2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)

Query-Friendly Compression of Graph Streams

Arijit Khan
Nanyang Technological University, Singapore

Charu Aggarwal
IBM T. J. Watson Research Center, NY, USA

***Abstract*—We study the problem of synopsis construction of massive graph streams arriving in real-time. Many graphs such as those formed by the activity on social networks, communication networks, and telephone networks are defined dynamically as rapid edge streams on a massive domain of nodes. In these rapid and massive graph streams, it is often not possible to estimate the frequency of individual items (e.g., edges, nodes) with complete accuracy. Nevertheless, sketch-based stream summaries such as Count-Min can preserve frequency information of high-frequency items with a reasonable accuracy. However, these sketch summaries lose the underlying graph structure unless one keeps information about start and end nodes of all edges, which is prohibitively expensive. For example, the existing methods can identify the high-frequency nodes and edges, but they are unable to answer more complex structural queries such as reachability defined by high-frequency edges.**

To this end, we design a 3-dimensional sketch, gMatrix that summarizes massive graph streams in real-time, while also retaining information about the structural behavior of the underlying graph dataset. We demonstrate how gMatrix, coupled with a one-time reverse hash mapping, is able to estimate important structural properties, e.g., reachability over high frequency edges in an online manner and with theoretical performance guarantees. Our experimental results using large-scale graph streams attest that gMatrix is capable of answering both frequency-based and structural queries with high accuracy and efficiency.

I. Introduction

In various practical settings, graphs are drawn over a massive set of nodes, and edges arrive rapidly in the form of a *graph stream*. Many domains of data such as web graphs, transportation networks, IP networks, and social networks belong to this category. As reported in the official Twitter blog (blog.twitter.com/2011/numbers), the average number of tweets sent per day was 140 million in 2011, which was around 1620 tweets per second. Similarly, according to Digital Buzz statistics on Facebook (digitalbuzzblog.com/facebook-statistics-stats-facts-2011/), there are about 2 million new friend requests and 3 million messages sent every 20 minutes.

In order to process fast streaming data, a growing number of applications relies on devices such as network interface cards, routers, switches, cell processors, FPGAs, and GPUs [19]; and usually these devices have very small on-chip memory. Whether in specialized hardware or in conventional architectures, efficient processing of rapid and massive stream data requires creation of a succinct synopsis in a single pass over that stream [7]. Such synopsis must be updated incrementally with new incoming items. The synopsis should also support online query answering. Due to its smaller size compared to the original stream, it is often not possible to answer queries with complete accuracy — reducing synopsis size increases the efficiency, but also reduces the accuracy.

IEEE/ACM ASONAM 2016, August 18-21, 2016, San Francisco, CA, USA
978-1-5090-2846-7/16/$31.00 ©2016 IEEE

In the literature, there are various synopsis structures for massive and rapid data streams, such as sketch [9], [7], [23] and space-saving [15]. These synopsis structures are suitable for frequency estimation, heavy-hitter, and top-k queries. However, a direct adaptation of the aforementioned stream compressing techniques over graph-edge or graph-node streams lose the underlying structural information in the graph data. Hence, these techniques cannot answer structural queries such as finding the aggregated frequency of all edges in a subgraph induced by a given set of nodes, or reachability queries defined over high-frequency edges.

In this paper, we develop a method for real-time synopsis construction of massive graph streams, which retains information about the structural and frequency behavior of the graph dataset. Our synopsis structure, gMatrix is *general-purpose* in the sense that it can answer all the above-mentioned frequency-based queries such as edge (or, node) frequency estimation, heavy-hitter, and top-k queries. In addition, it can process a diverse set of structural queries including aggregated subgraph edge frequency and reachability queries. Clearly, this is the problem of interest for most practical scenarios.

II. Related Work

The problem of synopsis construction has been studied extensively in data stream literature in the context of a variety of techniques such as wavelets, histograms, and sketches [7]. However, these techniques cannot be used for applications which utilize the structural properties of the graph data.

The problem of graph synopsis construction has found increasing interest because of its relevance to the web, XML, biological data, transportation, and social networks [18]. The graph stream model has also been explored for various analytical queries, such as clustering, classification, pattern mining, and matching [2], [5]. Techniques for compressing *static* graphs have been discussed in [16], [17], though these methods cannot be easily generalized to graph streams. Various query-specific graph compression methods are proposed for static graphs, such as [11] considers pattern matching and reachability queries.

Recently, the synopsis construction of graph streams has been considered in gSketch [23] when a stream sample is available. Clearly, such an approach will not work if either a workload is not available, or the stream evolves over time. Besides, gSketch was designed to answer only edge-frequency based queries, and it cannot answer more complex graph structural queries. The method in [10] constructs synopsis of graph streams for estimating the degree distributions of the nodes. Ahn et. al. [3] studied graph sketch for answering structural queries such as connectivity, minimum-cost spanning tree, maximum weighted matching, and subgraph pattern matching. Feigenbaum et al. also considered various structural graph queries, e.g., graph matching and shortest path queries with the semi-streaming model [12]. For a detailed survey, see [14]. Different from the aforementioned classical structural queries [10], [12], [3], we study novel graph structural queries that couple graph structure with edge frequencies. We design a more *general-purpose* synopsis, gMatrix that maintains structural and frequency properties of the underlying graph data; and hence, it can answer both frequency-based as well as structural queries over rapid graph streams. In addition, gMatrix does not require any *apriori* stream sample.

Very recently, TCM [22] sketch was proposed, which is the most relevant to our work. Although they aim at preserving the graph structure, unlike ours they did not consider reverse hashing queries, e.g., find all heavy-hitter edges, which would be critical for community detection over graph streams. Besides, they did not consider alternative options to extend sketch and space-saving synopses, such as concatenating end-node ids to construct an edge id, that we shall discuss in this paper.

