MARGIN: Maximal Frequent Subgraph Mining *

Lini T Thomas   Satyanarayana R Valluri   Kamalakar Karlapalem
Center For Data Engineering, IIIT, Hyderabad
{lini,satya}@research.iiit.ac.in, kamal@iiit.ac.in

Abstract

The exponential number of possible subgraphs makes the problem of frequent subgraph mining a challenge. The set of maximal frequent subgraphs is much smaller to that of the set of frequent subgraphs, thus providing ample scope for pruning. MARGIN is a maximal subgraph mining algorithm that moves among promising nodes of the search space along the "border" of the infrequent and frequent subgraphs. This drastically reduces the number of candidate patterns considered in the search space. Experimental results validate the efficiency and utility of the technique proposed.

1 Introduction

Discovering interesting patterns in large datasets has a wide range of applications. Data mining techniques are applied to extract patterns from complex data in a variety of domains. Many applications require the computation of maximal frequent subgraphs such as in mining contact maps [2], finding maximal frequent patterns in metabolic pathways [4], and finding the set of large cohesive web pages.

In this paper, we propose a technique that mines the maximal frequent subgraphs of a graph database. The set of maximal frequent subgraphs is significantly smaller than the set of frequent subgraphs [3] thus providing scope for ample pruning of the exponentially large search space.

Given a graph dataset $\mathbb{D} = \{G_1, G_2, .., G_n\}$ of n graphs, $Sup(g)$ denotes the number of graphs (in $\mathbb{D}$) in which g is a subgraph. A subgraph g is frequent if $Sup(g) \geq minSup$ (a minimum support threshold). The problem of maximal frequent subgraph mining is to find all frequent subgraphs g_i such that there exists no frequent subgraph g_j where g_i is a subgraph of g_j. A typical approach to the maximal frequent subgraph mining problem has been to modify the apriori based approach with additional pruning steps [3].

The set of candidate subgraphs which are likely to be maximally frequent are the set of n-edge frequent subgraphs that have a $n+1$-edge infrequent supergraph. We refer to such a set of nodes in the lattice as the set of f-cut-$nodes$. The MARGIN algorithm computes such a candidate set efficiently by recursively invoking the $ExpandCut$ step within the MARGIN algorithm. By a post-processing step it finds all maximally frequent subgraphs $\mathbb{MF}$. The search space of apriori based algorithms [7, 8, 3] corresponds to the region below the f-cut-$nodes$ in the graph lattice as shown in Figure 1. On the other hand, MARGIN explores a much smaller search space by visiting the lattice around the f-cut-$nodes$.

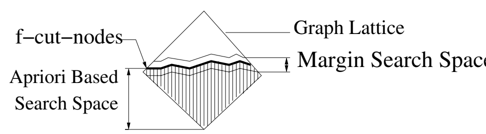


Figure 1. Search Space Explored

Contribution: A novel algorithm to find maximal frequent subgraphs is presented. The detailed proof of the algorithm has been given in the technical report [6]. The viability of this technique in efficiently finding maximal frequent subgraphs is shown through experimental results.

In section 2, we develop the formalism used in the paper. In section 3, we present the MARGIN algorithm. We report our performance result in section 4 and conclude our study in section 5.

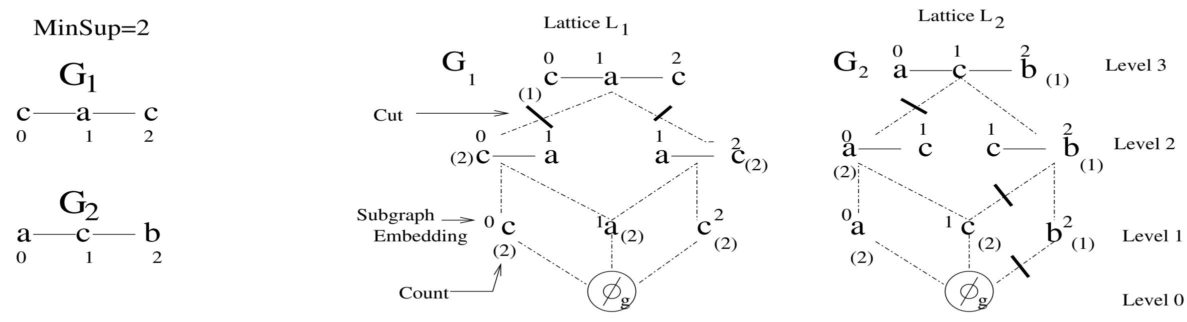
2 Preliminary Concepts

In this section we provide the necessary background and notation.

