Graph Classification Based on Pattern Co-occurrence

Ning Jin
University of North Carolina
at Chapel Hill
Chapel Hill, NC, USA
njin@cs.unc.edu

Calvin Young
University of North Carolina
at Chapel Hill
Chapel Hill, NC, USA
youngc@cs.unc.edu

Wei Wang
University of North Carolina
at Chapel Hill
Chapel Hill, NC, USA
weiwang@cs.unc.edu

ABSTRACT
Subgraph patterns are widely used in graph classification, but their effectiveness is often hampered by large number of patterns or lack of discrimination power among individual patterns. We introduce a novel classification method based on pattern co-occurrence to derive graph classification rules. Our method employs a pattern exploration order such that the complementary discriminative patterns are examined first. Patterns are grouped into co-occurrence rules during the pattern exploration, leading to an integrated process of pattern mining and classifier learning. By taking advantage of co-occurrence information, our method can generate strong features by assembling weak features. Unlike previous methods that invoke the pattern mining process repeatedly, our method only performs pattern mining once. In addition, our method produces a more interpretable classifier and shows better or competitive classification effectiveness in terms of accuracy and execution time.

Categories and Subject Descriptors
H.2.8 [Database management]: Database Applications---data mining; I.5.2 [Pattern Recognition]: Design Methodology---Classifier design and evaluation; Feature evaluation and selection

General Terms
Algorithms, Experimentation, Performance

Keywords
Graph mining, graph classification, classification rule

1. INTRODUCTION
Graphs are powerful data structures for organizing vast quantities of data. Mining for graph patterns has steadily grown as a topic of interest and has found applications in a wide range of fields, including bioinformatics and chemoinformatics [9, 15, 1], database indexing [8], and web information management [17]. The interest of this paper is to utilize these graph patterns to derive a classification model to distinguish between graphs of different class labels. We focus on a binary classification that assigns a graph to either a positive class or a negative class. Note that this binary graph classification model has many applications.

CIKM'09, November 2–6, 2009, Hong Kong, China.

For example, proteins, whose structures can be represented by graphs, can be classified into two classes: those which perform a certain function and those which do not. Similarly, chemical compounds can be classified into two classes: those which are active and those which are not. Note that in these applications, the positive and negative classes may not necessarily contain comparable number of graphs. It is also possible that graphs in the negative class may be much more diverse than graphs in the positive class.

Given a training set that contains both positive graphs and negative graphs, the objective of graph classification is to build a prediction model that separates these two classes. Early work [10, 7, 2] in graph classification took a straightforward two-step approach, which first generates a set of subgraph patterns and then employs a generic classification model in the feature space constructed by mapping the occurrence of a graph pattern to a feature. A major shortcoming of this approach is the decoupling of the subgraph pattern mining and classifier construction. The number of subgraph patterns generated in the first step is usually very large and includes many patterns which may not correspond to features of high classification power. This often leads to prolonged running time and poor classification accuracy.

To overcome this drawback, recent approaches in graph classification integrate subgraph pattern mining and classifier construction. Several boosting algorithms have been proposed which look for discriminative subgraph patterns without examining all possible subgraphs [14, 16, 18]. These algorithms mine patterns repeatedly in multiple iterations. During each subsequent iteration, misclassified graphs are given higher weights. However, this approach may take many iterations to reach a high classification accuracy, resulting in long execution time. The LEAP algorithm [22] takes a novel divergence from this standard and introduces two concepts: (i) *structural leap search* and (ii) *frequency-descending mining*. This method is faster than previous methods because it is able to quickly locate patterns that individually have high discrimination power, without exploring the whole pattern space. Furthermore, it gives a much smaller pattern set than traditional graph mining algorithms, which facilitates classification model training. However, this method focuses on the discriminative power of individual patterns and hence does not work well in two scenarios. (1) When no individual pattern has high discrimination power, a group of patterns may jointly have higher discrimination power (see example in Section 3.2). LEAP is not designed for evaluating joint discrimination power of multiple patterns that have low individual discrimination powers. LEAP is therefore apt to fail in identifying these patterns. (2) Furthermore, the top-*k* patterns found by LEAP may not necessarily compose the best classifier,

especially when these *k* patterns share most of their supporting graphs. Therefore, LEAP is not a suitable stand-alone graph classification algorithm. To construct a good classification model, we need to invoke LEAP multiple times. We adjust the weight of each graph after each invocation of LEAP so that the next invocation will identify discriminative patterns that are complementary to the ones returned by earlier invocations. The union of these patterns can then be used as features to train a classifier. Another algorithm gPLS [19] adapts the powerful mathematical tool of PLS (Partial Least Squares) regression to graph mining to collect informative subgraph patterns and build a classifier directly with fewer iterations than typical boosting methods. It creates latent variables involving response variables, thus leading to better predictions. However, these latent variables have the known disadvantage of poor interpretability. CORK [20] is a subgraph-based algorithm for binary graph classification which attempts to discover frequent subgraphs that remove correspondence between graphs in the positive and negative classes. Given a set of subgraphs, the number of correspondences is the total number of pairs of graphs that cannot be discriminated by these subgraphs. The number of correspondences is sub-modular and can usually achieve good results. However, it is not perfect since subgraphs of vastly different discrimination power may have the same number of correspondences (see example in Section 2).

Therefore, we propose to investigate the discrimination power of co-occurrence of subgraph patterns and design a method to mine co-occurrence rules that can be readily used to classify graphs into positive and negative classes. These co-occurrence rules are able to capture complex graph features that offer high discrimination power. We propose an algorithm, COM (Co-Occurrence rule Miner), which employs an efficient pattern exploration order to locate subgraph patterns whose co-occurrence is indicative of graph classification. These co-occurrence rules offer higher classification accuracy as well as better interpretability than previous approaches.

Table 1. Comparison of latest graph mining algorithms for graph classification

	Repeated mining	Generates classifier	Classifier interpretability	Joint discrimination power
LEAP	Yes	No	Med	No
gPLS	Yes	Yes	Low	Yes
CORK	Yes	No	Med	No
COM	No	Yes	High	Yes

The remainder of this paper is organized as follows. Section 2 reviews related work. We introduce the problem definition in Section 3 and the pattern exploration order of COM in Section 4. The COM algorithm is presented in Section 5. Experimental results are given in Section 6. Section 7 concludes the paper.

