New Estimation Algorithms for Streaming Data: Count-min Can Do More

Fan Deng
University of Alberta
fandeng@cs.ualberta.ca

Davood Rafiei
University of Alberta
drafiei@cs.ualberta.ca

Abstract

Count-min is a general-purpose data stream summary technique, which can be used to answer multiple types of approximate queries such as multiplicity (a.k.a point) queries, join and self-join size estimations, and it has some nice properties such as the one-sided error guarantee, better space bounds and more accurate estimates for highly skewed data in comparison with the best known alternatives. However, based on our experiments for multiplicity queries and self-join size estimations on both synthetic and real data sets, we find that in practice the previous Count-min estimation algorithms only perform well when the data set is highly skewed; in other cases, these algorithms give much less accurate results than Fast-AGMS (a.k.a Count-sketch), which is an improvement based on the influential sketching technique, AMS sketch.

In this paper, based on the existing Count-min data structure, we propose two new estimation algorithms for multiplicity queries and self-join size estimations, which significantly improve the estimation accuracies compared with the previous Count-min estimation algorithms when the data set is less skewed, exactly where the previous algorithms perform poorly. Moreover, we show both in theory and in practice that the performance of our algorithms are very similar to that of Fast-AGMS regardless of input data distributions. Thus, with both the new and previous estimation algorithms, we argue that Count-min is more flexible and powerful than Fast-AGMS, because Count-min performs almost the same as Fast-AGMS in terms of both estimation accuracy and time efficiency using our new estimation algorithms, while Count-min exhibits other nice properties using the previous estimation algorithms as mentioned before.

1 Introduction

Data stream processing has drawn great interests in the database community. Some of the work in the area focus on building data stream management systems, whereas others develop data stream processing algorithms aiming at solving particular problems, which can benefit both data stream systems and particular applications (see [3, 26] for an overview). A typical data stream application, say real-time IP traffic analyses, requires large amount of space and extremely high processing rate. Gigabits of data can arrive within one second at back-bone routers; this makes real-time analyses challenging because large storages (e.g. disks, sometime even ordinary DRAMs [15]) are not fast enough to keep up with the data arrival rate, and fast RAMs are usually limited compared with the data volume. Therefore, both time and space efficiencies of algorithms are extremely important in these kinds of real-time applications.

In general, a data stream is a sequence of tuples, usually ordered based on their arrivals. Data stream sketches (or sketches) are concise data summaries data constructed in one pass to answer particular queries, possibly with errors. Sketches can be useful and important in streaming applications such as real-time IP traffic analyses, sensor network monitoring, web click tracking and so on. In some other scenarios such as massive data analyses, sketches can be helpful as well. Although the data is static and stored on disks in those applications, it can be too expensive to go through the data set multiple times to answer certain queries precisely. Estimates with errors obtained in one pass may be more desirable. Different time and space efficient sketching techniques have been proposed, some dedicated to one type of query, and a few others such as Count-min [10] and Fast-AGMS sketch (a.k.a Count-sketch) [5] can be used to answer multiple queries. Two important queries that can be answered using Count-min or Fast-AGMS sketches are multiplicity queries and self-join size estimations. Although our estimations algorithms can be extended to answering others queries, in this paper we focus on these two queries.

A *multiplicity query*, also called a point query or a frequency query, is to find the number of times a given element appears in a data stream. This is an important query because the techniques for answering multiplicity queries can be often applied to answer other frequency related queries such as iceberg queries [6] (where the goal is to find the elements whose frequencies exceed a threshold), finding top-K frequent elements [5], range queries [10] (where the goal is to find the sum of frequencies of elements within a range), and approximating quantiles [10]. Multiplicity queries are also important in traditional non-streaming settings (see [6] for more examples).

The *self-join size*, also known as the second frequency moment, of a multi-set is $\sum_{i \in D} f_i^2$, where D is the domain from which the values are drawn, and f_i is the frequency of value i. The self-join size indicates the degree of skew of a data set. For data distributions such as Zipfian and exponential, the self-join size uniquely determines the parameter of the distribution [1]. Knowing the parameter of a distribution can improve the accuracy of estimations. For instance, in answering multiplicity queries, we can compute the Zipfian parameter of the input data stream (assuming it follows the Zipfian model), and accordingly choose an algorithm between the new one we propose in this paper and the one previously proposed since both algorithms are based on the same sketch. As another example, the self-join size can be also useful in selecting an optimal sampling strategy to estimate the number of distinct values [18]. More applications of the self-join size can be found in [2] and [1].

To answer multiplicity queries and self-join size estimations, we focus on Count-min sketches [10], which have been implemented on an operational data stream monitoring systems, AT&T's Gigascope [12, 9], for real-time IP traffic analyses (including multiplicity queries and self-join size estimations) and for other operational reasons [22]. Count-min has some nice properties such as one-sided errors and better space bounds (smaller by a factor of $1/\epsilon$, where ϵ is the relative error) in comparison with the best known alternative sketching techniques. However, better space bounds may not always guarantee better performance in practice. Based on our experiments, we find that the previous estimation algorithms using Count-min, referred to as CM, are not as accurate as those using Fast-AGMS [5] on a wide range of data sets. On slightly skewed or uniformly distributed data sets, in particular, Fast-AGMS performs significantly better. In this paper, we demonstrate that Count-min sketches can actually do as well as Fast-AGMS both in theory and in practice regardless of the data distributions using our new estimation algorithms. Furthermore, our new estimation algorithms can be combined with those previous algorithms without conflicts, hence making Count-min a more powerful and flexible sketch.

1.1 Our Contributions

First, we propose a new unbiased estimation algorithm, referred to as CMM, based on Count-min sketches [10] to approximately answer multiplicity queries of data streams. Our experiments on both synthetic and real data sets show that the new algorithm gives much more accurate results (e.g. orders of magnitude improvement on the real data set) than the CM estimation algorithm on a wide range of data sets except when data is highly skewed.

Second, we show through theoretical analyses and experimental evaluations that CMM performs very similarly to the algorithms based on Fast-AGMS sketches [5, 8], and all the analytical results reported for Fast-AGMS [5] also hold for Count-min with our CMM algorithm. Hence, Count-min can be also applied to the cases where Fast-AGMS is used as a building block without losing accuracy, time and space efficiency (e.g. [19] and [17]).

Having two estimation options with different properties, Count-min can do more than Fast-AGMS. For example, the CM estimation approach provides one-sided error approximations, which can be very useful in some cases. In finding frequent elements in a data stream, all candidates whose multiplicities exceed a given threshold are guaranteed to be returned using the CM estimation. Multiplicity estimates for all qualified candidates can be obtained using our CMM approach since it is usually much more accurate in practice. In contrast, Fast-AGMS fails to provide this deterministic guarantee no matter how much space is given. In addition, Count-min with the CM estimation is more accurate than Fast-AGMS when the data set is highly skewed, and CM has a better space bound, meaning that given an error bound and a confidence interval, Count-min using CM needs less space than Fast-AGMS.