We denote the relationship "subgraph of" using $\subseteq_g$. We conceptualise the search space for finding $\mathbb{MF}$ in the form of a graph lattice. Figure 2(b) shows the graph lattices of the graphs $G_i \in \mathbb{D}$ in Figure 2(a). Every node in the lattice is the embedding of a connected subgraph of G_i. Every embedding of a subgraph of G_i occurs exactly once in the lattice. In Figure 2(b), the graph a-c occurs twice in the Lattice L_1 since it is present twice in the graph G_1. The bottom most node corresponds to the empty subgraph ϕ_g

* This work was made possible by the grant from The Boeing Company.

Proceedings of the Sixth International Conference on Data Mining (ICDM'06)
0-7695-2701-9/06 $20.00 © 2006 IEEE



(a) Graph Database: D={G 1,G 2} **(b) Graph Lattice: L 1 ,L 2**

Figure 2. Example Lattice

and the top most nodes correspond to G_i. A node C is a child of the node $P \neq \phi_g$ in the lattice L_i, if $P \subseteq_g C$ and C and P *differ by exactly one edge*. The node P is a parent of such a node C in the lattice. We define all single node subgraphs to be children of the node ϕ_g and ϕ_g to be the parent of all the single node subgraphs. ϕ_g *is considered to be always frequent*. An edge exists in the lattice between every pair of child and parent nodes.

Example: Consider $\mathbb{D} = \{G_1, G_2\}$ in Figure 2(a). To keep the example simple, we assume that all the edge labels are identical and hence are not shown in the figure. The corresponding latttice L_1, L_2 of G_1, G_2 respectively are given in Figure 2(b). The bottom most node corresponds to the empty subgraph ϕ_g and the top most nodes correspond to graph $G_i \in \mathbb{D}$. The subgraphs a-c and c occur twice in L_1 since there are two embeddings 1-0,1-2 of a-c and 0,2 of c in G_1. The children of a node N in the lattice denote all the supergraphs of the embeddings of N that can be obtained by extending N by one edge. For instance, the child of either embeddings of the subgraph $a - c$ in L_1 is the embedding of subgraph c-a-c (by adding the edge c-a). Similarly, the embeddings of subgraphs a-c and c-b are the parents of the embedding of a-c-b in L_2.

For a given graph G, the size of the graph (denoted by $|G|$), refers to the number of edges present in G. All the subgraphs of equal size form a $level$ in the lattice L_i of G_i. The node corresponding to ϕ_g forms level 0, singleton vertex graphs form level 1 and the nodes of size i form level $i+1$ for $i > 0$ (Figure 2(b)).

Definition 1 ***Cut:*** *A cut between two nodes in a lattice represented by $(C \dagger P)$ is defined as an ordered pair (C, P) where P is the parent of $C \in L_i$ and C is not frequent while P is frequent. The frequent node P of a cut is represented by $f(\dagger)$ (frequent-†) and the infrequent node C is represented by $I(\dagger)$ (infrequent-†). The symbol † is read as 'cut'.*

Note that different embeddings of a graph g in the Lattice L_i will thus have the same count. However the subgraphs corresponding to the children of each embeddings might be different. Also while one embedding becomes a $f(\dagger)$-$node$, the other might not.

Example: Consider Figure 2(a) with $minSup$=2. The node in L_2 that corresponds to the subgraph c is a $f(\dagger)$-$node$ since it is frequent with count 2. Its parent node that corresponds to $c - b$ is infrequent with count 1 and thus is an $I(\dagger)$-$node$. Hence, this pair is marked as cut. Figure 2(b) shows the frequency count of each node in the example lattice along with all the existing cuts in the lattice L_1 and L_2 respectively.

3 The MARGIN Approach

In subsection 3.1, the intuition behind the algorithm proposed to find the maximal frequent subgraphs is presented. In subsection 3.2, the MARGIN algorithm is presented.

3.1 Intuition

We start by defining the $Upper \Diamond Property$ that holds in every lattice L_i of $G_i \in \mathbb{D}$ which we exploit in our algorithm.

