On Dense Pattern Mining in Graph Streams

[Extended Abstract]

Charu C. Aggarwal
IBM T. J. Watson Research Ctr
Hawthorne, NY
charu@us.ibm.com

Yao Li, Philip S. Yu
University of Illinois at Chicago
Chicago, IL
{yli70, psyu}@cs.uic.edu

Ruoming Jin
Kent State University
Kent, Ohio
jin@cs.kent.edu

ABSTRACT

Many massive web and communication network applications create data which can be represented as a massive sequential stream of edges. For example, conversations in a telecommunication network or messages in a social network can be represented as a massive stream of edges. Such streams are typically very large, because of the large amount of underlying activity in such networks. An important application in these domains is to determine frequently occurring dense structures in the underlying graph stream. In general, we would like to determine *frequent and dense patterns* in the underlying interactions. We introduce a model for dense pattern mining and propose probabilistic algorithms for determining such structural patterns effectively and efficiently. The purpose of the probabilistic approach is to create a summarization of the graph stream, which can be used for further pattern mining. We show that this summarization approach leads to effective and efficient results for stream pattern mining over a number of real and synthetic data sets.

1. INTRODUCTION

A number of recent network applications arise in the context of *graph data streams*. In the graph streaming scenario, it is assumed that edges are drawn on a network domain, and *are received in rapid succession*. Such situations are common for network applications, and are particularly challenging. Some examples are as follows: **(a)** In social networks, the participants may be represented by individual nodes. The huge number of interactions between participants are modeled as continuous edges streams. **(b)** In telecommunication networks, each phone number is defined as a node. The calls between participants are modeled as rapid edge streams. **(c)** In communication networks, the nodes correspond to IP-addresses and the edges correspond to communications among these nodes.

In these cases, the node labels (eg. URLs in a web graph) are distinct, but the complexity arises from the fact that the data is received in the form of a stream. The stream scenario creates constraints in terms of the choice of the algorithms which may be used for the mining process. We assume that the graph edge stream is received *more generally* in the form of *edge set streams*. This is a more general model than one which is based purely on the receipt of individual stream edges. For example, many information network applications such as bibliographic networks, military networks, or movie databases receive objects which are expressed as individual graphs. A bibliographic object may contain nodes corresponding to authors, and the edges corresponding to co-authorship relations among them for a *particular publication object* in the stream. Similarly, an object in a movie database may contain nodes, which correspond to actors or genres and the edges corresponding to relationships among them. In general, *any dynamic application which continuously creates small subgraphs of a massive underlying social or information network will create an edge-set stream.*

In this paper, we will design a probabilistic model for mining dense structural patterns in graph streams. Our notion of density is based both on node co-occurrence and edge density. This is particularly useful for problem domains such as social networks, because of the large number of possible combinations of nodes which could be considered relevant patterns. Our probabilistic approach uses min-hash summarization, in which the size of the summary is independent of the length of the data stream. We provide theoretical bounds on the accuracy of this approach, and present experimental results illustrating its effectiveness and efficiency.

This paper is organized as follows. The remainder of this section contains related work. In the next section, we will model the dense pattern mining problem. In section 3, we will present a carefully designed probabilistic algorithm for the problem. The experimental results are presented in section 4. Section 5 contains the conclusions.

1.1 Related Work

The problem of frequent and sequential pattern mining has been studied extensively in the literature in the context of market basket data both for the static and dynamic case [4, 7, 12]. This problem has also been studied in the context of data streams [7, 12]. The problem of frequent pattern mining in graph data has been studied extensively in recent work [2, 6, 10, 11]. Some of these techniques [6, 10] deviate from the standard model of frequent pattern mining and attempt to determine significant subgraphs. These techniques are designed for relatively small graphs which can fit in main memory. Furthermore, these techniques are not designed for determining structurally dense graphs which is the focus of

Proceedings of the VLDB Endowment, Vol. 3, No. 1
Copyright 2010 VLDB Endowment 2150-8097/10/09... $ 10.00.

this paper.

Some methods [8] have been proposed for graph summarization, though these techniques cannot be used directly for dense subgraph mining. The problem of dense graph mining has been studied in the literature in the context of determining dense regions in massive graphs [1, 5, 13]. The problem of determining dense subgraphs across multiple graphs has been studied in [9]. All of the afore-mentioned techniques are designed for the case of static graphs rather than graph streams. In this paper, we will propose the first method for determining significant and dense subgraphs in massive structural streams.

2. MODELING GRAPH PATTERNS

The stream $\mathcal{S}$ is defined as the sequence $G_1 \ldots G_r \ldots$, where each graph G_i is a set of edges. An important assumption here is that the graph is drawn over a *massive domain of nodes*, but the individual edge set G_i contains only a small fraction of the underlying nodes. This *sparsity property* is typical for massive domains. Before defining the pattern mining problem more formally, we discuss some *desired* properties of the mined patterns:
(1) Node Co-occurrence: Since each graph represents a relatively small sequence of edges in the stream, this means that the node set for the graph is also relatively small. This means that only a small fraction of the total set of nodes are included in the graph. We would like to determine nodes which co-occur frequently in the network. We refer to such node sets as pseudo-cliques, since they occur together frequently. We further note that node occurrence is defined in terms of *relative presence*, so that irrelevant patterns are pruned automatically.
(2) Edge Density: Within a given node set, we would like these interactions to be as dense as possible. In other words, we would like to determine pseudo-cliques, in which a *large fraction* of the possible edges are populated.

