# Processing Data-Stream Join Aggregates Using Skimmed Sketches

Sumit Ganguly, Minos Garofalakis, and Rajeev Rastogi

Bell Laboratories, Lucent Technologies, Murray Hill NJ, USA.
{sganguly,minos,rastogi}@research.bell-labs.com

**Abstract.** There is a growing interest in on-line algorithms for analyzing and querying data streams, that examine each stream element only once and have at their disposal, only a limited amount of memory. Providing (perhaps approximate) answers to aggregate queries over such streams is a crucial requirement for many application environments; examples include large IP network installations where performance data from different parts of the network needs to be continuously collected and analyzed. In this paper, we present the *skimmed-sketch* algorithm for estimating the join size of two streams. (Our techniques also readily extend to other join-aggregate queries.) To the best of our knowledge, our skimmed-sketch technique is the first comprehensive join-size estimation algorithm to provide tight error guarantees while: (1) achieving the lower bound on the space required by any join-size estimation method in a streaming environment, (2) handling streams containing general update operations (inserts and deletes), (3) incurring a low logarithmic processing time per stream element, and (4) not assuming any a-priori knowledge of the frequency distribution for domain values. Our skimmed-sketch technique achieves all of the above by first skimming the dense frequencies from random hash-sketch summaries of the two streams. It then computes the subjoin size involving only dense frequencies directly, and uses the skimmed sketches only to approximate subjoin sizes for the non-dense frequencies. Results from our experimental study with real-life as well as synthetic data streams indicate that our skimmed-sketch algorithm provides significantly more accurate estimates for join sizes compared to earlier sketch-based techniques.

## 1 Introduction

In a number of application domains, data arrives continuously in the form of a stream, and needs to be processed in an on-line fashion. For example, in the network installations of large Telecom and Internet service providers, detailed usage information (e.g., Call Detail Records or CDRs, IP traffic statistics due to SNMP/RMON polling, etc.) from different parts of the network needs to be continuously collected and analyzed for interesting trends. Other applications that generate rapid, continuous and large volumes of stream data include transactions in retail chains, ATM and credit card operations in banks, weather measurements, sensor networks, etc. Further, for many mission-critical tasks such as fraud/anomaly detection in Telecom networks, it is important to be able to answer queries in real-time and infer interesting patterns on-line. As a result, recent years have witnessed an increasing interest in designing *single-pass* algorithms for querying and mining data streams that examine each element in the stream *only once*.

The large volumes of stream data, real-time response requirements of streaming applications, and modern computer architectures impose two additional constraints on algorithms for querying streams: (1) the time for processing each stream element must be small, and (2) the amount of memory available to the query processor is limited. Thus, the challenge is to develop algorithms that can summarize data streams in a concise, but reasonably accurate, *synopsis* that can be stored in the allotted (small) amount of memory and can be used to provide *approximate answers* to user queries with some guarantees on the approximation error.

**Previous Work.** Recently, single-pass algorithms for processing streams in the presence of limited memory have been proposed for several different problems; examples include quantile and order-statistics computation [1, 2], estimating frequency moments and join sizes [3–5], distinct values [6, 7], frequent stream elements [8–10], computing one-dimensional Haar wavelet decompositions [11], and maintaining samples and simple statistics over sliding windows [12].

A particularly challenging problem is that of answering aggregate SQL queries over data streams. Techniques based on random stream sampling [13] are known to give very poor result estimates for queries involving one or more joins [14, 4, 15]. Alon et al. [4, 3] propose algorithms that employ small *pseudo-random sketch summaries* to estimate the size of self-joins and binary joins over data streams. Their algorithms rely on a single-pass method for computing a randomized *sketch* of a stream, which is basically a random linear projection of the underlying frequency vector. A key benefi t of using such linear projections is that dealing with *delete* operations in the stream becomes straightforward [4]; this is not the case, e.g., with sampling, where a sequence of deletions can easily deplete the maintained sample summary. Alon et al. [4] also derive a *lower bound* on the (worst-case) space requirement of *any* streaming algorithm for join-size estimation. Their result shows that, to accurately estimate a join size of $J$ over streams with $n$ tuples, any approximation scheme requires at least $\Omega(n^2/J)$ space. Unfortunately, the worst-case space usage of their proposed sketch-based estimator is much worse: it can be as high as $O(n^4/J^2)$, i.e., the square of the lower bound shown in [4]; furthermore, the required processing time per stream element is proportional to their synopsis size (i.e., also $O(n^4/J^2)$), which may render their estimators unusable for high-volume, rapid-rate data streams.

In order to reduce the storage requirements of the basic sketching algorithm of [4], Dobra et al. [5] suggest an approach based on partitioning domain values and estimating the overall join size as the sum of the join sizes for each partition. However, in order to compute good partitions, their algorithms require a-priori knowledge of the data distribution in the form of coarse frequency statistics (e.g., histograms). This may not always be available in a data-stream setting, and is a serious limitation of the approach.

**Our Contributions.** In this paper, we present the *skimmed-sketch* algorithm for estimating the join size of two streams. Our skimmed-sketch technique is the fi rst comprehensive join-size estimation algorithm to provide tight error guarantees while satisfying all of the following: (1) Our algorithm requires $O(n^2/J)$ bits of memory[1] (in the worst

[1] In reality, there are additional logarithmic terms; however, we ignore them since they will generally be very small.

case) for estimating joins of size $J$, which matches the lower bound of [4] and, thus, the best possible (worst-case) space bound achievable by *any* join-size estimation method in a streaming environment; (2) Being based on sketches, our algorithm can handle streams containing general update operations; (3) Our algorithm incurs very low processing time per stream element to maintain in-memory sketches – update times are only logarithmic in the domain size and number of stream elements; and, (4) Our algorithm does not assume any a-priori knowledge of the underlying data distribution. None of the earlier proposed schemes for join size estimation: sampling, basic sketching [4], or sketching with domain partitioning [5], satisfy all four of the above-mentioned properties. In fact, our skimmed-sketch algorithm achieves the same accuracy guarantees as the basic sketching algorithm of [4], using only square root of the space and guaranteed logarithmic processing times per stream element. Note that, even though our discussion here focuses on join size estimation (i.e., binary-join `COUNT` queries), our skimmed-sketch method can readily be extended to handle complex, multi-join queries containing general aggregate operators (e.g., `SUM`), in a manner similar to that described in [5]. More concretely, our key contributions can be summarized as follows.

• SKIMMED-SKETCH ALGORITHM FOR JOIN SIZE ESTIMATION. Our skimmed-sketch algorithm is similar in spirit to the *bifocal sampling* technique of [16], but tailored to a data-stream setting. Instead of samples, our skimmed-sketch method employs randomized hash sketch summaries of streams $F$ and $G$ to approximate the size of $F \bowtie G$ in two steps. It first skims from the sketches for $F$ and $G$, all the *dense* frequency values greater than or equal to a threshold $T$. Thus, each *skimmed sketch*, after the dense frequency values have been extracted, only reflects *sparse* frequency values less than $T$. In the second step, our algorithm estimates the overall join size as the sum of the subjoin sizes for the four combinations involving dense and sparse frequencies from the two streams. As our analysis shows, by skimming the dense frequencies away from the sketches, our algorithm drastically reduces the amount of memory needed for accurate join size estimation.

• RANDOMIZED HASH SKETCHES TO REDUCE PROCESSING TIMES. Like the basic sketching method of [4], our skimmed-sketch algorithm relies on randomized sketches; however, unlike basic sketching, our join estimation algorithm arranges the random sketches in a hash structure (similar to the COUNTSKETCH data structure of [8]). As a result, processing a stream element requires only a single sketch per hash table to be updated (i.e., the sketch for the hash bucket that the element maps to), rather than updating all the sketches in the synopsis (as in basic sketching [4]). Thus, the per-element overhead incurred by our skimmed-sketch technique is much lower, and is only logarithmic in the domain and stream sizes. To the best of our knowledge, ours is the first join-size estimation algorithm for a streaming environment to employ randomized hash sketches, and incur guaranteed logarithmic processing time per stream element.

