On Classification of Graph Streams

Charu C. Aggarwal*

Abstract

In this paper, we will examine the problem of classification of massive graph streams. The problem of classification has been widely studied in the database and data mining community. The graph domain poses significant challenges because of the structural nature of the data. The stream scenario is even more challenging, and has not been very well studied in the literature. This is because the underlying graphs can be scanned only once in the stream setting, and it is difficult to extract the relevant structural information from the graphs. In many practical applications, the graphs may be defined over a very large set of nodes. The massive domain size of the underlying graph makes it difficult to learn summary structural information for the classification problem, and particularly so in the case of data streams. In order to address these challenges, we proposed a probabilistic approach for constructing an "in-memory" summary of the underlying structural data. This summary determines the discriminative patterns in the underlying graph with the use of a 2-dimensional hashing scheme. We provide probabilistic bounds on the quality of the patterns determined by the process, and experimentally demonstrate the quality on a number of real data sets.

Keywords: Graphs, Classification

1 Introduction

The problem of graph classification arises in the context of a number of classical domains such as chemical and biological data, the web, and communication networks. Recent methods for representing semi-structured data have also lead to increasing interest in this field. A detailed discussion of graph mining and classification algorithms may be found in [4].

Recently, the emergence of internet applications has lead to increasing interest in *dynamic graph streaming applications* [2, 3, 10]. Such network applications are defined on a *massive underlying universe* of nodes. Some examples are as follows:

(1) The communication pattern of users in social network in a modest time window can be decomposed into a group of disconnected graphs, which are defined over a massive-domain of user nodes. **(2)** The browsing pattern of a single user (constructed from a proxy log) is typically a small sub-graph of the web graph. The browsing pattern over all users can be interpreted as a continuous stream of such graphs. **(3)** The intrusion traffic on a communication network is a stream of localized graphs on the massive IP-network.

This paper will examine the most challenging case in which the data is available only in the form of a stream of graphs (or edge streams with graph identifiers). Furthermore, we assume that the *number of distinct edges is so large, that it is impossible to hold summary information about the underlying structures explicitly.* For example, a graph with more than 10^7 nodes may contains as many as 10^{13} edges. Clearly, it may not be possible to store information about such a large number of edges explicitly. Thus, the complexity of the graph stream classification problem arises in two separate ways:

(1) Since the graphs are available only in the form of a stream, this restrains the kinds of algorithms which can be used to mine structural information for future analysis. An additional complication is caused by the fact the the edges for a particular graph may arrive *out of order* (i.e. not contiguously) in the data stream. Since re-processing is not possible in a data stream, such out-of-order edges create challenges for algorithms which extract structural characteristics from the graphs.

(2) The massive size of the graph creates a challenge for effective extraction of information which is relevant to classification. For example, it is difficult to even store summary information about the large number of distinct edges in the data. In such cases, the determination of frequent discriminative subgraphs may be computationally and space-inefficient to the point of being impractical.

The currently available algorithms for classification [13, 14, 21] are not designed for the enormous challenges inherent in the scenarios discussed above. In this paper, we will design a *discriminative subgraph mining approach* to graph classification. We will define discriminative subgraphs both in terms of edge pattern co-occurrence and the class label distributions. The presence of such subgraphs in test instances can be used in order to infer their class labels. Such discriminative subgraphs are difficult to determine with enumeration based algorithms because of the massive domain

*IBM T.J. Watson Research Center, charu@us.ibm.com

of the underlying data. Therefore, we will design an efficient probabilistic algorithm for determining the discriminative patterns. The probabilistic algorithm uses a hash-based summarization approach to capture the discriminative behavior of the underlying massive graphs. We will show how this compressed representation can be leveraged for effective classification of test (graph) instances.

This paper is organized as follows. In the next section, we will formalize the problem of classification of graph streams with the use of discriminative patterns. We will leverage this framework in order to design an effective algorithm for graph classification in section 3. Section 4 presents the experimental results. The conclusions and summary are presented in section 5.

1.1 Related Work The problem of graph mining is often studied in the context of *static* data such as chemical compounds or molecular biology in which the nodes are drawn from a modest base of possibilities. Since the graphs may be drawn on a limited domain, it significantly eases the ability to perform mining tasks in such cases. Furthermore, it is assumed that the data set is of modest size, and it can be stored on disk or main memory for mining purposes. Many recent graph mining algorithms for frequent pattern mining [12, 16, 20], clustering [1, 17] and classification [13, 14, 21] are designed in the context of this modest assumption.

The problem of graph classification is studied in two different contexts. In the problem of *node classification*, the goal is to determine labels for the nodes with the use of linkage information from other nodes [13, 14]. However, in *object based graph classification*, the goal is to classify *individual graphs* as objects [21]. We focus on the latter case, for which there are currently no known algorithms in the stream scenario. Some recent work proposes algorithms for clustering and dense pattern mining in graph streams [2, 3]. The work in [3] uses count-min sketches in order to cluster massive graph streams, though such an approach does not easily extend to classification because of the information loss associated with the sketch representation. The work in [2] also uses min-hash summaries for dense pattern mining, though the approach is focussed on determining *node patterns*, rather than *structural edge-based discriminative patterns*, as would be required in a classification application. As we will see, the latter is a much more challenging problem because of the significantly larger number of edges as compared to the nodes. This necessitates the design of a 2-dimensional hash structure which is a combination of a min-hash and a standard hashing scheme.

2 Discriminative Subgraph Patterns

In this section, we will present the model for mining classification patterns in graph streams. We will first introduce some notations and definitions. We assume that the graph is defined over the node set N. This node set is assumed to be drawn from a massive domain of identifiers. The individual graphs in the stream are denoted by $G_1 \ldots G_n \ldots$. Each graph G_i is associated with the class label C_i which is drawn from $\{1 \ldots m\}$. It is assumed that the data is *sparse*. The sparsity property implies that even though the node and edge domain may be very large, the number of edges in the individual graphs may be relatively modest. This is generally true over a wide variety of real applications such as those discussed in the introduction section. We further note that in the stream case, the edges of each graph G_i may not be neatly received at a given moment in time. Rather, the edges of different graphs may appear *out of order* in the data stream. This means that the edges for a given graph may not appear contiguously in the stream. This is definitely the case for many applications such as social networks and communication networks in which one cannot control the ordering of messages and communications across different edges. Our paper will therefore assume this difficult case in which the edges do not appear in order. For the purposes of this paper, we assume that the edges are received in the form $< GraphId >< Edge >< ClassLabel >$. Note that the class label is the same across all edges for a particular graph identifier. For notational convenience, we can assume that the class label is appended to the graph identifier, and therefore we can assume (without loss of generality or class information) that the incoming stream is of the form $< GraphId >< Edge >$. The value of the variable $< Edge >$ is defined by its two constituent nodes.

