A Framework for Clustering Massive Graph Streams†

Charu C. Aggarwal[1]*, Yuchen Zhao[2] and Philip S. Yu[2]

[1] *IBM T. J. Watson Research Center, Hawthorne, NY 10532, USA*

[2] *Department of Computer Science, University of Illinois at Chicago, Chicago, IL 60607, USA*

Received 2 May 2010; revised 6 July 2010; accepted 1 August 2010
DOI:10.1002/sam.10090
Published online 22 September 2010 in Wiley Online Library (wileyonlinelibrary.com).

Abstract: In this paper, we 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 multidimensional 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 propose a technique for creating *hash-compressed microclusters* from graph streams. The compressed microclusters are designed by using a hash-based compression of the edges onto a smaller domain space. We provide theoretical results which show that the hash-based compression continues to maintain bounded accuracy in terms of distance computations. Since clustering is a data summarization technique, it can also be naturally extended to the problem of evolution analysis. We provide experimental results which illustrate the accuracy and efficiency of the underlying method. © 2010 Wiley Periodicals, Inc. Statistical Analysis and Data Mining 3: 399–416, 2010

Keywords: data mining; clustering; graphs

1. INTRODUCTION

In recent years, there has been a renewed focus on graph mining algorithms because of applications involving the web, bioinformatics, social networking, and community detection. Numerous algorithms have been designed for graph mining applications such as clustering, classification, and frequent pattern mining [1–7]. A detailed discussion of graph mining algorithms may be found in Refs. [2,8].

The problem of clustering has been studied extensively in the data mining literature [9–12]. Recently, the problem has also been examined in the context of graph data [2,3,8,13]. 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 [14], minimum-cut determination [15] and dense subgraph determination [16,17].

Correspondence to: Charu C. Aggarwal (charu@us.ibm.com)
† Submission to Best of SDM 2010 Issue.

The problem of clustering in the graph domain allows for two different kinds of algorithms:

- **Node Clustering Algorithms:** In this case, we have a single large graph, and we wish to partition this large graph into densely connected regions of the graph. Many scenarios such as the web, social networks, and information networks fall into this category. Some examples of such graph-partitioning algorithms are found in Refs. [14, 18, 19].
- **Graph Clustering Algorithms:** In this case, we have many graphs which are defined over a particular domain of nodes. In such case, we cluster the graphs on the basis of the structural similarity across the different graphs. We further note that while the first kind of algorithms try to determine the *dense regions* in the graph, the latter type try to find similarity across different graphs whether or not they are dense. Thus, the goal of graph clustering algorithms is inherently different from that of node clustering algorithms.

Recently, the problem has also been studied in the context of *object clustering*, in which we attempt to cluster

many different *individual* graphs (as objects) [1,4], 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 [1,4] are designed for the most straightforward 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 [1,4], computational biology, or chemical compound analysis.

We study a much more challenging case in which a large number of 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 [20], 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 collection 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 [1,4]. Such algorithms require multiple passes over the data [1,4], 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 multidimensional data [21,22,26], there are no known algorithms for the case of clustering graph streams. In this paper, we 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. These additional challenges are related to the fact that the space-complexity of algorithms for summarizing the underlying graphs can be considerable. This is because the complexity of a summary is related to the number of distinct edges in the graph.

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

- Most graph mining algorithms use substructural analysis in order to perform the clustering. This may be difficult in the case of data streams because of the one-pass constraint. The one-pass constraint creates some natural algorithmic constraints in the data stream computation process.
- The number of possible edges scales up quadratically with the number of nodes. Therefore, the total number of distinct edges in the graph may be very large. 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 pairwise basis.

Most generalizations of known clustering algorithms are likely to require disk-based computations because of the large number of the edges which need to be processed. In this paper, we design the concept of *sketch-based microclusters* for graph data. The broad idea in sketch-based microclusters is to combine the idea of sketch-based compression with microcluster summarization. This approach helps in reducing the size of the representation of the underlying microclusters, so that they are easy to store and use. This also helps in creating an approach which can be utilized with memory-resident computations only. We 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 introduce the graph clustering problem, and the broad framework which is used to solve this problem. We introduce the concept of graph microclusters, 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. The application of the clustering technique to the problem of evolution analysis is discussed in Section 4. Section 5 contains the experimental results. Section 6 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 concentrate on the second class of algorithms in which different graphs are clustered as objects based on structural similarity.

The case of node-clustering algorithms traditionally been studied in the context of the minimum-cut problem [15], graph partitioning [14], network structure clustering [13], and dense subgraph determination [16,17]. In particular, the techniques for dense subgraph determination [16,23] 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 individual graphs as objects based on structural similarity. An interesting class of multilevel graph partitioning schemes has been presented in Refs. [18, 19]. In these methods, a large graph is first coarsened into a smaller graph with the use of node-collapsing techniques, and then a partitioning phase is applied to the much smaller graph. While this is an excellent technique to find dense regions in a given graph, this is not the goal of the paper, in which we wish to determine the similarities among different graphs *objects* in a stream of such objects. An approach for finding dense regions may not necessarily work for finding similar structures across multiple graphs and vice-versa.

Recently, methods have been studied for clustering graphs as objects in the context of XML data [1,4]. However, these techniques have two shortcomings: (i) These techniques are designed for the case when the nodes are drawn from a limited domain rather than from a very large set of possibilities. When the number of nodes is very large, the number of distinct edges may be too large to track effectively. (ii) 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 achieves 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 microclusters. We show that the hash-compression technique does not lose significant effectiveness during the clustering process. We show that the additional process of hash compression does not lose any effectiveness for the clustering process. We illustrate the effectiveness of the approach on a number of real and synthetic data sets.

2. GRAPH STREAM CLUSTERING FRAMEWORK

In this section, we 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. The label of each node in $\mathcal{N}$ is denoted by a corresponding string. 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 $G_1 \ldots G_k \ldots$ 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 $(X_1, Y_1) \ldots (X_i, Y_i) \ldots$, 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

$\mathcal{C}_1, \ldots, \mathcal{C}_k$, such that each graph from the data stream is dynamically assigned to one of the clusters in real time.