• EXPERIMENTAL RESULTS VALIDATING OUR SKIMMED-SKETCH TECHNIQUE. We present the results of an experimental study with real-life and synthetic data sets that verify the effectiveness of our skimmed-sketch approach to join size estimation. Our results indicate that, besides giving much stronger asymptotic guarantees, our skimmed-sketch technique also provides significantly more accurate estimates for join sizes compared to other known sketch-based methods, the improvement in accuracy ranging from a factor of five (for moderate data skews) to several orders of magnitude (when the skew in the frequency distribution is higher). Furthermore, even with a few kilobytes of memory, the relative error in the final answer returned by our method is generally less than 10%.

The idea of separating dense and sparse frequencies when joining two relations was first introduced by Ganguly et al. [16]. Their equi-join size estimation technique, termed *bifocal sampling*, computes samples from both relations, and uses the samples to individually estimate sizes for the four subjoins involving dense and sparse sets of frequencies from the two relations. Unfortunately, bifocal sampling is unsuitable for a one-pass, streaming environment; more specifically, for subjoins involving the sparse frequencies of a relation, bifocal sampling assumes the existence of *indices* to access (possibly multiple times) relation tuples to determine sparse frequency counts. (A more detailed etailed comparison of our skimmed-sketch method with bifocal sampling can be found in the full version of this paper [17].)

## 2 Streams and Random Sketches

### 2.1 The Stream Data-Processing Model

We begin by describing the key elements of our generic architecture for processing join queries over two continuous data streams $F$ and $G$ (depicted in Figure 1); similar architectures for stream processing have been described elsewhere (e.g., [5]). Each data stream is an *unordered* sequence of elements with values from the domain $\mathcal{D} = \{1, \ldots, m\}$. For simplicity of exposition, we implicitly associate with each element, the semantics of an insert operation; again, being based on random linear projections, our sketch summaries can readily handle deletes, as in [4, 5].

The class of stream queries we consider in this paper is of the general form $Q = \mathtt{AGG}(F \bowtie G)$, where AGG is an arbitrary aggregate operator (e.g., COUNT, SUM or AVERAGE). Suppose that $f_u$ and $g_u$ denote the frequencies of domain value $u$ in streams $F$ and $G$, respectively. Then, the result of the join size query $\mathtt{COUNT}(F \bowtie G)$ is $\sum_{u \in \mathcal{D}} f_u \cdot g_u$. Alternately, if $f$ and $g$ are the frequency vectors for streams $F$ and $G$, then $\mathtt{COUNT}(F \bowtie G) = f \cdot g$, the inner product of vectors $f$ and $g$. Similarly, if each element $e \in G$ has an associated *measure* value $M_e$ (in addition to its value $u$ from domain $\mathcal{D}$), and $h_u$ is the sum of the measure values of all the elements in $G$ with value $u \in \mathcal{D}$, then $\mathtt{SUM}_M(F \bowtie G) = \sum_u f_u \cdot h_u$. Thus, $\mathtt{SUM}_M(F \bowtie G)$ is essentially a special case of a COUNT query over streams $F$ and $H$, where $H$ is derived from $G$ by repeating each element $e$, $M_e$ number of times. Consequently, we focus exclusively on answering COUNT queries in the remainder of this paper. Without loss of generality, we use $n$ to represent the size of each of the data streams $F$ and $G$; that is, $|F| = |G| = n$. Thus, $\sum_u f_u = \sum_u g_u = n$.

In contrast to conventional DBMS query processors, our stream query-processing engine is allowed to see the elements in $F$ and $G$ *only once* and in fixed order as they are streaming in from their respective source(s). Backtracking over the data stream and explicit access to past elements are impossible. Further, the order of element arrival in each stream is arbitrary and elements with duplicate values can occur anywhere over the duration of the stream.

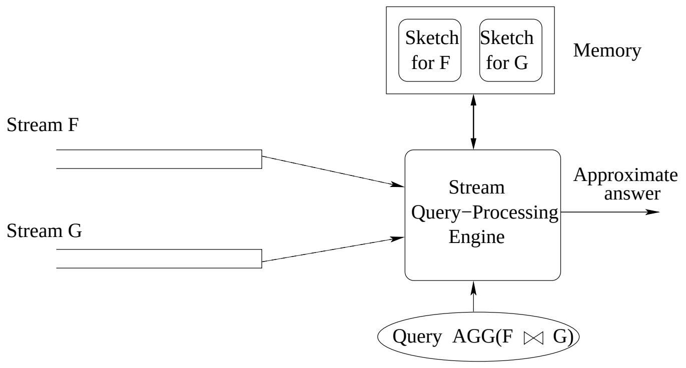

**Fig. 1.** Stream Query-Processing Architecture.

Our stream query-processing engine is also allowed a certain amount of memory, typically significantly smaller than the total size of the data stream(s). This memory is used to maintain a concise and accurate *synopsis* of each of the data streams $F$ and $G$, denoted by $\mathcal{S}(F)$ and $\mathcal{S}(G)$, respectively. The key constraints imposed on each such synopsis, say $\mathcal{S}(F)$, are that: (1) it is much smaller than the total number of tuples in $F$ (e.g., its size is logarithmic or polylogarithmic in $|F|$), and (2) it can be computed in a single pass over the data tuples in $F$ in the (arbitrary) order of their arrival. At any point in time, our query-processing algorithms can combine the maintained synopses $\mathcal{S}(F)$ and $\mathcal{S}(G)$ to produce an approximate answer to the input query $Q = \texttt{COUNT}(F \bowtie G)$. Once again, we would like to point out that our techniques can easily be extended to multi-join queries, as in [5]. Also, selection predicates can easily be incorporated into our stream processing model – we simply drop from the streams, elements that do not satisfy the predicates (prior to updating the synopses).

## 2.2 Pseudo-Random Sketch Summaries

**The Basic Technique: Self-Join Size Tracking.** Consider a simple stream-processing scenario where the goal is to estimate the size of the self-join of stream $F$ as elements of $F$ are streaming in; thus, we seek to approximate the result of query $Q = \texttt{COUNT}(F \bowtie F)$. More specifically, since the number of elements in $F$ with domain value $u$ is $f_u$, we want to produce an estimate for the expression $f^2 = \sum_{u \in \mathcal{D}} f_u^2$ (i.e., the *second moment* of $f$). In their seminal paper, Alon, Matias, and Szegedy [3] prove that *any deterministic algorithm* that produces a tight approximation to $f^2$ requires at least $\Omega(m)$ bits of storage, rendering such solutions impractical for a data-stream setting. Instead, they propose a *randomized technique* that offers strong probabilistic guarantees on the quality of the resulting $f^2$ approximation while using only $O(\log(m))$ space. Briefly, the basic idea of their scheme is to define a random variable $Z$ that can be easily computed over the streaming values of $F$, such that (1) $Z$ is an *unbiased* (i.e., correct on expectation) estimator for $f^2$, so that $E[Z] = f^2$; and, (2) $Z$ has sufficiently small variance

$Var(Z)$ to provide strong probabilistic guarantees for the quality of the estimate. This random variable $Z$ is constructed on-line from the streaming values of $F$ as follows:

- Select a family of *four-wise independent binary random variables* $\{\xi_u : u = 1, \ldots, m\}$, where each $\xi_u \in \{-1, +1\}$ and $P[\xi_u = +1] = P[\xi_u = -1] = 1/2$ (i.e., $E[\xi_u] = 0$). Informally, the four-wise independence condition means that for any 4-tuple of $\xi_u$ variables and for any 4-tuple of $\{-1, +1\}$ values, the probability that the values of the variables coincide with those in the $\{-1, +1\}$ 4-tuple is exactly $1/16$ (the product of the equality probabilities for each individual $\xi_u$). The crucial point here is that, by employing known tools (e.g., orthogonal arrays) for the explicit construction of small sample spaces supporting four-wise independent random variables, such families can be efficiently constructed on-line using only $O(\log(m))$ space [3].
- Define $Z = X^2$, where $X = \sum_{u \in \mathcal{D}} f_u \xi_u$. Note that $X$ is simply a randomized linear projection (inner product) of the frequency vector of $F$ with the vector of $\xi_u$'s that can be efficiently generated from the streaming values of $F$ as follows: Start with $X = 0$ and simply add $\xi_u$ to $X$ whenever an element with value $u$ is observed in stream $F$. (If the stream element specifies a deletion of value $u$ from $F$, then simply subtract $\xi_u$ from $X$).

