

On Clustering Graph Streams

Charu C. Aggarwal*

Yuchen Zhao†

Philip S. Yu‡

Abstract

In this paper, we will examine the problem of clustering massive graph streams. Graph clustering poses significant challenges because of the complex structures which may be present in the underlying data. The massive size of the underlying graph makes explicit structural enumeration very difficult. Consequently, most techniques for clustering multi-dimensional data are difficult to generalize to the case of massive graphs. Recently, methods have been proposed for clustering graph data, though these methods are designed for static data, and are not applicable to the case of graph streams. Furthermore, these techniques are especially not effective for the case of massive graphs, since a huge number of distinct edges may need to be tracked simultaneously. This results in storage and computational challenges during the clustering process. In order to deal with the natural problems arising from the use of massive disk-resident graphs, we will propose a technique for creating *hash-compressed micro-clusters* from graph streams. The compressed micro-clusters are designed by using a hash-based compression of the edges onto a smaller domain space. We will provide theoretical results which show that the hash-based compression continues to maintain bounded accuracy in terms of distance computations. We will provide experimental results which illustrate the accuracy and efficiency of the underlying method.

1 Introduction

In recent years, there has been a renewed focus on graph mining algorithms because of applications involving the web, bio-informatics, social networking and community detection. Numerous algorithms have been designed for graph mining applications such as clustering, classification, and frequent pattern mining [1, 4, 11, 10, 20]. A detailed discussion of graph mining algorithms may be found in [4].

The problem of clustering has been studied extensively in the data mining literature [13, 14, 22]. Recently, the problem has also been examined in the context of graph data [4, 11, 18]. The problem of clustering graphs has traditionally been studied in *node clustering of individual graphs*, in which we attempt to determine groups of nodes based on the density of linkage behavior. This problem has traditionally been studied in the context of graph-partitioning [15], minimum-cut determination [19] and dense subgraph

determination [12, 21].

Recently, the problem has also been studied in the context of *object clustering*, which we attempt to cluster many different *individual* graphs (as objects) [1, 10], which are defined on a base domain. This is distinct from the problem of node clustering in which the nodes are the entities to be clustered rather than the graphs themselves. However, the available techniques [1, 10] are designed for the most straight-forward case, in which the graphs are defined over a limited domain. Furthermore, it is assumed that the graph data sets are available on disk. This scenario arises in the context of certain kinds of XML data [1, 10], computational biology, or chemical compound analysis.

We study a much more challenging case in which the graphs are defined over a *massive domain of distinct nodes*. Furthermore, these graphs are not available at one time on disk, but are continuously received in the form of a stream. The node labels are typically drawn over a universe of distinct identifiers, such as the URL addresses in a web graph [17], an IP-address in a network application, or a user-id in a social networking application. Typically, the individual graphs constitute some kind of activity on the larger graph, such as the click-graph in a user web-session, the set of interactions in a particular time window in a social network, or the authorship graphs in a dynamically updated literature site. Often such graphs may individually be of modest size, though the *number of distinct edges may be very large on the aggregate data*. This property is referred to as that of *sparsity*, and is often encountered in a variety of real applications. This makes the problem much more challenging, because most clustering algorithms would require tracking the behavior of different nodes and edges. Since the number of possible edges may be of the order of the square of the number of nodes, it may often be difficult to explicitly store even modest summary information about the edges for all the incoming graphs. Clearly, such a problem becomes even more challenging in the stream scenario. Examples of such graph streams are as follows:

- The click-graph derived from a proxy-log in a user-session is typically a sparse graph on the underlying web graph.
- The interactions between users in a particular time-window in a social network will typically be a collec-

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

†University of Illinois at Chicago, yzhao@cs.uic.edu

‡University of Illinois at Chicago, psyu@cs.uic.edu

tion of disjointed graphs.

- The authorship graphs of individual articles in a large scientific repository will be small graphs drawn across millions of possible authors.

The currently available algorithms are designed either for the case of node-entity clustering, or for the case in which we have disk-resident data sets over a limited domain [1, 10]. Such algorithms require multiple passes over the data [1, 10], and are not applicable to the case of data streams. Furthermore, these algorithms do not scale very well with the underlying domain size. While the problem of stream clustering has been studied extensively in the context of multi-dimensional data [2, 7], there are no known algorithms for the case of clustering graph streams. In this paper, we will propose the first algorithm for clustering graph streams. We use a model in which a large number of graphs are received continuously by the data stream. It is also assumed that the *domain size* of the nodes is very large, and this makes the problem much more challenging. Thus, the large domain size of the stream and the structural characteristics of graphs data pose *additional challenges* over those which are caused by the data stream scenario.

In summary, the problem of clustering graph streams is particularly difficult because of the following reasons:

- Most graph mining algorithms use sub-structural analysis in order to perform the clustering. This may be difficult in the case of data streams because of the one-pass constraint.
- The number of possible edges scales up quadratically with the number of nodes. As a result, it may be difficult to explicitly hold the summary information about the massive graphs for intermediate computations.
- The individual graphs from the stream may exhibit the *sparsity property*. In other words, the graphs may be drawn from a large space of nodes and edges, but each individual graph in the stream may contain only a very small subset of the edges. This leads to representational problems, since it may be difficult to keep even summary statistics on the large number of edges. This also makes it difficult to compare graphs on a pair-wise basis.

In this paper, we propose the concept of *sketch-based micro-clusters* for graph data. The broad idea in sketch-based micro-clusters is to combine the idea of sketch-based compression with micro-cluster summarization. This approach helps in reducing the size of the representation of the underlying micro-clusters, so that they are easy to store and use. We will also show that the approach continues to maintain bounded accuracy for the underlying computations.

This paper is organized as follows. The remainder of this section discusses related work and contributions. In the next section, we will introduce the graph clustering problem, and the broad framework which is used to solve this problem. We will introduce the concept of graph micro-clusters, and how they can be used for the clustering process. Section 3 discusses the extension of these techniques with the use of sketch-based structures. Section 4 contains the experimental results. Section 5 contains the conclusions and summary.

1.1 Related Work and Contributions Graph clustering algorithms can be either of the *node clustering variety* in which we have single large graph and we attempt to cluster the nodes into groups of densely connected nodes. The second class of algorithms is the *object clustering variety*, wherein we have many graphs which are drawn from a base domain, and these different graphs are clustered together based on their structural similarity. In this paper, we will focus on the latter class of problems.

The case of node-clustering algorithms traditionally been studied in the context of the minimum cut problem [19], graph partitioning [15], network structure clustering [8, 16, 18], and dense subgraph determination [12, 21]. In particular, the techniques for dense subgraph determination [6, 12] use min-hash techniques in order to summarize massive graphs, and then use these summaries in order to cluster the underlying nodes. However, these techniques are limited to *node clustering of individual graphs*, rather than the clustering of a stream of many graph objects.

Recently, methods have been studied for clustering graphs as objects in the context of XML data [1, 10]. However these techniques have two shortcomings: (1) These techniques are not designed for the case when the nodes are drawn from a massive domain of possibilities. When the number of nodes is very large, the number of distinct edges may be too large to track effectively. (2) The available techniques are designed for disk-resident data, rather than graph streams in which the individual objects are continuously received over time. This paper will achieve both goals. This paper provides the first time- and space-efficient algorithm for clustering graph streams.

Space-efficiency is an important concern in the case of massive graphs, since the intermediate clustering data cannot be easily stored for massive graphs. This goal is achieved by performing a hash-compression of the underlying micro-clusters. We will show that the hash-compression technique does not lose significant effectiveness during the clustering process. We will show that the additional process of hash compression does not lose any effectiveness for the clustering process. We will illustrate the effectiveness of the approach on a number of real and synthetic data sets.

