SPIN: Mining Maximal Frequent Subgraphs from Graph Databases

Jun Huan, Wei Wang, Jan Prins
Department of Computer Science,
University of North Carolina at Chapel Hill
Chapel Hill,NC 27599, USA
{huan, weiwang, prins}@cs.unc.edu

Jiong Yang
Department of Computer Science,
University of Illinois
Urbana-Champaign, IL 61801, USA
jioyang@cs.uiuc.edu

ABSTRACT

One fundamental challenge for mining recurring subgraphs from semi-structured data sets is the overwhelming abundance of such patterns. In large graph databases, the total number of frequent subgraphs can become too large to allow a full enumeration using reasonable computational resources. In this paper, we propose a new algorithm that mines only *maximal* frequent subgraphs, i.e. subgraphs that are not a part of any other frequent subgraphs. This may exponentially decrease the size of the output set in the best case; in our experiments on practical data sets, mining maximal frequent subgraphs reduces the total number of mined patterns by two to three orders of magnitude.

Our method first mines all frequent trees from a general graph database and then reconstructs all maximal subgraphs from the mined trees. Using two chemical structure benchmarks and a set of synthetic graph data sets, we demonstrate that, in addition to decreasing the output size, our algorithm can achieve a five-fold speed up over the current state-of-the-art subgraph mining algorithms.

Categories and Subject Descriptors: H.2.8 [Database Applications]: Data Mining

General Terms: Algorithms

Keywords: Subgraph Mining, Spanning Tree

1. INTRODUCTION

In this paper, we focus on the problem of finding recurring subgraphs from graph databases, which is a very active topic in current data mining research. Graphs provide a general way to model a variety of relations among data, hence finding recurring subgraphs has many applications in interdisciplinary research such as chemical informatics [2] and bioinformatics [12]. There are also many applications in data management research such as efficient storage of semi-structured databases [5], efficient indexing [6], and web information management [23, 17].

One performance issue (among many others) in mining large graph databases is the huge number of recurring patterns. The phenomenon is well understood in mining "long" frequent itemsets. Given a frequent itemset I, any subset of I is also frequent hence the number of such frequent itemsets grows exponentially with $|I|$.

In this paper, we propose a new graph mining algorithm that mines only maximal frequent subgraphs. Given a set of graphs $\mathcal{G}$ (referred to as a *graph database*), the *support* of a graph G is defined as the fraction of graphs in $\mathcal{G}$ in which G occurs [10, 21]. G is *frequent* if its support is at least a user specified threshold; a frequent subgraph is *maximal* if none of its super graphs are frequent [11]. Mining only maximal frequent subgraph offers the following advantages in processing large graph databases. (1) It significantly reduces the total number of mined subgraphs. In experiments we performed on some realistic data sets, the total number of frequent subgraphs is up to one thousand times greater than the number of maximal subgraphs. We can save both space and subsequent analysis effort if the number of mined subgraphs is significantly reduced. (2) Several "pruning" techniques, which are detailed in this paper, can be efficiently integrated into the mining process and dramatically reduce the total mining time. (3) The non-maximal frequent subgraphs can be reconstructed from the maximal subgraphs reported. To get the actual frequency (support) of non-maximal subgraphs requires examination of the original database, but it is certain to be at least as high as the frequency of the maximal subgraph. In addition, the techniques used in [16] can be easily adapted to approximate the support of all frequent subgraphs within some error bound. (4) In some applications such as discovering structure motifs in a group of homology proteins [8, 12], maximal frequent subgraphs are the subgraphs of most interest since they encode the maximal structure commonalities within the group.

Our mining method is based on a novel graph mining framework in which we first mine all frequent tree patterns from a graph database and then construct maximal frequent subgraphs from trees. This approach offers asymptotic advantages compared to using subgraphs as building blocks, since tree normalization is a simpler problem than graph normalization. The proposed method enables us to integrate well-developed techniques from mining maximal itemsets and knowledge gained in graph mining into a new algorithm. According to our experimental study, such a combination can offer significant performance speedup in both synthetic and real data sets. The framework of our method is versatile. Depending on the particular tree mining algorithm, the search can be either breadth-first or depth-first (preferred due to its better memory utilization). It can also be designed to mine all frequent subgraphs without major modifications.

Technically, we make three contributions: (1) we propose a novel algorithm SPIN (**SP**anning tree based maximal graph m**IN**ing) to mine only maximal frequent subgraphs of large graph databases,

KDD'04, August 22–25, 2004, Seattle, Washington, USA.

