Space Efficient Mining of Multigraph Streams *

Graham Cormode†
Bell Laboratories
cormode@bell-labs.com

S. Muthukrishnan
Rutgers University
muthu@cs.rutgers.edu

ABSTRACT

The challenge of monitoring massive amounts of data generated by communication networks has led to the interest in data stream processing. We study streams of edges in massive communication multigraphs, defined by (source, destination) pairs. The goal is to compute properties of the underlying graph while using small space (much smaller than the number of communicants), and to avoid bias introduced because some edges may appear many times, while others are seen only once. We give results for three fundamental problems on multigraph degree sequences: estimating frequency moments of degrees, finding the heavy hitter degrees, and computing range sums of degree values. In all cases we are able to show space bounds for our summarizing algorithms that are significantly smaller than storing complete information. We use a variety of data stream methods: sketches, sampling, hashing and distinct counting, but a common feature is that we use *cascaded summaries*: nesting multiple estimation techniques within one another. In our experimental study, we see that such summaries are highly effective, enabling massive multigraph streams to be effectively summarized to answer queries of interest with high accuracy using only a small amount of space.

1. INTRODUCTION

The past few years have witnessed an emergence of monitoring applications that has lead to the study of *data streams*, where data is generated rapidly and continuously—in a stream—and one needs to perform real-time analyses on the observed stream of data records. The quintessential application is that of analyzing the traffic on links in an IP network. IP packets traverse links at very high rates (up to millions per second). For our discussion here, each packet p is a tuple (s_p, d_p, b_p) where s_p is the source IP address, d_p is the destination IP address and b_p is the size in bytes of the packet; s_p's and d_p's are integers in the range $0..2^{32}-1$. Data analysis needs to keep pace with the rate at which packets are forwarded in the IP link (typically one every few nano-seconds); at such speeds, it is prohibitive to have large memories [13]. This motivates the need for methods for managing and mining massive streams under severe resource—space, per item processing time—constraints. Fortunately, for such stream analyses, answers that are exact to the last decimal are not required and, in fact, *fast, approximate answers* (with reasonable guarantees on the approximation error) are often preferred. Thus, it is possible to trade accuracy for reduced resource requirements when analyzing massive streams.

The seminal work on these problems [2] presented methods for approximating *frequency moments* such as $F_2 = \sum_j (\sum_k p_{k|s_k=j})^2$ in small space; F_0 is the well-known distinct count and F_2 is the *repeat rate* that has mining applications. Since then, a number of results have been shown estimating quantiles, heavy-hitters, range aggregates, etc. This has led to an emerging theory of data stream algorithms [28]. Some of these techniques have been incorporated into specialized data stream management systems (DSMSs) like Gigascope [11] and others [3] for IP network management. More generally, manipulating stream data involves new stream operators, SQL extensions, query optimization methods, and operator scheduling techniques. General purpose DSMSs like NiagaraCQ, Stanford Stream, Telegraph, Aurora are being developed [4].

Our motivation also lies in analysis of IP traffic data which we view as defining a *communication network*. In communication networks, the pattern of communication is typically represented by a graph. Each node (IP address) represents a communicant, and a (directed or undirected) edge (a packet tuple (s_p, d_p, b_p)) connects two communicants (s_p, d_p) who communicate. Properties of this graph are important for network managers to understand the behavior of their network. For example, nodes of high degree are those that communicate with a large number of other participants in the network. Identifying such nodes in an IP network can indicate unusual activity on that host, such as port scans or virus activity, even though the overall bandwidth for that host may still be low (since each scan requires only a few packets). However, if we are monitoring packet streams, it may be difficult to detect such activity, since there are also very large packet streams between other hosts (e.g. network backups), and the volume of these dominates that of the suspicious communications. Similarly, one may be interested in the frequency moments such as F_0 and F_2 of

*Supported by NSF ITR 0220280, EIA 0205116 and 0354690.

†Bell Labs, 600 Mountain Avenue, Murray Hill NJ 07974. Work done while this author was at DIMACS.

PODS 2005 June 13-15, 2005, Baltimore, Maryland.

their degrees based on the *number* of different communicants rather than the total number of communications between hosts. For example, the second frequency moment is $M_2 = \sum_j (|\{d_k | s_k = j\}|)^2$ and similarly for M_0 that is an analog of F_0.

Formally, in the graph terminology, we see a *multigraph stream* — each edge can occur many times — and we are interested in properties of the underlying graph. If we could filter out multiple occurrences of the same edge, then we could more easily find the nodes of high degree, since they would stand out. But, doing such filtering exactly requires a large amount of space to be devoted to recording which edges have and have not been seen before; this is precisely the challenge with data streams where we do not have the space to memorize all the edges that have been seen.

Surprisingly, despite the robust collection of data stream algorithms known to date, few if any apply to estimating graph aggregates on multigraph streams. In this paper, we study problems of developing new approximate techniques to compute on massive multigraph streams. Even though we have motivated the discussion above in the case when the communication graph is derived from the traffic on IP networks, there are many other multi-graphs: telephone call graphs where nodes are telephone numbers and edges are calls, web graphs where nodes are webpages and edges are web links, etc. Transactions on such graphs—telephone calls, followed web links, IP flows—form the edges of massive (multi)graphs. Processing such edge streams in one pass is of great interest in many applications.

1.1 Problems

We study some of the most basic problems with multigraph streams. Formally, we see edges of a *multigraph* on n nodes in the input stream. Without loss of generality, edges are directed, so (i, j) indicates an edge from i to j. Each edge may be observed in the stream many times, however we only wish to count each edge once. Let m denote the number of (distinct) observed edges. We assume that $m = \Omega(n)$, i.e., n does not count nodes that do not participate in at least one edge. Let $G[i, j] = 1$ if (i, j) appears in the stream, and 0 if it does not. We study the problem of estimating various quantities on the underlying graph G. In particular, let d_i be the degree of i which is $\sum_{j=1}^{n} G[i, j]$. Our aim is to accurately approximate various functions of the d_i. We wish to get *sublinear* algorithms, that is, algorithms that use space and time $o(n)$. This is the well-known cash-register model of data streams [28]. Example problems that we focus on in this paper are:

- *Frequency estimates.* Frequency moments of the d_is, i.e. $M_k = \sum_{i=1}^{n} d_i^k$. In particular, M_0 and M_2 are generalizations of the well known F_0 and F_2 to the multigraph context and stand for count-distinct and repeat rate (Gini index) calculation respectively. M_1 is simply $2m$, the total number of (distinct) edges in the multigraph stream.

- *Heavy hitters.* Finding the largest degrees, that is, finding those i's where d_i is at least some fraction of the sum of the number of nodes. This corresponds to finding say the source IP address that communicates with a large number of destinations, a natural measure of heavy hitters generalized to the multigraph setting.

- *Rangesums.* Computing range sums of d_i, that is, given j and k, computing $\sum_{i=j}^{k} d_i$. This is the indexing problem for multigraph streams; it reduces to the "point" estimate problem when $j = k$, and it is the problem of counting the destinations of a "subnet" which is a hierarchical range of IP addresses associated with individual subnets/organizations, another natural problem. It is also the primitive that we can use to build simple, approximate, equidepth histograms (quantiles) for the distribution of d_i's.

There are two distinct challenges to overcome: first, there is the challenge faced by graphs where we need to coordinate information between multiple nodes but the space we have is sublinear in the number of nodes. This is the common challenge associated with streaming, be it graph or multigraph. Second, statistics we want to keep with sketches or samples are biased by repetitions of the same edge many times on streams since the input is a multigraph. As a result, even problems that are solvable on graph streams become nontrivial on multigraph streams. Computing F_2 on a graph stream can be approximated to $1 \pm \varepsilon$ with probability at least $1 - \delta$ in $O((1/\varepsilon^2) \log(1/\delta) \log n)$ space and time per edge [2]. But this cannot be directly applied to M_2 on multigraphs, where the same edge can appear many times and we do not want the multiple appearances to skew the outcome.

1.2 Previous Work

There has been a large amount of recent work on summarizing and querying massive data streams — see, for example, some of the surveys and tutorials in major conferences [18, 28, 24, 4]. We focus on work relevant to processing massive (multi)graphs. Despite the large amount of interest in data stream algorithms, there has been relatively little focus on (multi)graph streams. In [21], the authors studied special multigraphs and showed $\Omega(n^2)$ lower bounds on space needed for solving many problems. Computing with more general graph properties such as shortest paths, connectivity etc. is provably difficult in the cash register model, so the *semi-streaming* model was proposed [28] where the space is proportional to n, the number of nodes, and sublinear in m, the number of edges. This is of interest in structural analysis of web graphs and in estimating transitivity of binary relations in databases. Some interesting results have been obtained in the semi-streaming model for graph properties such as diameters, spanners etc. [14, 15]. In [6], the authors presented a way to estimate the count of triangles in graphs, using space potentially super-linear, but sublinear if the graphs had certain properties. We are not aware of any prior positive results for multigraphs in one pass. Recently, [7] there has been interest in the "superspreaders" problem, which corresponds to the heavy hitters problem on multigraph streams. The solution presented there relies on an appropriate sampling technique based on fully random hash functions, and is not immediately comparable to our results in terms of parameters ε, δ.