2 Graph Stream Clustering Framework

In this section, we will introduce the *GMicro* framework which is used for clustering graph streams. We assume that we have a node set \mathcal{N} over which the different graphs are defined. We note that each graph in the stream is typically constructed over only a small subset of the nodes. We assume that each element in \mathcal{N} is a string which defines the label of the corresponding node. For example, in the case of an intrusion application, each of the strings in \mathcal{N} correspond to the IP-addresses. The nodes of the incoming graphs $G_1 \dots G_k \dots$ are each drawn on the subset of nodes \mathcal{N} .

For purposes of notation, we assume that the set of distinct edges across all graphs are denoted by $(X_1, Y_1) \dots (X_i, Y_i) \dots$ respectively. Each X_i and Y_i is a node label drawn from the set \mathcal{N} . We note that this notation implicitly assumes directed graphs. In the event of an undirected graph, we assume that lexicographic ordering on node labels is used in order to convert the undirected edge into a directed edge. Thus, the approach can also be applied to undirected graphs by using this transformation. We also assume that the frequency of the edge (X_i, Y_i) in graph G_r is denoted by $F(X_i, Y_i, G_r)$. For example, in a telecommunication application, the frequency may represent the number of minutes of phone conversation between the edge-pair (X_i, Y_i) . In many applications, this frequency may be implicitly assumed to be 1, though we work with the more general case of arbitrary frequencies. We note that when the node set \mathcal{N} is very large, the total number of distinct edges received by the data stream may be too large to even enumerate on disk. For example, for a node set of 10^7 nodes, the number of distinct edges may be more than 10^{13} . This may be too large to explicitly store within current disk resident constraints. The graph clustering framework requires us to cluster the graphs into a group of k clusters $\mathcal{C}_1 \dots \mathcal{C}_k$, such that each graph from the data stream is dynamically assigned to one of the clusters in real time.

While micro-clustering [2] has been used in order to cluster multi-dimensional data, we will construct micro-clusters which are specifically tailored to the problem of graph data. In this paper, we will use a sketch-based approach in order to create *hash-compressed micro-cluster representations* of the clusters in the underlying graph stream. First, we will discuss a more straightforward representation of the uncompressed micro-clusters, and the storage and computational challenges in maintaining these representations. Then, we will discuss how the sketch-based techniques can be used in order to effectively deal with these challenges. We will show that the sketch-based techniques can be used in combination with the micro-clusters to construct the distance functions approximately over the different clusters.

Next, we will introduce a more straightforward and direct representation of graph-based micro-clusters. Let us

consider a cluster \mathcal{C} containing the graphs $\{G_1 \dots G_n\}$. We assume that the implicit graph defined by the summation of the graphs $\{G_1 \dots G_n\}$ is denoted by $H(\mathcal{C})$. Then, we define the micro-cluster $GCF(\mathcal{C})$ as follows:

DEFINITION 1. *The micro-cluster $GCF(\mathcal{C})$ is defined as the set $(L(\mathcal{C}), GCF2(\mathcal{C}), GCF1(\mathcal{C}), n(\mathcal{C}), T(\mathcal{C}))$, with size $(3 \cdot |L(\mathcal{C})| + 2)$, where $L(\mathcal{C})$ is the set of edges in micro-cluster \mathcal{C} . The individual components of $GCF(\mathcal{C})$ are defined as follows:*

- $L(\mathcal{C})$ is a set which contains a list of all the distinct edges in any of the graphs G_i in \mathcal{C} .
- $GCF2(\mathcal{C})$ is a vector of second moments of the edges in $L(\mathcal{C})$. Consider an edge $(X_q, Y_q) \in L(\mathcal{C})$. Then the corresponding second moment for that edge is defined as $\sum_{r=1}^n F(X_q, Y_q, G_r)^2$. We note that the value of $F(X_q, Y_q, G_r)$ is implicitly assumed to be zero, when (X_q, Y_q) is not present in the graph G_r . We refer to the second moment value for (X_q, Y_q) as $J(X_q, Y_q, H(\mathcal{C}))$.
- $GCF1(\mathcal{C})$ is a vector of first moments of the edges in L . Consider an edge $(X_q, Y_q) \in L(\mathcal{C})$. Then the corresponding first moment for that edge can be computed as $F(X_q, Y_q, H(\mathcal{C})) = \sum_{r=1}^n F(X_q, Y_q, G_r)$. This is because the first moment is simply the same as the definition of the frequency of the edge, except that the underlying graph (final argument of the function $F()$) is $H(\mathcal{C})$.
- The number of graphs in the micro-cluster \mathcal{C} is denoted by $n(\mathcal{C})$.
- The time stamp is of the last graph which was added to the cluster \mathcal{C} is denoted by $T(\mathcal{C})$.

One observation about micro-clusters is that for two clusters \mathcal{C}_1 and \mathcal{C}_2 , the value of $GCF(\mathcal{C}_1 \cup \mathcal{C}_2)$ can be computed as a function of $GCF(\mathcal{C}_1)$ and $GCF(\mathcal{C}_2)$. This is because the list $L(\mathcal{C}_1 \cup \mathcal{C}_2)$ is the union of the lists $L(\mathcal{C}_1)$ and $L(\mathcal{C}_2)$. Similarly, the frequencies may be obtained by pairwise addition, and $n(\mathcal{C}_1 \cup \mathcal{C}_2)$ may be determined by examining the size of the set $L(\mathcal{C}_1 \cup \mathcal{C}_2)$. The value of $T(\mathcal{C}_1 \cup \mathcal{C}_2)$ may be determined by computing the minimum of $T(\mathcal{C}_1)$ and $T(\mathcal{C}_2)$. We refer to this property as *additive separability*.

PROPERTY 2.1. *The graph micro clusters satisfy the additive separability property. This means that the micro-cluster statistics for $GCF(\mathcal{C}_1 \cup \mathcal{C}_2)$ can be computed as a function of $GCF(\mathcal{C}_1)$ and $GCF(\mathcal{C}_2)$.*

We note that graph micro-clusters also satisfy a limited version of the *subtractivity property*. All components of $GCF(\mathcal{C}_1 - \mathcal{C}_2)$ can be computed as a function of $GCF(\mathcal{C}_1)$ and $GCF(\mathcal{C}_2)$. We summarize as follows:

Algorithm *GMicro*(Number of Clusters: k)
begin
 $\mathcal{M} = \{\}$;
{ \mathcal{M} is the set of micro-cluster statistics }
repeat
Receive the next stream graph $\overline{G_r}$;
If less than k clusters currently exist, then
create micro-cluster statistics for singleton
graph G_r , and insert it into set of
micro-clusters \mathcal{M} ;
if k micro-clusters exist, then compute
edge structure similarity of
 $\overline{G_r}$ to each micro-cluster
in \mathcal{M} ;
if graph G_r lies outside the structure spread of
closest micro-cluster then replace the
least recently updated cluster with new
singleton cluster containing only G_r ;
else add $\overline{G_r}$ to the
statistics of the closest micro-cluster;
until data stream ends;
end

Figure 1: The *GMicro* Algorithm

PROPERTY 2.2. *The graph micro-clusters satisfy a limited version of the subtractivity property.*

We note that the size of graph micro-clusters may increase over time. This is different from the multidimensional case. This difference is because the number of edges in the list $L(\mathcal{C})$ will grow as more and more graphs are added to the data stream. As the size of $L(\mathcal{C})$ increases, the space-requirements increase as well. Furthermore, the computational complexity of distance computations also depends upon $L(\mathcal{C})$. Since the number of possible distinct edges in $L(\mathcal{C})$ is large, this will result in unmanageable space- and time-complexity with progression of the data stream. Therefore, we need a technique to further reduce the size of the micro-cluster representation. However, for simplicity, we will first describe the technique without the use of the compressed representation. This broad framework is retained even when sketch based compression is used. Therefore, we will first describe the use of the raw micro-cluster statistics in order to cluster the incoming graphs. Then, we will describe how sketch methods are used in order to make changes to specific portions of the micro-cluster representation and corresponding computations.