(2) we integrate several optimization techniques, some from existing maximal itemset mining research and some developed by ourselves, to speed up the mining process, (3) we perform an extensive analysis of the proposed algorithm and analyze how its performance on graph data sets with different characteristics.

The remainder of the paper is organized as follows. In Section 2, we present the data structure and the proposed algorithm. Section 3 presents the results of our experimental study using synthetic graph databases and two benchmark chemical data sets. We conclude the paper with a discussion, related works, and conclusion.

2. MAXIMAL SUBGRAPH MINING

In the following discussion, we present a novel framework for mining maximal frequent subgraphs from a graph database. The framework combines tree mining and subgraph mining; we first find all frequent trees from a graph database and then reconstruct the group of frequent subgraphs from the mined trees. There are two important components in the framework. The first is a graph partitioning method through which we group all frequent subgraphs into equivalence classes based on the spanning trees they contain. The second important component is a set of pruning techniques which aim to remove some partitions entirely or partially for the purpose of finding maximal frequent ones only.

There are three reasons we advocate this two-step method for finding maximal graph patterns. First, tree related operations such as isomorphism, normalization, and testing whether a tree is a subtree of another tree are asymptotically simpler than the comparable operations for graphs, which are NP-complete. Second, in certain applications, such as chemical compound analysis, most of the frequent subgraphs are really trees. Last but not least, this framework adapts well to *maximal* frequent subgraph mining, which is the focus of this paper. Using a chemical structure benchmark, we show 99% of cyclic graph patterns and 60% of tree patterns can be eliminated by our optimization technique in searching for maximal subgraphs; further details about the efficiency of the optimization techniques can be found in [11]. To the best of our knowledge, we are the first to combine the two distinct methodologies: mining frequent subgraphs in graph databases and mining frequent trees in forests (a set of trees) for the purpose of designing efficient subgraph mining algorithm.

2.1 Tree-based Equivalence Classes

We define a *subtree* of an undirected graph G as an acyclic connected subgraph of G. A subtree T is a *spanning tree* of G if T contains all nodes in G. Given a graph G, there are many spanning trees and we define the maximal one, according to a total order defined on trees [4, 11], and call it the *canonical spanning tree* of G.

EXAMPLE 2.1. *In Figure 1, we show an example of a labeled graph P (upper-left) with all four-node subtrees of P. Each subtree is represented by its canonical representation and sorted according to the total order $\succeq$ (as given in [11]). Each such tree is a spanning tree of the graph P and the first one (T_1) is the canonical spanning tree of P.*

DEFINITION 2.1. ***Tree-based Equivalence Classes:*** *Given two graphs P and Q, we defined a binary relation $\cong$ such that $P \cong Q$ if and only if their canonical spanning trees are isomorphic to each other. The relation $\cong$ is reflexive, symmetric, transitive, and hence an equivalence relation.*

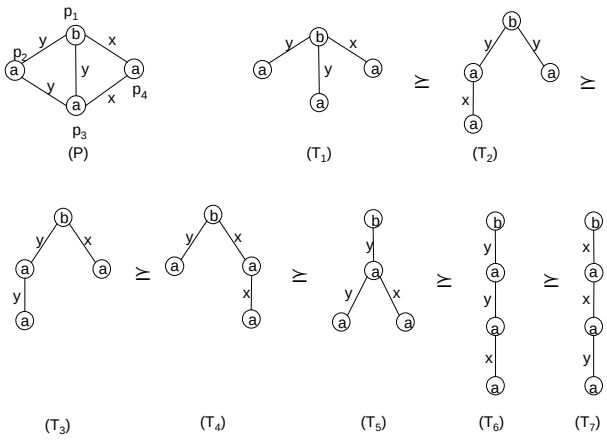

Figure 1: Example of a labeled graph P (upper-left), P's subtrees, spanning trees, and its canonical spanning tree (T_1).