III. Preliminaries

We assume that the data from the graph $G = (V, E)$ is received in the form of an *edge stream*. Each node $i \in V$ is drawn from the set $N = \{1, 2, \ldots, n\}$, and every edge $(i, j) \in E$ is a directed edge. The incoming graph stream contains elements $(i_1, j_1, f_{i_1 j_1})$, $(i_2, j_2, f_{i_2 j_2})$, $\ldots$, $(i_t, j_t, f_{i_t j_t})$, $\ldots$. Here, $(i_t, j_t, f_{i_t j_t})$ denotes the arrival of the t-th edge-stream with an associated frequency $f_{i_t j_t}$. In many applications, the value of $f_{i_t j_t}$ is naturally set to 1, though we assume an arbitrary frequency in order to retain the generality of our results. For example, in a telecommunication application, the frequency $f_{i_t j_t}$ may denote the number of seconds in a phone conversation from a person i_t to another person j_t starting at time-stamp t. An edge can appear multiple times in the stream. When we issue a query, e.g., find the frequency of an edge (i, j), we are looking for the aggregated frequency of that edge in the stream so far. While sketch-based methods including ours can be adapted for time-window queries [1], we do not consider them in this work. Also note that our results can be easily generalized to undirected graphs by assuming that the edge (i, j) is always lexicographically ordered.

A. Queries

We introduce our queries over graph streams as follows.

1) **Edge Frequency Query:** Determine the frequency of an edge (i, j).
2) **Heavy-hitter Edge Query:** Determine all edges with frequency larger than a given threshold F.
3) **Node Aggregated-Frequency Query:** Determine the aggregated frequency of all incoming/ outgoing edges for a query node i.
4) **Heavy-hitter Node Query:** Determine all nodes with aggregated-frequency (based on all its incoming/ outgoing edges) larger than a given threshold F.
5) **Subgraph Aggregated-Frequency Query:** Determine the aggregated frequency of all edges in a subgraph corresponding to a given subset of nodes S.
6) **Reachability Query:** Find if a source node is reachable to a destination node via edges having frequency larger than a given threshold F.

The first two queries involve only edge frequencies; and therefore, could be answered with existing sketch-based approaches, e.g., Count-Min [9], or gSketch [23]. Similarly, the third and fourth queries involve only node frequencies; and hence, could be answered by using a sketch over node frequencies. However, the last two queries attempt to determine the structural behavior of the graph stream based on edge frequencies. Hence, these queries could not be answered using existing sketches unless one keeps start and end nodes for all edges, which is prohibitively expensive. Therefore, *our contribution lies in designing one general-purpose sketch synopsis*, gMatrix, *which is capable of answering all the above queries over graph streams*.

Applications. The aforementioned queries are useful in a variety of applications. For example, in an email network, it is useful to determine frequent sender-receiver pairs (heavy-hitter edge query), most active receivers (heavy-hitter node query), or connected components of the graph that communicate frequently with one another (reachability query). Connected components formed by high-frequency edges, in fact, identify the communities in the network. As it will be evident from the reachability query defined over high-frequency edges, one can reconstruct the approximate structure of the high-frequency portion of the graph with gMatrix. Once the approximate structure of the high frequency portion of the graph has been reconstructed, it is possible to use a clustering algorithm, or any other graph mining algorithms such as frequent subgraph pattern mining over graph streams [2].

Our designed gMatrix is a 3- dimensional matrix sketch. In stead of a radically different solution, gMatrix is built following the principle of Count-Min [9], thereby retaining all the properties and benefits of it, such as (1) ensuring one-sided error guarantee, i.e., the estimated edge-frequency is always an *overestimation* of its true frequency. (2) The accuracy improves for high-frequency items in a skewed stream. For ease in further discussion, we first introduce Count-Min, which was proposed for frequency estimation.

B. Count-Min Sketch

In Count-Min, a hashing approach is utilized to approximately maintain the frequency counts of a large number of distinct items in a data stream (Figure 1). We use $w = \lceil \ln(1/\delta) \rceil$ pairwise independent hash functions, each of which maps 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 2-dimensional array with $h \times w$ cells of length

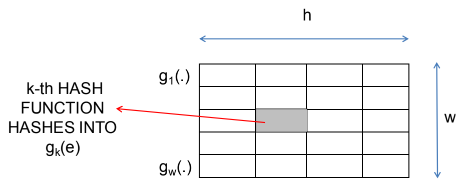


Fig. 1. Count-Min sketch for data streams

h and width w. Each hash function corresponds to one of w 1-dimensional arrays with h cells each. Next, consider a 1-dimensional data stream with elements drawn from a massive set of domain values. When a new element of the data stream is received, we apply each of the w hash functions to map onto a number in $[0 \ldots h-1]$. The count of each of these w cells is incremented by 1. 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 compute the minimum value among all these cells. Let c_t be the true value of the count being estimated. We note that the estimated count is at least equal to c_t, since we are dealing with non-negative counts only, and there may be an over-estimation because of collisions among hash cells. It has been shown in [9] that for a data stream with L arrivals, the estimate is at most $c_t + \epsilon \cdot L$ with probability at least $1-\delta$. In the event that the items have frequencies associated with them, we increment the corresponding count with the appropriate frequency. The same bounds hold in this case, except that we define L as the sum of the frequencies of the items received so far.

To identify the top-k frequent items from sketches, several efficient techniques were also proposed, such as the reversible hashing [21], which we use in this work, as well as heap [4], hierarchical sketches, and group testing [8].

We note that it is easy to adapt the sketches to estimate the frequencies of graph-edges by assigning each distinct edge a unique edge-id and hashing it into the sketch structure. This is indeed the same approach proposed in gSketch [23]. However, such an approach loses the structural behavior of the underlying graph data, because it cannot discern the connectivity relationships between nodes. In order to preserve important structural relationships, *it is important to at least approximately maintain the incidence behavior of edges on nodes.* To this end, we design gMatrix.