Third, we propose a new unbiased algorithm for self-join size estimations based on Count-min sketches. Unless there is a confusion, we will also refer to this algorithm as CMM. Similarly, the accuracy of this algorithm is much better than the previous Count-min estimation algorithm (also referred to as CM) in practice on a wide range of data sets except when the data set is highly skewed. Through our analytical and empirical evaluations we show that CMM performs very similarly to Fast-AGMS in terms of self-join size estimations. Again, having two estimation approaches with different properties makes Count-min a more powerful and flexible data stream summary.

1.2 Paper Outline

The rest of this paper is organized as follows. Section 2 gives background knowledge about Count-min sketches and two other alternatives to be compared with. Then we introduce our CMM estimation algorithm for multiplicity queries in Section 3, where analyses and experimental results are also provided. Section 4 discusses our CMM algorithm for self-join size estimations. In Section 5 we describe the research work closely related to ours. Section 6 concludes the paper.

2 Preliminaries

In this section, we review Count-min sketches [10] and the previous estimation algorithms for answering multiplicity queries and self-join size estimations based on Count-min. Also, for the sake of comparison, we briefly review the Fast-AGMS sketches [5, 8] and Spectral Bloom Filters [6].

2.1 Count-min Sketches

Sketch construction and maintenance. A count-min sketch $CM[i, j]$ is a 2-dimensional array of counters, with d (sketch depth) rows and w (sketch width) columns. All counters are initially set to 0. To insert an element x into the sketch, d hash functions $h_i(x) \in \{0, \ldots, w-1\}$ with $i = 0, \ldots, d-1$, picked uniformly at random are used to determine which counters to be updated. For each row i, counter $CM[i, h_i(x)]$ is incremented by 1. The procedure to delete an element x is similar: for each row i, counter $CM[i, h_i(x)]$ is decremented by 1.

The CM algorithm for answering multiplicity queries. To find the number of occurrences of an element x, all the d counters that x has touched, i.e. $CM[i, h_i(x)]$ with $(i = 0, \ldots, d-1)$, are checked, and the minimum counter value is returned as the estimated frequency of x. Clearly, the estimate is an upper bound of the true frequency.

The CM algorithm for self-join size estimations. For each row i of the sketch, sum up the square of each counter value in that row, and return the minimum sum of all d rows as the estimate. That is, the estimate $\hat{F_2} = min\{sum_{j=0}^{w-1}(CM[i, j])^2, i = 0, \ldots, d-1\}$. This estimate is an upper bound of the true value as well.

2.2 Spectral Bloom Filters

Cohen and Matias [6] propose Spectral Bloom Filters (SBF) to answer multiplicity queries. An SBF is a 1-dimensional array of counters, initially all set to 0. To insert an element into the SBF, k hash functions are used to pick k counters uniformly at random, and those counters are incremented by 1. To answer a query, the k counters the query element has touched are checked, and the minimum value of those k counters is returned as the approximate query answer.

To increase the accuracy, they also propose a heuristic, Minimal Increase (MI), which changes the way an SBF is constructed. To insert an element x into the SBF, only the minimum counter/counters rather than all of the counters x touches is increased by 1. This heuristics decreases the error because it makes the counters increase slower. However, the error reduction depends on data distribution and the order of element insertions. Therefore, analyses become hard and are given in [6] only in the case that element frequencies are uniformly distributed. Also MI does not support element deletions, unlike the basic SBF.

2.3 Fast-AGMS Sketches

Based on an influential sketching technique called AMS [2], Charikar, Chen and Farach-Colton [5] propose Count-sketches to estimate element multiplicities. The same sketches are also called Fast-AGMS sketches [8] in self-join size estimation scenarios. For the ease of presentation, we only use the term Fast-AGMS sketches, to refer to this data structure for the rest of the paper.

Sketch construction and maintenance. The Fast-AGMS sketches are organized as a 2-dimensional array of counters. To insert an element into the sketch, for each row of the sketch, a hash function is used to determine which counter should be updated according to the hash value of the element, and another independent hash function maps the element to either -1 or 1 uniformly at random, indicating the value to be added to the counter. To delete an element from the sketch, based on the same hash functions either -1 or 1 is deducted from the counters the element is hashed to.

The Fast-AGMS algorithm for multiplicity queries. To check the multiplicity of a query element, for each row of the sketch, map the element into a counter and a value (either -1 or 1), using the same two hash functions as in the sketch construction process. Obtain the product between the hash value (-1 or 1) and the value of the counter the element is mapped to, then report the median of those products from all rows as the multiplicity estimate. This estimate is shown [5] to be unbiased.

The Fast-AGMS algorithm for self-join size estimations. The main idea of this algorithm is as follows: for each row of the sketch, sum up the squares of all counter values, and return the median of those sums from all rows as the self-join size estimate. Again, this is also an unbiased estimate.

Next, we introduce our new estimation algorithms for multiplicity queries based on Count-min sketches.

3 Unbiased Estimates for Multiplicity Queries using Count-min Sketches

The estimation procedures described in Section 2.1 give upper bounds of the true values. We propose our estimation methods, *count-mean-min (CMM)* , which gives unbiased estimates for both multiplicity queries and self-join size estimations using exactly the same count-min sketch. We discuss the multiplicity query case in this section.

3.1 Basic Idea

Recall the estimation procedure of CM: given a query element q and hash functions h_i ($i = 0, \ldots, d-1$), the frequency estimate $\hat{f}_q$ is the minimum value of the counters q has touched (i.e. $CM[i, h_i(q)]$, $i = 0, \ldots, d-1$). Usually the counters q touches are also touched by other elements, thus even the minimal counter value is expected to be larger than the true value f_q. The source of the error is the contributions of other elements to the counters $CM[i, h_i(q)]$. We characterize the contributions made by elements other than q to the counters $CM[i, h_i(q)]$ as *noise*. The CM algorithm returns the counter value with the least noise. Our CMM algorithm tries to estimate the noise in each counter, removes the noise and returns the residue.

Of course we do not know exactly the value of the noise since the noise is a random variable, but we can estimate its expected value. For a counter $CM[i, h_i(q)]$, the noise can be estimated from the values of all other counters not touched by q in that row i. The value of each counter not touched by q can be considered as an independent random variable following the same distribution as the noise, assuming that the hash functions map each element i to the range $[0, d-1]$ uniformly at random (pair-wise independence is sufficiently for our theoretical results in this section). In fact, for a multiplicity query, the values of the counters that are not touched by the query element q in row i demonstrate the probability distribution of the noise in counter $CM[i, h_i(q)]$.

