A General Method for Estimating Correlated Aggregates Over a Data Stream

Srikanta Tirthapura
Dept. of Electrical and Computer Engg.
Iowa State University, Ames, IA, USA
snt@iastate.edu

David P. Woodruff
IBM Almaden Research Center
San Jose, CA, USA
dpwoodru@us.ibm.com

Abstract—On a stream of two dimensional data items (x, y) where x is an item identifier, and y is a numerical attribute, a correlated aggregate query requires us to first apply a selection predicate along the second (y) dimension, followed by an aggregation along the first (x) dimension. For selection predicates of the form $(y < c)$ or $(y > c)$, where parameter c is provided at query time, we present new streaming algorithms and lower bounds for estimating statistics of the resulting substream of elements that satisfy the predicate.

We provide the first sublinear space algorithms for a large family of statistics in this model, including frequency moments. We experimentally validate our algorithms, showing that their memory requirements are significantly smaller than existing linear storage schemes for large datasets, while simultaneously achieving fast per-record processing time.

We also study the problem when the items have weights. Allowing negative weights allows for analyzing values which occur in the symmetric difference of two datasets. We give a strong space lower bound which holds even if the algorithm is allowed up to a logarithmic number of passes over the data (before the query is presented). We complement this with a small space algorithm which uses a logarithmic number of passes.

I. Introduction

We consider the construction of a small-space summary of a massive data stream that enables the user to query for *correlated aggregates* over the stream. Consider a stream of tuples $\mathcal{S} = (x_i, y_i), i = 1 \ldots n$, where x_i is an item identifier, and y_i is a numerical attribute. A correlated aggregate query $C(\sigma, AGG, \mathcal{S})$ specifies two functions, a selection predicate σ along the y dimension, and an aggregation function AGG along the x dimension. It requires that the selection be applied first on stream $\mathcal{S}$, followed by the aggregation. More formally,

$$C(\sigma, AGG, \mathcal{S}) = AGG\{x_i | \sigma(y_i) \quad \text{is true}\}$$

Importantly, the selection predicate σ is completely specified only at query time, and is not known when the stream is being observed.

A traditional data stream summary (see, e.g., [1], [2], [3], [4]) provides an accurate answer to an aggregate query, such as frequency moments, quantiles, and heavy-hitters, over the entire data stream. In contrast, a summary for a correlated aggregate provides accurate answers to a family of queries that can be posed on the stream, where different queries in the family specify the same aggregation function, but over different subsets of the input stream. These subsets are selected by the predicate along the primary dimension. Correlated

aggregates arise naturally in analytics on multi-dimensional streaming data, and space-efficient methods for implementing such aggregates are useful in streaming analytics systems such as in the IBM System S [5].

Our summaries for correlated aggregates allow more flexible interrogation on the data stream than is possible with a traditional stream summary. For example, consider a data stream of IP flow records, such as those output by network routers equipped with Cisco's Netflow [6]. Suppose that we were interested in only two attributes per flow record, the destination address of the flow, and the size (number of bytes) of the flow. Using a summary for correlated aggregate AGG, along with a quantile summary for the y dimension (any of the well known stream quantile summaries can be used here, including [2], [7]), it is possible for a network administrator to execute the following *sequence* of queries on the stream: (1) First, the quantile summary can be queried to find the median size of a flow. (2) Next, using the summary for correlated aggregates, the administrator can query the aggregate AGG of all those flow records whose flow size was more than the median flow size. (3) If the answer to query (2) was "abnormal" in the administrator's opinion, and the administrator needed to find the properties of the very high volume flows, this can be accomplished by similarly querying for the aggregate of all those flow records whose flow size is in the 95 percent quantile or above (the 95 percent quantile can be found using the quantile summary).

Thus, a summary that can allow such queries for correlated aggregates can be used for deeper investigation of large data streams. Crucial to this is the flexibility that the parameter c for the selection predicate $(y < c)$ can be specified at query time, which allows the user to iteratively "adapt" his/her query based on the results of previous queries. For data-intensive analytics systems, such as network management systems and sensor data management, this can be a very powerful tool.

A. Contributions

In this work, we study both algorithms and lower bounds for summarizing data streams for correlated aggregate queries. We consider selection predicates of the form $(y \leq y_0)$, or $(y \geq y_0)$ where y_0 is specified at query time.

a) General Scheme for Correlated Aggregation: We present a general scheme for constructing a summary for correlated aggregation for any aggregation function that sat-

isfies a certain set of properties. For any such aggregation function, we show how to reduce the construction of a sketch for correlated aggregation to the construction of a sketch for aggregation over the entire stream. This reduction allows us to use previously known stream summaries for aggregation over streams in constructing summaries for correlated aggregation.