The concept of node co-occurrence and edge density provides a natural way to determine node patterns which are closely related both in terms of occurrence and linkage. The node co-occurrence over a set of nodes P is defined by a parameter called *node affinity*, and it is defined as follows:

Definition 1. *Let $f_\cap(P)$ be the fraction of graphs in $G_1 \ldots G_n$ in which* **all** *nodes of P occur. Let $f_\cup(P)$ be the fraction of graphs in which* **at least one** *of the nodes of P occur. Then, the node affinity $A(P)$ of pattern P is denoted by $f_\cap(P)/f_\cup(P)$.*

We note that the above definition of node-affinity is focussed on relative presence of nodes rather than the raw frequency. This ensures that only significant patterns are found. This is especially important in the graph domain because the use of absolute frequencies may result in a very large number of patterns, which may not be statistically interesting for the problem at hand. On the other hand, node-affinity is a more challenging measure from an algorithmic point of view. We define the edge density $D(P)$ of the node set P as follows:

Definition 2. *Let G_i be a graph which contains all nodes of P. Let $h(P, G_i)$ denote the fraction of the $|P| \cdot (|P| - 1)/2$ possible edges defined on P which are included in the edge set E_i of G_i. Then, the value of $D(P)$ is defined as the average of $h(P, G_i)$* **only over** *those graphs which contain* **all** *nodes in P.*

We note that both $A(P)$ and $D(P)$ are parameters which are drawn from the range $(0, 1)$. The closer these parameters are to 1, the more significant the graph pattern is for mining purposes. This suggests a natural way to define the significant pattern mining problem with the use of two threshold parameters (θ, γ) on node correlation and edge density.

Definition 3. *A set of nodes P is said to be be (θ, γ)-significant, if it satisfies the following two* node-affinity *and* edge-density *constraints: (a) The node-affinity $A(P)$ is at least θ. In other words, $A(P) \geq \theta$. (b) The edge-density $D(P)$ is at least γ. In other words, $D(P) \geq \gamma$.*

The above constraints are quite different from the case of the frequent pattern mining problem, and therefore a new approach needs to be designed. Furthermore, computational efficiency is an important concern for the stream scenario. In order to achieve our goal of finding significant patterns, we will use a probabilistic approach which utilizes the sparsity property of the underlying graphs. We will first discuss a solution which requires two passes over the data set. Then, we will discuss how to consolidate the two passes into a single one, so that the approach can be effectively used for data streams.

One of the algorithmic advantages of our model is that it can leverage the sparsity of the individual graphs in the stream in order to determine relevant patterns. This is not the case of the more general dense pattern mining problem in which such sparsity does not exist. The sparsity will be leveraged with the use of a *min-hash* approach which can effectively capture the co-occurrence behavior in the underlying graph structure in an efficient way. The min-hash approach has been used for the problem of finding interesting 2-itemsets in [4], and this paper will extend it to the dense-pattern mining problem.

3. PROBABILISTIC PATTERN MINING

The dense pattern mining algorithm requires two phases. The first phase determines the correlated node patterns. These are defined as patterns for which the affinity is greater than the user-specified threshold θ. The second phase determines the subset of those node patterns which satisfy the edge-density constraint with user-specified threshold γ. We will first describe a simplified algorithm in which each of these phases will require one pass. Then, we will discuss how to consolidate the two passes into a single pass, so that the technique can be used for a data stream.

We can model the *node inclusion data* for our graph as follows. Consider a binary data set with n rows and N columns. The rows correspond to graph identifiers, and the columns correspond to node identifiers. An entry takes on the value of 1 if the corresponding node is included in the graph. We would like to determine the probability that all entries in a given subset S of columns take on the value of 1, even if one of them takes on the value of 1. The basic idea of the min-hash approach is to implicitly sample subsets of columns for which at least one of the entries takes on the value of 1, by examining the first row for which at least one entry in subset S takes on the value of 1 after a row permutation process. The min-hash approach (implicitly) generates a random hash-value for each row in the data. Then it sorts each column by this hash-value. For each column in the subset S, it determines the first row number (or graph-identifier) for which the entry takes on the value of 1. We refer to this index as the *min-hash index* of that column. The probability that all columns take on the value of 1 (given that any column takes on the value of 1) is simply equal to the probability that the min-hash index for the subset S of columns is the same. This probability can be estimated by repeating this process for k different hash values. In such cases, it is possible to estimate the probability that the subset of nodes S occurs together in the graph, even if one of them occurs together. Note that this is simply an estimate of the affinity-probability $A(S)$. We do not explicitly use the tabular representation because of the space-inefficiency of such a representation for sparse data. The min-hash approach is a space- and time-efficient way to generate all possible groups of nodes satisfying the affinity property.

The overall algorithm proceeds by maintaining a set of min-hash statistics, which are constantly updated as the stream graphs are received. We maintain a set of $k \cdot N$ running minimum hash values together with a set of $k \cdot N$ corresponding hash indices. Specifically, we have k hash indices for each of the N nodes. *We will see that the size of the min-hash (and the corresponding accuracy bounds) are independent of data stream size, and therefore this approach scales particularly well for larger and larger data streams.* We scan the graphs in the data one-by-one and perform the following operations. We generate k random hash values independently for each graph. For each graph, we sequentially process its nodes in order to update their hash values and indices. We update the corresponding running hash values and indices if the newly generated hash values are less than the (corresponding) current minimum values. At the end of the process, we have a set of k independently generated minimum hash values together with the corresponding indices for each node. We create a table with k rows and N columns. For the ith row and jth column, we set its value to the ith hash index for the jth node. Note that every entry in this $k \times N$ table will be in the range $[1, n]$, where n is the total number of graphs. Furthermore, the value of k is chosen so that this table is significantly smaller than the original graph data, and can typically be held in main memory.

The process of creating a min-hash sample has the effect of transforming the *affinity-based mining problem* on the original data set, to a *support-based mining problem* on the transformed data set. Specifically, the problem of finding coherent patterns in the original data set is transformed to the following problem on the min-hash sample:

Problem 1. *Determine all subsets of columns for which the corresponding min-hash indices are the same for at least $\theta \cdot k$ rows.*

We note that this problem is closely related to the frequent pattern mining problem, since the former can be transformed to the latter. This transformation can be performed as follows. For each row, we break it up into a set of pseudo-transactions. Each pseudo-transaction contains the set of node indices which take on the same hash index in a particular row. Since each row may create more than one pseudo-transaction, the transformed data will contain a set of at least k pseudo-transactions. Thus, we have a set of pseudo-transactions which are drawn on the node indices which are drawn from 1 through N. Let us denote this transaction set by $\mathcal{T}$. Therefore, Problem 1 can be transformed to the following:

Problem 2. *Determine all frequent patterns in $\mathcal{T}$ with support at least $\theta \cdot k$.*

We note that since this is the standard version of the frequent pattern mining problem, we can use any of the well known algorithms for frequent pattern mining. Furthermore, since the frequent patterns are determined on a compact min-hash sample, the underlying transaction set $\mathcal{T}$ is main memory resident. This ensures that the process is very efficient. Since we used a randomized method to create the transaction set $\mathcal{T}$, this approach will not lead to exact results. Next, we provide a probabilistic bound on the error of this approach.

