GraphSig: A Scalable Approach to Mining Significant Subgraphs in Large Graph Databases

Sayan Ranu [1], Ambuj K. Singh [2]

Department of Computer Science
University of California, Santa Barbara
Santa Barbara, CA 93106, USA
[1]sayan@cs.ucsb.edu
[2]ambuj@cs.ucsb.edu

***Abstract*— Graphs are being increasingly used to model a wide range of scientific data. Such widespread usage of graphs has generated considerable interest in mining patterns from graph databases. While an array of techniques exists to mine frequent patterns, we still lack a scalable approach to mine statistically significant patterns, specifically patterns with low p-values, that occur at low frequencies. We propose a highly scalable technique, called GraphSig, to mine significant subgraphs from large graph databases. We convert each graph into a set of feature vectors where each vector represents a region within the graph. Domain knowledge is used to select a meaningful feature set. Prior probabilities of features are computed empirically to evaluate statistical significance of patterns in the feature space. Following analysis in the feature space, only a small portion of the exponential search space is accessed for further analysis. This enables the use of existing frequent subgraph mining techniques to mine significant patterns in a scalable manner even when they are infrequent. Extensive experiments are carried out on the proposed techniques, and empirical results demonstrate that GraphSig is effective and efficient for mining significant patterns. To further demonstrate the power of significant patterns, we develop a classifier using patterns mined by GraphSig. Experimental results show that the proposed classifier achieves superior performance, both in terms of quality and computation cost, over state-of-the-art classifiers.**

I. Introduction

Recent technological and scientific advances have generated large amounts of data that describe and model phenomena in terms of graphs. Graphs have been widely used to model chemical compounds [1], social networks [2], multimedia [3], and protein interaction networks [4]. Given such widespread use of graphs to model scientific data, there is a need for techniques that mine and analyze graphs. Mining graph patterns can help understand the inherent data and domain characteristics. In drug discovery for example, graph mining can reveal the conserved substructures in an active set of chemical compounds.

Frequent subgraph mining ([5], [6], [7], [8], [9], [10], [11], [12], [13]) is a well studied problem in the area of graph mining. Typically, in these techniques a frequency threshold is supplied by the user and all subgraphs above the frequency threshold are returned. Unfortunately, in spite of recent progress on frequent subgraph mining, no scalable technique exists to mine *significant* subgraphs from a graph database when the frequency of the subgraph is low. It is not enough to just find the frequent subgraphs as they may not provide the best characterization of the dataset. Significant subgraphs have the potential to unearth properties where the data deviates from the expected.

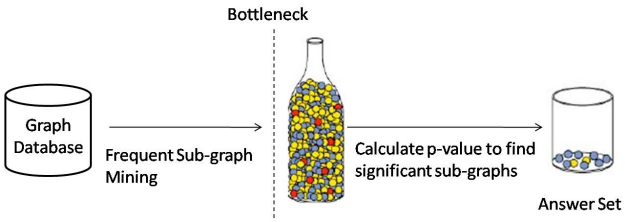


Fig. 1. A straightforward approach to mining significant subgraphs

Fig. 1 presents the straightforward approach to mine significant subgraphs. It is a two step process. First, a frequent subgraph mining technique is used to mine all frequent subgraphs above a frequency threshold θ. Next, the p-value (or some other significance measure) of each subgraph is computed and all subgraphs with a p-value below a user-specified threshold are returned.

The bottleneck lies in the first step where θ can be very low and therefore, generate an exponential search space. The behavior is reflected in Fig. 2 which plots the running time of gSpan [5] and FSG [6] against frequency. As can be seen, the running time grows exponentially with decreasing frequency due to the inevitable explosion in graph search space. Other frequent subgraph mining techniques have a similar behavior and their inability to overcome the scalability bottleneck forms our fundamental motivation.

A. Our approach

In this paper, we propose a scalable technique, called *GraphSig*, to mine significant subgraphs from large graph databases. Moreover, to demonstrate the immense promise of significant patterns, we design a classifier built on them and demonstrate its application in graph classification.

First, we define the preliminary concepts of *frequent subgraphs* and *significance of a subgraph*.

1084-4627/09 $25.00 © 2009 IEEE
DOI 10.1109/ICDE.2009.133

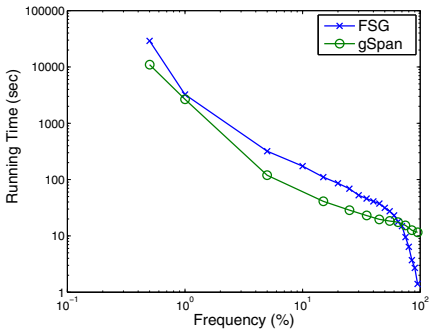


Fig. 2. Scalability of gSpan and FSG against frequency

***Definition** 1:* FREQUENT SUBGRAPH. *Given a graph dataset* $\mathbb{D}=\{G_1, ..., G_n\}$ *and a frequency threshold* θ*, a subgraph* g *with an observed support* μ_0 *is frequent if and only if* $\mu_0 \geq \frac{\theta|\mathbb{D}|}{100}$.

***Definition** 2:* STATISTICAL SIGNIFICANCE. *The statistical significance (or p-value) of a subgraph* g *with an observed support* μ_0 *is defined as the probability that it occurs in a random database with a support* μ*, where* $\mu \geq \mu_0$.

The goal of our work is to mine subgraphs with p-value below a user-specified threshold η. Mathematically, we want to find the answer set $\mathbb{A}$ such that

$$\mathbb{A} = \{g | p\text{-}value(g) \leq \eta, g \subseteq G, G \in \mathbb{D}\} \quad (1)$$

The challenge lies in devising a scalable technique that mines subgraphs in the presence of a low frequency threshold. The running time of existing techniques can be characterized as an exponential function of frequency. Since significant subgraphs exist at all frequencies, we ask: *can we avoid using a frequent subgraph mining technique with low frequency threshold and yet mine significant subgraphs?*

We address the challenge by partitioning graphs into sets such that all graphs in a set are likely to contain a common significant subgraph at a high frequency. Given this knowledge, frequent subgraph mining can be performed with a high threshold on each set to extract the significant subgraph. To achieve results in a timely fashion, we use domain knowledge to convert graphs into feature space for analysis. Significance of a graph is approximated by the significance of its feature vector representation. We make the assumption that a low p-value in the feature space will correspond to a low p-value in the graph space. As shown by the experimental results, the assumption is indeed true.

Fig. 3 outlines our approach. In the first phase, we slide a window across each graph in the dataset. A window is centered on a node and captures the neighborhood around it. This is repeated for all nodes in the graph and then for all graphs in the database. Thus, we obtain a set of feature vectors for each graph, where a feature vector represents the subgraph inside a window. Although there is some loss of structural information,

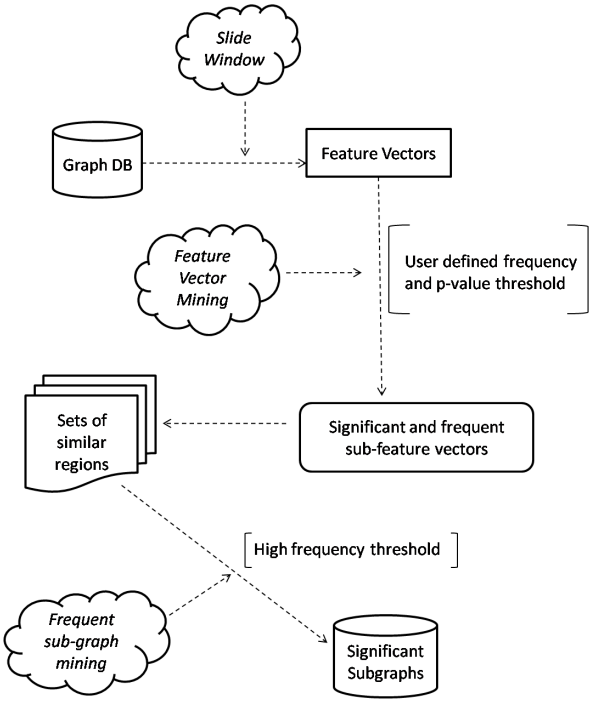


Fig. 3. Outline of GraphSig

the distribution of features within a window is captured. Next, we mine the generated feature vectors to identify *closed sub-feature vectors* which are significant. Each sub-feature vector could potentially describe a specific significant subgraph. To take advantage of this property, we group all regions likely to contain a subgraph described by a significant sub-feature vector into sets. Specifically, a set of graphs is formed for each significant sub-feature vector. Once grouped into sets, we mine each set for frequent subgraph with a high frequency threshold. We, thus, go from graph space to feature space to calculate approximate significance of graphs. Following the analysis, we return to graph space and extract the significant subgraphs.