2. RELATED WORK

Early graph mining researches focused on finding all patterns with frequency higher than a user-specified threshold. Such work includes AGM [12], FSG [13], gSpan [23], FFSM [11] and so on. They successfully solved the problem of efficient enumeration of patterns without repetition. Since then, the research focus has shifted to investigating sensible ways to confine the pattern space. gPrune [24] carried out an extensive study on graph pattern mining with constraints in which users can specify additional criterions defining the subset of patterns they are interested in. However, most of the constraints are conformation-based, so it does not work well when users have little knowledge about the graph set. Besides, only a few types of constraints can be effectively adopted in graph pattern mining. Chen et al. [6] proposed a method to represent similar patterns which have similar conformations and supporting sets by using a representative pattern. This can effectively reduce the number of patterns that must be investigated, but there is still much redundancy when these patterns are used in graph classification. Additionally, the process of finding representative patterns is performed after graph pattern mining, so it still suffers from the inefficiency of examining a huge pattern space. Yan et al. [22] made binary graph classification much more efficient by proposing the LEAP algorithm, which finds the top-*k* patterns evaluated using an objective function score that measures each pattern's significance. However, the set of top-*k* patterns may have a high level of redundancy if there is significant overlap in their supporting graphs, therefore forming a poor classification set. To solve this problem, an iterative mining framework is used in the LEAP experiments, mining optimal patterns iteratively until all graphs are covered by some patterns. Although adopting an iterative feature selection strategy can lead to high accuracy, it makes the process less efficient because LEAP needs to be called repeatedly. Furthermore, when there are few discriminative patterns, the effectiveness of LEAP may also be compromised. Shortly after LEAP, Saigo et al. [19] proposed a graph classification algorithm called gPLS by using PLS (Partial Least Square) regression, which also showed high efficiency and accuracy. Partial least squares regression has strong prediction power, but in its model PLS uses latent variables, generated from analysis of both observations and predictors, that are difficult to interpret. Thoma et al. [20] proposed a subgraph-based binary graph classification algorithm CORK. The goal is to find the most discriminative subgraph set instead of individual discriminative subgraphs. CORK uses the number of correspondences as quality criterion to measure the discrimination power of a subgraph set. It enables CORK to achieve near-optimal result because the number of correspondence is submodular. However, this criterion may be problematic in measuring discrimination power, because two subgraph sets can have exactly the same correspondence score but significantly different discrimination power. For example, if there are two subgraph sets *A* and *B*. *A* fails to discriminate 1 positive graph and 9 negative graphs; *B* fails to discriminate 3 positive graphs and 3 negative graphs. Then the number of correspondences of *A* and *B* are 9 in both scenarios, but *A* leads to a much better classifier than *B* as *A* only misclassifies one graph while *B* may misclassify three graphs.

3. PROBLEM ANALYSIS

We first introduce the terminology and notations used in this paper.

3.1 Definitions

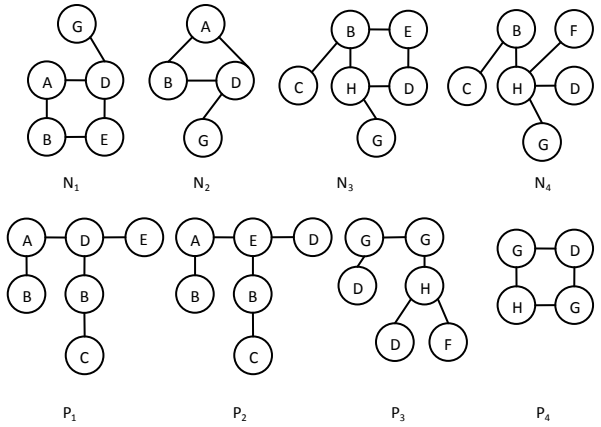

Figure 1. An example of two sets of graphs

DEFINITION 1 (Undirected Graph). A graph is denoted by $g = (V, E)$, where V is a set of nodes (vertices) and E is a set of edges connecting the nodes. Both nodes and edges may have labels.

Figure 1 shows eight undirected graphs: N_1, N_2, N_3, N_4 and P_1, P_2, P_3, P_4. For these simple illustrations, the edges are unlabeled, although our work handles graphs with labeled edges. The node set of a graph g is denoted as $V(g)$ and the edge set of g is denoted as $E(g)$. For example, in graph N_1, the node set $V(N_1)$ is $\{A, B, D, E\}$ and the edge set $E(N_1)$ is $\{A\text{-}B, A\text{-}D, D\text{-}E, B\text{-}E\}$.

DEFINITION 2 (Connectivity). For two nodes v_0 and v_n in graph g, if there exists a sequence of nodes $v_0, v_1, v_2, ..., v_n$ such that there is an edge in g connecting v_i and v_{i+1}, for any i, $0 \le i < n$, then v_0 and v_n are connected. For a graph g, if every two nodes in g are connected, then g is connected.

All graphs in Figure 1 are connected graphs.

DEFINITION 3 (Subgraph Isomorphism). The label of a node u is denoted as $l(u)$ and the label of an edge (u, v) is denoted as $l((u, v))$. For two graphs g and g', if there is an injection $f: V(g) \rightarrow V(g')$, such that for any node v in $V(g)$, $l(u) = l(f(u))$ and for any edge (u, v) in $E(g)$, $l((u, v)) = l((f(u), f(v)))$, then g is a subgraph of g' and g' is a supergraph of g, or g' supports g.

In this paper, a subgraph is also called a pattern and we are only interested in connected subgraphs with at least two nodes. For example, A-D is a pattern in Figure 1 because A-D is a 2-node connected subgraph of N_1, N_2 and P_1.

A pattern is *frequent* if it is supported by some threshold proportion of graphs in a graph set.

DEFINITION 3 (Frequency). Given a graph set S, for a subgraph g, let $S' = \{g' \mid g' \text{ is in } S \text{ and } g' \text{ supports } g\}$, then the *frequency* of g is $|S'| / |S|$.

For example, in graph set $\{N_1, N_2, N_3, N_4\}$ in Figure 1, the supporting set of A-D is $\{N_1, N_2\}$ and its frequency is 0.5.

In order to build a classification model, we use a training set containing a *positive* graph set and a *negative* graph set to generate rules that discriminate between graphs in the two sets. Without loss of generality, we assume that the positive set is the interesting set and the negative set is the decoy. In this paper, we use graph set $\{N_1, N_2, N_3, N_4\}$ in Figure 1 as the negative set and graph set $\{P_1, P_2, P_3, P_4\}$ as the positive set to illustrate our intuition and algorithm.

3.2 Challenges

The first and perhaps biggest challenge in using patterns in graph classification is feature selection. The number of patterns in a graph set may be exponential to the graph size. It is infeasible and unnecessary to use all of them in learning a classification model. Therefore, some measurement needs to be adopted to choose a subset of patterns as features. However, even if a measurement is given, applying it to all patterns is often an extremely time-consuming process because of the exponential pattern space.