Lemma 1. *Let us consider a set of nodes P for which $A(P) \leq \theta$. Let $A'(P)$ be the affinity probability estimated with the use of k min-hash samples. Then, the probability that $A'(P) \geq (1+\delta) \cdot \theta$ is given by $e^{-k \cdot \theta \cdot \delta^2/4}$. Here e is the base of the natural logarithm.*

Proof. We note that if the result holds true for the *boundary* case when $A(P) = \theta$, then it also holds true for the case when $A(P) \leq \theta$. Let Z_i represent the 0-1 random variable which determines whether or not the pattern P is included in the ith sample. Then, the estimated affinity probability is defined as $A'(P) = \sum_{i=1}^{k} Z_i$. We have effectively expressed $A'(P)$ as a sum of indicator random variables, which makes it a candidate for the Chernoff bound. The expected value of the random variable $A'(P)$ is $\mu = k \cdot \theta$ for the boundary case when $A(P) = \theta$. We can use the upper-tail Chernoff bound to estimate this probability as at most $e^{-\mu \cdot \delta^2/4}$. The result follows. □

We can derive an analogous result with an identical proof (but lower-tail bound):

Lemma 2. *Let us consider a set of nodes P for which $A(P) \geq \theta$. Let $A'(P)$ be the affinity probability estimated with the use of the probabilistic approach for k min-hash samples. Then, the probability that $A'(P) \leq (1-\delta) \cdot \theta$ is given by $e^{-k \cdot \theta \cdot \delta^2/2}$.*

3.1 Determining Dense Patterns

Once the coherent node patterns have been found, they can be used in order to determine the dense patterns in the data. Let us assume that the total number of coherent patterns that have been found so far is denoted by m. A straightforward way of generating the dense subgraphs is to use a single pass over the graph database in order to count the fraction of graphs which are γ-dense for each frequent pattern. This can however be prohibitively expensive for large databases. The time complexity of this is $O(n \cdot m \cdot p)$ operations, where p is the average number of edges in each graph in the original database. This is because the approach would require us to examine each edge in each database graph one by one and determine if the ends of that edge belong to each of the m node patterns. When the database size n and the number of coherent patterns m is large, the time complexity of performing these counts can be prohibitive. For example, for a database with $n = 10^7$, $m = 10^5$, and $p = 100$, the number of edge checks required is 10^{14} (hundred trillion). A modest "back-of-the-envelope" calculation suggests that if we assume 50 cycles for each edge-check operation, the time required by a 1 GHz computer would be at least $5 * 10^6$ seconds, or 5.8 days. Clearly, if we expand the database size or the number of patterns, the approach can become impractical in terms of running time.

It turns out that we can make good use of the min-hash table generated in the first phase for finding coherent node sets which have the required edge-density. More specifically, we use the transaction set $\mathcal{T}$ for doing this. The main idea is to generate a new *graph fragment database* from the transaction set $\mathcal{T}$. We note that each transaction $T \in \mathcal{T}$ corresponds to a particular input graph (or graph identifier), and consists of a subset of nodes from that graph. We create a graph fragment which consists of all the edges between this subset of nodes, and add it to the fragment database $\mathcal{F}$. We note that the creation of the graph fragments from the node transactions requires one more scan over the database. This scan is required in order to determine the *edges* incident on the high affinity node sets. These edges are used in order to create the corresponding induced graph fragments. Thus, one graph fragment is created for each transaction in $\mathcal{T}$. We make the following observation:

PROPERTY 1. *The computed edge-density for a given subset of nodes S from the fragment database $\mathcal{F}$ is an unbiased estimate of the edge-density in the original graph database.*

This observation is easy to prove, since the graph fragments are samples from the original data, and they do not use the edge distribution in the sampling process. Since the graph fragment database is significantly smaller than the original database, it is much more efficient to perform the validation on $\mathcal{F}$. As in the case of the transaction database, we can assume that the graph fragment database $\mathcal{F}$ can be held in main memory, and therefore more efficient counting algorithms are possible. Furthermore, we know that each coherent-node subset is guaranteed to be present at least $\theta{\cdot}k$ times in the fragment database, because of the properties of the first phase. This property will also help us develop bounds on the accuracy of our edge-density estimate.

The graph fragment database is used in order to compute the edge density estimates for the different node subsets as follows. We assume that we already have the frequencies of presence of the difference coherent node subsets in the fragment database. This is available as an output of the frequent pattern mining process in the first phase on the transaction set $\mathcal{T}$. It remains to determine the fraction of these node containments, which are also dense from the edge perspective. In order to do so, we maintain a counter associated with each coherent node-pattern. This counter determines the number of times that **both** the containment and the edge-density conditions hold true. We can scan through the graph fragment database one by one. For each graph fragment, we increment the counter for a coherent node-pattern, if both conditions are satisfied. We note that aside from the smaller size of the database, the fact that we are working only with graph fragments rather than the individual graphs is useful from the perspective of computational efficiency.

We further note that the graph fragment database $\mathcal{F}$ has the nice property that all the coherent node patterns have a support of at least $\theta{\cdot}k$. This ensures that a sufficient number of samples are available for each coherent node pattern in order to test the edge-density property. This also ensures that it is possible to determine a bound on the accuracy of the edge-density estimation.

