Mining Closed Frequent Free Trees in Graph Databases

Peixiang Zhao and Jeffrey Xu Yu

The Chinese University of Hong Kong, China
{pxzhao,yu}@se.cuhk.edu.hk

Abstract. Free tree, as a special graph which is connected, undirected and acyclic, has been extensively used in bioinformatics, pattern recognition, computer networks, XML databases, etc. Recent research on structural pattern mining has focused on an important problem of discovering frequent free trees in large graph databases. However, it can be prohibitive due to the presence of an exponential number of frequent free trees in the graph database. In this paper, we propose a computationally efficient algorithm that discovers only *closed frequent free trees* in a database of labeled graphs. A free tree t is *closed* if there exist no supertrees of t that has the same frequency of t. Two pruning algorithms, the *safe position pruning* and the *safe label pruning*, are proposed to efficiently detect unsatisfactory search spaces with no closed frequent free trees generated. Based on the special characteristics of free tree, the *automorphism-based pruning* and the *canonical mapping-based pruning* are introduced to facilitate the mining process. Our performance study shows that our algorithm not only reduces the number of false positives generated but also improves the mining efficiency, especially in the presence of large frequent free tree patterns in the graph database.

1 Introduction

Recent research on frequent pattern discovery has progressed from mining itemsets and sequences to mining structural patterns including (ordered, unordered, free) trees, lattices, graphs and other complicated structures. Among all these structural patterns, graph, a general data structure representing relations among entities, has been widely used in a broad range of areas, such as bioinformatics, chemistry, pattern recognition, computer networks, etc. In recent years, we have witnessed a number of algorithms addressing the frequent graph mining problem [5,9,4,6]. However, discovering frequent graph patterns comes with expensive cost. Two computationally expensive operations are unavoidable: (1) to check if a graph contains another graph (in order to determine the frequency of a graph pattern) is an instance of *subgraph isomorphism* problem, which is NP-complete [3]; and (2) to check if two graphs are isomorphic (in order to avoid creating a candidate graph for multiple times) is an instance of *graph isomorphism* problem, which is not known to be either P or NP-complete [3].

With the advent of XML and the need for mining semi-structured data, a particularly useful family of general graph — free tree, has been studied and

R. Kotagiri et al. (Eds.): DASFAA 2007, LNCS 4443, pp. 91–102, 2007.
© Springer-Verlag Berlin Heidelberg 2007

applied extensively in various areas such as bioinformatics, chemistry, computer vision, networks, etc. Free tree — the connected, undirected and acyclic graph, is a generalization of linear sequential patterns, and hence reserves plenty of structural information of databases. At the same time, it is a specialization of general graph, therefore avoids undesirable theoretical properties and algorithmic complexities incurred by graph. As the middle ground between two extremes, free tree has provided us a good compromise in data mining research [8,2].

Similar to frequent graph mining, the discovery of frequent free trees in a graph database shares a common *combinatorial explosion* problem: the number of frequent free trees grows exponentially although most free trees deliver nothing interesting but redundant information if all of them share the same frequency. This is the case especially when graphs of a database are strongly correlated.

Our work is inspired by mining *closed frequent itemsets and sequences* in [7]. According to [7,11], a frequent pattern $\mathcal{I}$ is *closed* if there exists no proper super-pattern of $\mathcal{I}$ with the same frequency in the dataset. In comparison to frequent free trees, the number of closed ones is dramatically small. At the same time, closed frequent free trees maintain the same information (w.r.t frequency) as that held by frequent free trees with less redundancy and better efficiency.

There are several previous studies on discovering closed frequent patterns among large tree or graph databases. **CMTreeMiner** [1] discovers all closed frequent ordered or unordered trees in a rooted-tree database by traversing an *enumeration tree*, a special data structure to enumerate all frequent (ordered or unordered) subtrees in the database. However, some elegant properties of ordered (unordered) trees do not hold in free trees, which makes infeasible to apply their pruning techniques directly to mine closed frequent free trees. **CloseGraph** [10] discovers all closed frequent subgraphs in a graph database by traversing a search space representing the complete set of frequent subgraphs. The novel concepts of *equivalent occurrence* and *early termination* help CloseGraph prune certain branches of the search space which produce no closed frequent subgraphs. We can directly use CloseGraph to mine closed frequent free trees because free tree is a special case of general graph, but CloseGraph will introduce a lot of inefficiencies. First, all free trees are computed as general graphs while the intrinsic characteristics of free tree are omitted; Second, the early termination may fail and CloseGraph may miss some closed frequent patterns. Although this failure of early termination can be detected, the detection operations should be applied case-by-case, which introduce a lot of complexities.