We refer to the above randomized linear projection $X$ of $F$'s frequency vector as an *atomic sketch* for stream $F$. To further improve the quality of the estimation guarantees, Alon, Matias, and Szegedy propose a standard *boosting technique* that maintains several independent identically-distributed (iid) instantiations of the random variables $Z$ and uses averaging and median-selection operators to boost accuracy and probabilistic confidence. (Independent instances can be constructed by simply selecting independent random seeds for generating the families of four-wise independent $\xi_u$'s for each instance.) Specifically, the synopsis $\mathcal{S}(F)$ comprises of a two-dimensional array of $s_1 \cdot s_2$ atomic sketches, where $s_1$ is a parameter that determines the *accuracy* of the result and $s_2$ determines the *confidence* in the estimate. Each atomic sketch in the synopsis array, $X[i, j]$, $1 \leq i \leq s_2$, $1 \leq j \leq s_1$, uses the same on-line construction as the variable $X$ (described earlier), but with an independent family of four-wise independent variables $\{\xi_u^{ij} : u = 1, \ldots, m\}$. Thus, atomic sketch $X[i, j] = \sum_u f_u \xi_u^{ij}$. The final boosted estimate $Y$ of $f^2$ is the median of $s_2$ random variables $Y_1, \ldots, Y_{s_2}$, each $Y_i$ being the average of the squares of the $s_1$ iid atomic sketches $X[i, j]$, $j = 1, \ldots, s_1$. (We denote the above-described procedure as ESTSJSIZE$(X, s_1, s_2)$.)

The following theorem [3] demonstrates that the above sketch-based method offers strong probabilistic guarantees for the second-moment estimate while utilizing only $O(s_1 \cdot s_2 \cdot (\log(m) + \log(n)))$ space – here $O(\log(m))$ space is required to generate the $\xi_u^{ij}$ variables for each atomic sketch, and $\log(n)$ bits are needed to store the atomic sketch value.

**Theorem 1 ([3]).** The estimate $Y$ computed by ESTSJSIZE satisfies: $P[|Y - f^2| \leq (4/\sqrt{s_1}) f^2] \geq 1 - 2^{-s_2/2}$. This implies that ESTSJSIZE estimates $f^2$ with a relative error of at most $\epsilon$ with probability at least $1 - \delta$ (i.e., $P[|Y - f^2| \leq \epsilon \cdot f^2] \geq 1 - \delta$) while using only $O\left(\frac{\log(1/\delta)}{\epsilon^2}(\log(m) + \log(n))\right)$ bits of memory. $\square$

In the remainder of the paper, we will use the term *sketch* to refer to the overall synopsis array $X$ containing $s_1 \cdot s_2$ atomic sketches for a stream.

**Binary-Join Size Estimation.** In a more recent paper, Alon et al. [4] show how their sketch-based approach applies to handling the size-estimation problem for binary joins over a pair of distinct streams. More specifically, consider approximating the result of the query $Q = \texttt{COUNT}(F \bowtie G)$ over two streams $F$ and $G$. As described previously, let $X_F$ and $X_G$ be the sketches for streams $F$ and $G$, each containing $s_1 \cdot s_2$ atomic sketches. Thus, each atomic sketch $X_F[i,j] = \sum_{u \in \mathcal{D}} f_u \xi_u^{ij}$ and $X_G[i,j] = \sum_{u \in \mathcal{D}} g_u \xi_u^{ij}$. Here, $\{\xi_u^{ij} : u = 1, \ldots, m\}$ is a family of four-wise independent $\{-1,+1\}$ random variables with $E[\xi_u^{ij}] = 0$, and $f_u$ and $g_u$ represent the frequencies of domain value $u$ in streams $F$ and $G$, respectively. An important point to note here is that the atomic sketch pair $X_F[i,j]$, $X_G[i,j]$ share the same family of random variables $\{\xi_u^{ij}\}$. The binary-join size of $F$ and $G$, i.e., the inner product $f \cdot g = \sum_{u \in \mathcal{D}} f_u \cdot g_u$, can be estimated using sketches $X_F$ and $X_G$ as described in the ESTJOINSIZE procedure (see Figure 2). (Note that ESTSJSIZE$(X, s_1, s_2)$ is simply ESTJOINSIZE$(X, X, s_1, s_2)$.)

```
Procedure ESTJOINSIZE($X_F, X_G, s_1, s_2$)
Input: Sketches $X_F$ and $X_G$ for streams $F$ and $G$ (respectively).
Output: Estimate of binary-join size of $F$ and $G$.
begin
1.  for $i = 1$ to $s_2$ do  $Y_i$ := $(\sum_{j=1}^{s_1} X_F[i,j] \cdot X_G[i,j])/s_1$;
2.  return  median $\{Y_1, Y_2, \ldots, Y_{s_2}\}$;
end
```

**Fig. 2.** Join-Size Estimation using Basic Sketching.

The following theorem (a variant of the result in [4]) shows how sketching can be applied for accurately estimating binary-join sizes in limited space.

**Theorem 2 ([4]).** The estimate $Y$ computed by ESTJOINSIZE satisfies: $P[|Y - (f \cdot g)| \leq 4\sqrt{\frac{f^2 \cdot g^2}{s_1}}] \leq 1 - 2^{-s_2/2}$. This implies that ESTJOINSIZE estimates $f \cdot g$ with a relative error of at most $\epsilon$ with probability at least $1-\delta$ (i.e., $P[|Y-(f \cdot g)| \leq \epsilon \cdot (f \cdot g)] \geq 1 - \delta$) while using only $O\left(\frac{\log(1/\delta) \cdot f^2 \cdot g^2}{\epsilon^2 \cdot (f \cdot g)^2}(\log(m) + \log(n))\right)$ bits of memory. $\square$

Alon et al. [4] also prove that no binary-join size estimation algorithm can provide good guarantees on the accuracy of the final answer unless it stores $\Omega(n^2/(f \cdot g))$ bits. Unfortunately, their *basic sketching* procedure ESTJOINSIZE, in the worst case, requires $O(\frac{n^4}{\epsilon^2 \cdot (f \cdot g)^2})$ space, which is the square of their lower bound. This is because, as indicated in Theorem 2, for a given $s_1$, the maximum cumulative error of the estimate $Y$ returned by ESTJOINSIZE can be as high as $O(\sqrt{\frac{f^2 \cdot g^2}{s_1}})$. Stated alternately, for a desired level of accuracy $\epsilon$, the parameter $s_1$ for each sketch is $s_1 = O(\frac{f^2 \cdot g^2}{\epsilon^2 \cdot (f \cdot g)^2})$. Since, in the worst case, $f^2 = g^2 = n^2$, the required space $s_1$ becomes $O(\frac{n^4}{\epsilon^2 \cdot (f \cdot g)^2})$, which is the square of the lower bound, and can be quite large. Another drawback of the basic sketching procedure ESTJOINSIZE is that processing each stream element involves updating every one of the $s_1 \cdot s_2$ atomic sketches. This is clearly undesirable given that basic sketching needs to store $O(n^4/(f \cdot g)^2)$ atomic sketches and, furthermore, *any* sketch-based join-size estimation algorithm requires at least $\Omega(n^2/(f \cdot g))$ atomic sketches. Ideally, we would like for our join size estimation technique to incur an overhead per element that is only logarithmic in $m$ and $n$. So far, no known join size estimation method for streams meets the storage lower bound of $\Omega(n^2/(f \cdot g))$ while incurring at most logarithmic processing time per stream element, except for simple random sampling which, unfortunately, (1) cannot handle delete operations, and (2) typically performs *much* worse than sketches in practice [4].