3.2 Our Estimation Algorithm

Given a query element q, we use the same set of hash functions h_i ($i = 0, \ldots, d-1$) as used in constructing the Count-min sketch, and check the d counters q is mapped to, i.e. $CM[i, h_i(q)]$ ($i = 0, \ldots, d-1$). Instead of returning the minimum value of the d counters, we deduct the value of estimated noise from each of those d counters, and return the median of the d residues. The estimated noise in each counter $CM[i, h_i(q)]$ can be computed as the average value of all counters in row i except counter $CM[i, h_i(q)]$ itself. That is, the noise is estimated to be $(N - CM[i, h_i(q)])/(w-1)$, where N is the stream size and w is the sketch width.

3.3 Analyses of Our Algorithm

Since for each row of the sketch, the analysis is the same, we just discuss the case for a particular row i. Let X_x be a Bernoulli random variable indicating if element x is hashed to the same counter that the query element q is hashed to, i.e.

$$X_x = \begin{cases} 1, & x \text{ is hashed to the same counter as } q \text{ is;} \\ 0, & \text{otherwise.} \end{cases}$$

Assuming that the hash function maps each element to one of the w counters uniformly at random, the probabilities of the above two cases are as follows: $Pr[X_x = 1] = 1/w$ and $Pr[X_x = 0] = 1 - 1/w$. The value of the counter q is hashed to is also a random variable, $f_q + \sum_{x \neq q} f_x X_x$, where f_q and f_x are the true frequencies of q and x respectively.

Lemma 1. *Given a hash function picked uniformly at random from a pairwise independent family, for a multiplicity query of element q and each row of the sketch, our CMM estimate $\hat{f}_q$ is expected to be f_q, and the variance is $\frac{1}{w-1} \sum_{x \neq q} f_x^2$, where w is the sketch width.*

Proof. We only discuss the case for row i of the sketch here since the analysis for other rows is exactly the same. Recall that our estimation procedure described in Section 3.2 is to deduct an estimated noise, the average value of other counters, from the counter q is hashed to. Thus, the estimate

$$\begin{aligned} \hat{f}_q &= CM[i, h_i(q)] - \frac{1}{w-1}(N - CM[i, h_i(q)]) \\ &= f_q + \sum_{x \neq q} f_x X_x - \frac{1}{w-1}(N - (f_q + \sum_{x \neq q} f_x X_x)) \\ &= f_q + \frac{w}{w-1} \sum_{x \neq q} f_x X_x - \frac{1}{w-1}(N - f_q) \\ &= f_q + \frac{1}{w-1} \sum_{x \neq q} w f_x X_x - \frac{1}{w-1} \sum_{x \neq q} f_x \\ &= f_q + \frac{1}{w-1} \sum_{x \neq q} f_x (w X_x - 1). \end{aligned}$$

The expectation of $\hat{f}_q$

$$\begin{aligned} E[\hat{f}_q] &= f_q + \frac{1}{w-1} \sum_{x \neq q} f_x (w E[X_x] - 1) \\ &= f_q + \frac{1}{w-1} \sum_{x \neq q} f_x (w \frac{1}{w} - 1) = f_q. \end{aligned}$$

Therefore, $\hat{f}_q$ is an unbiased estimate.

The variance of $\hat{f}_q$

$$\begin{aligned} VAR[\hat{f}_q] &= E[(\hat{f}_q - E[\hat{f}_q])^2] \\ &= E[(\frac{1}{w-1} \sum_{x \neq q} f_x (w X_x - 1))^2]. \end{aligned}$$

To simplify the formula, let $Z_x = w X_x - 1$, then

$$E[Z_x] = 0$$

$$E[Z_x^2] = E[w^2 X_x^2 - 2 w X_i + 1] = w - 1.$$

Thus,

$$VAR[\hat{f}_q] = \frac{1}{(w-1)^2} E[(\sum_{x \neq q} f_x Z_x)^2]$$

$$= \quad \frac{1}{(w-1)^2} E[\sum_{x \neq q} f_x^2 Z_x^2 + 2 \sum_{x \neq q} \sum_{\substack{y \neq q \\ y < x}} f_x f_y Z_x Z_y]$$

$$(\because Z_x, Z_y \text{ pairwise independent})$$

$$= \quad \frac{1}{(w-1)^2} (\sum_{x \neq q} f_x^2 E[Z_x^2] + 2 \sum_{x \neq q} \sum_{\substack{y \neq q \\ y < x}} f_x f_y E[Z_x] E[Z_y])$$

$$= \quad \frac{1}{w-1} \sum_{x \neq q} f_x^2.$$

□

Comparison with Fast-AGMS. As discussed in Section 2, Fast-AGMS [5] can be used to answer multiplicity queries as well. It is not hard to show the following statement.

Lemma 2. *Given sketches of the same width and depth and pairwise independent hash functions, the expectation and variance of the estimates from Fast-AGMS are the same as those from our CMM algorithm.*

Proof. Similar to the proof of Lemma 1, we can obtain the expected value and the variance of the Fast-AGMS estimate; the expectation is the same as that of CMM's, and the variance is $\frac{1}{w} \sum_{x \neq q} f_x^2$. For a given element q, let $\bar{f}_q$ be the frequency estimate from a Fast-AGMS sketch with width w and depth d. X_x is still defined as the same indicator variable, indicating whether element x is hashed to the same counter as q is. Let Y_x be another random variable indicating x is hashed to either -1 or 1 with the same probability $1/2$ by another independent hash function. That is

$$Y_x = \begin{cases} 1, & Pr[Y_x = 1] = \frac{1}{2} \\ -1, & Pr[Y_x = -1] = \frac{1}{2}. \end{cases}$$

then we have

$$\bar{f}_q = f_q + \sum_{x \neq q} f_x X_x Y_x,$$

$$E[\bar{f}_q] = f_q + \sum_{x \neq q} f_x E[X_x] E[Y_x] = f_q,$$

$$VAR[\bar{f}_q] = E[(\bar{f}_q - f_q)^2] = E[(\sum_{x \neq q} f_x X_x Y_x)^2]$$

$$= \quad E[\sum_{x \neq q} f_x^2 X_x^2 Y_x^2] + E[2 \sum_{x \neq q} \sum_{\substack{y \neq q \\ y < x}} f_x f_y X_x X_y Y_x Y_y]$$

$$(\because X_x X_y, Y_x Y_y \text{pairwise independent})$$

$$= \quad \sum_{x \neq q} f_x^2 E[X_x^2] E[Y_x^2] +$$

$$2 \sum_{x \neq q} \sum_{\substack{y \neq q \\ y < x}} f_x f_y E[X_x] E[X_y] E[Y_x] E[Y_y]$$

$$= \quad \frac{1}{w} \sum_{x \neq q} f_x^2.$$

Recall that the variance of our CMM estimate is $\frac{1}{w-1} \sum_{x \neq q} f_x^2$, meaning that if CMM is given one more counter in each row, the variances of these two methods will be exactly the same. Given that CMM needs one less hash function in each row, and this can lead to some saving in the storage of the hash functions, we consider the two variances the same. Even if there is any, the difference is negligible especially when the depth of the sketch is small due to the time cost. □

Theorem 1. *The analytical results reported for Fast-AGMS [5] are all applicable to the Count-min sketch using the CMM algorithm. Due to the space limit, we choose not to duplicate them here.*

Proof. Because the expectations and variances of the two methods are the same, all proofs in [5] can be adapted to our CMM estimation. See [5] for the detailed proof. Note that the presentation style and some of the notations in [5] are different from ours. □

3.4 Experiments for Multiplicity Queries

In this section, we experimentally compare CMM to the related estimation algorithms (reviewed in Section 2): CM, Fast-AGMS and SBF with the MI heuristic.