In this paper, we fully study the closed frequent free tree mining problem and develop an efficient algorithm, `CFFTree` which is short for **C**losed **F**requent, **F**ree **Tree** mining, to systematically discover the complete set of closed frequent free trees in large graph databases. The main contributions of this paper are: (1) We first introduce the concept of closed frequent free trees and study its properties and its relationship to frequent free trees; (2) Our algorithm `CFFTree` depth-first traverses the enumeration tree to discover closed frequent free trees. Two original pruning algorithms, the *safe position pruning* and the *safe label pruning* are proposed to prune search branches of the enumeration tree in the early

stage, which is confirmed to output no desired patterns; (3) Based on the intrinsic characteristics of free tree, we propose the *automorphism-based pruning* and the *canonical mapping-based pruning* to alleviate the expensive computation of equivalent occurrence sets and candidate answer sets during the mining process. We carried out different experiments on both synthetic data and real application data. Our performance study shows that `CFFTree` outperforms up-to-date frequent free mining algorithms by a factor of roughly 10. To the best of our knowledge, `CFFTree` is the first algorithm that, instead of using post-processing methods, directly mines closed frequent free trees from graph databases.

The rest of the paper is organized as follows. Section 2 provides necessary background and detailed problem statement. We study the closed frequent free tree mining problem in Section 3, and propose a basic algorithmic framework to solve the problem. Advanced pruning algorithms are presented in Section 4. Section 5 formulates our algorithm, `CFFTree`. In Section 6, we report our performance study and finally, we offer conclusions in Section 7.

2 Preliminaries

A *labeled graph* is defined as a 4-tuple $G = (V, E, \Sigma, \lambda)$ where V is a set of vertices, E is a set of edges (unordered pairs of vertices), Σ is a set of labels, and λ is a labeling function, $\lambda : V \cup E \to \Sigma$, that assigns labels to vertices and edges. A *free tree*, denoted *ftree*, is a special undirected labeled graph that is connected and acyclic. Below, we call a *ftree* with n vertices a n-*ftree*.

Let t and s be two *ftrees*, and g be a graph. t is a *subtree* of s (or s is the *supertree* of t), denoted $t \subseteq s$, if t can be obtained from s by repeatedly removing vertices with degree 1, a.k.a *leaves* of the tree. Similarly, t is a *subtree* of a graph g, denoted $t \subseteq g$, if t can be obtained by repeatedly removing vertices and edges from g. *Ftrees* t and s are *isomorphic* to each other if there is a one-to-one mapping from the vertices of t to the vertices of s that preserves vertex labels, edge labels, and adjacency. An *automorphism* is an isomorphism that maps from a *ftree* to itself. A *subtree isomorphism* from t to g is an isomorphism from t to some subtree(s) of g.

Given a graph database $\mathcal{D} = \{g_1, g_2, \ldots, g_N\}$ where g_i is a graph $(1 \leq i \leq N)$. The problem of *frequent ftree mining* is to discover the set of all frequent *ftrees*, denoted FS, where $t \in FS$ iff the ratio of graphs in $\mathcal{D}$ that has t as its subtree is greater than or equal to a user-given threshold ϕ. Formally, let t be a *ftree* and g_i be a graph. We define

$$\varsigma(t, g_i) = \begin{cases} 1 & \text{if } t \subseteq g_i \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

and

$$\sigma(t, \mathcal{D}) = \sum_{g_i \in \mathcal{D}} \varsigma(t, g_i) \tag{2}$$

where $\sigma(t, \mathcal{D})$ denotes the *frequency* or *support* of t in $\mathcal{D}$. The frequent *ftree* mining problem is to discover the *ftree* set FS of $\mathcal{D}$ which satisfies

$$FS = \{t \mid \sigma(t, \mathcal{D}) \geq \phi N\} \tag{3}$$

The problem of *closed frequent ftree mining* is to discover the set of frequent *ftrees*, denoted CFS, where $t \in CFS$ iff t is frequent and the support of t is strictly larger than that of any supertree of t. Formally, the closed frequent *ftree* mining problem is to discover the *ftree* set CFS of $\mathcal{D}$ which satisfies

$$CFS = \{t \mid t \in FS \wedge \forall t' \supset t, \sigma(t, \mathcal{D}) > \sigma(t', \mathcal{D})\} \tag{4}$$

Since CFS contains no *ftree* that has a supertree with the same support, we have $CFS \subseteq FS$.