While microclustering [21] has been used in order to cluster multidimensional data, we construct microclusters which are specifically tailored to the problem of graph data. In this paper, we use a sketch-based approach in order to create *hash-compressed microcluster representations* of the clusters in the underlying graph stream. First, we discuss a more straightforward representation of the uncompressed microclusters, and the storage and computational challenges in maintaining these representations. Then, we discuss how the sketch-based techniques can be used in order to effectively deal with these challenges. We show that the sketch-based techniques can be used in combination with the microclusters to construct the distance functions approximately over the different clusters.

Next, we introduce a more straightforward and direct representation of graph-based microclusters. Let us consider a cluster $\mathcal{C}$ containing the graphs $\{G_1, \ldots, G_n\}$. We assume that the implicit graph defined by the summation of the graphs $\{G_1, \ldots, G_n\}$ is denoted by $H(\mathcal{C})$. Then, we define the microcluster $GCF(\mathcal{C})$ as follows:

DEFINITION 2.1: The microcluster $GCF(\mathcal{C})$ 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 microcluster $\mathcal{C}$. 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 is defined as $F(X_q, Y_q, H(\mathcal{C})) = \sum_{r=1}^{n} F(X_q, Y_q, G_r)$.
- The number of graphs in the microcluster $\mathcal{C}$ 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 microclusters 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 1: *The graph microclusters satisfy the additive separability property. This means that the microcluster 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 microclusters also satisfy a limited version of the *subtractivity property*. All components of the microcluster other than the time stamp can be used to compute $GCF(\mathcal{C}_1 - \mathcal{C}_2)$ as a function of $GCF(\mathcal{C}_1)$ and $GCF(\mathcal{C}_2)$. We summarize as follows:

PROPERTY 2: *The graph microclusters satisfy a limited version of the subtractivity property.*

We note that graph microclusters differ from multidimensional microclusters in the sense that while the size of the multidimensional microcluster representation remains bounded and easy to compute, this is not the case for graph microclusters. This 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 microcluster representation. However, for simplicity, we 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 first describe the use of the raw microcluster statistics in order to cluster the incoming graphs. Then, we describe how sketch methods are used in order to make changes to specific portions of the microcluster representation and corresponding computations.

The microclustering algorithm uses the number of microclusters as input. We refer to the Graph MICROclustering algorithm as the *GMicro* method. Initially, the clustering algorithm starts off with the null set of microclusters. As new graphs arrive from the data stream, they are added to the data as singleton microclusters. Thus, the first k graph records are created as singleton microclusters. Subsequently, when the next graph G_r arrives, the distance to each microcluster centroid is computed. The distance measure that we compute is constructed as a variant on the $L2$-distance measure for multidimensional data and can be

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

Fig. 1 The *GMicro* algorithm.

computed with the use of microcluster 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 microcluster is less than the distance of G_r to that microcluster. The structural spread is defined as a factor[1] of the mean-square radius $S(\mathcal{C})$ of the elements of microcluster $\mathcal{C}$ from its centroid. As in the case of the structural similarity measure, we see that the spread can also be computed as a function of the microcluster statistics. If the structural spread is less than the distance of G_r to the closest microcluster, then we create a new singleton microcluster containing only G_r. This microcluster replaces the most stale (least recently updated) microcluster. The information on the last update time of the microclusters is available from the corresponding time stamps in the microcluster statistics. The overall algorithm is illustrated in Fig. 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 microcluster can also be considered a pseudograph for which the frequencies of the corresponding edges are defined as the sum of the corresponding frequencies of the edges from which the microcluster was created. Therefore, we simply need to define a distance (similarity) function between two graphs in order to compute the similarity between microclusters and centroids. Let $\mathcal{C}$ be a given microcluster which contains the graphs $G_1, \ldots, G_n$. Let the implicit graph for the microcluster $\mathcal{C}$ (corresponding to the summation of the graphs $G_1, \ldots, G_n$) 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 microcluster. Let the edges in L be denoted by $(X_1, Y_1), \ldots, (X_m, Y_m)$. 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:

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

A second possibility for the computation of the similarity function is the *dot product*. Unlike, the $L2$-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:

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

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

[1] In accordance with normal distribution assumptions, we pick this factor to be 3.

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 factor[2] t multiplied with $S(C)$. Both the structural distance measure and the structural spread can be computed as a function of the microcluster statistics. This is because the value of $F(X_q, Y_q, H(\mathcal{C}))$ is directly available from the first-order statistics of microcluster $\mathcal{C}$. Similarly, the value of $n(\mathcal{C})$ is included in the microcluster 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 microcluster statistics.

The spread $S(\mathcal{C})$ can also be directly computed from the microcluster 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 microcluster 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 microcluster. As discussed earlier, this can be computed directly from the microcluster statistics. By substituting in the expression for $S(\mathcal{C})$, we get the following:

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

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

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

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

We note that all the terms in the above definition are drawn from the microcluster definition. Therefore, the spread $S(C)$ can be computed directly from the microcluster statistics. ■

3. SKETCH-BASED MICROCLUSTER COMPRESSION

The storage and computational requirements for updating the microclusters depend upon the number of edges in them. The number of edges in the microclusters will typically increase as new graphs arrive in the stream. As the size of the microcluster 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 microcluster 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 microcluster 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 [24,25] were designed for enumeration of different kinds of frequency statistics of data sets. A commonly-used sketch is the count-min method [25]. 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 one-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 two-dimensional data structure. For example, consider a one-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 $[0, \ldots, h-1]$. 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 nonnegative counts only, and there may be an overestimation 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 Ref. [25], 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 in incremented by the frequency of the corresponding edge.

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

DEFINITION 3.1: The microcluster $GCF(\mathcal{C})$ 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 microcluster. This requires a table with size $(e/\epsilon) \cdot \ln(1/\delta)$. The actual microcluster 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 microcluster 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 microcluster. 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 microclusters such as the dot product.

