Nearly exact mining of frequent trees in large networks

Ashraf M. Kibriya and Jan Ramon

Department of Computer Science
Katholieke Universiteit Leuven
Leuven, Belgium
ashraf.kibriya@cs.kuleuven.be, jan.ramon@cs.kuleuven.be

Abstract. Mining frequent patterns in a single network (graph) poses a number of challenges. Already only to match one path pattern to a network (upto subgraph isomorphism) is NP-complete. Matching algorithms that exist, become intractable even for reasonably small patterns, on networks which are large or have a high average degree. Based on recent advances in parameterized complexity theory, we propose a novel miner for rooted trees in networks. The miner, for a fixed parameter k (maximal pattern size), can mine all rooted trees with *delay* linear in the size of the network and only mildly exponential in the fixed parameter k (2^k). This allows us to mine tractably, rooted trees, in large networks such as the WWW or social networks. We establish the practical applicability of our miner, by presenting an experimental evaluation on both synthetic and real-world data.

1 Introduction

Mining frequent patterns is one of the fundamental tasks of data mining. Traditionally, patterns have consisted of simple sets of items. However, since the last decade interest has been building up in mining more structured forms of patterns, such as trees and arbitrary graphs. This has especially been the case due to the phenomenal growth of structured data sources such as the WWW, and social and citation networks.

There are generally two settings for mining in graphs. A first graph mining setting is the transactional setting, where we are given a set of graphs and a threshold t, and we want to find patterns that occur in at least t graphs in the set. The second graph mining setting, which we will consider in this paper, is the single network setting, where we are given a single graph and a threshold t, and we want to find patterns that have a support of at least t in the given single graph according to some appropriate frequency measure.

Central to any pattern mining task is the notion of pattern matching. In graph mining, subgraph isomorphism is usually the matching operator of choice. Checking for even a simple path in a graph under subgraph isomorphism is well known to be NP-complete. Indeed, the problem of finding a path in a graph can be reduced to the Hamiltonian path problem. However, recent advances in the

theory of parameterized complexity have produced algorithms that can tractably solve several of the computationally hard graph problems, including subgraph isomorphism, when certain parameters of the problems are bounded. The work of [20] is particularly relevant in this case. It gives a randomized algorithm for deciding subgraph isomorphism of a tree in a network with an asymptotic complexity of $O(k^2 \log^2(k) m 2^k)$, where m is the number of edges in the network and k the size of the pattern.

In this paper we build on the work of [20]. We present an algorithm to mine *all* frequent rooted tree patterns with delay $O(k^2 \log^2(k) m 2^k)$, i.e. the time between any two consecutive frequent patterns being output is bounded by $O(k^2 \log^2(k) m 2^k)$. We present an implementation and experiments on both synthetic and real-world data. To the best of our knowledge, our work is the first to tractably mine tree patterns under subgraph isomorphism in single networks.

The rest of the paper is structured as follows. In the next section we give a brief overview of related work, followed by a section of preliminaries required for explaining our work. We then proceed to presenting our work by building on the work of [20], followed by a thorough experimental evaluation to establish the practical applicability of our mining algorithm. We then conclude with some final remarks, and future direction of our research.

2 Related Work

Most of the work done so far in graph mining is in the transactional setting. For example, graph miners AGM [18], FSG [21], FFSM [16], gSpan [30], MoFa/MoSS [3, 4] and Gaston [25], are all designed for the transactional setting. These miners employ a variety of optimizations to reduce the overall runtime and memory use, including canonical forms to avoid duplicate generation of candidates and extraneous isomorphism tests. The earlier generation of miners, AGM and FSG, work in an apriori style fashion, whereas the newer generation (FFSM and beyond) use depth first search (instead of apriori style breadth-first) to further optimize memory use [29]. Furthermore, as in itemset mining, considerable work has also been done in pattern summarization, by mining more representative and comprehensive graph patterns such as closed graphs [31, 7] and maximal graphs [17, 27].

For mining in single networks, work has so far been limited. To our knowledge, the only ones that exist are for mining evolution of networks [1, 2], node/edge classification [15, 12], or mining that uses homomorpshism as matching operator [11]. Even though in [1] the authors mine patterns when mining their evolution rules, the language of the patterns is specific to time evolving networks, and excludes more general (unlabeled) patterns like trees or cycles.

For pattern matching, with (sub)graph isomorphism as the matching operator, usually the miners use one of a number of base matching algorithms. For general arbitrary graphs, Ullman[28] and VF2[8] are often popular choices, whereas Nauty[22] is also sometimes used if the matching operator is restricted graph isomorphism. Ullman and VF2 are both branch-and-bound based algo-

rithms that employ backtracking and pruning strategies to eliminate large parts of the search space. Nauty on the other hand, uses results from group theory to create unique canonical labelings for (automorphic) graphs that are then compared for equivalency. Note that graph isomorphism is just a special case of subgraph isomorphism, and any method for subgraph isomorphism can generally also be used for graph isomorphism.

VF2 is newer and in comparison provides mostly better runtime than Ullman, whereas in comparison to Nauty (on graph isomorphism) it is usually better on real-world structured graphs [9].

An alternative to subgraph isomorphism is to use homomorphism. Homomorphism, as used in [11], has a lower computational cost (only polynomial in the pattern size), but has other disadvantages. One of the major reasons why interest in mining conjunctive queries has been declining is that candidate generation is problematic, as illustrated by the fact that no optimal refinement operator exists [23, 24]. Moreover, depending on the application requiring pattern vertices to map to different network vertices (as in isomorphism) may be the most natural choice.