## 3 Intuition Underlying our Skimmed-Sketch Algorithm

In this section, we present the key intuition behind our skimmed-sketch technique which achieves the space lower bound of $\Omega(n^2/(f \cdot g))$, while providing guarantees on the quality of the $f \cdot g$ estimate. For illustrative purposes, we describe the key, high-level ideas underlying our technique using the earlier sketches $X_F$ and $X_G$ and the basic-sketching procedure ESTJOINSIZE described in Section 2. As mentioned earlier, however, maintaining these sketches incurs space and time overheads proportional to $n^4/(f \cdot g)^2$, which can be excessive for data-stream settings. Consequently, in the next section, we introduce random hash sketches, which, unlike the sketches $X_F$ and $X_G$, arrange atomic sketches in a hash structure; we then present our skimmed-sketch algorithm that employs these hash sketches to achieve the space lower bound of $\Omega(n^2/(f \cdot g))$ while requiring only logarithmic time for processing each stream element.

Our skimmed-sketch algorithm is similar in spirit to the *bifocal sampling* technique of [16], but tailored to a data-stream setting. Based on the discussion in the previous section, it follows that in order to improve the storage performance of the basic sketching estimator, we need to devise a way to make the self-join sizes $f^2$ and $g^2$ small. Our *skimmed-sketch* join algorithm achieves this by *skimming* from sketches $X_F$ and $X_G$ all frequencies greater than or equal to a certain threshold $T$, and then using the *skimmed* sketches (containing only frequencies less than $T$) to estimate $f \cdot g$. Specifically, suppose that a domain value $u$ in $F$ ($G$) is *dense* if its frequency $f_u$ (resp., $g_u$) exceeds or is equal to some threshold $T$. Our skimmed-sketch algorithm estimates $f \cdot g$ in the following two steps.

1. Extract dense value frequencies in $F$ and $G$ into the frequency vectors $\hat{f}$ and $\hat{g}$, respectively. After these frequencies are *skimmed* away from the corresponding sketch, the skimmed sketches $X'_F$ and $X'_G$ correspond to the *residual* frequency vectors $f' = f - \hat{f}$ and $g' = g - \hat{g}$, respectively. Thus, each skimmed atomic sketch $X'_F[i,j] = \sum_u f'_u \xi_u^{ij}$ and $X'_G[i,j] = \sum_u g'_u \xi_u^{ij}$. Also note that, for every domain value $u$, the residual frequencies $f'_u$ and $g'_u$ in the corresponding skimmed sketches $X'_F$ and $X'_G$, are less than $T$.
2. Individually estimate the subjoins $\hat{f} \cdot \hat{g}$, $\hat{f} \cdot g'$, $f' \cdot \hat{g}$, and $f' \cdot g'$. Return the sum of the four estimates as the estimate for $f \cdot g$. For each of the individual estimates,

(a) Compute $\hat{f} \cdot \hat{g}$ accurately (that is, with zero error) using the extracted dense frequency vectors $\hat{f}$ and $\hat{g}$.

(b) Compute the remaining three estimates by invoking procedure ESTJOINSIZE with the skimmed sketches $X'_F$, $X'_G$, and newly constructed sketches $\hat{X}_F$ and $\hat{X}_G$ for frequency vectors $\hat{f}$ and $\hat{g}$, respectively. For instance, in order to estimate $f' \cdot \hat{g}$, invoke ESTJOINSIZE with sketches $X'_F$ and $\hat{X}_G$.

The maximum additive error of the final $f \cdot g$ estimate is the sum of the errors for the three estimates computed in Step 2(b) above (since $\hat{f} \cdot \hat{g}$ is computed exactly with zero error), and due to Theorem 2, is $O(\frac{\sqrt{\hat{f}^2 \cdot g'^2} + \sqrt{f'^2 \cdot \hat{g}^2} + \sqrt{f'^2 \cdot g'^2}}{\sqrt{s_1}})$. Clearly, if $F$ and $G$ contain many dense values that are much larger than $T$, then ${f'}^2 \ll f^2$ and ${g'}^2 \ll g^2$. Thus, in this case, the error for our skimmed-sketch join algorithm can be much smaller than the maximum additive error of $O(\sqrt{\frac{f^2 \cdot g^2}{s_1}})$ for the basic sketching technique (described in the previous section).

In the following section, we describe a variant of the COUNTSKETCH algorithm of [8] that, with high probability, can extract from a stream all frequencies greater than $T = O(n/s_1)$. As a result, in the worst case, ${f'}^2$ and ${g'}^2$ can be at most $n \cdot T = O(n^2/s_1)$ (which happens when there are $n/T$ values with frequency $T$). Thus, in the worst case, the maximum additive error in the estimate computed by skimming dense frequencies is $O(\sqrt{\frac{n^2 \cdot (n^2/s_1)}{s_1}}) = O(n^2/s_1)$. It follows that for a desired level of accuracy $\epsilon$, the space $s_1$ required in the worst case, becomes $O(n^2/(\epsilon \cdot (f \cdot g)))$, which is the square root of the space required by the basic sketching technique, and matches the lower bound achievable by any join size estimation algorithm [4].

*Example 1.* Consider a streaming scenario where $\mathcal{D} = \{1, 2, 3, 4\}$, and frequency vectors for streams $F$ and $G$ are given by: $f = [50, 50, 10, 5]$ and $g = [50, 5, 10, 50]$. The join size is $f \cdot g = 3100$, and self-join sizes are $f^2 = g^2 \approx 5000$. Thus, with the basic sketching algorithm, given space $s_1$, the maximum additive error is $4\sqrt{(5000)^2/s_1} = 20000/\sqrt{s_1}$. Or alternately, for a given relative error $\epsilon$, the space $s_1$ required is $(20000/(\epsilon \cdot 3100))^2 \approx 42/\epsilon^2$.

Now suppose we could extract from sketches $X_F$ and $X_G$ all frequencies greater than or equal to a threshold $T = 10$. Let the extracted frequency vectors for the dense domain values be $\hat{f} = [40, 40, 0, 0]$ and $\hat{g} = [40, 0, 0, 40]$. In the skimmed sketches (after the dense values are extracted), the residual frequency vectors $f' = [10, 10, 10, 5]$ and $g' = [10, 5, 10, 10]$. Note that $\hat{f}^2 = \hat{g}^2 = 3200$, ${f'}^2 = {g'}^2 \approx 300$. Thus, the maximum additive errors in the estimates for $\hat{f} \cdot \hat{g}$, $f' \cdot \hat{g}$, $\hat{f} \cdot g'$ and $f' \cdot g'$ are 0, $4\sqrt{(3200 \cdot 300)/s_1} \approx 4000/\sqrt{s_1}$, $4\sqrt{(300 \cdot 3200)/s_1} \approx 4000/\sqrt{s_1}$ and $4\sqrt{(300 \cdot 300)/s_1} = 1200/\sqrt{s_1}$, respectively. Thus, the total maximum additive error due to our skimmed sketch technique is $\approx 9200/\sqrt{s_1}$. Or alternately, for a given relative error $\epsilon$, the space $s_1$ required is $(9200/(\epsilon \cdot 3100))^2 = 9/\epsilon^2$, which is smaller than the memory needed for the basic sketching algorithm by more than a factor of 4. $\square$

## 4 Join Size Estimation Using Skimmed Sketches

We are now ready to present our skimmed-sketch algorithm for tackling the join size estimation problem in a streaming environment. To the best of our knowledge, ours is the first known technique to achieve the space lower bound[2] of $\Omega(n^2/(f \cdot g))$ while requiring only logarithmic time for processing each stream element. The key to achieving the low element handling times is the hash sketch data structure, which we describe in Section 4.1. While hash sketches have been used before to solve a variety of data-stream computation problems (e.g., top-$k$ frequency estimation [8]), we are not aware of any previous work that uses them to estimate join sizes. In Section 4.2, we first show how hash sketches can be used to extract dense frequency values from a stream, and then in Section 4.3, we present the details of our skimmed-sketch join algorithm with an analysis of its space requirements and error guarantees.