3 Closed Frequent Ftree Mining: Proposed Solutions

Based on the definition in Eq.(4), a naive two-step algorithm of discovering CFS from $\mathcal{D}$ can be easily drafted. First, using current frequent *ftree* mining algorithms to discover FS from $\mathcal{D}$; Second, for each $t \in FS$, examining all $t' \in FS$ where $t \subset t'$ to tell whether t' satisfies $\sigma(t', \mathcal{D}) < \sigma(t, \mathcal{D})$. This algorithm is straightforward, but far from efficient. It indirectly discovers CFS by computing FS in the first place whose size is exponentially larger than that of CFS. The postprocessing operation of filtering non-closed frequent *ftrees* from FS also incurs unnecessary computation. We want an alternative method which directly computes CFS instead of computing FS in advance, i.e., under the traditional search space for mining frequent *ftrees*, efficient pruning algorithms should be proposed to detect branches that do not correspond to closed frequent *ftrees* as early as possible, and prune them to avoid unnecessary computation, which finally facilitate the total mining process.

In [12], we demonstrate `F3TM`, a fast frequent *ftree* mining algorithm, which outperforms up-to-date algorithms `FreeTreeMiner`[2,8] by an order of magnitude. In `F3TM`, an enumeration tree representing the search space of all frequent *ftrees* is built by a pattern-growth approach. Given a frequent *n-ftree* t, the potential frequent $(n+1)$-*ftree* t' originated from t is generated as

$$t' = t \circ_{ef} v, v \in \Sigma \tag{5}$$

where ef means pattern growth can be conducted on the *extension frontier* of t instead of each vertex of t, while at the same time ensuring the completeness of frequent *ftrees* discovered from the graph database . Figure 1 illustrates the extension frontier of a *ftree*, which is composed of vertices `3`, `4`, `5` and `6`, and the candidate generation of t, based on Eq. 5.

For each frequent *ftree* in the enumeration tree discovered by `F3TM`, we can check the closeness condition in Eq. 4. Given a frequent *n-ftree* t, its *immediate supertree set*, denoted $CS(t)$, which contains all $(n+1)$-*ftrees* $t' \supset t$ can be generated as

$$CS(t) = \{t' \mid t' = t \circ_x v, v \in \Sigma\} \tag{6}$$

where x means v can be grown on any vertex of t, which is shown in Figure 2. t's *immediate frequent supertree set*, denoted $FS(t)$, which contains all frequent $(n+1)$-*ftrees* $t' \supset t$ can be generated as

$$FS(t) = \{t' | t' \in CS(t) \wedge \sigma(t, \mathcal{D}) \geq \phi N\} \tag{7}$$

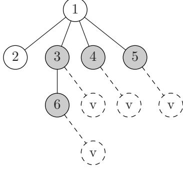


Fig. 1. $t' = t \circ_{ef} v$

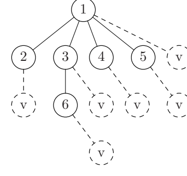


Fig. 2. $t' = t \circ_x v$

Given a frequent *ftree* $t' \in FS(t)$, we denote the vertex which is grown on t to get t' as $(t' - t)$, and the vertex of t at which $(t' - t)$ is grown on as p, i.e., the parent of $(t' - t)$ in t'.

The basic algorithmic framework for mining closed frequent *ftrees* can be formalized as follows: if for every $t' \in FS(t)$, $\sigma(t', \mathcal{D})$ is strictly smaller than $\sigma(t, \mathcal{D})$, then t is closed; Otherwise, t is non-closed, i.e., we can tell the closeness of t by checking the support values of all its immediate frequent supertrees in $FS(t)$ during the traversal of the enumeration tree for mining frequent *ftrees*.

4 Pruning the Search Space

In the previous section, we traverse the enumeration tree to discover all frequent *ftrees* in a graph database. However, the final goal of our algorithm is to find only closed frequent *ftrees*. Therefore, it is not necessary to grow the complete enumeration tree, because under certain conditions, some branches of the enumeration tree are guaranteed to produce no closed frequent *ftrees* and therefore can be pruned efficiently. In this section, we introduce algorithms that prune unwanted branches of the search space.

4.1 Equivalent Occurrence

Given a *ftree* t and a graph $g \in \mathcal{D}$, let $f(t, g)$ represents a subtree isomorphism from t to g. $f(t, g)$ is also referred to as an *occurrence* of t in g. Notice that t can occurs more than once in g. Let $\omega(t, g)$ denote the number of occurrences of t in g. The number of occurrences of t in a graph database $\mathcal{D}$ can be formally defined as

Definition 1. *Given a ftree t and a graph database $\mathcal{D} = \{g_1, g_2, \ldots, g_N\}$, the number of occurrence of t in $\mathcal{D}$ is the sum of the number of subtree isomorphisms of t in $g_i \in \mathcal{D}$, i.e., $\sum_{i=1}^{N} \omega(t, g_i)$, denoted by $\mathcal{O}(t, D)$.*