The *GMicro* algorithms can be used with this new definition of microclusters except that the intermediate computations may need to be performed as sketch-based estimates. In the remaining portion of this section, we 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 microclusters. 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 [25] 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 hashing $X_i \oplus Y_i$ into the hash table with the use of the w hash functions, and computing the minimum of the *square of the* corresponding entries. However, the bound on this estimate is quite loose, and therefore we will not use the approach of estimating $J(X_i, Y_i, H(\mathcal{C}))$. Rather, we will see that the additional information $R(\mathcal{C}) = \sum_{i=1}^{m} J(X_i, Y_i, \mathcal{C})$, which is maintained in the sketch-based statistics is sufficient to perform the intermediate computations for clustering.

The above observations emphasize that intermediate information on the microcluster statistics can be constructed approximately with the sketch technique. This is useful, since all the important properties of the clusters are encoded in the microcluster 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 microcluster statistics need to be derived. Next, we discuss the estimation of important quantitative computations which are performed during the clustering process.

3.1. Distance Estimations

We expand the expression for *L2Dist* in order to express it in terms of sketch-based microcluster statistics:

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

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

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 by Cormode and Muthukrishnan [25]. 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 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 overestimated:* This forms the lower bound for the range, since the overestimated term has a negative sign attached before it. We know from [25], that the overestimation 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 overestimated:* This forms the upper bound for the range. As in the previous case, we can show that the level of overestimation 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 Ref. [25]. ■

3.2. Estimation of Spread

In this section, we 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:

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

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 microcluster 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 first term in the above expression is estimated as $P_{\min}/n(\mathcal{C})$, and the second term is estimated as $P_{\min}/n(\mathcal{C})^2$. Next, we 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 $\sum_{i=1}^{n} p_i^2$, which is computed as a self dot product, and therefore overestimated. Since this term adds negatively to the overall estimation, it follows that the overall computation is always underestimated. 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 overestimated 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 ith 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 $\sum_{i=1}^{m} F(X_i, Y_i, H(\mathcal{C}))^2 = n(\mathcal{C})^2 \cdot \sum_{i=1}^{n} p_i^2$. 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 [25], it can be shown that this overestimate is at most $\epsilon \cdot \sum_{i=1}^{m} p_i^2$ with probability at least $1 - \delta$. This establishes the lower bound. ■

3.3. Time Complexity

In this section, we study the time complexity of the graph stream clustering algorithm. We compute this time complexity both for the case of the disk-based clustering algorithm and the sketch-based clustering algorithm. This analysis also show the advantage of the disk-based clustering algorithm over the sketch-based clustering method. We assume that the entire data stream contains N data points, which are divided into k clusters. Furthermore, we assume that the *average* number of edges in an incoming graph in the stream is given by p. Furthermore, let the *average* number of distinct edges in the different microclusters in the stream be given by t.

3.3.1. Disk-based clustering algorithm

The main time complexity of the disk-based clustering algorithm is in computing the distances of the incoming graph to the different microclusters. We note that the process of updating the microclusters is strictly dominated by this running time. The time to compute the distances between between the ith graph with l_i edges and the jth microcluster with $L(\mathcal{C}_j)$ distinct edges is given by $l_i + L(\mathcal{C}_j)$. For each incoming graph, this needs to be repeated over the k different clusters. Furthermore, this process needs to be repeated over the N different graphs in the stream. Thus, the two terms in the expression $l_i + L(\mathcal{C}_j)$ need to be summed over $N \cdot k$ different possibilities. Since the average number of components in the graphs and the microclusters are given by p and t respectively, it follows that the total number of operations is given by $O((p + t) \cdot N \cdot k)$.

One observation about the time complexity is that it is dependent on p and t, which are highly dependent on the characteristics of the underlying data. Furthermore, the value of t is dependent on the number of distinct edges in the microclusters, which may be quite large. At a given time, the number of distinct edges in the microclusters may vary, as a result of which the processing efficiency of the stream clustering method may vary considerably over the computation process. These operations are also disk-resident, and are therefore not particularly efficient. The random-accesses to the microclusters on disk can be slow, because each random access requires seeks from the disk for the computation process.

3.3.2. Time complexity of sketch-based method

In this section, we study the time complexity of the sketch-based method. In the case of the hash-table, for each incoming graph with l_i edges, we need to compute the distance to the sketch-based microclusters. For a given graph with l_i edges, we need to compute w different hash functions. We note that (unlike the previous case), this is independent of the number of distinct edges in the microcluster. As in the previous case, a total of $N \cdot k$ different similarity computations need to be performed over the data stream. Since the average number of edges in a graph is p, the average efficiency per similarity computation is given by $p \cdot w$. Thus, the total requirement over the entire data stream is given by $O(N \cdot k \cdot p \cdot w)$. We note that unlike the previous case, this is not dependent upon the distinct edges in the data stream. This makes the efficiency requirements more stable over the course of the data stream. Furthermore, these operations are all main memory operations, and are therefore much more efficient than the disk resident case. As we will see later, these improved efficiencies of the sketch-based method will be reflected in the experimental results.

4. EVOLUTION ANALYSIS WITH CLUSTERING

Since clustering is a natural method for data summarization, it can also be used for evolution analysis. The changes in the summarized cluster representation provide an idea of the changes in the high level trends in the data. Specifically, if t_c be the current time, then for a user-defined window q, we would like to compare the clusters at time t_c with those at time $t_c - q$. Such changes provide an understanding of the nature of the changes in the underlying graph structure, and can be useful for a wide variety of applications. There

are two kinds of evolutionary changes which can be tracked in the underlying clusters:

- We can track new clusters which are added during the clustering process; or clusters which disappear over the course of stream clustering. This provides a direct view of the obvious changes which occur during the clustering process. Such situations typically arise because of the sudden arrival of either completely new trends in the data stream, or the sudden disappearance of important trends in the original data stream.
- In some cases, the changes may not be directly obvious as in the cluster additions and deletions. Rather, the *aggregate distributions of the data points among different clusters in different segments may be different*. This is a more challenging situation to detect, and may occur as a results of slower evolution over a continuous period.