In addition to the enormous pattern space, another challenge is that the graph set may not have many individual patterns that are highly discriminative. In a binary classification, for instance, all patterns may occur equally frequently (or infrequently) in both the positive set and negative set, which makes it difficult to separate the two sets merely based on individual patterns. In the example in Figure 1, there is no individual subgraph that occurs in more than one positive graph and cannot be found in negative graphs. Thus, we propose to consider pattern co-occurrence in building classification model. Even if all patterns occur almost equally frequently in both positive and negative sets, co-occurrence of several patterns may still be discriminative. For example, in Figure 1, pattern A-B and pattern B-C both occur in half of the positive graphs and half of the negative graphs. Normally these two patterns are not considered discriminative by most previous methods. However, A-B and B-C always occur together in positive graphs but never co-occur in negative graphs. Therefore the co-occurrence of A-B and B-C is very discriminative.

A third challenge arises from the asymmetry of the positive and negative graph sets in terms of the number of graphs in each set and the similarity of these graphs. This requires the classification model to be able to give different treatments for positive and negative graphs.

3.3 Our Contribution

We propose a method COM to mine co-occurrence rules. Our method can be integrated into any commonly used subgraph mining algorithms. In this paper, we use FFSM [11] as an example algorithm for frequent subgraph mining to illustrate the principle of COM. Several key features of the FFSM algorithm make it an ideal choice for this purpose: (i) a simple graph canonical form, (ii) an algebraic graph framework 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.

The COM algorithm starts with the set of single-edge patterns and incrementally extends these patterns using the candidate-proposing operation FFSM-Extension [11]. The discrimination score is then defined as $d(p) = log\left(\frac{f_p(p)}{f_n(p)}\right)$, where $f_p(p)$ and $f_n(p)$ represent the pattern's frequency in the positive set and the negative set, respectively. Focusing on discriminative patterns reduces the pattern space significantly.

COM organizes patterns into teams of co-occurrence rules to form a rule set. Whenever a new pattern is generated, the

discrimination score of every rule is calculated with the pattern's inclusion and then the pattern is inserted into the rule that yields the greatest increase in discrimination score. The algorithm terminates when either all patterns have been found or the rule set can successfully identify all positive graphs. A graph g is classified to be positive if it satisfies *at least one* rule from the rule set. Taking advantage of co-occurrence information of patterns enables us to find features with high discrimination power even when there are few discriminative patterns because it is possible that the co-occurrence of several patterns may be frequent in the positive set and rare in the negative set when each individual pattern is almost equally frequent in both sets. Using co-occurrence information may also improve time efficiency since co-occurrence of several weakly discriminative patterns can be as powerful as a strongly discriminative pattern, therefore our method does not require global optimization. Additionally, co-occurrence rules are formed by co-occurring patterns, and thus have better interpretability than most other classifiers, such as SVM, based on mathematical models. The idea of subgraph co-occurrence rules may seem similar to CBA (Classification Based on Association) or the usage of co-occurrence in text mining. However, in our graph classification problem, subgraph patterns (analogous to "items" in CBA or "units of text" in text mining) are not available prior to mining and it is impractical to enumerate all of them due to the exponential pattern space. Our subgraph co-occurrence discovery task is more challenging than CBA and text mining using co-occurrence because we need to efficiently integrate subgraph mining and co-occurrence mining, which, as far as we know, has not been thoroughly studied before.

4. PATTERN EXPLORATION ORDER

4.1 Pattern Exploration Order Based on CAM

All patterns in a graph set can be organized in a tree structure. Each tree node represents a pattern and is a supergraph of its parent node, with the root node being an empty graph. Traversing this tree can enumerate all distinct patterns without repetition. To facilitate this, a graph canonical code is often employed. Several graph coding methods have been proposed for this purpose. We adopt the CAM (Canonical Adjacency Matrix) code [11] in this paper, but our method can be easily applied to other graph coding strategies.

DEFINITION 4 (Code). The code of a graph g is the sequence formed by row-wise concatenating the lower triangle entries of an adjacency matrix M of g.

The code of a graph g is not unique because g may have up to $(n!)$ different adjacency matrices. So we use standard lexicographic order on sequences to define a total order on all possible codes. The matrix that produces the maximal code for a graph g is called the Canonical Adjacency Matrix of g and the corresponding code is the CAM code of g. The CAM code of a graph g is unique. It is proved that exploring a pattern tree with the CAM codes [11] can enumerate all patterns without repetition.

A	1	0
1	D	1
0	1	E

adjacency matrix M

D	1	1
1	A	0
1	0	E

adjacency matrix N

Figure 2. An example of adjacency matrices

For example, in Figure 1, *A-D-E* is a pattern in graph P_1. Figure 2 shows two different adjacency matrices of *A-D-E*. A "1" indicates the existence of an edge between two nodes while a "0" indicates the absence of an edge. Adjacency matrix M leads to code *A1D01E* and adjacency matrix N leads to code *D1A10E*. Although both of them are correct codes of *A-D-E, D1A10E* is less than *A1D01E* lexicographically. In fact, *A1D01E* is the largest code for *A-D-E*, so it is the CAM code and adjacency matrix M is the canonical adjacency matrix.

4.2 Scoring Function

Even with an efficient graph coding scheme, it is still intractable to find graph features by exploring the entire pattern tree because of its prohibitive size. However, not all patterns are suitable to be used as graph features and usually a small number of discriminative patterns are sufficient for effective classification. Therefore, we only need to find a subset of patterns that can promise an effective classifier.

Selecting graph features by answering whether a subset of patterns can lead to an effective classifier is extremely inefficient because of the huge number of pattern combinations. Therefore, in most cases, individual patterns are evaluated for their effectiveness in classification rather than pattern combinations. Let f_p be the frequency of a pattern p in the positive set and f_n be the frequency in the negative set. The effectiveness of p in classification is usually measured by the value of a scoring function $d(f_p, f_n)$. The larger this value is, the more effective p is in classification. Most scoring functions require balanced contributions of f_p and f_n to the value. However, in many applications, such as those considered in this paper, graphs in the positive set shared some (unknown) commonality but the negative set are much more diverse and lacks common patterns. Thus, there may not exist any discriminative patterns in the negative set. In addition, discriminative patterns found in the positive set are of much more interest to users. In this paper, we choose the following function:

$$d(p) = d(f_p, f_n) = log\left(\frac{f_p}{f_n}\right)$$

The rationale for this simple scoring function is that the more frequent p is in the positive set and less frequent p is in the negative set, the more discriminative p is. For example, in Figure 1, the positive frequency of pattern *A-B* is 0.5 and its negative frequency is also 0.5, so the score of *A-B* is $log\left(\frac{0.5}{0.5}\right) = 0$. Additionally, in our experiments this scoring function led to better classification accuracy than G-test score [22] and Delta criterion [20].

This scoring function cannot give a value when f_p or f_n is equal to zero. We solve this problem as follows:

- If f_p of a pattern p is 0, then we do not consider this pattern because we are only interested in patterns found in the positive set
- If f_n of a pattern p is 0, we replace it with a positive value very close to zero.