In this paper, we will design a rule-based approach which associates discriminative subgraphs to specific class labels. Therefore, we need a way of quantifying the significance of a particular subset of edges P for the purposes of classification. Ideally, we would like the subgraph P to have significant statistical presence in terms of the *relative frequency* of its constituent edges. At the same time, we would like P to be discriminative in terms of the class distribution.

The first criterion retains subgraphs with high relative frequency of presence, whereas the second criterion retains only subgraphs with high discriminative behavior. We will discuss effective techniques for determining patterns which satisfy both of these characteristics. First, we will define the concept of a significant subgraph in the data. A significant subgraph P is defined as a subgraph (set of edges), for which the edges are cor-

related with one another in terms of absolute presence. We also refer to this as *edge coherence*. This concept is formally defined as follows:

DEFINITION 1. *Let $f_{\cap}(P)$ be the fraction of graphs in $G_1 \ldots G_n$ in which* **all** *edges of P are present. Let $f_{\cup}(P)$ be the fraction of graphs in which* **at least one or more** *of the edges of P are present. Then, the edge coherence $C(P)$ of the subgraph P is denoted by $f_{\cap}(P)/f_{\cup}(P)$.*

We note that the above definition of edge coherence is focussed on *relative presence* of subgraph patterns rather than the absolute presence. This ensures that only significant patterns are found. Therefore, large numbers of irrelevant patterns with high frequency but low significance are not considered. While the coherence definition is more effective, it is computationally quite challenging because of the size of the search space which may need to be explored. The randomized scheme discussed in this paper is specifically designed in order to handle this challenge.

Next, we define the class discrimination power of the different subgraphs. For this purpose, we define the *class confidence* of the edge set P with respect to the class label $r \in \{1 \ldots m\}$.

DEFINITION 2. *Among all graphs containing subgraph P, let $s(P, r)$ be the fraction belonging to class label r. We denote this fraction as the confidence of the pattern P with respect to the class r.*

Correspondingly, we define the concept of the dominant class confidence for a particular subgraph.

DEFINITION 3. *The dominant class confidence $DI(P)$ or subgraph P is defined as the maximum class confidence across all the different classes $\{1 \ldots m\}$.*

A significantly large value of $DI(P)$ for a particular test instance indicates that the pattern P is very relevant to classification, and the corresponding dominant class label may be an attractive candidate for the test instance label.

In general we would like to determine patterns which are interesting in terms of absolute presence, and are also discriminative for a particular class. Therefore, we use the parameter-pair (α, θ) which correspond to threshold parameters on the edge coherence and class interest ratio.

DEFINITION 4. *A subgraph P is said to be be (α, θ)-significant, if it satisfies the following two* edge-coherence *and* class discrimination *constraints:*
(a) Edge Coherence Constraint: *The edge-coherence $C(P)$ of subgraph P is at least α. In other words, we have $C(P) \geq \alpha$.*
(b) Class Discrimination Constraint: *The dominant class confidence $DI(P)$ is at least θ. In other words, we have $DI(P) \geq \theta$.*

The above constraints are quite challenging because of the size of the search space which needs to be explored in order to determine relevant patterns. We have chosen this approach, because it is well suited to massive graphs in which it is important to prune out as many irrelevant patterns as possible. The edge coherence constraint is designed to prune out many patterns which are abundantly present, but are not very significant from a relative perspective. This helps in more effective classification.

3 A Probabilistic Approach for Finding Discriminative Subgraphs

In this section, we will describe a probabilistic *min-hash approach* for determining discriminative subgraphs. The min-hash technique is an elegant probabilistic method, which has been used for the problem of finding interesting 2-itemsets [8]. This technique cannot be easily adapted to the graph classification problem, because of the large number of distinct edges in the graph. Therefore, we will use a 2-dimensional compression technique in which a min-hash function will be used in combination with a more straightforward randomized hashing technique. We will see that this combination approach is extremely effective for graph stream classification.

The aim of constructing this synopsis is to be able to design a continuously updatable data structure, which can determine the most relevant discriminative subgraphs for classification. Since the size of the synopsis is small, it can be maintained in main memory and be used at any point during the arrival of the data stream. The ability to continuously update an in-memory data structure is a natural and efficient design for the stream scenario. At the same time, this *structural synopsis* maintains sufficient information which is necessary for classification purposes.

For ease in further discussion, we will utilize a tabular binary representation of the graph data set with N rows and n columns. We note that this table is only *conceptually* used for description purposes, but it is *not explicitly maintained* in the algorithms or synopsis construction process. The N rows correspond to the N different graphs present in the data. While columns represent the different edges in the data, this is not a one-to-one mapping. This is because the number of possible edges is so large that it is necessary to use a uniform random hash function in order to map the edges onto the n columns. The choice of n depends upon the space available to hold our synopsis effectively, and

affects the quality of the final results obtained. Since many edges are mapped onto the same column by the hash function, the support counts of subgraph patterns are over-estimated with this approach. Since the edge-coherence $C(P)$ is represented as a ratio of supports, the edge coherence may either be over- or under-estimated. In order to simplify our discussion, we will first describe the algorithm without the use of a hash-mapping of the columns. Then, we will discuss the changes needed to implement a column-wise hash mapping. Therefore, these two cases are sequentially discussed in order of increasing complexity:

• **Complete Table with no column-wise hash-mapping:** In this case, we have a column corresponding to each distinct edge. Then, we apply min-hash compression on the rows only. While this is not a practical solution for real applications, it is useful to discuss it first in order to set the stage for the discussion of the 2-dimensional compression scheme.

• **Compressed Table with column-wise hash mapping:** In this case, we map multiple columns into a single column with the use of a conventional hash function. Then, we apply the min-hash compression on the rows. This is the main scheme designed by our approach.

The use of these two kinds of virtual tables is helpful for analysis of the scheme with increasingly complex assumptions. Specifically, we will analyze our scheme with or without hash-compression on the columns. First, we will propose the simple scheme without hash compression on the columns (and only a min-hashing scheme on the rows), and analyze it in the context of a complete tabular representation of the edge representation. Then, we will analyze the additional effects of the 2-dimensional approach in which we also apply a uniform random hash compression on the columns. We will show that the overall effect of this approach is to create a highly effective classifier which can work with fast graph streams.