Both cases require the comparisons of the clusters across different snapshots. The first case is simple, because we can simply compare the clusters at snapshot t_c, with the clusters at snapshot $t_c - q$, and report the clusters which are new, or the clusters which have disappeared. In order to accurately distinguish new clusters from old ones, we maintain *cluster identifiers*, which are defined by the time stamp at which they are created. The identifier is very useful in tracking the behavior of the clusters throughout the entire process. Since each cluster is created at a different time stamp, this ensures that the different cluster identifiers are unique. Therefore, in order to track new or disappearing clusters in the data, we track the following two quantities:

- The new cluster identifiers at time t_c, which were not present at time $t_c - q$. This immediately provides us with the new clusters which have appeared in the stream over the last horizon of length q.
- The cluster identifiers which have disappeared between time $t_c - q$ and t_c. This immediately provides us with the clusters which have disappeared in the last horizon of length q.

The above quantities provide us with a good understanding of trends appearing or disappearing in the data. The process of determining the changes in the aggregate frequencies is slightly different. For this purpose, we maintain the *vector-space frequency vectors* of the cluster identifiers. The vector-space representation of a cluster identifier contains the identifier itself along with the frequency of the corresponding cluster identifiers. The frequency of a cluster identifier corresponds to the number of data points inside the cluster. This is required in order to keep track of the 'popularity' of the corresponding cluster. Note that the set of the cluster identifiers in the vector-space representation will change over time, as new clusters are added and old ones are deleted. Therefore, if $id_1(t), \ldots, id_k(t)$ and $f_1(t), \ldots, f_k(t)$, be the corresponding cluster identifiers and frequencies, the clusters and their corresponding membership frequencies at time t, then all of these values are maintained in the vector-space representation. Thus, a total of $2 \cdot k$ values are maintained, where k is the number of clusters.

In order to compute the changes which have occurred in the evolution of the clusters from time $t_c - q$ to time t_c, we first compute the *differential cluster statistics* from time $t_c - q$ to time t_c, as well as the differential cluster statistics from time $t_c - 2 \cdot q$ to time $t_c - q$. The differential cluster statistics refer to the frequencies of the data points in the different clusters at time t_c, which correspond only to data points which have arrived between times $t_c - q$ and t_c. Since the cluster identifiers between times $t_c - q$ and t_c may not exactly be the same, we compute the statistics only for the clusters which are available at the time t_c. We formally define the differential cluster statistics between time periods (t_1, t_2) as follows:

DEFINITION 4.1: The differential cluster statistics in the time period (t_1, t_2), is the frequencies of the data points in the cluster identifiers at time t_2, which have arrived in the time period (t_1, t_2). In other words, we use only the data points which have arrived in the time-interval (t_1, t_2).

It is evident that the differential cluster statistics can be computed directly from the microclusters because of the additivity property. This is done matching the cluster identifiers between times t_1 and t_2, and then subtracting the corresponding statistics between the two time intervals.

In order to compute the aggregate changes from the clusters in the time-interval $(t_c - 2 \cdot q, t_c - q)$ to the interval $(t_c - q, t_c)$, we compute the normalized-cosine similarity between the vector-space representation of the differential cluster statistics. The smaller the cosine similarity, the greater the differences from $(t_c - q)$ to t_c. Since the normalized cosine similarity lies in the range (0, 1), we can output a *change alarm level*, which is defined as the $1 - cs(t)$, where $cs(t)$ is the current cosine similarity at time t. This alarm level can be computed continuously over horizons of length q throughout the life of the data stream. Peaks in this alarm level correspond to sudden changes in the trends in the data stream.

We note that the construction of the differential statistics requires the storage of the microclusters at different snapshots. Therefore, we need the snapshots of the past horizons in order to construct the differential statistics. If the horizon q is known *a priori*, then it suffices to maintain a single snapshot at time $t_c - q$, in addition to the current snapshot at time t_c. A complicating issue is that the user may *dynamically choose any horizon* during stream

processing, and therefore it is not sufficient to maintain a single snapshot at time $t_c - q$. While a natural solution is to store the snapshots at evenly spaced intervals, this can be impractical for an application receiving large amounts of data during long periods of time. This is because the number of snapshots will continuously increase over time, and the storage requirements will become impractical over a long period of time.

Therefore, a natural solution is to store the snapshots at pyramidally defined intervals, so as to approximate the corresponding horizons well. Ideally, we would like to obtain the differential vector of a horizon which is as close to the target horizon as possible. For this purpose, we use a pyramidal pattern [21] of storage of the snapshots. Of course, the snapshot for the current time instance is always maintained dynamically in main memory. The pyramidal storage pattern is defined by constructing a set of *levels* across which the snapshots are stored. These levels are defined and stored as follows:

(1) For a user-defined parameter $\alpha > 1$, snapshots at level i are stored at clock times which are divisible by α^i.

(2) For a user-defined integer parameter l, the last α^l snapshots of each level are retained at any moment in time.

There can be overlap between snapshots at different levels, if $l > 0$. In such cases, we eliminate duplicates in order to maximize storage efficiency. We make the following observations:

(1) At clock time T, the total number of levels is $\log_\alpha(T)$.

(2) At clock time T, the total number of snapshots is at most $\alpha^l \cdot \log_\alpha(T)$.

(3) For any user-specified horizon q and current time t_c, a snapshot can be found at time $t_c - q'$ such that $|q - q'|/h < 1/(2 \cdot \alpha^{l-1})$. Therefore, the horizon q can be approximated with a high degree of accuracy.

While the first two properties are quite evident, the third needs to be proved explicitly. The following Lemma proves this explicitly:

LEMMA 4.1: For any user-specified horizon q and current time t_c, a snapshot can be found at time $t_c - q'$ such that $|q - q'|/q < 1/(2 \cdot \alpha^{l-1})$.

Proof: See Ref. [21]. ■

We note that these results show that the number of stored snapshots increase logarithmically with time T, while guaranteeing a high degree of accuracy. For example, let us consider an extreme example of a sensor network running over 100 years, with clock intervals of 1 s. Further, let us assume that we pick $\alpha = 2$ and $l = 10$. We note that this choice of parameters guarantees a 99.99% accuracy in horizon estimation. In such a case, we need to store $2^{10} \cdot \log_2(100 \cdot 365 \cdot 24 \cdot 3600) \approx 31\,554$ snapshots. Assuming that the *historical behavior* is stored on disks at leader (or intermediate) nodes, this is a fairly modest storage requirement. This ensures that the change diagnosis can be performed *dynamically for any time-horizon according to user requirements*. This also ensures that the evolution analysis can be performed simultaneously with the stream clustering process, without a significant additional overhead.