4.3 A Better Pattern Exploration Order

With a given scoring function, we can rank all patterns by their scores. Unlike LEAP, which looks for patterns with the top-k scores, we want to reorganize the pattern tree to increase the probability that we visit patterns with higher score ranks earlier than those with lower score ranks. The need for a more effective pattern exploration order is due to the fact that most pattern enumeration algorithms tend to visit patterns with similar conformations together since they usually have similar codes. This does not cause any side effect on effectiveness of pattern enumeration, but it has a huge negative impact on finding complementary discriminative patterns because patterns with similar conformations are much more likely to have overlapping supporting sets.

We want to take advantage of the following observation: let p be a pattern in the pattern tree and p' be the parent pattern of p, the score rank of p is correlated with the value of $\Delta(p) = d(p) - d(p')$. For patterns with two nodes, we set their Δ values equal to their scores $d(p)$.

Therefore, when we explore the pattern space, we first enumerate all patterns with 2 nodes as candidates and insert them into a heap structure with the candidate having the highest Δ value at the top. Ties are broken by favoring higher positive frequency and then by CAM code order. Then we always take the pattern at the top of the heap and generate all of its super-patterns with one more edge by performing the CAM extension operation [11]. We insert new patterns into the heap structure. In this way, we are able to visit patterns with high score ranks early and patterns with overlapping supporting sets late. The algorithm is as follows:

1. $P \leftarrow \{all\ ordered\ patterns\ with\ 2\ nodes\}$
2. $p \leftarrow next\ unvisited\ pattern\ in\ P\ with \max \Delta(p)$
3. $while\ (p \neq NULL)$
4. $\quad e \leftarrow \{Extension(p)\}$
5. $\quad P \leftarrow P \cup e$
6. $\quad P \leftarrow P - \{p\}$
7. $\quad p \leftarrow next\ unvisited\ pattern\ in\ P\ with \max \Delta(p)$

5. GENERATING CO-OCCURRENCE RULES

5.1 Classification Rules

DEFINITION 5 (Co-occurrence Rule). Given a positive graph set S_p and a negative graph set S_n, let $P = \{p \mid p$ *is supported by at least one graph in* $S_p\}$ be the set of all subgraphs in S_p, any subset P' of P can form a co-occurrence rule $P' \rightarrow PositiveGraph$. Since all co-occurrence rules we are interested in have the same right hand side, in the following discussion, we omit the right hand side of the rule and use the pattern set at the left hand side to represent the rule.

For example, *{A-B, B-C}* is a co-occurrence rule in Figure 1.

DEFINITION 6 (Satisfying a Rule). Given a graph g and a rule r, g satisfies r iff g supports all patterns in r.

In Figure 1, *{A-B, B-C}* is satisfied by P_1 and P_2.

Let S_p' be the set of all positive graphs that satisfy r and S_n' be the set of all negative graphs satisfying r, the positive frequency of r is denoted as $f_p(r) = \frac{|S_p'|}{|S_p|}$; the negative frequency of r is denoted as $f_n(r) = \frac{|S_n'|}{|S_n|}$. Scoring functions can be applied to a co-occurrence rule $r: d(r) = d\big(f_p(r), f_n(r)\big)$.

The output of our algorithm is a set of co-occurrence rules $R = \{r_0, r_1, \ldots, r_n\}$. It is straightforward to use R as a classifier to classify graphs. Given graph g, if g satisfies at least one rule in R, then it is classified as positive; otherwise g is classified as negative. In addition, because each co-occurrence rule is formed by co-occurred patterns, these pattern co-occurrences can be treated as complex graph features.

5.2 Co-occurrence Rule Generation

Any set of patterns can form a co-occurrence rule, but not all of them have high classification accuracy. Ideally, we want co-occurrence rules consisting of patterns with high frequency in the positive graph set and low frequency in the negative graph set. On one hand, as long as a graph g satisfies a rule, it will be classified as positive, so a strong rule should have low negative frequency; on the other hand, co-occurrence rules are prone to the overfitting problem if each of them is satisfied by only a small portion of the positive set. Therefore, we use two user-specified parameters t_p and t_n to quantify the quality of a rule, where t_p is the minimal positive frequency allowed for a resulting rule and t_n is the maximal negative frequency permitted. The goal of our algorithm is to find a co-occurrence rule set R to maximize the number of graphs that can be classified correctly, where each rule in R has positive frequency no less than t_p and negative frequency no greater than t_n.

This problem can be proved to be equivalent to the set cover problem and is therefore NP complete. It is intractable to find an optimal solution in the enormous pattern space. Therefore, we adopt a greedy approach for rule generation. Let the candidate rule set be R_t and the resulting rule set be R. The algorithm explores the pattern space with the heuristic order in Section 4 and whenever it comes to a new pattern p that has not been processed before, if there exists one positive supporting graph of p that does not satisfy any rule generated so far, the algorithm generates a new candidate co-occurrence rule containing only p and examines the possibility of merging this new rule into existing candidate rules. Given a new pattern p and a candidate rule r_t, $\Delta(p, r_t) = d(r_t \cup \{p\}) - d(p)$. Pattern p is to be inserted into candidate rule r', $r' = argmax_{r_t \in R'}(\Delta(p, r_t))$, $\Delta(p, r_t) \geq 0$. If there are patterns in r' whose supporting sets are supersets of the supporting set of p, then inclusion of p into r' will make these patterns redundant. These patterns will be removed from r' when p is inserted. Then, for either the newly generated rule $\{p\}$ or the updated r', if it has $f_p \geq t_p$ and $f_n \leq t_n$ and it can cover at least one positive graph that does not satisfy any rule in R, it will be removed from R_t and inserted into R. The algorithm terminates either when all patterns are explored or when all positive graphs can satisfy some resulting rules. Although in the worst case the algorithm is still exhaustive, experiments show that it is time efficient in practice.

For example, let t_p = 50% and t_n = 0%, in Figure 1, the frequent subgraphs of 2 nodes in the positive set are *A-B, B-C, D-E, D-G,* and *G-H*. Only positive patterns with frequency no less than t_p need to be considered because (1) as mentioned earlier we only consider positive patterns and (2) the frequency of a rule with patterns less frequent than t_p must be less than t_p as well. We initialize the rule sets to be empty: $R' = \{\}$ and $R = \{\}$.