The micro-clustering algorithm uses the number of micro-clusters as input. We refer to the the Graph MICRO-clustering algorithm as the *GMicro* method. Initially, the clustering algorithm starts off with the null set of micro-clusters. As new graphs arrive from the data stream, they are added to the data as singleton micro-clusters. Thus, the first k graph records are created as singleton micro-clusters. While this may not distinguish well between the records in

the initial clustering, the steady state behavior is impacted by the subsequent steps of the clustering process. Subsequently, when the next graph G_r arrives, the distance to each micro-cluster centroid is computed. The distance measure that we compute is constructed as a variant on the L_2 -distance measure for multi-dimensional data and can be computed with the use of micro-cluster statistics. We assign the incoming record G_r to the closest centroid based on this structural distance computation. However, in some cases, an incoming data point in an evolving data stream may not fit well in any cluster. In order to handle this case, we check if the *structural spread* of the picked micro-cluster is less than the distance of G_r to that micro-cluster. The structural spread is defined as a factor¹ of the mean-square radius $S(\mathcal{C})$ of the elements of micro-cluster \mathcal{C} from its centroid. We also apply the spread criterion to only clusters which contain at least *min_init* points. As in the case of the structural similarity measure, we will see that the spread can also be computed as a function of the micro-cluster statistics. If the structural spread is less than the distance of G_r to the closest micro-cluster, then we create a new singleton micro-cluster containing only G_r . This micro-cluster replaces the most stale (least recently updated) micro-cluster. The information on the last update time of the micro-clusters is available from the corresponding time stamps in the micro-cluster statistics. The overall algorithm is illustrated in Figure 1.

2.1 Computing Edge Structural Similarity and Spread

Since we are dealing with the case of sparse structural data, we need to normalize for the frequency of edges included in both the incoming record and the centroid to which the similarity is being computed. We note that a micro-cluster can also be considered a pseudo-graph for which the frequencies of the corresponding edges are defined as the sum of the corresponding frequencies of the edges from which the micro-cluster was created. Therefore, we simply need to define a distance (similarity) function between two graphs in order to compute the similarity between micro-clusters and centroids. Let \mathcal{C} be a given micro-cluster which contains the graphs $G_1 \dots G_n$. Let the implicit graph for the micro-cluster \mathcal{C} (corresponding to the summation of the graphs $G_1 \dots G_n$) be denoted by $H(\mathcal{C})$. Let the corresponding normalized graph (by dividing each edge frequency of $H(\mathcal{C})$ by $n(\mathcal{C})$) be denoted by $\overline{H(\mathcal{C})}$. Thus, $\overline{H(\mathcal{C})}$ represents the centroid graph of all the graphs in the micro-cluster. Let the edges in $L(\mathcal{C})$ be denoted by $(X_1, Y_1) \dots (X_m, Y_m)$. Let G_t be the incoming graph. Then, the L_2 -distance $L2Dist(G_t, \overline{H(\mathcal{C})})$ between the graphs G_t and $\overline{H(\mathcal{C})}$ is defined as follows:

$$L2Dist(G_t, \overline{H(\mathcal{C})}) = \sum_{i=1}^m \left(F(X_i, Y_i, G_t) - \frac{F(X_i, Y_i, H(\mathcal{C}))}{n(\mathcal{C})} \right)^2$$

¹We pick this factor to be 3 by normal distribution assumption.

A second possibility for the computation of the similarity function is the *dot product*. Unlike, the L_2 -distance function, higher values imply greater similarity. The dot product $Dot(G_t, \overline{H(\mathcal{C})})$ between the graphs G_t and $\overline{H(\mathcal{C})}$ is defined as follows:

$$Dot(G_t, \overline{H(\mathcal{C})}) = \sum_{i=1}^m F(X_i, Y_i, G_t) \cdot \frac{F(X_i, Y_i, H(\mathcal{C}))}{n(\mathcal{C})}$$

Next, we will define the structural spread of a given cluster. Since the structural spread is defined as a function of the mean-square radius, we will first define the mean square radius of the micro-cluster \mathcal{C} . Then, the mean-square radius $S(\mathcal{C})$ of \mathcal{C} is defined as follows:

$$S(\mathcal{C}) = \frac{1}{n} \sum_{j=1}^n L2Dist(G_j, \overline{H(\mathcal{C})})$$

$$= \frac{1}{n} \cdot \sum_{j=1}^n \sum_{i=1}^m \left(F(X_i, Y_i, G_j) - \frac{F(X_i, Y_i, H(\mathcal{C}))}{n(\mathcal{C})} \right)^2$$

The spread is defined as a factor² t multiplied with $S(\mathcal{C})$. Both the structural distance measure and the structural spread can be computed as a function of the micro-cluster statistics. This is because the value of $F(X_q, Y_q, H(\mathcal{C}))$ is directly available from the first-order statistics of micro-cluster \mathcal{C} . Similarly, the value of $n(\mathcal{C})$ is included in the micro-cluster statistics. Therefore, we summarize as follows:

LEMMA 2.1. *The structural similarity measures denoted by $L2dist(G_t, \overline{H(\mathcal{C})})$ and $Dot(G_t, \overline{H(\mathcal{C})})$ between G_t and the centroid graph $\overline{H(\mathcal{C})}$ for cluster \mathcal{C} can be computed from the micro-cluster statistics.*

The spread $S(\mathcal{C})$ can also be directly computed from the micro-cluster statistics. The corresponding value can be obtained by simplifying the expression for $S(\mathcal{C})$.

LEMMA 2.2. *The structural spread $S(\mathcal{C})$ can be computed from the micro-cluster statistics.*

Proof. Let us denote $F(X_i, Y_i, H(\mathcal{C}))/n(\mathcal{C})$ by p_i . This represents the average frequency of the edges for the centroid of the micro-cluster. As discussed earlier, this can be computed directly from the micro-cluster statistics. By substituting in the expression for $S(\mathcal{C})$, we get the following:

$$(2.1) \quad S(\mathcal{C}) = \frac{1}{n(\mathcal{C})} \sum_{j=1}^n \sum_{i=1}^m (F(X_i, Y_i, G_j) - p_i)^2$$

²We use $t = 3$ in accordance with the normal distribution assumption.

By expanding the expression for $S(\mathcal{C})$, we can simplify as follows:

$$S(\mathcal{C}) = \frac{\sum_{i=1}^m J(X_i, Y_i, H(\mathcal{C}))}{n(\mathcal{C})} - 2 \sum_{i=1}^m \frac{p_i \cdot \sum_{j=1}^n F(X_i, Y_i, G_j)}{n(\mathcal{C})} + \sum_{i=1}^m p_i^2$$

$$= \frac{\sum_{i=1}^m J(X_i, Y_i, H(\mathcal{C}))}{n(\mathcal{C})} - 2 \cdot \sum_{i=1}^m p_i \cdot p_i + \sum_{i=1}^m p_i^2$$

$$= \frac{\sum_{i=1}^m J(X_i, Y_i, H(\mathcal{C}))}{n(\mathcal{C})} - \sum_{i=1}^m p_i^2$$

We note that all the terms in the above definition are drawn from the micro-cluster definition. Therefore, the spread $S(\mathcal{C})$ can be computed directly from the micro-cluster statistics.

3 Sketch Based Micro-cluster Compression

The storage and computational requirements for updating the micro-clusters depend upon the number of edges in them. The number of edges in the micro-clusters will typically increase as new graphs arrive in the stream. As the size of the micro-cluster increase, so does the time for computing the distance functions of incoming data points from the clusters. When the domain size is extremely large, the number of distinct edges can be too large for the micro-cluster statistics to be maintained explicitly. For example, consider the case when the number of possible nodes is 10^7 . This is often the case in many real applications such as IP-network data. Since the number of possible edges may be as large as 10^{13} , the size of the micro-cluster statistics may exceed the disk limitations, after a sufficient number of graphs are received. This leads to challenges in storage and computational efficiency.