With our approach, we save time on two accounts. First, due to feature space analysis of significance, we avoid the need to generate a large number of random graph databases and calculate the frequency distribution of a query graph. Secondly, due to grouping of similar graphs into sets, frequent subgraph mining can be performed on them with a frequency threshold that is able to mine significant patterns scalably.

In the second half of the paper, we explore an application of mining significant subgraphs. We design an algorithm to classify graphs based on the subgraphs mined by GraphSig. First, the proposed classifier mines the set of significant subgraphs from the positive and negative training set and then treats the presence of a significant subgraph in a query graph as an identifier of its class.

We validate the efficiency and effectiveness of our technique through extensive experiments on twelve chemical com-

pound datasets. The experiments reveal that the most frequent subgraph may not necessarily be significant. On the other hand, subgraphs from a class of active compounds turn out to be highly significant. Moreover, a number of significant subgraphs have frequencies well below 1%, which validates our claim that specialized techniques are necessary to mine significant subgraphs at low frequency thresholds. We also evaluate our proposed classifier against the most up-to-date kernel-based and feature-based classifiers. The experiments reveal that our classifier outperforms both other techniques in quality and computation cost.

The main contributions of our work are as follows:

1) We propose a novel mining framework to mine significant subgraphs from large graph databases. Efficient methods are developed to remove the scalability bottleneck of operating at low frequencies in frequent subgraph mining techniques. The significance of a subgraph is measured by its p-value. To the best of our knowledge, no prior work has been done on mining graphs at low frequencies.
2) We develop a classifier built on significant subgraphs which demonstrates superior performance over recent graph classifiers.

The remainder of the paper is organized as follows. Section II discusses how we simulate the process of sliding a window across a graph and its subsequent transformation to feature space. Section III presents the model used to evaluate the p-value of a feature vector. Section IV discusses the use of feature vector mining to identify interesting regions in graphs and presents the GraphSig algorithm. Section V presents the classification algorithm based on the significant subgraphs. In section VI, we present the experimental evaluation of our techniques. Section VII discusses related work. We conclude with a brief discussion in Section VIII.

II. Sliding Window across Graphs

In this section, we describe how we simulate the process of sliding a window across a graph. Each graph is converted to a set of feature vectors, where a feature vector represents a window region. The size of the window is defined in terms of radius, such that all nodes within the window-radius from the target node (node on which the window is centered) are considered within the window. First, we discuss how to select features. We specifically concentrate on selecting features for chemical compounds, although we discuss a greedy approach to select features in a general setting. Next, we describe our method to capture the feature vector representation of the subgraph within a window.

A. Feature Selection

A graph can be viewed as a collection of features. The appropriate set of features is domain-dependent and can be chosen based on the demands of the application. Given a feature set $\mathbb{F}$=$\{f_1, ..., f_n\}$, a graph G can be viewed as a feature vector $\underline{x} = [x_1, ..., x_n]$, where $x_i \neq 0$ if $f_i \subseteq$ G, otherwise $x_i = 0$. $\mathbb{F}$ can consist all types of nodes, edges, or small subgraphs. Each approach has its limitations. Choosing edge types or small subgraphs as features will preserve some structural information, however, the feature set will attain enormous sizes due to the combinatorial explosion. On the other hand, choosing just node-types as features will lose all structural information.

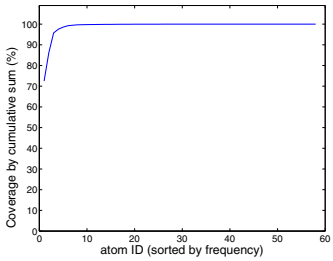


Fig. 4. Cumulative percentage coverage of atoms in the AIDS antiviral dataset

Generally, larger the feature in size, more complicated is the process of choosing the optimal set of features. For example, one can choose top-k frequent subgraphs as features. However, if two frequent features have a large overlapping region, selecting both may not be the best choice. Selecting the optimal set of features is therefore a difficult problem. If a similarity function can be defined on the features, a *greedy approach* could be adopted [14]. One can first enumerate all feature candidates. Then, choose features one at a time, such that f_k is the best choice based on its importance (eg. frequency or size) and similarity (eg. structural overlap) to the $k-1$ features already in $\mathbb{F}$. Mathematically, it can be formulated as,

$$f_k = \arg\max_f \{w_1 imp(f_k) - \frac{w_2}{k-1} \sum_{i=1}^{k-1} sim(f_i, f_k)\} \quad (2)$$

where w_1 and w_2 are weighing factors, $imp(f_k)$ is importance of f_k, and $sim(f_i, f_k)$ is the similarity between f_i and f_k. Both $imp(f_k)$ and $sim(f_i, f_k)$, and the weights can be domain-dependent.

B. Feature Selection for Chemical Compounds

Chemical compounds are often represented as graphs where nodes represent atoms and edges represent bonds. The bond types are preserved as edge labels. Fig. 5 shows the graph representation of benzene. To select features, we first investigate the frequency distribution of atoms in a chemical compound database. Fig. 4 is a plot of the cumulative sum of

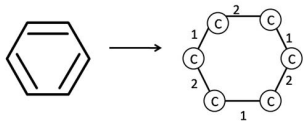


Fig. 5. Conversion of benzene to graph

the atoms in the NCI/NIH AIDS database. The x-axis contains the atom ids, such that the leftmost atom is the most frequent and the rightmost atom is least frequent. The y-axis plots the cumulative sum of percentage coverage. As can be seen, although 58 different atom-types exist in the database, 99% of the composition is from the top 5 most frequent atoms. This is actually a common phenomenon in chemical compound databases. Inspired by this property, we include the edge-types between top 5 atoms in $\mathbb{F}$, since it ensures we retain a considerable amount of structural information and, at the same time, keep the vector size manageable. We also include all atom-types in the feature set. An atom-based feature is updated only when the edge-type traversed is not in $\mathbb{F}$.

C. Transforming graphs into feature vectors

We perform *Random Walk with Restarts (abbr. RWR)* on each node in a graph to simulate sliding a window across it. RWR simulates the trajectory of a random walker that starts from the target node and jumps from one node to a neighbor. Each neighbor has an equal probability of becoming the new station of the walker. At each jump, we update the feature traversed, which can either be an edge-type or atom-type. At the same time, we do not want the walker to step out of the window. We thus attach a restart probability α to bring the walker back to the starting node within approximately $\frac{1}{\alpha}$ jumps. For instance, if we set $\alpha = 0.25$, in the average case, after every four jumps, the walker comes back to the starting node.

We iterate the random walk till the feature distribution converges. As a result, RWR produces a continuous distribution of features for each node where a feature value lies in the range [0, 1]. To cut down on the infinitely possible values for a feature, the features are discretized into 10 bins. For example, a feature value of 0.07 will be discretized as 1, and a value of 0.34 will be discretized as 3. RWR can therefore be visualized as placing a window at each node of a graph and capturing a feature vector representation of the subgraph within it. As a result, a graph of *m* nodes is represented by *m* feature vectors. Hereon, we denote $vector(n_i)$ for the vector produced from RWR on node n_i and $label(v_i)$ to denote the label of the node corresponding to v_i.