1. $p \leftarrow next\ pattern$
2. $while\ (p \neq NULL\ and$
   $R\ does\ not\ cover\ all\ positive\ graphs)$
3. $r' \leftarrow argmax_{r_t \in R'}(\Delta(p, r_t))$
4. $if\ (\Delta(p, r_t) \geq 0)$
5. $r' \leftarrow p \cup r'$
6. $R' = \{\{p\}\} \cup R'$
7. $if\ (\{p\}\ covers\ any\ graph\ not\ covered\ by\ R)$
8. $R' = R' - \{\{p\}\}$
9. $R = R \cup \{\{p\}\}$
10. $if\ (r'\ covers\ any\ graph\ not\ covered\ by\ R)$
11. $R' = R' - \{r'\}$
12. $R = R \cup \{r'\}$
13. $p \leftarrow next\ pattern$

According to the pattern exploration order introduced in Section 4, *A-B* is the first pattern to process. For simplicity, the example is designed so that these edges cannot extend to any larger patterns with f_p no less than t_p. A new candidate rule *{A-B}* is added into *R'*. Note that *R'* was empty and thus there does not exist any rule in *R'* to insert *A-B*. Next, *{B-C}* is added into *R'* and *B-C* is added into candidate rule *{A-B}* because $\Delta(\{B\text{-}C\}, \{A\text{-}B\})$ is no less than 0. The modified candidate rule *{A-B, B-C}* have $f_p \geq t_p$ and $f_n \leq t_n$, therefore it is removed from *R'* and added into *R*. Next, *D-E* is at the top of the heap, but there is no need to consider it because both of its supporting graphs, P_1 and P_2, satisfy rule *{A-B, B-C}* and therefore considering *D-E* cannot lead to a better classifier. Then, following a similar procedure, we can generate rule *{D-G, G-H}* and add it into *R*. Now the algorithm terminates because: 1) the heap structure for candidate patterns is empty and 2) *{A-B, B-C}* and *{D-G, G-H}* are sufficient to cover all graphs in the positive set. For each step, the initial status of *R'*, *R*, the pattern at the heap top and the set of positive graphs not yet covered by *R* are shown in Figure 3.

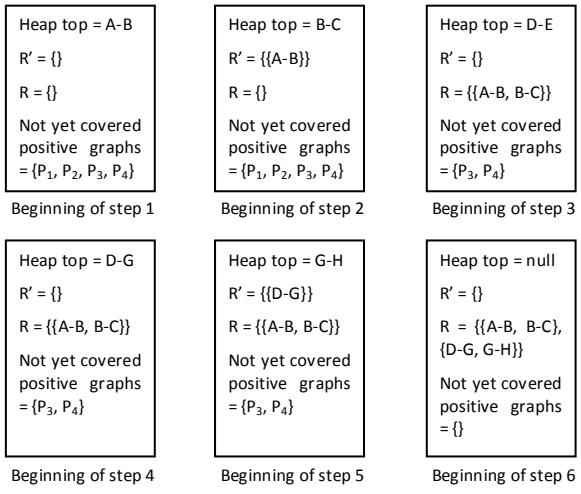

Figure 3. An example of rule generation

6. EXPERIMENTS

The algorithm is implemented in C++ and compiled with g++. The experiments are performed on a PC with 2.00 GHz dual core and 3 GB memory. We use protein datasets and small molecule datasets in our experiments. The protein datasets consist of protein structures from Protein Data Bank [1] classified by SCOP [2] (Structural Classification of Proteins), which organizes protein structures into a 4-level hierarchy: class, fold, superfamily and family, from high level to low level. The lower the level is, the more details are considered and thus more useful to the scientists. We select protein structures in the same families as positive sets. In order to remove redundancy and possible bias in graph sets, we only use proteins with pairwise sequence identity less than 90% from the culled PDB list created by Dunbrack Lab [3]. Table 2 shows the 6 protein families used in experiments. We then randomly select 256 other proteins (i.e., not members of the 6 families) from the culled PDB list as a common negative set. To generate a protein graph, each graph node denotes an amino acid, whose location is represented by the location of its alpha carbon. We perform 3-D Almost Delaunay Tessellation [3] on locations of all alpha carbons in the protein to generate the edges. Nodes are labeled with their amino acid type and edges are labeled with the distance between the alpha carbons. We only consider edges shorter than 11.5 angstroms because amino acids have little long-distance interaction. On average, each protein graph has 250 nodes and 1600 edges. The small molecule datasets consist of chemical compound structures from PubChem[4] classified by their biological activities, listed in Table 3. Each compound can be either active or inactive (we do not consider inconclusive and discrepant records) in a bioassay. For each bioassay, we randomly select 400 active compounds as the positive set and 1600 inactive compounds (the sample size is similar to what is used in the original report of LEAP+SVM) as the negative set for performance evaluation. The graph representation of compounds is straightforward. Each atom is represented by a graph node labeled with the atom type and each chemical bond is represented by a graph edge labeled with the bond type. On average, each compound graph has 47 nodes and 49 edges.

Table 2. List of selected protein families

SCOP_ID	Family name	Number of selected proteins
56437	C-type lectin domains	38
48623	Vertebrate phospholipase A2	29
48942	C1 set domains (antibody constant domain like)	38
52592	G proteins	33
88854	Protein kinases, catalytic subunit	41
56251	Proteasome subunits	35

[1] http://www.rcsb.org/pdb/

[2] http://scop.mrc-lmb.cam.ac.uk/scop/

[3] http://dunbrack.fccc.edu/PISCES.php

[4] http://pubchem.ncbi.nlm.nih.gov

Table 3. List of selected bioassays

Assay ID	Tumor Description	Total Number of Actives	Total Number of Inactives
1	Non-Small Cell Lung	2047	38410
41	Prostate	1568	25967
47	Central Nerv Sys	2018	38350
83	Breast	2287	25510
109	Ovarian	2072	38551
145	Renal	1948	38157

We evaluate the classification power using the following three measures:

$$Sensitivity = \frac{number\ of\ true\ positives}{numer\ of\ positives}$$

$$Specificity = \frac{number\ of\ true\ negatives}{number\ of\ negatives}$$

$$Normalized\ accuracy = \frac{sensitivity + specificity}{2}$$

6.1 Comparison with Other Methods

We compare our method COM with two alternative approaches: (1) LEAP+SVM [22] and (2) gPLS [19]. LEAP+SVM invokes LEAP iteratively until every training example can be represented by some discovered subgraphs and then takes the discriminative subgraphs found by LEAP as features to train a Support Vector Machine (SVM) [21] classifier. LIBSVM [5] is used in the experiments. We use a linear kernel with parameter C selected from $\{2^{-11}, 2^{-10}, \ldots, 2^{10}, 2^{11}\}$ by cross-validation on the training set only. We use 5-fold cross validation in our experiments. We could not furnish a thorough comparison with CORK because the current release of CORK[5] entails very long execution time. It takes hours to days (if not longer) to process a small dataset. As a result, CORK is not able to finish its execution within reasonable time except for one compound dataset (bioassay ID 1). We will show its result in Section 6.1.2.