Implementation issues. In our CMM algorithm for answering multiplicity queries, we also use the median of all counters in a sketch row as the estimated noise besides using the mean as described in the algorithm, because median is less sensitive to outliers in data values. Computing the median of the counters not touched by the query element for each query is costly. To improve the time efficiency, we consider the median of all counters as the noise, which can be obtained once and used for all queries. This estimate is still accurate because the median of all counters in one sketch row is approximately the same as the median of that row with one less counter.

To further increase accuracy for both CMM and Fast-AGMS, we return 0 if CMM or Fast-AGMS gives a negative estimate since the estimate is clearly wrong. Similarly, if CMM gives an estimate larger than the one from CM, we return the latter instead since an estimate above the upper bound is also obviously wrong. Having multiple estimates from multiple sketch rows, we return the median as the final estimate for both CMM and Fast-AGMS. The hash functions we use are obtained from MassDal [24].

Synthetic and real data sets. We generated synthetic data sets whose element frequencies followed Zipfian distributions with different Zipfian parameters between 0 and 2. Each data set had 1 million elements, where the elements are integers drawn from the domain from 1 to 1 million. The code used for generating the data sets were also obtained from MassDal [24]. We also ran experiments on a Web crawl data set, originally obtained from Internet Archive [20], containing a stream of URLs sequentially extracting from the crawled pages. We hashed each URL in this collection to a 64-bit fingerprint, verified the data set and find no hash collisions between the URLs. The stream size (number of URL fingerprints) we used was 1 million. Using the second frequency moment of this URL stream, we approximately computed the Zipfian parameter assuming the URL frequencies follow the Zipfian model, and found that the Zipfian parameter were between 0.8 and 0.9. We also used longer and shorter stream sizes, but found similar Zipfian parameters and experimental results.

Experimental settings. In the experiments, we queried the multiplicities of all elements in the domain and the multiplicities of the top-100 frequent elements appeared in the data set using different sketching techniques. We obtained true frequencies of the elements using a sufficiently large buffer, and computed the absolute values of the differences between the estimates and the true frequencies as the error measurement.

Varying the skew of the synthetic data sets. In our first experiment, we query each element in the domain once and return the average of the absolute errors of all queries. The result from data sets with different Zipfian parameters is shown in Figure 1. CMM-mean represents the algorithm using the mean value of counters as the noise, while CMM represents the algorithm using the median of all counter values in a row as the noise. From the figure we can see that when the data set is less skewed, CMM-mean, Fast-AGMS and CMM all perform significantly better than CM and MI, while CM and MI become more accurate than Fast-AGMS when the data set is highly skewed.

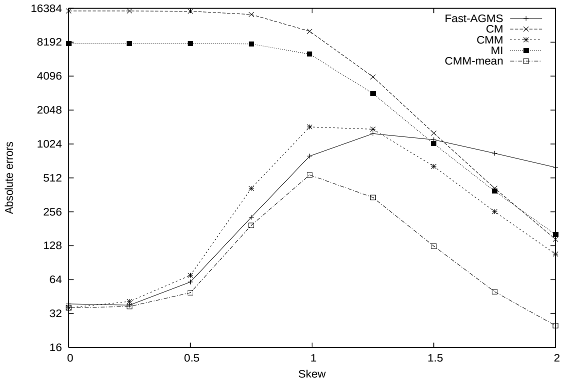

Figure 1. Average absolute errors vs. data set skew, comparing Fast-AGMS, CM, MI and our CMM; queries are all elements in the domain. The sketch width and depth are 64 and 3 respectively.

Among CMM-mean, Fast-AGMS and CMM we also see some differences: CMM-mean and CMM both perform better than Fast-AGMS when the data set is highly skewed because of the CM bound applied to both CMM-mean and CMM; when the data set is less skewed, the performance of Fast-AGMS is between those of CMM-mean and CMM. Actually, CMM-mean performs well mainly because of the 0 bound we used. When the data set is skewed, it is very likely that there are some large outliers in row counters, which make CMM-mean significantly overestimate the noise, and accordingly return a negative estimate. This is good for those 0-frequency elements which do not appear in the data set, because the final CMM-mean estimate will be 0 whenever CMM-mean returns a negative estimate. In contrast, CMM has less chance of overestimating the noise, thus CMM is less likely to take advantage of the 0 bound. Regarding Fast-AGMS, the chance of returning a negative estimate is one half for those 0-frequency elements. Given that a large fraction of query elements in the domain have frequency 0 in the synthetic data sets, which makes the 0 bound a dominant factor, in the rest of our experiments we focus on finding the multiplicities of frequent elements, where the 0 bound has much less impact on the experimental results.

In our second experiment, we query the multiplicities of the top-100 frequent elements. The average of the absolute errors of the 100 query answers on the data sets with different skew is shown in Figure 2. Some general trends observed from the figure are as follows. First, the accuracy difference between CMM and Fast-AGMS is small. Second, CMM and Fast-AGMS significantly outperform CM when the data set is less skewed; the difference becomes smaller when the skew increases; when the data set is highly

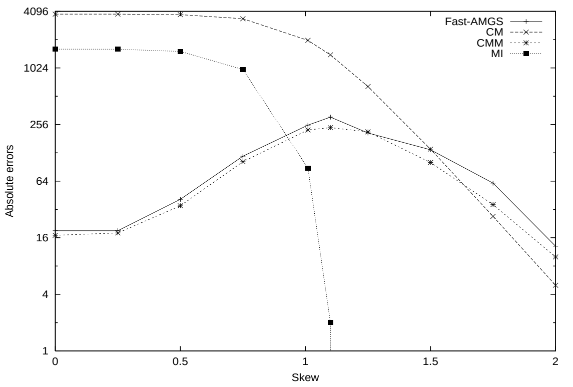

Figure 2. Average absolute errors vs. data set skew, for top-100 frequent element queries using Fast-AGMS, CM, CMM and MI. The sketch width and depth are 256 and 5 respectively.

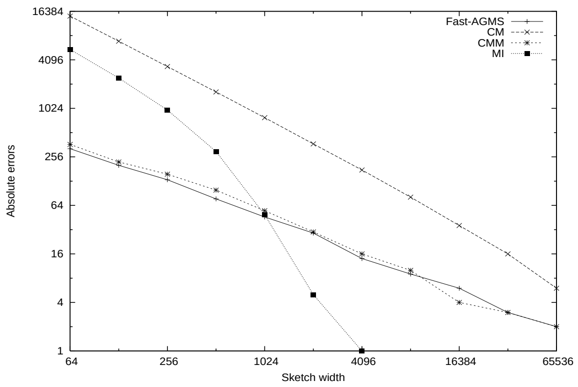

Figure 3. Average absolute errors vs. sketch width, for top-100 frequent element queries using Fast-AGMS, CM, CMM and MI on the 1M URL data set. The sketch depth is 5.