Sketch based approaches [5, 9] were designed for enumeration of different kinds of frequency statistics of data sets. A commonly-used sketch is the count-min method [9]. In this sketch, we use $w = \lceil \ln(1/\delta) \rceil$ pairwise independent hash functions, each of which map onto uniformly random integers in the range $h = [0, e/\epsilon]$, where e is the base of the natural logarithm. The data structure itself consists of a two dimensional array with $w \cdot h$ cells with a length of h and width of w . Each hash function corresponds to one of w 1-dimensional arrays with h cells each. In standard applications of the count-min sketch, the hash functions are used in order to update the counts of the different cells in this 2-dimensional data structure. For example, consider a 1-dimensional data stream with elements drawn from a massive set of domain values. When a new element of the data stream is received, we apply each of the w hash functions to

map onto a number in $[0 \dots h - 1]$. The count of each of the set of w cells is incremented by 1. In the event that each item is associated with a frequency, the count of the corresponding cell is incremented by the corresponding frequency. In order to *estimate* the count of an item, we determine the set of w cells to which each of the w hash-functions map, and determine the minimum value among all these cells. Let c_t be the true value of the count being estimated. We note that the estimated count is at least equal to c_t , since we are dealing with non-negative counts only, and there may be an over-estimation because of collisions among hash cells. As it turns out, a probabilistic upper bound to the estimate may also be determined. It has been shown in [9], that for a data stream with T arrivals, the estimate is at most $c_t + \epsilon \cdot T$ with probability at least $1 - \delta$. The sketch can also be used in order to estimate the frequencies of groups of items by using these same approach. The count-min sketch can be used in order to estimate the frequency behavior of individual edges by treating each edge as an item with a unique string value. We note that each edge (X_i, Y_i) can be treated as the string $X_i \oplus Y_i$ where \oplus is the concatenation operator on the node label strings X_i and Y_i . This string can be hashed into the table in order to maintain the statistics of different edges. The corresponding entry is incremented by the frequency of the corresponding edge.

We can use the sketch based approach in order to construct the sketched micro-cluster. The idea is that the portions of the micro-cluster whose size is proportional to the number of edges are not stored explicitly, but implicitly in the form of sketch table counts. We will then see how well the individual components of the micro-cluster representation are approximated. Since all clustering computations can be performed in terms of micro-cluster statistics, it follows that the clustering computations can be effectively performed as long as the underlying micro-cluster statistics can be approximated. The compressed micro-clusters are defined as follows:

DEFINITION 2. *The micro-cluster $GCF(\mathcal{C})$ is defined as the set $(GSketch(\mathcal{C}), R(\mathcal{C}), n(\mathcal{C}), T(\mathcal{C}))$ of size $(e/\epsilon) \cdot \ln(1/\delta) + 3$. The individual components of $GCF(\mathcal{C})$ are defined as follows:*

- *The data structure $GSketch(\mathcal{C})$, contains a sketch-table of all the frequency-weighted graphs which are included in the micro-cluster. This requires a table with size $(e/\epsilon) \cdot \ln(1/\delta)$. The actual micro-cluster update is performed as follows. For each edge (X_i, Y_i) for an incoming graph, we compute the concatenation string $X_i \oplus Y_i$, and hash it into the table with the use of w hash functions. We add the frequency of the incoming edge to the corresponding w entries.*
- *We maintain $R(\mathcal{C}) = \sum_{i=1}^m J(X_i, Y_i, H(\mathcal{C}))$ explicitly. This is done by adding the square of the frequency of*

the incoming edge to $R(\mathcal{C})$.

- *The number of graphs in the micro-cluster \mathcal{C} is denoted by $n(\mathcal{C})$.*
- *The time stamp is of the last graph which was added to the cluster \mathcal{C} is denoted by $T(\mathcal{C})$.*

The above definition implies that a *separate sketch-table* is maintained with each micro-cluster. It is important to note that we use the same set of corresponding hash functions for each sketch table. This is important in order to compute important statistics about the micro-clusters such as the dot-product.

The *GMicro* algorithms can be used with this new definition of micro-clusters except that the intermediate computations may need to be performed as sketch-based estimates. In the remaining portion of this section, we will discuss how these estimates are computed, and the accuracy associated with such computation. We note that the first-order and second-order statistics are implicitly coded in the sketch table. Let $W(\mathcal{C})$ be the sum of the edge frequencies in $H(\mathcal{C})$. The sketch encodes implicit information about the micro-clusters. The first-order and second-order statistics can be estimated as follows:

- $F(X_i, Y_i, H(\mathcal{C}))$ can be estimated by hashing $X_i \oplus Y_i$ into the hash table with the use of the w hash functions, and computing the minimum of the corresponding entries. It can be directly shown [9] that the corresponding estimate $\hat{F}(X_i, Y_i, H(\mathcal{C}))$ lies in the range $[F(X_i, Y_i, H(\mathcal{C})), F(X_i, Y_i, H(\mathcal{C})) + \epsilon \cdot W(\mathcal{C})]$ with probability at least $1 - \delta$.
- We note that $J(X_i, Y_i, H(\mathcal{C}))$ cannot be estimated effectively. One possibility is to compute an estimate on $J(X_i, Y_i, H(\mathcal{C}))$ by computing the minimum of the *square of the* corresponding entries for $X_i \oplus Y_i$ in the sketch table. The bound on this estimate is quite loose, and therefore we will not use the approach of estimating $J(X_i, Y_i, H(\mathcal{C}))$. The additional information $R(\mathcal{C}) = \sum_{i=1}^m J(X_i, Y_i, \mathcal{C})$ (maintained in the sketch statistics) is sufficient to perform the intermediate computations for clustering.

The above observations emphasize that intermediate information on the micro-cluster statistics can be constructed approximately with the sketch technique. This is useful, since all the important properties of the clusters are encoded in the micro-cluster statistics. For example, if the behavior of the different portions of the graph (or specific edges) need to be examined in the context of different clusters, the corresponding micro-cluster statistics need to be derived. Next, we will discuss the estimation of important quantitative computations which are performed during the clustering process.

3.1 Distance estimations We will expand the expression for $L2Dist$ in order to express it in terms of sketch-based micro-cluster statistics:

$$\begin{aligned} L2Dist(G_t, \overline{H(\mathcal{C})}) &= \sum_{i=1}^m \left(F(X_i, Y_i, G_t) - \frac{F(X_i, Y_i, H(\mathcal{C}))}{n(\mathcal{C})} \right)^2 \\ &= \sum_{i=1}^m F(X_i, Y_i, G_t)^2 - 2 \sum_{i=1}^m \frac{F(X_i, Y_i, G_t) \cdot F(X_i, Y_i, H(\mathcal{C}))}{n(\mathcal{C})} + \\ &\quad + \sum_{i=1}^m \frac{F(X_i, Y_i, H(\mathcal{C}))^2}{n(\mathcal{C})^2} \end{aligned}$$

All of the three expressions in the above expansion are dot products. Of these dot products, the value of the expression $\sum_{i=1}^m F(X_i, Y_i, G_t)^2$ can be computed exactly, by using the statistics of the incoming graph G_t . The other two dot products are estimated using the technique discussed for [9]. Specifically, in the second term, $F(X_i, Y_i, G_t)$ can be computed exactly, while $F(X_i, Y_i, H(\mathcal{C}))$ can be computed directly from the sketch table $GSketch(\mathcal{C})$. We perform a pairwise multiplication for each edge appeared in the incoming graph G_t , and use the sum to estimate $\sum_{i=1}^m F(X_i, Y_i, G_t) \cdot F(X_i, Y_i, H(\mathcal{C}))$ in the second term. The value of the third term ($\sum_{i=1}^m F(X_i, Y_i, H(\mathcal{C}))^2$) can be estimated by performing the dot product between two copies of $GSketch(\mathcal{C})$. There is a one-to-one correspondence among the cells of both copies. We perform pairwise dot products for the w different rows in the sketch table, and pick the minimum of these values as the estimate.