5. EXPERIMENTAL RESULTS

In this section, we present the experimental results from the use of this approach. We test the techniques for both efficiency and effectiveness. We test both the exact clustering approach and the compression-based clustering approach. We 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 disk-based approach requires the storage of a large number of edges on disk. This slows down the disk-based approach significantly.

5.1. Data Sets and Default Parameters

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 DataBase systems and Logic Programming (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 $a_1, a_2, \ldots, a_q$. 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**[3] contained a stream of intrusion graphs from June 1, 2007 to June 3, 2007. The data stream contained more than 1.57×10^6 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**[4] contained a stream of intrusion graphs from June 4, 2007 to June 6, 2007. The data stream contained more than 1.54×10^6 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 $E = 508\,960$ (using the Carnegie Mellon University (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 $\mathcal{Z}_1 \ldots \mathcal{Z}_\kappa$ 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 ith distribution. In other words, we pick r other distributions and add them to the ith distribution after adding a randomly picked scale factor. We define the distribution $\mathcal{S}_i$ from the original distribution $\mathcal{Z}_i$ as follows:

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

$\alpha_1, \ldots, \alpha_r$ 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 $S_1 \ldots S_r$ 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.

5.2. 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 microclusters.

Therefore, *for the purposes of evaluation only*, we also retain the exact representations of the microclusters 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.

[3] Available at http://www.charuaggarwal.net/sens1/gstream.txt.
[4] Available at http://www.charuaggarwal.net/sens2/gstream.txt.

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

We 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 provide some quantitative measures slightly later.

5.4. Case Studies for Evolution Analysis

We first present some examples of results obtained by our framework on the DBLP data set. We present the evolution trends in the underlying clusters with the passage of time. We use some of the clusters containing the author Jiawei Han as a frequently occurring node (among the clustered objects) as an example. The most frequently occurring nodes (authors) in the corresponding clusters are listed in Table 1.

The cluster A is a big cluster with around 80 authors, and it spans a relatively long time period. Other clusters (B, C, D, and E) are relatively small, and may also be temporally transient. The cluster A was created when Jiawei Han published his paper in the first KDD conference in the year 1995. From then on, this cluster was frequently updated, and actually it was one of the most frequently updated clusters in DBLP. We will describe more details about this cluster later. The cluster B was created in the year 1994, and our framework found that it was frequently updated from the year 1994 to 1996. We noticed that Jiawei Han and Yongjian Fu published several papers during that time on the topic of association rules. However, this cluster remained unchanged since 1999 and finally was replaced by a new incoming graph due to the fact that it was rarely updated. The possible reason might be that Yongjian Fu graduated after this point, and did not publish significantly after this point. The cluster C mined through our framework reveals the co-authorship among Jiawei Han, Ke Wang, and Jian Pei, which accurately presents their research work on sequential pattern mining and frequent itemset mining from 2001 to 2003. Clusters D and E are two relatively new clusters, and they continued to grow steadily till the end of the stream. We noticed that Xiaolei Li and Deng Cai were two PhD students supervised by Jiawei Han, who started their research in 2003 and 2005, respectively. The information provided by the clustering process is consistent with the results generated by our proposed framework.

Table 1. Microcluster examples.

Cluster ID	Most frequent coauthors	Starting year
A	Jian Pei, Xifeng Yan, Philip S. Yu, and Anthony K. H. Tung	1995
B	Yongjian Fu	1994
C	Ke Wang and Jian Pei	2001
D	Xiaolei Li and Hector Gonzalez	2003
E	Deng Cai and Xiaofei He	2005

Since cluster A is one of the most frequently updated clusters in DBLP, it is interesting to analyze its aggregate distributions. This cluster represented a number of general data mining papers which could not be neatly assigned to any particular subtopic of data mining. We show the frequencies of the graphs in the cluster A at different time periods in Fig. 2. The increasing frequency represents more researchers are involved and more papers are published. From the figure, we can learn that the community

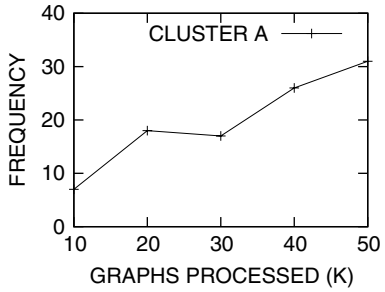

Fig. 2 Frequencies of graphs in Cluster A.

represented by the cluster A is growing fast in the past decade. This cluster increased in size over the years, as the number of authors, collaborations, and papers also increased in size over time.

5.5. Quantitative Comparisons

Next, we compare the effectiveness of the exact clustering approach with one that uses hash-compressed microclusters. The effectiveness for both the exact and compression-based clustering approach for the different data sets are illustrated in the panel (a) of Figs. 3–6. 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