LEMMA 3. *Let P be a pattern for which at least $\theta \cdot k$ supporting samples are available in the min-hash scheme. Let the true density $D(P)$ be less than γ. Then, the probability that the pattern P appears as a false positive with estimated edge density of $D'(P)$ at least $\gamma \cdot (1 + \delta)$ is given by at most $e^{-2\cdot\theta\cdot k\cdot\gamma^2\cdot\delta^2}$.*

PROOF. Let W be a random variable which represents the sum of the corresponding fraction of edges in each graph. Note that the fraction of edges in each graph is a random variable which lies in the range $(0, 1)$. The mean of this random variable is less than γ, since the true density is less than γ. Since, the variable W can be expressed as an i.i.d. sum of bounded random variables, this is a candidate for the use of the Hoeffding inequality. Then, by using $L \geq \theta \cdot k$ samples, the estimated density must exceed the expected density by at least $R = L \cdot \gamma \cdot \delta$. By using the Hoeffding's inequality, this probability is at most $e^{-2\cdot R^2/L} = e^{-2\cdot L\cdot(\gamma\cdot\delta)^2}$. By substituting the inequality for L, the result follows. □

We can derive a similar result with the use of an identical proof (but using the lower tail bound).

LEMMA 4. *Let P be a pattern which has affinity probability of at least θ in the entire data stream, and for which at least $\theta{\cdot}k{\cdot}(1{-}\delta)$ supporting samples are available in the min-hash scheme. Let the true density $D(P)$ be at least γ. Then the probability that the pattern P appears as a false negative with estimated edge density of $D'(P)$ at most $\gamma \cdot (1 - \delta)$ is given by at most $e^{-2\cdot\theta\cdot k\cdot(1-\delta)\cdot\gamma^2\cdot\delta^2}$.*

Next, we will combine the results for the affinity probability and density constraints in order to determine the probabilities of δ-false positives and δ-false negatives. We first define the concept of a δ-false positive and δ-false negative.

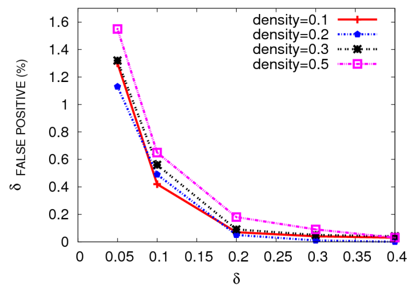
DEFINITION 4. *A pattern P is a δ-false positive, if it does not satisfy either the affinity or edge-density constraint in the overall data stream, and the corresponding constraint is satisfied in the min-hash sample by a factor of at least $(1{+}\delta)$.*

DEFINITION 5. *A pattern P is a δ-false negative, if it satisfies both the affinity and edge-density constraint in the overall data stream, and the corresponding constraint is not satisfied in the min-hash sample by a factor of at least $(1{-}\delta)$.*

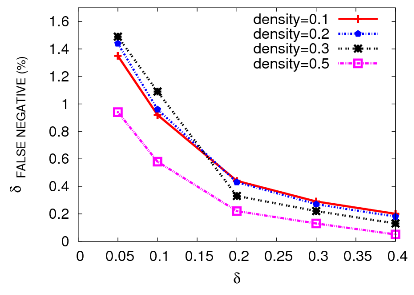
We note that δ-false positives and negatives become less desirable for larger values of δ. For smaller values of δ such errors are reasonably acceptable, since they do not affect the perceived quality of the output too much. We Will now use our previous results in order to derive the probability of δ-false positives and δ-false negatives.

THEOREM 1. *The probability of a pattern P being a δ-false positive is given by at most $e^{-k\cdot\theta\cdot\delta^2/4} + e^{-2\cdot\theta\cdot k\cdot\gamma^2\cdot\delta^2}$.*

PROOF. The probability that a pattern is a false positive on the basis of at least the affinity probability is given by *at most* $e^{-k\cdot\theta\cdot\delta^2/4}$ (this expression is an upper bound since we are ignoring the effects of the density constraint). The probability that a pattern is a false positive on the basis of at least the density constraint is given by $e^{-2\cdot\theta\cdot k\cdot\gamma^2\cdot\delta^2}$. The probability of the union of these two events is at most the sum of the probability of these two events. The result follows. □



(a) δ-false positives



(b) δ-false negatives

Figure 1: DBLP data set, δ-error variation with δ (different densities, affinity = 0.2, Min-Hash Sample Size= 800)

THEOREM 2. *The probability of a pattern P being a δ-false negative is given by at most* $e^{-k\cdot\theta\cdot\delta^2/2}+e^{-2\cdot\theta\cdot k\cdot(1-\delta)\cdot\gamma^2\cdot\delta^2}$.

PROOF. In order for a pattern P to be a δ-false negative, at least one of two mutually exclusive events must happen: **(a)** The affinity probability of pattern P is greater than θ in the data stream, but shows up with a affinity probability of less than $\theta\cdot(1-\delta)$ in the min-hash sample. Ignoring the effects of the density constraint, an upper bound on the probability of this event is *at most* $e^{-k\cdot\theta\cdot\delta^2/2}$. **(b)** The affinity probability of pattern P is at least $\theta\cdot(1-\delta)$. However, the edge-density for pattern P is less than $\gamma\cdot(1-\delta)$ Ignoring the effects of the affinity probability, the probability of this is *at most* $e^{-2\cdot\theta\cdot k\cdot(1-\delta)\cdot\gamma^2\cdot\delta^2}$.

The probability of a δ-false negative is the sum of the probabilities of the above two mutually exclusive events. The result follows. □

3.2 Extension to Data Streams

The approach can be easily extended to the case of data streams. The main purpose of the second pass is to reconstruct the graph fragments, which are attached to the transactions. The graph fragments can be reconstructed with the use of the edges which are specific to the particular graph which is referred to in the corresponding transaction. Since each fragment is specific to a *particular input graph*, it is possible to keep track of these fragments during the first-pass, which creates the min-hash index. This saves the time for the second pass. Recall that in the process of min-hash index creation, we need to store the graph identifier along with the min-hash value. In this case, the only additional information which needs to be stored along with the graph identifier is the set of edges which are incident to that particular node and graph identifier combination. Therefore, if the (i,j)th entry in the min-hash table corresponds to the graph[1] $G_{r(i)}$ and node j, then we need to store the adjacency list for the jth node in graph $G_{r(i)}$. This information is sufficient to completely reconstruct the graph fragments.