C. gMatrix Sketch

gMatrix is a 3-dimensional sketch of the graph data. The two dimensions in the sketch correspond to the source and destination nodes. Each hash function defines a mapping of the node set N to an integer in the range $[0, h-1]$. Thus, if the k-th hash function is $g_k(\cdot)$, then the edge (i,j) is mapped to $(g_k(i), g_k(j))$. As in the previous case, we use w pairwise-independent hash functions, and therefore k may range from 1 to w. Our hash table is a 3-dimensional construct with $h \times h \times w$ cells. Correspondingly, we define a cell-coordinate as (p,q,r), where p and q are indices of the *hash-mapped nodes* and r

[3]As one may realize, gMatrix is not the only way how sketches can be extended to preserve edge connectivity information. We shall discuss these alternative options in Sec. III-F, and also highlight their disadvantages so to emphasize our design of gMatrix.

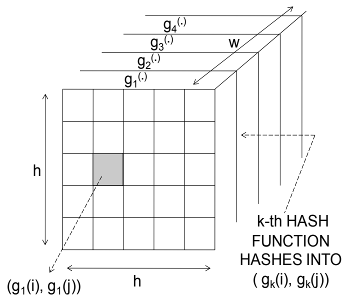


Fig. 2. gMatrix sketch for graph streams

is the index of the hash function being used for the mapping. The value of the cell-coordinate (p,q,r) is the integer counter $V(p,q,r)$ and it maintains a hash frequency of the hash-edge (p,q) for the case of the r-th hash function. Figure 2 provides an example of gMatrix synopsis [3].

The value of h is typically much smaller than the number of nodes, and this is the key to the compression realized with gMatrix. For example, let us consider a graph containing 10^7 nodes. In such a case, if we use $h = 10^3$ and $w = 10$, the sketch would contain 10^7 cells, which corresponds to about 40 MB. This is quite modest and can even be stored in the on-chip memory of most modern hardware, e.g., FPGA (Xilinx Spartan-6 LX with 128MB memory), and GPU units (NVIDIA GeForce GT 640 with 2048MB memory).

D. Updating the gMatrix Synopsis

The process of updating gMatrix is fairly straightforward. We start by initializing each cell in the sketch structure to 0. For each incoming edge (i,j) with frequency f_{ij}, we compute the *hash-edge* $(g_k(i), g_k(j))$ for each value of $k \in \{1 \ldots w\}$. Then, we increment the frequency of each of these w cells by f_{ij}. This process is repeated for each incoming edge.

Space and Time Complexity. The size of gMatrix is $\mathcal{O}(h^2 w)$. The time complexity to update an incoming edge stream in gMatrix is $\mathcal{O}(2w)$. Our complexity results show that gMatrix can be built in one pass over the graph stream, and can also be updated incrementally with the arrival of a new edge stream.

E. Hash Function and Reverse Hashing

For gMatrix to be effective, the hash functions are required to be pairwise independent [9]. In addition, for processing of heavy-hitter and structural queries as mentioned in Sec. III-A, we need to compute the *reverse hash mapping* [21]. Given a hash function g, we define its reverse hash mapping g^{-1} as:

$$g^{-1}(p) = \{i : g(i) = p\} \tag{1}$$

Thus, we select a hash function such that (1) one can quickly compute the corresponding reverse hash mapping, and (2) the reverse hash mapping size $|g^{-1}(p)|$ is relatively small.

In our implementation of gMatrix, we assume that the graph-nodes have integer identifiers, and we select the modular hash function as given in Equation 2.

$$g(i) = ((a \times i + b) \mod P) \mod h \tag{2}$$

P is a prime larger than the maximum value of any node identifier. We select a and b uniformly from the interval $(1, P-1)$. Note that the range of our hash function g is $(0, h-1)$, where h is the length of gMatrix. The reason for selecting the modular hash function is two-fold: (1) our hash functions for different values of a and b are pairwise independent [9], and (2) the reverse hash mapping size is only $\lfloor P/h \rfloor$, and it can be computed efficiently using the extended Euclidean algorithm [6] with time complexity $\mathcal{O}(\lfloor P/h \rfloor \log P)$. We emphasize that the original Count-Min sketch implementation also uses modular hash functions. For a more sophisticated hashing scheme, one may apply the Galois Extension Field operations [21].

Reverse Hash Mapping Computation. We next discuss how the extended Euclidean algorithm can be applied to compute our reverse hash mapping.

$$g^{-1}(p) = \{i : g(i) = p\} \tag{3}$$

Recall that we used the modular hash function, i.e.,

$$g(i) = ((ai + b) \mod P) \mod h \tag{4}$$

P is a prime larger than the maximum value of any node identifier (i.e., larger than the maximum value of i). We select a and b from the interval $(1, P-1)$. The range of hash function, denoted by h, is smaller than P. Therefore, we essentially find all i' such that the following holds for some k.

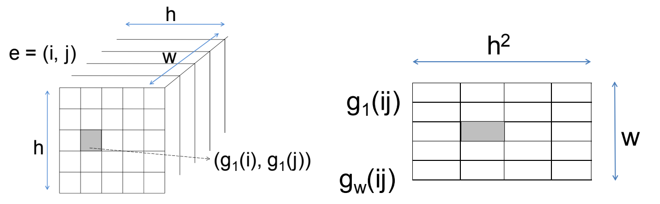
$$(ai' + b) \mod P = p + kh \tag{5}$$

Here, $k \geq 0$ can be any integer satisfying $(p + kh) < P$. Therefore, the reverse hash mapping size is $\lfloor P/h \rfloor$. Each i' can be computed efficiently using the extended Euclidean algorithm [6] with time complexity $\mathcal{O}(\log P)$. Thus, the overall time complexity of our reverse hash mapping is $\mathcal{O}(\lfloor P/h \rfloor \log P)$. Since P is a prime number larger than the total number of nodes, the complexity of our reverse hashing is log-linear to the number of nodes. However, as discussed later in gMatrix applications (Sec. IV) and experiments (Sec. V), reverse hashing is applied only once before a series of complex structural queries (e.g., reachability) can be answered in an online manner. Hence, we believe that one-time computation of reverse hash mapping is tolerable in our approach [21].