3.1 Min-hash Approach with Complete Virtual Table The main idea of the min-hashing scheme is to determine co-occurrence in the values on different columns by using the sort-order on the rows in the data. In order to create this sort-order, we generate a single uniformly random hash value for each row in the data. All the columns in the data are sorted by order of this hash value. Note that such an approach results in each column being sorted in exactly the same random order. We observe the following;

OBSERVATION 1. *Consider a set P of edges. Let P' be the columns in the virtual table corresponding to P. We examine these columns in sorted order by hash value. The probability that the indices of the first row for each column in P' with a 1-value are the same is equal to the edge coherence $C(P)$.*

The above observation is easy to verify, since the index of the first row with a 1 in *any of the* columns in P' is also the first row in which *any of the* edges of P' are present. The probability that this row contains *all 1-values* across the relevant columns is simply the edge coherence probability $C(P)$. We can use this approach in order to efficiently construct a sampling estimate of the edge coherence probability. We simply need to use k different hash values for constructing the sort order, and estimate the above-mentioned probability by computing the fraction of the k samples for which all values are 1.

First, we will discuss the implementation of the min-hash approach with the use of a complete virtual table, and no compression on the columns. In this case, we assume that the table has E columns, where $E = N \cdot (N-1)/2$ is the number of possible distinct edges in the data. The min-hash index is used in order to construct a *transactional representation* of the underlying data. First, we apply the min-has approach with k different hash values in order to create a table of size $k \times E$. Now, let us examine a particular sample from this min-hash representation. Let $r_1 \ldots r_E$ be the row indices of the first row with a 1 in the corresponding column. We partition the set $r_1 \ldots r_E$ into groups for which the min-hash index values are the same. Thus, if $r_1 \ldots r_E$ are partitioned into $Q_1 \ldots Q_h$, then each $r_i \in Q_j$ has the same value (for fixed j). For each partition Q_j, we create a categorical transaction containing the indices of the corresponding columns. For example, if $Q_j = \{r_2, r_{23}, r_{45}, r_{71}\}$, then we create the transaction $T_j = \{2, 23, 45, 71\}$. We can construct transactions $T_1 \ldots T_h$ corresponding to the equi-index partitions $Q_1 \ldots Q_h$. Since each index set creates a corresponding transaction set, we can improve the accuracy of our estimation process by repeating this sampling process k times and creating k different index sets. For each index set, we add corresponding transaction sets to the overall transaction set $\mathcal{T}$. We make the following observation:

OBSERVATION 2. *Let $\mathcal{T}$ be the transactions constructed from the min-hash index set. Then, the coherence probability $C(P)$ of an edge set P can be estimated as the absolute support of P' in the transaction set $\mathcal{T}$, divided by k.*

Observation 2 is a direct extension of Observation 1. This is because the column set P' supports a transaction in $T_1 \ldots T_h$ if and only if the min-hash indices of the edges in P' are the same in the corresponding min-hash sample from which that transaction was constructed.

It remains to explain the details of the implementation in the graph stream scenario. We note that the

```
Algorithm UpdateMinHash(Incoming Edge: e ∈ G,
        Sampling Rate: k, Edge-specific Min-hash values: V
          Edge-specific Min-hash indices: I);
{ Assume that graph G is associated with a numerical
  identifier Id(G) which denotes the order in which it
  was received in data stream; }
begin
  Generate k random hash values p_1 ... p_k,
     where p_i = h(i, Id(G));
    for each p_i in the k generated hash values do
      begin
      if edge e is not included in current set of
         min-hash statistics (I, V) then
          Add (e, p_i) to V and (e, Id(G)) to I;
          else begin
          Let (e, m) ∈ V and (e, CurrentId) ∈ I be the
            current tuples in V and I associated with
            the ith min-hash value for e;
          if p_i < m replace (e, m) and (e, CurrentId) with
            (e, p_i) and (e, Id(G)) respectively;
          end
    end
end
```

Figure 1: Updating Min-hash Index for each Incoming Edge

rows of our (virtual) table representing edge presence in a particular graph may not be available at a given time. This is because the edges for a given graph may not arrive contiguously. Furthermore, the data is typically very sparse and most values in the table are 0. Therefore, we need an update process which can work with *sparse unordered edges* and does not use the virtual table explicitly.

For each incoming graph edge $e \in G$ in the stream (with graph identifier $Id(G)$), we generate k random hash values $h(1, Id(G)) \ldots h(k, Id(G))$. The ith hash value $h(i, Id(G))$ is denoted by p_i. We note that the hash function $h(\cdot, \cdot)$ uses $Id(G)$ as an input, so that the same random hash value can be generated when an edge from the same graph G is encountered at a later stage in the stream. In order to generate the random hash value, we use a standard hash function on the concatenation of the strings corresponding to the two arguments of $h(\cdot, \cdot)$.

Let L be the running estimate of the number of distinct edges encountered so far in the stream. We maintain a set $\mathcal{V}$ of $k \cdot L$ running minimum hash values together with a set $\mathcal{I}$ of $k \cdot L$ corresponding graph identifiers for which these minimum values were obtained. The value of L will increase during progression of the data stream, as more and more distinct edges are received by the stream. Each entry in $\mathcal{V}$ is of the form $(e, MinHashValue)$, and each entry in $\mathcal{I}$ is of the form $(e, GraphIdentifier)$. There are k such entries in both $\mathcal{V}$ and $\mathcal{I}$ *for each distinct edge* encountered so far in the data stream. For each edge e, the ith minimum hash samples in $\mathcal{V}$ represents the minimum value of $h(i, Id(G))$ *over all graphs* G *which contain* e. The k entries in $\mathcal{I}$ for a particular edge e represent the k graph identifiers for which these minimum hash values in $\mathcal{V}$ were obtained.

As mentioned earlier, the edges in the stream may appear out of order, and therefore each edge is associated with its corresponding graph identifier in the stream. For each incoming edge e and graph identifier $Id(G)$, the sets $\mathcal{V}$ and $\mathcal{I}$ are updated as follows. We generate the k different hash values $p_1 = h(1, Id(G)) \ldots p_k = h(k, Id(G))$. In the event that the edge e has not been encountered so far, its information will not be present in $\mathcal{V}$ and $\mathcal{I}$. In such a case, the value of L is incremented, and the corresponding entries for e are included in $\mathcal{V}$ and $\mathcal{I}$ respectively. Specifically, the k hash values in $\mathcal{V}$ for this newly tracked edge are set to the k recently generated hash values $p_1 \ldots p_k$. Therefore, we include the entries $(e, p_1) \ldots (e, p_k)$ in $\mathcal{V}$. Correspondingly, we add the entries $(e, Id(G)) \ldots (e, Id(G)))$ to $\mathcal{V}$. On the other hand, if the edge e has already been encountered in the stream, then we examine the corresponding entries $(e, MinHashValue_1) \ldots (e, MinHashValue_k)$ in $\mathcal{V}$. In the event that the newly generated hash value p_i is less than $MinHashValue_i$, then we replace $(e, MinHashValue_i)$ by (e, p_i). We also replace the corresponding entry in $\mathcal{I}$ by $(e, Id(G))$. The overall approach for maintaining the hash indices *for each incoming edge* is illustrated in Figure 1.