We use this scheme to construct small space summaries for estimating correlated frequency moments over a data stream. For $k > 0$, the k-th frequency moment of a stream of identifiers, each assumed to be an integer from $\{1, \ldots, m\}$, is defined as $F_k = \sum_{i=1}^{m} f_i^k$, where f_i is the number of occurrences of the i-th item. The estimation of frequency moments over a data stream has been the subject of much study over the past decade or so, starting with the work of Alon *et al.* [1]. See, e.g., the references in [8]. Our algorithms are the first small-space (in fact, the first sub-linear in m space) algorithms for estimating the correlated frequency moments with provable guarantees on the relative error. Our memory requirements are optimal up to small factors, namely, factors that are logarithmic in m and the error probability δ, and factors polynomial in the relative error parameter ϵ.

Our technique for general correlated aggregation builds on the algorithm due to Busch and Tirthapura [9] for the correlated estimation of the basic count. While the basic data structure used in [9] is an exact counter, the data structure used in our algorithm is a "sketch" (such as the sketch for F_2 from [1]) which can accurately estimate an aggregate on a stream, and provide a (possibly) probabilistic guarantee on the correctness. We analyze the error due to the randomized sketches, and the combination of different sketches. A central new issue is that of dependency between random variables.

We also present the first space-efficient algorithms for correlated aggregates of the number of distinct elements (F_0), and other aggregates related to frequency moments, such as the F_k-heavy hitters and rarity. We give a technique for achieving low amortized update time. For correlated F_2, the update time is $\widetilde{O}(\log n)$ per stream update.

b) General Streaming Models and Lower Bounds: We next consider the case of streams where items can have an associated positive or negative integer weight. Allowing negative weights is useful for analyzing the symmetric difference of two datasets, since the items in the first data set can be inserted into the stream with positive weights, while the items from the second data set can be inserted into the stream with negative weights.

In this model, each stream element is a 3-tuple (x_i, y_i, z_i) where z_i specifies the weight of the item. We show that even if $z_i \in \{-1, 1\}$ for all i, then for a general class of functions that includes the frequency moments, any summary that can estimate correlated aggregates accurately and that is constructed in a single pass must use memory that is linear in the size of the stream. This is to be contrasted with the estimation of frequency moments over a stream in the non-correlated case, where it is known how to estimate these aggregates in sublinear space even in the presence of positive and negative weights [10].

We then consider the model with arbitrary positive and negative weights in which we allow multiple passes over the stream. This more general model allows the algorithm to make a small (but more than one) number of passes over the stream and store a small-space summary of what it has seen. At a later time a query is made and must be answered using only the summary. Such a setting arises if data is collected and stored from previous days for the purpose of creating a summary, but is then deleted or archived while only the summary is retained.

In this case, we show a smooth pass-space tradeoff for these problems, showing that with a logarithmic number of passes there are space-efficient algorithms for a large class of correlated aggregates even with negative weights, but with fewer passes no such space-efficient algorithms can exist.

c) Aggregation Over Asynchronous Streams: A closely related problem is that of computing aggregates over a sliding window on an asynchronous stream. In this scenario, we are given a stream of (v, t) tuples, where v is a data item, and t is the timestamp at which it was generated. Due to asynchrony in the transmission medium, it is possible that the stream elements do not arrive in order of timestamps. In other words, it is possible that $t_1 < t_2$, but (v_1, t_1) is received later than (v_2, t_2). This was not possible in the traditional definition of count-based or time-based sliding windows [11]. There is a straightforward reduction from the problem of aggregating over asynchronous streams to that of computing correlated aggregates, and vice versa [12], [9]. Hence, all of our results for correlated aggregates apply also to aggregation over a sliding window on an asynchronous stream with essentially the same space and time bounds. We thus achieve the first low-space algorithms for asynchronous streams for a wide class of statistics.

d) Experimental Results: We present experimental results showing the practical performance of our algorithms for correlated estimation of the second frequency moment (F_2) and the number of distinct elements (F_0). Our experiments show that the sketches for correlated aggregation are useful in practice, and provide significant space savings when compared with storing the entire dataset. Moreover, for a given accuracy requirement, the size of the sketch remains nearly the same, and does not increase significantly as the stream size increases. Hence, these sketches are a very effective way to summarize extremely large data sets and streams.