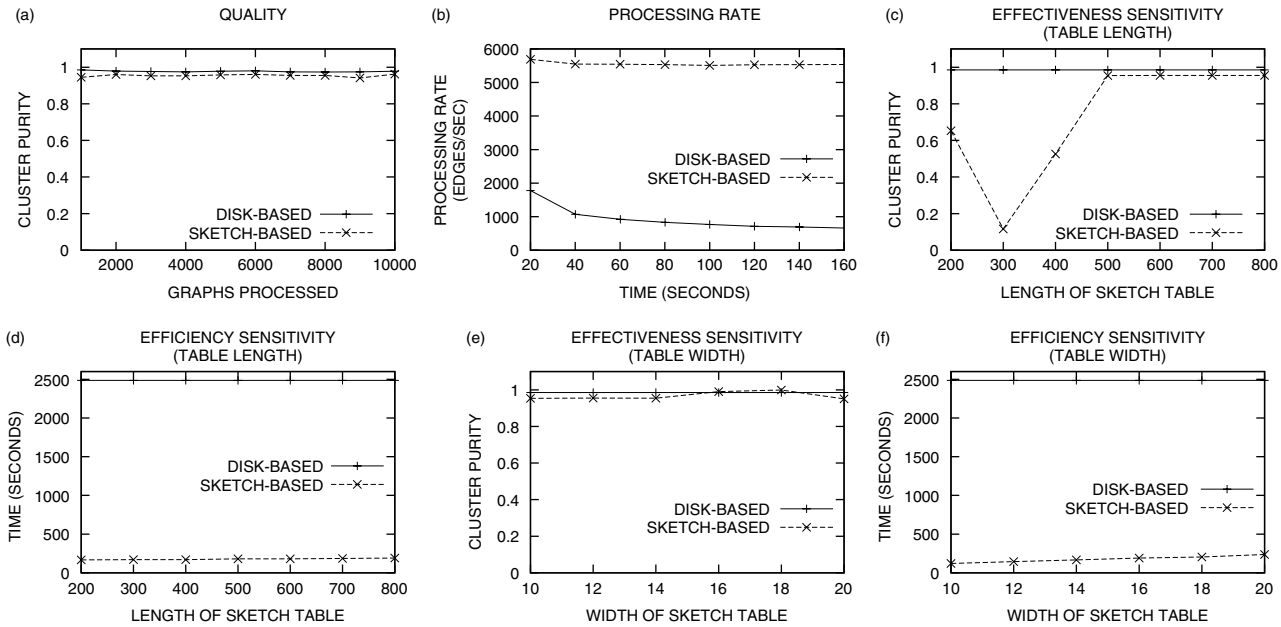

Fig. 3 Performance on synthetic data set (10K graphs).

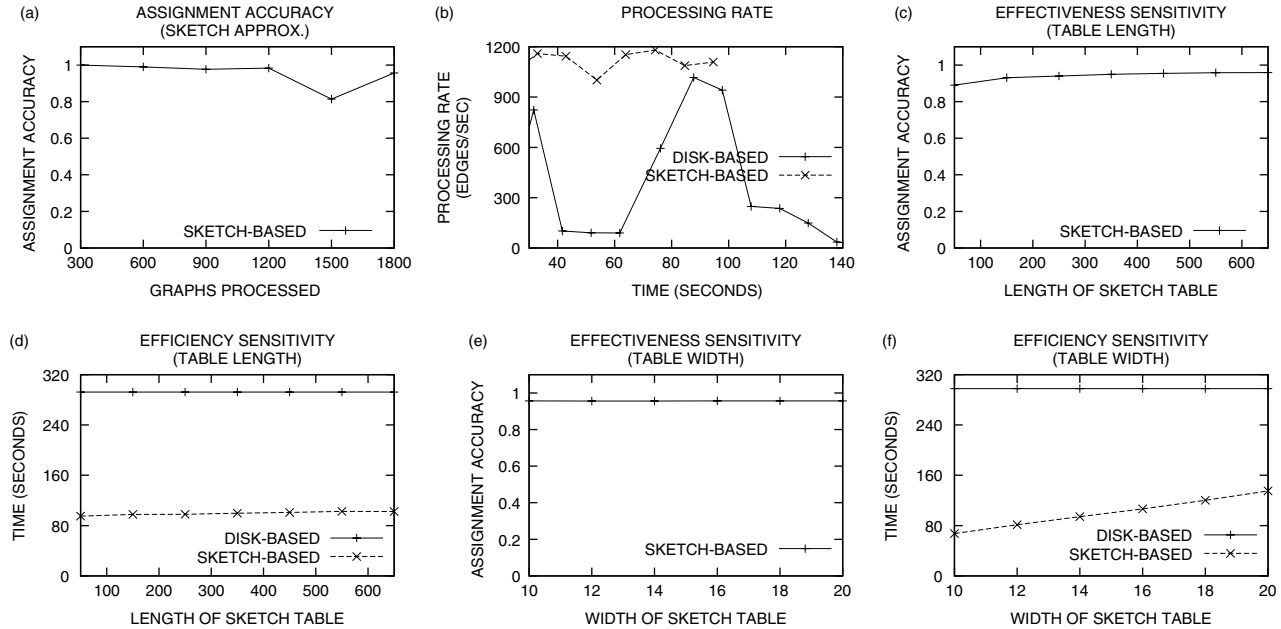

Fig. 4 Performance on Igraph0103-07 data set.

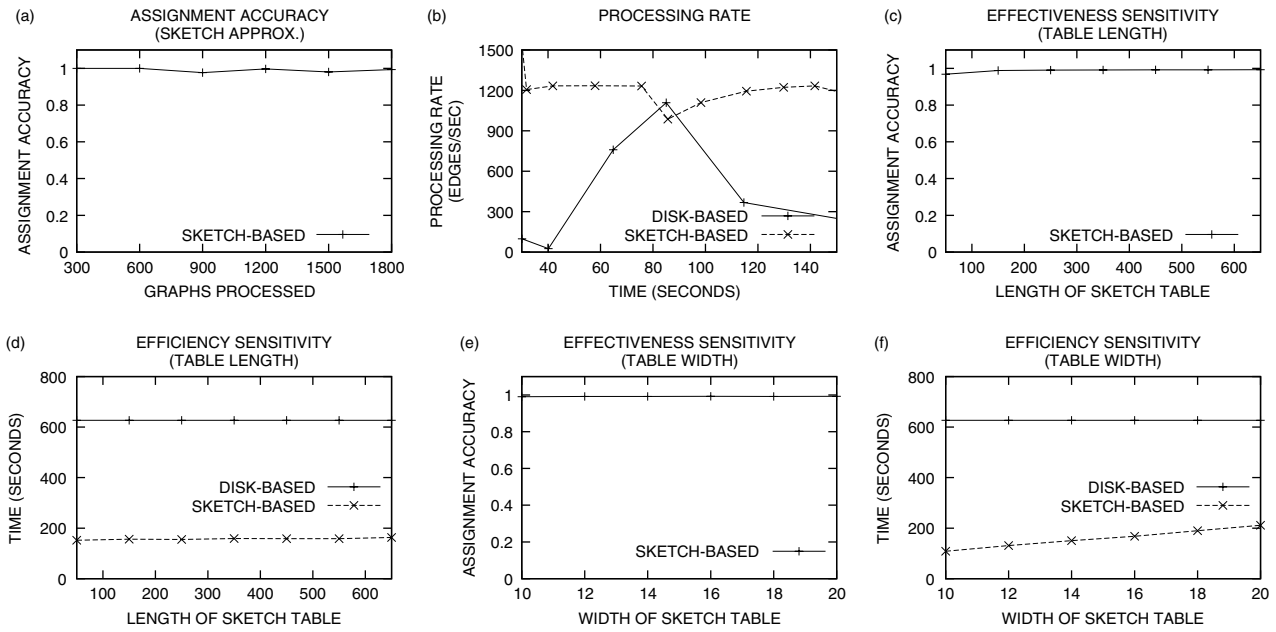

Fig. 5 Performance on Igraph0406-07 data set.