EXAMPLE 2.2. *In Figure 2, we show subgraphs of the graph P in Figure 1 which are not necessarily trees. Subgraphs are grouped together if they share the same canonical spanning tree. The five non-singleton groups are shown here and the remaining twelve groups are all singletons* [1]

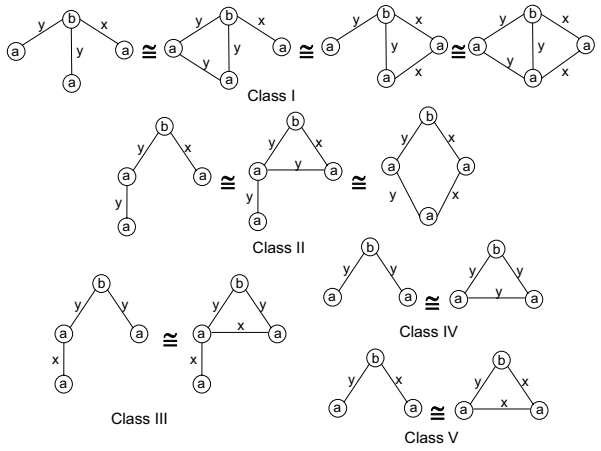

Figure 2: Example of tree based equivalence classes for subgraphs in graph P, presented in Figure 1.

We can use a simple greedy search algorithm to find the canonical spanning tree of a graph G, the details of which are given in [11]. The frequent subgraph mining can conceptually be broken into two steps: (1) mine all the frequent trees from a graph database and (2) for each such frequent tree T, find all frequent subgraphs whose canonical spanning trees are isomorphic to T. Maximal frequent subgraphs can be found among frequent ones. We skip the first step in the following discussion for two reasons. First, as pointed out in [21], the current subgraph mining algorithms can be easily tailored to find only trees from a graph database by limiting the topology of the patterns. This is true for Closegraph [21] as well as for FFSM [10], which is our recently developed depth-first subgraph mining algorithm. Second, most of the techniques developed for mining subtrees from a forest can also be easily adapted

[1] Throughout the paper, we are interested only in subgraphs with at least an edge (i.e. excluding frequent nodes as trivial cases).

for the same purpose. Therefore, in the following discussion, we focus on step 2, which is how to enumerate the equivalence class of a tree T and how maximal subgraph mining is related to this enumeration. We want to point out that the two-step division of the mining algorithm is artificial but it makes it easy to explain the key ideas of the algorithm without introducing too many details. In our longer version of this paper [11], we discuss a fully optimized algorithm which (1) uses a modified FFSM algorithm to enumerate trees from graph databases and (2) integrates tree discovery and maximal pattern mining for maximal performance.

2.2 Enumerating Graphs from Trees

We first outline a basic enumeration scheme to search the equivalence class of a tree. We define a *joining* operation $\oplus$ between a graph(tree) G and a hypothetical edge connecting any two nodes i, j in G with label e_l such that $G \oplus (i, j, e_l) = G'$ where G' is a supergraph of G with one additional edge between nodes i and j with label e_l. If the graph G already contains an edge between nodes i and j, the joining operation fails and produces nothing. If G' is frequent, we denote the hypothetical edge (i, j, e_l) as a *candidate edge* for G. The above definition can serve as the basis for a recursive definition of the joining operation between a graph G and a candidate edge set $E = \{e_1, e_2, \ldots, e_n\}$ such that $G \oplus E = (G \oplus e_1) \oplus \{e_2, \ldots, e_n\}$.

Let's assume we already calculated the set of candidate edges $C = \{c_1, c_2, \ldots, c_n\}$ from the set of all possible frequent hypothetical edges. We define the *search space* of G, denoted by $G : C$, as the set of graphs which might be produced by joining the graph G and a candidate edge set in the powerset set of C (denoted by 2^C). That is:

$$G : C = \{G \oplus y | y \in 2^C\} \quad (1)$$

In the following discussion, the group of candidate edges are sometimes referred to as the "tail" of the graph G in its search space. We present a recursive algorithm in Table 2 to enumerate the search space for a graph G. The procedure we use to calculate the set of candidate edges for a tree pattern can be found in [11].

EXAMPLE 2.3. *in Figure 3, we single out the largest equivalence class (Class One) from Figure 2. We show a tree K together with its tail $C = \{(k_2, k_3, y), (k_3, k_4, x)\}$. K's search space is hence composed of four graphs $\{K, K_{S1}, K_{S2}, K_{S3}\}$ (K is always included in its search space) and is organized into a "search tree" in analogy to frequent item set mining. This tree structure follows the recursive procedure we present in Table 2 which can be used to explore the search space for a given graph.*

Before we proceed to details about mining maximal frequent subgraphs, we outline the enumeration scheme discussed so far in Table 1 and Table 2. Our strategy is quite straightforward: we first find all frequent trees; trees are expanded to cyclic graphs by searching their search spaces; and maximal frequent subgraphs are constructed from frequent ones. We notice that this algorithm is *correct*, which means we are guaranteed to find all maximal frequent subgraphs. However, it is not *efficient* in that we still need to enumerate all frequent subgraphs to construct maximal ones. In the next section, we introduce optimization techniques to improve the search for maximal frequent subgraphs.