F. Alternative Design Options

Before discussing gMatrix applications, *we consider some alternative options to extend sketch and space-saving synopses for achieving similar functionalities as* gMatrix.

As an example, to preserve edge connectivity information, one may construct an edge-id by concatenating its source and destination node-ids (Figure 3(b)), and then use a Count-Min sketch with w hash functions, each having a range of h^2. This technique would work well for subgraph aggregated-frequency query (query 5, Sec. III-A). However, such an approach will be inefficient for reverse hash mapping (required for reachability query: query 6, Sec. III-A) compared to gMatrix. As explained in [21], assume that one cell of each hash function has



(a) gMatrix (b) Count-Min with concatenated end-node ids

Fig. 3. gMatrix vs. alternative design options with Count-Min: The end-nodes are concatenated to form the edge-ids in Figure (b), which is more expensive for reverse hash mapping.

frequency more than the threshold, and we want to identify the actual graph edges that are hashed to those cells. In case of Count-Min sketch with the setting as discussed above, this would require $w(\frac{n^2}{h^2})$ intersection computations, where n is the number of nodes in the graph. This is because each cell would produce $\frac{n^2}{h^2}$ candidate edges via reverse hash mapping, and the potential answer set would be an intersection of those edge sets. However, in case of gMatrix, the source and destination nodes for each edge are hashed with two hash functions, each having a range of h. Therefore, the intersections could be computed over the node sets than over the edge sets, and this would require $2w(\frac{n}{h})$ intersection computations for finding the potential answer set. This shows the benefit of gMatrix compared to Count-Min with concatenation of source and destination node-ids.

Another possibility to retrieve the high-frequency edges along with their start and end nodes will be to directly apply the space-saving synopsis [15]. However, space saving has high storage overhead per data item, and it only monitors the high-frequency data items. Hence, it is not very effective to estimate the frequency of those items which are not stored in space saving [20]. Therefore, we do not consider the aforementioned alternative design options.

IV. GMatrix Applications

In the following, we introduce our techniques for resolving the aforementioned queries (Sec. III-A) with gMatrix.

A. Edge Frequency Queries

To determine the frequency of edge (i, j) using gMatrix, we compute the frequencies of w different cells which correspond to the coordinates $(g_k(i), g_k(j), k)$ and values $V(g_k(i), g_k(j), k)$ for w different values of k. The minimum of these values is returned as the estimate of the frequency $Q(i, j)$ of the edge (i, j). We denote this frequency estimate as $\overline{Q(i, j)}$. Next, we shall illustrate the accuracy of our approach.

Probabilistic Accuracy Guarantee. We show that the probability of incurring an error beyond a pre-defined limit is small.

Theorem 1. *Let the total frequency of edges received so far in the graph stream be denoted by L. Let $Q(i, j)$ be the true frequency of the edge (i, j) over the course of the entire data stream, and let $O(i, j)$ be the sum of the frequencies of the edges incident on i or j. Let $\epsilon \in (0, 1)$ be a very small fraction. Consider a* gMatrix *structure with node-compression length h*

and width w. Then, with probability at least $1 - 1/(h^2 \cdot \epsilon/2)^w$, the estimated frequency $\overline{Q(i,j)}$ is related to the true frequency by the following relationship:

$$Q(i,j) \leq \overline{Q(i,j)} \leq Q(i,j) + L \cdot \epsilon + h \cdot O(i,j) \cdot \epsilon \qquad (6)$$

Proof. We note that $\overline{Q(i,j)}$ is always an over-estimate on $Q(i,j)$ since all frequencies are assumed to be non-negative. Any incoming edge, for which both end points are neither i nor j, is equally likely to map onto one of h^2 cells in the data. The probability that any incoming edge maps onto a particular cell is given by $1/h^2$. Therefore, the expected number of *spurious* edges for which the end points are neither i nor j, yet they get mapped onto the cell $(g_k(i), g_k(j), k)$ is given by at most L/h^2. Let the number of such spurious edges for the k-th hash function be denoted by the random variable R_k. Then, by using the Markov inequality, we have:

$$P(R_k > L \cdot \epsilon) \leq E[R_k]/(L \cdot \epsilon) \leq 1/(h^2 \cdot \epsilon) \qquad (7)$$

Next, we examine the case of spurious edges for which at least one end point is either i or j. The number of such edges is $O(i,j)$ and the expected number of such edges which map onto the entry $(g_k(i), g_k(j), k)$ is given by $O(i,j)/h$. Let U_k be the random variable representing the number of such edges. Then, by using the Markov inequality, we have:

$$P(U_k > O(i,j) \cdot h \cdot \epsilon) \leq E[U_k]/(h \cdot O(i,j) \cdot \epsilon) \leq 1/(h^2 \cdot \epsilon) \quad (8)$$

Note that Equations 7 and 8 can be combined as follows:

$$P(R_k + U_k > L \cdot \epsilon + O(i,j) \cdot h \cdot \epsilon) \leq 2/(h^2 \cdot \epsilon) \qquad (9)$$

This uses $P(A \cup B) \leq P(A) + P(B)$. For the estimate to violate Equation 6, we require the above condition to be true for all $k \in (1, w)$. The probability that this is true is given by at most $1/(h^2 \cdot \epsilon/2)^w$. The result follows. □

Implication of Accuracy Guarantee. We note that for the above probability to be less than 1, we need the value of $h^2 > \frac{1}{\epsilon/2}$. Furthermore, since w occurs in the exponent, the robustness of the above result can be easily magnified even for modest values of w. For example, if $h^2 = \frac{20}{\epsilon}$, a choice of $w = 9$ ensures that the above result is true with probability at least $1 - 10^{-9}$. This is quite acceptable for most practical scenarios. On the other hand, in Equation 6, the error term is relatively small if the true frequency $Q(i,j)$ is a significant fraction of the aggregated frequency L. For real-world streams, which often has a skew [13], this holds for the high-frequency items. Thus, gMatrix is particularly well in estimating the frequency for the high-frequency edges.