RWR inherently takes proximity of features into account and, therefore, preserves more structural information rather than simply counting occurrence of features inside the window. For example, a feature close to a source node will be visited more often than a feature on the boundary of the window. As a result, more structural information is preserved than a simple count. The property is reflected in Fig. 6 and Table II (discussed later).

III. Calculating p-value of a feature vector

In this section, we explain the process of calculating p-value of a feature vector [15]. First, we model the occurrence of a feature vector $\underline{x}$ in a vector generated from a random graph. The frequency distribution of a vector is generated using the prior probabilities of features obtained empirically. For example, given the vectors in Table I, the prior probabilities of $P(a\text{-}b \geq 2) = \frac{1}{4}$ and $P(b\text{-}b \geq 1) = \frac{2}{4}$. The p-value is then calculated by comparing the observed support of $\underline{x}$ with its expected support.

TABLE I
Sample Feature Vector Database

Vector	a-b	a-c	b-b	b-c
v_1	1	0	0	2
v_2	1	1	0	2
v_3	2	0	1	2
v_4	1	0	1	0

A. Probability of $\underline{x}$ occurring in a random vector

***Definition** 3:* Sub-feature Vector. *For two feature vectors $\underline{x} = [x_1, ..., x_n]$ and $\underline{y} = [y_1, ..., y_n]$, $\underline{x}$ is a sub-feature vector of $\underline{y}$ if and only if $x_i \leq y_i$ for i=1,...,n. The relation is denoted as $\underline{x} \subseteq \underline{y}$. At the same time, $\underline{y}$ is a super-feature vector of $\underline{x}$.*
Example: $v_4 \subseteq v_3$ *whereas* $v_2 \not\subseteq v_3$.

The probability of $\underline{x}$ occurring in a random feature vector $\underline{y} = [y_1, ..., y_n]$ can be expressed as a joint probability

$$P(\underline{x}) = P(y_1 \geq x_1, ..., y_n \geq x_n) \tag{3}$$

where each event is the probability of a feature in the random vector $\underline{y}$ having a higher or equal value than the same feature in $\underline{x}$.

The transformation from graphs to feature space attempts to keep the features orthogonal. We thus assume independence of the features. This is also a simplifying assumption that keeps our computations tractable. Moreover, we always return to the graph space to verify all our predictions. As a result, Eqn. 3 can be expressed as a product of the individual probabilities where,

$$P(\underline{x}) = \prod_{i=1}^{n} P(y_i \geq x_i) \tag{4}$$

In other words, Eqn. 4 gives us the probability of finding $\underline{x}$ in a random feature vector.
Example:

$$\begin{aligned} P(v_2) = & \; P(y_1 \geq 1) \times P(y_2 \geq 1) \times P(y_3 \geq 0) \times P(y_4 \geq 2) \\ = & \; 1 \times \tfrac{1}{4} \times 1 \times \tfrac{3}{4} \\ = & \; \tfrac{3}{16} \end{aligned}$$

B. P-value of $\underline{x}$

Once we know $P(\underline{x})$, the support of $\underline{x}$ in a database of random feature vectors can be modeled as a binomial distribution. To illustrate, a random vector can be viewed as a trial and $\underline{x}$ occurring in it as "success". A database consisting *m* feature vectors will involve *m* trials for $\underline{x}$. The support of $\underline{x}$ in the database is the number of successes. Therefore, the probability of $\underline{x}$ having a support μ is

$$P(\underline{x}; \mu) = \binom{m}{\mu} P(\underline{x})^{\mu} (1 - P(\underline{x}))^{m-\mu} \tag{5}$$

The *probability distribution function (abbr. pdf)* of $\underline{x}$ can be generated from Eqn. 5 by varying μ in the range [0,m]. Therefore, given an observed support μ_0 of $\underline{x}$, its p-value can be calculated by measuring the area under the pdf in the range $[\mu_0, m]$, which is

$$\textit{p-value}(x, \mu_0) = \sum_{i=\mu_0}^{m} P(\underline{x}; i) \quad (6)$$

Eqn. 6 reduces to the regularized Beta function $I(P(\underline{x}); \mu_0, m)$ [16]. When both $mP(\underline{x})$ and $m(1 - P(\underline{x}))$ are large, the binomial distribution can be approximated using a normal distribution.

Smaller the p-value of a feature vector, more statistically significant it is. We make the following observations on monotonicity of p-values of feature vectors:

1) Given two feature vectors $\underline{x}$ and $\underline{y}$, if $\underline{x} \subseteq \underline{y}$, then *p-value*$(x, \mu) \geq$ *p-value*(y, μ) for any μ.
2) Given a feature vector $\underline{x}$, if $\mu_1 \geq \mu_2$, then *p-value*$(x, \mu_1) \leq$ *p-value*(x, μ_2) for any $\underline{x}$.

***Definition** 4:* CLOSED VECTOR. *A feature vector $\underline{x}$ is closed if none of its super-feature vectors has the same support as $\underline{x}$.*

The above monotonicity properties give us the liberty to consider only *closed* feature vectors since the p-value of a non closed feature vector is at least the p-value of its closed super-feature vector.

IV. IDENTIFYING REGIONS OF INTEREST

With the conversion of graphs into feature vectors, and a model to evaluate significance of a graph region in the feature space, we explore how the feature vectors can be analyzed to extract the significant regions.

For illustrative purposes, assume G_1-G_4 in Fig. 6 form a sample graph database. To keep the example simple, assume our feature set consists of all edges in the database. The graphs can be converted to feature space by performing RWR on each node. Table II shows the vectors produced from nodes labeled 'a' in each graph at a restart probability of 0.25. As can be seen, only the edge-types *a-b, b-c,* and *b-d* have non-zero values across G_1, G_2, G_3. This indicates that there could be a subgraph formed by the common non-zero features, which is true in this particular case as shown in Fig. 7. At the same time, no feature has a non-zero value across G_1-G_4 since there is no common subgraph among them.

As one can see, the presence of a "common" sub-feature vector among a set of graphs points to a common subgraph. Similarly, the absence of a "common" sub-feature vector indicates the non-existence of any common subgraph. Mathematically, the *floor* of the feature vectors produces the "common" sub-feature vector.

***Definition** 5:* FLOOR OF VECTORS. *The floor of a set of vectors $\{\underline{v_1}, ..., \underline{v_n}\}$ is a vector $\underline{v_f}$ where,*

TABLE II
FEATURE VECTORS OF NODES LABELED *'a'* IN G_1-G_4

Vector	a-b	a-d	a-e	a-f	b-c	b-d	c-e	c-f	d-f
G_1	2	0	3	0	1	1	0	0	0
G_2	4	0	0	0	2	1	0	0	1
G_3	3	0	0	0	1	2	1	1	0
G_4	0	3	0	3	0	0	0	0	2

$v_{f_i} = min(v_{1_i}, ..., v_{n_i})$ *for* $i = 1...n$. *Ceiling of a set of vectors is defined analogously.*

To utilize this property, we first assume that the presence of a floor with non-zero feature values indicates the presence of a common subgraph among the set of graphs under consideration. The property also sets us up nicely to evaluate the significance of the common subgraph from its feature space representation in the form of the floor. However, this brings us to the obvious issue of managing false positives. Clearly, there could be false positives, where structurally different graphs have similar feature vector representations. We tackle the problem when we return from the feature space to graph space and prune the false positives. Section IV-B explains the method in detail.