Suppose a *ftree* $t' = t \circ_x v$, f is a subtree isomorphism of t in g and f' is a subtree isomorphism of t' in g. If $\exists \rho$, ρ is subtree isomorphism of t in t', i.e., $\forall v, f(v) = f'(\rho(v))$, we call t and t' *simultaneously occur* in graph g. Intuitively, as we can derive t' from t by $t' = t \circ_x v$, we can get t' in the same pattern-growth way from t in g. We denote the number of such *simultaneous occurrences* of t' w.r.t t in g by $\omega(t, t', g)$. Similarly, the number of simultaneous occurrences of t' w.r.t t in $\mathcal{D}$ is defined as

Definition 2. *Given a ftree $t' = t \circ_x v$ and a graph database $\mathcal{D} = \{g_1, g_2, \ldots, g_N\}$, the number of simultaneous occurrence of t' w.r.t. t in $\mathcal{D}$ is the sum of the number*

of simultaneous occurrences of t' *w.r.t* t *in* $g_i \in \mathcal{D}$*, i.e.,* $\sum_{i=1}^{N} \omega(t, t', g_i)$*, denoted by* $\mathcal{SO}(t, t', D)$*.*

Definition 3. *Given* $t' = t \circ_x v$ *and a graph database* $\mathcal{D} = \{g_1, g_2, \ldots, g_N\}$*, if* $\mathcal{O}(t, D) = \mathcal{SO}(t, t', D)$*, we say that* t *and* t' *have* ***equivalent occurrences****.*

Lemma 1. *For a frequent ftree* t *in the enumeration tree, if there exists a* $t' \in FS(t)$ *such that (1)* t *and* t' *have equivalent occurrences; (2) the vertex* $(t' - t)$ *is not grown on the extension frontier of any descendants of* t*, including* t*, in the enumeration tree, then (1)* t *is not a closed frequent ftree and (2) for each child* t'' *of* t *in the enumeration tree, there exists at least one supertree* t''' *of* t''*, such that* t''' *and* t'' *have equivalent occurrences.*

Proof. The first statement can be easily proved. Since t and t' have equivalent occurrences in $\mathcal{D}$, then $\mathcal{O}(t', D) = \mathcal{O}(t, D)$. For the second statement, we notice that $(t' - t)$ occurs at each occurrence of t in $\mathcal{D}$, so it occurs at each occurrence of t'' in $\mathcal{D}$. In addition, the vertex $(t' - t)$ never be grown on the extension frontier of any descendant of t, so it will not be a vertex of t'' (Notice t'' is a child of t in the enumeration tree by growing a vertex on t's extension frontier). Therefore, we can obtain t''' by adding $(t' - t)$ on t'', so that t'' and t''' have equivalent occurrences.

By inductively applying Lemma 1 to t and all t's descendants in the enumeration tree, we can conclude that all branches originated from t in the enumeration tree are guaranteed to produce no closed frequent *ftrees*. However, the conditions mentioned in Lemma 1, especially the condition (2) is hard to be justified. Since when mining frequent *ftree* t, we have no information of all t's descendants in the enumeration tree. The following sections will present more detailed techniques to prune the search space.

4.2 The Safe Position Pruning

Given a *ftree* t and a vertex $v \in t$, the *depth* of v can be defined as follows

$$depth(v) = \begin{cases} 1 & \text{if } v \text{ is a leaf} \\ \min_{u \in t, u \text{ is child of } v}\{depth(u) + 1\} & \text{otherwise} \end{cases} \quad (8)$$

Intuitively, the depth of a vertex v is the minimum number of vertices from v to the nearest leaf of t. For a frequent *ftree* $t' \in FS(t)$ where t and t' have equivalent occurrences, the vertex $(t' - t)$ can be grown at different positions, i.e., there are the following possibilities for the position of p in t. (1)$depth(p) \leq 2$ and p is on the extension frontier of t; (2) $depth(p) \leq 2$ but p is not on the extension frontier; (3) $depth(p) > 2$.

If p occurs in position (1), vertex$(t' - t)$ is grown on the extension frontier of t. If p occurs in position (2), there are possibilities that for some descendant t'' of t in the enumeration tree, the vertex p can still be on the extension frontier of t''. A example is shown in Figure 3. In frequent *ftree* t, $depth(p) = 2$ and p is not located on the extension frontier. After the vertex a is grown on the extension frontier (vertex b), we get another frequent *ftree* t'' in which p is now located on

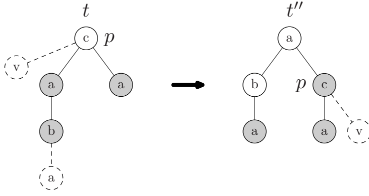


Fig. 3. A Special Case in Position (2)

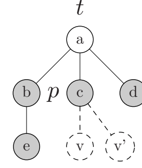


Fig. 4. The Safe Label Pruning

the extension frontier. So the first two possible positions of p are *unsafe* when growing vertex $(t' - t)$, which disallows the conditions mentioned in Lemma 1.