2.3 Optimizations: Global and Local Maximal Subgraphs

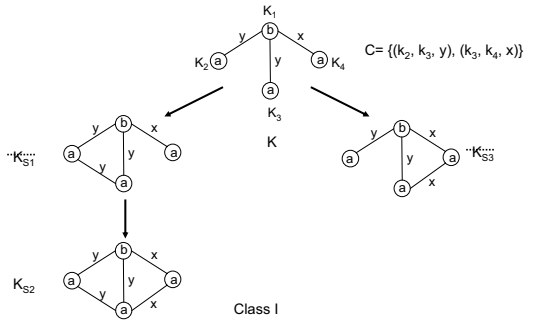

Figure 3: Example of enumerating graph's search space. We use dashed lines on the subgraph K_{S1} and K_{S3} to denote the fact that they will be pruned by an optimization technique which is discussed in Section 2.3.1

Algorithm Maximal Subgraph Mining($\mathcal{G}, \sigma$)
begin
1. $\mathcal{R} \leftarrow \{T \mid T$ is a frequent tree in $\mathcal{G}\}$
2. $S \leftarrow \{G \mid G \in$ Expansion(T) and $T \in \mathcal{R}\}$
3. return $\{G \mid G \in S$ and G is maximal $\}$
end

Table 1: An outline of the maximal subgraph mining algorithm

Algorithm Expansion(T)
begin
1. $C \leftarrow \{c \mid c$ is a candidate edge for $T\}$
2. $S \leftarrow$ Search Graphs (T, C)
3. return $\{G \mid G \in S$, G is frequent, and G has the same canonical spanning tree as T has$\}$
end
Algorithm Search Graphs($G, C = \{c_1, c_2, \ldots, c_n\}$)
begin
1. $Q \leftarrow \emptyset$
2. **for** each $c_i \in C$
3. $\quad Q \leftarrow Q \cup$ Search Graphs($G \oplus c_i$, $\{c_{i+1}, c_{i+2} \ldots, c_n\}$)
4. **endfor**
5. return Q
end

Table 2: An algorithm for exploring the equivalence class of a tree T

In this section, we explore several techniques for fast maximal frequent subgraph mining. These techniques ("pruning" techniques) dynamically remove a set of frequent subgraphs that can not be maximal from a search space. To that end, we define a frequent subgraph G to be *locally maximal* if it is maximal in its equivalence class i.e. G has no frequent supergraph(s) that share the same canonical spanning tree as G; we refer to a subgraph as *globally maximal* if it is maximal frequent in a graph database. Clearly, every global maximal subgraph must be locally maximal but not every local maximal subgraph is necessarily globally maximal. Our pruning techniques aim to avoid enumerating subgraphs which are not locally maximal.

Not surprisingly, the problem of finding all locally maximal frequent subgraphs can be transformed to the well-known maximal frequent itemset mining problem. Each candidate edge is an item; the joining operation can be viewed as the union operation for itemsets; and each local maximal subgraph corresponds to a maximal frequent itemset in its search space. Hence, we advocate the following pruning techniques, which are partially adapted from the maximal itemset mining and partially developed in the graph min-

ing context, for maximal frequent subgraph mining.

2.3.1 Bottom-Up Pruning

The search space of a graph G is exponential in the cardinality of the candidate edges set C. One heuristic to avoid such an exponential search space is to check whether the largest possible candidate $G' = G \oplus C$ is frequent or not. If G' is frequent, each graph in the search space is a subgraph of G' and hence not maximal. This heuristics is referred to as the *Bottom-Up Pruning* and can be applied to every step in the recursive search procedure presented in Table 2. By applying bottom-up pruning to the equivalence class I presented in Figure 2, graph K_{S1} and K_{S3} are pruned.