### 4.1 Hash Sketch Data Structure

The hash sketch data structure was first introduced in [8] for estimating the top-$k$ frequency values in a stream $F$. It consists of an array $H$ of $s_2$ hash tables, each with $s_1$ buckets. Each hash bucket contains a single counter for the elements that hash into the bucket. Thus, $H$ can be viewed as a two-dimensional array of counters, with $H[p,q]$ representing the counter in bucket $q$ of hash table $p$. Associated with each hash table $p$, is a pair-wise independent hash function $h_p$ that maps incoming stream elements uniformly over the range of buckets $\{1,\ldots,s_1\}$; that is, $h_p : \{1,\ldots,m\} \rightarrow \{1,\ldots,s_1\}$. For each hash table $p$, we also maintain a family of four-wise independent variables $\{\xi_u^p : u = 1,\ldots,m\}$.

Initially, all counters $H[p,q]$ of the hash sketch $H$ are 0. Now, for each element in stream $F$, with value say $u$, we perform the following action for each hash table $p$: $H[p,q] = H[p,q] + \xi_u^p$, where $q = h_p(u)$. (If the element specifies to delete value $u$ from $F$, then we simply subtract $\xi_u$ from $H[p,q]$). Thus, since there are $s_2$ hash tables, the time to process each stream element is $O(s_2)$ – essentially, this is the time to update a single counter (for the bucket that the element value maps to) in each hash table. Later in Section 4.3, we show that it is possible to obtain strong probabilistic error guarantees for the join size estimate as long as $s_2 = O(\log m)$. Thus, maintaining the hash sketch data structure for a stream requires only logarithmic time per stream element.

Note that each counter $H[p,q]$ is essentially an *atomic sketch* constructed over the stream elements that map to bucket $q$ of hash table $p$.

### 4.2 Estimating Dense Frequencies

In [8], the authors propose the COUNTSKETCH algorithm for estimating the top-$k$ frequencies in a stream $F$. In this section, we show how the COUNTSKETCH algorithm can be adapted to extract, with high probability, all dense frequencies greater than or equal to a threshold $T$. The COUNTSKETCH algorithm maintains a hash sketch structure $H$ over the streaming values in $F$. The key idea here is that by randomly distributing

[2] Ignoring logarithmic terms since these are generally small.

```
Procedure SkimDense($H$)
Input: Hash sketch $H$ for stream $F$.
Output: Skimmed sketch and frequency estimates $\hat{f}$ for dense values.
begin
1.  for every domain value $u \in \mathcal{D}$ do  $\hat{f}_u := 0$;
2.  $E := \phi$;  $T' := \Theta(n/s_1)$;
3.  for every domain value $u \in \mathcal{D}$ do  {
4.      for each hash table $p$ do  {  $q$ := $h_p(u)$;  $\hat{f}^p_u$ := $H[p,q] \cdot \xi^p_u$;  }
5.      Est($u$) := median{$\hat{f}^1_u, \ldots, \hat{f}^{s_2}_u$};
6.      if (Est($u$) $\geq 2T'$) then { $\hat{f}_u$ := Est($u$);  $E$ := $E \cup \{u\}$;  }
7.  }
8.  for every domain value $u$ such that $\hat{f}_u > 0$ do
9.      for each hash table $p$ do  { $q$ := $h_p(u)$;  $H[p,q]$ := $H[p,q] - (\hat{f}_u \cdot \xi^p_u)$;  }
10. return  $(H, \hat{f}, E)$;
end
```

**Fig. 3.** Variant of CountSketch Algorithm [8] for Skimming Dense Frequencies.

domain values across the $s_1$ buckets, the hash functions $h_p$ help to separate the dense domain values. As a result, the self-join sizes (of the stream projections) within each bucket are much smaller, and the dense domain values $u$ spread across the buckets of a hash table $p$ can be estimated fairly accurately (and with constant probability) by computing the product $H[p,q] \cdot \xi^p_u$, where $q = h_p(u)$. The probability can then be boosted to $1-\delta$ by selecting the median of the $s_2 = O(\log(m/\delta))$ different frequency estimates for $u$, one per table. Procedure SkimDense, depicted in Figure 3, extracts into vector $\hat{f}$, all dense frequencies $f_u \geq T = 3T'$, where $T' = \Theta(n/s_1)$. In order to show this, we first present the following adaptation of a result from [8].

**Theorem 3 ([8]).** Let $s_2 = O(\log(m/\delta))$, and $T' = \Theta(n/s_1)$. Then, for every domain value $u$, procedure SkimDense computes an estimate Est($u$) of $f_u$ (in Step 5) with an additive error of at most $T'$ with probability at least $1-\delta$ (i.e., $P[|\text{Est}(u) - f_u| \leq T'] \geq 1-\delta$) while using only $O(s_1 \cdot s_2 \cdot (\log(m) + \log(n)))$ bits of memory. □

Based on the above property of estimates Est($u$), it is easy to show that, in Step 6, procedure SkimDense extracts (with high probability) into $\hat{f}$ all frequencies $f_u \geq T$, where $T = 3T'$. Furthermore, the residual element frequencies can be upper-bounded as shown in the following theorem (the proof can be found in [17]).

**Theorem 4.** Let $s_2 = O(\log(m/\delta))$, $T' = \Theta(n/s_1)$ and $T = 3T'$. Then, with probability at least $1-\delta$, procedure SkimDense returns a frequency vector $\hat{f}$ such that for every domain value $u$, (1) $|\hat{f}_u - f_u| \leq T$ and (2) $|\hat{f}_u - f_u| \leq f_u$. □

Note that Point (1) in Theorem 4 ensures that the residual frequency $|\hat{f}_u - f_u|$ does not exceed $T$ for all domain values, while Point (2) ensures that the residual frequency $|\hat{f}_u - f_u|$ is no larger than the original frequency $f_u$. Also, observe that, in Steps 8–9, procedure SkimDense skims the dense frequencies from the $s_2$ hash tables, and returns (in addition to the dense frequency vector $\hat{f}$) the final skimmed hash sketch containing only the residual frequencies.

The runtime complexity of procedure SKIMDENSE is $O(m)$ since it examines every domain value $u$. This is clearly a serious problem when domain sizes are large (e.g., 64-bit IP addresses). Fortunately, it is possible to reduce the execution time of procedure SKIMDENSE to $O(s_1 \log m)$ using the concept of *dyadic intervals* as suggested in [9]. Consider a hierarchical organization of domain values $\mathcal{D} = \{1, \ldots, m\}$ into $\log(m)$ levels[3]. At level $l$, $0 \leq l < \log(m)$, the domain $\mathcal{D}$ is split into a sequence of $\frac{m}{2^l}$ dyadic intervals of size $2^l$, and all domain values in the $u^{th}$ dyadic interval $[(u-1) \cdot 2^l + 1, u \cdot 2^l]$ are mapped to a single value $u$ at level $l$. Thus, for level $l = 0$, each domain value $u$ is mapped to $u$ itself. For $l = 1$, the sequence of intervals is $[1, 2]$, $[3, 4], \ldots, [\frac{m}{2} \cdot 2 - 1, \frac{m}{2} \cdot 2]$, which are mapped to values $1, 2, \ldots, \frac{m}{2}$ at level 1. For $l = 2$, the sequence of intervals is $[1, 4], [5, 8], \ldots, [\frac{m}{4} \cdot 4 - 3, \frac{m}{4} \cdot 4]$, which are mapped to values $1, 2, \ldots, \frac{m}{4}$ at level 2, and so on. Let $\mathcal{D}^l = \{1, \ldots, \frac{m}{2^l}\}$ denote the set of values at level $l$, and for every $u \in \mathcal{D}^l$, let $f_u^l$ be the frequency of value $u$ at level $l$. Thus, $f_u^0 = f_u$, and in general, $f_u^l = \sum_{v=(u-1)\cdot 2^l+1}^{u \cdot 2^l} f_v$. For example, $f_2^2 = \sum_{v=5}^{8} f_v$.