Property 1 ***Upper Diamond property*** *($Upper$-$\Diamond$-**property**): Any two children C_i, C_j of a node P, where $C_i, C_j \in$ Lattice L_k of G_k for $G_k \in \mathbb{D}$, will have a common child A.*

Proof: Let e_1 and e_2 be the edges incident on the vertices n_1 and n_2 in P respectively. Let $P \cup \{e_1\} = C_j$ and $P \cup \{e_2\} = C_i$. Hence e_1 would be incident on n_1 in C_i and e_2 would be incident on n_2 in C_j. Let $A = C_j \cup \{e_2\}$.
Hence, $A = (P \cup \{e_1\}) \cup \{e_2\} = (P \cup \{e_2\}) \cup \{e_1\}$
Hence, $A = C_i \cup \{e_1\} = C_j \cup \{e_2\}$ is the common child of C_i and C_j. $\square$

The set of candidate subgraphs that are likely to become maximally frequent are the $f(\dagger)$ nodes. This is because they

Proceedings of the Sixth International Conference on Data Mining (ICDM'06)
0-7695-2701-9/06 $20.00 © 2006 IEEE

are frequent subgraphs having an infrequent child. In this paper, we present an approach that avoids traversing the lattice bottom up and instead traverses the cuts alone in each lattice L_i for $G_i \in \mathbb{D}$. We prune the set of $f(\dagger)$ nodes to give the set of maximal frequent subgraphs. The MARGIN algorithm unlike the apriori based algorithms goes directly to any one of the $f(\dagger)$ nodes of the lattice L_i and then finds all other $f(\dagger)$ nodes by cutting across the lattice L_i. We give an insight below into the approach developed.

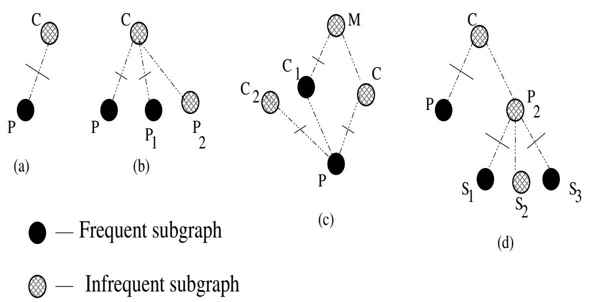


Figure 3. $ExpandCut$

Finding the initial $f(\dagger)$ node is a trivial dropping of edges one by one from the initial graph $G_1 \in \mathbb{D}$, ensuring that the resulting subgraph is connected until we find the first frequent subgraph R_i. We call the frequent subgraph found by such dropping of edges as the $Representative\ R_i$ of G_i. Our initial cut is thus $(CR_i \dagger R_i)$ where CR_i is the infrequent child of R_i. We devise an algorithm $ExpandCut$ which for one cut discovered in $G_i \in \mathbb{D}$, recursively extends the cut to generate all cuts in G_i.

Next, we provide an intuition to the $ExpandCut$ algorithm used to find the nearby cuts given any cut $(C \dagger P)$ (Figure 3(a)) as input in the lattice L_i of G_i. Recursively invoking $ExpandCut$ on each newly found cut finds all cuts in G_i using the steps given below, the proof of which is included in the technical report [6].

Step1: The node C in lattice L_i can have many parents that are frequent or infrequent, one of which is P. Consider the frequent parent P_1 in Figure 3(b). The cut $(C \dagger P_1)$ exists since P_1 is frequent while C is infrequent. Thus, for an initial cut $(C \dagger P)$, all frequent parents of C are reported as $f(\dagger)$ nodes.

Step2: Consider all the children C_1, C_2, C of any frequent parent P of C as in Figure 3(c). Each of them can be frequent or infrequent.

(a): Consider an infrequent child C_2. The cut $(C_2 \dagger P)$ exists since P is frequent while C_2 is infrequent. Thus, for an initial cut $(C \dagger P)$, for each frequent parent P_f of C that has an infrequent child C_i, the cut $(C_i \dagger P_f)$ is reported.

(b): Consider a frequent child C_1. By $Upper\text{-}\Diamond\text{-}Property$, the nodes C and C_1 have a common child M. M is infrequent as its parent C is infrequent. Hence, the cut $(M \dagger C_1)$ exists. Thus, for an initial cut $(C \dagger P)$, for each frequent parent P_f of C consider each of its frequent child C_i. The cut $(M \dagger C_i)$ is reported where M is the common child of C_i and C.