Dynamic Reordering: An important technique related to the efficiency of the bottom-up pruning is the so-called *dynamic reordering* technique, which works in two ways. First, it trims infrequent candidate edges from the tail of a graph to reduce the size of the search space (an edge candidate can become infrequent after several iterations since other edges are incorporated into the patterns). Second, it rearranges the order of the elements in the tail according to their support value. For example, given a graph's tail C, by dynamic reordering, we sort the elements in C by their support values, from lowest to highest. After this sorting, the infrequent "heads" are trimmed. At the end of the remaining tail is a family of elements individually having high support and hence the pattern obtained by grouping them together is likely to still have high support value. This heuristics is widely used in mining maximal itemsets to gain performance. However, without the spanning tree framework, applying dynamic ordering is very difficult in any of the current subgraph mining algorithms, which intrinsically have a fixed order of adding edges to an existing pattern for various performance considerations.

2.3.2 Tail Shrink

Given a graph G and a supergraph G' of G, an *embedding* of G in G' is a subgraph isomorphism f from G to G'. We prefer the term embedding to subgraph isomorphism, though they are interchangeable, for the purpose of intuitive descriptions. In Figure 4, we show a subgraph L and its supergraph P. There are two embeddings of L in P: $(l_1 \rightarrow p_1, l_2 \rightarrow p_2, l_3 \rightarrow p_3, l_4 \rightarrow p_4)$ and $(l_1 \rightarrow p_1, l_2 \rightarrow p_3, l_3 \rightarrow p_2, l_4 \rightarrow p_4)$. We define a candidate edge (i, j, e_l) to be *associative* to a graph G if it appears in every embedding of G in a graph database. In other words, a candidate edge (i, j, e_l) of G is associative if and only if for every embedding f of G in a graph G', G' has the edge $(f(i), f(j))$ with label e_l. One example of associative edge is edge (l_1, l_3, y) to the tree L shown in Figure 4.

If a tree T contains a set of associative edges $\{e_1, e_2, \ldots, e_n\}$, any maximal frequent graph G which is a supergraph of T must contain all such edges. Hence we can remove these edges from the tail of T and augment them to T without missing any maximal ones. This technique is referred to as the *tail shrink* technique. Tail shrink has three advantages: (1) it reduces the search space and (2) it can be used to prune the entire equivalence class in certain cases. To elaborate the latter point, we define a set of associative edges C of a tree T to be *lethal* if the resulting graph $G' = T \oplus C$ has a canonical spanning tree other than that of T. For example, in Figure 4, associative edge $e = (1, 3, y)$ of L is lethal since $G' = L \oplus e$ has a different canonical spanning tree than that of L. In the same example, the lethal edge e can be augmented to each member of the class II to produce a supergraph with the same support. Therefore the whole class can be pruned away once we detect a lethal edge(s) to the tree L. Detecting a group of lethal edges can do further pruning other than trimming off the whole equivalence class. Those details as well as the formal proof of the optimization are discussed in [11].

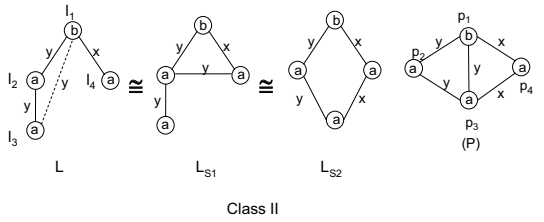

Figure 4: An example showing how tail shrink might be used to prune the whole equivalence class. Edge $e = (l_1, l_3, y)$, denoted by a dashed line to be distinguished from other edges, is associative to tree L and lethal to L as well. The graph obtained by joining L and e should belong to equivalence class I shown in Figure 2.

2.3.3 External-Edge Pruning

In this section, we introduce a technique to remove one equivalence class without any knowledge about its candidate edges. We refer to this technique as the *external-edge pruning*. We define an edge to be an *external edge* for a graph G if it connects a node in G and a node which is not in G. We represent an external edge as a three-element tuple (i, e_l, v_l) to stand for the fact that we introduce an edge with label e_l incident on the node i in a graph G and a node which does not belong to G with node label v_l. An external edge (i, e_l, v_l) is *associative* to a graph G if and only if:

- for every embedding f of G in a graph G', G' has a node v with the label v_l,
- v connects to the node $f(i)$ with an edge label e_l in G', and
- $\nexists$ node $j \in V[G]$ such that $v = f(j)$.

.

EXAMPLE 2.4. *We show two examples of associative external edges in Figure 5. One is (m_1, x, a) for the tree M and another one is $(n_f 1, y, a)$ for the tree N. If a tree T has at least one associative external edge, the entire equivalence class of T can be pruned since the same edge can be augmented to every member of the class. In this example, both equivalence classes IV and V can be eliminated due to the external-edge pruning.*

Once we find a tree T has an associative external edge, the same edge can be augmented to each members in T's equivalence class and therefore none of them are maximal.