The following theorem shows that only position (3) of p is *safe* to grow the vertex $(t' - t)$, while not violating the conditions mentioned in Lemma 1.

Theorem 1. *For a frequent ftree $t' \in FS(t)$ such that t and t' have equivalent occurrences in $\mathcal{D}$. If $depth(p) > 2$, then neither t nor any t's descendants in the enumeration tree can be closed.*

Proof. Since for every vertex u on the extension frontier of a *ftree*, it is located at the bottom two levels, i.e., $depth(u) \leq 2$. If $depth(p) > 2$, the vertex p can never appear on the extension frontier of any *ftree*, i.e., the vertex $(t' - t)$ will not be grown on the extension frontier of any descendant of t, including t, in the enumeration tree. According to Lemma 1, the branches originated from t can not generate closed frequent *ftrees*.

The pruning algorithm mentioned in Theorem 1 is called the *safe position pruning*, since the vertex $(t' - t)$ can only be grown on a safe vertex $p \in t$, where $depth(p) > 2$. Given a *n-ftree*, the depth of every vertex of t can be computed in $O(n)$, so the safe position pruning is quite efficient to testify whether a certain branch in the enumeration tree should be pruned or not.

4.3 The Safe Label Pruning

If p is on the extension frontier of t, obviously, $depth(p) \leq 2$. We can not prune t from the enumeration tree. However, depending on the vertex label of $(t' - t)$, we can still possibly prune some children of t in the enumeration tree.

Theorem 2. *For a frequent ftree $t' \in FS(t)$ such that t and t' have equivalent occurrences in $\mathcal{D}$, if p is located on the extension frontier of t, we do not need to grow t by adding to p a new vertex with label lexicographically greater than $(t' - t)$.*

Proof. For any $t'' \in FS(t)$ such that p is the parent of$(t'' - t)$ and $(t'' - t)$ is lexicographically greater than $(t' - t)$, a *ftree* $t''' = t'' \circ_p (t' - t)$ have equivalent occurrence with t'' and $t''' \in FS(t'')$. Note $t'' \circ_p (t' - t)$ means growing vertex $(t' - t)$ on p of *ftree* t''. According to Lemma 1, t'' is not closed. And for every descendant of t'' in the enumeration tree, $(t' - t)$ never be grown on its extension frontier. Because during frequent *ftrees* mining, we generate candidates in a lexicographical order. Since $(t'' - t)$ is lexicographically greater than $(t' - t)$,

the vertex $(t' - t)$ will not be reconsidered to be grown on t'' and all t'''s descendants in the enumeration tree. According to Lemma 1, neither t'' nor any of its descendants can be closed.

The pruning algorithm mentioned in Theorem 2 is called the *safe label pruning*. The vertex label of $(t' - t)$ is *safe* because all vertices with labels lexicographically greater than $(t' - t)$ can be exempted from growing on p of t, and all descendants of corresponding *ftrees* in the enumeration tree are also pruned. An example is shown in Figure 4. p is located on the extension frontier of t and $v = (t' - t)$. If v''s label is lexicographically greater than v's label, the frequent *ftree* $t'' = t \circ_p v'$ and the frequent *ftree* $t''' = t'' \circ_p v$ have equivalent occurrences, so that t'' is not closed. Similarly, all t'''s descendants in the enumeration tree are not closed, either.

4.4 Efficient Computation of $FS(t)$

Based on the above analysis, both candidate generation and closeness test of the frequent *ftree*, t, need to compute $FS(t)$. Depending on if t can be pruned from the enumeration tree during closed frequent *ftree* mining, we can divide $FS(t)$ into the following mutually exclusive subsets:

$$
\begin{aligned}
EO(t) &= \{t' \in FS(t) \mid t' \text{ and } t \text{ have equivalent occurrences}\} \\
EN(t) &= \{t' \in FS(t) \mid \sigma(t, \mathcal{D}) = \sigma(t', \mathcal{D})\} \\
F(t) &= \{t' \in FS(t) \mid t' \text{ is frequent}\}
\end{aligned}
$$

Based on Theorem 1 and Theorem 2, the set $EO(t)$ can be further divided into the following mutually exclusive subsets:

$$
\begin{aligned}
EO_1(t) &= \{t' \in EO(t) \mid p \in t \text{ is safe}\} \\
EO_2(t) &= \{t' \in EO(t) \mid p \text{ is on the extension frontier of } t\} \\
EO_3(t) &= EO(t) - EO_1(t) - EO_2(t)
\end{aligned}
$$

When computing the sets mentioned above, we map t to each occurrence in $g_i \in \mathcal{D}$ and select the possible vertex $(t' - t)$ to grow. However, this procedure is far from efficient since a lot of redundant t' are generated. Now we study how to speed up the computation of $FS(t)$ based on the characteristics of *ftree*. The detailed analysis can be found in [12].