6.1.1 Protein datasets

For protein datasets, we use t_p=30%, t_n=0% for COM and leap length = 0.1 for LEAP+SVM[6]. For gPLS, we use frequency threshold = 30% and exhaustively examine all combinations from m = {2, 4, 8, 16} and k = {2, 4, 8, 16} where m is number of iterations and k is number of patterns obtained per search. This candidate parameter set is adapted for protein datasets. For each dataset, we report the best test accuracy among all settings[7]. In addition, we need to set a subgraph size threshold = 3 for gPLS because it runs out of memory for higher subgraph size threshold. Fortunately, this small size threshold has little impact on classification accuracy as most subgraph patterns found by COM and LEAP+SVM have size = 3 or 4.

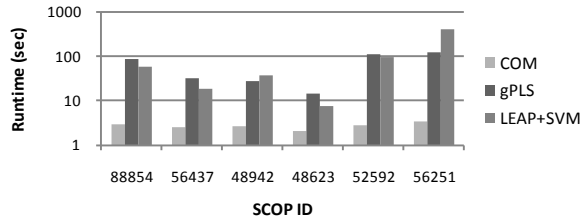

Figure 4. Runtime comparison (protein datasets): COM vs. gPLS vs. LEAP+SVM

Figure 4 compares the time efficiency of COM, gPLS[8], and LEAP+SVM. It shows that gPLS and LEAP+SVM have similar runtime while COM is an order of magnitude faster than them for most protein datasets. This is because COM finds discriminative features sooner than LEAP and gPLS. Usually the early stage of pattern exploration only enumerates weak pattern features. COM can generate strong features from weak features in the early exploration by taking advantage of co-occurrence information while LEAP and gPLS only use these weak features to refine further mining. The time difference is also due to COM's heuristic exploration order, greedy strategy and unnecessity of repeated executions.

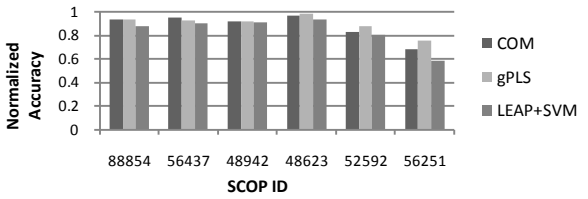

Figure 5. Normalized accuracy comparison (protein datasets): COM vs. gPLS vs. LEAP+SVM

Figure 5 compares the normalized accuracy of the classifiers generated by COM, gPLS and LEAP+SVM. COM outperforms LEAP+SVM for all 6 protein datasets, although for most datasets the margin may be small. Compared with gPLS, COM has competitive result for the first 4 datasets, 5% lower accuracy for dataset 52592 and 7% lower accuracy for dataset 56251.

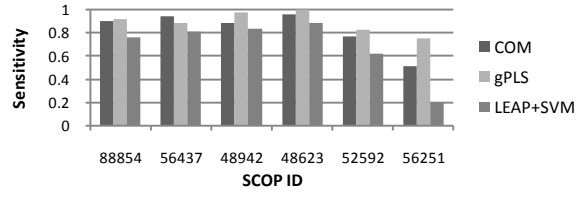

Figure 6. Sensitivity comparison (protein datasets): COM vs. gPLS vs. LEAP+SVM

[5] http://www.dbs.ifi.lmu.de/~thoma/pub/sdm09/

[6] Setting leap length = 0.1 significantly improved LEAP's efficiency with only minor impact to the resulting pattern's score. We also experimented with leap length = 0.05 which delivered the same accuracy but required longer runtime.

[7] Please note that, for COM and LEAP+SVM, we use the same parameters for all protein datasets. We perform exhaustive search and report the best result for gPLS because this is how gPLS was evaluated originally in [23].

[8] The time for searching for optimal parameter setting is NOT included.

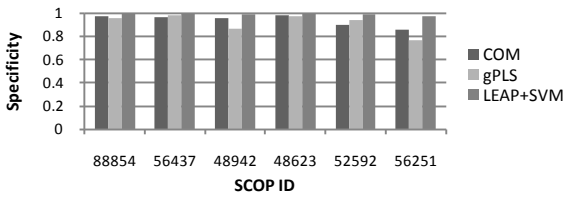

Figure 7. Specificity comparison (protein datasets): COM vs. gPLS vs. LEAP+SVM

Figure 6 and Figure 7 decompose the normalized accuracy into sensitivity and specificity to provide more insight. Compared with LEAP+SVM, COM has higher sensitivity at the cost of slightly lower specificity. The substantial difference in sensitivity between COM and LEAP+SVM is because of the iterative feature selection method that LEAP relies on. As soon as a positive graph g is covered by a certain number of features, g becomes much less important or even removed from the positive set completely. The advantages of iterative feature selection are (1) positive graphs that are relatively harder to discriminate will be emphasized and (2) the positive set may shrink as the algorithm runs, which reduces the runtime of the next execution of LEAP. However, it is prone to the overfitting problem when most graphs in the dataset are removed. This is because when most graphs can be covered by discriminative patterns and removed from the dataset, the number of remaining graphs is so small that LEAP tends to discover very large subgraphs. An extreme example is that when there is only one positive graph left against an enormous negative set, then the positive graph itself is the optimal discriminative pattern which is, however, useless in classifying other positive graphs. In contrast, COM does not remove graphs from the positive set and thus will always find features covering a large number of graphs. As a result, COM is less likely to misclassify positive graphs. As for specificity, it is understandable that COM has slightly lower specificity because, as long as a graph satisfies one rule, it will be classified as positive. Therefore, COM tends to correctly classify more positive graphs and misclassify a few negative graphs as positive graphs (yet only slightly lower than LEAP+SVM). This explains why a simple classifier generated by COM can outperform the sophisticated SVM.

Comparing COM and gPLS, we find that gPLS generally produces classifiers with competitive accuracy and better sensitivity (1-7% for 5 protein families and 25% higher for family 56251) at the cost of lower specificity in most cases, but it requires exhaustive search for the best parameter setting. While COM and LEAP+SVM use the same parameter setting for all protein families, gPLS requires different parameter settings for different training sets of a family to generate classifiers with high classification power on the corresponding test sets. We take protein family 48942 as an example. As we use 5-fold cross validation, we have 5 different pairs of training set and test set for protein family 48942. Below we list the optimal parameter setting for each pair and the average normalized accuracy for each pair if its optimal parameter setting is used for all 5 training-test pairs.