Next, we will bound the quality of the distance estimation. Since the distance estimation is expressed as a function of individual dot-products (which can themselves be bounded), this also helps in bounding the overall quality of the estimation. Let $V(G_t)$ be the sum of the frequencies of the edges in G_t . Then, we can show the following:

LEMMA 3.1. *With probability at least $(1 - 2 \cdot \delta)$, the estimate of $L2Dist(G_t, \overline{H(\mathcal{C})})$ with the use of the sketch-based approach lies in the range $[L2Dist(G_t, \overline{H(\mathcal{C})}) - 2 \cdot \epsilon \cdot V(G_t) \cdot W(\mathcal{C})/n(\mathcal{C}), L2Dist(G_t, \overline{H(\mathcal{C})}) + \epsilon \cdot W(\mathcal{C})^2/n(\mathcal{C})^2]$.*

Proof. We note that the computation of $L2Dist$ requires the estimation of two terms, which have opposite effects on the overall estimate. Each extreme case is when one of the terms is estimated as exactly as possible, and the other is overestimated as much as possible. We deal with these cases below:

Extreme Case I: $\sum_{i=1}^m F(X_i, Y_i, H(\mathcal{C}))^2$ is exactly estimated, but $F(X_i, Y_i, H(\mathcal{C})) \cdot F(X_i, Y_i, G_t)$ is over-estimated: This forms the lower bound for the range, since the over-estimated term has a negative sign attached before it. We know from [9], that the over-estimation of this dot product is no more than $\epsilon \cdot V(G_t) \cdot W(\mathcal{C})$ with probability at least $(1 - \delta)$. Scaling by the constant $2/n(\mathcal{C})$ to account for the constant factor in front of the term, we derive that the lower bound is at least $L2Dist(G_t, \overline{H(\mathcal{C})}) - 2 \cdot \epsilon \cdot V(G_t) \cdot$

$W(\mathcal{C})/n(\mathcal{C})$ with probability at least $1 - \delta$.

Extreme Case II: $F(X_i, Y_i, H(\mathcal{C})) \cdot F(X_i, Y_i, G_t)$ is exactly estimated, but $\sum_{i=1}^m F(X_i, Y_i, H(\mathcal{C}))^2$ is over-estimated: This forms the upper bound for the range. As in the previous case, we can show that the level of over-estimation is at most $\epsilon \cdot W(\mathcal{C})^2$ with probability at least $1 - \delta$. Scaling by the factor $1/n(\mathcal{C})^2$ to account for the constant factor in front of the term, we derive that the upper bound is at most $L2Dist(G_t, \overline{H(\mathcal{C})}) + \epsilon \cdot W(\mathcal{C})^2/n(\mathcal{C})^2$ with probability at least $1 - \delta$.

Since the bounds for either of the two cases are violated with probability at most δ , the probability of neither bound being violated is at least $1 - 2 \cdot \delta$. This proves the result.

The above result can also be used for dot-product estimation.

LEMMA 3.2. *With probability at least $(1 - \delta)$, the estimate of $Dot(G_t, \overline{H(\mathcal{C})})$ with the use of the sketch-based approach lies in the range $[Dot(G_t, \overline{H(\mathcal{C})}), Dot(G_t, \overline{H(\mathcal{C})}) + \epsilon \cdot V(G_t) \cdot W(\mathcal{C})/n(\mathcal{C})]$.*

Proof. This follows directly from the dot product results in [9].

3.2 Estimation of Spread In this section, we will discuss the estimation of the spread $S(\mathcal{C})$ with the sketch-based approach. It was shown earlier that the value of $S(\mathcal{C})$ is estimated as follows:

$$\begin{aligned} S(\mathcal{C}) &= \frac{\sum_{i=1}^m J(X_i, Y_i, H(\mathcal{C}))}{n(\mathcal{C})} - \sum_{i=1}^m p_i^2 \\ &= \frac{R(\mathcal{C})}{n(\mathcal{C})} - \sum_{i=1}^m p_i^2 \end{aligned}$$

The above expression can be estimated directly from the sketch statistics. Both $R(\mathcal{C})$ and $n(\mathcal{C})$ are maintained directly in the sketched micro-cluster statistics. Next, we discuss the computation of the second term. The value of $\sum_{i=1}^m p_i^2$ can be estimated as the sum of the squares of the sketch components in each row. We compute the minimum value across w rows, and denote this value by P_{min} . The corresponding term is estimated as $P_{min}/n(\mathcal{C})^2$. Next, we will bound the quality of the estimation with the use of the sketch-based approach.

LEMMA 3.3. *With probability at least $1 - \delta$, the sketch based estimation of $S(\mathcal{C})$ lies in the range $[S(\mathcal{C}) - \epsilon \cdot \sum_{i=1}^m p_i^2, S(\mathcal{C})]$.*

Proof. We note that $S(\mathcal{C})$ is expressed using two terms, the first of which is known exactly. The only source of inaccuracy is in the estimation of $\sum_{i=1}^m p_i^2$, which is computed as a self dot-product, and therefore over-estimated. Since this term adds negatively to the overall estimation, it follows that

the overall computation is always under-estimated. Therefore, the upper bound on the estimation is the true value of $S(\mathcal{C})$.

In order to compute the lower bound, we consider the case when the second term $\sum_{i=1}^m p_i^2$ is over-estimated as much as possible. In order to estimate this value, we consider a hypothetical graph $Q(\mathcal{C})$ in which all edges of $H(\mathcal{C})$ are received exactly once, and the frequency of the i th edge is $F(X_i, Y_i, H(\mathcal{C}))$. We note that the sketch of this graph is exactly the same as that of $H(\mathcal{C})$, since the aggregate frequencies are the same. Therefore, the dot product of $Q(\mathcal{C})$ with itself will estimate $\sum_{i=1}^m F(X_i, Y_i, H(\mathcal{C}))^2 = n(\mathcal{C})^2 \cdot \sum_{i=1}^m p_i^2$. The dot-product estimation for the graph $Q(\mathcal{C})$ is exactly equal to P_{min} . Therefore, the value of $P_{min}/n(\mathcal{C})^2$ is an estimate of $\sum_{i=1}^m p_i^2$. By using the bounds discussed in [9], it can be shown that this over-estimate is at most $\epsilon \cdot \sum_{i=1}^m p_i^2$ with probability at least $1 - \delta$. This establishes the lower bound.

4 Experimental Results

In this section, we will present the experimental results from the use of this approach. We will test the techniques for both efficiency and effectiveness. We will test both the exact clustering approach and the compression-based clustering approach. We will show that the two techniques are almost equally good in terms of quality, but the compression-based technique is significantly superior in terms of efficiency. This is because the size of the underlying micro-clusters in the disk-based technique increases with progression of the data stream. This results in a slow down of the exact clustering technique with progression of the data stream.

4.1 Data Sets We used a combination of real and synthetic data sets in order to test our approach. The real data sets used were as follows:

(1) DBLP Data Set: The DBLP data set contains scientific publications in the computer science domain. We further processed the data set in order to compose author-pair streams from it. All conference papers ranging from 1956 to March 15th, 2008 were used for this purpose. There are 595,406 authors and 602,684 papers in total. We note that the authors are listed in a particular order for each paper. Let us denote the author-order by a_1, a_2, \dots, a_q . An author pair $\langle a_i, a_j \rangle$ is generated if $i < j$, where $1 \leq i, j \leq q$. There are 1,954,776 author pairs in total. Each conference paper along with its edges was considered a graph. We used a clustering input parameter of $k = 2000$ clusters.