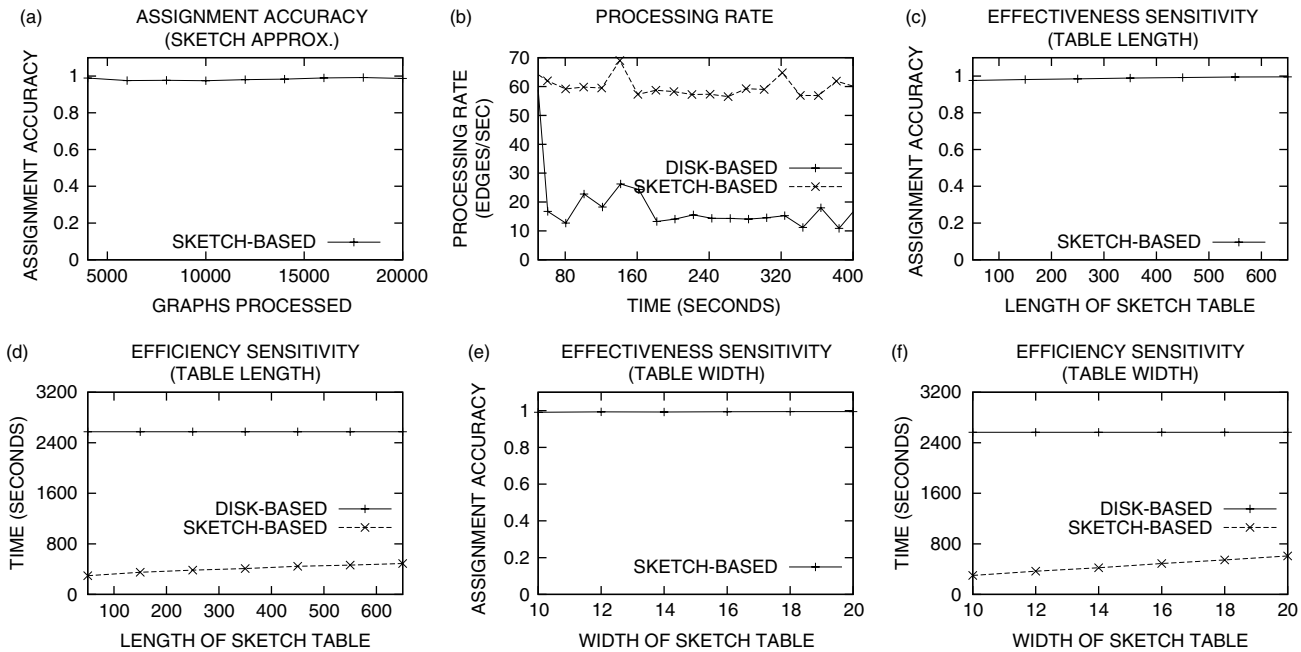

Fig. 6 Performance on DBLP data set.

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 microclusters 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, 4 GB RAM). In panel (b) of Figs 3–6, we have illustrated the efficiency of the clustering method on different data sets. The progression of the stream is illustrated in the X-axis, whereas the stream processing rate (the number of edges processed per second) is illustrated in 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 microclusters increases with progression of the stream, since the number of edges tracked by each microcluster 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. The disk-based approach is affected by the changes in the underlying trends. When new distinct edges are encountered, the scheme slows down as it is unable to find the corresponding edges in the underlying statistics and adds to the existing statistics on disk. The efficiency results with stream progression are illustrated in panel (b) of Figs. 4 and 5. It is evident that the processing rate of the disk-based approach is heavily influenced by the distribution of the edges in 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 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 the panels (a) and (b) of Figs. 6. 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 use only the first 5000 graphs for sensitivity analysis.

Panels (c) and (d) of Fig. 3 illustrate the impact of sketch table length on effectiveness and efficiency of the clustering process. We use the results of the exact clustering approach 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

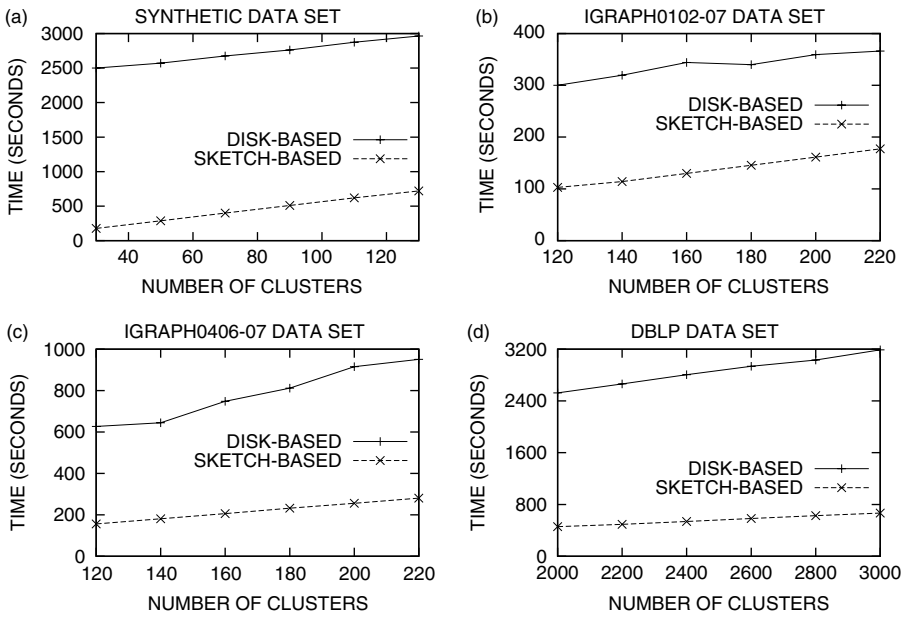

Fig. 7 Sensitivity analysis on the number of clusters.