B. Related Work

Correlated aggregates arise naturally when forming SQL queries on data. Their use in online analytical processing (OLAP) has been investigated in the work of Chatziantoniou *et al.* [13], [14], [15]. Single pass computation of summaries for correlated aggregates on streams was first considered in the work of Gehrke *et al.* [16], who provided heuristics for approximating the correlated sum of elements, but these did not come with a provable bound on the quality of the answers. Subsequent work of Ananthakrishna *et al.* [17] presented a summary that allowed the estimation of the correlated sum

over streams with a provable bound on the *additive* error of the estimates.

Xu *et al.* [12] considered the computation of sum and median over a sliding window of an asynchronous stream, and presented summaries that allowed the estimation of aggregates within a small *relative* error with high probability. The space taken by the summary for sum in [12] is $O(\frac{\log^2 N}{\epsilon^2} \log(1/\delta))$ bits, where ϵ is the relative error, and δ is the failure probability. The above results also yield summaries for correlated sum with relative error guarantees, and using the same space. Cormode et al. [18] presented algorithms for correlated quantiles and heavy hitters (frequent elements). The space complexity of these was subsequently improved by Chan et al. [19].

Significant prior work (for example, [11], [20], [21]) has focused on sketching a synchronous stream (where elements arrive in order of timestamps) to answer aggregate queries over a sliding window. The problem of aggregation over a sliding window on a synchronous stream is a special case of the asynchronous streams – note that a sliding window over a synchronous stream is always a suffix of the entire stream.

Roadmap: In Section II, we present the algorithm for general correlated aggregates, and for the frequency moments. Deletions, Lower bounds, and multipass algorithms are considered in Section III, and experimental results in Section IV. Due to space constraints, some of the proofs are presented in the appendix.

II. CORRELATED ESTIMATION OF A CLASS OF STATISTICS

We consider the estimation of correlated aggregates for any aggregation function that satisfies a set of properties. Consider an aggregation function f that takes as input a multi-set of real numbers R and returns a real number $f(R)$. In the following, we use the term "set of real numbers" to mean a "multi-set of real numbers". Also, union of sets implies a multi-set union, when the context is clear.

For any set of tuples of real numbers $T = \{(x_i, y_i) | 1 \leq i \leq n\}$ and real number y, let $f(T, c)$ denote the correlated aggregate $f(\{x_i | ((x_i, y_i) \in T) \wedge (y_i \leq c)\})$. For any function f satisfying the following properties, we show a reduction from the problem of space-efficient estimation of correlated aggregate $f(T, c)$ to the problem of space-efficient estimation of f in an uncorrelated manner on the entire set R.

We use the following definition of an (ϵ, δ) estimator.

Definition 1: Given parameters ϵ, δ, where $0 < \epsilon < 1$, and $0 < \delta < 1$, an (ϵ, δ) estimator for a number Y is a random variable X such that with probability at least $1 - \delta$, the following is true.

$$(1 - \epsilon)Y \leq X \leq (1 + \epsilon)Y$$

In the following description, we use the term "sketching function" to denote a compression of the input set with certain properties. More precisely, we say that f has a sketching function $sk_f(\upsilon, \gamma, R)$ if

   a Using $sk_f(\upsilon, \gamma, R)$ it is possible to get an (υ, γ)-estimator of $f(R)$.

   b For two sets R_1 and R_2, given $sk(\upsilon, \gamma, R_1)$ and $sk(\upsilon, \gamma, R_2)$, it is possible to compute $sk(\upsilon, \gamma, R_1 \cup R_2)$.

Many functions f have sketching functions. For instance, the second frequency moment F_2 has a sketch due to Alon, Matias, and Szegedy [1], while the k-th frequency moment F_k has a sketch due to Indyk and Woodruff [10]. In these two examples, the sketching function is obtained by taking random linear combinations of the input.

We require the following conditions from f in order to construct a sketch for estimating correlated aggregates. These conditions intuitively correspond to "smoothness" conditions of the function f, bounding how much f can change when new elements are inserted or deleted from the input multi-set. Informally, the less the function value is sensitive to small changes, the easier it is to apply to estimating correlated aggregates.