Automorphism-based Pruning: In the example shown in Figure 5, The leftmost *ftree* t is a frequent 7-*ftree*, where vertices are identified with a unique number as *vertex id*. When growing a new vertex v on vertex 3 of t, we get a 8-*ftree* $t' \in CS(t)$, shown in the middle of Figure 5. However, when growing v on vertex 5 of t, we get another 8-*ftree* $t'' \in CS(t)$, shown on the right of Figure 5. Notice $t' = t''$ in the sense of *ftree* isomorphism, so t'' can be pruned when computing $FS(t)$.

Based on the observation mentioned above, We propose an *automorphism-based pruning* algorithm to efficiently avoid redundant generation of *ftrees* in $FS(t)$. Given a *ftree*, all vertices can be partitioned into different equivalence classes based on *ftree* automorphism. Figure 6 shows how to partition vertices

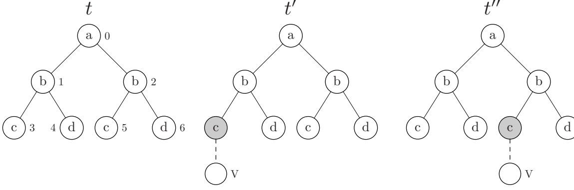


Fig. 5. $t', t'' \in CS(t)$ and $t' = t''$

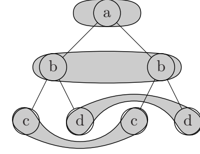


Fig. 6. Equivalence Class

of t in Figure 5 into four equivalence classes. When computing $FS(t)$, only one representative for each equivalence class of t is considered, instead of growing vertices on every position within an equivalence class.

Canonical Mapping-based Pruning: When computing $FS(t)$, we maintain *mappings* from t to all its occurrences in $g_i \in \mathcal{D}$. However, there exist redundant mappings because of *ftree* automorphism. Given a *n-ftree* t, and assume that the number of equivalence classes of t is c, and the number of vertices in each equivalence class C_i is n_i, for $1 \leq i \leq c$. The number of mappings from t to an occurrence in g_i is computed as $\omega(t, g_i) = \prod_{i=1}^{c}(n_i)!$. When either the number of equivalence classes, or the number of vertices in some equivalence class is large, $\omega(t, g_i)$ can be huge. However, among all mappings describing the same occurrence of $t \in g_i$, one out of $\prod_{i=1}^{c}(n_i)!$ mappings is selected as *canonical mapping* and all computation of $FS(t)$ is based on the canonical mapping of t in $\mathcal{D}$. While other $(\prod_{i=1}^{c}(n_i)! - 1)$ mappings can be pruned so that the computation of $FS(t)$ can be greatly facilitated.

5 The CFFTree Algorithm

In this section, we summarize our `CFFTree` algorithm, which is short for *Closed Frequent Ftree* Mining. Algorithm 1 illustrates the framework of `CFFTree`. The algorithm simply calls `CF-Mine` which recursively mines closed frequent *ftrees* of a graph database by a depth-first traversal on the enumeration tree.

Algorithm 2 outlines the pseudo-code of `CF-Mine`. For each frequent *ftree* t, `CFFTree` check all candidate frequent *ftree* $t' = t \circ_x v$, to obtain $\mathcal{SO}(t, t', \mathcal{D})$, which is useful to compute $EO(t)$ (Line 1) and $EN(t)$ (Line 2). However, for $t' \in F(t)$, `CFFTree` only grows t on its extension-frontier, i.e. $t' = t \circ_{ef} v$, which ensures the completeness of frequent *ftrees* in $\mathcal{D}$ (Line 7-12). Automorphism-based pruning and canonical mapping-based pruning can be applied to facilitate the computation of the three sets $EO(t)$, $EN(t)$ and $F(t)$. For the frequent *ftree* t, if there exists $t' \in EO_1(t)$, then neither t nor any of t's descendants in the enumeration tree can be closed, and hence can be efficiently pruned (Line 3-4). If $EO_1(t) = \emptyset$ but there exists $t' \in EO_2(t)$, although we cannot prune t from the enumeration tree, we can apply Theorem 2 to prune some children of t in the enumeration tree (Line 11-12). If $EO(t) = \emptyset$, then no pruning is possible and we have to compute $EN(t)$ to determine the closeness of t, i.e., the naive algorithm mentioned in Section 3 (Line 2). If $EN(t) \neq \emptyset$, t is not closed, otherwise, t