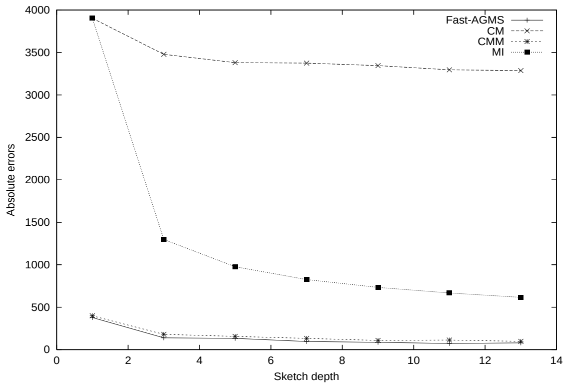

Figure 4. Average absolute errors vs. sketch depth, for top-100 frequent element queries using Fast-AGMS, CM, CMM and MI on the 1M URL data set. The sketch width is 256.

skewed, CM becomes more accurate than CMM and Fast-AGMS. Third, the MI heuristic consistently outperforms CM; but it is still much less accurate than CMM and Fast-AGMS for less skewed data; in the high skew cases, MI is much better.

One clear inconsistence between Figure 1 and 2 is the performance of MI. In the high skew cases, when query elements are the frequent ones, MI performs much better than all others, while MI performs much worse in Figure 1. This is because when the data set is highly skewed, there are less high frequent elements. The counters those elements are mapped to are very likely to be increased to a high value by the frequent elements themselves. When a non-frequent element arrives, it will only increase the minimum counters it is mapped to, which are less likely to be the ones frequent elements have touched because the values of those counters are likely to be very large already. Therefore, when the query elements are frequent ones, MI only gives very small errors. As discussed in Section 2.2, the benefit of this method depends on the frequency distribution and the order in which elements arrive. So it is difficult to be further analyzed.

Varying the sketch width on the real data set. In this set of experiments, we fixed the sketch depth to 5 and varied the sketch width. The results are shown in Figure 3. Similar to the results from the previous experiments, CMM performs very closely to Fast-AGMS, and they both performs significantly better than CM. MI does not perform well when the space is small; but it becomes better when the space is large.

Varying the sketch depth on the real data set. In this set of experiments, we fixed the sketch width to 256 and varied the sketch depth. The results are shown in Figure 4. Similar to the results from the previous experiments, CMM performs very similarly to Fast-AGMS, and they both performs significantly better than CM. MI is better than CM, but not as good as CMM and Fast-AGMS.

3.5 Summary of Comparisons

In this paper, we discuss 4 algorithms for approximately answering multiplicity queries: CM, Fast-AGMS, CMM and MI.

CMM, CM and Fast-AGMS. In general, CMM and Fast-AGMS give better estimates over a larger range of data sets. They perform similarly both in theory and in practice. But CM and CMM are 2 different estimation algorithms using exactly the same sketch. Therefore, the Count-min sketches can be more powerful than Fast-AGMS sketches as discussed in Section 1.

MI and other techniques. The MI heuristic consistently improves the accuracy of CM estimates, especially when the queries are frequent elements. In general, it may perform better than CMM and Fast-AGMS for highly skewed data sets when querying frequent elements. When the data set is less skewed, CMM and Fast-AGMS seem to perform better. But we are unable to reach a conclusion for our comparison because the results of MI may vary greatly even for data sets with the same skew but different element arrival orders. Furthermore, MI does not have certain nice properties, such as the ability to handle element deletions and the ease of analysis, which CMM, CM and Fast-AGMS all have. This is again because the arrival order of elements will change the performance of MI, while this order has no effect on CM, CMM and Fast-AGMS. In other words, the sketch will be the same for CM, CMM or Fast-AGMS as long as the frequencies of elements do not change, and hence the estimation will be the same regardless of the element order. Because MI is hard to be analyzed, the space bound remains the same as that of CM.

Time cost comparisons. The time cost of per element update for CM, CMM and Fast-AGMS is the same, i.e. $O(d)$ where d is the depth of the sketch. The time cost for MI depends on the number of hash functions used, and we are not how to set the number of hash functions properly to minimize the error.

As for the query time cost, CM needs $O(d)$ time to find the minimum counter. Both CMM and Fast-AGMS can find the median in $O(d)$ time as well using the SELECT algorithm [7], under the condition that the mean of all counters except the one touched by the query element is used to estimate the noise in CMM. But if CMM uses the median of each row for noise estimation, as we did in our experiments, then CMM needs $O(w)$ preprocessing time to find the medians of counters for each sketch row. But those medians need to be computed only once and can be used for all queries.

4 Unbiased Self-join Size Estimates from Count-min Sketches

Count-min sketches can be also used to estimate the self-join size of a data stream as discussed in Section 2, where the estimate is an upper bound of the true value. Similar to the case of multiplicity queries, we propose a new estimation algorithm which gives an unbiased self-join size estimate of a data stream.

4.1 Our Estimation Algorithm

The CM algorithm [10] computes the sum of squares of all counters in each sketch row, and returns the minimum sum of all rows as the self-join size estimate. Our approach (CMM) use the same sketch with width w and depth d, but the estimation procedure is different: for each counter in a sketch row, we compute the average value of all other counters in the row except the counter itself, and deduct the average from that counter; by doing this, w residues are obtained, one for each counter. We then calculate the sum of the squares of the w residues, and return the product of the sum and $(w-1)/w$ as the self-join size estimate from that row. The final estimate is the median of the estimates from all d rows.

Formally, given a Count-min sketch $CM[0\dots d-1, 0\dots w-1]$ with d rows and w columns, we return the median of the following d values as the estimate:

$$\frac{w-1}{w}\sum_{j=0}^{w-1}(CM[i,j]-\frac{1}{w-1}(N-CM[i,j]))^2, \qquad 0\le i\le d-1,$$

where N is the stream size, i is the row index and j is the counter index within a row. Next we show that this CMM algorithm gives an unbiased estimate for the self-join size and the variance is the same as that of AMS and Fast-AGMS.