3 Preliminaries

3.1 Graphs

We recall basic graph theoretic notions used in this paper. For more background in this area, see also [10]. A *graph* is a pair $G = (V_G, E_G)$, where $V_G \neq \emptyset$ is a finite set of vertices/nodes, and $E_G \subseteq \{\{x, y\} \mid x, y \in V_G\}$ a set of edges connecting those vertices. For any graph G its vertex and edge set will also be referred to as $V(G)$ and $E(G)$ respectively. If $\{u, v\} \in E(G)$, we say u and v are *adjacent* vertices and the edge $\{u, v\}$ is *incident* with the vertex v. In this paper, we call $|V(G)|$ the *size* of G. A *path* in a graph G is a sequence $\{v_1, v_2, \ldots, v_n\}$ of pairwise distinct vertices of G such that $\{v_i, v_{i+1}\} \in E(G)$ for all $1 \leq i < n$. A *tree* is a graph such that there is a unique path between any pair of its vertices. A rooted tree is a tree T in which a single vertex $r \in V(T)$, denoted by $root(T)$, is distinguished and is called the root.

A *labeled graph* is a quadruple $G = (V_G, E_G, \Sigma_G, \lambda_G)$, where (V_G, E_G) is a graph, $\Sigma_G \neq \emptyset$ a set of labels, and $\lambda_G : V_G \cup E_G \to \Sigma_G$ a function assigning labels to vertices and edges.

A graph $H = (V_H, E_H)$ is called a *subgraph* of $G = (V_G, E_G)$, if $V_H \subseteq V_G$ and $E_H \subseteq E_G$. It is an *induced* subgraph if for all $\forall v \in V_H, (v, v') \in E_H \Leftrightarrow (v, v') \in E_G$, otherwise it is a *non-induced* subgraph.

A graph $H = (V_H, E_H)$ is said to be *isomorphic* to $G = (V_G, E_G)$ (denoted by $H \cong G$), if there exists an edge-preserving bijective mapping of H onto G. For labeled graphs the mapping, in addition to edge-preserving, also has to be label-preserving. Formally, $H \cong G$ if there exists a function $\varphi : V_H \to V_G$ such that $\forall u, v \in V_H, (u, v) \in E_H \Leftrightarrow (\varphi(u), \varphi(v)) \in E_G$, and for the labeled case additionally, $\forall u \in V_H, \lambda_H(u) = \lambda_G(\varphi(u))$. If H is isomorphic to a subgraph of

G, then we call H *subgraph isomorphic* to G and write $H \preceq G$. In that case, the mapping is called an *embedding* of H in G.

If a mapping from H to G is edge-preserving but not bijective (and hence not one-to-one), then it defines a *homomorphism* between H and G.

We denote with $\text{Emb}(H, G)$, the set of all isomorphic embeddings of H in G. Note, that (i) the number of embeddings $|\text{Emb}(H, G)|$ can be exponential, and (ii) that in this paper we consider normal subgraph isomorphism rather than the more restrictive induced subgraph isomorphism.

3.2 Group Theory

A *group* G is a set of elements endowed with an arbitrary binary operation $(\cdot)$, such that it satisfies the following four properties, known as the group axioms:

- **Closure.** If $a, b \in G$, then so does $a \cdot b$.
- **Associativity.** $(a \cdot b) \cdot c = a \cdot (b \cdot c)$.
- **Identity.** There exists an element $i \in G$, such that for every $a \in G$: $a \cdot i = i \cdot a = a$.
- **Inverse.** For every $a \in G$, there exists an element a^{-1}, such that $a \cdot a^{-1} = i$

A *ring* is a set of elements with two specified binary operations, addition $(+)$ and multiplication $(\times)$. It must satisfy all the group axioms for $(+)$, all but the inverse axiom for $(\times)$, and the following additional axioms:

- **Commutativity of** $(+)$**.** For all $a, b \in G$, $a + b = b + a$.
- **Distributivity of** $(\times)$ **over** $(+)$**.** For all $a, b, c \in G$, $a \times (b + c) = a \times b + a \times c$ and $(a + b) \times c = a \times c + b \times c$.

If in addition to the above if it also satisfies commutativity of $(\times)$, then it is called a *commutative ring*, otherwise a *non-commutative ring*.

A *field* is a commutative ring for which also the inverse axiom for $\times$ holds.

For a ring R, an integer n and $a \in R$, $n.a = \sum_{i=1}^{n} a$ is called the scalar multiplication between n and a. For any finite field F, it is necessarily the case that there exists an integer $n > 0$, such that for every $a \in F$, $na = 0$. The smallest such n for a field is called its *characteristic*.

4 Problem Statement

Let $\mathcal{G}$ be the *language* of all graphs, let $\mathcal{L}_{\mathcal{P}} \subseteq \mathcal{G}$ be a language of *patterns*, let $M(\mathcal{L}_{\mathcal{P}}, \mathcal{G})$ be some measure of interestingness, let t be a given threshold of interestingness and $G \in \mathcal{G}$ be the network we want to mine patterns in. Then, we would like to compute the set $\mathcal{F}_{(\mathcal{L}_{\mathcal{P}}, \mathcal{G})}$ of interesting patterns defined by:

$$\mathcal{F}_{(\mathcal{L}_{\mathcal{P}}, \mathcal{G})} = \{T \in \mathcal{L}_{\mathcal{P}} : M(T, G) \geq t\}$$

In our case, our pattern language is the class of rooted trees.