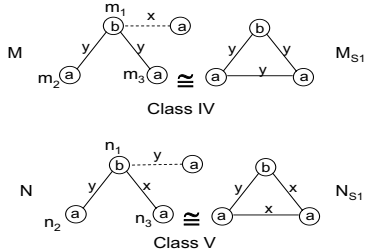

Figure 5: Examples showing external edges and associative external edges.

In a brief summary, we present three pruning techniques to speed up maximal subgraph mining. For the graph P shown in Figure 1, there are a total of twenty five subgraphs of P, including itself and excluding the null graph. These subgraphs are partitioned into five non-singleton classes, shown in Figure 2, and twelve singleton classes (not shown). There is only one maximal subgraph, namely, graph P itself. We have successfully pruned every one of the five non-singleton equivalence classes (P of the equivalence class I is left untouched since it is maximal). What we do not show further is that we can apply the same techniques to the remaining twelve singleton equivalence classes to eliminate all of them. Interested readers might verify that themselves.

Table 3 and Table 4 integrate these optimizations into the basic enumerate technique we presented in Table 1 and Table 2.

Algorithm MaxSubgraph-Expansion(T)
begin
1. $C \leftarrow \{c|\ c$ is a candidate edge for $G\}$
2. $A \leftarrow \{c|\ c \in C$ and c is associative $\}$
3. **if** A is lethal return $\emptyset$ #tail shrinking
2. $S \leftarrow$ Search Graphs ($T \oplus A$, $C - A$)
3. return $\{G|\ G \in S$, G is frequent, and G has the same canonical spanning tree as T has$\}$
end

Algorithm Search Graphs($G, C = \{c_1, c_2, \ldots, c_n\}$)
begin
1. **if** $G \oplus C$ is frequent, return $G \oplus C$ #bottom-up pruning
2. $Q \leftarrow \emptyset$
3. **for** each $c_i \in C$
4. $\quad Q \leftarrow Q \cup$ Search Graphs($G \oplus c_i$, $\{c_{i+1}, c_{i+2} \ldots, c_n\}$)
5. **endfor**
6. return Q
end

Table 3: An algorithm for exploring the equivalence class of a tree T for maximal subgraph mining

Algorithm Maximal Subgraph Mining($\mathcal{G}, \sigma$)
begin
1. $\mathcal{R} \leftarrow \{T|T$ is a frequent tree in $\mathcal{G}\}$
2. $S \leftarrow \{G|G \in$ Expansion(T), T has no external associative edge, and $T\ \in \mathcal{R}\}$ #external-edge prunning
3. return $\{G|G \in S$ and G is maximal $\}$
end

Table 4: An outline of the maximal subgraph mining algorithm

Due to the space limitation, several important details are omitted which include: (1) how to enumerate frequent trees from a graph database using a modified FFSM algorithm, (2) how to interleave the tree mining algorithm and the maximal subgraph mining algorithm and deliver the final optimized algorithm, (3) how we guarantee that each reported pattern is (a) frequent, (b) maximal, and (c) unique, (4) how to calculate the edge candidates for a tree, and (5) how to determine associative external edges. Those can be found in [11].

3. EXPERIMENTAL STUDY

We performed our empirical study using a single processor of a 2.8GHz Pentium Xeon with 512KB L2 cache and 2GB main memory, running RedHat Linux 7.3. The SPIN algorithm is implemented using the C++ programming language and compiled using g++ with O3 optimization. We compared SPIN with two alternative subgraph mining algorithms: FFSM ([10]) and gSpan [20]. Every maximal subgraph reported by SPIN in synthetical and real data sets are cross validated using results from FFSM and gSpan to make sure it is (a) frequent, (b) maximal, and (c) unique.

3.1 Synthetic Dataset

To evaluate the performance of the SPIN algorithm, we first generate a set of synthetic graph databases using a synthetic data generator [14].

In Figure 6, we represent the performance comparison of SPIN, FFSM, and gSpan algorithms for a synthetic data set with different support values. When the support is set to a pretty high value e.g. 5%, the performance of all three algorithms are pretty close. SPIN scales much better than the other two algorithms as we decrease the support values. At support value 1%, SPIN provides a six and ten fold speed-up over FFSM and gSpan, respectively. We do not show data with support value great than 5% since there is little difference among the three methods.

