Efficient Mining of Frequent Subgraphs in the Presence of Isomorphism

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

Abstract

Frequent subgraph mining is an active research topic in the data mining community. A graph is a general model to represent data and has been used in many domains like cheminformatics and bioinformatics. Mining patterns from graph databases is challenging since graph related operations, such as subgraph testing, generally have higher time complexity than the corresponding operations on itemsets, sequences, and trees, which have been studied extensively. In this paper, we propose a novel frequent subgraph mining algorithm: FFSM, which employs a vertical search scheme within an algebraic graph framework we have developed to reduce the number of redundant candidates proposed. Our empirical study on synthetic and real datasets demonstrates that FFSM achieves a substantial performance gain over the current start-of-the-art subgraph mining algorithm gSpan.

1. Introduction

Mining frequent patterns of semi-structured data such as trees and graphs has attracted much research interest because of its wide-range application areas such as bioinformatics and cheminformatics [1, 4], web log mining [10], video indexing [6], and efficient database indexing [2]. Given a set S of labeled graphs (referred to as a *graph database*), the *support* of an arbitrary graph g is the fraction of all graphs in S of which g is a subgraph [9, 8, 3]. Graph g is *frequent* if the support of g meets a certain support threshold (*minSupport*). The problem of *frequent subgraph mining* is to find all (connected) frequent subgraphs from a graph database.

At the core of any frequent subgraph mining algorithm are two computationally challenging problems 1) subgraph isomorphism: determining whether a given graph is a subgraph of another graph and 2) an efficient scheme to enumerate all frequent subgraphs. Generally the number of possible isomorphisms/subgraphs increases with the graph size, the graph complexity, and the number of graphs in graph databases. To develop a solution that scales to complex graphs and large databases, it is imperative to focus on efficient frequent pattern enumeration and isomorphism test algorithms. This is the basic motivation for this work.

Several efficient subgraph mining algorithms have been proposed and a recent review is presented in [3]. The algorithms most closely related to our current effort are [9, 1].

In [9], a novel canonical form of graphs called DFS code and a novel data structure called *DFS code tree* are proposed. Involving a simple single-edge growth scheme and Ullman's subgraph matching algorithm, the preorder traversal of the DFS code tree enumerates all frequent connected subgraphs of a graph database.

In [1], a depth first scheme is proposed following the idea of the Eclat association mining algorithm [11]. Candidate graphs are proposed guided by information from sibling nodes.

Contributions We developed FFSM (**F**ast **F**requent **S**ubgraph **M**ining) targeting efficient subgraph testing and a better candidate subgraph enumeration scheme. The key features of our method are: (i) a novel graph canonical form and two efficient candidate proposing operations: FFSM-Join and FFSM-Extension, (ii) an algebraic graph framework (suboptimal CAM tree) to guarantee that all frequent subgraphs are enumerated unambiguously and (iii) completely avoiding subgraph isomorphism testing by maintaining an embedding set for each frequent subgraph.

Our experimental study shows that FFSM is competitive with gSpan on all inputs and outperforms gSpan by a factor of seven on a commonly studied chemical compound benchmark.

2. Mining Frequent Subgraphs

2.1. Canonical Adjacency Matrix

In FFSM, every graph is represented by an adjacency matrix M. Slightly different from the adjacency matrix used for an unlabeled graph, every diagonal entry of M is filled with the label of the corresponding node and every

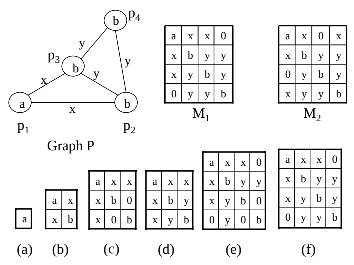

Figure 1. Top: A labeled graph P and two adjacency matrices for P. After applying the total ordering, we have $code(M_1)$ = "$axbxyb0yyb$" $\geq$ $code(M_2)$ = "$axb0ybxyyb$". For an adjacency matrix M, each off-diagonal none-zero entry in the lower triangular part is referred to as an *edge entry*. All edge entries are ordered according to their relative positions in the code. For M_1, $m_{2,1}, m_{4,3}$, and $m_{4,2}$ are the first, last, second-to-last edge entries of M_1, respectively. $m_{4,4}$ is denoted as the last node of M_1. Bottom: examples of submatrices. Matrix (a) is the submatrix of matrix (b), which itself is the submatrix of (c) and so forth.

off-diagonal entry is filled with the label of the corresponding edge, or zero if there is no edge.