Time Complexity. The time to estimate the frequency of an edge is $\mathcal{O}(2w)$, where w is the number of hash functions.

B. Heavy-Hitter Edge Queries

We discuss our techniques for resolving heavy-hitter edge queries, that is, to retrieve all edges with frequency larger than a given threshold. It is important to note that a direct application of Count-Min over edge streams would only retrieve heavy-hitter edge-ids, but not the corresponding start and end nodes. *With* gMatrix, *we can retrieve not only the heavy hitter edges, but also their start and end nodes*. This provides an example of how we can reconstruct the approximate structure of the high-frequency portion of the graph using gMatrix.

Since gMatrix is a probabilistic synopsis, we cannot deterministically find all the heavy-hitters. Therefore, we design a method to retrieve no false negatives, but an edge may be a false positive. Correspondingly, we return the probability that the edge is a false positive. We design a two-step approach.

- In the first phase, we scan gMatrix and determine all *hash-edges* for which the frequency is at least F under different hash functions.
- For each frequent hash-edge (p, q) under some hash function g_k, we compute the set of all possible frequent graph-edges by applying the reverse hash mapping technique, which we described earlier in Sec. III. Let us denote by edge-set $E_k = \{(i,j) : i \in g_k^{-1}(p), j \in g_k^{-1}(q), V(g_k(p), g_k(q), k) \geq F\}$. Finally, we compute the intersection among different E_k edge-sets for all $k \in (1, w)$, and that provides us the heavy-hitter edges.

We note that the second phase is the most expensive step in our method. Therefore, we propose optimization techniques to improve the efficiency of heavy-hitter edge queries.

Query Optimization for Heavy-hitter Edge Queries. Our query optimization technique is based on an efficient representation of the edge-sets E_k, and we perform our intersection over the set of source and destination nodes rather than directly over the set of edges. Below, we first define a *cross-edge*.

Definition 1. *A* cross-edge *is denoted by $S(Q_1, Q_2)$, where Q_1 and Q_2 are two sets of nodes. This is also defined as the set of all edges (i,j), such that $i \in Q_1$ and $j \in Q_2$. Formally,*

$$S(Q_1, Q_2) = \{(i,j) : i \in Q_1, j \in Q_2\} \qquad (10)$$

We note that each of the frequent hash-edges determined in the first phase is a cross-edge. Therefore, set-edges E_k can be written as: $E_k = \bigcup S(Q_1, Q_2)$, where the union is taken over all $Q_1 = g_k^{-1}(p)$ and $Q_2 = g_k^{-1}(q)$, such that $V(g_k(p), g_k(q), k) \geq F$. The implicit representation $S(Q_1, Q_2)$ is much more convenient and compact, since the value of $|Q_1| \times |Q_2|$ may be much larger than either Q_1 or Q_2. Next, we make the following observation:

Observation 1. *The edge-set function and intersection function satisfy sequence symmetry. In other words, we have:*

$$S(Q_1, Q_2) \cap S(P_1, P_2) = S(Q_1 \cap P_1, Q_2 \cap P_2) \qquad (11)$$

Thus, the intersection of the edge-sets for two different hash functions can be computed efficiently without explicitly enumerating the underlying edges. In order to find the intersection of two edge-sets, we compute the intersection for all combinations of cross-edges of these two edge-sets. Here, we emphasize that if either $Q_1 \cap P_1$ or $Q_2 \cap P_2$ is null, then the corresponding set $S(Q_1 \cap P_1, Q_2 \cap P_2)$ is null. Because of the high selectivity of each hash-cell, the null case is quite common when one computes the intersection of two edge-sets.

Probabilistic Accuracy Guarantee. Finally, we compute the probability that each of these heavy-hitter edges as obtained by our method indeed has frequency greater than F.

Theorem 2. *Let L be the total frequency of edges received so far. Let $O(i,j)$ be the number of edges incident on either i or j. Also assume that the estimated frequency of edge (i,j) is $\overline{Q(i,j)}$ according to our* gMatrix *structure. Then, the probability that the true frequency of edge (i,j) is at least F, is given by at least $1 - min\{1, ((L + h \cdot O(i,j))/(h^2 \cdot (\overline{Q(i,j)} - F)))^w\}$.*

Proof. The true frequency of edge (i,j) would also be at least F, if at most $(\overline{Q(i,j)} - F)$ spurious edges map onto the hash cell $(g_k(i), g_k(j), k)$. We note that the expected number of spurious edges, which map onto the cell $(g_k(i), g_k(j), k)$, depends upon the two cases corresponding to whether or not that edge is incident on i or j. If the edge is not incident on any one of these nodes, the probability is $1/h^2$ of the total frequency L. This is equal to L/h^2. Therefore, by using the Markov inequality, we can derive the probability that more than $(\overline{Q(i,j)} - F)$ such spurious edges map onto that cell, which is given by at most $L/(h^2 \cdot (\overline{Q(i,j)} - F))$. For the second case, if the edge is incident on one of those nodes i or j, the probability of this happening is given by $O(i,j)/(h \cdot (\overline{Q(i,j)} - F))$. The sum of these two probabilities gives an upper bound of this happening over all edges. Therefore, the probability that more than $(\overline{Q(i,j)} - F)$ spurious edges map onto cell $(g_k(i), g_k(j), k)$ in all w hash functions is given by $((L + h \cdot O(i,j))/(h^2 \cdot (\overline{Q(i,j)} - F)))^w$. Note that if this bound is greater than 1, then no interesting bound on the probability is derived. Therefore, the probability that the frequency of edge (i,j) is at least F is given by at least: $1 - \min\{1, ((L + h \cdot O(i,j))/(h^2 \cdot (\overline{Q(i,j)} - F)))^w\}$. □