Several frequency measures have been proposed, as measures of interestingness, in the literature for single graph mining [5, 6]. This paper primarily focuses on the matching of patterns, and though our methods are general, for simplicity we restrict ourselves to the frequency measure obtained by counting the number of possible images of the root of a rooted tree pattern. This support measure is defined as follows.

Definition 1 (root image). *The root image of a rooted tree T in G is the set of all vertices $v \in G$ to which $root(T)$ can be mapped under subgraph isomorphism, i.e.,*

$$\mathcal{RI}(T, G) = \{\varphi(root(T)) \mid \varphi \in Emb(T, G)\},$$

Definition 2 (support). *Let T be a rooted tree and G be a graph. Then, we define the support of T in G as the size of its root image, i.e.,*

$$supp(T, G) = |\mathcal{RI}(T, G)|.$$

This support measure is anti-monotone w.r.t. increasing pattern size.

For the remainder of the paper, we will consider a network G and for brevity we will use $n = |V(G)|$ and $m = |E(G)|$. Moreover, the symbol T will be used to refer to rooted tree patterns and denote its size with $k = |V(T)|$. We will abuse terminology and use 'tree' for 'rooted tree' if it is clear from the context.

5 Mining Frequent Rooted Trees

In order to realise a pattern miner for rooted trees in single networks, the two most important ingredients are efficient generation of candidate trees and frequency counting of candidates in the network (using subgraph isomorphism). We proceed by first outlining our candidate generation method, and then reviewing the subgraph isomorphism method of [20] and showing how it can be employed to compute the above defined frequency measure. Finally we give complexity bounds for our complete miner.

5.1 Candidate Generation

In our miner, we use the same technique for generating rooted trees as in [25, 26]. It generates rooted ordered candidate trees that are *left heavy*, i.e. children of a node are ordered and each left sub-tree is larger than the right sub-tree according to their canonical form. The left heavy property avoids generating trees that are isomorphically equivalent.

The method works by adding nodes only to the right most path, and ensuring that the condition of left heavy subtree is met with each new added node. The method thus produces a new tree for each added node, and can do so with delay $O(k)$ for size k trees. The left heavy subtree condition is met by maintaining a canonical form for the trees, which for unlabeled case is just the depth sequence of nodes in pre-order traversal. If the trees are labeled, then vertex and edge labels are inserted. Figure 1 gives some example trees with their corresponding depth sequence to illustrate the technique.

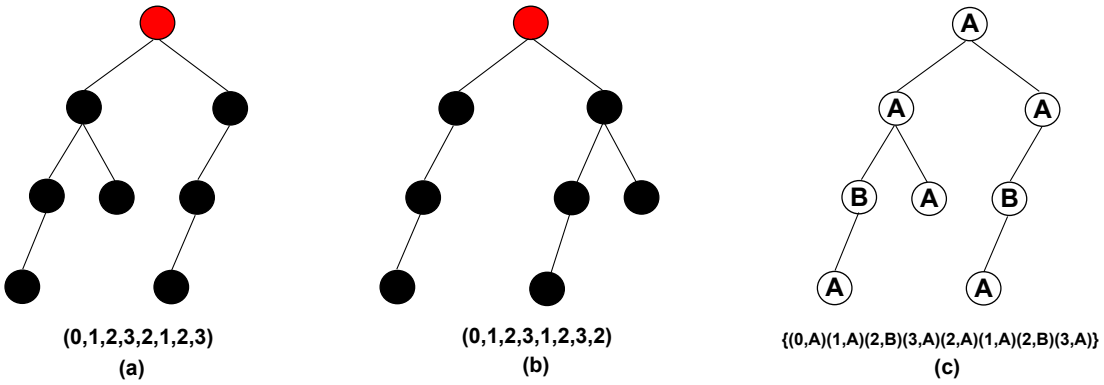

Fig. 1: The (rooted) unlabeled trees (a) and (b) are isomorphic to each other, but (a) is left-heavy compared to (b). We can treat as canonical form, the lexicographically heaviest string of pre-order traversal of depth sequence (given below each tree), and generate only trees that are like (a). Figure (c) shows an example of a left-heavy *labeled* tree with its corresponding canonical string.

5.2 Subgraph isomorphism and frequency counting

Let us first briefly outline the subgraph isomorphism method of [20], which from here on we will call the Koutis&William's method. The method exploits the fact that for trees, subgraph homomorphisms can be computed in polynomial time. The method essentially consists of two core parts. In the first part, it constructs an arithmetic circuit computing a polynomial P representing all possible homomorphisms of a tree in a network. In particular, with every network vertex v a variable x_v is associated, and every homomorphism π from the pattern T to the network G corresponds to a term (monomial) $\prod_{v \in V(T)} x_{\pi(v)}$ in the polynomial, i.e. the product of the variables corresponding to the images of the vertices of the pattern. A multilinear term is a term where every variable occurs with degree at most 1. Isomorphisms are injective, and therefore the terms in P corresponding to isomorphisms will be exactly the multi-linear terms of P. In the second part, the method then evaluates the polynomial on an appropriate commutative group algebra, that ensures squares evaluate to 0. Hence, all terms which are not multi-linear (i.e. all homomorphisms which are not isomorphisms) vanish. The randomization of the values for which the variables x_v are substituted is such that multi-linear terms evaluate to non-zero with probability at least $1/4$ and the randomization of the coefficients of the polynomial P is such that the summation of non-zero monomials evaluates to non-zero with probability at least $7/8$.