Although there are a few works that have proposed space-efficient summaries for data streams that implicitly combines various synopses (for example, [17] cascades a set of counters within the Flajolet-Martin sketch structure), our explicit construction of *cascaded summaries* is novel and more fully developed.

1.3 Our Contributions

Our main results are space efficient (sublinear) algorithms for the three problems. Keeping approximate per-node information (eg, approximate degree counts) is not allowed in this space limited model. All our algorithms are randomized and approximate, that is, they produce answers to some approximation ε and succeed with probability at least $1 - \delta$. Our results are as follows:

- Our algorithm for computing M_2 of multigraph streams takes space $O(\frac{1}{\varepsilon^4}\sqrt{n}\log n)$ and per-edge processing time $O(\sqrt{n}\log n)$. The space used is sublinear in the number of nodes. This result can be extended to approximating M_k.

- We find all nodes with degree at least ϕm, and report no nodes with degree less than $(\phi - \varepsilon)m$ in space $O(\frac{1}{\varepsilon^3}\log^2 n)$ and time per edge $O(\log^2 n)$. This is the multigraph analog of heavy hitters [8, 26, 10] for graph or numerical streams, but harder, using more than the $O(\frac{1}{\varepsilon}\log n)$ space solutions for non-multigraph streams [8, 26, 10].

- We can compute range sums of degree sequences in space $O((\frac{\log n}{\varepsilon})^3)$ and time per edge $O(\log^2 n)$. The error is less than εn with high probability. This can be used to compute approximate quantiles of the distribution of d_i's and maintain a summary.

Our results are the first sublinear results known for multigraph streaming problems. In addition to above algorithmic contributions, we make following technical contributions:

1. We introduce *cascaded data stream summaries*. Data Stream summaries — such as samples or sketches — typically focus on estimating a single aggregate (such as COUNT DISTINCT, MEDIAN, etc.). Here, our problems are more complicated aggregates, requiring second or third order functions (our F_2 problem on multigraphs can be thought of as sum of squares of count distinct, i.e. an aggregate of aggregates). To solve these we create approximation methods that combine multiple sketch or sample methods, to give "sketch within sketch" or similar ideas. Here, the challenge is both to design the appropriate cascading summary structure and to prove strong guarantees about its quality. Within the context of "cascaded data stream summaries", many different combinations of data stream summaries can be cascaded. Careful choice of summaries gives us our provable results above; many of the other rather natural choice turn out to be provably inefficient or inaccurate in the worst case and have various shortcomings that we point out; still, sometimes they yield reasonable heuristic methods to compare against our guaranteed methods.

2. We observe that multigraph problems such as above can be thought of more generally as what we call *cascaded aggregates*, that is, composition of some aggregate P on a stream that is derived from the input stream by application of aggregate Q. We present many results classifying the complexity of cascaded aggregate problems on data streams for various P's and Q's.

3. By giving methods that are invariant under repetitions of the same input, they are *idempotent*, and can be merged to summarize the union of multiple multigraph streams. This makes them immediately applicable to a wider variety of settings, such as distributed measurement, and diffusing synopses within a sensor network [29]. Summaries can be "flooded" and merged throughout a network to avoid loss of information, without any danger of over-counting.

4. Lastly, we perform a detailed experimental study with real communication streams and compare our provable methods with other heuristics; we also explore their performance over synthetic data such as Zipfian distributions that are found in communication networks. We observe that our methods are highly efficient, giving very accurate answers while using limited space, between 100 to 200 KB.

Map. Section 2 gives background for the data stream primitives that we make use of. Our algorithms for computing M_2, the second frequency moment of the degrees, are in Section 3; in Section 4 we describe how to compute individial degrees; and degree range sums in Section 5. In Section 6 we present our experimental results Extensions are present in Section 7, with concluding remarks in Section 8.

2. PRELIMINARIES

Throughout, we make use of algorithms which approximate the number of distinct items observed. This is a key problem in data streams, and several methods have been proposed [16, 20, 19, 5, 9]. We omit detailed discussion of these methods for space reasons, and instead summarize the properties that they guarantee. Throughout, we will write $x = (1 \pm \varepsilon)y$ as shorthand for $(1 - \varepsilon)y \leq x \leq (1 + \varepsilon)y$, and we assume $\varepsilon < 1$.

FACT 1 (DUE TO [16, 5]). *There exist* approximate distinct counter *algorithms that take parameters ε and δ and create a data structure, D, of size $O(\frac{1}{\varepsilon^2}\log 1/\delta)$ machine words. The time to process each update is $O(\log 1/\delta)$. They observe a stream of (integer) values, and report a value $|D|$ such that, if S is the set of values observed thus far, then $|D| = (1 \pm \varepsilon)|S|$ with probability at least $1 - \delta$.*

Throughout, we will set $\delta = \text{poly}(1/n)$, so taking the union bound of all probabilities means that all inequalities hold simultaneously with high probability. These data structures can have additional properties: give two data structures that summarize two sets, say S and T, then one can merge the data structures to get a summary of $S \cup T$. Also, in many implementations, repetitions of the same item in the stream do not alter the data structure. In our discussion, we treat the approximate distinct counters as sets, and so use set notation to denote operations upon them: $D = D \cup \{i\}$ means we add item i to the distinct counter D, $|D|$ means we take the approximate count of distinct items, etc. Other implementations of approximate distinct counters also allow the effect of the insertion of an item to be undone [9], ie $D = D \backslash \{i\}$.

To build our algorithms, we also make use of hash functions with special properties, *min-wise hash functions*:

FACT 2. *Let $h : \{1 \dots n^2\} \to \{1 \dots n^2\}$ be drawn from a family of approximately min-wise independent hash functions [22]. For given ε, ε-approximate min-wise independent hash functions h can be computed in time, and represented in space, $O(\log(1/\varepsilon))$; for any i, they guarantee $\mathsf{Pr}[h(i) = \min_{1 \le j \le n^2} h(j)] = (1 \pm \varepsilon)\frac{1}{n^2}$.*

3. M_K ESTIMATION METHODS

We first show a simple lower bound in Section 3.1 for sampling methods to illuminate the nature of our main result, then present our M_k estimation algorithm based on sampling in Section 3.2. As mentioned earlier, our algorithm will rely on cascading one summary (the distinct counters from above) into another (min-wise hash sampling) and we show it provides strong approximation guarantees. However, other combinations of summaries within each other suggest themselves. These turn out to have poor worst case guarantees, but still some are useful as heuristics, as discussed in Section 3.3.

3.1 Lower Bounds for Sampling Techniques

We show that a simple lower bound for the most simple sampling technique.

THEOREM 1. *Any algorithm based on the uniform random sampling technique to estimate M_2 (in fact, even F_2) requires $\Omega(\sqrt{n})$ bits of storage.*

PROOF. We create a multigraph stream of edges on n nodes. We randomly divide the nodes into two sets, X and Y each of size $n/2$. We choose a random mapping from X to Y, and create a stream based on this mapping as $n/2$ edges (x, y). We now pick from two possibilities: either create a new random mapping and insert a further $n/2$ edges, or randomly permute the same mapping and insert the same first $n/2$ edges again. Observe that in the first case, $M_2 = 4n$ whereas in the second $M_2 = n$. Clearly, in the second case, sampling will find no node with degree two. However, in the first case, if the sample size is $o(n^{1/2})$ then with high probability, random sampling of the edge stream will not include both (x, y) and (x, y') in the sample (from the birthday paradox). Consequently, the two cases will be indistinguishable, and so such sampling cannot approximate M_2 to a factor better than 2. □

Having shown a $\Omega(\sqrt{n})$ lower bound for uniform sampling, we now give an algorithm based on a more sophisticated sampling method that approximates M_2 in space proportional to $\sqrt{n} \log n$.