Table 4. List of optimal parameter settings for gPLS and their average normalized accuracy for 5 training-test pairs of family 48942

Training-test pairs	Optimal parameter setting (m, k)	Average normalized accuracy using (m, k)
1	(8, 4)	0.8377
2	(2, 2)	0.8377
3	(2, 4)	0.6847
4	(4, 16)	0.6847
5	(2, 8)	0.6963

The normalized accuracy reported in Figure 5 is 92.11% as we use the optimal parameter setting for each training-test pair. We can see from Table 4, if we use one of the optimal parameters for all 5 training-test pairs, then we will have at least 8% drop in normalized accuracy. In fact, for family 48942, m=16, k=8 leads to the best average normalized accuracy 86% if all 5 pairs use the same parameter setting, still 6% lower than the 92.11% which is a result of using the best parameter setting for each training-test pair. In addition, the relationship between the parameters and classification result is obscure, making it very difficult to tune parameters.

6.1.2 Compound datasets

For compound datasets, we only compare COM and LEAP+SVM because gPLS needs exhaustive parameter search with a much larger search space for the compound datasets and it runs out of memory when the subgraph size threshold is larger than 9. We use t_p = 1%, t_n = 0.4% for COM because the datasets are much larger and more diverse than the protein datasets. Leap length for LEAP+SVM is still set to 0.1. We also set a subgraph pattern size threshold s = 5 for COM. The size of patterns found by LEAP is typically 10-20 and the number of patterns in a co-occurrence rule is usually 2-4.

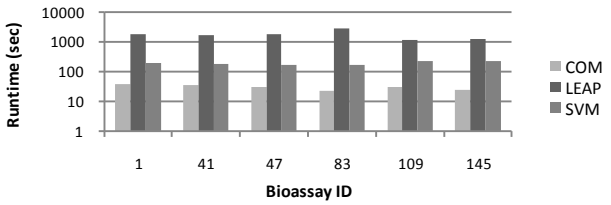

Figure 8. Runtime comparison (compound datasets): COM vs. LEAP vs. SVM

Figure 8 compares the runtime of COM and LEAP+SVM. Here we divide the runtime of LEAP+SVM into runtime of LEAP and runtime of SVM because, for the compound datasets, the computational cost of SVM is no longer trivial compared with the computational cost of COM. It shows that COM is 40-120 times faster than LEAP. In fact, even the SVM classifier building step takes much longer time than COM because the compound datasets have a large number of graphs and subgraph features.

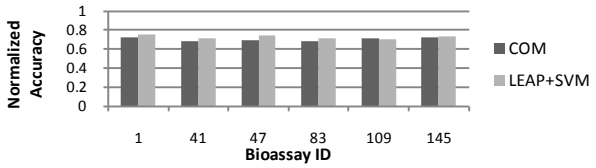

Figure 9. Normalized accuracy comparison (compound datasets): COM vs. LEAP+SVM

Although COM uses much less time to generate classifiers, its classifiers are still very competitive to the classifiers by LEAP+SVM in terms of normalized accuracy. Figure 9 shows the normalized accuracy of these two approaches. LEAP+SVM has slightly higher normalized accuracy than COM for 5 compound datasets, but the difference is merely 2.45% on average and is always less than 5%.

Again we decompose the normalized accuracy into sensitivity and specificity, shown in Figure 10 and Figure 11 respectively. Figure 10 demonstrates that COM has more than 10% higher sensitivity than LEAP+SVM and its disadvantage in specificity (15% lower) is also obvious as shown in Figure 11, which is similar to what we have from the protein datasets except that the difference in specificity between COM and LEAP+SVM is larger for the compound datasets than that for the protein datasets. The larger difference in specificity is a result of the relatively high negative frequency threshold t_n (0.4%) used for compound datasets. If we further lower t_n and have higher specificity, then the sensitivity drops because the classifier overfits the training set and the normalized accuracy is barely affected. Figure 12 compares the average normalized accuracy, sensitivity and specificity between using t_n=0.4%, t_n=0.25% and t_n=0.1%.

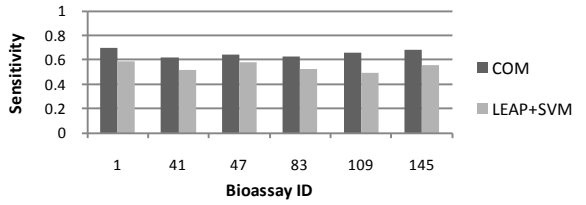

Figure 10. Sensitivity comparison (compound datasets): COM vs. LEAP+SVM

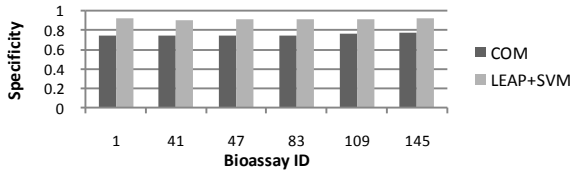

Figure 11. Specificity comparison (compound datasets): COM vs. LEAP+SVM

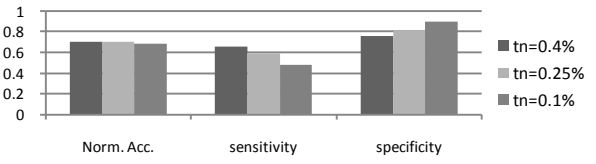

Figure 12. Comparison of average normalized accuracy, sensitivity, specificity between using t_n = 0.4%, t_n = 0.25% and t_n = 0.1% for COM (t_p = 1%, subgraph size threshold = 5, compound datasets)

We also run CORK on all of our datasets and are only able to get its result on one dataset (bioassay ID=1) with frequency threshold=10%, due to its long runtime (more than 20 hours). Additionally, because LIBSVM is too slow to train a classifier using subgraph features from CORK, we use Random Forests [4] instead. We generate 100 trees and examine all possible values for the number of features used to split a node. The best result is comparable to COM, with normalized accuracy=71.6% (72.8% for COM), sensitivity=49% (64%) and specificity=94.2% (81.5%). Clearly, COM is far more efficient than CORK.

6.2 COM Performance Analysis

We first study the effectiveness of the pattern exploration order used in COM with t_p = 0.3 and t_n = 0 on the protein datasets. In COM, patterns are explored in the descending order of ancestors' Δ values (illustrated in Section 4.3). An alternative is to explore patterns in the descending order of their ancestors' scores (by replacing Δ in Section 4.3 with score value d). Figure 13 shows that exploration in the order of Δ values is more efficient than in the order of scores. The runtime difference seems marginal for some protein families mainly because standard operations such as collecting occurrences of frequent edges dominate the overall runtime for those families. We observe that using score value d causes patterns with similar supporting sets to be explored together, which is not ideal to the generation of rules. Let p and q be two patterns complementary to each other and the score of p is higher than that of q. If we explore pattern space guided by ancestors' scores, all superpatterns (or supergraphs) of p are examined before q because they have higher scores than q. However, they cannot complement pattern p because their supporting sets are subsets of p's supporting set. However, if we use ancestors' Δ values, there is a much better chance that q is explored before many of p's superpatterns.