Given an $n \times n$ adjacency matrix M of a graph G with n nodes, we define the *code* of M, denoted by $code(M)$, as the sequence of lower triangular entries of M (including entries on the diagonal) in the order: $m_{1,1}m_{2,1}m_{2,2}...m_{n,1}m_{n,2}...m_{n,n-1}m_{n,n}$ where $m_{i,j}$ represents the entry at the ith row and jth column in M.

We use standard lexicographic order on sequences to define a total order of two arbitrary codes p and q. Given a graph G, its *canonical form* is the maximal code among all its possible codes. The adjacency matrix M which produces the canonical form is denoted as G's *canonical adjacency matrix* (CAM). For example, the adjacency matrix M_1 shown in Figure 1 is the CAM of the graph P in the same figure, and $code(M_1)$ is the canonical form of the graph.

Notice that we use maximal code rather than the minimal code used by [5, 4] in the above canonical form definition. This definition provides important properties for subgraph mining, as explained below.

Theorem 2.1 *Given a connected graph G and one of its subgraphs H, let G's CAM be A and H's CAM be B, then we have $code(A) \geq code(B)$.*

We define the *maximal proper submatrix* (submatrix in short) of an adjacency matrix M as the matrix obtained by removing the last edge entry e of M (also removing the symmetric entry of e in M and the resulting unconnected node, if applicable). For submatrices we have the following two corollaries:

Corollary 2.2 *Given a CAM M of a connected graph G and M's submatrix N, N represents a connected subgraph of G.*

Corollary 2.3 *Given a connected graph G with CAM M, M's submatrix N, and a graph H which N represents, N is the CAM of H.*

Several examples of submatrices are given at the bottom of Figure 1. The formal proof of the theorem and corollaries are presented in [3] and omitted here due to space limitations.

If we let an empty matrix be the submatrix of any matrix with size 1, we can organize the CAMs of all connected subgraphs of a graph G into a rooted tree as follows: (i) The root of the tree is an empty matrix; (ii) Each node in the tree is a distinct connected subgraph of G, represented by its CAM; (iii) For a given none-root node (with CAM M), its parent is the graph represented by M's submatrix.

The tree obtained in this fashion is denoted as the *CAM tree* of the graph G.

2.2 Exploring the CAM Tree: Join, Extension and Suboptimal CAMs

The current methods for enumerating all the subgraphs might be classified into two categories: one is the join operation adopted by FSG and AGM [4, 5] and another one is the extension operation proposed by [1, 9]. The major concerns for the join operation are that a single join might produce multiple candidates and that a candidate might be redundantly proposed by many join operations [5]. The concern for the extension operation is to restrict the nodes that a newly introduced edge may attach to.

We list some of the key design challenges to achieve efficient subgraph enumeration:

(i) Can we design a join operation such that every distinct CAM is generated only once?

(ii) Can we improve the join operation such that only a few (say at most two) CAMs are generated from a single join operation?

(iii) Can we design an extension operation such that every edge might be attached to only one node in a graph represented by its CAM?

In order to tackle these challenges, we augment the CAM tree with a set of *suboptimal canonical adjacency matrices*, and introduce two new operations: FFSM-Join and FFSM-Extension.

Given two adjacency matrices M and N, we define a binary operation *join* which produces a set of matrices as the result of superimposing M and N. We define a unary operation *extension* on a matrix M to produce a set of matrices, each of which has one additional node v and one additional edge entry connecting v and the last node in M. Examples of the join and extension operations are given in Figure 2

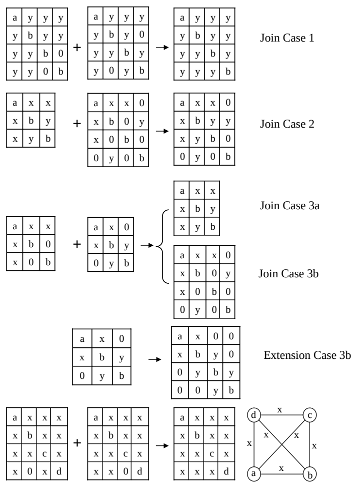
Figure 2. Examples of the join/extension operation

(further details are given in [3]). At the bottom of Figure 2 we show a case in which a graph might be redundantly proposed by FSG $\binom{6}{2}$ = 15 times (Joining any two distinct five-edge subgraphs G_1 and G_2 of the graph G will restore G by the join operation proposed by FSG). As shown in the graph, FFSM-Join completely removes the redundancy after "sorting" the subgraphs by their canonical forms.