In particular, [20] evaluates the polynomial P over $GF(2^l)\mathbb{Z}_2^k$. $\mathbb{Z}_2^k$ contains all bitvectors of length k. For $x, y \in \mathbb{Z}_2^k$, the multiplication is defined by componentwise addition of the elements of the bit vectors. The neutral element, the vector containing k zeros, is denoted W_0. Then, $GF(2^l)\mathbb{Z}_2^k$ is the ring of linear combi-

nations of elements of $\mathbb{Z}_2^k$ with coefficients from $GF(2^l)$, the unique field with 2^l elements. $GF(2^l)$ has characteristic 2, i.e. $x + x = 0$ holds for any x.

The polynomial is evaluated by assigning to each variable x_v a value $W_0 + y_v$ where y_v is a random value from $\mathbb{Z}_2^k$ (i.e. a random k-bit vector). The result of [20] is based on the following observations:

- For a set $S \subseteq V(G)$ and variables $x_v = W_0 + y_v$ with $y_v \in \mathbb{Z}_2^k$, it holds that $\prod_{v \in S} x_v \neq 0$ iff the multiset $\{y_v | v \in S\}$ is a set of linearly independent vectors. Non-multilinear terms therefore evaluate to 0.
- A set of k randomly chosen bitvectors of length k is independent with probability at least $1/4$.
- For any set of element $b_i \in GF(2^l)\mathbb{Z}_2^k$ and randomly chosen coefficients $a_i \in GF(2^l)$, if any of the b_i is non-zero, then $\sum_i a_i b_i$ is nonzero with probability $1/2^l$.

Figure 2 gives an illustration of the above concept. It shows the mapping of a rooted tree to a network, and the corresponding polynomial for this mapping. The two multi-linear terms $x_1x_2x_3$ in the polynomial represent isomorphisms, while the rest represent homomorphisms.

In Algorithm 1, we outline the subgraph isomorphism method of [20]. The **occur** method in Algorithm 1 defines an arithmetic circuit of a polynomial for all homomorphic mappings starting from the mapping of root $r \in T$ to some $v' \in G$. The creation and evaluation of such circuits for all $v' \in G$, in method **countFreq**, gives us our above defined support measure of root images, for our root $r \in T$.

The $a_{r,j}$ and x_j, in the **occur** method, are chosen randomly from $\mathbb{Z}_2^k$ and $GF(2^l)$, and the arithmetic on the elements of array $C_{i,j}$ is performed based on their defined group algebra. In our implementation we use the representation-theoretic technique similar to [19] for doing the evaluations is memory linear in k (rather than linear in 2^k).

As per [20], the theoretical space complexity of **occur** method is $O(km)$ and its time complexity is $O(k^2m2^kl^2)$[1]. By extension the time complexity of **countFreq** method would be $O(k^2mn2^kl^2)$. However, we note that it is possible with only a single evaluation of the arithmetic circuit to obtain the values **occur(*T,G,r,j*)** for all j, and hence the time complexity is only $O(k^2m2^kl^2)$.

Remark 1. The **occur** method's success probability $p > 1/5$ can be increased to an arbitrary p', by repeating the method $\lceil log(1-p')/log(1-p) \rceil$ times (thereby decreasing the probability of failure) .

5.3 Complete miner and complexity bounds

Algorithm 2 gives pseudo-code for our complete pattern miner. It brings together all the core components to make an apriori style miner.

We now proceed to prove its theoretical bounds. Note, the space complexity of our miner is bounded by our main datastructure $C_{i,j}$, and is $O(kn)$.

[1] $O(km)$ being the size of the circuit and $O(k2^kl^2)$ to do arithmetic over $GF(2^l)[\mathbb{Z}_2^k]$

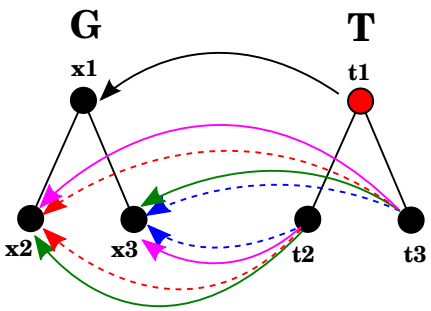

Fig. 2: All homomorphisms of T unto G, when mapping of $t1 \rightarrow x1$ is fixed. Solid lines represent mappings that are also isomorphic.

Theorem 1. *Given a network G with n nodes and m edges labeled by label set Σ_G, and a frequency threshold t we can mine all frequent tree patterns of size $\leq k$ with time $O(|\Sigma_G| \log^2(k) k^2 m 2^k)$*

Proof. As mentioned earlier, the **countFreq** method takes $O(\log^2(k) k m 2^k)$ time for each candidate tested. At each call made at level $i \geq 2$ the generateCandidates function in algorithm 2 produces at most $O(|\Sigma_G|)$ candidates per frequent pattern in S_{i-1}. For each such candidate, **countFreq** is called, taking time $O(\log^2(k) k^2 m 2^k)$. Therefore, for each new solution output, the algorithm will need $O(|\Sigma_G| \log^2(k) k^2 m 2^k)$ more time to finish. The theorem follows by noting that we can delay the printing of the solutions (frequent patterns) in such a way that between the printing of each pair of consecutive solutions the time is bounded by $O(|\Sigma_G| \log^2(k) k^2 m 2^k)$.

5.4 Further Optimizations

A number of optimizations are still possible with algorithm 2. Below we mention some of the more high level optimizations we implemented in our miner.