Step3: Consider all parents S_1, S_2, S_3 of an infrequent parent P_2 of C as in Figure 3(d). Each such parent can be frequent or infrequent. Consider frequent parents S_1, S_3 (Figure 3(d)) of an infrequent parent P_2 of C. Hence, the cuts $(P_2 \dagger S_1)$ and $(P_2 \dagger S_3)$. However, if step 1 is called on the cut $(P_2 \dagger S_1)$, the cut $(P_2 \dagger S_3)$ is found. Thus, for an initial cut $(C \dagger P)$, for each infrequent parent P_i of C, consider any one frequent parent S_f of P_i. $ExpandCut$ is invoked on the cut $(P_i \dagger S_f)$.

3.2 The MARGIN Algorithm

Algorithm 1 shows the $MARGIN$ algorithm to find the globally maximal frequent subgraphs $\mathbb{MF}$. Initially, $\mathbb{MF} = \emptyset$ (line 1) and the graphs in $\mathbb{D}$ are unexplored. $\mathbb{LF}$ is the set of locally maximum subgraphs in each G_i which is initially ϕ (line 3). Initially, given the graphs $\mathbb{D} = \{G_1, G_2, \ldots, G_n\}$, for each $G_i \in \mathbb{D}$, we find the representative R_i for G_i (line 4). This is done by iteratively dropping an edge from G_i until a connected frequent subgraph is found. The $ExpandCut$ algorithm is initially invoked on the cut $(CR_i \dagger R_i)$(line 5) with $LF{=}\phi$ where CR_i is the infrequent child of R_i. $ExpandCut$ finds the nearby cuts and recursively calls itself on each newly found cut. The algorithm functions in a manner that finding one cut in $G_i \in \mathbb{D}$ would find all cuts in G_i. In line 6, the globally maximal frequent subgraph set is updated by finding the maximal subgraphs among $\mathbb{MF}$ and $\mathbb{LF}$ found in G_i.

Algorithm 1: MARGIN

Input: Graph Database $\mathbb{D} = \{G_1, G_2, \ldots, G_n\}$,

Output: Set of Maximal Frequent Graphs $\mathbb{MF}$

1. $\mathbb{MF}{=}\emptyset$
2. For each $G_i \in \mathbb{D}$ do
3. $\quad \mathbb{LF}{=}\phi$
4. $\quad$ Find the representative R_i of G_i
5. $\quad ExpandCut(\mathbb{LF}, CR_i \dagger R_i)$ where CR_i is the infrequent child of R_i
6. $\quad$ Merge($\mathbb{MF}$,$\mathbb{LF}$)

Algorithm 2 shows the $ExpandCut$ algorithm which expands a given cut such that its neighboring cuts will be explored. The input to the algorithm are the set of maximal frequent subgraphs $\mathbb{LF}$ found so far (initially empty) and the cut $(C \dagger P)$.

For each parent Y_i of C, if Y_i is frequent then Y_i is added to $\mathbb{LF}$ (lines 3-4).

• For each infrequent child CY_i of Y_i, $ExpandCut$ is called on the cut $(CY_i \dagger Y_i)$ (line 6-7).

• For each frequent child CY_i of Y_i, let M be the com-

Proceedings of the Sixth International Conference on Data Mining (ICDM'06)
0-7695-2701-9/06 $20.00 © 2006 IEEE

Algorithm 2: ExpandCut($\mathbb{LF}$, $C \dagger P$)

Input:
$\mathbb{LF}$: The maximal frequent subgraphs seen so far in G_i.
Cut: $C \dagger P$

Output: The updated set of maximal frequent subgraphs $\mathbb{LF}$.

1. Let $Y_1, Y_2, \ldots, Y_c$ be the parents of C.
2. **for** each Y_i, $i = 1, \ldots, c$ **do**
3.   **if** Y_i is frequent
4. $\mathbb{LF} = \mathbb{LF} \cup Y_i$
5.   **for** each child CY_i of Y_i **do**
6.   **if** CY_i is infrequent **do**
7.   ExpandCut($\mathbb{LF}$,$CY_i \dagger Y_i$)
8.   **if** CY_i is frequent **do**
9.   Find common child M of C and CY_i
10.   ExpandCut($\mathbb{LF}$,$M \dagger CY_i$)
11.   **if** Y_i is infrequent
12.   **if** one frequent parent PY_i of Y_i exists
13.   ExpandCut($\mathbb{LF}$,$Y_i \dagger PY_i$)

mon child of C and CY_i. $ExpandCut$ is called on the cut ($M \dagger CY_i$) (line 8-10).
On the otherhand, if Y_i is infrequent and there exists atleast one frequent parent PY_i of Y_i, then, $ExpandCut$ is called on the cut ($Y_i \dagger PY_i$)(lines 11-13).