We note that the process discussed in Figure 1 continuously maintains a summary representation of the underlying graphs. Since these summary statistics are maintained in main memory, we can utilize them at any time in order to determine the coherent edge patterns. This is because an analyst may require the classification approach to be applied at any time during the data stream computation. To do so, we construct *transaction sets* associated with the statistics $(\mathcal{V}, \mathcal{I})$. As discussed earlier, each transaction contains a set of edges which take on the same min-hash identifier in $\mathcal{I}$ (for a particular hash sample from the k possible samples). This creates the transaction set $\mathcal{T}$. Then, the problem of determining edge sets with coherence at least α is as follows:

PROBLEM 1. *Determine all frequent patterns in* $\mathcal{T}$ *with support at least* $\alpha \cdot k$.

This part of the procedure is applied only when an analyst is required to classify a set of examples, and can be performed offline by using the currently available statistics $(\mathcal{V}, \mathcal{I})$. We note that since this is the standard

version of the frequent pattern mining problem, we can use any of the well known algorithms for frequent pattern mining. Furthermore, since the transaction data is typically small enough to be main-memory resident, we can use specialized frequent pattern mining algorithms which are very efficient for the case of main-memory resident data.

It remains to analyze the accuracy of this randomized approach. Because of the massive nature of the underlying data sets, we may generate a fairly large number of patterns. We would like to minimize the spuriously generated patterns in order to increase the effectiveness of our approach. Therefore, we increase the coherence threshold, so that we generate no false positives with high probability. Specifically, we set the coherence threshold to $\alpha \cdot (1+\gamma) \cdot k$, where $\gamma \in (0,1)$.

LEMMA 3.1. *The probability of a pattern P determined from $\mathcal{T}$ to be a false positive (based on coherence probability), when using a coherence threshold of $\alpha \cdot (1+\gamma)$ and k samples is given by at most $e^{-\alpha \cdot k \cdot \gamma^2/3}$, where e is the base of the natural logarithm.*

Proof. Let Y_i be a 0-1 variable which takes on the value of 1, if the pattern P is included in the ith min-hash sample. Thus, Y_i is a bernoulli random variable with probability less than α. The support of pattern P can be expressed as $\sum_{i=1}^{k} Y_i$. By using the Chernoff bound, we obtain the desired result.

The above result can be restated to determine the minimum number of samples required in order to guarantee the bound.

COROLLARY 3.1. *The number of samples k required in order to guarantee a probability at most δ for any of the determined patterns to be a false positive is given by $3 \cdot ln(1/\delta)/(\alpha \cdot \gamma^2)$.*

Next, we will study the further enhancement of our approach with a compressed virtual table.

3.2 Min Hash Approach with Compressed Virtual Table We note that the main constraint with the use of the above approach is that the summary statistics $\mathcal{V}$ and $\mathcal{I}$ have size $O(k \cdot L)$, where L is the number of distinct edges encountered so far. The problem here is that the value of L may be very large because of the *massive graph* assumptions of this paper. Such large summarizations cannot easily be maintained even on disk, and certainly not in main memory, which is required for an efficient update process in the data stream scenario. Therefore, we will supplement our min-hash row compression with a more direct column compression in order to further reduce the size of the required synopsis. This results in a simultaneous row and column compression, though the column compression is performed differently.

Algorithm *UpdateCompressedMinHash*(Incom. Edge: $e \in G$,
Sampling Rate: k, Min-hash values: $\mathcal{V}$
Min-hash indices: $\mathcal{I}$, Column Compression Size: n);
{ Assume that graph G is associated with a numerical identifier $Id(G)$ which denotes the order in which it was received in data stream; }
begin
Generate k random hash values $h(1, Id(G)) \ldots h(k, Id(G))$;
Map e onto an integer i_e in $[1, n]$ with the use of a uniform random hash function;
Apply same min-hash approach of of Figure 1 to this edge except that we apply to i_e instead of e;
end

Figure 2: Updating Min-hash Index for each Incoming Graph with Column Compression

The overall approach is similar to that of Figure 1, except that each edge in the data is first mapped to an integer in the range $[1, n]$ with the use of uniform random hash function, and then the steps of Figure 1 are applied. Thus, in this case, instead of determining patterns on the actual edges, we are determining patterns on integers in the range $[1, n]$. The overall approach is illustrated in Figure 2. Note that this algorithm differs from Figure 1 in the sense that it uses the column compression size n as an input, and uses it to map each edge onto an integer in $[1, n]$. This mapping is done with the use of a uniform random hash function. We construct a string representation of the edge by concatenating the node labels of the two ends. We apply the hash function to this string representation. Other than this, most of the steps in Figure 2 are the same as those in Figure 1.

The value of n is picked based in storage considerations, and is typically much lower than the number of distinct edges. The choice of n results from a natural tradeoff between storage size and accuracy. Since multiple edges may map onto the same integer, this approach may result in some further degradation of the accuracy of computation. Therefore, we will examine the level to which the edge coherence computation is *additionally* degraded *purely because of* column compression. Thus, the total degradation would be the product of the degradations because of row and column compression.