sensitivity analysis of sketch table length on the real data sets in the panels (c) and (d) of Figs. 4–6. 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 panels (e) and (f) of Fig. 3, and those for the real data sets are illustrated in panels (e) and (f) of Figs. 4–6. 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 panels (a)–(d) of Fig. 7, 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 Fig. 7, 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.

6. 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 microcluster technique. The goal of this technique is to use a summarized microcluster 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 also presented methods for performing evolution analysis with the resulting clusters. Since clustering is a natural way of performing data summarization, the approach is able to provide interesting insights into the evolving clusters. We presented case studies which show the interesting clusters which were determined, as well as the evolution of the underlying clusters. We also present experimental results illustrating the effectiveness and efficiency of the method.

Acknowledgements

The work of the second and third authors are supported in part by NSF through grants IIS 0905215, DBI-0960443, OISE-0968341 and OIA-0963278. The research of the first author was sponsored in part by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-09-2-0053. The views and conclusions in this document are those of the authors and should not be interpreted as representing the official policies, either express or implied, of the Army Research Laboratory or the United States Government. The US Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

REFERENCES

[1] C. Aggarwal, N. Ta, J. Feng, J. Wang, and M. J. Zaki, XProj: a framework for projected structural clustering of XML documents, In KDD Conference, San Jose, California, USA, 2007.
[2] D. Cook and L. Holder, Mining Graph Data, John Wiley & Sons Inc, 2007.
[3] G. Flake, R. Tarjan, and M. Tsioutsiouliklis, Graph clustering and minimum cut trees, Internet Math 1(4) (2003), 385–408.
[4] T. Dalmagas, T. Cheng, K.-J. Winkel, and T. Sellis, Clustering XML documents using structural summaries, Lect Notes Comput Sci 3268 (2005), 547–556.
[5] X. Yan and J. Han, CloseGraph: mining closed frequent graph patterns, I ACM KDD Conference, Washington, DC, USA, 2003.
[6] X. Yan, H. Cheng, J. Han, and P. S. Yu, Mining significant graph patterns by scalable leap search, In SIGMOD Conference, Vancouver, BC, Canada, 2008.
[7] M. J. Zaki and C. C. Aggarwal, XRules: An Effective Structural Classifier for XML Data, In KDD Conference, Washington, DC, USA, 2003.

[8] C. Aggarwal and H. Wang, Managing and Mining Graph Data, Springer, 2010.
[9] S. Guha, R. Rastogi, and K. Shim, CURE: an efficient clustering algorithm for large databases, In ACM SIGMOD Conference, Seattle, Washington, USA, 1998.
[10] A. Jain and R. Dubes, Algorithms for Clustering Data, New Jersey, Prentice Hall, 1998.
[11] L. Kaufmann and P. J. Rousseeuw, Finding Groups in Data: An Introduction to Cluster Analysis, New York, John Wiley & Sons, Inc., 1990.
[12] T. Zhang, R. Ramakrishnan, and M. Livny, BIRCH: An Efficient Data Clustering Method for Very Large Databases, In ACM SIGMOD Conference, Montreal, Quebec, Canada, 1996.
[13] M. Rattigan, M. Maier, and D. Jensen, Graph Clustering with Network Sructure Indices, In ICML Conference, Corvalis, Oregon, USA, 2007.
[14] B. W. Kernighan and S. Lin, An efficient heuristic procedure for partitioning graphs, Bell Syst Tech J 49 (1970), 291–307.
[15] A. A. Tsay, W. S. Lovejoy, and D. R. Karger, Random sampling in cut, flow, and network design problems, Math Oper Res 24(2) (1999), 383–413.
[16] D. Gibson, R. Kumar, and A. Tomkins, Discovering Large Dense Subgraphs in Massive Graphs, In VLDB Conference, Trondheim, Norway, 2005.
[17] Z. Zeng, J. Wang, L. Zhou, and G. Karypis, Out-of-core coherent closed quasi-clique mining from large dense graph databases, ACM Trans Database Syst 31(2) (2007), 1–40.
[18] A. Abou-Rjeili and G. Karypis, Multilevel algorithms for partitioning power-law graphs, In IEEE International Parallel and Distributed Processing Symposium (IPDPS), Rhodes Island, Greece, 2006.
[19] G. Karypis and V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, In SIAM J. Sci. Comput. 20(1) (1998), 359–392.
[20] S. Raghavan and H. Garcia-Molina, Representing web graphs, In ICDE Conference, Bangalore, India, 2003, 405–416.
[21] C. Aggarwal, J. Han, J. Wang, and P. Yu., A framework for clustering evolving data streams, In VLDB Conference, Berlin, Germany, 2003.
[22] L. O'Callaghan, N. Mishra, A. Meyerson, S. Guha, and R. Motwani, Streaming-Data Algorithms For High-Quality Clustering, In ICDE Conference, San Jose, California, USA, 2002.
[23] A. Z. Broder, M. Charikar, A. Frieze, and M. Mitzenmacher, Syntactic clustering of the web, WWW Conference, Comput Netw 29(8–13) (1997), 1157–1166.
[24] N. Alon, Y. Matias, and M. Szegedy, The space complexity of approximating the frequency moments, In ACM Symposium on Theory of Computing, Philadelphia, Pennsylvania, USA, 1996.
[25] G. Cormode and S. Muthukrishnan, An improved data-stream summary: the count-min sketch and its applications, J Algorithms 55(1) (2005), 58–75.
[26] C. Aggarwal, Data Streams: Models and Algorithms, Springer, 2007.