Given a graph G, a *suboptimal canonical adjacency matrix (simply, suboptimal CAM)* of G is an adjacency matrix M of G such that its submatrix N is the CAM of the graph N represents. By definition, every CAM is a suboptimal CAM (by Corollary 2.3) and we denote a *proper suboptimal CAM* as a suboptimal CAM that itself is not a CAM.

Clearly, all suboptimal CAMs of a graph G could be organized as a tree in a similar way to the construction of the CAM tree. One such example for the graph in Figure 1 is shown in Figure 3. The suboptimal CAM tree is "complete" in the sense that all nodes could be enumerated by either a join operation or an extension operation. This is formally stated in the following theorem.

Theorem 2.4 *For a graph G, let $C_{k-1}(C_k)$ be the set of the suboptimal CAMs of all (k-1)-edge (k-edge) subgraphs of G ($k \geq 2$). Every member of C_k can be enumerated unambiguously either by joining two members of C_{k-1} or by extending a member in C_{k-1}.*

2.3. Embeddings of a Frequent Subgraph

Given a graph $G = (V, E, \Sigma, l)$ where Σ is a set of available labels and $l : V \cup E \rightarrow \Sigma$ is a function assign labels to vertices and edges [3], a node list $L = u_1, u_2, \ldots, u_n \subset V$ is *compatible* with an $n \times n$ adjacency matrix M iff: (i)$\forall i, (m_{i,i} = l(u_i))$ and (ii)$\forall i, j(i \neq j), (m_{i,j} \neq 0 \Rightarrow (m_{i,j} = l(u_i, u_j))$, where $0 < i, j \leq n$.

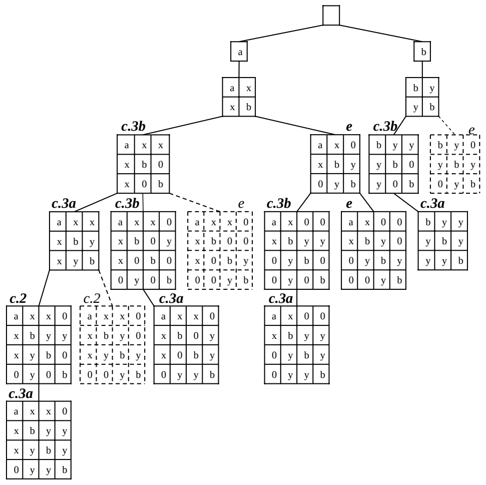
Figure 3. the Suboptimal CAM Tree for graph P in Figure 1. Matrices with solid boundary are CAMs and those with dash boundary are proper suboptimal CAMs. The label on top of each adjacency matrix M indicates the operation by which M might be proposed. A join operation is specified by a label **c.** and then the type of the operation (e.g. **c.3a** stands for join case3a). **e** specifies an extension operation.

Given a suboptimal CAM M with size n, a graph G in a graph database GD, and a node list L of G compatible with M, an *embedding* o_M of M is a two-element tuple o_M = (g_i, L) where g_i is the graph G's transaction id. The set of all possible embeddings of a suboptimal CAM is defined as its *embedding set*.

Given two suboptimal CAMs P and Q, and a suboptimal CAM $A \in join(P, Q)$, the relation between A's embedding set and those of P and Q can easily be established. For example, for join case 1, we have $O_A = O_P \cap O_Q$, where O_A, O_P, and O_Q are the embedding sets of suboptimal CAM A, P, and Q, respectively. For other join types/extensions, similar relations can be obtained with details given in [3].

FFSM

```
1: $S \leftarrow$ { the CAMs of the frequent nodes }
2: $P \leftarrow$ { the CAMs of the frequent edges }
3: FFSM-Explore($P, S$);
```

FFSM-Explore (P, S)

```
1: for $X \in P$ do
2:   if $(X.isCAM)$ then
3:     $S \leftarrow S \cup \{X\}, C \leftarrow \Phi$
4:     for $Y \in P$ do
5:       $C \leftarrow C\cup$ FFSM-Join($X, Y$)
6:     end for
7:     $C \leftarrow C\cup$ FFSM-Extension($X$)
```

```
 8:         remove CAM(s) from $C$ that is either infrequent
            or not suboptimial
 9:         FFSM-Explore($C, S$)
10:     **end if**
11: **end for**
```

3. Experimental Study

We performed our experimental study using a single processor of a 2GHz Pentium Xeon with 512KB L2 cache and 2GB main memory, running RedHat Linux 7.3. The FFSM algorithm is implemented using the C++ programming language and compiled using g++ with O3 optimization. For gSpan, we used an executable kindly provided by Xifeng Yan and Jiawei Han for performance comparison purpose.