3.2 Algorithm based on Min-wise Hashing

We approximate the number of distinct edges that are seen using an approximate data structure D. The output $|D|$ gives an approximation, $\hat{m}$ of $M_1 = \sum_{i=1}^{n} d_i = m$. We will define an estimator, X, whose expectation is the quantity that we desire and whose variance is bounded in terms of the square of its expectation. By keeping $O(KL)$ copies of this estimator independently and in parallel (for suitably chosen values of K and L), we can combine these to get an estimate with guaranteed accuracy bounds. The estimator is based on the technique in [2] for F_2 based on sampling; here, the adaptation to M_2 requires the combination of min-wise hashing and distinct items counters, and a new proof. The (k, l)th estimator is created as follows:

Min-wise hash estimator. Initialize $min_{k,l} = n^2 + 1$, and $v = 0$. For each edge (i, j) in the stream, compute $h_{k,l}(ni + j)$, where $h_{k,l}$ is an approximate-min-wise hashing function in Section 2. If $h_{k,l}(ni + j) < min_{k,l}$ then set $min_{k,l} = h_{k,l}(ni + j)$, and set $v_{k,l} = i$. We keep the set of (distinct) edges $(v_{k,l}, j')$ that are seen in the stream from this point onward as $E_{k,l}$. If the size of this set $|E_{k,l}| \ge \frac{1}{\varepsilon^2}$, then we initialize an instance of an approximate distinct elements counter, $D_{k,l}$. Then, for every subsequent edge $(v_{k,l}, j')$, we insert this into $D_{k,l}$, if it is not already stored in $E_{k,l}$. When we reset v, we also reset $E_{k,l}$ and $D_{k,l}$ to $\emptyset$. When needed, we output an estimate. If $D_{k,l}$ is not being used, then set $\hat{d} = |E_{k,l}|$. Else, denote by $|D_{k,l}|$ the output of the estimator, and let $\hat{d}$ be $|E_{k,l}| + |D_{k,l}|$. The estimate is $\hat{m}(2\hat{d} - 1)$. □

Intuitively, we use the min-wise hash function to approximately sample nearly uniformly from the set of all distinct edges that has been seen at any time. For the sampled edge, we maintain the distinct count estimate of its degree. This is an instance of our "cascaded summaries": first we sample, then for each member of the sample we keep an approximate distinct elements counter. We compute the appropriate estimator based on this sample degree and are able to prove the following.

THEOREM 2. *Each estimator $X_{k,l} = \hat{m}(2\hat{d} - 1)$ has expectation $(1 + O(\varepsilon))M_2$ and variance $(1 + O(\varepsilon))n^{1/2}M_2^2$.*

PROOF. Let v be the value of the variable $v_{k,l}$ at the end of the stream. Using the property of h, $\mathsf{Pr}[v = i]$
$= \sum_{G[i,j]=1} \mathsf{Pr}[h(ni + j) < \min_{i' \ne i, G[i',j']=1} h(ni' + j')]$
$= \frac{d_i}{m}(1 \pm \varepsilon)$.

The expectation of this estimator is the probability that any given edge (i, j) is chosen by the min-wise hash function $(1/m)$, times the contribution of that edge to the total, which is related to a, the number of unique edges (i, k) that occur after (i, j) in the stream.

$$\begin{aligned}
\text{Thus}\quad \mathsf{E}(X_{k,l}) &= \tfrac{1\pm\varepsilon}{m} \textstyle\sum_{i=1}^{n} \sum_{a=1}^{d_i} \hat{m}(2\hat{d} - 1) \\
&= (1 \pm \varepsilon)^2 \tfrac{\hat{m}}{m} \textstyle\sum_{i=1}^{n} \sum_{a=1}^{d_i} 2\hat{d} - 1 \\
&= (1 \pm \varepsilon)^2 \tfrac{\hat{m}}{m} \textstyle\sum_{i=1}^{n} \sum_{a=1}^{d_i} (1 \pm 2\varepsilon)(2a - 1) \\
&= (1 \pm \varepsilon)^2 (1 \pm 2\varepsilon) \textstyle\sum_{i=1}^{n} (d_i^2 + d_i - d_i) \\
&= (1 \pm \varepsilon)^4 M_2
\end{aligned}$$

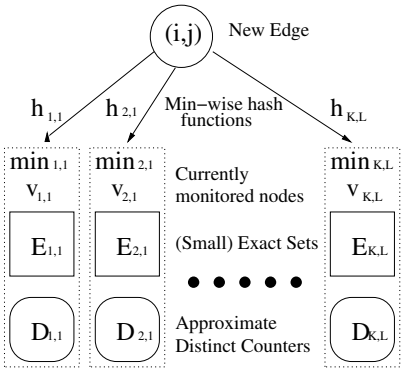
The main step relies on the fact that we have computed $\hat{d} = a$ exactly if $a \le \frac{1}{\varepsilon^2}$. Otherwise, if $\hat{d}$ is approximated, then $1 < \varepsilon^2 a$ and so $2a - 1 = (1 \pm \varepsilon^2)2a$. Since $\hat{d} = (1 \pm \varepsilon)a$, then $2\hat{d} - 1 = (1 \pm 2\varepsilon)(2a - 1)$.

Similarly, for the variance,

$$\begin{aligned}
\mathsf{Var}[X_{k,l}] \le\ & \mathsf{E}(X_{k,l}^2) = \tfrac{1\pm\varepsilon}{m} \textstyle\sum_{i=1}^{n} \sum_{a=1}^{d_i} (\hat{m}(2\hat{d} - 1))^2 \\
\le\ & (1 + \varepsilon)^3 m \textstyle\sum_{i=1}^{n} \sum_{a=1}^{d_i} (4\hat{d}^2 - 4\hat{d} + 1) \\
\le\ & (1 + \varepsilon)^3 m \textstyle\sum_{i=1}^{n} \sum_{a=1}^{d_i} (1 + 2\varepsilon)^2 (4a^2 - 4a + 1) \\
\le\ & (1 + \varepsilon)^3 (1 + 2\varepsilon)^2 m \textstyle\sum_{i=1}^{n} \tfrac{4}{3} d_i^3 \\
\le\ & (1 + \varepsilon)^7 \tfrac{4}{3} M_1 M_3 \\
\le\ & (1 + \varepsilon)^7 \tfrac{4}{3} n^{1/2} M_2^2 \le O(n^{1/2}) \mathsf{E}(X_{k,l})^2
\end{aligned}$$

This follows from the bounds on ε; the accuracy of the estimation $\hat{d}$; and that $M_1 M_{2k-1} \le n^{1-1/k} M_k^2$ [2]. □

The probability that the mean of $K = O(\frac{n^{1/2}}{\varepsilon^2})$ estimates diverges from the correct answer by more than a $1 \pm \varepsilon$ factor is bounded by constant probability using the Chebyshev inequality. Taking the median of $L = O(\log n)$ such averages amplifies this to arbitrary small probability using standard Chernoff bounds arguments. The full algorithm is



M2UpdateMinHash(i, j)

```
1: D = D ∪ (i, j);
2: for l = 1 to L do
3:   for k = 1 to K do
4:     if (h_{k,l}(n * i + j) < min_{k,l}) then
5:       min_{k,l} = h_k(n * i + j);
6:       v_{k,l} = i; D_{k,l} = ∅; E_{k,l} = ∅;
7:     if (i = v_{k,l}) then
8:       if (|E_{k,l}| < 1/ε^2) then
9:         E_{k,l} = E_{k,l} ∪ (i, j);
10:      else if ((i, j) ∉ E_{k,l}) then
11:        D_{k,l} = D_{k,l} ∪ (i, j);
```

M2OutputMinHash

```
1: m = |D|;
2: for l = 1 to L do
3:   s_l = 0;
4:   for k = 1 to K do
5:     if (|E_{k,l}| ≤ 1/ε^2) then
6:       d = |E_{k,l}|;
7:     else
8:       d = |E_{k,l}| + |D_{k,l}|;
9:     s_l = s_l + m * (2 * d − 1);
10:  s_l = s_l/K;
11: return(median(s_l));
```

Figure 1: Min-wise hash based data structure and algorithms for computing M_2 of multigraph streams.

illustrated and given in pseudo-code in Figure 1. The routine M2UpdateMinHash takes a new edge (i, j) from the multigraph stream and updates the appropriate data structures; M2OutputMinHash uses these data structures to give the estimate of M_2 of the stream.

The space required is $O(\frac{1}{\varepsilon^2} n^{1/2})$ estimators, each of which requires space to store the min-wise hash function, the exact list of edges E and the approximate distinct elements counter D, giving an overall space bound of $O(\frac{1}{\varepsilon^4} n^{1/2} \log n)$. We now summarize these results with the following theorem:

Theorem 3. *There exists a randomized algorithm that estimates M_2 over multigraph streams to $(1 \pm \varepsilon)$ accuracy with high probability using $O(\frac{1}{\varepsilon^4} \sqrt{n} \log n)$ space and time $O(\frac{1}{\varepsilon^2} \sqrt{n} \log n)$ per edge.*

This approach naturally extends to estimating M_k for $k > 2$ with $O(\frac{k^2}{\varepsilon^4} n^{1-1/k} \log n)$ space; we omit the details here.