4.2 Analyses of Our Algorithm

Lemma 3. *The estimate from each row of a Count-min sketch using the above CMM algorithm is expected to be the true self-join size (pairwise independent hash functions), and the variance is* $\frac{4}{w-1}\sum_{x<y}f_x^2f_y^2$ *(4-wise independent hash functions), where* x *and* y *is an arbitrary pair of distinct elements of the stream.*

Proof. For a particular row of the sketch, let F_2 be the true self-join size, and $X_{x,j}$ be a Bernoulli random variable indicating if an element x is hashed to counter j ($j=0,\dots,w-1$). That is

$$X_{x,j}=\begin{cases}1, & x \text{ is hashed to the counter } j, (j=0,\dots,w-1);\\ 0, & \text{otherwise.}\end{cases}$$

Then the self-join size estimate

$$\begin{aligned}\hat{F_2} &= \frac{w-1}{w}\sum_{j=0}^{w-1}(\sum_x f_xX_{x,j}-\frac{1}{w-1}(N-\sum_x f_xX_{x,j}))^2\\ &= \frac{1}{w(w-1)}\sum_{j=0}^{w-1}(w\sum_x f_xX_{x,j}-N)^2\\ &= \frac{w}{w-1}\sum_{j=0}^{w-1}(\sum_x f_x^2X_{x,j}^2+2\sum_{x<y}f_xf_yX_{x,j}X_{y,j})\\ &\quad -\frac{1}{w-1}(2N\sum_x f_x\sum_{j=0}^{w-1}X_{x,j}-N^2)\end{aligned}$$

$$
\begin{aligned}
& (\because \sum_{j=0}^{w-1} X_{x,j}^2 = \sum_{j=0}^{w-1} X_{x,j} \equiv 1, \sum_x f_x = N) \\
= \ & \frac{w}{w-1}\sum_x f_x^2 + \frac{2w}{w-1}\sum_{x<y} f_x f_y \sum_{j=0}^{w-1} X_{x,j}X_{y,j} - \frac{N^2}{w-1} \\
= \ & \sum_x f_x^2 + \frac{1}{w-1}(\sum_x f_x^2 - N^2 + 2w\sum_{x<y} f_x f_y \sum_{j=0}^{w-1} X_{x,j}X_{y,j}).
\end{aligned}
$$

The expected estimate

$$
\begin{aligned}
E[\hat{F_2}] \ = \ & \sum_x f_x^2 + \frac{1}{w-1}(\sum_x f_x^2 - N^2 \\
& +2w\sum_{x<y} f_x f_y \sum_{j=0}^{w-1} E[X_{x,j}X_{y,j}]) \\
& (\because \text{the hash function 2-wise independent}, \\
& \therefore E[X_{x,j}X_{y,j}] = \frac{1}{w^2}) \\
= \ & \sum_x f_x^2 + \frac{1}{w-1}((\sum_x f_x)^2 - N^2) = \sum_x f_x^2.
\end{aligned}
$$

The variance

$$
\begin{aligned}
& VAR[\hat{F_2}] = E[\hat{F_2}^2] - (E[\hat{F_2}])^2 \\
= \ & \frac{2}{w-1}(\sum_x f_x^2)E[\sum_x f_x^2 - N^2 \\
& +2w\sum_{x<y} f_x f_y \sum_{j=0}^{w-1} X_{x,j}X_{y,j}] + \frac{1}{(w-1)^2} \\
& E[(\sum_x f_x^2 - N^2 + 2w\sum_{x<y} f_x f_y \sum_{j=0}^{w-1} X_{x,j}X_{y,j})^2] \\
= \ & \frac{1}{(w-1)^2}E[(\sum_x f_x^2 - N^2)^2 \\
& +4w(\sum_x f_x^2 - N^2)(\sum_{x<y} f_x f_y \sum_{j=0}^{w-1} X_{x,j}X_{y,j}) \\
& +4w^2(\sum_{x<y} f_x f_y \sum_{j=0}^{w-1} X_{x,j}X_{y,j})^2] \\
= \ & \frac{1}{(w-1)^2}(2\sum_{x<y} f_x f_y)^2 - \frac{8}{(w-1)^2}(\sum_{x<y} f_x f_y)^2 \\
& +\frac{4w^2}{(w-1)^2}\sum_{x<y}(f_x^2 f_y^2 E[\sum_{j=0}^{w-1} X_{x,j}^2 X_{y,j}^2 \\
& +2\sum_{0\le j<k\le w-1} X_{x,j}X_{y,j}X_{x,k}X_{y,k}]) \\
& +\frac{8w^2}{(w-1)^2}\sum_{\substack{y_1<y_2 \\ x\neq y_1\neq y_2}}(f_x^2 f_{y_1} f_{y_2} E[\sum_{j=0}^{w-1} X_{x,j}^2 X_{y_1,j}X_{y_2,j} \\
& +\sum_{j\neq k} X_{x,j}X_{y_1,j}X_{x,k}X_{y_2,k}]) + \frac{24w^2}{(w-1)^2} \\
& \sum_{x_1<x_2<y_1<y_2}(f_{x_1}f_{x_2}f_{y_1}f_{y_2}E[\sum_{j=0}^{w-1} X_{x_1,j}X_{y_1,j}X_{x_2}X_{y_2,j} \\
& +\sum_{j\neq k} X_{x_1,j}X_{y_1,j}X_{x_2,k}X_{y_2,k}]) \\
= \ & \frac{-4}{(w-1)^2}(\sum_{x<y} f_x f_y)^2 + \frac{4w}{(w-1)^2}\sum_{x<y} f_x^2 f_y^2 \\
& +\frac{8}{(w-1)^2}\sum_{\substack{y_1<y_2 \\ x\neq y_1\neq y_2}} f_x^2 f_{y_1} f_{y_2} + \frac{24w^2}{(w-1)^2} \\
& \sum_{x_1<x_2<y_1<y_2}(f_{x_1}f_{x_2}f_{y_1}f_{y_2}(\frac{1}{w^3} + \frac{w(w-1)}{w^4}) \\
= \ & \frac{4}{w-1}\sum_{x<y} f_x^2 f_y^2.
\end{aligned}
$$

□

Note that the variances of estimates from AMS [2, 1] and Fast-AGMS [8] are both $\frac{4}{w}\sum_{x<y} f_x^2 f_y^2$ given 4-wise independent hash functions. The difference between the expression of this variance and that of our CMM is in the terms w and $w-1$, meaning that CMM needs one more counter to reach the same variance. Since our CMM only needs one hash function per sketch row, while Fast-AGMS needs two per row and AMS needs w per row, CMM needs less space in storing hash functions. Thus, we consider the CMM variance the same as that of Fast-AGMS, and slightly smaller than that of AMS.

Theorem 2. *Let $\hat{F_2}$ be the self-join size estimate of a data stream using our CMM algorithm, and F_2 be the true self-join size. Given $O(log(1/\delta)/\epsilon^2)$ counters, with probability $1-\delta$, the relative error $|\hat{F_2} - F_2|/F_2 \le \epsilon$.*

Proof. This result is the same as that of AMS [2, 1] and Fast-AGMS [8] (the result for Fast-AGMS is shown in the form of join size of two data streams). Since in Lemma 3, we have shown that the variances of the estimates from these algorithms are all the same, the rest of the proof is just applying Chebyshev's Inequality. Details can be found in [2]. □

4.3 Experiments for Self-join Size Estimations

To verify the performance of CMM in estimating self-join sizes, we ran two sets of experiments comparing CMM with CM and Fast-AGMS. Since AMS needs to update all sketch counters for each element, which is too slow for many real-time data stream applications, and Fast-AGMS is a much faster but similar alternative with the same estimation expectation and variance, we do not include AMS in our experiments.