There are further optimizations possible to reduce the number of revisited cuts which are discussed briefly below. See technical report [6] for details:
1. The lines 5-10 of the $ExpandCut$ algorithm that iterate over all the children of Y_i can be replaced by calling $ExpandCut$ on just one cut ($M \dagger CY_i$), where M is the common child of CY_i and C and CY_i is a frequent child of Y_i if such a frequent CY_i exists.
2. In the invocation of $ExpandCut$ on the cut ($C_{c_i} \dagger P$) where $C_{c_i} \neq C$ is an infrequent child of P, the children of P are recomputed and revisited as they are already explored in the invocation of $ExpandCut$ on the cut ($C \dagger P$). This can be avoided by passing the appropriate information.
3. Lines 11-13 of the algorithm checks for infrequent parents Y_i of C. If Y_i is found among the infrequent subgraphs already visited, then $ExpandCut$ invoked on the cut ($C \dagger P$) skips executing lines 12-13 on Y_i.

4 Results

We implemented the MARGIN algorithm and tested it on synthetic datasets, the results of which are discussed below and on real-life datasets which are included in the technical report [6]. We ran our experiments on a 1.8GHz Intel Pentium IV PC with 1 GB of RAM, running Fedora Core 4. The code is implemented in C++ using STL and Graph Template Library [1]. We conducted experiments for comparative results with the gSpan [7] executable and *our implementation* [1] of the SPIN algorithm [3]. We compare with gSpan in order to state the saving MARGIN makes against the time of an algorithm that explores the major portion of the lattice space below the "border". Since SPIN generates maximal frequent subgraphs, we compare with it. We give time comparative results with gSpan and both time and generic operation comparisons with SPIN. Our experimental results show that MARGIN runs upto three to four times faster than SPIN, twenty times faster than gSpan on synthetic datasets and gives about seven times performance better than that of gSpan on a real-life dataset. For low support values, the number of lattice nodes visited by the MARGIN algorithm was found to be one-fifth of that of SPIN as seen in Table 1. Also, the cost of the operations involved in SPIN and MARGIN are comparable while the difference in the number of operations is huge. We generated all maximal frequent subgraphs from the frequent subgraphs obtained by gSpan and cross validated the results with that of MARGIN and SPIN.

Table 1. Lattice Space

DataSet	Lattice Nodes Visited	
Size(Support%)	**SPIN**	**MARGIN**
100 (2)	43,861	9,311
100 (5)	42,584	9,930
200 (2)	54,026	10,916
200 (5)	49,767	12,318
500 (2)	32,556	12,619
500 (5)	4,162	8,264

We generated the synthetic datasets using the graph generator software provided by [5]. The graph generator generates the datasets based on six parameters: D (the number of graphs), E,V (the number of distinct edge and vertex labels respectively), T (the average size of each graph), I (the average size of frequent graphs) and L (the number of frequent patterns as frequent graphs).

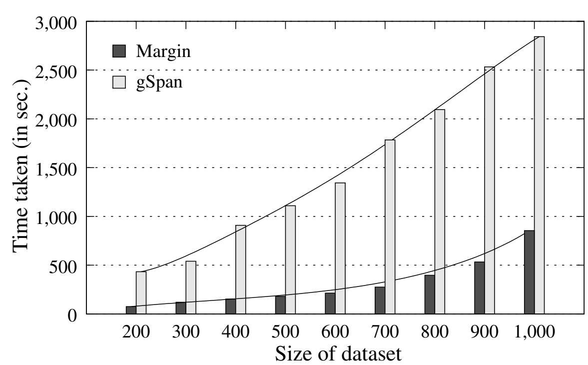


Figure 4. Running time with 2% Support

[1] The SPIN executable was not available

Proceedings of the Sixth International Conference on Data Mining (ICDM'06)
0-7695-2701-9/06 $20.00 © 2006 IEEE

Figure 4 shows the result where D is varied between 200 and 1000 graphs. The other values of the parameters used for this experiment are: L=5, E=50, V=50, I=12 and T=15. The minimum support used for each case is 2% of D. As this figure shows, MARGIN algorithm outperforms gSpan algorithm by three to eight times. Since the average size of each graph is 15 and the average size of each frequent subgraph is 12, the maximal frequent subgraphs tend to lie in the higher levels of the lattice for which MARGIN is more suited.