3.3 Heuristic methods for M_2

Many other algorithms suggest themselves for this problem, however several of the obvious solutions exhibit drawbacks or limitations. One might try sampling uniformly over the domain of items, rather than from those seen in the stream. An estimator could then be generated by using a distinct items algorithm, and squaring the result. While this estimator has the correct expectation, the variance is much higher, meaning that $\Omega(n)$ samples must be taken to give a guaranteed approximation of the desired quantity. Similarly, sampling directly from the stream without using the min-wise hash functions that we do here means that repetitions of an edge in the stream can skew the sampling, and fail to result in an unbiased estimator. Hence, although the above hashing procedure seems involved, it appears necessary in order to give an approximation with guaranteed bounds. We consider one seemingly promising approach that suggests itself as a heuristic to compare against, and point out why it cannot give strong guarantees for M_2.

In a seminal paper, Alon, Matias and Szegedy gave a method for computing the second frequency moment of a stream of values, also known as F_2 [2]. They showed that in small space and with bounded randomness, high quality estimators for F_2 could be constructed with a single pass over the stream of values. The space is essentially independent of the size of the stream, requiring only $O(\frac{1}{\varepsilon^2} \log \frac{1}{\delta})$ in order to give an approximation of the second frequency moment up to a $1 \pm \varepsilon$ factor with probability at least $1 - \delta$.

A natural idea is to try to take this algorithm and generalize it to work on multigraph streams. Where their sketch algorithm kept sums of counts of observed items, one could imagine replacing these with approximate distinct counters, and applying the estimation technique on these. That is, create a cascaded summary based on approximate distinct counters cascaded within AMS sketches. In fact, such an approach contains a subtle flaw, meaning that one cannot give the desired tight bounds on the result. We will describe the details of such a method, which can be used as a heuristic method for this problem.

Sketch-based M_2 estimation. The key idea is to take the "tug-of-war" sketch of [2, 1], and replace the counts of items with approximate distinct counters. In order to do this, we also need to "open up" the structure, which mapped values onto $+1$ or -1, and then added these values to counters. We take $2b$ approximate distinct counters. Given each update (i, j), we use a 2-wise independent hash function, σ, to map i onto $\{0, 1, \ldots b-1\}$. Then we use a 4-wise independent hash function π to map i onto -1 or 1, and compute $\tau(i) = 2\sigma(i) + (\pi(i) + 1)/2$. We then place (i, j) into the $\tau(i)$th estimator for F_0, denoted $D_{\tau(i)}$.

By analogy with [2], we can compute an estimate for M_2 as $X = \sum_{k=0}^{b-1} (|D_{2k}| - |D_{2k+1}|)^2$. However, this is not an *unbiased* estimator: each $|D_i|$ estimator gives a $1 \pm \varepsilon$ estimate of the number of items placed into it, but because we must *subtract* two such estimates, the result is no longer a $1 \pm \varepsilon$ estimator. Although we can use this method as a heuristic, in our experiments we see bad cases where this bias causes the estimate to be wildly wrong.

4. APPROXIMATING DEGREES AND DEGREE HEAVY HITTERS

The next problem we consider is to find nodes with large degrees, that is, $d_i > \phi m$, for a parameter $\phi < 1$. In massive graph scenarios such as web graphs, often few nodes have large degrees and finding nodes with overwhelming degrees is of interest. Existing methods for finding items with high frequencies in a data stream are suited for estimating edges or nodes that occur frequently but do not find nodes with large multigraph degrees. For example, consider an approach based on sampling from the stream of edges. Then it is easy to see that this sampling will be biased if the same edge appears multiple times in the stream. Equally, sampling from the domain (that is, choosing in advance a set

PointUpdateMinHash(i, j, v)
```
1: for k = 1 to K do
2:   if (h_k(n * i + j) < min_k) then
3:     min_k = h_k(n * i + j); i_k = i;
```

PointQueryMinHash(i)
```
1: for k = 1 to K do
2:   if (i_k = i) then d = d + 1;
3: return(d * |D|/K);
```

PointUpdateSketch(i, j, v)
```
1: for k = 1 to L do
2:   D[k, f_k(i)] = D[k, f_k(i)] ∪ v;
```

PointQuerySketch(i)
```
1: d = ∞;
2: for k = 1 to L do
3:   d = min(d, |D[k, f_k(i)]|);
4: return d;
```

ProcessEdge(i, j)
```
1: D = D ∪ (i, j);
2: PointUpdate(i, j, j);
3: if (PointQuery(i) ≥ φ|D|) then
4:   V = V ∪ i;
5:   if |V| = 2/φ then
6:     for all v ∈ V do
7:       if (PointQuery(v) < φ|D|) then
8:         V = V\v;
```

Figure 2: Algorithms for finding nodes of high degree in multigraph streams

of edges to sample, and then recording whether they were seen or not) gives a method that is correct in expectation, but has very high variance, meaning that the size of the sample required to give an accurate answer has to be linear in the size of the stream, m. In what follows we give algorithms for the problem of estimating d_i's (denoted by PointQuery(i)), and finding heavy hitter i's, ie., i's with large d_i's.

Given a subroutine implementing PointQuery, one can then build an algorithm to track only those d_is that exceed the threshold ϕn as further updates are observed, in the style of [8]. In order to maintain the largest degrees, we keep the current set of heavy degrees in a heap. After each update (i, j), we compute PointQuery(i); if it is above $\phi\hat{m}$, then retain i in a set V of nodes. We need to check that the this set of nodes does not grow too large, so if $|V| > 2/\phi$, we remove any $i \in V$ for which $D[f(i)] < \phi\hat{m}$. When needed, output every node $i \in V$ for which PointQuery(i) the estimated count of i is at least $\phi\hat{m}$. Following [8], this procedure guarantees to find the heavy hitters, provided the point query routines give accurate estimates. This algorithm to maintain the heavy hitter nodes is illustrated in Figure 2 as ProcessEdge.

Hence, our discussion now focuses on how to estimate the d_is in order to implement PointQuery.

4.1 Sampling using Min-wise Hashing

We first give a sophisticated sampling procedure to draw a sample uniformly from the edges of the graph, rather than the edges of the *multigraph* stream. From this sample, one can then use the degrees of the sampled subgraph to approximate the frequency of nodes in the original graph. The key idea is to use the min-wise sampling technique as used above in the M_2 estimation algorithm to draw from the stream.

Theorem 4. PointQuery *can be implemented approximately with error εm with high probability in space and time per update and per query $O(\frac{1}{\varepsilon^2}\log(1/\varepsilon)\log n)$.*

Proof. We use K min-wise hash functions $h : \{1 \dots n^2\} \to \{1 \dots n^2\}$. For each edge in the stream (i, j) we compute $h(n * i + j)$, and for each hash function retain the edge that maps to the smallest value. After processing the whole stream, the probability of each edge being retained by each hash function is $\frac{1\pm\varepsilon}{m}$. Let S denote the set of edges retained in the sample, clearly $|S| = K$. Given a particular node i, the approximate degree is computed by $|\{(i, j)|(i, j) \in S\}| * \frac{\hat{m}}{K}$. Here, $\hat{m}$ is our $1 \pm \varepsilon$ estimate of the number of distinct edges m. This estimator is correct in expectation. By application of the Hoeffding inequality [27], one can show that the estimator is correct within $(1 \pm \varepsilon)m$ with probability δ, provided we set $K = O(1/\varepsilon^2 \log 1/\delta)$. To maintain the sample, we need to compute the K min-wise hash functions on each edge, and maintain the K smallest edges. Since each hash function requires space $O(\log \frac{1}{\varepsilon})$, the total space required is $O(\frac{1}{\varepsilon^2}\log(1/\varepsilon)\log n)$ to guarantee the approximation quality with high probability. The algorithm is illustrated in Figure 2 as routines PointUpdateMinHash and PointQueryMinHash. □

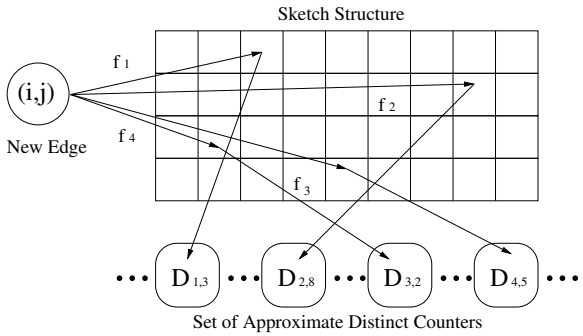


Figure 3: Sketch Structure for Degree Estimation

Corollary 1. *Using the above routine, we can find heavy hitters in multigraph streams in space $O(\frac{1}{\varepsilon^2}\log(1/\varepsilon)\log n)$ to guarantee all nodes with $d_i > (\phi + \varepsilon)m$ are output, and no nodes with $d_i < (\phi - 2\varepsilon)m$ are output, with high probability.*

4.2 Sketch Based Algorithm

A disadvantage of the above sampling based method is that it requires a lot of processing time per update. We design an alternate approach, which is another example of a *cascaded summary*, or a "sketch within a sketch": combining the sketch structure of [10] with distinct element algorithms. The important feature is that this sketch structure relies on additions only and therefore, it can give bounded guarantees on the accuracy of estimates (unlike the sketch-based algorithm for M_2).