Implication of Accuracy Guarantee. If the estimated frequency $\overline{Q(i,j)} > F$, and also $\overline{Q(i,j)}$ is a significant fraction of the aggregated frequency L, then the above probability gets close to 1. Thus, heavy-hitter query results would be very accurate for those edges whose estimated frequency is higher.

Time Complexity. The complexity of scanning gMatrix synopsis is $\mathcal{O}(h^2 w)$. Next, let us assume that the k-th hash function ($1 \leq k \leq w$) of gMatrix has total c_k cells with frequency higher than or equal to threshold F. Based on our design of modular hash function (see Equation 2), each of these cells maps to $\lfloor P/h \rfloor$ source nodes and $\lfloor P/h \rfloor$ destination nodes. It requires $\mathcal{O}(2c_k \lfloor P/h \rfloor \log P)$ time to find all these node sets. We recall that P is a prime number larger than the maximum number of nodes in the graph dataset. Finally, we compute the intersection over the node sets as discussed in Equation 11. Therefore, the overall time-complexity of heavy-hitter edge query is $\mathcal{O}(h^2 w + 2\lfloor P/h \rfloor \log P \prod_{k=1}^{w} c_k)$.

Asking Heavy-hitter Edge Queries Differently. We note that the heavy-hitter edge query can be posed in a slightly different way in which we attempt to find those edges for which the frequency is at least F with probability at least $1 - \delta$.

Definition 2. *Determine all edges with frequency greater than F with probability at least $1 - \delta$.*

Nodes	Edges	Agg. Stream Freq.	Max. Edge Freq.	Stream Size	Compressed Stream Size
66M	3 612M	10^{10}	4.43×10^8 (z=1.0) 1.81×10^9 (z=1.2) 3.22×10^9 (z=1.4)	80GB	16.47 GB (z=1.0) 2.37 GB (z=1.2) 0.25GB (z=1.4)

TABLE I
GRAPH STREAM CHARACTERISTICS

This is done by using the same approach, but with resetting the threshold frequency to a higher value: $F + (L + O(i,j) \cdot h)/(h^2 \cdot \delta^{1/w})$. The probabilistic accuracy guarantee is given in Theorem 3. We omit the proof as this is similar to Theorem 2.

Theorem 3. *Let L be the total frequency of received edges. Let (i,j) be an edge with estimated frequency at least $F + (L + h \cdot O(i,j))/(h^2 \cdot \delta^{1/w})$. Then, the probability that the true frequency of the edge is at least F is given by at least $1 - \delta$.*

C. Reachability Queries

The heavy-hitter edge query processing technique can also be used for reachability queries. This query also shows the power of gMatrix in retaining structural information about the graph, which is not achieved by state-of-the-art sketch synopses. The reachability query is defined as follows:

Definition 3. *Determine if a source node is reachable to a destination node via edges that have frequency at least F with probability at least $1 - \delta$.*

This query is fairly straightforward to resolve with the use of the approach discussed in the previous subsection, since we can retrieve not only the heavy-hitter edges, but also their start and end nodes. Thus, in the first step, we determine all edges (along with its start and end nodes) for which the frequency is at least F with probability at least $1 - \delta$ (see Definition 3). Then, we answer reachability queries using these edges. When F is relatively high, due to the presence of skew in the graph stream, there are only few edges with frequency higher than F. This helps us to answer reachability queries over high-frequency edges in an online manner.

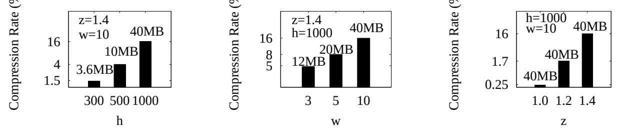
V. EXPERIMENTAL RESULTS

We present experimental results which illustrate the effectiveness, efficiency, and compression rate of gMatrix.

A. Environment Setup

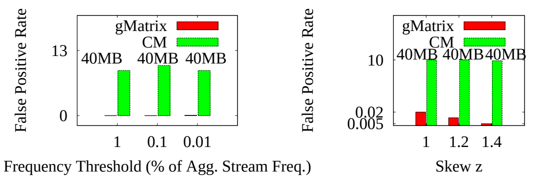
Dataset: We downloaded the *Friendster* graph from snap.stanford.edu (Table I). The edges (undirected) do not have frequency information in this dataset. Hence, we assign frequency to every edge with the Zipf distribution, and vary the skew from $z = 1$ to $z = 1.4$. We demonstrate our results with *Friendster*, because by assigning the edge frequency distribution synthetically, we can test our results for different skew. It has been a standard practise in data stream literature [9], [23] to consider the Zipf frequency distribution. Indeed, as reported in [13], [20], most real-world datasets have skew in the range tested by ours. Due to brevity, we omit experimental results over other datasets, since we found that they are similar to the results with *Friendster*.