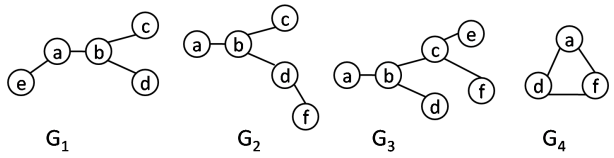


Fig. 6. A sample graph database

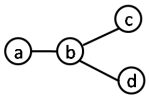


Fig. 7. A common subgraph in G_1, G_2, G_3

A. Feature Vector Mining to Identify Significant Subgraphs

The above discussion shows that it is possible to identify regions likely to contain a common subgraph in the feature space. We therefore concentrate on how to mine common sub-feature vectors which are also significant. The mined vectors will point us to significant subgraphs.

We use a slightly modified version of the feature vector mining algorithm in [15] to find all sub-feature vectors below a p-value threshold. Algorithm 1 presents the FVMine algorithm which explores closed sub-vectors in a bottom-up, depth-first manner. At each step, the floor of the supporting set is evaluated for significance. Next, it moves to a state with a smaller supporting set along a branch and repeats the evaluation process. The algorithm stops branching from a state when all its descendants are guaranteed to have either a support below the support threshold (lines 5-6), or a significance higher than the p-value threshold (lines 10-11), or produces a duplicate

Algorithm 1 FVMine($\underline{x}, \mathbb{S}, b$)

Require: $\underline{x}$ is current sub-feature vector
Require: $\mathbb{S}$ supporting set of $\underline{x}$
Require: b current starting position
Ensure: $\mathbb{A}$ is the set of all significant sub-feature vectors
1: **if** *p-value*$(\underline{x}) \leq maxPvalue$ **then**
2: $\mathbb{A} \leftarrow \mathbb{A} + x$
3: **for** $i = b$ to m **do**
4: $\mathbb{S}' \leftarrow \{\underline{y} | y \in \mathbb{S}, y_i > x_i\}$
5: **if** $|\mathbb{S}'| < minSup$ **then**
6: **continue**
7: $\underline{x}' = floor(\mathbb{S}')$
8: **if** $\exists j < i$ such that $x'_j > x_j$ **then**
9: **continue**
10: **if** *p-value*$(ceiling(\mathbb{S}'), |\mathbb{S}'|) \geq maxPvalue$ **then**
11: **continue**
12: $FVMine(\underline{x}', \mathbb{S}', i)$

Algorithm 2 GraphSig($\mathbb{G}, minSup, maxPvalue$)

Require: $\mathbb{G}$ is a graph database
Require: minSup is the support threshold
Require: maxPvalue is the p-value threshold
Ensure: $\mathbb{A}$ is the answer set of all significant subgraphs
1: $\mathbb{D} \leftarrow \emptyset$
2: $\mathbb{A} \leftarrow \emptyset$
3: **for** each $g \in \mathbb{G}$ **do**
4: $\mathbb{D} \leftarrow \mathbb{D} + RWR(g)$
5: **for** each atom-type a in $\mathbb{G}$ **do**
6: $\mathbb{D}_a \leftarrow \{\underline{v} | \underline{v} \in \mathbb{D}, label(\underline{v}) = a\}$
7: $\mathbb{S} \leftarrow FVMine(floor(\mathbb{D}_a), \mathbb{D}_a, 1)$
8: **for** each vector $\underline{v} \in \mathbb{S}$ **do**
9: $\mathbb{N} \leftarrow \{n | n$ *is a node of label* $a, \underline{v} \subseteq vector(n)\}$
10: $\mathbb{E} \leftarrow \emptyset$
11: **for** each node $n \in \mathbb{N}$ **do**
12: $\mathbb{E} \leftarrow \mathbb{E} + CutGraph(n, radius)$
13: $\mathbb{A} \leftarrow \mathbb{A} + Maximal\_FSM(\mathbb{E}, freq)$

state (lines 8-9). FVMine explores all possible "common" vectors satisfying the significance and support constraints [15].

Fig. 8 demonstrates a running example in a database of four vectors with a support and p-value threshold of 1. In the first step, the floor of all vectors in database is evaluated for significance. Next, it jumps to a state with a smaller supporting set of $\{\underline{v}1, \underline{v}2, \underline{v}3\}$ in the left-most branch and repeats evaluation of the significance. The underlined position shows the value of b in the state. Whenever a duplicate state is reached, it is pruned out. For example, the state with supporting set $\{\underline{v}1\}$ is reached thrice, and is evaluated for significance only once when it is reached for the first time.

B. Mining Significant Subgraphs

Equipped with a model to measure significance of a vector, and a technique to mine closed significant sub-feature vectors, we integrate them to build the *significant graph mining* framework. The idea is to mine significant sub-feature vectors, and use them to locate similar regions which are significant. Algorithm 2 outlines the *GraphSig* algorithm.

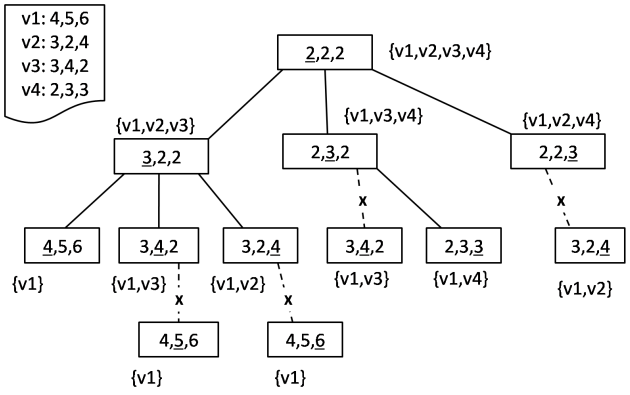


Fig. 8. A running example of FVMine

The algorithm first converts each graph into a set of feature vectors and puts all the vectors together in a single set $\mathbb{D}$ (lines 3-4). $\mathbb{D}$ is next divided into sets, such that $\mathbb{D}_a$ contains all vectors produced from RWR on a node labeled a. Essentially, we are grouping all feature vectors based on the node-type of their source nodes (line 6). On each set $\mathbb{D}_a$, FVMine is performed with a user-specified support and p-value thresholds to retrieve the set of significant sub-feature vectors (line 7).

Given the knowledge that each sub-feature vector could describe a particular subgraph, we scan the database to identify the regions where the current sub-feature vector occurs. This involves finding all nodes labeled a and described by a feature vector $\underline{f}$, such that $\underline{f}$ is a super-vector of the current sub-feature vector (line 9). Each of these nodes is located in a region of interest. We isolate the subgraph centered at each node by using a user-specified radius (line 12). This produces a set of (sub)graphs for each significant sub-feature vector. The cutoff radius could be selected based on some prior knowledge about the typical size of subgraphs that one wants to study. In the worst case, one can select the entire graph where the node occurs.

At this stage, a set of graphs is created for the current significant sub-feature vector. Next, we perform *maximal* frequent subgraph mining on the set with a high frequency threshold since we expect all graphs in the set to contain a common subgraph (line 13). The frequency threshold, for example, could be something in the range [80,100]. A frequent subgraph is maximal if it is not a subgraph of any other frequent subgraph. In our case, we are looking for a particular significant subgraph which is highly frequent in the set. Clearly, this reduces to finding the maximal subgraph as all its subgraphs are going to be frequent as well. As evident in the experimental section later, the largest subgraph often forms the core substructure in the context of chemical compounds. To mine maximal subgraphs, any of the existing techniques

Algorithm 3 Classify($\mathbb{P}, \mathbb{N}, g, k$)

Require: $\mathbb{P}$ is a set of positive significant vectors
Require: $\mathbb{N}$ is a set of negative significant vectors
Require: g is the vector representation of query graph
Ensure: returns classification of g
1: $PQ \leftarrow$ priority queue of size k
2: $score \leftarrow 0$
3: **for** each node $n \in g$ **do**
4: $posDist \leftarrow minDist(vector(n), P)$
5: $negDist \leftarrow minDist(vector(n), N)$
6: **if** $negDist < posDist$ **then**
7: insert $(negDist, -1)$ into PQ
8: **else**
9: insert $(posDist, 1)$ into PQ
10: **for** each tuple $t \in PQ$ **do**
11: $score \leftarrow score + (\frac{1}{t[1]+\delta} \times t[2])$
12: **if** $score > 0$ **then**
13: **return** *"positive"*
14: **else**
15: **return** *"negative"*

([8], [7], [6]) could be used.