Sketch Algorithm for Degree Estimation. Keep an array of $L \times B$ distinct element counters, labeled $D[1, 1] \dots D[L, B]$. Let $f_k : \{1 \dots n\} \to \{1 \dots B\}$ be a hash function drawn from a pairwise independent family. For each edge (i, j), insert j into $D[k, f_k(i)]$. We will show that taking $\min_k |D[k, f_k(i)]|$ is an accurate estimate for d_i. The algorithm and data structures are illustrated graphically in Figure 3 and in pseudocode in Figure 2 as routines PointUpdateSketch and PointQuerySketch.

DRSUPDATESKETCH($i, j, sumdi$)

1: $\alpha = i$;
2: for $\beta = 0$ to $\log n$ do
3: POINTUPDATESKETCH$_\beta$($\beta * n + \alpha, j, sumdi * (i * n) + j$)
4: $\alpha = \lfloor \alpha/2 \rfloor$;

Figure 4: Processing an edge to compute distinct range sums

THEOREM 5. *We can implement* POINTQUERY *approximately with error εm with high probability in space $O(\frac{1}{\varepsilon^3} \log^2 n)$. The time per update and per query is $O(\log^2 n)$.*

PROOF. We show that the estimate $\min_k |D[k, f_k(i)]|$ is a good estimator of d_i. We set $B = 2/\varepsilon$.

$$\begin{aligned} \mathsf{E}(|D[k, f(i)|]) &= (1 \pm \varepsilon)(\textstyle\sum_{j, f(j)=f(i)} d_j) \\ &= (1 \pm \varepsilon) d_i + (1 \pm \varepsilon) \textstyle\sum_{j \neq i, f(j)=f(i)} d_j \\ &= d_i \pm \varepsilon d_i + (1 \pm \varepsilon) \frac{\varepsilon(m - d_i)}{2}. \end{aligned}$$

Hence, $\mathsf{Pr}[|D[k, f(i)]| - (1 \pm \varepsilon) d_i > \varepsilon(1 \pm \varepsilon)(m - d_i)] < \frac{1}{2}$. So with constant probability, $|D[k, f(i)]| = d_i + (1 \pm \varepsilon)(\varepsilon m)$. By taking the estimate of d_i to be the minimum of $L = O(\log n)$ independent repetitions of the estimator, we can obtain guarantees to high probability. The space required is $O(\frac{1}{\varepsilon^3} \log^2 n)$, and the time per edge is $O(\log^2 n)$. □

Using this routine within the heap-based solution to tracking the heavy hitter degrees gives the following guarantees about the output. If $d_i > (\phi + \varepsilon)m$ then $|D[k, f(i)]| > (\phi + \varepsilon)m > \phi(1 + \varepsilon)m > \phi\hat{m}$ with constant probability, and so it will be retained (assuming $\varepsilon \leq \phi \leq \frac{1}{2}$). Symmetrically, if $|D[k, f(i)]| > \phi\hat{m} = \phi(1 \pm \varepsilon)m > \phi(1 - \varepsilon)m$ then $d_i + (1 + \varepsilon)(\varepsilon m) > (1 - \varepsilon)\phi m \Rightarrow d_i > m(\phi - \varepsilon(1 + \phi + \varepsilon)) > m(\phi - 2\varepsilon)$. In summary,

COROLLARY 2. *Using the above routine, we can find heavy hitters in multigraph streams in space $O(\frac{1}{\varepsilon^3} \log n)$ to guarantee all nodes with $d_i > (\phi + \varepsilon)m$ are output, and no nodes with $d_i < (\phi - 2\varepsilon)m$ are output, with high probability.*

4.3 Lossy Counting Heuristic

There are several methods which find frequently occurring items in a data stream and report ε-accurate counts for them. A natural approach to the problem of finding large multigraph degrees is to take some of these methods and try to modify them to avoid multiply counting the same edge. For example, one could take the Lossy Counting method of Manku and Motwani [26], which stores counts of certain observed items, and replace the exact counters with approximate distinct counters. Intuitively, this seems a reasonable modification to make, but it turns out that in analyzing the new algorithm one can no longer make tight bounds on its accuracy. Where the original algorithm assumes accurate item counts, we have only approximations, and so the decision about whether to retain an item in memory or to prune it is no longer clear cut. In the worst case, one can argue that the space required to track item counts could grow without bound. However, this algorithm is a reasonable heuristic to try out in practice and to see how it performs on realistic data.

5. DISTINCT RANGE SUMS

We can make use of the above methods in order to compute other functions, over multigraphs or in other streaming settings. In many cases the nodes of a streaming multigraph are comparable under a total order, and consecutive nodes under the ordering have a natural interpretation. For example, in IP networks, nodes can be indexed by IP address, and consecutive nodes make up subnetworks. It is natural then to ask range queries over ranges of node values, say $[j, k]$. There are two equally reasonable ways to interpret such queries given the range $[j, k]$:

- Sum d_i: $S_{j,k} = \sum_{i=j}^{k} d_i = |\{(i, y) | (i, y) \in E \wedge j \leq i \leq k\}|$.
- Union d_i: $U_{j,k} = |\{y | (i, y) \in E \wedge j \leq i \leq k\}|$.

The first query computes the sum of the degrees of nodes in the range query; the second treats the nodes in the range as a single node, and computes its degree. Using appropriate calls to the POINTQUERYSKETCH routine from the previous section, both types of distinct range query can be answered in the same bounds.

The problem can be reduced to querying a logarithmic number of non-overlapping *dyadic ranges* (see, for example, the discussion in [10]). These are ranges of the form $DR(\alpha, \beta) = [\alpha 2^\beta, (\alpha + 1)2^\beta - 1]$ for integer α and β. We can then treat each dyadic range as a single point, so by tracking information about all $\log n$ dyadic range lengths in $\log n$ separate sketches, and updating the $\log n$ ranges affected by each new edge, we can answer range queries.

In order to give guaranteed bounds on the accuracy of our results, we focus on the sketch-based methods, since in order to give answers with high accuracy to a range query, we must merge the results of the dyadic range queries to give an answer that is accurate over the whole range. Our actions are different depending on whether we want to answer Sum d_i or Union d_i queries. For Sum d_i queries, we must ensure that all edges are treated as distinct, and so we will insert a value $(n * i + j)$ into our data structure to encode this. For Union d_i queries, we do not want to count the same destination j multiple times, and so we insert the value j instead. The full update algorithm is illustrated in Figure 4. It makes use of an indicator variable, *sumdi*, which is 1 if the goal is Sum d_i queries, and 0 for Union d_i. It calls the function POINTUPDATE from Figure 2. Note that for Union d_i queries in particular we faced a non-trivial challenge, since we cannot simply sum the results of the dyadic range queries: this will over-count nodes that are linked to by multiple dyadic ranges. Instead, we must make crucial use of the properties of the distinct range counters and so require a careful analysis.

In order to compute the approximate Union d_i, we compute the *union* of the estimators for the range $[j, k]$. Just as in the point query case, we go to the estimators for the query, but this time we have at most $2 \log n$ places to probe (at most two per value of β). We take all the corresponding estimators at probe points, and merge them together (as mentioned in Section 2, merging approximate distinct counters is supported by many implementations). The result is a single distinct element counter that contains the value of interest, plus some error due to other items which were mapped to the same counters. The expected error from collisions under hash functions can again be bounded in terms of m, n, and the number of buckets, b. We adjust the parameters to get the necessary bounds: since we are taking up to $2 \log n$ individual estimators, the error can potentially increase by this factor. We counteract this by increasing the number of buckets by this factor also. However, we do not

need to adjust the approximation factor of the distinct counters, since they give a relative error guarantee. Formally, we show the bounds as follows:

THEOREM 6. *With space $O(\frac{1}{\varepsilon^3}\log^3 n)$, and update time $O(\log^3 n)$, we can answer Distinct Range Union d_i queries with error at most εm with high probability.*