The storage of additional adjacency lists increases the storage requirement of the min-hash index. However, this additional storage requirement is small, because it is assumed that each *individual graph* is sparse and has a relatively small adjacency list. Thus, the number of distinct edges adjacent to a node may be very large *over the entire data stream*, but is quite modest *over a single graph.* For example, the *total* number of co-authors of a given author node in the DBLP data set may be large, but the number of co-authors which are specific to a particular paper may be extremely small. It is these kinds of *specific* adjacent nodes which need to be stored in the transaction database. Therefore, the size of this extended min-hash structure (with fragment database) is quite small.

Next, we derive the storage requirement of this representation in terms of the different parameters. When the adjacency lists are not stored, the total storage requirement of the min-hash index is $n\cdot k$. This is independent of the number of elements in the data stream. Let m be the average number of nodes in the different graphs. We note that the size of the adjacency list is smaller than the size of the corresponding graph. Therefore, the average storage requirement of a min-hash index (with storage of the adjacency list) will typically be less than $n\cdot m\cdot k$. Since m is not dependent on the graph domain size (which is large), but on the size of the individual graph itself, it implies that the value of m is relatively small in the sparse scenario that is implied by practical applications. Therefore, such an overhead could be easily supported over practical values of m and k.

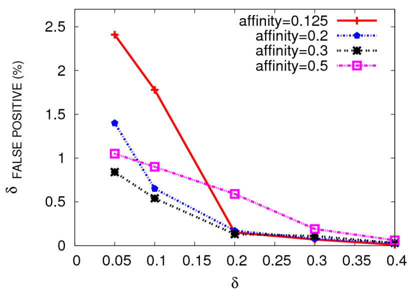
4. EXPERIMENTAL RESULTS

The dense pattern mining algorithm was tested for effectiveness and efficiency. We present results on two real data sets (DBLP data set[2] and IBM Sensor Stream data set) here. The detailed descriptions of the data sets are provided in the Appendix. We also present the results for an additional synthetic data set in the Appendix.

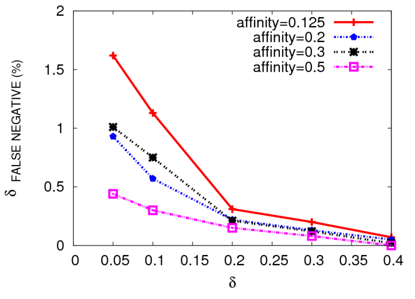
Since our approach is an approximation of the exact method with the use of a min-hash approach, it is useful to determine the number of false positives and false negatives which are incurred with the scheme. Therefore, we determined the number of δ-false positives and δ-false negatives determined by the algorithm. While false-positives are easy to identify with the use of direct validation over the data streams, false negatives are much more difficult to identify, because we need to know the true set of patterns which satisfy these constraints. We note that the true set of patterns can always be determined with the use of a brute-force level-wise algorithm on the overall data set. This is possible, since the affinity constraint satisfies downward closed properties. Of course, such a solution is too time-consuming and requires multiple passes over the data; therefore it is not possible to use it for the stream scenario. However, it is useful for

[1]Note that $r(i)$ is the min-hash index for the entry, and also the identifier for the input graph referred to by the min-hash index.

[2]Available at www.charuaggarwal.net/dblpcl/gstream.txt

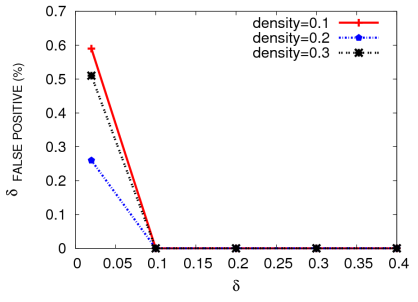


(a) δ-false positives

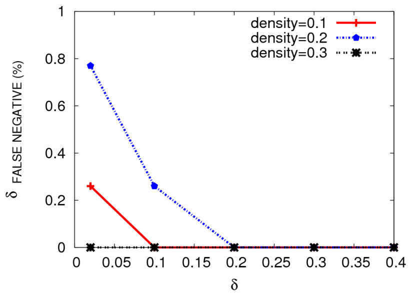


(b) δ-false negatives

Figure 2: DBLP data set, δ-error variation with δ (different affinities, density = 0.5, Min-Hash Sample Size=800)

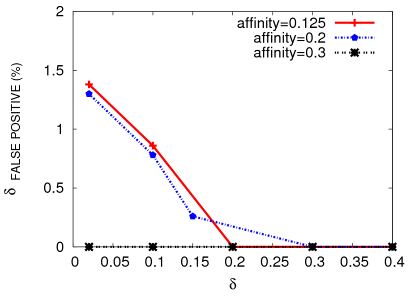


(a) δ-false positives

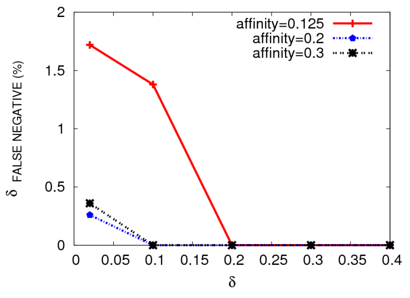


(b) δ-false negatives

Figure 3: Sensor data set, δ-error variation with δ (different densities, affinity = 0.2, Min-Hash Sample Size= 800)



(a) δ-false positives



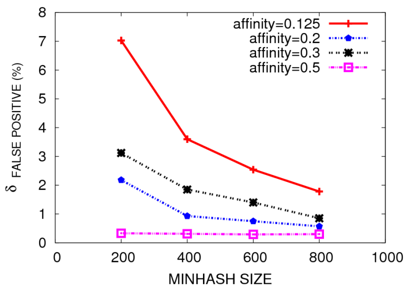
(b) δ-false negatives

Figure 4: Sensor data stream, δ-error variation with δ (different affinities, density = 0.05, Min-Hash Sample Size=800)

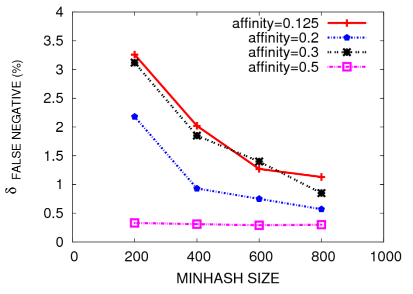
validation purposes. Therefore, we determined the percentage of δ-false positives and δ-false negatives obtained by the scheme and reported them. *We note that δ is a logical (evaluation) parameter only, and is not one of the inputs to the algorithm.* For higher values of δ, the same output yields a smaller number of false positives and false negatives. We will examine the behavior of the obtained outputs over different values of the logical parameter δ. We will present results over different values of the affinity and the density parameters. We also tested the sensitivity of the method to increasing values of the min-hash data structure size.