The key observation we make is the following: for a level $l$, if $f_u^l$ is less than threshold value $T$, then for every domain value $v$ in the interval $[(u-1) \cdot 2^l + 1, u \cdot 2^l]$, $f_v$ must be less than $T$. Thus, our optimized SKIMDENSE procedure simply needs to maintain hash sketches at $\log(m)$ levels, where the sketch at level $l$ is constructed using values from $\mathcal{D}^l$. Then, beginning with level $\log(m) - 1$, the procedure estimates the dense frequency values at each level, and uses this to prune the set of values estimated at each successive lower level, until level 0 is reached. Specifically, if for a value $u$ at level $l > 0$, the estimate $\hat{f}_u^l$ is $< 2T'$, then we know that $f_u^l < T = 3T'$ (due to Theorem 3), and thus, we can prune the entire interval of values $[(u-1) \cdot 2^l + 1, u \cdot 2^l]$ since these cannot be dense. Only if $\hat{f}_u^l \geq 2T'$, do we recursively check values $2u - 1$ and $2u$ at level $l - 1$ that correspond to the two sub-intervals $[(2u-2) \cdot 2^{l-1} + 1, (2u-1) \cdot 2^{l-1}]$ and $[(2u-1) \cdot 2^{l-1} + 1, 2u \cdot 2^{l-1}]$ contained within the interval for value $u$ at level $l$. Thus, since at each level, there can be at most $O(n/T')$ values with frequency $T'$ or higher, we obtain that the worst-case time complexity of our optimized SKIMDENSE algorithm is $O(s_1 \log m)$.

### 4.3 Join Size Estimation Using Skimmed Sketches

We now describe our skimmed-sketch algorithm for computing $f \cdot g = \sum_u f_u \cdot g_u$. Let $H_F$ and $H_G$ be the hash sketches for streams $F$ and $G$, respectively. (Sketches $H_F$ and $H_G$ use identical hash functions $h_p$). The key idea of our algorithm is to first skim all the dense frequencies from sketches $H_F$ and $H_G$ using SKIMDENSE, and then use the skimmed sketches (containing no dense frequencies) to estimate $f \cdot g$. Skimming enables our algorithm to estimate the join size more accurately than the basic sketching algorithm of [4] that uses sketches $X_F$ and $X_G$ directly (without skimming). As already discussed in Section 3, the reason for this is that skimming causes every residual frequency in the skimmed sketches to drop below $T$ (see Theorem 4), and thus

[3] For simplicity of exposition, we assume that $m$ is a power of 2.

```
Procedure EstSkimJoinSize($H_F, H_G$)
Input: $H_F$ and $H_G$ are the hash sketches for streams $F$ and $G$.
Output: Estimate of join size.
begin
1. $(H'_F, \hat{f}, E_F)$ := SkimDense($H_F$); $(H'_G, \hat{g}, E_G)$ := SkimDense($H_G$);
2. $\hat{J}_{d,d}$ := $\hat{f} \cdot \hat{g}$; $\hat{J}_{ds}$ := EstSubJoinSize($\hat{f}, H'_G$); $\hat{J}_{sd}$ := EstSubJoinSize($\hat{g}, H'_F$);
3. for $p = 1$ to $s_2$ do {
4.     $\hat{J}^p_{ss}$ := 0;
5.     for $q = 1$ to $s_1$ do $\hat{J}^p_{ss}$ := $\hat{J}^p_{ss}$ + $H'_F[p,q] \cdot H'_G[p,q]$;
6. }
7. $\hat{J}_{ss}$ := median $\{\hat{J}^1_{ss}, \ldots, \hat{J}^{s_2}_{ss}\}$;
8. $\hat{J}$ := $\hat{J}_{dd} + \hat{J}_{ds} + \hat{J}_{sd} + \hat{J}_{ss}$;
9. return $\hat{J}$;
end

Procedure EstSubJoinSize($\hat{v}, H'$)
Input: Frequency vector $\hat{v}$ of dense frequencies and hash sketch $H'$.
Output: Estimate of subjoin size.
begin
1. for $p = 1$ to $s_2$ do {
2.     $\hat{J}^p$ := 0;
3.     for each domain value $u$ s.t. $\hat{v}_u > 0$ do { $q$ := $h_p(u)$; $\hat{J}^p$ := $\hat{J}^p$ + $H'[p,q] \cdot (\hat{v}_u \cdot \xi^p_u)$; }
4. }
5. return median$\{\hat{J}^1, \ldots, \hat{J}^{s_2}\}$;
end
```

**Fig. 4.** Skimmed-sketch Algorithm to Estimate Join Size.

the self-join sizes of the residual frequency vectors in the skimmed sketches become significantly smaller. Consequently, the join size estimate computed using skimmed sketches can potentially be more accurate (because of the dependence of the maximum additive error on self-join sizes), and the space requirements of our skimmed-sketch algorithm can be shown to match the lower bound of $\Omega(n^2/(f \cdot g))$ for the join-size estimation problem.

**Skimmed-Sketch Algorithm** Our skimmed-sketch algorithm for estimating the join size of streams $F$ and $G$ from their hash sketches $H_F$ and $H_G$ is described in Figure 4. Procedure EstSkimJoinSize begins by extracting into vectors $\hat{f}$ and $\hat{g}$, all frequencies in $f$ and $g$ (respectively) that exceed the threshold $T = \Theta(n/s_1)$. For each domain value $u$, let $f'_u = f_u - \hat{f}_u$ and $g'_u = g_u - \hat{g}_u$ be the residual frequencies for $u$ in the skimmed sketches $H'_F$ and $H'_G$ (returned by SkimDense). We will refer to $f'_u$ as the sparse component, and to $\hat{f}_u$ as the dense component of the frequency for value $u$.

The first observation we make is that the join size $J = f \cdot g$ can be expressed entirely in terms of the sparse and dense frequency components. Thus, if $J_{dd} = \hat{f} \cdot \hat{g}$, $J_{ds} = \hat{f} \cdot g'$, $J_{sd} = f' \cdot \hat{g}$, and $J_{ss} = f' \cdot g'$, then $J = J_{dd} + J_{ds} + J_{sd} + J_{ss}$.

Consequently, our skimmed-sketch algorithm estimates the join size $\hat{J}$ by summing the individual estimates $\hat{J}_{dd}$, $\hat{J}_{ds}$, $\hat{J}_{sd}$ and $\hat{J}_{ss}$ for the four terms $J_{dd}$, $J_{ds}$, $J_{sd}$ and $J_{ss}$, respectively.

The second observation is that the subjoin size $J_{dd}$ between the dense frequency components can be computed accurately (that is, with zero error) since $\hat{f}$ and $\hat{g}$ are known exactly. Thus, sketches are only needed to compute subjoin sizes for the cases when one of the components is sparse. Let us consider the problem of estimating the subjoin size $J_{ds} = \hat{f} \cdot g'$. For each domain value $u$ that is non-zero in $\hat{f}$, an estimate for the quantity $\hat{f}_u \cdot g'_u$ can be generated from each hash table $p$ by multiplying $(\hat{f}_u \cdot \xi_u^p)$ with $H'_G[p, q]$, where $q = h_p(u)$. Thus, by summing these individual estimates for hash table $p$, we can obtain an estimate $\hat{J}^p_{ds}$ for $J_{ds}$ from hash table $p$. Finally, we can boost the confidence of the final estimate $\hat{J}_{ds}$ by selecting it to be the median of the set of estimates $\{\hat{J}^1_{ds}, \ldots, \hat{J}^{s_2}_{ds}\}$. Estimating the subjoin size $J_{sd} = f' \cdot \hat{g}$ is completely symmetric; see the pseudo-code for procedure ESTSUBJOINSIZE in Figure 4. To estimate the subjoin size $J_{ss} = f' \cdot g'$ (Steps 3–7 of procedure ESTSKIMJOINSIZE), we again generate estimates $\hat{J}^p_{ss}$ for each hash table $p$, and then select the median of the estimates to boost confidence. Since the $p^{th}$ hash tables in the two hash sketches $H_F$ and $H_G$ employ the same hash function $h_p$, the domain values that map to a bucket $q$ in each of the two hash tables are identical. Thus, estimate $\hat{J}^p_{ss}$ for each hash table $p$ can be generated by simply summing $H'_F[p, q] \cdot H'_G[p, q]$ for all the buckets $q$ of hash table $p$.