PROOF. Let E denote the set of edges and let the queried range be represented by $2\log n$ (possibly empty) dyadic ranges, $DR[\alpha_{\beta,l}, \beta]$ and $DR[\alpha_{\beta,r}, \beta]$ for $\beta = 0 \ldots \log n/2$. We will write $\gg$ for the right shift operator where $x \gg y = \lfloor \frac{x}{2^y} \rfloor$. Our estimate is given by:

$$
\begin{aligned}
&(1 \pm \varepsilon)|\cup_{\beta=0}^{\log n/2} \{y|(i,y) \in E, \quad f(i \gg \beta) = f_\beta(\alpha_{\beta,l}) \\
&\qquad\qquad\qquad\qquad \vee \quad f(i \gg \beta) = f_\beta(\alpha_{\beta,r})\}| \\
&= (1 \pm \varepsilon)|\cup_{\beta=0}^{\log n/2} \{y|(i,y) \in E, j \le i \le k\}| \\
&\quad \cup\{y|(i,y) \in E, \quad f_\beta(i \gg \beta) = f_\beta(\alpha_{\beta,l}) \\
&\qquad\qquad \vee \quad f_\beta(i \gg \beta) = f_\beta(\alpha_{\beta,r})\}| \\
&= (1 \pm \varepsilon)U_{j,k} \\
&+(1 \pm \varepsilon)\textstyle\sum_{\beta=0}^{\log n/2} \quad |\{y|(i,y) \in E, f_\beta(i \gg \beta) = f_\beta(\alpha_{\beta,l})\}| \\
&\qquad\qquad + \quad |\{y|(i,y) \in E, f_\beta(i \gg \beta) = f_\beta(\alpha_{\beta,r})\}| \\
&= (1 \pm \varepsilon)U_{j,k} + (1 \pm \varepsilon)\textstyle\sum_{\beta=0}^{\log n/2} 2m/b \\
&= (1 \pm \varepsilon)U_{j,k} + (1 \pm \varepsilon)\frac{4m \log n}{b}
\end{aligned}
$$

By setting the number of distinct element counters for each value of β to be $b = \frac{10 \log n}{\varepsilon}$ and applying the Markov inequality, the total error is more than εm with probability at most $\frac{2}{5}(1+\varepsilon)$. Because $\varepsilon < 1$, this is bounded by a constant.

Since there are $\log n$ values of β to consider and each requires $b \log n = O(\log n/\varepsilon)$ distinct counters each of size $O(\frac{1}{\varepsilon^2}\log n)$, the total space required is $O((\frac{\log n}{\varepsilon})^3)$. The time for each update is $O(\log^2 n)$ for each sketch structure and the time to query the data structure is the same as the update time, $O(\log^3 n)$. □

The analysis for Distinct Range Sum d_i queries is similar:

THEOREM 7. *With space $O(\frac{1}{\varepsilon^3}\log^3 n)$, and update time $O(\log^3 n)$, we can answer Distinct Range Sum d_i queries with error at most $2\varepsilon m$ with high probability.*

Heuristic approaches for Distinct Range Sums. One can easily generate heuristic methods to compare against, by again using dyadic ranges, but using a method such as the LossyCounting heuristic to compute the values for each range, and summing these. This may perform badly for Union d_i queries, since the same destination may be counted multiple times when it should only be counted once. For similar reasons, note that the solution of using the Minwise Hash based sampling to draw a uniform sample of the edges and then estimating the result based on the fraction of the sample that is within the queried range does not give an estimator that is correct in expectation; so this gives an alternate heuristic approach.

Applications to quantile tracking. A direct application of the Distinct Range Sum queries is to answering quantile queries over the degrees of nodes. For example, one problem is to find the median node, which is the one for which half the edges are from nodes less than it, and half from nodes greater than it (under the ordering of the node identifiers). More generally, one can specify an arbitrary quantile value ϕ and ask to find the node with ϕm edges below it. This is solved by observing that this can be rephrased as a Distinct Range Sum Query (as in [10]) to find the node k such that the range sum $[0,k] = \phi m$. The Union and Sum variants give different semantics: Sum corresponds to the standard notion of equi-depth histograms, whereas Union can be thought of as generating the boundaries for equi-width histograms. Our result then follows from the above theorems on distinct range sums:

COROLLARY 3. *With space $O(\frac{1}{\varepsilon^3}\log^3 n)$ we can answer quantile queries with error at most εm with high probability.*

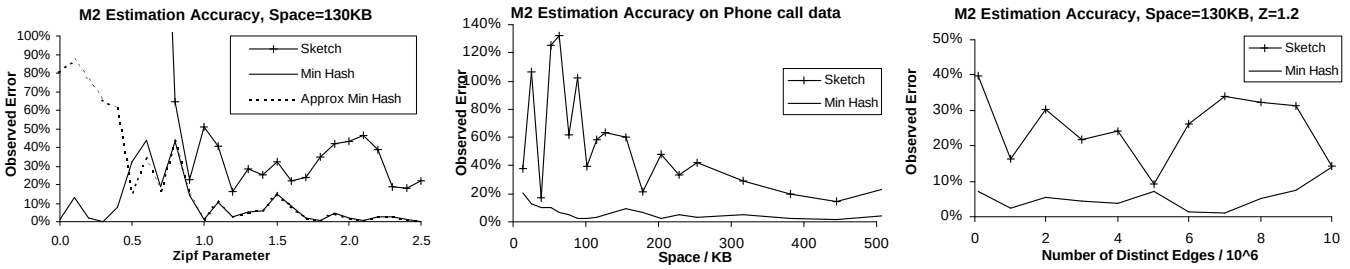
This problem has application to a different setting: in sensor networks, we wish to compute quantiles over sets of values observed at different sites. If we model the sites by d_p and the values by s_p, then we can capture this in terms of our multigraph model. A standard approach in sensor networks to aggregate computation is *synopsis diffusion* [29], where synopses of individual sensors are sent around the network and merged with others to build an overall summary. For simple aggregates such as the minimum value, this approach is simple and resilient to the order and number of merges, since the combining process is *idempotent*: we can repeatedly merge two values by taking their minimum, and this is resilient to repetitions of the same summary being merged in at different points. Our summaries are also idempotent, so they can be used to solve the quantile computation problem.

6. EXPERIMENTS

We implemented the key methods for M_2 estimation and approximating degrees of nodes in multigraphs. Both of these methods are vital for subsequent mining tasks, such as finding nodes of high degree, and computing distinct range sums. The implementation used C, and experiments were run on a 2.4GHz desktop machine.

We experimented on a variety of real and synthetic datasets representing massive multigraphs by a stream of edges. The real data sets were drawn from networking scenarios, and consisted of a set of 2.2 million phone calls and 600,000 Internet connections for one day from the Internet Traffic Archive [25, 30]. In each case, we represented the dataset as a series of (source, destination) pairs: (calling local exchange, dialed local exchange) in the telephone case, and (source IP, destination IP) in the network case. We analyzed the induced graphs offline, and saw that there were over 1 million distinct edges in the phone data, and 35,000 in the network case. In both cases, the data displayed noticeable skew, with some nodes have degrees of thousands or tens of thousands, and many more having degree close to one.

To generate the synthetic data, we adopted a generalized Zipfian distribution, where the ith largest degree is proportional to $1/i^z$, where z is the Zipfian parameter. Hence, to generate the data we repeatedly performed the following procedure: create the jth (directed) edge as (i,j) where i is selected from a Zipfian distribution with parameter z. We then applied a hash function on the source and destination nodes of each edge to mask the structure. Thus, the degree sequence of the nodes follows a Zipfian distribution. Such skewness of degree sequences has been observed in many network settings, including communication and social networks [23]. Hence, this allows us to generate an unbounded sized multigraph that resembles realistic



(a) (b) (c)

Figure 5: Experiments on M_2 Estimation

communication networks. Note that this method does not generate multiple copies of the same edge. However, all of our methods are invariant under repetitions of an edge: if an edge is added which has been observed before, then in every case the data structure does not change. So for testing the accuracy of our methods we only have to use graph streams with no repeats.

6.1 M_2 Estimation

We carried out a set of experiments on computing M_2 on multigraph streams. We computed the exact result so we could empirically evaluate the quality of approximations of the two techniques, the guaranteed bounds of the min-wise hashing approach and the heuristic of using an AMS sketch with distinct item counters. In our experiments, we always allocated the same amount of space to the sketch approach and the min-wise hashing approach. Throughout, we used the Flajolet-Martin data structure ("FM sketches") to implement the approximate distinct counter [16]. We set the number of FM estimators in each FM sketch to be 20 and L, the number of repetitions of our estimators, to be 5. Increasing these values did not improve the quality of the output significantly. We then experimented with varying the remaining parameters: for synthetic data, the Zipf skewness parameter, z, and for real data the amount of space dedicated to the approximation task.