4.1 Effectiveness and Performance Results

As discussed above, we tested the effectiveness of the schemes in terms of the δ-false positive and δ-false negative variation with δ. In Figures 1(a) and 1(b), we (respectively) plot the variation in the number of δ-false positives and δ-false negatives over different values of the parameter δ for the DBLP data stream. The value of δ is illustrated on the X-axis, whereas the false positive (or false negative) rate is illustrated on the Y-axis. Each of the curves in the two figures illustrate the behavior for different values of the density, when the affinity was fixed at 0.2. We used a min-hash sample size of 800. In each case, the percentage of false positives and false negatives was less than 2% for all values of δ. For example, when we used $\delta = 0.05$, we obtained a false positive (and negative) rate of only about 1.6%. The result did not vary very much over different values of the density parameter. We have illustrated similar results by varying the affinity parameter in Figures 2(a) and 2(b) respectively. In this case, the density was fixed at 0.5. The min-hash sample size was fixed at 800. The results are quite similar to the case when the affinity was fixed and the density was varied. While the results were often superior for lower values of the

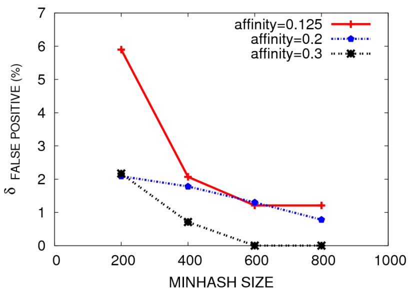


(a) δ-false positives

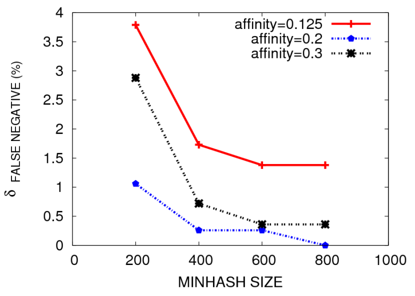


(b) δ-false negatives

Figure 5: DBLP data stream, δ-error variation with Min-Hash Sample Size (different affinities, density= 0.5, $\delta = 0.1$)



(a) δ-false positives



(b) δ-false negatives

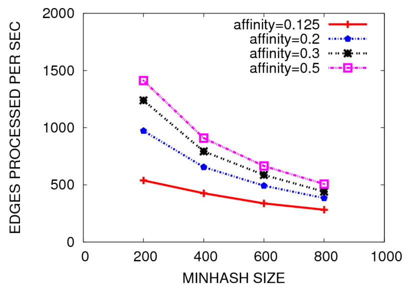
Figure 6: Sensor data stream, δ-error variation with Min-Hash Sample Size (different affinities, density= 0.05, $\delta = 0.05$)

affinity, the trends were not significant enough to overcome the variations within a randomized scheme. Therefore, in some cases, the trends were reversed.

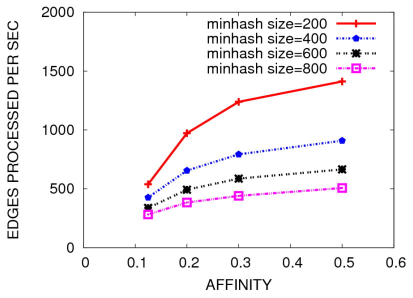
We also tested the approach for the case of the IBM sensor data stream. The results for the sensor data stream with increasing value of δ, and with varying affinity (or density) are illustrated in Figures 3(a), 3(b), 4(a) and 4(b) respectively. For the first pair of figures, the value of the affinity was fixed at 0.2. For the second pair of figures the value of the density was fixed at 0.05. As in the previous case, the δ-false positives were lower than 1.5% in all cases. Furthermore, the variation with affinity and density was not very significant in these cases. This is consistent with the behavior of the DBLP data set.

Since the size of the minimum hash data structure plays a key role in the accuracy of the scheme, we tested the effectiveness of the scheme with increasing minimum hash-size. The results for the DBLP data stream are illustrated in Figures 5(a) and 5(b) respectively. The min-hash size is illustrated on the X-axis, whereas the error is illustrated on the Y-axis. The result was tested for different affinities, when the density was fixed at 0.5, and the value of δ was fixed at 0.1. We tested the scheme for different values of the affinity. We note that the range of sizes for the minimum hash was quite compact and varied between 200 and 800. It is evident that with increasing size of the min-hash data structure, the accuracy of the scheme increases. This is reasonable to expect, because a larger size of the minimum hash table increases the statistical accuracy of representation. Therefore, it also increases the statistical accuracy of determining the dense patterns. In all cases, when we used a minimum hash table size of about 800, it was found that the δ-false positive and the δ-false negative rates were typically less than 2%. The results for the sensor data streams are illustrated in Figures 6(a) and 6(b) respectively. The results are quite similar to the case of the DBLP data stream. It is evident that in all cases, the use of larger minimum-hash sizes increased the accuracy of the scheme. At the same time, the absolute error rate continued to be lower than 2%, when a min-hash size of 800 was used.