(2) IBM Sensor 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. We used two different data sets with the following characteristics:

Igraph0103-07: The data set **Igraph0103-07** contained a stream of intrusion graphs from June 1, 2007 to June 3, 2007. The data stream contained more than $1.57 \cdot 10^6$ edges in the aggregate, which were distributed over approximately 2250 graphs. We note that this graph was much denser compared to the DBLP data set, since each individual graph was much larger. Thus, even though the number of graphs do not seem very large, the size of the underlying edge streams were huge, since each graph occupies a large amount of space. We intentionally chose a graph structure which was very different from the DBLP data set, since this helps in evaluating our algorithm on different kinds of graphs.

Igraph0406-07: The data set **Igraph0406-07** contained a stream of intrusion graphs from June 4, 2007 to June 6, 2007. The data stream contained more than $1.54 \cdot 10^6$ edges in the aggregate, which were distributed over approximately 2760 graphs. We used a clustering input parameter of $k = 120$ clusters.

(3) Synthetic Data Set: We used the R-Mat data generator in order to generate a base template for the edges from which all the graphs are drawn. The input parameters for the R-Mat data generator were $a = 0.5$, $b = 0.2$, $c = 0.2$, $S = 17$, and $E = 508960$ (using the CMU NetMine notations). If an edge is not present between two nodes, then the edge will also not be present in *any* graph in the data set. Next, we generate the base clusters. Suppose that we want to generate κ base clusters. We generate κ different zipf distributions with distribution function $1/i^\theta$. These zipf distributions will be used to define the probabilities for the different nodes. The base probability for an edge (which is present on the base graph) is equal to the product of the probabilities of the corresponding nodes. However, we need to adjust these base probabilities in order to add further correlations between different graphs.

Next, we determine the number of edges in each graph. The number of edges in each of the generated graph is derived from a normal distribution with mean $\mu = 100$ and standard deviation $\sigma = 10$. The proportional number of points in each cluster is generated using a uniform distribution in $[\alpha, \beta]$. We used $\alpha = 1$, and $\beta = 2$. In order to generate a graph, we first determine which cluster it belongs to by using a biased die, and then use the probability distributions to generate the edges. The key here is that the different node distributions be made to correlate with one another. One way of doing so is as follows. Let $\mathcal{Z}_1 \dots \mathcal{Z}_\kappa$ be the κ

different Zipf distributions. In this case, we used $k - 20$ in order to generate the data set. In order to add correlations, we systematically add the probabilities for some of the other distributions to the i th distribution. In other words, we pick r other distributions and add them to the i th distribution after adding a randomly picked scale factor. We define the distribution S_i from the original distribution Z_i as follows:

$$S_i = Z_i + \alpha_1 \cdot (\text{randomly picked } Z_j) + \dots \\ \dots + \alpha_r \cdot (\text{randomly picked } Z_q)$$

$\alpha_1 \dots \alpha_r$ are small values generated from a uniform distribution in $[0, 0.1]$. The value of r is picked to be 2 or 3 with equal probability. We use $S_1 \dots S_r$ to define the node probabilities. We used a clustering input parameter of $k = 20$.

For all data sets (unless otherwise mentioned), the default length of the hash table was 500, and the default number of hash functions was 15.

4.2 Evaluation Metrics We used a variety of metrics for the purposes of evaluation. For the case of the synthetic data sets, we used the *cluster purity measure*. For each generated cluster, we determined the dominant cluster id (based on the synthetic generation identifier), and reported the average cluster purity over the different clusters. The higher the cluster purity, the better the quality of the clustering. On the other hand, this is not an effective metric for the case of real data sets. This is because the “correct” definition of an unsupervised cluster is unknown in real data sets. Therefore we rely on two criteria to test the effectiveness: (1) We explicitly examine the clusters anecdotally to illustrate their coherence and interpretability. (2) A possible source of error can be the use of the sketch-based approximation. Therefore, we test the percentage of time that the sketch-based approximation results in a different assignment than one which is achieved by using the exact representation of the micro-clusters.

Therefore, *for the purposes of evaluation only*, we also retain the exact representations of the micro-clusters which are constructed on the clusters maintained with the approximate sketch-based approach. We note that this option is for effectiveness evaluation purposes only and is disabled during efficiency measurements. Then, we compute the assignment using the exact as well as sketch-based computation. We compute the fraction of time that the two assignments are the same. A perfect situation would be one in which the value of this fraction is 1.

4.3 Clustering Evaluation We first present results on effectiveness. For the case of real data sets, no realistic “ground-truth” can be inferred for an unsupervised problem. Therefore, we will intuitively examine some anecdotal evi-