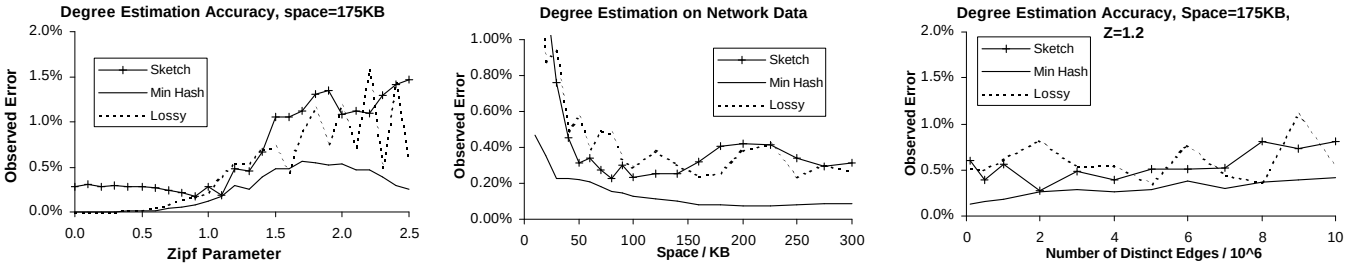
Our first results are shown in Figure 5. We plotted the observed error on one million edges drawn from a Zipfian distribution in Figure 5 (a). We show here the importance of keeping track of the number of distinct edges incident on a node exactly when it is small: we have also plotted the accuracy observed if we always used approximate distinct counters as "approx min hash". We see that when the distribution is more uniform ($z < 1$), keeping exact counts when possible brings significant gains, when many counts are small. Above $z = 1$, there is no difference, since the largest degrees dominate the computation of M_2, and these must be approximated. Using the heuristic sketch method is always worse than the min-wise sampling, averaging 30% error for $z > 1$. For $z < 1$, the sketch method is so inaccurate that its error is sometimes many orders of magnitude wrong. Meanwhile, for skewed data, the min-wise hashing approach achieves an error that is mostly less than 10%. We argue that this skew, with z between 1 and 1.5, is that most commonly seen in network data [23]. Indeed, on the real data (Figure 5 (b)), we see that the min-wise hashing achieves good accuracy, while the sketch approach is much more variable, with accuracy improving as more space is given to the estimation procedure. Note that each plotted point is a single experiment rather than averaging or otherwise trying to smooth out the variability of these methods.

We examined the effect of increasing the number of edges, up to 10 million, on the observed accuracy. This is shown in Figure 5 (c). Our analysis shows that the space required to give ε accuracy is $O(\frac{1}{\varepsilon^4}\sqrt{n})$. Although this looks off-putting, we have seen that with a relatively small amount of space — in this case, 130KB — the Min-wise hashing approach is accurate within a few percentage points. By fixing the space and increasing n, our bound suggests that ε should grow as $n^{1/8}$. That is, for every factor of 10 increase in n, the error should grow by $10^{1/8}$, an increase of approximately a third. Although it is hard to be precise about how the error is growing in the figure, one can see a weak trend of increasing error over the whole domain.

The main drawback of the approach based on min-wise hashing is its speed, compared to the sketching heuristic which consistently takes about 10 seconds to process one million edges, irrespective of the distribution. For distributions with small skew, the min-wise hashing cost is low, comparable to the fast update cost of making a sketch. However, as the skew increases above 1, the update cost rises, roughly linearly, to around 300 seconds for a million edges. This is because of the algorithm: for each edge (i, j) and each node currently in the sample $v_{k,l}$, we test whether $i = v_{k,l}$. For skewed distributions, the chance that i is sampled rises much higher, and so we incur the cost of updating the distinct counters. For less skewed distributions, the node gets sampled by few if any of the hash functions, and so has less chance of requiring an update to a distinct counter method. This leaves as a challenge for future work to combine the accuracy of the min-wise hashing approach with the greater update speed of the sketching approach.

6.2 Degree Estimation

In order to give good quality results for finding the nodes with largest degree, and approximating distinct range sums both require at heart high quality estimators for the degree of a single node. Hence, we evaluate our algorithms for both the problems based on the accuracy that individual degrees are approximated. We set the size of FM sketches to be 10 repetitions, and $L = 5$ throughout. To measure accuracy, we computed the 20 nodes with largest degree, and



(a) (b) (c)

Figure 6: Experiments on Degree Estimation

found the error in each, as a fraction of m, the sum of the degrees, and returned the average error (Computing error on nodes with small degree is not very meaningful: because the approximations are measured as a fraction of m, nodes with low degree have their degrees well approximated by 0).

Figure 6 (a) shows the accuracy when all methods — the sketch based method, using min-wise hashing to draw a sample, and the heuristic of augmenting Lossy Counting with approximate distinct counters — are allocated 175KB of space. For Lossy Counting, which has a pessimistic bound on the amount of space required, we instead adjusted its parameter ε until the amount of space used closely matched that of the other methods. We see very low error with low skew (below $z = 0.8$), but this is not meaningful, since no counts are very high, and approximating all counts with zero achieves similar error. The most interesting region is between $z = 1.0$ and 2.0, where sampling based on min-wise hashing gets the best results, around half as much as sketching and lossy counting. Similar results are seen on real network data as we increase the amount of space available for approximation (Figure 6 (b)). For sufficiently large space, the error of min-wise hashing is up to four times less than that observed for the other two methods.

As the number of edges increases, min-wise hashing still wins out (Figure 6 (c)). There seems little to choose between Lossy Counting and Sketching in terms of accuracy, although one could perhaps argue that the Lossy Counting heuristic is more variable. In terms of time taken, the cost of the min-wise hashing is again apparent. The costs for min-wise hashing and sketching do not vary much as the skewness increases, but the hashing technique is almost two orders of magnitude slower. Sketching costs consistently 3 seconds for a million edges for all values of the zipf parameter. For the Lossy Counting heuristic, the cost is 10–15 seconds for most data, although this cost drops for extremely skewed data to a few seconds for a million edges.

7. EXTENSIONS AND RELATED RESULTS

7.1 Distributed Computation

Our analysis so far has implicitly assumed that all updates are observed at one single location, so that the esimates can be built in a centralized fashion. In general network scenarios, this is not always the case: typically, the relevant multigraph streams are observed at a variety of locations, and of course it is not practical to send these observations to a central site since this would overload the network. However, a common feature of all the algorithms described here is that two data structures created with the same parameters (eg., the same L, K and choice of hash functions) can be merged to give the summary of the union of the streams that created the contents of each data structure. The merging procedure follows from the semantics of the data structure. In our sketch-based solutions, one combines the two data structures by taking pairs of corresponding distinct elements counters and merging their contents, as discussed in Section 2. For min-wise hashing based solutions, the method is a little more involved. For each estimator, if the two values of $min_{k,l}$ are different, then we simply retain the information from the one of the two with the lowest value of $min_{k,l}$. If they are the same, then we merge the information from the two sets of edges: we compute the union of the sets of exact edges, and the union of the approximate distinct counters. The result is identical to that which would have been obtained had all streams been observed at a single location. We omit complete details of the merging procedure.

7.2 Other Cascaded Aggregates

A generalization of this work is to study pairs of cascaded aggregates $P(Q)$, meaning we compute one aggregate—Q—over a data set (such as a graph, matrix, set of values) and then compute P over the vector of values of Q. The problems we have studied here use the distinct elements (number of non-zeros) aggregate on rows (corresponding to nodes) and then further aggregates on the vector of values, i.e. our problem of M_2 computation can be written as the cascaded aggregate $F_2(F_0)$. In general, we are interested in arbitrary $P(Q)$ estimation. The table below gives the streaming space complexity for constant factor, constant probability approximations of various pairings of P (rows) and Q (columns) (we use $L_2 = \sqrt{F_2}$ and Freq denotes the problem of finding values greater than $\phi F_1(Q)$ for some constant ϕ.) The bold entries follow from this paper and those marked "?" are unknown and open. In particular, computing $F_2(F_2)$ and $F_1(L_2)$ remain to be fully understood.

	F_0	F_1	F_2	L_2
F_0	$\mathbf{\Omega(n)}$	$\Omega(n)$	$\Omega(n)$	$\Omega(n)$
F_1	$O(1)$	$O(1)$	$O(1)$	?
F_2	$\mathbf{O(n^{1/2})}$	$O(1)$	?	$O(1)$
$Freq$	$\mathbf{O(1)}$	$O(1)$	$O(1)$	?

Hardness for M_0. A particular cascaded aggregate of interest is M_0, which corresponds to approximating the number of distinct degrees in the underlying graph. By contrast to the above positive results, M_0 cannot be computed in sublinear space. One can prove that $\Omega(n)$ bits are required by reducing to the Disjointness problem:

THEOREM 8. *Approximating M_0 requires space $\Omega(n)$.*

A similar lower bound holds for several related functions of the d_is in a graph or multigraph model. For example, M_0 of the d_i^2 values, (equivalently, $F_0(F_2(F_0))$), and more strongly $F_0(F_2)$ both require as much space, linear in the size of the input graph.