This last step (line 13) also prunes out false positives where dissimilar subgraphs are grouped into a set due to similar vector representation. Due to the absence of a common subgraph, when frequent subgraph mining is performed on the set, no frequent subgraph will be produced and as a result the set is filtered out.

V. Application to Graph Classification

With a well-defined technique to mine significant subgraphs, we explore its potential in the domain of graph classification. A common approach in graph classification is to mine patterns on the training set and then create a vector representation of the graphs. For example if $g_1, ..., g_n$ are the patterns mined from training set, then a graph G can be represented by a feature vector $\underline{x} = [x_1, ..., x_n]$, such that $x_i = 1$ if $g_i \subseteq G$, otherwise $x_i = 0$. Once converted to feature space, traditional classifiers can be used to classify graphs.

As one can see, the crucial step lies in the mining process and choosing a meaningful feature set. Take the example of a classifier built on frequent subgraphs such as benzene in the context of chemical compounds. The classifier is unlikely to achieve good results since even though benzene is frequent, it is not discriminative enough. However, a significant subgraph contains more distinctive information since it is able to describe a property where the dataset deviates from expected. To utilize this potential, we develop a classifier built on significant patterns mined by GraphSig.

From an abstract point of view, given a query graph, the algorithm finds the k-closest significant subgraphs in the training set. The majority vote by the k-closest subgraphs, which can be either positive or negative, decides the classification. The significant subgraphs in the positive or negative dataset represent those subgraphs that occur more often than expected in a random database of graphs. Therefore, the occurrence of any of these subgraphs in the query acts as an identifier of its class. We simulate this process in the feature space.

TABLE III
Sample Training Set

Vector	a-b	a-c	b-b	b-c
N_1	0	0	1	1
N_2	0	1	0	0
N_3	1	1	0	1
P_1	2	0	1	3
P_2	1	0	0	0
P_3	0	0	0	1

Algorithm 3 outlines the pseudocode. The feature space representation of the query graph and sets of significant sub-feature vectors from the positive dataset $\mathbb{P}$ and negative dataset $\mathbb{N}$ are fed to the classifier. For each node in the query graph, the distance to the closest sub-feature vector in the training set is calculated (Algorithm 4) and inserted into a priority queue along with the class identifier (lines 3-9). The priority queue keeps track of the k-closest significant sub-feature vectors for the query graph. After all nodes are scanned, a distance-weighted score is calculated for classification (lines 10-15).

The score calculation is similar to a distance weighted k-NN classifier, where we not only consider the majority vote, but also the distance between the query and its neighbor. Thus, a neighbor closer to the query has more say in the voting than a neighbor in the top k but further away. The score from a particular neighbor is calculated as the inverse of its distance to the query. We add a small value δ to the distance before taking its inverse to avoid division by zero. The majority vote can therefore be formulated as the difference between the total score from positive and negative neighbors.

Example: For illustrative purposes, consider the vectors in Table I as the vectors of nodes in a query graph of size 4. Assume that the vectors in Table III are the significant sub-feature vectors from the training set where P_i indicates a positive vector and N_i indicates a negative vector. For vector v_1, N_1-N_3 and P_1 are not sub-vectors of v_1, and thus the distance to them is ∞. For both P_2 and P_3 the distance is 2, and therefore they are the closest significant vectors to v_1. If

Algorithm 4 $minDist(\underline{x}, \mathbb{V})$

Require: $\underline{x}$ is a vector
Require: $\mathbb{V}$ is a set of vectors
Ensure: returns the closest sub-feature to $\underline{x}$ vector in $\mathbb{V}$
1: $min \leftarrow \infty$
2: **for** each vector $\underline{v} \in \mathbb{V}$ **do**
3: **if** $\underline{v} \subseteq \underline{x}$ **then**
4: $dist \leftarrow 0$
5: **for** $i = 1$ to length of $\underline{v}$ **do**
6: $dist \leftarrow dist + (x_i - v_i)$
7: **if** $dist < min$ **then**
8: $min \leftarrow dist$
9: **return** min

k is set to 3, the globally closest sub-feature vectors for the query graph are P_2 or P_3, N_3, and P_2 at a distance of 2, 1, 1 for the vectors v_1, v_2, and v_4 respectively. Therefore, the total score comes to $\frac{1}{2} - 1 + 1 = 0.5$, which classifies the query as positive.

TABLE IV
DEFAULT PARAMETER VALUES

Parameter	Description	Value
α	The restart probability in random walk	0.25
maxPvalue	The p-value threshold for FVMine	0.1
minFreq	The frequency threshold for FVMine	0.1%
cutoff radius	The radius specifying the extent of the sub-graph around a node to cutoff	8
fsgFreq	The frequency threshold for FSG to mine significant subgraphs from sets of similar graphs	80%

VI. EXPERIMENTS

In this section we report experimental results that validate the quality and efficiency of the proposed technique on a series of real-life graph datasets. Our experiments demonstrate:

1) Scalability: The proposed technique outperforms current frequent sub-graph mining techniques and removes the bottleneck of operating at low frequency thresholds. Moreover, our technique scales linearly with database size.
2) Quality: GraphSig can identify significant subgraphs and prune out the non-significant ones. The quality of the patterns is further verified by the proposed classifier built on them which outperforms recent graph classifiers in terms of scalability and quality.

All our algorithms are implemented in Java using SUN JDK 1.6.0. The experiments are performed on a 3.2GHz, 4GB memory PC running Debian Linux 4.0. We use FSG [6] to mine maximal frequent subgraphs from sets generated by GraphSig. Table IV shows the default parameter values used for experiments unless specifically mentioned.

TABLE V
ANTI-CANCER SCREEN DATASETS

Name	Size	Description
MCF-7	28972	Breast
MOLT-4	41810	Leukemia
NCI-H23	42164	Non-Small Cell Lung
OVCAR-8	42386	Ovarian
P388	46440	Leukemia
PC-3	28679	Prostate
SF-295	40350	Central Nervous System
SN12C	41855	Renal
SW-620	42405	Colon
UACC-257	41864	Melanoma
Yeast	83933	Yeast anticancer

A. Graph Datasets

Twelve real-life chemical compound datasets are used in our experiments. The first dataset is DTP-AIDS Antiviral Screen chemical compound dataset from NCI/NIH (http://dtp.nci.nih.gov/). The dataset consists of 43,905 classified chemical molecules, and a total of 1.09 million atoms. On average, each molecule contains 25.4 atoms (vertices) and 27.3 bonds (edges). There are 58 distinct atoms in total, although a majority of them are C, O, and N.

We also use eleven anti-cancer screen datasets available at the PubChem website (http://pubchem.ncbi.nlm.nih.gov). PubChem is a well-maintained compilation of the biological activities of various molecules, containing the bioassay records for anti-cancer screen datasets against various cancer cell lines. Each dataset contains molecules tested against a cancer cell line and its outcome *active* or *inactive*. We selected eleven such graph datasets from the screen-tests. Table V contains a brief summary of each of these NCI bioassays. The number of active molecules is roughly 5% across all datasets.

B. Scalability

In this section, we examine the scalability of our method against frequency, dataset size, and the p-value threshold. We first look at how the running time grows against frequency which is the bottleneck for all frequent sub-graph mining techniques. We choose FSG [6] and gSpan [5] to compare our technique. Although, the output set of FSG and gSpan are different from GraphSig, they are part of the pipeline in the alternative approach to mine significant subgraphs, and therefore, provides a lowerbound on the running time.