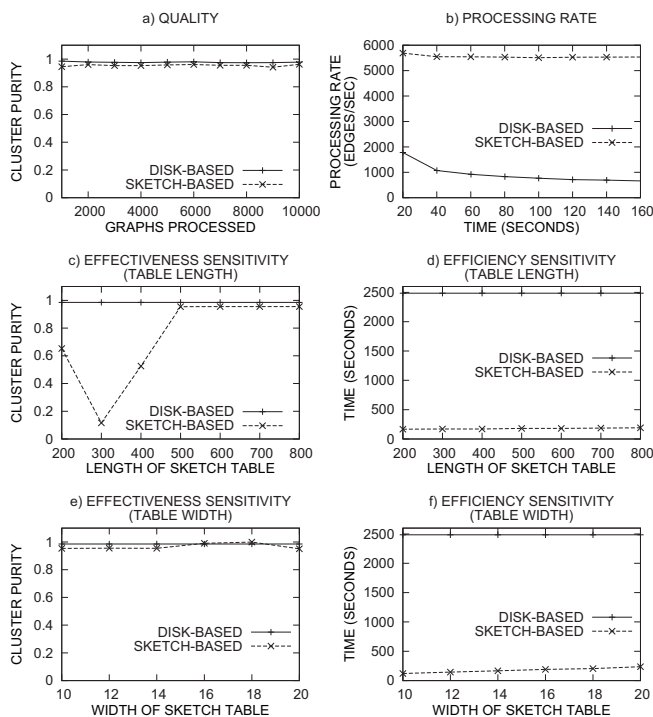


Figure 2: Performance on Synthetic Data Set

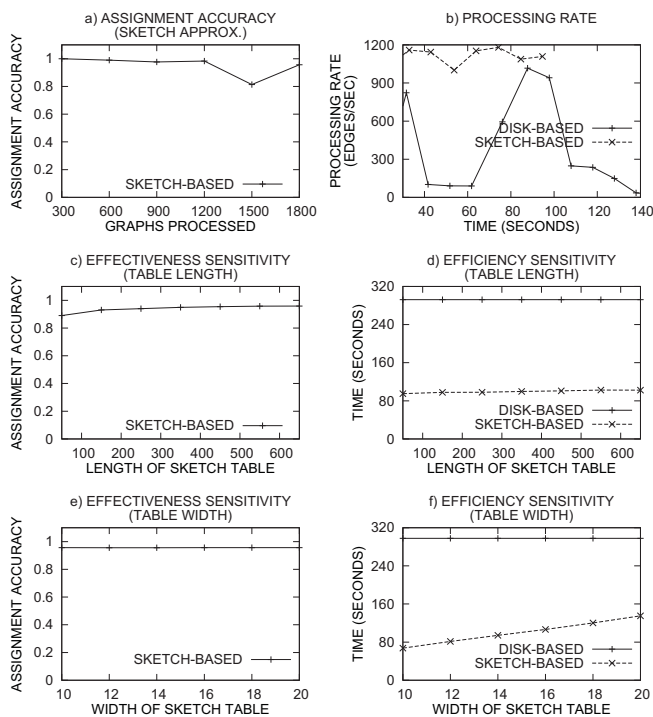


Figure 3: Performance on IGRAPH0103-07 Data Set

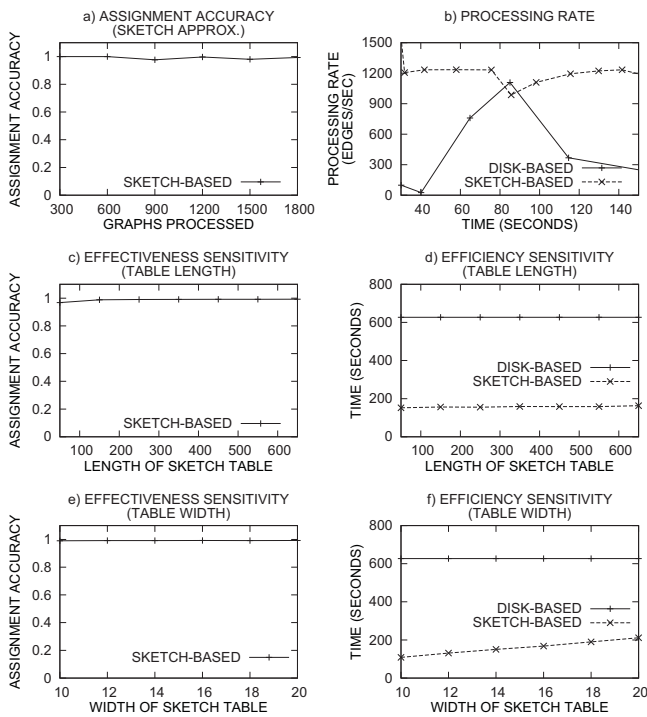


Figure 4: Performance on IGRAPH0406-07 Data Set

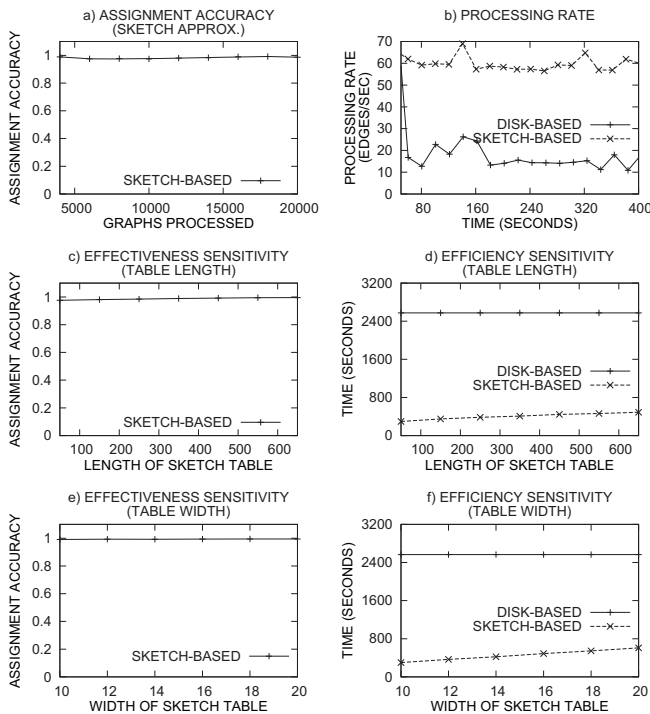


Figure 5: Performance on DBLP Data Set

dence about the clusters in order to explore their natural coherence. Then, we will examine the accuracy of the sketch-based approximation with quantitative comparisons of the sketch based assignments with those that use the original data.

We will first present some summary results for a group of clusters obtained by the algorithm in the case of the DBLP data set. The most frequently occurring authors in these clusters are as follows:

Cluster A: Frequent Authors: Jiawei Han, Umeshwar Dayal, Jian Pei, Ke Wang

Cluster A: Description: Sequential Pattern Mining Papers Published between 2000 and 2004

Cluster B: Frequent Authors: Herbert Edelsbrunner, Bernard Chazelle, Leonidas J. Guibas, John Hershberger, Micha Sharir, Jack Snoeyink, Emo Welzl

Cluster B: Description: The cluster contains a group of papers on computational geometry.

Cluster C: Frequent Authors: Mahmut T. Kandemir, Alok N. Choudhary, J. Ramanujam, Prithviraj Banerjee

Cluster C: Description: The cluster contains papers on parallel computing written between 1999 and 2003.

It is clear that in each case, the clusters contain a coherent set of authors together with papers on the same subject matter. This was generally the case across all the clusters. The aim of the above examples is to simply provide an intuitive idea of the meaningfulness of the underlying clusters. We will provide some quantitative measures slightly later.

Next, we will compare the effectiveness of the exact clustering approach with one that uses hash-compressed micro-clusters. The effectiveness for both the exact and compression-based clustering approach for the different data sets are illustrated in Figures 2(a), 3(a), 4(a) and 5(a). In the case of the synthetic data set, we use cluster purity as the measure, whereas in the case of the real data sets, we used the fraction of time that the same assignment was performed by both the two approaches. In the case of real data sets, we computed the assignment with the use of both exact and sketch-based statistics, and we checked if they were the same. For the synthetic data set, we note that the quality of the two approaches is almost identical. The purity of the compression-based clustering approach is only slightly lower than the purity of the exact clustering approach. For the real data sets, the percentage of time that the two approaches result in the same assignment of an incoming data point to a given cluster was typically over 90%. Some cases in which the distance differences were small between the first and second choices resulted in a different ordering of assignments. However, such differences do not typically result in qualitative differences in the underlying clustering process. This suggests that the sketch-based approximation of the micro-clusters maintains the accuracy of the clustering process.

We also tested the efficiency of the two methods. All results were tested on a Debian GNU/Linux server (double dual-core 2.4 GHz Opteron processors, 4GB RAM). In Figures 2(b), 3(b), 4(b) and 5(b) we have illustrated the efficiency of the clustering method on different data sets. The progression of the stream is illustrated on the X -axis, whereas the stream processing rate (the number of edges processed per second) is illustrated on the Y -axis. Since the exact clustering approach uses a disk-based scheme to handle the large space requirements of data sets, its processing rate is significantly lower than the sketch-based approach. Furthermore, the size of the micro-clusters increases with progression of the stream, since the number of edges tracked by each micro-cluster increases with stream progression. Therefore, the technique slows down further with stream progression. On the other hand, the processing rate of the sketch-based approach is significantly faster than the exact clustering approach, and it is not affected by the progression of the stream. This is because the size and update of the memory-resident sketch table remains unaffected with the progression of the data stream. On the other hand, the number of distinct edges increases with progression of the data stream. This slows down the disk-based approach significantly. The efficiency results with stream progression are illustrated in Figures 3(b) and 4(b). It is evident that the processing rate of the disk-based approach is heavily influenced by the distribution of the incoming graphs, while the sketch-based approach maintains a relatively stable performance with time progression. The low variability of the processing rate of the sketch-based method is a clear advantage from the perspective of practical applications.

We also tested the sensitivity of the clustering approach with sketch-table parameters. From the estimation analysis in section 3, it is evident that we can obtain better quality results by increasing the number of hash functions w and the range h . On the other hand, it is also not advantageous to increase the sketch table size unnecessarily, since this results in inefficiency on account of poor cache locality. It is desirable that a high quality clustering can be obtained with the use of a reasonably small sketch table. In this section, we will conduct sensitivity analysis of the effectiveness and efficiency with sketch table size.

The effectiveness and efficiency results of the two methods on the DBLP data set are illustrated in Figures 5(a) and (b). It is evident that the two approaches resulted in a very similar assignment, and the processing rate of the sketch-based approach is significantly higher than the disk-based approach. Since the disk-based approach requires too much time to process the whole DBLP data set, we will use only the first 5,000 graphs for sensitivity analysis.