7.3 Sliding Window Computations

In many settings, we do not wish to compute a function over the entire history of observed data, but rather we are only interested in things that happened recently: say, the last week's data, or the last billion transactions. This gives the notion of a "sliding window", dropping the old values which fall outside of the window as it moves to cover new updates. A simple solution is to keep summaries at coarse granularity — one for each day, or one for every ten million transaction — and to combine the most recent of these to make a summary for the window of interest. It is a simple observation that one can combine the results of min-wise hashing or sketches estimators over different time windows, provided that they use the same hash functions. For example, for min-wise hash functions, if both estimators are monitoring the same node v, then we merge the additional information being kept; if not, then we just keep the most recent estimator. However, this requires that we fix the granularity of windows to track *a priori*. With some extra space, we can achieve a stronger result for sliding windows. We show an example below; similar results can be obtained for other multigraph aggregates we have studied in this paper.

COROLLARY 4. *We can compute an approximation of M_2 over windowed multigraph streams. In order to approximate M_2 of the last $w < W$ items, we use (expected) space $O(\frac{1}{\varepsilon^4}\sqrt{n}\log n \log W)$.*

PROOF. Our approach relies on observations made by [12] in the context of tracking the number of infrequently occurring items observed in a data stream. We make some minor changes to our algorithm from Section 3.2. Instead of keeping details about just the v that achieves the smallest hash value, we keep details for a queue of values of v, with associated D and E for each item in the queue. Thus, we add edges (v, j) to the record of distinct edges for every entry in the queue that is monitoring v. Every time we see a new edge, we compute its hash value $h(ni+j)$. If this is less than the current value of $min(h)$ for the most recent item(s) in the queue, we replace the most recent item(s) in the queue, and continue as usual. However, if the hash value is larger than the currently monitored item, we add the new item to the queue as the most recent item. We also check the age of the least recent item in the queue, and remove it from the queue if its age exceeds the size of the window, W.

Note that, at any time, we can retrieve exactly the estimator that we would have computed over just the last window of size w: it is the oldest surviving item in the queue that was added less than w steps before, since its hash value is smallest over that range. Hence, in order to prove the corollary, we just have to demonstrate that the size of the queue is not too large. Using Lemma 3 from [12], we have with high probability that the length of the queue is $\Omega(\ln Dom(h))$, where $Dom(h)$ is the domain of the min-wise hash function h. Hence, with high probability the space required is bounded by $O(\log n)$. □

Other bounds are similar, for example:

COROLLARY 5. *We can compute approximate heavy hitters of windowed multigraph streams for arbitrary windows up to size W using (expected) space $O(\frac{1}{\varepsilon^2}\log(1/\varepsilon)\log n \log W)$.*

7.4 Edge Deletions

The interpretation of a deletion of an edge is it counteracts a previous (or future) insertion of the same edge. Hence, the net occurrence of an edge is the number of insertions of that edge, less the number of deletions of it. We now must count the degree of nodes as the number of edges incident on that node that have non-zero count. Our sketch based algorithms can be modified to work with deletions as well. To accomplish this, we use an approximate distinct counter that is capable of processing deletions, such as [16, 9]. A deletion of an edge is processed as the inverse of an insertion: we hash the edge to a set of counters, and subtract its effect from the distinct counter. The result is equivalent to the state if the corresponding insertion of the edge had never occurred. In particular, for the algorithm of Section 4.2, it follows that estimates given by the POINTQUERY routine of degrees of nodes have the same accuracy bounds as previously stated. In summary:

THEOREM 9. *We can estimate any d_i with error at most εm following a stream of edges with insertions and deletions. The space required is $O(\frac{1}{\varepsilon^3}\log^2 n)$.*

8. CONCLUDING REMARKS

Our results are amongst the first known for multigraph computations on streams with small space usage. Many open problems remain: improved space and time bounds; results in the insert *and* delete models (we have given a solution for approximating the d_is in such a model, but the problem for M_ks remains open); statistics on multigraphs such as subgraph counts. Another direction is to look for multigraph properties such as spanners or diameter in the semi-streaming model. We have mentioned the applicability of our cascaded summaries to distributing summaries in sensor networks because of their idempotent properties; it remains to fully understand their applications in that domain.

Acknowledgments. We thank Avrim Blum and Minos Garofalakis for useful discussions and helpful pointers.

9. REFERENCES

[1] N. Alon, P. Gibbons, Y. Matias, and M. Szegedy. Tracking join and self-join sizes in limited storage. In *Proceedings of the Eighteenth ACM Symposium on Principles of Database Systems*, pages 10–20, 1999.

[2] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing*, pages 20–29, 1996. Journal version in *Journal of Computer and System Sciences*, 58:137–147, 1999.

[3] A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, I. Nishizawa, J. Rosenstein, and J. Widom. STREAM: the Stanford Stream Data Manager (demonstration description). In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 665–665, 2003.

[4] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proceedings of ACM Principles of Database Systems*, pages 1–16, 2002.

[5] Z. Bar-Yossef, T. Jayram, R. Kumar, D. Sivakumar, and L. Trevisian. Counting distinct elements in a data stream. In *Proceedings of RANDOM 2002*, pages 1–10, 2002.

[6] Z. Bar-Yossef, R. Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 623–632, 2002.

[7] A. Blum, P. Gibbons, D. Song, and S. Venkataraman. New streaming algorithms for fast detection of superspreaders. Technical Report IRP-TR-04-23, Intel Research, 2004.

[8] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. In *Procedings of the International Colloquium on Automata, Languages and Programming (ICALP)*, pages 693–703, 2002.

[9] G. Cormode, M. Datar, P. Indyk, and S. Muthukrishnan. Comparing data streams using Hamming norms. In *Proceedings of the International Conference on Very Large Data Bases*, pages 335–345, 2002. Journal version in *IEEE Transactions on Knowledge and Data Engineering* 15(3):529–541, 2003.

[10] G. Cormode and S. Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. In *Latin American Informatics*, pages 29–38, 2004.

[11] C. Cranor, T. Johnson, O. Spatscheck, and V. Shkapenyuk. Gigascope: A stream database for network applications. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 647–651, 2003.

[12] M. Datar and S. Muthukrishnan. Estimating rarity and similarity over data stream windows. In *Proceedings of 10th Annual European Symposium on Algorithms*, volume 2461 of *Lecture Notes in Computer Science*, pages 323–334, 2002.

[13] C. Estan and G. Varghese. New directions in traffic measurement and accounting. In *Proceedings of ACM SIGCOMM*, volume 32, 4 of *Computer Communication Review*, pages 323–338, 2002.

[14] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang. On graph problems in a semi-streaming model. In *Proceedings of the International Colloquium on Automata, Languages, and Programming*, 2004.

[15] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang. Graph distances in the streaming model: The value of space. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms*, 2005.

[16] P. Flajolet and G. N. Martin. Probabilistic counting. In *24th Annual Symposium on Foundations of Computer Science*, pages 76–82, 1983. Journal version in *Journal of Computer and System Sciences*, 31:182–209, 1985.

[17] S. Ganguly, M. Garofalakis, and R. Rastogi. Processing set expressions over continuous update streams. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 265–276, 2003.

[18] M. Garofalakis, J. Gehrke, and R. Rastogi. Querying and mining data streams: You only get one look. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, 2002.

[19] P. Gibbons. Distinct sampling for highly-accurate answers to distinct values queries and event reports. In *Proceedings of the International Conference on Very Large Data Bases*, pages 541–550, 2001.

[20] P. Gibbons and S. Tirthapura. Estimating simple functions on the union of data streams. In *Proceedings of the 13th ACM Symposium on Parallel Algorithms and Architectures*, pages 281–290, 2001.

[21] M. Henzinger, P. Raghavan, and S. Rajagopalan. Computing on data streams. Technical Report SRC 1998-011, DEC Systems Research Centre, 1998.

[22] P. Indyk. A small approximately min-wise independent family of hash functions. *Journal of Algorithms*, 38(1):84–90, 2001.

[23] E. Kohler, J. Li, V. Paxson, and S. Shenker. Observed structure of addresses in IP traffic. In *ACM SIGCOMM Internet Measurement Workshop*, pages 253–266, 2002.

[24] N. Koudas and D. Srivastava. Data stream query processing: A tutorial. In *Proceedings of the International Conference on Very Large Data Bases*, page 1149, 2003.

[25] Internet traffic archive. `http://ita.ee.lbl.gov/`.

[26] G. Manku and R. Motwani. Approximate frequency counts over data streams. In *Proceedings of the International Conference on Very Large Data Bases*, pages 346–357, 2002.

[27] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[28] S. Muthukrishnan. Data streams: Algorithms and applications. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2003.

[29] S. Nath, P. B. Gibbons, S. Seshan, and Z. R. Anderson. Synopsis diffusion for robust aggrgation in sensor networks. In *ACM SenSys*, 2004.

[30] V. Paxson. Empirically derived analytic models of wide-area TCP connections. *IEEE ACM Transactions on Networking*, 2(4):316–336, 1994.