Algorithm 1. *CFFTree* $(\mathcal{D}, \phi)$

Input: A graph database $\mathcal{D}$, the minimum support threshold ϕ
Output: The closed frequent *ftrees* set $\mathcal{CF}$
1: $\mathcal{CF} \leftarrow \emptyset$;
2: $\mathcal{F} \leftarrow$ frequent 1-*ftrees*;
3: **for all** frequent 1-*ftree* $t \in \mathcal{F}$ **do**
4: *CF-Mine*$(t, \mathcal{CF}, \mathcal{D}, \phi)$;
5: **return** $\mathcal{CF}$

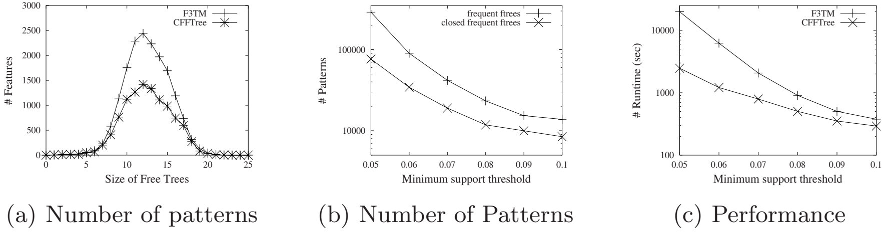
Algorithm 2. *CF-Mine* $(t, \mathcal{CF}, \mathcal{D}, \phi)$

Input: A frequent *ftree* t, the set of closed frequent *ftrees*, $\mathcal{CF}$, A graph database $\mathcal{D}$ and the minimum support threshold ϕ
Output: The closed frequent *ftrees* set $\mathcal{CF}$
1: Compute $EO(t)$;
2: **if** $EO(t) = \emptyset$ **then** Compute $EN(t)$;
3: **if** $\exists t' \in EO_1(t)$ **then**
4: return; // The safe position pruning;
5: **else**
6: $F(t) \leftarrow \emptyset$
7: **for each** equivalence class ec_i on the extension frontier of t **do**
8: **for each** valid vertex v which can be grown on ec_i of t **do**
9: $t' \leftarrow t \circ_{ef} v$, where p, a representative of ec_i, is v's parent
10: **if** support$(t') \geq \phi|\mathcal{D}|$ **then**
11: **if** $\nexists t'' \in EO_2(t)$, where $(t'' - t)$ is p and the label of $(t' - t)$ is lexicographically greater than that of $(t'' - t)$ **then**
12: $F(t) \leftarrow F(t) \cup \{t'\}$ // the safe label pruning
13: **for each** frequent t' in $F(t)$ **do**
14: *CF-Mine*$(t', \mathcal{CF}, \mathcal{D}, \phi)$
15: **if** $EO(t) = \emptyset$ and $EN(t) = \emptyset$ **then**
16: $\mathcal{CF} \leftarrow \mathcal{CF} \cup \{ t\}$

is closed (Line 15-16). The set $F(t)$ is computed by extending vertices on the extension frontier of t, which grows the enumeration tree for frequent *ftree* mining (Line 8-12). This procedure proceeds recursively (Line 13-14) until we find all closed frequent *ftrees* in the graph database.

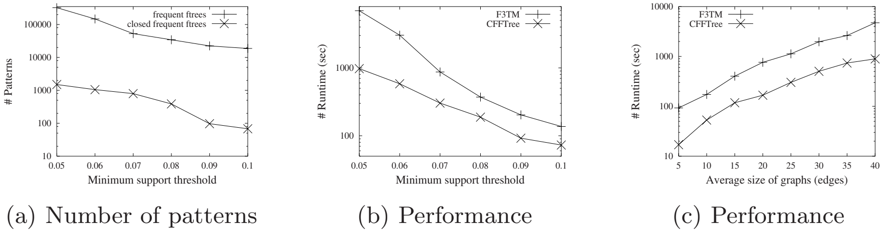
6 Experiments

In this section, we report a systematic performance study that validates the effectiveness and efficiency of our closed frequent free tree mining algorithm: `CFFTree`. We use both a real dataset and a synthetic dataset in our experiments. All experiments were done on a 3.4GHz Intel Pentium IV PC with 2GB main memory, running MS Windows XP operating system. All algorithms are implemented in C++ using the MS Visual Studio compiler. We compare `CFFTree` with `F3TM` plus post-processing, thus, the performance curve mainly reflects the effectiveness of pruning techniques mentioned in Section 4.