A key property of real data sets is that they are often sparse. This property can be leveraged in order to guarantee the effectiveness of column compression. Let us assume that the average number of edges contained in the different graphs in the data stream is given by a. The sparsity assumption implies that a is typically extremely small compared to the number of distinct edges and is also quite small compared to the

compressed column cardinality n. Therefore, we assume that $a << n$. We note that the coherence probability of pattern P is expressed as the ratio $f_\cap(P)/f_\cup(P)$. As long as both $f_\cap(P)$ and $f_\cup(P)$ are accurately estimated, the coherence probability is accurately estimated as well. We note that both $f_\cup(P)$ and $f_\cap(P)$ are over-estimated because all correct patterns are counted, whereas the hash result for some spurious edges may collide with some of the columns corresponding to P. Let $f'_\cap(P)$ and $f'_\cup(P)$ be the values of $f_\cap(P)$ and $f_\cup(P)$ when exhaustively counted over the column compressed table. Let $|P|$ denote the number of edges in the subgraph P. We make the following observations:

LEMMA 3.2. *Let $f'_\cup(P)$ be the estimated support of P on the column-compressed data with the use of a uniform hash functions. Then, the expected value of $f'_{cup}(P)$ satisfies the following relationship:*

$$f_\cup(P) \leq E[f'_\cup(P)] \leq f_\cup(P) + \frac{a \cdot |P|}{n} \tag{3.1}$$

Proof. We know that $f_\cup(P) \leq E[f'_\cup(P)]$, since all true patterns are counted, but some spurious patterns are also counted because of collisions.

In order to compute the fraction of spurious patterns, we need to compute the probability that a transaction contains spurious edges because of collisions. Let m_i be the number of edges in the ith graph. The probability that any of the m_i edges get mapped to one of the $|P|$ edges because of a collision is given by $m_i \cdot |P|/n$. Therefore, by summing this probability over all graphs, we obtain the expected over-estimation of the support. This is equal to $\frac{|P|}{n} \cdot \sum_{i=1}^{N} m_i/N$. We note that $\sum_{i=1}^{N} m_i/N$ is simply equal to a. By making the corresponding substitution in the expression, we obtain the desired result.

We can obtain a similar result for $f'_\cap(P)$.

LEMMA 3.3. *Let $f'_\cap(P)$ be the estimated support of P on the column-compressed data with the use of a uniform hash functions. Then, the expected value of $f'_{cap}(P)$ approximately satisfies the following relationship:*

$$f_\cap(P) \leq E[f'_\cap(P)] \leq f_\cap(P) + \frac{a \cdot |P|}{n} \tag{3.2}$$

Proof. The lower bound is trivially true since all true patterns are always counted.

For the case of the upper bound, we need to estimate the probability that a graph contains a proper subset Q of the pattern P ($Q \subset P$), but it gets counted as a true pattern because of collisions which spuriously map edges onto $P - Q$. We compute only the probability that a pattern which is one edge away from P gets counted as P, because in the case of $a << n$, the probability of multiple collisions is extremely small. There are $|P|$ such patterns, and by using a similar analysis to the previous proof, we can show that the probability of each such pattern being counted spuriously is given by a/n. Therefore, the overall probability of spurious over-counting is given by $\frac{a \cdot |P|}{n}$ as in the previous case.

The above results show that both $f_\cup(P)$ and $f_\cap(P)$ are very closely approximated by this approach. This is especially the case since useful values of $|P|$ are quite small and typically less than 4 or 5. For example, consider a data set with $a = 10$, $n = 10,000$, and $|P| = 10$. In such cases, both $f_\cap(P)$ and $f_\cup(P)$ are both approximated to within 0.01 of their true values.Therefore, the column compression scheme reduces the space requirements significantly at a relatively small degradation in quality.

3.3 Determining Discriminative Patterns In this section, we will discuss how the discriminative patterns are determined for classification. In order to determine the discriminative patterns, we also need to keep track of the class labels during the min-hashing scheme. A mentioned earlier, we assume that the class labels of the graphs are appended to the identifier $Id(G)$ for each graphs G in Figure 1. These class labels can be used in order to compute the discriminative patterns from the compressed summary of the data. We note that the global distribution of class labels in the min-hash summary *may not be the same* as the original data stream, because of its inherent bias in representing graph identifiers with larger number of edges in the summary transaction set $\mathcal{T}$. Therefore, we directly maintain the *global class distribution* while scanning the data stream, since the global statistics are required in order to compute the dominant interest ratio for any particular pattern. However, for a particular pattern containing a *fixed number of edges*, we can make the following observation:

OBSERVATION 3. *The class fraction for any particular pattern P and class computed over the transaction set $\mathcal{T}$ is an unbiased estimate of its true value.*

The above observation is true because the class distribution is not used during either the column compression or the row based min-hashing. Furthermore, we are comparing the class distribution among graphs containing only a fixed number of edges. Therefore, the probability of a particular graph being picked as the first graph containing that pattern in the min-hash scheme is exactly

the same. Therefore, while the overall distribution of classes will not be the same in the *global* min-hash summary, the distribution of classes *for a particular pattern* will be the same. As in the case of coherent patterns, we will re-define our thresholds in such a way so as to allow no false positives. We note that the constraint on the coherence probability ensures that at least $k \cdot \alpha$ patterns of each type are available in the min-hash index. This ensures that sufficient number of samples are available to measure the class-discrimination behavior as well. As in the case of the coherence threshold, we can set a dominant confidence threshold which is a factor $(1+\gamma)$ of its true value.

LEMMA 3.4. *The probability of a pattern P determined from $\mathcal{T}$ to be a false positive (based on class-confidence), when using a dominant confidence threshold of $\theta \cdot (1+\gamma)$ and k samples for the min-hash approach is given by at most $e^{-\alpha \cdot \theta \cdot k \cdot \gamma^2/3}$, where e is the base of the natural logarithm.*

Proof. Since each of the relevant patterns is coherent, we can assume that at least $\alpha \cdot k$ samples of that pattern are available in the min-hash index in order to estimate the class distribution of a particular pattern. In order for a pattern to be a false-positive, its class presence must be less than θ in the original data, but must have a presence greater than $\theta \cdot (1+\gamma)$ in the min-hash table. Therefore, the Chernoff bound implies that the estimation of the class fraction within a factor $(1+\gamma)$ of the true value with the use of at least $\alpha \cdot k$ samples has probability at least $e^{-\alpha \cdot \theta \cdot k \cdot \gamma^2/3}$, where e is the base of the natural logarithm.

We can immediate invert the result in order to determine the minimum number of samples required for a fixed probability δ.

COROLLARY 3.2. *The number of samples k required in order to guarantee a probability at most δ for any of the determined patterns to be a false positive (based on dominant class confidence) is given by $3 \cdot ln(1/\delta)/(\alpha \cdot \theta \cdot \gamma^2)$.*

Note that Corollary 3.1 differs from Corollary 3.2 in the sense that the latter has an additional factor $\theta < 1$ in the denominator. Therefore, the sample size k from Corollary 3.2 will dominate the required sample size from Corollary 3.1.