Experiments on varying the average size of the frequent graphs have been included in the technical report [6]. It was observed that gSpan and SPIN perform very efficiently with lower values of I (5-7). With increasing values of I, MARGIN performs better with considerable difference in the reporting time. This should be expected as for higher values of I, the lattice space explored by apriori based algorithms would increase since larger graphs are expected to be frequent.

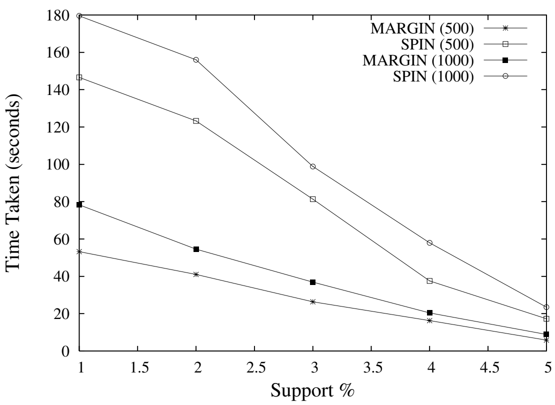


Figure 5. Comparison with SPIN

Figure 5 shows a time comparison of SPIN and MARGIN. Time with varying support has been shown for D=500 and D=1000, with other parameters set to E=10, V=10, L=10, I=5 and T=6 and varying support from 1 to 5%. With an increase in support, the number of graphs that are frequent reduce and hence the lattice space below the "border" is smaller. It can be seen that with an increase in support the time taken by MARGIN and SPIN reduce to comparable values. However, for smaller values of support which causes the "border" to be much higher up the lattice, MARGIN performs about three times better than SPIN as expected.

Since time comparison is not a good measure, we include a comparison of the most frequent complex operations of both the algorithms: the subgraph and graph isomorphic operations of the MARGIN algorithm and the subtree isomorphism and maximal-CAM-tree operations of the SPIN algorithm. It was observed that with an increase in the database size for a constant support, the number of operations of MARGIN is two to three times lesser than that of SPIN. For datasets with small graphs, as the support increases, the lattice space below the "border" decreases. The performance of MARGIN to that of SPIN thus degraded with increase in support for small graphs leading to better performance of SPIN in some cases. As T increases from 5 to 20, it was noticed that the ratio of number of operations of SPIN to that of MARGIN goes up to 20. This is because as T increases, the lattice space below the "border" increases and thus SPIN explores a bigger space as compared to MARGIN.

5 Conclusions

We present an approach to find the maximal frequent subgraphs. The candidate set that is likely to be maximally frequent are the n-edge frequent subgraphs having a $n+1$-edge infrequent supergraph. The MARGIN algorithm computes such a set efficiently and finds the maximal frequent subgraphs by a post-processing step. Experimental results show that our algorithm performs up to three(twenty) times faster than SPIN(gSpan).

Acknowledgements: We thank A. Kokkula, D. Cheboli, N. Pandey and R. Makin for helping us with some of the implementation.

References

[1] The graph template library.

[2] J. Hu, X. Shen, Y. Shao, C. Bystroff, and M. J. Zaki. Mining protein contact maps. pages 3–10. BIOKDD, 2002.

[3] J. Huan, W. Wang, J. Prins, and J. Yang. Spin: mining maximal frequent subgraphs from graph databases. pages 581–586. KDD, 2004.

[4] M. Koyuturk, A. Grama, and W. Szpankowski. An efficient algorithm for detecting frequent subgraphs in biological networks. pages 200–207. ISMB, 2004.

[5] M. Kuramochi and G. Karypis. Frequent subgraph discovery. pages 313–320. ICDM, 2001.

[6] L. Thomas, S. R. Valluri, and K. Karlapalem. Margin: Maximal frequent subgraph mining. Technical Report IIIT/TR/2006/24, IIIT, Hyderabad, July 2006.

[7] X. Yan and J. Han. gspan: Graph-based substructure pattern mining. pages 721–724. ICDM, 2002.

[8] X. Yan and J. Han. Closegraph: mining closed frequent graph patterns. pages 286–295. KDD, 2003.

Proceedings of the Sixth International Conference on Data Mining (ICDM'06)
0-7695-2701-9/06 $20.00 © 2006 IEEE