Figures 2(c) and (d) illustrate the impact of sketch table length on effectiveness and efficiency of the clustering process. We use the results of the exact clustering ap-

proach as the baseline. These results are constant across the range of sketch-table parameters, because the exact clustering approach does not use the sketch table. We can see that the clustering quality improves with increasing sketch table length. This is because collisions are more likely to occur in smaller hash tables. On the other hand, the efficiency is not affected much by the sketch table length due to the constant lookup time of hash tables. We also reported the sensitivity analysis of sketch table length on the real data sets in Figures 3(c), (d), 4(c), (d) and 5(c), (d). As the case of synthetic data sets, when the length of the sketch table is larger than 500, the similarity of assignment between the two methods is more than 90%. Thus, for modestly large sketch-table sizes, the approach is able to closely mimic the behavior of the exact clustering approach.

We also tested the sensitivity of the approach to the number of hash functions. The number of hash functions was varied from 10 to 20. The results for the synthetic data sets are illustrated in Figures 2(e), and (f), and those for the real data sets are illustrated in 3(e), (f), 4(e), (f) and 5(e), (f). As in the previous case, we present the results of the exact clustering approach as the baseline in each figure. The processing time increases linearly when the number of hash functions increases. That is because the process of determining the minimum value among all the cells requires us to look up each hash function once. We also note that the quality is not affected very significantly by the number of hash functions. While increasing the number of hash functions improves the robustness, its effect on the absolute quality is relatively small. This implies that we can use a small number of hash-functions in order to maintain efficiency, while retaining effectiveness.

It is also valuable to test the efficiency of the proposed approach over varying numbers of clusters. The results for synthetic and real data sets are illustrated in Figures 6 (a), (b), (c) and (d), respectively. As a comparison, we present the results of the exact clustering approach for each data set. In all figures, the X -axis illustrates the number of clusters, and the Y -axis represents the running time in seconds. The DBLP data set contains many authors and papers which tend to have numerous underlying clusters. Because of this characteristic of the DBLP data set, we vary the number of clusters from 2000 to 3000. We vary the number of clusters from 30 to 130 for the synthetic data set, and vary the number from 120 to 220 for the two sensor data sets. Other settings are the same as the previous figures. From Figures 6, it is evident that our approach scales linearly with increasing number of clusters for all data sets. This is because the number of sketch tables and the distance function computations scale linearly with increasing number of clusters.

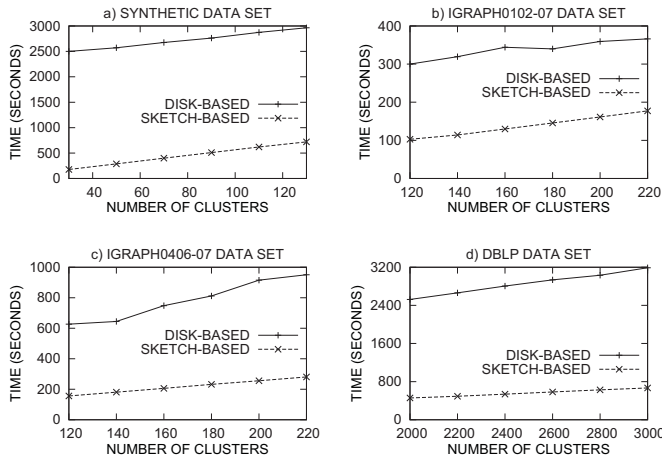


Figure 6: Sensitivity Analysis to the Number of Clusters

5 Conclusions and Summary

In this paper, we presented a new algorithm for clustering massive graph streams. While the problem of clustering graph data has been discussed in the literature, the currently available techniques are not designed for fast data streams. Furthermore, available methods are not designed for the case of massive graphs. In such cases, the number of distinct edges is too large to manage effectively. This case leads to unique challenges because it is no longer possible to efficiently hold even summary information. In this paper, we address these challenges with the use of a novel hash-compressed micro-cluster technique. The goal of this technique is to use a summarized micro-cluster representation which can be efficiently maintained in limited space (and therefore in main memory). This technique continues to maintain the effectiveness of the method without losing efficiency. We present experimental results illustrating the effectiveness and efficiency of the method.

Acknowledgement

This work is partially supported by the National Science Foundation under Grants No. IIS-0905215.

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 Proceedings (2007), pp. 46–55.
- [2] C. Aggarwal, J. Han, J. Wang, and P. Yu, *A Framework for Clustering Evolving Data Streams*, VLDB Conference Proceedings (2003), pp. 81–92.
- [3] C. Aggarwal, *Data Streams: Models and Algorithms*, Springer (2007).
- [4] C. Aggarwal, and H. Wang, *Managing and Mining Graph Data*, Springer (2010).

- [5] N. Alon, Y. Matias, and M. Szegedy, *The Space Complexity of Approximating the Frequency Moments*, ACM Symposium on Theory of Computing (1996), pp. 20–29.
- [6] A. Z. Broder, M. Charikar, A. Frieze, and M. Mitzenmacher, *Syntactic clustering of the Web*, WWW Conference, Computer Networks (1997), 29(8–13), pp. 1157–1166.
- [7] L. O’Callaghan, N. Mishra, A. Meyerson, S. Guha, and R. Motwani, *Streaming-Data Algorithms For High-Quality Clustering*, ICDE Conference Proceedings (2002), pp. 685.
- [8] D. Chakrabarti, R. Kumar, and A. Tomkins, *Evolutionary clustering*, KDD Conference Proceedings, (2006) pp. 554–560.
- [9] G. Cormode, and S. Muthukrishnan, *An Improved Data-Stream Summary: The Count-min Sketch and its Applications*, Journal of Algorithms (2005), 55(1), pp. 58–75.
- [10] T. Dalmagas, T. Cheng, K.-J. Winkel, and T. Sellis, *Clustering XML Documents using Structural Summaries*, in Lecture Notes in Computer Science (2005), Vol. 3268, pp. 547–556.
- [11] G. Flake, R. Tarjan, and M. Tsioutsoulouklis, *Graph Clustering and Minimum Cut Trees*, Internet Mathematics (2003), 1(4), pp. 385–408.
- [12] D. Gibson, R. Kumar, and A. Tomkins, *Discovering Large Dense Subgraphs in Massive Graphs*, VLDB Conference Proceedings (2005), pp. 721–732.
- [13] A. Jain, and R. Dubes, *Algorithms for Clustering Data*, Prentice Hall, New Jersey, (1998).
- [14] L. Kaufmann, and P. J. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*, John Wiley & Sons, Inc., NY, (1990).
- [15] B. W. Kernighan, and S. Lin, *An efficient heuristic for partitioning graphs*, Bell Systems Technical Journal, (1970), 49, pp. 291–307.
- [16] M. S. Kim, and J. Han, *A Particle-and-Density Based Evolutionary Clustering Method for Dynamic Networks*, PVLDB (2009), 2(1), pp. 622–633.
- [17] S. Raghavan, and H. Garcia-Molina, *Representing web graphs*. ICDE Conference Proceedings (2003), pp. 405–416.
- [18] M. Rattigan, M. Maier, and D. Jensen, *Graph Clustering with Network Structure Indices*, ICML Conference Proceedings (2007), pp. 783–790.
- [19] A. A. Tsay, W. S. Lovejoy, and D. R. Karger, *Random Sampling in Cut, Flow, and Network Design Problems*, Mathematics of Operations Research (1999), 24(2), pp. 383–413.
- [20] M. J. Zaki, and C. C. Aggarwal, *XRules: An Effective Structural Classifier for XML Data*, ACM KDD Conference Proceedings (2003), pp. 316–325.
- [21] Z. Zeng, J. Wang, L. Zhou, and G. Karypis, *Out-of-core Coherent Closed Quasi-Clique Mining from Large Dense Graph Databases*, ACM Transactions on Database Systems (2007), Vol 31(2), pp. 13.
- [22] T. Zhang, R. Ramakrishnan, and M. Livny, *BIRCH: An Efficient Data Clustering Method for Very Large Databases*, ACM SIGMOD Conference Proceedings (1996), pp. 103–114.