- **Sharing common subtrees.** Note that for each C_i for $i \geq 2$, the candidate trees may share a number of subtrees common among them. We do not need to test each candidate tree in isolation. Instead we can reuse the previously computed results of the common subparts.
  In our implementation we made a directed acyclic graph structure to represent all $T \in C_i$, i.e. a tree is represented by a node of the directed acyclic graph (the root) and all nodes below it. Several nodes can be parents of the same node and hence the corresponding trees can share subtrees. This method shares computation at the expense of additional memory.

Algorithm 1 Count frequency of tree T in a network G

let $C_{i,j}$ be an array containing the result of mapping each $v_i \in V_T$ to each $v_j \in V_G$

function occur(T, G, r, j):
if $C_{r,j}$ is filled **then**
  return $C_{r,j}$
else if $\lambda_T(v_r) \neq \lambda_G(v_j)$ **then**
  let $C_{r,j} := 0$.
else
  let $C_{r,j} := a_{r,j}.x_j$ {where x_j is a randomly chosen k-bit vector and $a_{r,j}$ a random scalar}
  if $|V(T)| > 1$ **then**
    let $S_{T'} :=$ {subtrees after removing v_r from T}
    $C_{r,j} := C_{r,j}. \prod_{T' \in S_{T'}} \left(\sum_{j':(v_j, v_{j'}) \in E_G} occur(T', G, r', j') \right)$.
  end if
end if
return $C_{r,j}$

function countFreq(T, G):
let: $v_r \in V_T$ be the root of T
$freq := 0$
for $j = 1$ to $|V_G|$ **do**
  if occur(T, G, r, j) **then**
    $freq = freq + 1$
  end if
end for
return $freq$

– **Checking for homomorphisms.** As mentioned earlier, homomorphisms for trees in networks can be found in polynomial time. In fact in our case all we have to do is evaluate our circuit over the infinite field of integers, instead of the group algebra $GF(2^l)[\mathbb{Z}_2^k]$, thereby avoiding all the expensive arithmetic. Evaluation is linear in the size of the circuit and is only $O(km)$. We can store and use results of these inexpensive tests, and avoid the more expensive isomorphism tests for any part of network and common subtrees that are not homomorphic. In case of labeled networks especially, this can offer considerable speedup.

6 Experimental Evaluation

6.1 Experimental setup

In our experimental evaluation, we are interested in the following experimental questions:

Q1 What size of patterns and networks can our algorithm handle within reasonable time?

Algorithm 2 Find all patterns of size upto k

```
1: function findPatterns(G, k, t, Σ):
2: T := ∅ {set of all frequent trees}
3: S_{1...k} := ∅ {frequent trees of size 1...k}
4: C_{1...k} := ∅ {candidate trees of size 1...k}
5: for i = 1 to k do
6:    if i = 1 then
7:       C_1 := {single vertex graphs labeled with a label in Σ}
8:    else
9:       C_i := generateCandidates(S_{i-1}, Σ)
10:   end if
11:   S_i := {c_j : c_j ∈ C_i ∧ countFreq(c_j, G) ≥ t}
12:   T := T ∪ S_l
13: end for
14: return T
```

Q2 How does our pattern matching strategy compare to state of the art strategies, in particular with VF2[8]?
Q3 Does our implementation scale as well as Koutis-William's theoretical algorithm?
Q4 What is the influence of pattern mining parameters and optimizations?

To perform our experiments, we implemented a system which we will call MINT (MIning Networks for Trees), containing a breadth-first pattern mining algorithm using the candidate generation method outlined in Section 5.1. We implemented the frequency counting based Koutis&William's algorithm as described in Section 5.2, and a baseline MINT-VF2 using frequency counting based on the VF2 algorithm[8]. We consider several versions of our new algorithm: First, MINT-STD implements a vanilla version of Koutis&William's algorithm. Second, MINT-HOMO implements Koutis&William's with homomorphism checking optimization, and third, MINT-BATCH which includes homomorphism checking as well as sharing common subtrees among the candidates. We call the last one MINT-BATCH, as we share subtrees only among *batchsize* number candidates in each pass; otherwise the memory requirements get intractably large due to the exponential number of frequent patterns.

In order to be able to compare the randomized algorithm to the deterministic VF2, the subgraph isomorphism tests were repeated a sufficient number of times to achieve a very high probability of success ($1 - 10^{-6}$). The result was that in all cases except one, the randomized algorithm found the same set of frequent patterns as the deterministic one (the only exception was MINT-STD which classified one out of 124,687 frequent patterns of size 7 as infrequent for the 10^2 network in Table 2).

6.2 Data sets

We present results on both synthetic as well as real-world data.

Table 1: Real datasets' summary

Dataset	# vertices	# edges	# vertex labels	# edge labels	Avg. degree
Facebook-uniform	984,830	185,508	17	1	0.38
Facebook-mhrw	957,359	1,792,188	16	1	3.74
Dblp-9202	129,073	277,081	1	11	4.29
Dblp-0305	109,044	233,961	1	3	4.29
Dblp-0507	135,116	290,363	1	3	4.28
IMDB	30,835,467	53,686,381	144	1	1.74

For synthetic data we generated power-law graphs with degree distribution $P(d) \propto d^{-4}$. Such graphs show significant clustering, as is often seen in real-world data. We generated networks of size $n = \{10^2, 10^3, 10^4, 10^5, 10^6, 10^7\}$, and then randomly assigned 1 of 4 labels to each of the vertices.