(a) Number of patterns (b) Number of Patterns (c) Performance

Fig. 7. Mining patterns in real datasets



(a) Number of patterns (b) Performance (c) Performance

Fig. 8. Mining patterns in synthetic datasets

The real dataset we tested is an AIDS antiviral screen chemical compound database from Developmental Theroapeutics Program in NCI/NIH. The database contains up to 43,905 chemical compounds. There are total 63 kinds of atoms in this database, most of which are C, H, O, S, etc. Three kinds of bonds are popular in these compounds: single-bond, double-bond and aromatic-bond. We take atom types as vertex labels and bond types as edge labels. On average, compounds in the database has 43 vertices and 45 edges. The graph of maximum size has 221 vertices and 234 edges.

Figure 7(a) shows the number of frequent patterns w.r.t. the size of patterns (vertex number). We select 10000 chemical compounds from the real database and set the minimum threshold ϕ to be 10%. As shown, most frequent and closed frequent *ftrees* have vertices ranging from 8 to 17. While the number of small *ftrees* with vertex number less than 5 and large *ftrees* with vertex number greater than 20 is quite limited. Figure 7(b) shows the number of patterns of interest with ϕ varying from 5% to 10% and the running time is shown in Figure 7(c) on the same dataset. As we can see, `CFFTree` outperforms `F3TM` by a factor of 10 in average and the ratio between frequent *ftrees* and closed ones is close from 10 to 1.5. It demonstrates that closed pattern ming can deliver more compact mining results.

We then tested `CFFTree` on a series of synthetic graph databases, which are generated by the widely-used graph generator [5]. The synthetic dataset is characterized by different parameters, which is described in detail in [5]. Figure 8(a) shows the number of patterns of interest with ϕ varying from 5% to 10% and the running time is shown in Figure 8(b) for the dataset $\mathcal{D}10000I10T30V50$. Compared with the real dataset, `CFFTree` has a similar performance gain in this synthetic dataset. We then test the mining performance by changing the

parameter T in the synthetic data, while other parameters keep fixed. The experimental results are shown in Figure 8(c). Again, `CFFTree` performs better than `F3TM`.

7 Conclusion

In this paper, we investigate the problem of *mining closed frequent ftrees from large graph databases*, a critical problem in structural pattern mining because mining all frequent *ftrees* are inherently inefficient and redundant. Several new pruning algorithms are introduced in this study including the *safe position pruning* and the *safe label pruning* to efficiently prune branches of the search space. The *automorphism-based pruning* and the *canonical mapping-based pruning* are applied in the computation of candidate sets and equivalent occurrence sets, which dramatically facilitate the total mining process. A `CFFTree` algorithm is implemented and our performance study demonstrates its high efficiency over the up-to-date frequent *ftree* mining algorithms. To our best knowledge, this is the first piece of work on closed frequent *ftree* mining on large graph databases.

Acknowledgment. This work was supported by a grant of RGC, Hong Kong SAR, China (No. 418206).

References

1. Yun Chi, Yi Xia, Yirong Yang, and Richard R. Muntz. Mining closed and maximal frequent subtrees from databases of labeled rooted trees. *IEEE Transactions on Knowledge and Data Engineering*, 17(2):190–202, 2005.
2. Yun Chi, Yirong Yang, and Richard R. Muntz. Indexing and mining free trees. In *Proceedings of ICDM03*, 2003.
3. Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* 1979.
4. Jun Huan, Wei Wang, and Jan Prins. Efficient mining of frequent subgraphs in the presence of isomorphism. In *Proceedings of ICDM03*, 2003.
5. Michihiro Kuramochi and George Karypis. Frequent subgraph discovery. In *Proceedings of ICDM01*, 2001.
6. Siegfried Nijssen and Joost N. Kok. A quickstart in frequent structure mining can make a difference. In *Proceedings of KDD04*, 2004.
7. Nicolas Pasquier, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. Discovering frequent closed itemsets for association rules. In *Proceeding of ICDT99*, 1999.
8. Ulrich Rückert and Stefan Kramer. Frequent free tree discovery in graph data. In *Proceedings of SAC04*, 2004.
9. Xifeng Yan and Jiawei Han. gspan: Graph-based substructure pattern mining. In *Proceedings of ICDM02*, 2002.
10. Xifeng Yan and Jiawei Han. Closegraph: mining closed frequent graph patterns. In *Proceedings of KDD03*, 2003.
11. Xifeng Yan, Jiawei Han, and Ramin Afshar. Clospan: Mining closed sequential patterns in large databases. In *Proceedings of SDM03*, 2003.
12. Peixiang Zhao and Jeffrey Xu Yu. Fast frequent free tree mining in graph databases. In *Proceedings of MCD06 - ICDM 2006 Workshop*, Hong Kong, China, 2006.