Fig. 9 demonstrates the scalability of our method against frequency for mining significant subgraphs on the AIDS dataset. The frequency threshold is varied between 0.1% to 10%. The times on the 0.1% frequency threshold for FSG and gSpan are not reported since they fail to complete even after 10 hours. For GraphSig, we depict the time taken to construct the sets of similar subgraphs which are likely to contain a significant sub-graph. GraphSig+FSG shows the total execution time of GraphSig and then running FSG on the sets outputted with a frequency threshold of 80% to extract the significant subgraphs. As expected, this running time is small due to the high frequency threshold.

As can be seen, the execution times of both FSG and gSpan grow exponentially whereas GraphSig grows linearly. The GraphSig+FSG plot merges with GraphSig since the number of significant sub-histograms decreases with increasing frequency. Thus, at higher frequencies, the additional cost of FSG becomes negligible.

Around 20% of the computation cost of GraphSig is spent on computing the steady state distribution of RWR. Fig. 10 presents the profile of running time on each of the cancer datasets. The y-axis depicts the percentage of time spent on RWR, feature space analysis, and frequent subgraph mining. Since we perform RWR on all nodes in the database regardless of the frequency threshold, the computation cost of GraphSig is bounded by the RWR cost. This is the reason behind FSG and gSpan having lower computation costs at higher frequency thresholds. However, when compared to the cost of mining frequent subgraphs at low frequencies, this RWR cost is negligible.

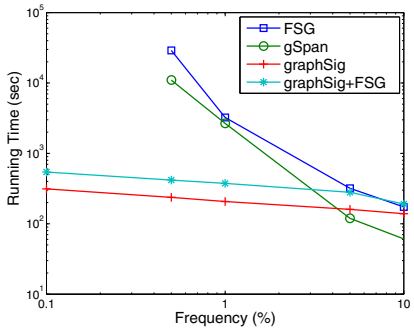


Fig. 9. Time vs. Frequency

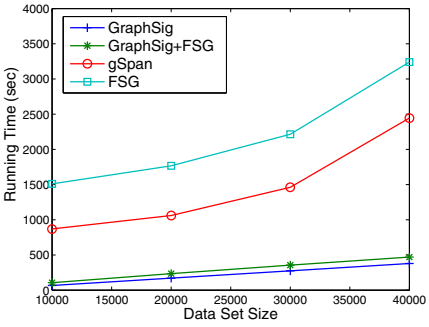


Fig. 11. Time vs. Dataset Size

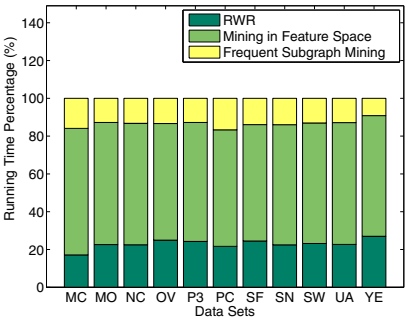


Fig. 10. Profiling of computation cost

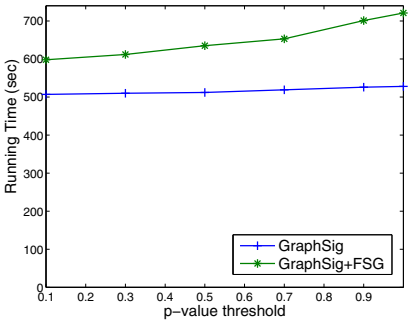


Fig. 12. Time vs. p-value threshold

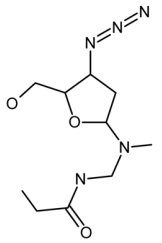
Fig. 11 demonstrates the scalability of our method as we vary the database size. The datasets for this experiment are populated by randomly drawing graphs from the AIDS dataset. The dataset size is varied between 10000 to 40000. For GraphSig, we use a p-value and frequency threshold of 0.1. Due to the enormous execution times of gSpan and FSG at a frequency threshold of 0.1, we set their frequency threshold to 1. Even in this setting, GraphSig easily achieves better performance than gSpan and FSG. As can be seen, gSpan and FSG grow exponentially with database size, whereas GraphSig displays a linear growth rate. FSG on top of GraphSig adds negligible overhead and, as expected, grows linearly.

Fig. 12 demonstrates the growth rate of our techniques against p-value threshold. As can be seen, GraphSig displays a slow growth rate with increasing p-value. Most of the pruning in GraphSig happens due to the support threshold in FVMine and therefore, the effect of increasing the p-value threshold is minimal. GraphSig+FSG also grows at a linear rate since more candidates are generated at a higher p-value and consequently, more time is spent on frequent subgraph mining.

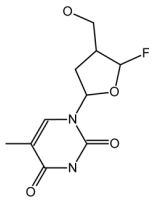
C. Quality Evaluation

In this section, we examine the quality of the significant substructures retrieved from the chemical compound databases. To make the quality assessment more focused, we separate the set of compounds medically active against a disease and run our algorithm on it to retrieve the significant substructures.

Fig. 13(a) shows the most significant subgraph mined by GraphSig in the medically active set of compounds against AIDS. The retrieved structure is a substructure of one of the most potent class of drugs against AIDS, namely Azido Pyrimidines [17]. 3-azido-thymidine (abbr. AZT, NSC 602670), which is the closest active molecule to the graph mined, has one more Carbon and Oxygen atoms, and 3 more edges. AZT is currently the most widely used medicine to control the HIV virus.



(a) Substructure of AZT



(b) Substructure in FDT class of compounds

Fig. 13. Some significant substructures in molecules medically active against AIDS

Fig. 13(b) shows another subgraph retrieved from the active group of compounds in AIDS dataset. The subgraph is a substructure of a class of active compounds called 3'-Fluoro-3'-

deoxy-thymidine (abbr. FDT)[18]. FDT is a fluorinated analog of AZT. It is more active against AIDS than AZT, however, it also displays higher toxicity. The subgraph retrieved is the core structure in the FDT class of compounds, where it binds with a Chlorine, Iodine, or Bromine to form the actual compound.

We also analyze our methods on the anti-cancer screen datasets against Leukemia (MOLT-4) and Melanoma (UACC-257). As shown in the following sections, GraphSig is able to correctly mine substructures that form the core of a class of molecules known to be active against a cancer cell line.

Fig. 14 is a significant subgraph mined from the set of molecules active against Melanoma. The subgraph, methyl-triphenylphosphonium, is the core structure of a class of phosphonium salts where, the binding occurs on the single free Carbon attached to Phosphorus. It displays cytotoxic behavior against a number of cancer cell lines [19].

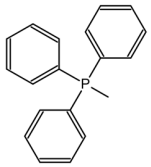
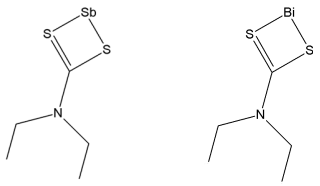


Fig. 14. The core substructure in the class of Phosphonium salts

Fig. 15(a) and 15(b) shows two significant substructures from the molecules active against Leukemia (MOLT-4). It is interesting to note that the only difference in the structures is the presence of Antimony (Sb) and Bismuth (Bi). Incidentally, both Antimony and Bismuth are part of the same group of metals in the periodic table. This phenomenon is particularly interesting since it may lead chemists to try other metals from the group of Antimony and Bismuth and discover new drugs against the target disease. Moreover, the frequency of both Antimony and Bismuth are below 1%, and none of the current frequent subgraph mining techniques can scale to such low frequencies and capture their significance. To investigate this issue further, we plot the relationship between p-value and frequency in Fig. 16. The y-axis depicts the p-value of significant subgraphs when mined with a p-value threshold of 0.1, and the x-axis depicts the frequency. As can be seen, a high number of significant subgraphs have a frequency below 1%. Moreover, benzene, which is a ubiquitous subgraph in chemical compound databases with frequency around 70%, turns out to be non-significant. This result clearly demonstrates that significant subgraphs exist at all frequencies.