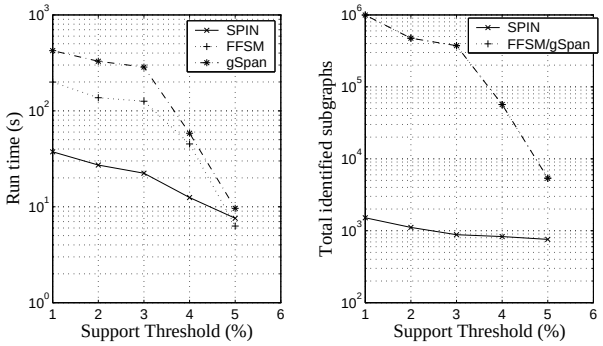

Figure 6: Left: performance comparison under different support values for data set $D10kT30L200I11V4E4$ using SPIN, FFSM and gSpan. Here we follow the common convention of encoding the parameters of a synthetic graph database as a string. Right: Total frequent patterns identified by the algorithms.

3.2 Chemical Data Set

We also applied SPIN to two widely used chemical data sets to test its performance. The data sets are obtained from the DTP AIDS Antiviral Screen test, conducted by U.S. National Cancer Institute. In the DTP data set, chemicals are classified into three sets: confirmed active (**CA**), confirmed moderately active (**CM**) and confirmed inactive (**CI**) according to experimentally determined activities against the HIV virus. The DTP data can be downloaded from http://dtp.nci.nih.gov/docs/aids/aids_data.html.

In Figure 7, we show the performance comparison of SPIN, FFSM, and gSpan using the DTP CA data set. We report that SPIN is able to expedite the program up to five(eight) fold, comparing with FFSM(gSpan) at support value 3.3%. Mining only maximal subgraphs can reduce the total number of mined patterns by a factor up to three orders of magnitude in this data set. We also applied the same algorithms to the data set DTP CM. In this case, SPIN has a performance very close to FFSM and both are around eight fold speed-up over gSpan. However, if we impose an additional constraint to let FFSM output the maximal patterns it finds among the set of frequent patterns, SPIN offers a three fold speed-up from FFSM.

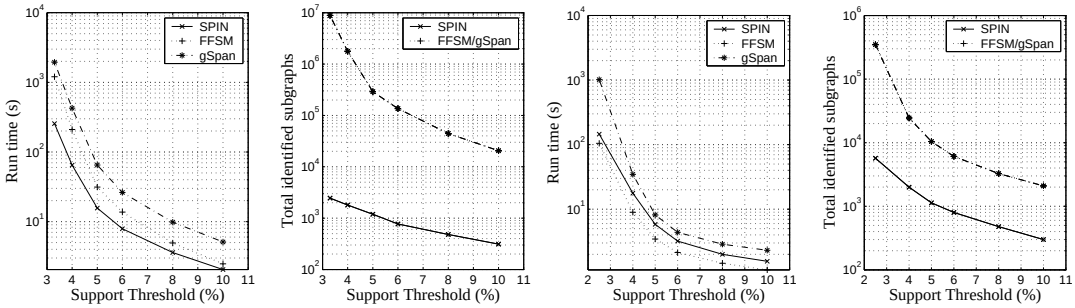

Figure 7: Left: performance comparison under different support values for DTP CA data set using SPIN, FFSM and gSpan. Right: Total frequent patterns identified by the algorithms.

4. RELATED WORK

Knowledge discovery from semi-structured data sets is an active topic in the data mining/machine learning community. Many different pattern definitions were proposed from different perspectives such as finding patterns from a single large network [15], finding approximately matched patterns [18], mining patterns using domain knowledge from bioinformatics [9], and finding frequent subgraphs. The later one is the focus of our paper.

Recent subgraph mining algorithms can be roughly classified into two categories. Algorithms in the first category use a level-wise search scheme based on the Apriori property to enumerate the recurrent subgraphs [13, 14]. Rather than growing a graph by one single node/edge at a time, Vanetik *et al.* recently proposed an Apriori-based algorithm using paths as building blocks with a novel support definition [19].

Algorithms in the second category use a depth-first search to enumerate candidate frequent subgraphs [20, 21, 2, 10]. As demonstrated in these papers, depth first algorithms provide advantages over level-wise search for (1) better memory utilization and (2) efficient subgraph testing, e.g. it usually permits the subgraph test to be performed incrementally at successive levels during the search [10].

Our current work benefits extensively from existing algorithms for maximal itemset mining such as [3, 7] and frequent subtree mining algorithms [1, 22].