For real-world data, we used the DBLP citation network[2], the Facebook social network[3], and the IMDB movie database[4]. The DBLP data is a snapshot of their citation network from 1992-2007. It is the same data as was used in [1]. The Facebook data is the Facebook social network obtained through random sampling (one through uniform sampling, and the other through independent Metropolis-Hastings random walks [13]). For IMDB, we extracted the movie-actor network from the raw database. Our extracted network consists of movie, year, role and actor nodes. Movie and role nodes were labeled by movie and role type, whereas year nodes were labeled by the year the movie was released in. Actor nodes are left with a default label. Also, Table 1 gives basic statistics of our real-world networks.

6.3 Results

Complete mining of synthetic data We ran the algorithms on synthetic datasets, and mined for as large patterns as we could in 10 hours. Table 2 gives the number of frequent patterns found in that time period for frequency threshold **0.1**, as a function of the network size and pattern size. It is noteworthy that as the network size grows, due to the asymptotic properties of the powerlaw graphs the number of frequent patterns of a given size converges. Figure 3 plots for each network size the total time used against the pattern size, for each of the considered algorithms. Note, that we could not run the MINT-BATCH for larger networks, due to its large memory requirements.

Sampled frequent patterns of synthetic data The number of patterns grows exponentially. Nevertheless, large patterns may be of interest. A strategy which gained popularity recently [14] is to not mine all frequent patterns but only generate a sample of them. Here, we adopt a simple sampling strategy of randomly

[2] http://www-kdd.isti.cnr.it/GERM/
[3] http://odysseas.calit2.uci.edu/doku.php/public:online_social_networks
[4] http://www.imdb.com/interfaces

Table 2: Number of frequent patterns for synthetic data

network size / pattern size	$\mathbf{10^2}$	$\mathbf{10^3}$	$\mathbf{10^4}$	$\mathbf{10^5}$	$\mathbf{10^6}$	$\mathbf{10^7}$
1	3	2	2	2	2	2
2	10	4	4	4	4	4
3	48	22	22	22	22	22
4	295	144	142	142	142	142
5	2077	1076	1066	1066	1066	
6	15,698	8605	8534	8534		
7	124,687	72084				
8	1,024,557					

Table 3: Number of frequent patterns for real data

network / pattern size	FB-uniform	FB-mhrw	Dblp0305	Dblp0507	Dblp9202	IMDB
1	2	1	1	1	1	6
2	1	2	3	3	8	10
3	2	5	12	13	10	38
4	3	11	51	57	10	149
5	4	30	189	277	6	692
6	5	88	648	1099	1	
7	10				0	
8	15					

selecting only 100 frequent patterns at each level (denoted pattern size in our breadth-first mining) of the mining process, to make extensions for the next level. This experiment allows us to study more closely the delay (time used per pattern found) of our miner.

Figure 4 plots for each network the delay (time used per pattern) as a function of the size of the patterns, and also as a function of the size of the network for patterns of size 4, for each of the considered algorithms.

Real-world datasets Here we followed essentially the same procedure as for synthetic data, the only differences being that a smaller frequency threshold of 0.05 for FB-uniform and IMDB was used to allow for larger patterns to be mined, and that a higher cut-off point for runtime was used (we allowed 16 hours for DBLP, 24 hours for Facebook, and 48 hours of runtime for IMDB data). Table 3 lists the number of frequent patterns found for each network. Figure 5 plots for each network the total time used against the pattern size, for each of the considered algorithms.

6.4 Discussion

Based on the results reported above, we can answer the experimental questions as follows:

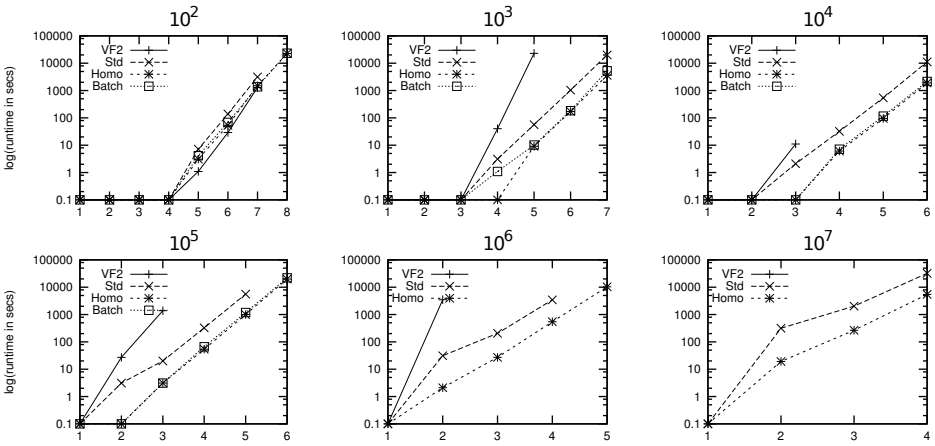

Fig. 3: log-runtime as a function of pattern size

Q1 Using the new pattern matching method, it is computationally feasible to match patterns up to size 15 (see figure 4). The main bottleneck when mining all patterns is the number of frequent patterns found. As this number increases exponentially, in many settings we don't get further than size 5 patterns. One can observe however, that for real-world pattern mining tasks one often has prior domain knowledge allowing for pruning the search space towards the type of patterns one is interested in.

Q2 It is clear from all experiments that the new pattern method is orders of magnitude better than the VF2 algorithm, especially for larger patterns.

Q3 From figure 4 one can see that the pattern matching algorithm scales at least as well as the theoretical upper bound. In particular, in contrast to VF2, our new method scales linearly in the network size and scales indeed as $O(k^2 \log^2(k) 2^k)$ in the pattern size.

Q4 The homomorphism check prunes away a significant amount of subgraph isomorphism tests for patterns which are clearly infrequent. This especially holds for the real-world dataset.