The min-hash structure is continuously maintained over the progress of the data stream. Since this structure is fairly compact, it can be used at any time during the progress of the data stream in order to classify a given set of test instances. This classification can easily be done offline, by using standard frequent pattern mining algorithms on $\mathcal{T}$ in order to determine the coherent patterns and then determining the discriminative patterns among these. In order to perform the classification, we determine all the α-coherent and θ-confident patterns in the data. These patterns are sorted in order of the dominant class confidence. For a given test graph, we determine the first r patterns which occur as a subgraph of the test graph. The majority vote among these patterns is reported as the relevant class label.

4 Experimental Results

In this section we will present the experimental techniques for the effectiveness and efficiency of our classification technique. Specifically, we will test over the following measures. We tested our technique for effectiveness, efficiency, and sensitivity to a variety of parameters. We used the following data sets to test our algorithms:

DBLP Data Set[1]**:** In this case, we constructed graphs on the data set in which authors were defined as nodes, and co-authorship in a particular paper was defined as an edge. Each paper constituted a graph. The class labels constituted the following three possibilities: **(1) Database Related Conferences (2) Data Mining Related Conferences**, and **(3) All remaining conferences**.

We note that there was considerable overlap in co-authorship structure across different classes, especially the first two. This made the classification problem particularly challenging. The final data set contained over $5 * 10^5$ authors and about $9.75 * 10^5$ edges (over all graphs) for the training data. This corresponds to about $3.55 * 10^5$ different graphs. Thus, even though the data set was large, the graphs were fairly sparse with a very small number of edges in each.

Sensor Streaming Data Set: This data contained information about local traffic on a sensor network which issued a set of intrusion attack types. Each graph constituted a local pattern of traffic in the sensor network. The nodes correspond to the IP-addresses, and the edges correspond to local patterns of traffic. We note that each intrusion typically caused a characteristic local pattern of traffic, in which there were some variations, but also considerable correlations. Each graph was associated with a particular intrusion-type. Since there were over 300 different intrusion types, this made the problem particularly difficult in terms of accurate classification. Our data set **Igraph0103-07** contained a stream of intrusion graphs from June 1, 2007 to June 3, 2007. The training

[1]Data Set URL: www.charuaggarwal.net/dblpcl/

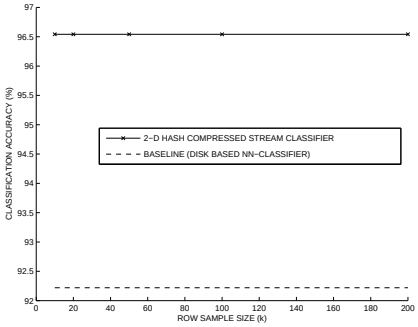

Figure 3: Effectiveness with different min-hash row sizes k for data set *DBLP* ; $n = 10000$, $\alpha = 0.05$, $\theta = 0.4$

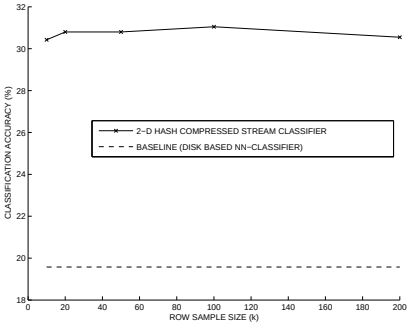

Figure 4: Effectiveness with different min-hash row sizes k for data set *Igraph0103-07* ; $n = 10000$, $\alpha = 0.05$, $\theta = 0.4$

data was constructed by uniformly sampling about 90% of the size of a larger base data set. The test data contained the remaining 10% of the size of the base data. The training data stream contained more than $1.57 * 10^6$ edges in the aggregate.

4.1 Effectiveness Results Since there are no known competing graph algorithms for the stream case, we designed a baseline which works for the disk resident case. While such a technique will not scale over massive data streams, it is useful to test the effectiveness of our technique over competing methods. Therefore, we designed a *disk-based nearest neighbor classifier* which scans the data from disk, re-organizes the edges into graphs, and then computes the closest graph to the test instance. We note that the approach needs two scans; one for reorganizing the edges into graphs, and another for determining the closest graph to the current target. Since the approach requires two scans, it can be used for disk-resident data, but not for data streams. Furthermore, the efficiency of the nearest neighbor approach reduces with increasing data set size; therefore, it cannot practically be used for very large data sets in an

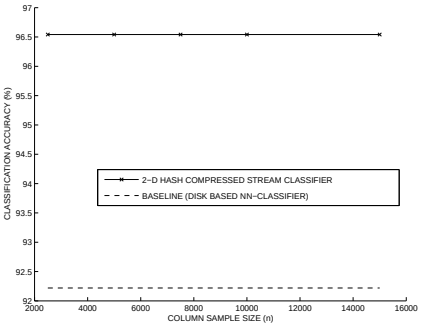

Figure 5: Effectiveness with different column hash sizes n for data set *DBLP* ; $k = 50$, $\alpha = 0.05$, $\theta = 0.4$

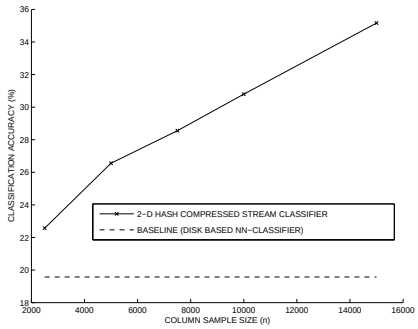

Figure 6: Effectiveness with different column hash sizes n for data set *Igraph0103-07* ; $k = 50$, $\alpha = 0.05$, $\theta = 0.4$

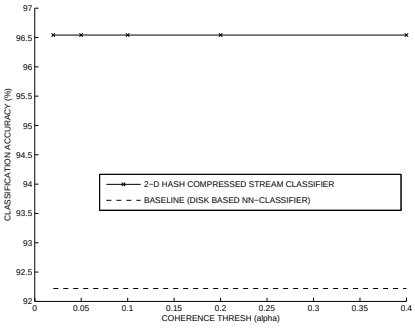

Figure 7: Effectiveness with different coherence thresholds α for data set *DBLP* ; $k = 50$, $n = 10000$, $\theta = 0.4$

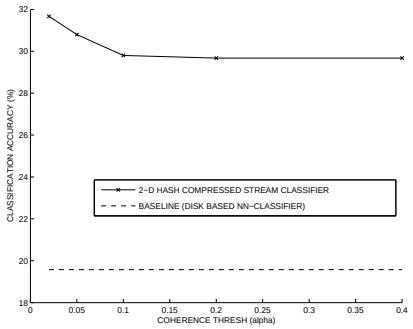

Figure 8: Effectiveness with different coherence thresholds α for data set *Igraph0103-07* ; $k = 50$, $n = 10000$, $\theta = 0.4$

efficient way. Nevertheless, the approach is useful as a baseline for testing the quality of the approach.