The above results highlight the practical usefulness of GraphSig in a number of areas. GraphSig successfully retrieved the core structures of various classes that are medically active against Cancer or AIDS, thereby establishing a link between significance and medicinal values of the structures. Our technique also displays potential in providing leads to design new drugs by learning from the already discovered drugs. Moreover, GraphSig successfully overcomes the scalability bottleneck of mining significant subgraphs at low frequencies



(a) Substructure made of Sb

(b) Substructure made of Bi

Fig. 15. Some significant substructures in molecules medically active against Leukemia

and produces accurate results.

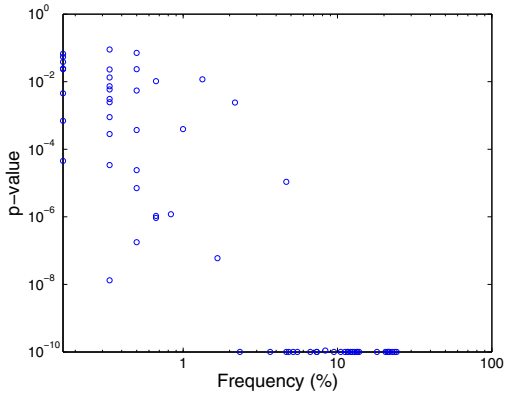


Fig. 16. Relationship between frequency and p-value

D. Classification Application

Having verified the composition of the patterns mined by GraphSig, in this section we cross-check the quality of the patterns by employing them to classify graphs. In the following experiments, we compare our classification algorithm with two recent graph classification approaches: pattern-based graph classification and kernel-based method. We chose the most recent pattern based classifier - Scalable Leap Search (abbr. **LEAP** [20]) for comparison. For the kernel based method, we chose the state-of-the-art graph kernel - optimal assignment kernel (abbr. **OA** [21]). LEAP converts each graph in the training set to a feature vector. Each feature corresponds to a pattern mined by LEAP. The feature value is based on the occurrence of its corresponding pattern in the graph. On the other hand, kernel-based methods attempt to design effective kernel functions to measure the similarity between graphs. The implementations of these methods are provided by the authors.

To construct a balanced training set we sample 30% of the active compounds and an equal number of inactive compounds. However, as shown in the experimental results later, the OA kernel is unable to scale to such large training set. To overcome this problem we run OA on a balanced training set of 10% active compounds. The classification accuracy is

TABLE VI
AUC COMPARISON BETWEEN OA, LEAP, AND GRAPHSIG

Dataset	OA Kernel	Leap	GraphSig
MCF-7	0.68 ± 0.12	0.76 ± 0.04	**0.77 ± 0.02**
MOLT-4	0.65 ± 0.06	0.72 ± 0.06	**0.74 ± 0.02**
NCI-H23	0.79 ± 0.08	0.79 ± 0.05	**0.80 ± 0.02**
OVCAR-8	0.67 ± 0.04	0.78 ± 0.02	**0.79 ± 0.02**
P388	0.79 ± 0.07	**0.84 ± 0.03**	**0.84 ± 0.02**
PC-3	0.66 ± 0.09	**0.76 ± 0.04**	**0.76 ± 0.03**
SF-295	0.75 ± 0.11	0.77 ± 0.02	**0.80 ± 0.02**
SN12C	0.75 ± 0.08	**0.80 ± 0.02**	**0.80 ± 0.03**
SW-620	0.70 ± 0.02	0.76 ± 0.04	**0.77 ± 0.02**
UACC-257	0.65 ± 0.05	0.75 ± 0.03	**0.81 ± 0.02**
Yeast	0.64 ± 0.04	0.71 ± 0.02	**0.73 ± 0.04**
Average	0.702 ± 0.07	0.767 ± 0.03	**0.782 ± 0.02**

evaluated with 5-fold cross validation. The running time of LEAP is measured as the time to compute the feature vector representation of all graphs in the training set. For OA, the time to compute the kernel is treated as its running time. For both methods, we use support vector machine to classify. We use LIBSVM [22] along with the parameters originally used by the respective authors. For our classification algorithm, we use $k = 9$. For GraphSig, the running time is measured as the total time to classify all graphs in the testing set.

We compare the classification quality by measuring the area under the ROC curve (AUC) achieved by these three methods. ROC curve is a graphical plot of the true positive rate against false positive rate for a classifier as the discrimination threshold is varied. The area under this curve (AUC) is a measure of the classifier accuracy. The area is always within the range of [0,1] and a perfect model will have an area of 1.

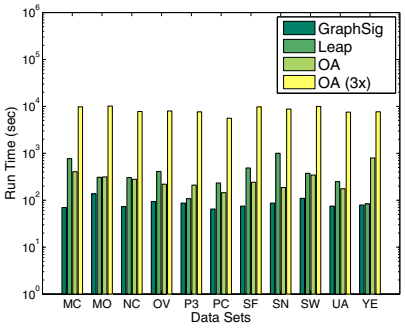


Fig. 17. Running time: OA vs. LEAP vs. GraphSig

Table VI shows AUC for OA, LEAP and GraphSig. Clearly, GraphSig outperforms OA and LEAP. On average, GraphSig achieves an AUC of 0.08 and 0.015 higher than OA and LEAP respectively. Fig. 17 shows the running time in log scale for OA, LEAP, and GraphSig. We also show the running time of OA on a balanced training set of 30% (abbr. OA(3X)) active compounds to demonstrate that it is not scalable to large datasets. All running times are averaged over 5 folds, except for OA(3X). Since OA(3X) takes too long to finish computing over all 5 folds, we report the time for only one fold. The average running time for GraphSig on all eleven datasets is 87 seconds, whereas LEAP takes 395 seconds on average. Overall, the proposed classifier performs 4.5 times faster than LEAP and 80 times faster than OA(3X), while achieving better quality, or in some cases, similar to LEAP.

VII. RELATED WORK

Graph mining has been an active research topic in the last few years. In particular, frequent subgraph mining has generated a lot of interest and has been extensively studied in the data mining community. Various efficient algorithms have been developed, such as AGM [12], FSG [6], gSpan [5], CloseGraph [13], Spin [7], Margin [8], and Gaston [23]. In general, two prominent approaches exist in mining frequent subgraphs: apriori-based approach, and pattern-growth approach.

In AGM, Inokuchi et al. adopted an apriori approach to address the problem. Kuramochi et al. proposed FSG, an apriori based approach to mine frequent subgraphs. Yan and Han proposed gSpan using the pattern growth approach to mine frequent patterns. Later, they devised CloseGraph, a technique to mine only closed frequent patterns. Huan et al. explored frequent subgraphs using a canonical adjacency matrix representation of graphs [9]. Later they developed SPIN to mine maximal frequent subgraphs. Vanetik et al. introduced an apriori-based approach that uses edge-disjoint paths as building blocks for their technique [10]. They later addressed the problem in the context of partially labeled graph patterns [11]. While all of the above techniques are efficient in mining frequent patterns, none of them is able to address the problem of mining significant patterns due to their inability to scale to low frequency thresholds.

Besides data analysis, significant pattern mining also finds application in areas such as graph indexing and graph classification. Yan and Han demonstrated that pattern-based indexing can achieve fast graph search results in their work GIndex [24]. Cheng et al. proposed a graph query system built on frequent subgraphs [25]. In the area of graph classification, pattern based classifiers were developed in [1], [26], [27].