7 Conclusion and Future Work

We present a novel algorithm for mining trees in single networks. It scales well with respect to network size, and is only mildly exponential in pattern size, which makes it tractable for moderately sized patterns. We show the effectiveness of the method in practice, on real as well as synthetic data.

As for future work, we expect that several heuristic optimizations are possible which can improve performance on real-world datasets. Furthermore, we would also like to extend our method to graph classes other than trees.

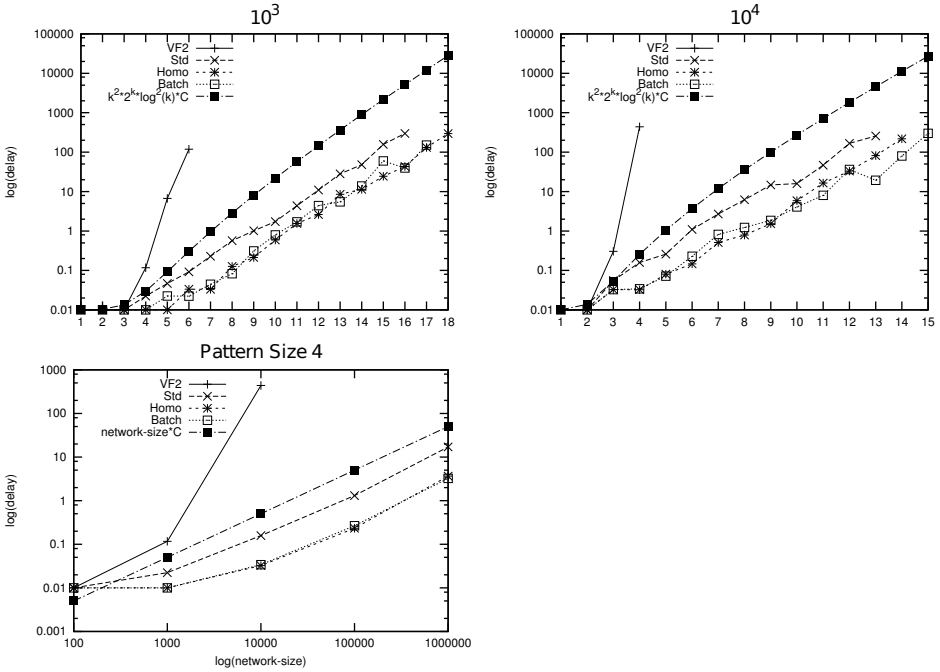

Fig. 4: log-delaytime (time per pattern) as a function of pattern size, as well as a function of network size

Acknowledgements

This research is supported by ERC Starting Grant 240186 "MiGraNT: Mining Graphs and Networks, a Theory-based approach". We thank Anton Dries and Constantin Commendant for the valuable suggestions.

References

1. Berlingerio, M., Bonchi, F., Bringmann, B., Gionis, A.: Mining graph evolution rules. In: Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases: Part I. pp. 115–130. ECML-PKDD '09, Springer-Verlag, Berlin, Heidelberg (2009)
2. Bogdanov, P., Mongiovì, M., Singh, A.K.: Mining heavy subgraphs in time-evolving networks. In: Proceedings of the 2011 IEEE 11th International Conference on Data Mining. pp. 81–90. ICDM '11, IEEE Computer Society, Washington, DC, USA (2011)
3. Borgelt, C., Berthold, M.R.: Mining molecular fragments: Finding relevant substructures of molecules. In: Proceedings of the 2002 IEEE International Conference on Data Mining. pp. 51–58. ICDM '02, IEEE Computer Society, Washington, DC, USA (2002)