The *stream size* in Table I represents the graph-stream size with repetition of edges. The *compressed stream size*, on the



(a) (b) (c)

Fig. 4. gMatrix compression rate over compressed stream



(a) skew=1.4 (b) Freq. Threshold=0.01%

Fig. 5. Accuracy of heavy-hitter edge queries, h=1000, w=10

contrary, is the size of all distinct edges that have non-zero frequency, along with their frequency counts. In our system, each integer consumes 4-bytes of memory space. Hence, for the original stream, each (*possibly repeated*) edge requires 8-bytes corresponding to its source and destination nodes. In the compressed representation, each *distinct* edge consumes 12-bytes, for its two end nodes as well as for its frequency count. Both *original stream* and *compressed stream* can answer our queries with complete accuracy. We shall demonstrate that gMatrix, although a small fraction of original and compressed stream representations, achieves high accuracy.

Queries and Comparing Method: Due to lack of space, we demonstrate our results with two representative queries: heavy-hitter edge query and reachability via heavy-hitter edges. The first query can be answered with existing sketches; and hence, we compare our results with Count-Min [9] by allocating the same amount of storage to both Count-Min and gMatrix. While gMatrix can identify the source and destination nodes of those heavy-hitter edges, Count-Min only returns the corresponding edge-ids. Thus, gMatrix can answer our second query, i.e., reachability over heavy-hitter edges. However, traditional sketches cannot process such structural queries.

System Description. The code is implemented in C++ and the experiments were performed on a single core of 10GB, 2.4GHz Xeon server. One may note that *the* Friendster *stream with skew* 1, *even in its compressed form, cannot be entirely loaded in the main memory of our commodity server*.

B. gMatrix Compression Rate

The compression rate is defined as the ratio of gMatrix size to the size of the *compressed stream*. We recall that the compressed stream representation is much smaller than the original stream, since the former does not have edge repetition.

It is evident from Figure 4 that the size of gMatrix is usually a small percentage of the compressed stream size; and hence, an even smaller percentage of the original stream. In case of skew $z = 1$, gMatrix with $h = 1000$ and $w = 10$ (i.e., 40MB) is only 0.25% of the compressed stream size. We will demonstrate later that these values of h and w are often sufficient to achieve reasonably high accuracy over a wide variety of applications.

Freq. Th. F % Agg. Stream Freq.	Run Time gMatrix	Run Time Count-Min	Memory Use gMatrix	Memory Use Count-Min
1%	28 sec	1 sec	29 MB	0.8 MB
0.1%	149 sec	2 sec	128 MB	5 MB
0.01%	771 sec	7 sec	260 MB	21 MB

TABLE II
HEAVY-HITTER EDGE QUERY: EFFICIENCY, H=1000, W=10, SKEW=1.4

Freq. Th. F (% Agg. Stream Freq.)	Reachability Error
1%	0
0.1%	0
0.01%	0.012

TABLE III
H=1000, W=10, SKEW=1

Skew	Reachability Error
1	0.012
1.2	0.008
1.4	0.004

TABLE IV
H=1000, W=10, FREQ. TH.=0.01%

C. Query Processing: Accuracy and Efficiency

1) Heavy-Hitter Edge Queries: We performed experiments to identify the heavy-hitter edges having frequencies above a given threshold. We denote the value of this threshold frequency as a percentage of the aggregate stream frequency, since the frequency error estimate of an edge is sensitive to the aggregate stream frequency. gMatrix ensures that the reported edges are a superset of the true edges. In all our figures, we report the *false positive rate* for heavy-hitter edge queries.

$$\text{False Positive Rate} = \frac{\text{\# edges incorrectly reported as heavy-hitter edges}}{\text{\# true heavy-hitter edges}}$$

Figures 5 illustrates the accuracy of reported heavy-hitter edges under different skew and frequency thresholds. The results show that the false positive rate for heavy-hitter queries is about 10 times smaller for gMatrix as compared to Count-Min (CM). This is because many high-frequency edges often have the same source or destination nodes. In other words, the distinct number of source nodes or the distinct number of destination nodes over all high frequency edges is usually less than the distinct number of high-frequency edges. Therefore, intersection computation over the node sets in gMatrix is able to reduce more false positives.

However, this reduced false positive rate is achieved at the cost of higher memory and running time (Table II). We recall that the number of node intersections for gMatrix is: $\mathcal{O}(2n/h \prod_{k=1}^{w} c_k)$, whereas for Count-Min, the number of edge intersection is: $\mathcal{O}(e/h^2 \prod_{k=1}^{w} c_k)$. Since, $e < nh$, gMatrix performs more number of intersection operations than that of the Count-Min. On the other hand, gMatrix returns the end nodes of those heavy-hitter edges, which is the key to answer more complex structural queries as illustrated later. Besides, heavy-hitter edge queries are executed periodically and in an off-line manner. Hence, we believe that the additional running time cost is tolerable, and one may use the main-memory of a connected server to satisfy the extra space requirements due to heavy-hitter queries.

2) Reachability Queries: In these experiments, we consider reachability via edges having frequency greater than or equal to a threshold value. *Such reachability queries defined by high-frequency edges cannot be answered using existing sketches.*

Note that the use of gMatrix results in the addition of some spurious edges. Such spurious edges can change the connected components if they occur as "bridge edges" between two components. In order to measure the accuracy of reachability queries over high frequency edges, we first consider the starting and ending nodes of all edges for which gMatrix-estimated frequency is more than a predefined threshold F. Then, we randomly construct 500 distinct pairs of nodes from this set and verify if gMatrix representation and the original graph report the same result in terms of whether or not the pair is connected by edges with frequency more than F. Finally, we report the *reachability error* over these 500 node-pairs, which is defined as the ratio of node-pairs incorrectly reported as reachable over the total number of node-pairs considered.

$$\text{Reachability Error} = \frac{\text{\# node-pairs falsely reported as reachable}}{\text{\# node-pairs queried}}$$

Tables III and IV show our accuracy results for reachability query considering various frequency thresholds and skew. We observe that the reachability error is always less than 0.015, and often zero. These results illustrate that (1) the reachability via high-frequency edges does not change much by spurious edges, and (2) gMatrix is effective at reconstructing the underlying graph structure defined by high-frequency edges.