Experimental settings. For each sketch row, we computed a self-join size estimate using CMM, Fast-AGMS and CM respectively. Then for CMM and Fast-AGMS, we return the median of estimates obtained from all sketch rows; for CM, we return the minimum value of estimates obtained from all sketch rows.

The data sets used in this experiments and the sketch construction process were the same as in the multiplicity query experiments described in Section 3.4. Cormode and Muthukrishnan [11] propose a variation of the CM algorithm, called CM-, for less skewed (Zipfian parameter $<$ 1) and uniform data sets. Their experiments on data sets similar to ours shows that CM and CM- performs similarly, hence we did not include CM- in our experiments.

Varying data set skew. In this experiment we fixed the sketch width and depth and varied the skew of the synthetic data sets. The results are shown in Figure 5. The two subfigures are the same except that the second one shows a small error range so that the difference between CMM and Fast-AGMS can be seen.

From the figure we can see that when the data set is low skewed, CMM and Fast-AGMS perform significantly better than CM; when the data set is more skewed, the difference becomes smaller. Furthermore, the difference between CMM and Fast-AGMS is always small.

Varying the sketch width. In this experiment, we fixed the sketch depth varying the sketch width and ran our experiments on the URL fingerprint data set. The results are shown in Figure 6. From the sub-figures we can see that CMM and Fast-AGMS always perform similarly, and they both outperform CM significantly especially when the space is small.

Varying the sketch depth. In this experiment, we fixed the sketch width varying the sketch depth and ran our experiments on the URL fingerprint data set. The results are shown in Figure 7. Again, from the figure we can see that CMM and Fast-AGMS perform similarly, and they both outperform CM significantly. Furthermore, increasing the sketch depth within a small range (e.g. from 1 to 10) has almost no impact on the estimation accuracy for all tested algorithms. Because of the time cost, exponentially increasing the sketch depth is infeasible in most real-time applications.

Time cost comparisons. The time efficiencies for CMM, CM and Fast-AGMS are the same. In terms of per element update, CMM, CM and Fast-AGMS all need $O(d)$ time. Regarding the query answering time, the costs of these methods are still the same: they all scan counters in a sketch row linearly, i.e. $O(w)$ time; in CMM and Fast-AGMS, finding the median of estimates from all rows requires $O(d)$ time using the SELECT algorithm [7]; finding the minimum value of the counters in CM also requires the same time.

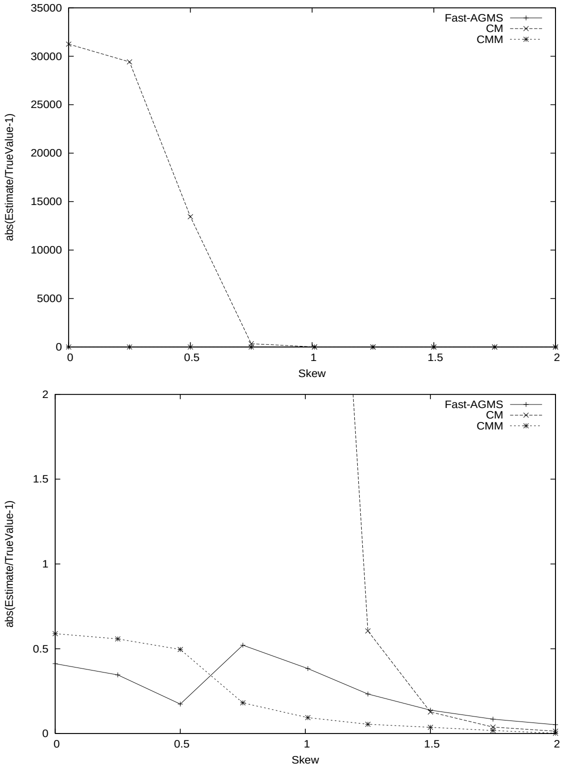

Figure 5. Self-join size estimation errors vs. data set skew, comparing Fast-AGMS, CM and CMM. The sketch width and depth are 16 and 5 respectively. (The 2nd sub-figure zooms in the 1st one). The stream sizes are all 1 million.

5 Related Work

There are many data stream summary techniques, each proposed for different purposes. In this section, we only discuss the work closely related to ours and not covered earlier in this paper.

Bloom filters and their extensions. In 1970, Bloom propose a simple space-efficient data structure, known as Bloom Filters (BF) [4], to approximately answer membership queries, where users give an element and want to know if this element belongs to a particular set. In addition to static data set scenarios, BF is also extended to answer membership queries in a streaming environment [14].

Elements can be inserted into a BF, but cannot be deleted. To handle deletions, Fan et al. [16] extend BF to Counting Bloom Filters (CBF). Even though CBF is orig-

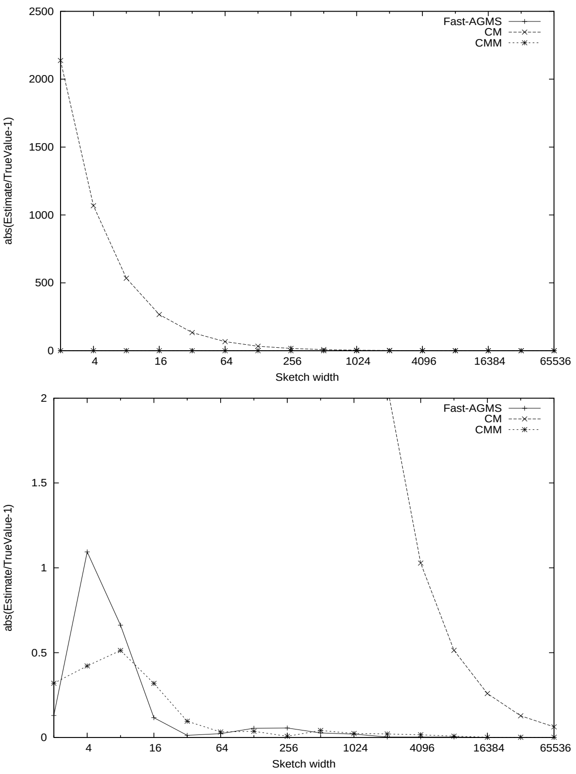

Figure 6. Self-join size estimation error vs. sketch width, comparing Fast-AGMS, CM and CMM on the 1M URL data set. The sketch depth is 3. The 2nd sub-figure zooms in the 1st one.

inally proposed for membership queries, it can be used to answer multiplicity queries as well. In a sketch called Spectral Bloom Filters (SBF), Cohen and Matias [6] use CBF as their basis for answering multiplicity queries. To increase the accuracy, they propose two independent (but incompatible) heuristics: Minimal Increase (MI) and Recurring Minimum (RM). Since RM is less accurate than MI [6], and we have difficulties in setting the parameters (e.g. the secondary SBF size) of RM properly, it is not included in our experiments.