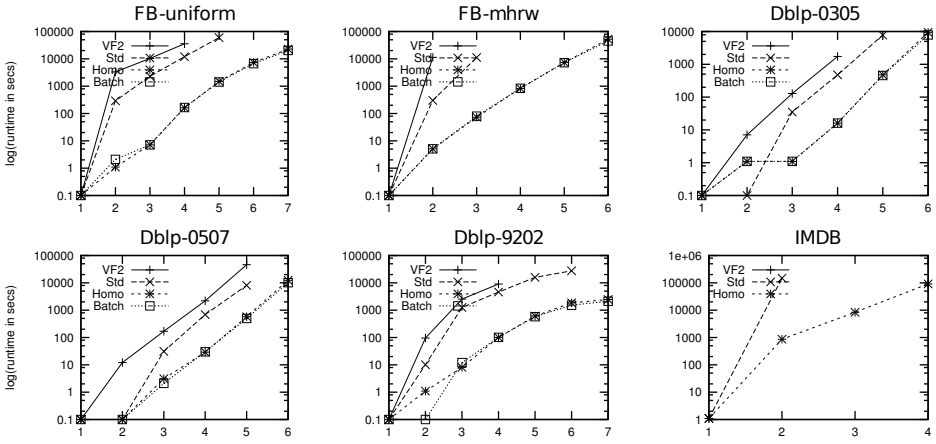

Fig. 5: log-runtime as a function of pattern size

4. Borgelt, C., Meinl, T., Berthold, M.: Moss: a program for molecular substructure mining. In: Proceedings of the 1st international workshop on open source data mining: frequent pattern mining implementations. pp. 6–15. OSDM '05, ACM, New York, NY, USA (2005)
5. Bringmann, B., Nijssen, S.: What is frequent in a single graph? In: Frasconi, P., Kersting, K., Wrobel, S. (eds.) Proceedings of MLG-2007: 5th International Workshop on Mining and Learning with Graphs. pp. 1–4 (2007)
6. Calders, T., Ramon, J., Van Dyck, D.: All normalized anti-monotonic overlap graph measures are bounded. Data Mining and Knowl. Disc. 23(3), 503–548 (2011)
7. Chi, Y., Xia, Y., Yang, Y., R. Muntz, R.: Mining closed and maximal frequent subtrees from databases of labeled rooted trees. IEEE Trans. on Knowl. and Data Eng. 17, 190–202 (2005)
8. Cordella, L.P., Foggia, P., Sansone, C., Vento, M.: An improved algorithm for matching large graphs. In: In: 3rd IAPR-TC15 Workshop on Graph-based Representations in Pattern Recognition, Cuen. pp. 149–159 (2001)
9. Cordella, L.P., Foggia, P., Sansone, C., Vento, M.: A (sub)graph isomorphism algorithm for matching large graphs. IEEE Transactions on Pattern Analysis and Machine Intelligence 26, 1367–1372 (2004)
10. Diestel, R.: Graph Theory. Springer, fourth ed., electronic edn. (2010)
11. Dries, A., Nijssen, S.: Mining Patterns in Networks using Homomorphism. In: Proceedings of the Twelfth SIAM International Conference on Data Mining,. pp. 260–271. Omnipress (Apr 2012), https://lirias.kuleuven.be/handle/123456789/350328
12. Gallagher, B., Tong, H., Eliassi-Rad, T., Faloutsos, C.: Using ghost edges for classification in sparsely labeled networks. In: Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 256–264. KDD '08, ACM, New York, NY, USA (2008)
13. Gjoka, M., Kurant, M., Butts, C., Markopoulou, A.: Walking in Facebook: A Case Study of Unbiased Sampling of OSNs. In: Proc. of IEEE INFOCOM '10 (2010)
14. Hasan, M.A., Zaki, M.J.: Output space sampling for graph patterns. Proceedings of the VLDB Endowment 2(1), 730–741 (2009)
15. Henderson, K., Gallagher, B., Li, L., Akoglu, L., Eliassi-Rad, T., Tong, H., Faloutsos, C.: It's who you know: graph mining using recursive structural features. In:

Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 663–671. KDD '11, ACM, New York, NY, USA (2011)
16. Huan, J., Wang, W., Prins, J.: Efficient mining of frequent subgraphs in the presence of isomorphism. In: Proceedings of the 2003 Third IEEE International Conference on Data Mining. pp. 549–556. ICDM '03, IEEE Computer Society, Washington, DC, USA (2003)
17. Huan, J., Wang, W., Prins, J., Yang, J.: Spin: mining maximal frequent subgraphs from graph databases. In: Proceedings of the 10th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 581–586. KDD '04, ACM, New York, NY, USA (2004)
18. Inokuchi, A., Washio, T., Motoda, H.: An apriori-based algorithm for mining frequent substructures from graph data. In: Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery. pp. 13–23. PKDD '00, Springer-Verlag, London, UK, UK (2000)
19. Koutis, I.: Faster algebraic algorithms for path and packing problems. In: Proceedings of ICALP '08. pp. 575–586. Springer (2008)
20. Koutis, I., Williams, R.: Limits and applications of group algebras for parameterized problems. In: Proceedings of ICALP '09. pp. 653–664. Springer-Verlag (2009)
21. Kuramochi, M., Karypis, G.: Frequent subgraph discovery. In: Proceedings of the 2001 IEEE International Conference on Data Mining. pp. 313–320. ICDM '01, IEEE Computer Society, Washington, DC, USA (2001)
22. McKay, B.D.: Practical graph isomorphism. Congr. Numerantium 10, 45–87 (1981)
23. Nienhuys-Cheng, S.H., De Wolf, R.: Foundations of Inductive Logic Programming, Lecture Notes in Computer Science and Lecture Notes in Artificial Intelligence, vol. 1228. Springer-Verlag, New York, NY, USA (1997)
24. Nijssen, S., Kok, J.: There is no optimal, theta-subsumption based refinement operator, personal communication
25. Nijssen, S., Kok, J.N.: A quickstart in frequent structure mining can make a difference. In: Proceedings of the 10th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 647–652. KDD '04, ACM, New York, NY, USA (2004)
26. Nijssen, S., Kok, J.N.: The gaston tool for frequent subgraph mining. Electronic Notes in Theoretical Computer Science 127(1), 77–87 (2005), proceedings of the International Workshop on Graph-Based Tools (GraBaTs 2004)
27. Thomas, L.T., Valluri, S.R., Karlapalem, K.: Margin: Maximal frequent subgraph mining. ACM Trans. Knowl. Discov. Data 4, 10:1–10:42 (2010)
28. Ullmann, J.: An algorithm for subgraph isomorphism. JACM 23(1), 31–42 (1976)
29. Wörlein, M., Meinl, T., Fischer, I., Philippsen, M.: A quantitative comparison of the subgraph miners mofa, gspan, ffsm, and gaston. In: Proceedings of the 9th European conference on Principles and Practice of Knowledge Discovery in Databases. pp. 392–403. PKDD '05, Springer-Verlag, Berlin, Heidelberg (2005)
30. Yan, X., Han, J.: gspan: Graph-based substructure pattern mining. In: Proceedings of the 2002 IEEE International Conference on Data Mining. pp. 721–724. ICDM '02, IEEE Computer Society, Washington, DC, USA (2002)
31. Yan, X., Han, J.: Closegraph: mining closed frequent graph patterns. In: Proceedings of the 9th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 286–295. KDD '03, ACM, New York, NY, USA (2003)