After we identify the heavy-hitter edges, which is much less than the total number of distinct edges in the graph, each reachability query can be processed in about 0.1 sec. This shows our efficiency in answering *online structural queries*.

D. gMatrix Stream Processing Throughput

In Figure 6, we analyze the stream processing throughputs of gMatrix and Count-Min. Both gMatrix and Count-Min achieve throughput about a few hundreds of edges per millisecond. However, gMatrix throughput is around two-times slower than that of Count-Min. This is because gMatrix performs hashing twice per edge-stream, while Count-Min performs hashing only once per edge-stream.

VI. Conclusions

We presented gMatrix, a sketch synopsis for massive and rapid graph streams. It is the first synopsis which *maintains information about the structural and frequency behavior of the underlying network*. This is achieved with the use of a 3-dimensional sketch structure, which stores information about the node-based structural relationships between different edges. Thus, gMatrix is useful for a variety of structural queries such as the determination of subgraph edge frequencies and reachability over high-frequency edges. Since gMatrix maintains the approximate structure of the high-frequency regions of the underlying graph, we expect that it can be used for a wide variety of structural queries, as long as the node identifiers can be disambiguated with the use of different hash functions. This will be the focus of our future work. One may also consider how to further improve the efficiency of gMatrix using early aggregation techniques and specialized hardware.

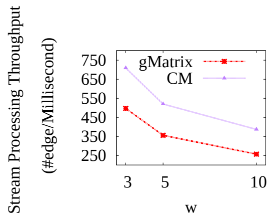


Fig. 6. Throughput of stream processing, z=1.4, h=1000

VII. Acknowledgements

Research of the second author was sponsored by the Army Research Laboratory under Cooperative Agreement Number W911NF-09-2-0053.

References

[1] C. Aggarwal, J. Han, J. Wang, and P. Yu. A Framework for Clustering Evolving Data Streams. In *VLDB*, 2003.
[2] C. Aggarwal, Y. Li, P. S. Yu, and R. Jin. On Dense Pattern Mining in Graph Streams. In *VLDB*, 2010.
[3] K. J. Ahn, S. Guha, and A. McGregor. Graph Sketches: Sparsification, Spanners, and Subgraphs. In *PODS*, 2012.
[4] M. Charikar, K. Chen, and M. Farach-Colton. Finding Frequent Items in Data Streams. In *ICALP*, 2002.
[5] S. Choudhury, L. Holder, G. Chin, A. Ray, S. Beus, and J. Feo. StreamWorks: A System for Dynamic Graph Search. In *SIGMOD*, 2013.
[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. Section 31.2: Greatest Common Divisor. In *Introduction to Algorithms, Second Edition*, pages 859–861. MIT Press and McGraw-Hill, 2001.
[7] G. Cormode, M. Garofalakis, P. J. Haas, and C. Jermaine. Synopses for massive data: Samples, histograms, wavelets, sketches. *Foundations and Trends in Databases*, 4(1–3):1–294, 2012.
[8] G. Cormode and M. Hadjieleftheriou. Finding Frequent Items in Data Streams. In *VLDB*, 2008.
[9] G. Cormode and S. Muthukrishnan. An Improved Data-Stream Summary: The Count-min Sketch and its Applications. *J. of Algorithms*, 55(1), 2005.
[10] G. Cormode and S. Muthukrishnan. Space Efficient Mining of Multigraph Streams. In *PODS*, 2005.
[11] W. Fan, J. Li, X. Wang, and Y. Wu. Query Preserving Graph Compression. In *SIGMOD*, 2012.
[12] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang. On Graph Problems in a Semi-Streaming Model. *Theor. Comput. Sci.*, 348(2-3):207–216, 2005.
[13] N. Manerikar and T. Palpanas. Frequent Items in Streaming Data: An Experimental Evaluation of the State-of-the-art. *Data Knowl. Eng.*, 68(4):415–430, 2009.
[14] A. McGregor. Graph Stream Algorithms: A Survey. *SIGMOD Rec.*, 43(1), 2014.
[15] A. Metwally, D. Agrawal, and A. E. Abbadi. Efficient Computation of Frequent and Top-k Elements in Data Streams. In *ICDT*, 2005.
[16] S. Navlakha, R. Rastogi, and N. Shrivastava. Graph Summarization with Bounded Error. In *SIGMOD*, 2008.
[17] C. Qun, A. Lin, and K. W. Ong. D(k)-Index: An Adaptive Structural Summary for Graph Structured Data. In *SIGMOD*, 2003.
[18] S. Raghavan and H. Garcia-Molina. Representing Web Graphs. In *ICDE*, 2003.
[19] O. Rottenstreich, Y. Kanizo, and I. Keslassy. The Variable-Increment Counting Bloom Filter. *IEEE/ACM Trans. Netw.*, 22(4):1092–1105, 2014.
[20] P. Roy, A. Khan, and G. Alonso. Augmented Sketch: Faster and More Accurate Stream Processing. In *SIGMOD*, 2016.
[21] R. Schweller, Z. Li, Y. Chen, Y. Gao, A. Gupta, Y. Zhang, P. A. Dinda, M.-Y. Kao, and G. Memik. Reversible Sketches: Enabling Monitoring and Analysis over High-speed Data Streams. *IEEE/ACM Trans. Netw.*, 15(5):1059–1072, 2007.
[22] N. Tang, Q. Chen, and P. Mitra. Graph Stream Summarization: From Big Bang to Big Crunch. In *SIGMOD*, 2016.
[23] P. Zhao, C. Aggarwal, and M. Wang. gSketch: On Query Estimation in Graph Streams. In *VLDB*, 2012.