Chemical Compound Datasets We used a set of benchmark chemical compound datasets to evaluate the performance of the FFSM algorithm. The first two we used are from the DTP AIDS Antiviral Screen dataset from National Cancer Institute. In this dataset, chemicals are classified into three classes: confirmed active (**CA**), confirmed moderately active (**CM**) and confirmed inactive (**CI**) according to experimentally determined activities against AIDS virus. There are total 423, 1083, and 42115 chemicals in the three classes, respectively. For our own purposes, we formed two datasets consisting of all CA compounds and of all CM compounds and refer to them as DTP CA/DTP CM thereafter. The DTP datasets can be downloaded from *http://dtp.nci.nih.gov/docs/aids/aids_data.html*.

The third dataset we used is the Predicative Toxicology Evaluation Challenge (PTE) [7], which can be downloaded from *http:://web.comlab.ox.ac.uk/oucl/research/areas/ machlearn/PTE/*. We follow exactly the same procedure described in [1, 9] for building graphs from all the above datasets.

Figure 4 shows the performance comparison between FFSM and gSpan, using various support thresholds for the DTP CM dataset. From the curve, we find that FFSM always outperforms gSpan by a factor up to seven. We observe a similar trend using DTP CA and PTE dataset and find a speedup of two fold and three fold, respectively [3].

Synthetic Datasets We also tested FFSM and gSpan on various synthetic graph datasets and found FFSM is always competitive with gSpan. Details are provided in [3].

4 Conclusions

We presented a new algorithm FFSM for the frequent subgraph mining problem. Comparing to existing algorithms, FFSM achieves substantial performance gain by efficiently handling the underlying subgraph isomorphism problem, which is a time-consuming step and by introducing two efficient subgraph enumeration operations, together with an algebraic graph framework developed for reduce the number of redundant candidates proposed. Performance evaluation using various real datasets demonstrated a wide margin performance gain of FFSM over gSpan. The efficiency of FFSM is further confirmed using the synthetic datasets.

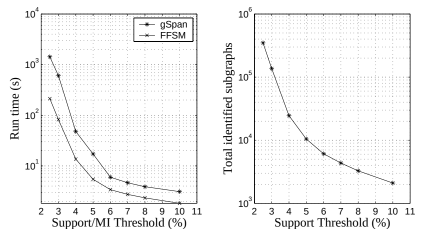

Figure 4. Left: FFSM and gSpan performance comparison under different support values for DTP CM dataset. Right: Total frequent pattern identified by the algorithms.

Acknowledgement We thank Mr. Michihiro Kuramochi and Dr. George Karypis in University of Minnesota for providing the synthetic data generator. We thank Mr. Xifeng Yan and Dr. Jiawei Han in University of Illinois at Urbana Champaign for providing the gSpan executable.

References

[1] C. Borgelt and M. R. Berhold. Mining molecular fragments: Finding relevant substructures of molecules. *In ICDM'02*.

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

[3] J. Huan, W. Wang, and J. Prins. Efficient mining of frequent subgraph in the presence of isomorphism. *UNC computer science technique report TR03-021*, 2003.

[4] A. Inokuchi, T. Washio, and H. Motoda. An apriori-based algorithm for mining frequent substructures from graph data. *In PKDD'00*.

[5] M. Kuramochi and G. Karypis. Frequent subgraph discovery. *In ICDM'01*.

[6] K. Shearer, H. Bunks, and S. Venkatesh. Video indexing and similarity retrieval by largest common subgraph detection using decision trees. *Pattern Recognition,* 34(5):1075–91, 2001.

[7] A. Srinivasan, R. D. King, S. H. Muggleton, and M. Sternberg. The predictive toxicology evaluation challenge. *In Proc. of the 15th International Joint Conferrence on Artificial Intelligence (IJCAI)*, pages 1–6, 1997.

[8] N. Vanetik, E. Gudes, and E. Shimony. Computing frequent graph patterns from semi-structured data. *ICDM'02*, 2002.

[9] X. Yan and J. Han. gspan: Graph-based substructure pattern mining. *In ICDM'02*.

[10] M. J. Zaki. Efficiently mining frequent trees in a forest. *In SIGKDD'02*.

[11] M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New algorithms for fast discovery of association rules. *In SIGKDD'97*.