We also tested the efficiency of the scheme with increasing affinity and min-hash sample sizes. We picked these two parameters to show the variation, because the scheme was relatively insensitive to the other main input parameter (density). In Figures 7(a) and 7(b), we have illustrated the processing rate variations with increasing affinity and min-hash size respectively for the DBLP data stream. The processing rate in terms of the number of edges per second is illustrated on the Y-axis, and the affinity (or min-hash size) are illustrated on the X-axis. It is clear that the scheme is extremely efficient, and is able to process thousands of edges per second. Similar results for the sensor and synthetic data streams are illustrated in Figures 8(a) and 8(b). It is evident that the processing rate increased with increased values of the affinity. This is essentially because more patterns are found at lower values of the affinity, and therefore this increases the processing time. The only case for which the curves were insensitive with affinity values was for the case of the sensor data stream. This was because the number of patterns generated was more insensitive to the affinity value for the particular case of the sensor data stream. As a result, the processing rates also did not vary much with affinity. The processing rates also varied linearly with the min-hash sample size. This suggests that the dense pattern mining scheme can be implemented efficiently with min-hash samples of modest size. Since we have already shown that the scheme is quite effective for these choices of parameters,

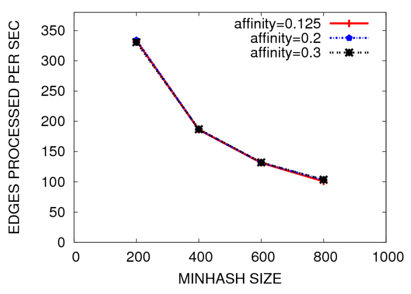


(a) Variation with Min-Hash Sample Size (density=0.5)

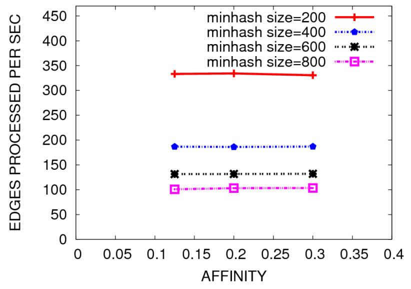


(b) Variation with Affinity (density=0.5)

Figure 7: Processing Rates for DBLP data stream



(a) Variation with Min-Hash Sample Size (density=0.05)



(b) Variation with Affinity (density=0.05)

Figure 8: Processing Rates for Sensor data stream

it suggests that our dense stream pattern mining scheme is robust to choice of parameter size.

5. CONCLUSIONS

In this paper, we designed a method for mining dense patterns in graph streams. We designed a min-hash approach in order to summarize the graph stream and leverage the summarization for efficiently determining patterns. We presented theoretical bounds quantifying the accuracy of the technique. Furthermore, the technique is extremely efficient because it requires only a linear pass over the data stream, and the computations during this pass are very limited. We presented the effectiveness and efficiency of our approach on a number of real and synthetic data streams.

6. ACKNOWLEDGEMENTS

This work is supported in part by NSF through grant IIS-0905215.

7. REFERENCES

[1] J. Abello, M. G. Resende, and S. Sudarsky. Massive quasi-clique detection. *LATIN Conference*, 2002.

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

[3] Y. Chi, H. Wang, P.S. Yu, and R. R. Muntz. Moment: Maintaining closed frequent itemsets over a stream sliding window. ICDM Conference, 2004.

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

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

[6] H. He, and A. K. Singh. Efficient Algorithms for mining significant substructures from graphs with quality guarantees. *ICDM Conference*, 2007.

[7] R. Jin, and G. Agrawal. An algorithm for in-core frequent itemset mining on streaming data. *ICDM Conference*, 2005.

[8] S. Navlakha, R. Rastogi, and N. Shrivastava. Graph summarization with bounded error. *SIGMOD*, 2008.

[9] J. Pei, D. Jiang, and A. Zhang. On mining cross-graph quasi-cliques. *KDD*, 2005.

[10] S. Ranu, and A. K. Singh. GraphSig: A scalable approach to min- ing significant subgraphs in large graph databases. *ICDE Conference*, 2009.

[11] X. Yan, and J. Han. CloseGraph: Mining Closed Frequent Graph Patterns. *KDD*, 2003.

[12] J. Xu Yu, Z. Chong, H. Lu, and A. Zhou. False positive or false negative: Mining frequent itemsets from high speed transactional data streams. *VLDB Conference*, 2004.

[13] Z. Zeng, J. Wang, L. Zhou, and G. Karypis. Out-of-core Coherent Closed Quasi-Clique Mining from Large Dense Graph Databases. *ACM TODS*, Vol 31(2), 2007.

APPENDIX

A. DATA SET DESCRIPTIONS

The detailed data set descriptions are as follows:
(1) DBLP Data Stream: 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. 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 $1 \leq i, j \leq q$. We removed the authors with an extremely small number of papers, because these were not interesting for the dense pattern mining algorithm. We also removed some outlier papers with a large number of authors. Each conference paper along with its edges was considered a graph. The data set is presented temporally in the form of a stream.
(2) Sensor Data Stream: This data aet contained information about local traffic on a sensor network from a private corporate network. This local traffic 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. The data set contained a stream of intrusion graphs from June 1, 2007 to June 3, 2007. Each consecutively received set of edges, which corresponds to an intrusion attack type was converted into a single graph. The graphs were presented temporally in order to create a stream.
(3) Synthetic Data Stream: 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 $a = 0.5$, $b = 0.2$, $c = 0.2$, and $S = 16$ (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 graphs is derived from a normal distribution with mean $\mu = 10$ and standard deviation $\sigma = 2$. 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 can 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 $\kappa = 20$ 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:

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

$\alpha_1 \ldots \alpha_r$ 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. The derived values $S_1 \ldots S_r$ are used in order to define the node probabilities.

B. ADDITIONAL EXPERIMENTAL RESULTS FOR SYNTHETIC DATA STREAM

In this section, we present some experimental curves for the synthetic data set. The detailed charts are presented in Figures 9 to 16. These figures are exactly analogous to those presented for the case of the real data sets. The results on the synthetic data set are quite similar to the case of the real data sets. The results are presented for completeness, and additional evidence of effectiveness of the algorithm.