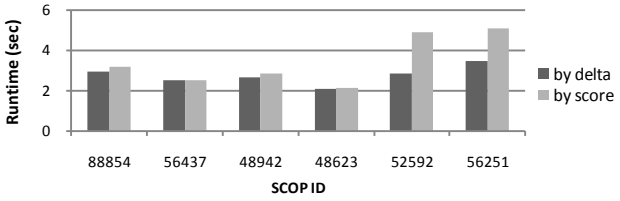

Figure 13. COM runtime comparison between different exploration orders (protein datasets): by delta value vs. by score

Now we study the relationship between performance and the two parameters t_p and t_n. We fix t_p and adjust t_n for the compound datasets and fix t_n and adjust t_p for the protein datasets. The difference in average runtime using different parameters is marginal. We compare the average normalized accuracy, sensitivity and specificity in Figure 12 and Figure 14. Figure 12 shows that when t_n decreases, the specificity increases accordingly and the normalized accuracy remains the same while the sensitivity drops. Figure 14 demonstrates that, when t_p decreases, the sensitivity goes down but the specificity is almost unaffected. Therefore, these two parameters can be used by users to adjust the trade-off between the sensitivity and specificity.

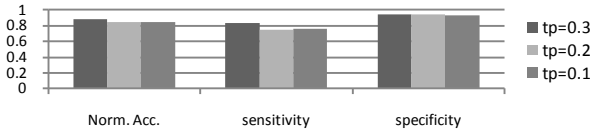

Figure 14. Comparison of average normalized accuracy, sensitivity, specificity between using $t_p = 0.3$, $t_p = 0.2$ and $t_p = 0.1$ for COM ($t_n = 0\%$, no size limit, protein datasets)

7. CONCLUSIONS

In this paper, we investigate the problem of using subgraph patterns for graph classification and propose the algorithm COM to meet the pressing need for efficient graph classification methods. By using an efficient pattern exploration order and grouping patterns into co-occurrence rules, COM is easy to implement and understand. Even though we adopt FFSM as the basic subgraph mining routine in COM, the pattern exploration order and co-occurrence rule generation routine can be integrated with any other subgraph mining algorithm. In spite of its seemingly simple classification model, experiments show that COM is time-efficient and delivers high classification accuracy. Another advantage of COM is the high interpretability of its classifiers. In the future, we plan to incorporate the connectivity between subgraph patterns into the classification model.

8. REFERENCES

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

[2] D. Bandyopadhyay, J. Huan, J. Liu, J. Prins, J. Snoeyink, W.Wang, and A. Tropsha. Structure-based function inference using protein family-specific fingerprints, *Protein Science*, vol. 15, pp. 1537-1543, 2006.

[3] D. Bandyopadhyay and J. Snoeyink. "Almost Delaunay Simplices: Nearest Neighbor Relations for Imprecise Points". *ACM-SIAM Symposium On Discrete Algorithms (SODA 2004),* New Orleans, Jan 11-13, 2004, pages 403-412.

[4] L. Breiman. "Random Forests". *Machine Learning* 45 (1): 5-32, 2001.

[5] C. Chang and C. Lin. *LIBSVM: a library for support vector machines, 2001.* Software available at: http://www.csie.ntu.edu.tw/~cjlin/libsvm/

[6] C. Chen, C. X. Lin, X. Yan, and J. Han, On Effective Presentation of Graph Patterns: A Structural Representative Approach, in *Proc. 2008 ACM Conf. on Information and Knowledge Management (CIKM'08*), Napa Valley, CA, Oct. 2008.

[7] M. Deshpande, M. Kuramochi, N. Wale, and G. Karypis. Frequent Sub-structure Based Approaches for Classifying Chemical Compounds. *IEEE Trans. Knowl. Data Eng. 17(8): 1036-1050*, *2005*.

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

[9] C. Helma, T. Cramer, S. Kramer, and L.D. Raedt. Data mining and machine learning techniques for the identification of mutagenicity inducing substructures and structure activity relationships of noncongeneric compounds. J. Chem. Inf. *Comput. Sci*., 44:1402-1411, 2004.

[10] J. Huan, W. Wang, D. Bandyopadhyay, J. Snoeyink, J. Prins, and A. Tropsha. Mining spatial motifs from protein structure graphs, *Proceedings of the 8th Annual International Conference on Research in Computational Molecular Biology (RECOMB)*, pp. 308-315, 2004.

[11] J. Huan, W. Wang, and J. Prins. Efficient mining of frequent subgraph in the presence of isomorphism, *Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM)*, pp. 549-552, 2003.

[12] A. Inokuchi, T. Washio, and H. Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In *Proc. of 2000 European Symp. Principle of Data Mining and Knowledge Discovery*, pages 13-23, 2000.

[13] M. Kuramochi and G. Karypis. Frequent subgraph discovery. In *Proc. of ICDM*, pages 313-320, 2001.

[14] T. Kudo, E. Maeda, and Y. Matsumoto. An application of boosting to graph classification. In *Advances in Neural Information Processing Systems* 17, pages 729–736. MIT Press, 2005.

[15] J. Kazius, S. Nijssen, J. Kok, and T. Back A.P. Ijzerman. Substructure mining using elaborate chemical representation. *J. Chem. Inf. Model.*, 46:597-605, 2006.

[16] S. Nowozin, K. Tsuda, T. Uno, T. Kudo, and G. Bakir. Weighted substructure mining for image analysis. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE Computer Society, 2007.

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

[18] H. Saigo, T. Kadowaki, and K. Tsuda. A linear programming approach for molecular QSAR analysis. In *International Workshop on Mining and Learning with Graphs (MLG),* pages 85–96, 2006.

[19] H. Saigo, N. Kraemer and K. Tsuda: Partial Least Squares Regression for Graph Mining, In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD2008)*, 578-586, 2008.

[20] M. Thoma, H. Cheng, A. Gretton, J. Han, H. Kriegel, A. Smola, L. Song, P. Yu, X. Yan, K. Borgwardt. "Near-optimal supervised feature selection among frequent subgraphs", In *SDM 2009*, Sparks, Nevada, USA.

[21] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, 1995.

[22] X. Yan, H. Cheng, J. Han, and P. S. Yu. Mining significant graph patterns by leap search. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 433–444, 2008.

[23] X. Yan and J. Han. gSpan: graph-based substructure pattern mining. In *Proceedings of the 2002 IEEE International Conference on Data Mining*, pages 721–724. IEEE Computer Society, 2002.

[24] F. Zhu, X. Yan, J. Han, and P. S. Yu, gPrune: A Constraint Pushing Framework for Graph Pattern Mining, in *Proc. 2007 Pacific-Asia Conf. on Knowledge Discovery and Data Mining (PAKDD'07)*, Nanjing, China, May 2007.