5. CONCLUSION AND FUTURE WORK

In this paper we present SPIN, an algorithm to mine maximal frequent subgraphs from a graph database. A new framework, which partitions frequent subgraphs into equivalence classes is proposed together with a group of optimization techniques. Compared to current state-of-the-art subgraph mining algorithms such as FFSM and gSpan, SPIN offers very good scalability to large graph databases and at least an order of magnitude performance improvement in synthetic graph data sets. The efficiency of the algorithm is also confirmed by a benchmark chemical data set. The algorithm of compressing large number of frequent subgraphs to a much smaller set of maximal subgraphs will help us to investigate demanding applications such as finding structure patterns from proteins in the future.

Acknowledgement We thank Dr. Jack Snoeyink at the University of North Carolina for helpful discussions about the paper.

6. REFERENCES

[1] T. Asai, K. Abe, S. Kawasoe, H. Arimura, and H. Sakamoto. Efficiently substructure discovery from large semi-structured data. *SDM*, 2002.

[2] C. Borgelt and M. R. Berhold. Mining molecular fragments: Finding relevant substructures of molecules. In *Proc. International Conference on Data Mining'02*.

[3] D. Burdick, M. Calimlim, and J. Gehrke. Mafia: A maximal frequent itemset algorithm for transactional databases. *ICDE*, 2001.

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

[5] A. Deutsch, M. F. Fernandez, and D. Suciu. Storing semistructured data with STORED. *in SIGMOD*, pages 431–442, 1999.

[6] R. Goldman and J. Widom. Dataguides: Enabling query formulation and optimization in semistructured databases. *In VLDB'97*.

[7] K. Gouda and M. J. Zaki. Efficiently mining maximal frequent itemsets. *ICDM*, 2001.

[8] J. Hu, X. Shen, Y. Shao, C. Bystroff, and M. J. Zaki. Mining protein contact maps. *2nd BIOKDD Workshop on Data Mining in Bioinformatics*, 2002.

[9] J. Huan, W. Wang, D. Bandyopadhyay, J. Snoeyink, J. Prins, and A. Tropsha. Mining protein family specific residue packing patterns from protein structure graphs. In *Eighth Annual International Conference on Research in Computational Molecular Biology (RECOMB)*, pages 308–315, 2004.

[10] J. Huan, W. Wang, and J. Prins. Efficient mining of frequent subgraphs in the presence of isomorphism. *in ICDM'03*, 2003.

[11] J. Huan, W. Wang, J. Prins, and J. Yang. Spin: Mining maximal frequent subgraphs from graph databases. *UNC Technical Report TR04-018*, 2004.

[12] J. Huan, W. Wang, A. Washington, J. Prins, and A. Tropsha. Accurately classify protein family based on coherent subgraph mining. *in Pacific Symposium on Biocomputing*, 2004.

[13] A. Inokuchi, T. Washio, and H. Motoda. An apriori-based algorithm for mining frequent substructures from graph data. *In Proc. of the 4th European Conf. on Principles and Practices of Knowledge Discovery in Databases (PKDD)*, pages 13–23, 2000.

[14] M. Kuramochi and G. Karypis. Frequent subgraph discovery. In *Proc. International Conference on Data Mining'01*.

[15] M. Kuramochi and G. Karypis. Finding frequent patterns in a large sparse graph. *SDM*, 2004.

[16] J. Pei, G. Dong, W. Zou, and J. Han. On computing condensed frequent pattern bases. *ICDM*, 2002.

[17] S. Raghavan and H. Garcia-Molina. Representing web graphs. *In Proceedings of the IEEE Intl. Conference on Data Engineering*, 2003.

[18] N. Vanetik and E. Gudes. Mining frequent labeled and partially labeled graph patterns. *ICDE*, 2004.

[19] N. Vanetik, E. Gudes, and E. Shimony. Computing frequent graph patterns from semi-structured data. *Proc. International Conference on Data Mining'02*, 2002.

[20] X. Yan and J. Han. gspan: Graph-based substructure pattern mining. In *Proc. International Conference on Data Mining'02*.

[21] X. Yan and J. Han. Closegraph: Mining closed frequent graph patterns. *KDD'03*, 2003.

[22] M. Zaki. Efficiently mining freqeunt trees in a forest. *SIGKDD*, 2002.

[23] M. J. Zaki and C. J. Hsiao. Charm: An efficient algorithm for closed itemset mining. *In SDM'02*.