 33, no. 2, pp. 267–280, 1999.

[31] J. Ullmann, "An algorithm for subgraph isomorphism," *JACM*, vol. 23, no. 1, pp. 31–42, 1976.

[32] L. Valiant, "The complexity of computing the permanent," *Theor Comp Sci*, vol. 8, pp. 189–201, 1979.

[33] R. Verma and S. Reyner, "An analysis of a good algorithm for the subtree problem, corrected," *SIAM J Comp*, vol. 18, no. 5, pp. 906–908, 1989.

[34] P. Welke, T. Horváth, and S. Wrobel, "Probabilistic frequent subtrees for efficient graph classification and retrieval," *Mach Learn*, vol. 107, no. 11, pp. 1847–1873, 2018.

[35] ——, "Probabilistic and exact frequent subtree mining in graphs beyond forests," *Mach Learn*, vol. 108, no. 7, pp. 1137–1164, 2019.

[36] P. Welke, F. Seiffarth, M. Kamp, and S. Wrobel, "HOPS: Probabilistic subtree mining for small and large graphs," in *KDD*, 2020.

[37] M. Wörlein, T. Meinl, I. Fischer, and M. Philippsen, "A quantitative comparison of the subgraph miners MoFa, gSpan, FFSM, and gaston," in *PKDD*, 2005.

[38] X. Yan and J. Han, "gSpan: Graph-based substructure pattern mining," in *ICDM*, 2002.

[39] M. Zaki, "Efficiently mining frequent trees in a forest," in *KDD*, 2002.

[40] P. Zhao and J. Yu, "Fast frequent free tree mining in graph databases," *WWW*, vol. 11, no. 1, pp. 71–92, 2008.

[41] Z. Zou, J. Li, H. Gao, and S. Zhang, "Mining frequent subgraph patterns from uncertain graph data," *TKDE*, vol. 22, no. 9, pp. 1203–1218, 2010.