AMS sketches and their extensions In their seminal paper [2], Alon, Matias and Szegedy propose sketching techniques to approximate frequency moments, including a general algorithm approximating the k-th ($k > 1$) frequency moment and an improved algorithm (AMS) specialized in estimating the second frequency moment. In another work [1], Alon, Gibbons, Matias and Szegedy extend their techniques to approximate join and self-join size.

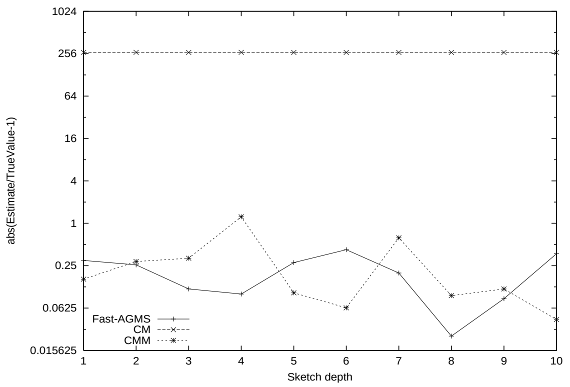

Figure 7. Self-join size estimation error vs. sketch depth, comparing Fast-AGMS, CM and CMM on the 1M URL data set. The sketch width is 16.

Count-sketches [5] and Fast-AGMS [8, 17] are extensions of AMS to answer multiplicity queries and join/self-join size estimations respectively. Both extensions are included in our experiments. The expectation and variance of Fast-AGMS estimates are the same as those of AMS, but Fast-AGMS is much more time-efficient. Thus, we only compare our CMM with Fast-AGMS for self-join size estimations.

Finding frequent elements. There are some work (e.g. [23, 21, 25]) focusing on finding frequent elements approximately in a data stream. These algorithms also construct data summaries in one pass, but they are specialized for finding frequent items and not for other queries.

Recent applications of Count-min and Fast-AGMS. Korn et al. [22] use Count-min sketches as underlying data structures to answer multiplicity queries, self-join size estimations, range sum queries, quantile approximations. Cormode and Garofalakis [8] apply Fast-AGMS in a distributed environment to answer multiple queries such as multiplicity queries, iceberg queries, range queries, join and self-join size estimations. Indyk and Woodruff [19] use Fast-AGMS as a building block to find the k-th ($k > 2$) frequency moments.

6 Conclusions and Future Work

In this paper, we propose new estimation algorithms, CMM, for multiplicity queries and self-join size estimations based on a data stream summary technique, Count-min. Compared to the previous estimation algorithms based on Count-min, our new methods significantly improve the estimation accuracy on a wide range of data sets. In contrast with another influential general-purpose data stream summary technique Fast-AGMS, Count-min sketches can give estimates with the same accuracy, time and space efficiency using CMM. Moreover, there are other attractive estimation options and error bounds for Count-min, which are not applicable to Fast-AGMS; with our new estimation algorithms, we make a case that Count-min is more flexible and powerful.

In addition to the applications of finding the top-k frequent elements and answering iceberg queries, CMM can be potentially extended to answer other queries such as SUM aggregates (i.e. a generalization of multiplicity queries where frequency updates are not limited to 1 and -1), range queries, quantiles approximations and join size estimations, as shown by Cormode and Muthukrishnan [10] for CM. Some of these extensions can be straightforward, but others may need further research.

References

[1] N. Alon, P. B. Gibbons, Y. Matias, and M. Szegedy. Tracking join and self-join sizes in limited storage. In *PODS*, 1999.

[2] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *JCSS*, 58(1), 1999. Also in: *STOC*, 1996.

[3] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *PODS*, 2002.

[4] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *CACM*, 13(7), 1970.

[5] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. *Theoretical Computer Science (TCS)*, 312(1), 2004. Also in: *ICALP*, 2002.

[6] S. Cohen and Y. Matias. Spectral bloom filters. In *SIGMOD*, 2003.

[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, 2nd Edition*. The MIT Press, 2001.

[8] G. Cormode and M. N. Garofalakis. Sketching streams through the net: Distributed approximate query tracking. In *VLDB*, 2005.

[9] G. Cormode, T. Johnson, F. Korn, S. Muthukrishnan, O. Spatscheck, and D. Srivastava. Holistic udafs at streaming speeds. In *SIGMOD*, 2004.

[10] G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1), 2005. Also in: *LATIN*, 2004.

[11] G. Cormode and S. Muthukrishnan. Summarizing and mining skewed data streams. In *SIAM International Conference on Data Mining (SDM)*, 2005.

[12] C. D. Cranor, T. Johnson, O. Spatscheck, and V. Shkapenyuk. Gigascope: A stream database for network applications. In *SIGMOD*, 2003.

[13] F. Deng and D. Rafiei. New estimation algorithms for streaming data: Count-min can do more (extended version). `http://www.cs.ualberta.ca/~fandeng/paper/cmm_full.pdf`.

[14] F. Deng and D. Rafiei. Approximately detecting duplicates for streaming data using stable bloom filters. In *SIGMOD*, 2006.

[15] C. Estan and G. Varghese. New directions in traffic measurement and accounting. In *SIGCOMM*, 2002.

[16] L. Fan, P. Cao, J. M. Almeida, and A. Z. Broder. Summary cache: a scalable wide-area web cache sharing protocol. *IEEE/ACM Transactions on Networking*, 8(3), 2000. also in:*SIGCOMM*, 1998.

[17] S. Ganguly, M. N. Garofalakis, and R. Rastogi. Processing data-stream join aggregates using skimmed sketches. In *EDBT*, 2004.

[18] P. J. Haas, J. F. Naughton, S. Seshadri, and L. Stokes. Sampling-based estimation of the number of distinct values of an attribute. In *VLDB*, 1995.

[19] P. Indyk and D. P. Woodruff. Optimal approximations of the frequency moments of data streams. In *STOC*, 2005.

[20] InternetArchive. `http://www.archive.org/`.

[21] R. M. Karp, S. Shenker, and C. H. Papadimitriou. A simple algorithm for finding frequent elements in streams and bags. *TODS*, 28(1), 2003.

[22] F. Korn, S. Muthukrishnan, and Y. Wu. Modeling skew in data streams. In *SIGMOD*, 2006.

[23] G. S. Manku and R. Motwani. Approximate frequency counts over data streams. In *VLDB*, 2002.

[24] MassDal. Massdal public code bank. `http://www.cs.rutgers.edu/~muthu/massdal-code-index.html`, 2006.

[25] A. Metwally, D. Agrawal, and A. E. Abbadi. Efficient computation of frequent and top-k elements in data streams. In *ICDT*, 2005.

[26] S. Muthukrishnan. Data streams: algorithms and applications. In *SODA*, 2003.