We tested the variation in the quality of our approach by changing various input parameters. Two kinds of parameters were varied: **(1) Compression parameters which determine the size of the data structure:** These include the row compression size k and column compression size n. Clearly, the use of larger row and column sizes can increase the effectiveness of the approach at the expense of greater space. We will examine this tradeoff in some detail. **(2) Variation in coherence threshold α and confidence threshold θ:** We will test the sensitivity of the approach with the parameters α and θ.

Unless otherwise mentioned, the default values of the parameters were $k = 50$, $n = 10000$, $\alpha = 0.05$ and $\theta = 0.4$. Furthermore, since our goal is to test the experimental effectiveness for different thresholds values of α and θ, we did not reset them with the use of the γ parameter. This means that γ was effectively set to 0, and this allowed a greater number of false positives over small values of k. We will show that even in this scenario, our technique provides very high quality results in terms of final classification. This shows that our technique provides great experimental robustness in addition to its theoretical properties. We further note that at the default value of the parameters, the space occupied by the min-hash structure is of the order of only 1MB. This can typically be captured in the memory limitations of even very space-constrained devices. In Figure 3 and 4, we have illustrated the variation in effectiveness of the scheme with increasing min-hash row size for the two data sets. The X-axis contains the min-hash row size k, while the Y-axis contains the classification accuracy. The other parameters n, α and θ were set to their default values discussed above. We have also illustrated the accuracy of the baseline nearest neighbor technique with the use of a horizontal line. We note that the parameter on the X-axis corresponds to a parameter of the min-hash scheme (only), and therefore the accuracy of the baseline technique does not vary across different values of the parameter. In all cases, the classification accuracy of the compressed hash-based classifier was much higher than the baseline nearest neighbor technique. This was in spite of the fact that the hash-based classifier can be used for streams of arbitrary length, whereas the baseline technique requires two passes, and can only be used with a disk resident database. Furthermore, the technique is not scalable to very large databases, because of its need to compute an increasing number of distance values with progression of the stream. In each of these cases, the classification accuracy was not very sensitive to the variation in the min-hash row size. The data set *DBLP* was the least sensitive, whereas the data set *Igraph0103-07* was somewhat more sensitive. This is because the sizes of the individual graphs in *Igraph0103-07* were larger. Therefore, the min-hash technique was able to take better advantage of the structural information in the graphs with increased row sizes. In spite of the differences in accuracies across the various data sets, it is clear that the min-hash technique was significantly superior to the baseline method in all cases. We note that the absolute classification accuracies should be understood in context of the fact that these data sets contained over 300 different classes. Therefore, this is an extreme case of classification into a large number of rare classes, and the absolute classification accuracies are quite high when we take this fact into consideration.

In Figures 5, and 6 we have illustrated the variations in the effectiveness of the scheme with different levels of column compression for the two data sets. The X-axis illustrates the increasing column sample size n, whereas the Y-axis illustrates the classification accuracy. As in the previous case, we have illustrated the (constant) baseline accuracy with a horizonal time, since the parameters on the X-axis relate to the min-hashing scheme only. It is evident that the variation in accuracy depends upon the compression scheme used. As in the previous case, there was very little variation in accuracy across different column sizes for the case of the *DBLP* data set. The variations are much more significant for the *Igraph0103-07* data set. We also note that the variations are more significant for the case of column compression as compared to row compression. This is partially because the column-wise hashing scheme can sometimes map edges with correlations to different classes to the same column. Such "mixing" can result in some reduction of the classification accuracy. However, in each case, the 2-d hash compressed scheme was

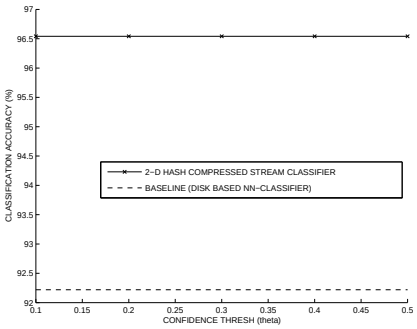

Figure 9: Effectiveness with different confidence thresholds θ for data set *DBLP* ; $k = 50$, $n = 10000$ $\alpha = 0.05$

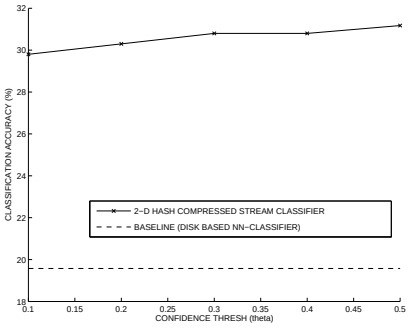

Figure 10: Effectiveness with different confidence thresholds θ for data set *Igraph0103-07* ; $k = 50$, $n = 10000$ $\alpha = 0.05$

much more accurate than the baseline nearest neighbor classifier.

We also tested the sensitivity of the approach with respect to the coherence and confidence thresholds. In Figures 7, and 8, we have illustrated the variation in classification accuracy with increasing coherence threshold for the two data sets. Picking coherence thresholds which were too high sometimes resulted in loss of information. Therefore, the classification accuracy reduced slightly with increasing coherence thresholds. However, the scheme was quite robust over different values of the coherence thresholds, and always significantly superior to the baseline classifier. In Figures 9 and 10, we have illustrated the variation in classification accuracy with increasing confidence threshold for the two data sets. As in the case of the coherence threshold, the different schemes were quite robust across different choices of the confidence threshold, and significantly superior to the baseline classifier accuracy. As in the previous cases, the difference between the hash-based classification scheme and the nearest neighbor classifier was quite significant.