**Analysis** We now give a sketch of the analysis for the accuracy of the join size estimate $\hat{J}$ returned by procedure ESTSKIMJOINSIZE. First, observe that on expectation, $\hat{J} = J$. This is because $\hat{J}_{dd} = J_{dd}$, and for all other $i, j$, $E[\hat{J}_{ij}] = J_{ij}$ (shown in [4]). Thus, $E[\hat{J}] = J_{dd} + J_{ds} + J_{sd} + J_{ss} = J$. In the following, we show that, with high probability, the additive error in each of the estimates $\hat{J}_{ij}$ (and thus, also the final estimate $\hat{J}$) is at most $\Theta((n^2/s_1)(\log n)^{1/2})$. Intuitively, the reason for this is that these errors depend on hash bucket self-join sizes, and since every residual frequency $f'_u$ in $H'_F$ and $H'_G$ is at most $T = \Theta(n/s_1)$, each bucket self-join size is proportional to $\Theta(n^2/s_1^2)$ with high probability. Due to space constraints, the detailed proofs have been omitted – they can be found in the full version of this paper [17].

**Lemma 1.** Let $s_2 = O(\log(m/\delta))$. Then the estimate $\hat{J}_{sd}$ computed by ESTSKIMJOINSIZE satisfies: $P[|\hat{J}_{sd} - J_{sd}| \leq \Theta((n^2/s_1)(\log n)^{1/4})] \geq 1 - \Theta(\delta)$. □

**Lemma 2.** Let $s_2 = O(\log(m/\delta))$. Then the estimate $\hat{J}_{ss}$ computed by ESTSKIMJOINSIZE satisfies: $P[|\hat{J}_{ss} - J_{ss}| \leq \Theta((n^2/s_1^{1.5})(\log n)^{1/2})] \geq 1 - \Theta(\delta)$. □

Note that a result similar to that in Lemma 1 above can also be shown for $\hat{J}_{ds}$ [17]. Using the above lemmas, we are now ready to prove the analytical bounds on worst-case additive error and space requirements for our skimmed-sketch algorithm.

**Theorem 5.** Let $s_2 = O(\log(m/\delta))$. Then the estimate $\hat{J}$ computed by ESTSKIMJOINSIZE satisfies: $P[|\hat{J} - (f \cdot g)| \leq \Theta((n^2/s_1)(\log n)^{1/2})] \geq 1 - \delta$. This implies that ESTSKIMJOINSIZE estimates $f \cdot g$ with a relative error of at most $\epsilon$ with probability at least $1 - \delta$ (i.e., $P[|\hat{J} - (f \cdot g)| \leq \epsilon \cdot (f \cdot g)] \geq 1 - \delta$) while using only $O\left(\frac{n^2 \cdot \log(m/\delta) \cdot (\log n)^{1/2}}{\epsilon \cdot (f \cdot g)}(\log(m) + \log(n))\right)$ bits of memory (in the worst case). □

*Proof.* Due to Lemmas 1 and 2, it follows that with probability at least $1-\delta$, the total additive error in the estimates $\hat{J}_{ds}$, $\hat{J}_{sd}$ and $\hat{J}_{ss}$ is at most $\Theta((n^2/s_1)(\log n)^{1/2})$. Thus, since $\hat{J} = \hat{J}_{dd} + \hat{J}_{ds} + \hat{J}_{sd} + \hat{J}_{ss}$, and the error in estimate $\hat{J}_{dd}$ is 0, the statement of the theorem follows. □

Thus, ignoring the logarithmic terms since these will generally be small, we obtain that in the worst case, our skimmed-sketch join algorithm requires approximately $O(\frac{n^2}{\epsilon\cdot(f\cdot g)})$ amount of space, which is equal to the lower bound achievable by any join size estimation algorithm [4]. Also, since maintenance of the hash sketch data structure involves updating $s_2$ hash bucket counters per stream element, the processing time per element of our skimmed-sketch algorithm is $O(\log(m/\delta))$.

## 5 Experimental Study

In this section, we present the results of our experimental study in which we compare the accuracy of the join size estimates returned by our skimmed-sketch method with the basic sketching technique of [4]. Our experiments with both synthetic and real-life data sets indicate that our skimmed-sketch algorithm is an effective tool for approximating the size of the join of two streams. Even with a few kilobytes of memory, the relative error in the fi nal answer is generally less than 10%. Our experiments also show that our skimmed-sketch method provides signifi cantly more accurate estimates for join sizes compared to the the basic sketching method, the improvement in accuracy ranging from a factor of fi ve (for moderate skew in the data) to several orders of magnitude (when the skew in the frequency distribution is higher).

### 5.1 Experimental Testbed and Methodology

**Algorithms for Query Answering.** We consider two join size estimation algorithms in our performance study: the basic sketching algorithm of [4] and a variant of our skimmed-sketch technique. We do not consider histograms or random-sample data summaries since these have been shown to perform worse than sketches for queries with one or more joins [4, 5]. We allocate the same amount of memory to both sketching methods in each experiment.

**Data Sets.** We used a single real-life data set, and several synthetically generated data sets with different characteristics in our experiments.

• *Census data set (www.bls.census.gov).* This data set was taken from the Current Population Survey (CPS) data, which is a monthly survey of about 50,000 households conducted by the Bureau of the Census for the Bureau of Labor Statistics. Each month's data contains around 135,000 tuples with 361 attributes, of which we used two numeric attributes to join, in our study: weekly wage and weekly wage overtime, each with domain size 288416. In our study, we use data from the month of September 2002 containing 159,434 records[4].

• *Synthetic data sets.* The experiments involving synthetic data sets evaluate the size of the join between a Zipfi an distribution and a right-shifted Zipfi an distribution with

[4] We excluded records with missing values.

the same Zipf parameter $z$. A right-shifted Zipfian distribution with Zipf parameter $z$ and shift parameter $s$ is basically the original distribution shifted right by the shift parameter $s$. Thus, the frequency of domain values between 1 and $s$ in the shifted Zipfian distribution is identical to the frequencies in the original Zipfian distribution for domain values between $m - s + 1$ to $m$, where $m$, the domain size, is chosen to be $2^{18}$ (or 256 K). We generate 4 million elements for each stream.

In our experiments, we use the shift parameter $s$ to control the join size; a shift value of 0 causes the join to become equivalent to a self-join, while as the shift parameter is increased, the join size progressively decreases. Thus, parameter $s$ provides us with a knob to "stress-test" the accuracy of the two algorithms in a controlled manner. We expect the accuracy of both algorithms to fall as the shift parameter is increased (since relative error is inversely proportion to join size), which is a fact that is corroborated by our experiments. The interesting question then becomes: how quickly does the error performance of each algorithm degenerate?

Due to space constraints, we omit the presentation of our experimental results with the real-life Census data; they can be found in the full paper [17]. In a nutshell, our numbers with real-life data sets are qualitatively similar to our synthetic-data results, demonstrating that our skimmed-sketch technique offers roughly half the relative error of basic sketching, even though the magnitude of the errors (for both methods) is typically significantly smaller [17].