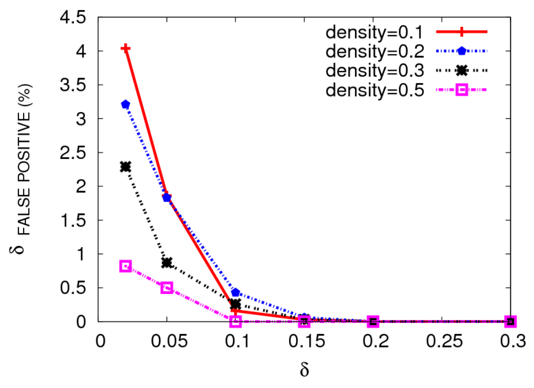


Figure 9: Synthetic data stream, δ-false positive variation with δ (different densities, affinity = 0.2, Min-Hash Sample Size= 800)

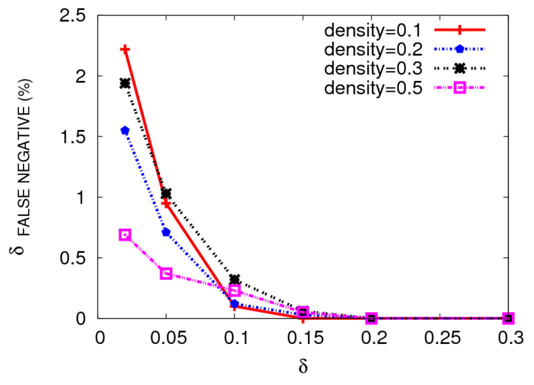


Figure 10: Synthetic data stream, δ-false negative variation with δ (different densities, affinity = 0.2, Min-Hash Sample Size=800)

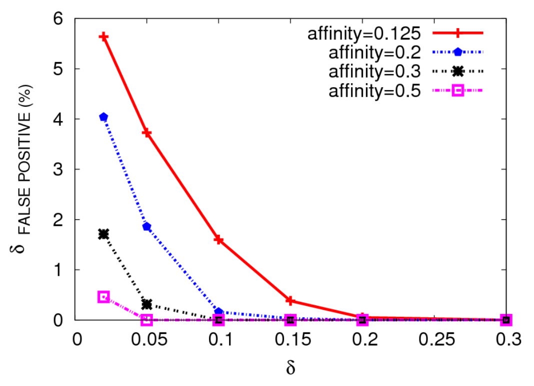


Figure 11: Synthetic data stream, δ-false positive variation with δ (different affinities, density = 0.1, Min-Hash Sample Size=800)

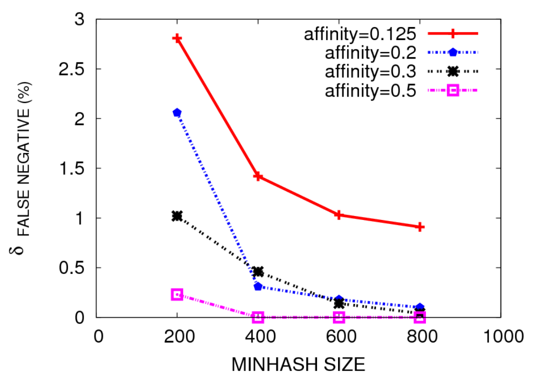


Figure 14: Synthetic data stream, δ-false negative variation with Min-Hash Sample Size (different affinities, density= 0.1, $\delta = 0.1$)

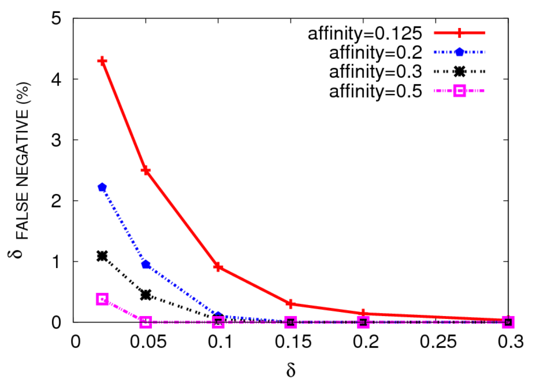


Figure 12: Synthetic data stream, δ-false negative variation with δ (different affinities, density = 0.1, Min-Hash Sample Size=800)

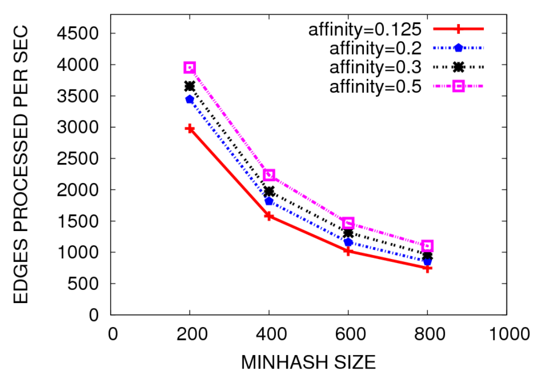


Figure 15: Synthetic data stream, Running time variation with Min-Hash Sample Size (different affinities, density= 0.1)

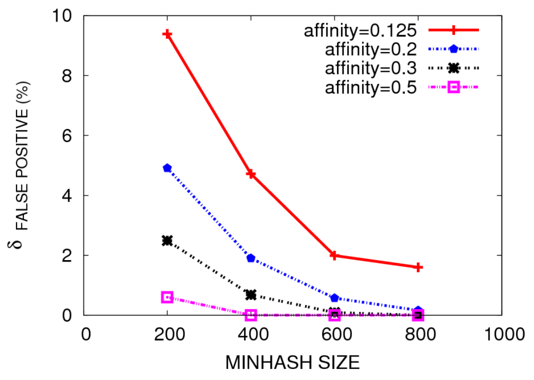


Figure 13: Synthetic data stream, δ-false positive variation with Min-Hash Sample Size (different affinities, density= 0.1, $\delta = 0.1$)

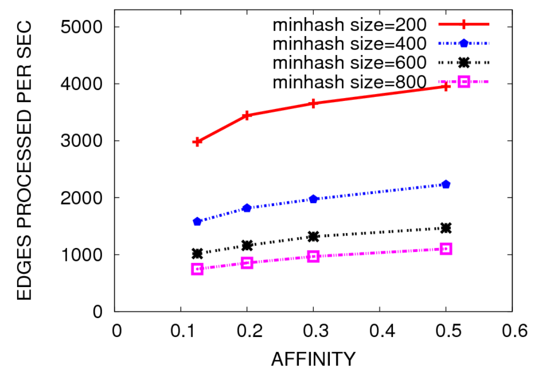


Figure 16: Synthetic data stream, Running time variation with affinity (different Min-Hash Sample Size, density= 0.1)