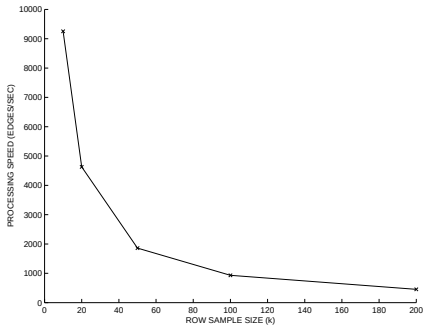

Figure 11: Processing rate (edges/sec) with varying row min-hash size k for data set *DBLP* ; $n = 10000$, $\alpha = 0.05$, $\theta = 0.4$

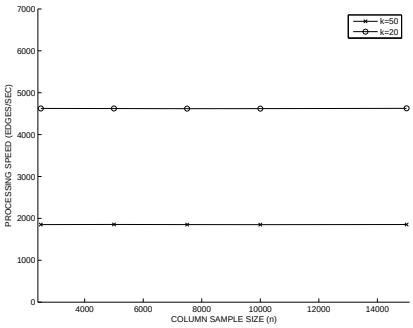

Figure 12: Processing rate (edges/sec) with varying column hash sizes n for data set *DBLP* ; $k = 20$ or $k = 50$, $\alpha = 0.05$, $\theta = 0.4$

4.2 Efficiency Results All tests were implemented on an IBM T61 Thinkpad running Window XP version 2.11, with a 2GHz CPU and 2GB of main memory. The algorithms were implemented with Microsoft Visual C++ 5.0. We note that the majority of the time was spent in processing the training stream and constructing the 2-d hash compressed data structure. The size of the compressed data structure was so small (typically about 1MB), that the algorithms for determining the frequent patterns from it required negligible time compared to the time for actually constructing the structure. This is because the process of structure construction requires us to process the entire data stream, which is several orders of magnitude larger than the structure itself. We further note that this portion of the running time is independent of the size of the data stream. Therefore, the overall running times were essentially insensitive to coherence and confidence parameters, and we will focus on the more interesting issue of determining the variations of the training rate (rate of 2-d hash structure updates) with different row and column sizes of the min-hash data structure. We further note that since the

processing rates of different data set were quite similar, we have presented the results only for the case of the DBLP data set.

In Figure 11, we have illustrated the processing rates for different values of the row size k, while keeping the column size constant at $n = 10000$. The row size k is illustrated on the X-axis, whereas the processing rate in edges per second is illustrated on the Y-axis. Since the time for processing each data point is linearly proportional to the number of min-hash rows, the corresponding processing rate is inversely proportional to the number of rows. We further note that at the default value of the parameter $k = 50$, the technique processed about 1750 edges per second. This is a very fast rate of processing for most practical applications.

We also tested the scalability of the technique with increasing hash-column size n. The results are illustrated in Figure 12. We have illustrated the results for two different row sizes of $k = 20$ and $k = 50$ respectively. The columns size n is illustrated on the X-axis, whereas the processing rate is illustrated on the Y-axis. The plots in Figure 12 suggest that both data sets showed very similar behavior. Furthermore, this behavior did not change very much for different column compression size n. This is essentially because the value of n only affected the size of the underlying data structure, but did not change the number of operations required for the algorithm. In each case, thousands of edges were processed per second. Thus, our graph classification approach is not only effective, but also efficient over a wide variety of practical scenarios.

5 Conclusions and Summary

In this paper, we provided a first approach to the problem of classification of massive graph streams. Graph streams in massive networks are a challenge not only because of the fast rate at which the streams arrive, but also because of the massive number of distinct edges which need to be tracked. This makes the problem very challenging from a computational and storage point of view. In this paper, we use a 2-dimensional hash based compression approach in order to efficiently mine significant subgraph patterns which are discriminative towards a particular class. These subgraph patterns are used to perform the classification of individual test graph instances. We present theoretical results which illustrate the effectiveness of mining the relevant subgraph patterns. We also implemented the hash-based classification algorithm, and show that it is extremely effective and efficient on a wide variety of real data sets.

References

[1] C. Aggarwal, N. Ta, J. Feng, J. Wang, and M. J. Zaki, *XProj: A Framework for Projected Structural Clustering of XML Documents*, KDD Conference, (2007).

[2] C. Aggarwal, Y. Li, P. Yu, and R. Jin, *On Dense Pattern Mining in Graph Streams*, VLDB Conference, (2010).

[3] C. Aggarwal, Y. Zhao, and P. Yu, *On Clustering Graph Streams*, SDM Conference, (2010).

[4] C. Aggarwal, and H. Wang, *Managing and Mining Graph Data*, Springer, (2010).

[5] C. Aggarwal, *Social Network Data Analytics*, Springer, (2011).

[6] C. Aggarwal, *Data Streams: Models and Algorithms*, Springer, (2007).

[7] D. Chakrabarti, and C. Faloutsos, *Graph Mining: Laws, Generators and Algorithms*, ACM Computing Surveys, 38(1), (2006).

[8] E. Cohen, M. Datar, S. Fujiwara, A. Gionis, P. Indyk, R. Motwani, J. Ullman, and C. Yang, *Finding Interesting Associations without Support Pruning*, IEEE TKDE, 13(1), (2001), pp. 64–78.

[9] M. Faloutsos, P. Faloutsos, and C. Faloutsos, *On power-law relationships of the internet topology*, ACM SIGCOMM Conference, (1999).

[10] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang, *Graph Distances in the Data Stream Model*, SIAM Jour. on Comp., 38(5), (2005), pp. 1709–1727.

[11] D. Gibson, R. Kumar, and A. Tomkins, *Discovering Large Dense Subgraphs in Massive Graphs*, VLDB Conference, (2005).

[12] H. He, and A. K. Singh, *Efficient Algorithms for Mining Significant Substructures in Large Graphs with Quality Guarantees*, ICDM Conference, (2007).

[13] H. Kashima, K. Tsuda, and A. Inokuchi, *Marginalized Kernels between Labeled Graphs*, ICML Conference, (2003).

[14] T. Kudo, E. Maeda, and Y. Matsumoto, *An Application of Boosting to Graph Classification*, NIPS Conf. (2004).

[15] S. Raghavan, and H. Garcia-Molina, *Representing web graphs*, ICDE Conference, (2003).

[16] S. Ranu, and A. K. Singh, *GraphSig: A Scalable Approach to Mining Significant Subgraphs in Large Graph Databases*, ICDE Conference, (2009).

[17] M. Rattigan, M. Maier, and D. Jensen, *Graph Clustering with Network Sructure Indices*. ICML, (2007).

[18] H. Wang, W. Fan, P. S. Yu, and J. Han, *Mining Concept-Drifting Data Streams using Ensemble Classifiers*, ACM KDD Conference, (2003).

[19] H. Wang, J. Yin, J. Pei, P. S. Yu, and J. X. Yu, *Suppressing Model Overfitting in Mining Concept-Drifting Data Streams*, ACM KDD Conference, (2006).

[20] X. Yan, and J. Han, *CloseGraph: Mining Closed Frequent Graph Patterns*, ACM KDD Conference, (2003).

[21] M. J. Zaki, and C. C. Aggarwal, *XRules: An Effective Structural Classifier for XML Data*, KDD Conf. (2003).