**Answer-Quality Metrics.** In our experiments, we compute the error of the join size estimate $\hat{J}$ as $\frac{|J-\hat{J}|}{\min\{J,\hat{J}\}}$, where $J$ is the actual join size. The reason we use this alternate error metric instead of the standard relative error $(|J - \hat{J}|)/J$, is that the relative error measure is biased in favor of underestimates, and penalizes overestimates more severely. For example, the relative error for a join size estimation algorithm that always returns 0 (the smallest possible underestimate of the join size), can never exceed 1. On the other hand, the relative error of overestimates can be arbitrarily large. The error metric we use remedies this problem, since by being symmetric, it penalizes underestimates and overestimates about equally. Also, in some cases when the amount of memory is low, the join size estimates $\hat{J}$ returned by the sketching algorithms are very small, and at times even negative. When this happens, we simply consider the error to be a large constant, say 10 (which is equivalent to using a sanity bound of $J/10$ for very small join size results).

We repeat each experiment between 5 and 10 times, and use the average value for the errors across the iterations as the final error in our plots. In each experiment, for a given amount of space $s$, we consider $s_1$ values between 50 and 250 (in increments of 50), and $s_2$ from 11 to 59 (in increments of 12) such that $s_1 \cdot s_2 = s$, and take the average of the results for $s_1, s_2$ pairs.

## 5.2 Experimental Results

Figures 5(a) and 5(b) depict the error for the two algorithms as the amount of available memory is increased. The Zipf parameters for the Zipfian distributions joined in Figures 5(a) and 5(b) are 1.0 and 1.5, respectively. The results for three settings of the shift parameter are plotted in the graph of Figure 5(a), namely, 100, 200 and 300. On the

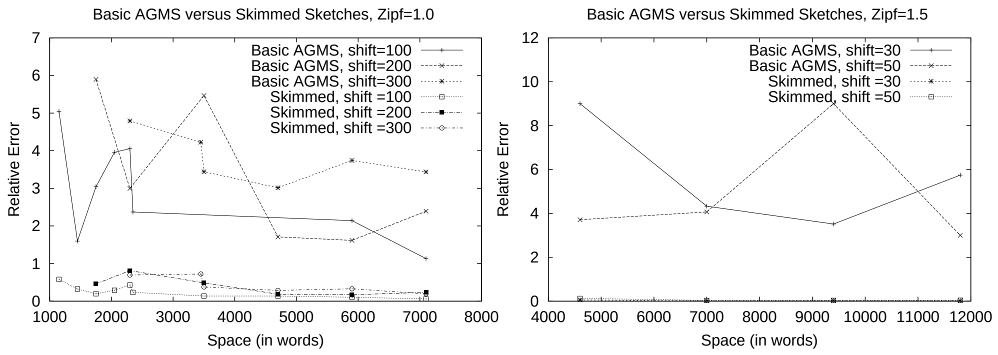

**Fig. 5.** Results for Synthetic Data Sets: (a) $z = 1.0$, (b)$z = 1.5$.

other hand, smaller shifts of 30 and 50 are considered for the higher Zipf value of 1.5 in 5(b). This is because the data is more skewed when $z = 1.5$, and thus, larger shift parameter values cause the join size to become too small.

It is interesting to observe that the error of our skimmed-sketch algorithm is almost an order of magnitude lower than the basic sketching technique for $z = 1.0$, and several orders of magnitude better when $z = 1.5$. This is because as the data becomes more skewed, the self-join sizes become large and this hurts the accuracy of the basic sketching method. Our skimmed-sketch algorithm, on the other hand, avoids this problem by eliminating from the sketches, the high frequency values. As a result, the self-join sizes of the skimmed sketches never get too big, and thus the errors for our algorithm are small (e.g., less than 10% for $z = 1$, and almost zero when $z = 1.5$). Also, note that the error typically increases with the shift parameter value since the join size is smaller for larger shifts. Finally, observe that there is much more variance in the error for the basic sketching method compared to our skimmed-sketch technique – we attribute this to the high self-join sizes with basic sketching (recall that variance is proportional to the product of the self-join sizes).

## 6 Conclusions

In this paper, we have presented the *skimmed-sketch* algorithm for estimating the join size of two streams. (Our techniques also naturally extend to complex, multi-join aggregates.) Our skimmed-sketch technique is the first comprehensive join-size estimation algorithm to provide tight error guarantees while (1) achieving the lower bound on the space required by any join-size estimation method, (2) handling general streaming updates, (3) incurring a guaranteed small (i.e., logarithmic) processing overhead per stream element, and (4) not assuming any a-priori knowledge of the data distribution. Our experimental study with real-life as well as synthetic data streams has verified the superiority of our skimmed-sketch algorithm compared to other known sketch-based methods for join-size estimation.

## References

1. Greenwald, M., Khanna, S.: "Space-efficient online computation of quantile summaries". In: Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data, Santa Barbara, California (2001)
2. Gilbert, A., Kotidis, Y., Muthukrishnan, S., Strauss, M.: "How to Summarize the Universe: Dynamic Maintenance of Quantiles". In: Proceedings of the 28th International Conference on Very Large Data Bases, Hong Kong (2002)
3. Alon, N., Matias, Y., Szegedy, M.: "The Space Complexity of Approximating the Frequency Moments". In: Proceedings of the 28th Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania (1996) 20–29
4. Alon, N., Gibbons, P.B., Matias, Y., Szegedy, M.: "Tracking Join and Self-Join Sizes in Limited Storage". In: Proceedings of the Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Philadeplphia, Pennsylvania (1999)
5. Dobra, A., Garofalakis, M., Gehrke, J., Rastogi, R.: "Processing Complex Aggregate Queries over Data Streams". In: Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, Madison, Wisconsin (2002)
6. Gibbons, P.: "Distinct Sampling for Highly-accurate Answers to Distinct Values Queries and Event Reports". In: Proceedings of the 27th International Conference on Very Large Data Bases, Roma, Italy (2001)
7. Cormode, G., Datar, M., Indyk, P., Muthukrishnan, S.: "Comparing Data Streams Using Hamming Norms". In: Proceedings of the 28th International Conference on Very Large Data Bases, Hong Kong (2002)
8. Charikar, M., Chen, K., Farach-Colton, M.: "Finding frequent items in data streams". In: Proceedings of the 29th International Colloquium on Automata Languages and Programming. (2002)
9. Cormode, G., Muthukrishnan, S.: "What's Hot and What's Not: Tracking Most Frequent Items Dynamically". In: Proceedings of the Twentysecond ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, San Diego, California (2003)
10. Manku, G., Motwani, R.: "Approximate Frequency Counts over Data Streams". In: Proceedings of the 28th International Conference on Very Large Data Bases, Hong Kong (2002)
11. Gilbert, A.C., Kotidis, Y., Muthukrishnan, S., Strauss, M.J.: "Surfing Wavelets on Streams: One-pass Summaries for Approximate Aggregate Queries". In: Proceedings of the 27th International Conference on Very Large Data Bases, Roma, Italy (2001)
12. Datar, M., Gionis, A., Indyk, P., Motwani, R.: "Maintaining Stream Statistics over Sliding Windows". In: Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, California (2002)
13. Vitter, J.: Random sampling with a reservoir. ACM Transactions on Mathematical Software **11** (1985) 37–57
14. Acharya, S., Gibbons, P.B., Poosala, V., Ramaswamy, S.: "Join Synopses for Approximate Query Answering". In: Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data, Philadelphia, Pennsylvania (1999) 275–286
15. Chakrabarti, K., Garofalakis, M., Rastogi, R., Shim, K.: "Approximate Query Processing Using Wavelets". In: Proceedings of the 26th International Conference on Very Large Data Bases, Cairo, Egypt (2000) 111–122
16. Ganguly, S., Gibbons, P., Matias, Y., Silberschatz, A.: "Bifocal Sampling for Skew-Resistant Join Size Estimation". In: Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Quebec (1996)
17. Ganguly, S., Garofalakis, M., Rastogi, R.: "Processing Data-Stream Join Aggregates Using Skimmed Sketches". Bell Labs Tech. Memorandum (2004)