In spite of all the above mentioned advances in graph mining, few algorithms exist that mine significant patterns. He and Singh [15] introduced a statistical model to evaluate significance of subgraphs in the feature space. While the technique is useful for ranking patterns based on significance, it remains dependent on existing frequent subgraph mining techniques to generate all subgraphs. Yan et al. developed a framework to mine significant patterns based on structural leap search [20]. They consider a group of objective functions defined in [28] for measuring significance and concentrate on a setting where the graph database can be divided into a positive set and a negative set.

Milo et al. [29] developed a technique to mine network motifs as graph patterns that appear more frequently than in randomized networks. However, their technique assumes a single large graph, whereas our technique works over a large collection of graphs. Moreover, their approach is based on

simulation where a large number of randomized networks is first generated maintaining some empirical measures such as degree of vertices and number of edges and then the frequency of the query graph is enumerated to compute its p-value. As a result, the approach has the obvious scalability problem since subgraph isomorphism is NP-complete [30]. Moreover, the approach is not precise for computing and comparing small p-values since the least possible p-value is 1/N where, N is the number of randomized networks.

VIII. Conclusion

In this paper, we studied an increasingly important problem of mining significant subgraphs from large graph databases. Existing frequent subgraph mining techniques fail to solve the problem due to their inability to scale to low frequencies. We proposed a new scalable approach that is capable of mining significant subgraphs at a low frequency threshold. Our technique accesses only a small portion of the exponential search space, and groups candidate subgraphs into sets based on their similarity. As a result, frequent subgraph mining can be performed on each set with a high frequency threshold. The unique approach produced a drastic reduction in the computation cost. Experimental results on chemical compound datasets revealed the practical usefulness of our method. GraphSig opens up a new direction in graph-pattern based applications by unleashing the potential of significant patterns. For example, in the context of chemical compounds, it displayed potential in identifying important substructures. The proposed classifier, built on significant patterns, highlights another important aspect of the study.

Acknowledgements: We thank X. Yan for sharing the code of LEAP. The work was supported by NSF Grant IIS-0612327.

References

[1] M. Deshpande, M. Kuramochi, and G. Karypis, "Frequent Sub-Structure-Based Approaches for Classifying Chemical Compounds," in *ICDM '03: Proceedings of the Third IEEE International Conference on Data Mining*. Washington, DC, USA: IEEE Computer Society, 2003, p. 35.

[2] S. White and P. Smyth, "Algorithms for estimating relative importance in networks," in *Proceedings of the ninth ACM SIGKDD International Conference on Knowledge discovery and data mining*, 2003, pp. 266–275.

[3] J. Lee, J. Oh, and S. Hwang, "STRG-Index: spatio-temporal region graph indexing for large video databases," in *Proceedings of the 2005 ACM SIGMOD*, 2005, pp. 718–729.

[4] R. Sharan, S. Suthram, R. M. Kelley, T. Kuhn, S. McCuine, P. Uetz, T. Sittler, R. M. Karp, and T. Ideker, "Conserved patterns of protein interaction in multiple species," *Natl Acad Sci*, 2005.

[5] X. Yan and J. Han, "gSpan: Graph-Based Substructure Pattern Mining," in *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM'02)*, 2002, p. 721.

[6] M. Kuramochi and G. Karypis, "Frequent subgraph discovery," in *ICDM*, 2001, pp. 313–320.

[7] J. Huan, W. Wang, J. Prins, and J. Yang, "SPIN: mining maximal frequent subgraphs from graph databases," in *Proceedings of the tenth ACM SIGKDD International Conference on Knowledge discovery and data mining*, 2004, pp. 581–586.

[8] L. T. Thomas, S. R. Valluri, and K. Karlapalem, "MARGIN: Maximal Frequent Subgraph Mining," in *Proceedings of the Sixth International Conference on Data Mining*, 2006, pp. 1097–1101.

[9] J. Huan, W. Wang, and J. Prins, "Efficient Mining of Frequent Subgraphs in the Presence of Isomorphism," in *Proceedings of the Third IEEE International Conference on Data Mining*, 2003, p. 549.

[10] N. Vanetik, E. Gudes, and S. E. Shimony, "Computing Frequent Graph Patterns from Semistructured Data," *International Conference on Data Mining*, vol. 00, p. 458, 2002.

[11] N. Vanetik and E. Gudes, "Mining Frequent Labeled and Partially Labeled Graph Patterns," in *Proceedings of the 20th International Conference on Data Engineering*, 2004, p. 91.

[12] A. Inokuchi, T. Washio, and H. Motoda, "An apriori-based algorithm for mining frequent substructures from graph data," in *Principles of Data Mining and Knowledge Discovery*, 2000, pp. 13–23.

[13] X. Yan and J. Han, "CloseGraph: mining closed frequent graph patterns," in *Proceedings of the ninth ACM SIGKDD International Conference on Knowledge discovery and data mining*, 2003, pp. 286–295.

[14] S. Theodoridis and K. Koutroumbas, *Pattern Recognition, Third Edition*. Academic Press, Inc., 2003.

[15] H. He and A. K. Singh, "GraphRank: Statistical Modeling and Mining of Significant Subgraphs in the Feature Space," in *Proceedings of the Sixth International Conference on Data Mining*, 2006, pp. 885–890.

[16] "http://mathworld.wolfram.com/binomialdistribution.html." [Online]. Available: http://mathworld.wolfram.com/BinomialDistribution.html

[17] "http://dtp.nci.nih.gov/docs/aids/searches/list.html."

[18] I. K. Wilson, S. Chatterjee, and W. Wolf, "Synthesis of 3'-fluoro-3'-deoxythymidine and studies of its 18F-radiolabeling, as a tracer for the noninvasive monitoring of the biodistribution of drugs against AIDS," *Journal of Fluorine Chemistry*, vol. 55, pp. 283–289, 1991.

[19] A. Manetta, G. Gamboa, A. Nasseri, Y. D. Podnos, D. Emma, G. Dorion, L. Rawlings, P. M. Carpenter, A. Bustamante, J. Patel, and D. Rideout, "Novel phosphonium salts display in vitro and in vivo cytotoxic activity against human ovarian cancer cell lines," *Gynecologic oncology*, vol. 60, pp. 203–212, 1996.

[20] X. Yan, H. Cheng, J. Han, and P. S. Yu, "Mining Significant Graph Patterns by Scalable Leap Search," in *Proceedings of SIGMOD*, 2008.

[21] H. Fröhlich, J. K. Wegner, F. Sieker, and A. Zell, "Optimal assignment kernels for attributed molecular graphs," in *Proceedings of the 22nd International Conference on Machine learning*, 2005, pp. 225–232.

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

[23] S. Nijssen and J. N. Kok, "The Gaston tool for Frequent Subgraph Mining," in *Proceedings of the International Workshop on Graph-Based Tools, Grabats 2004, Rome, Italy, October 2, 2004*. Elsevier, 2004.

[24] X. Yan, P. S. Yu, and J. Han, "Graph indexing: a frequent structure-based approach," in *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD*, 2004, pp. 335–346.

[25] J. Cheng, Y. Ke, W. Ng, and A. Lu, "Fg-index: towards verification-free query processing on graph databases," in *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD*, 2007, pp. 857–872.

[26] N. Wale and G. Karypis, "Comparison of Descriptor Spaces for Chemical Compound Retrieval and Classification," in *ICDM '06: Proceedings of the Sixth International Conference on Data Mining*, 2006, pp. 678–689.

[27] S. Kramer, L. D. Raedt, and C. Helma, "Molecular feature mining in HIV data," in *KDD '01: Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, 2001, pp. 136–143.

[28] P.-N. Tan, V. Kumar, and J. Srivastava, "Selecting the right interestingness measure for association patterns," in *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2002, pp. 32–41.

[29] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon, "Network motifs: simple building blocks of complex networks." *Science*, vol. 298, no. 5594, pp. 824–827, October 2002.

[30] S. Fortin, "The graph isomorphism problem," Department of Computer Science, University of Alberta, Canada., Tech. Rep., 1996.