   I. $f(R)$ is bounded by a polynomial in $|R|$.
   II. For sets R_1 and R_2, $f(R_1 \cup R_2) \geq f(R_1) + f(R_2)$
   III. There exists a function $c_1^f(\cdot)$ such that for sets $R_1, \ldots R_j$, if $f(R_i) \leq \alpha$ for all $i = 1 \ldots j$, then: $f(\cup_{i=1}^{j} R_i) \leq c_1^f(j) \cdot \alpha$.
   IV. For $\epsilon < 1$, there exists a function $c_2^f(\epsilon)$ with the following properties. For two sets A and B such that $B \subseteq A$, if $f(B) \leq c_2^f(\epsilon) \cdot f(A)$, then $f(A - B) \geq (1 - \epsilon)f(A)$.
   V. f has a sketching function $sk_f(\gamma, \upsilon, R)$ where $\gamma \in (0, 1)$ and $\upsilon \in (0, 1)$.

For any function f with a sketch sk_f with the above properties, we show how to construct a sketch $sk_f^{cor}(\epsilon, \delta, T)$ for estimating the correlated aggregate $f(T, c)$ with the following properties:

   A. Using $sk_f^{cor}(\epsilon, \delta, T)$, it is possible to get an (ϵ, δ)-estimator of $f(T, c)$ for any real $c > 0$.
   B. For any tuple (x, y), using $sk_f^{cor}(\gamma, \epsilon, T)$, it is possible to construct $sk_f^{cor}(\gamma, \epsilon, T \cup \{(x, y)\})$.

A. Algorithm Description

Let f_{max} denote an upper bound on the vaue of $f(\cdot, \cdot)$ over all input streams that we consider. The algorithm uses a set of levels $\ell = 0, 1, 2, \ldots, \ell_{max}$, where ℓ_{max} is such that $2^{\ell_{max}} > f_{max}$ for any input stream T and real number c. From Property I, it follows that ℓ_{max} is logarithmic in the stream size.

Choose parameters α, γ, υ as follows:

$$\alpha = \frac{64 c_1^f(\log y_{max})}{c_2^f(\epsilon/2)}, \upsilon = \frac{\epsilon}{2}, \gamma = \frac{\delta}{4 y_{max}(\ell_{max} + 1)},$$

where y_{max} is the largest possible y value.

Without loss of generality, assume that y_{max} is of the form $2^\beta - 1$ for some integer β. The dyadic intervals within $[0, y_{max}]$ are defined inductively as follows. (1) $[0, y_{max}]$ is a dyadic interval (2) If $[a, b]$ is a dyadic interval and $a \neq b$, then $[a, (a + b - 1)/2]$ and $[(a + b + 1)/2, b]$ are also dyadic intervals.

Within each level ℓ, from 0 to ℓ_{max}, there is a "bucket" for each dyadic interval within $[0, y_{max}]$. Thus, there are $2 y_{max} - 1$ buckets in a single level. Each bucket b is a

triple $\langle k(b), l(b), r(b)\rangle$, where $[l(b), r(b)]$ is a dyadic interval that corresponds to a range of y values that this bucket is responsible for, and $k(b)$ is defined below.

When a stream element (x, y) arrives, it is inserted into each level $\ell = 0, \ldots, \ell_{max}$. Within level ℓ, it is inserted into exactly one bucket, as described in Algorithm 2. For a bucket b in level ℓ, let $S(b)$ denote the (multi-)set of stream items that were inserted into b. Then, $k(b) = sk_f(v, \gamma, S(b))$ is a sketch of $S(b)$.

Within each level, no more than α of the $2y_{max} - 1$ buckets are actually stored. In the algorithm, S_ℓ denotes the buckets that are stored in level ℓ. The level S_0 is a special level which just consists of singletons. Among the buckets that are not stored, there are two types of buckets, those that were discarded in Algorithm 2 (see the "Check for overflow" comment), and those that were never used by the algorithm. We call the above three types of buckets 'stored", "discarded", and "empty" respectively. Note that $S(b)$ is defined for each of these three types of buckets (if b is an empty bucket, then $S(b)$ is defined as the null set ϕ).

The buckets in S_ℓ are organized into a tree, induced by the relation between the dyadic intervals that these buckets correspond to.

The initialization for the algorithm for a general function is described in Algorithm 1. The update and query processing are described in Algorithms 2 and 3 respectively.

Algorithm 1: General Function: Initialization

1   $S_0 \leftarrow$ null set ϕ; $Y_0 \leftarrow \infty$;
2   **for** ℓ *from* 1 *to* ℓ_{max} **do**
3   S_ℓ is a set with a single element
     $\langle sk_f(\cdot, \cdot, \phi), 0, y_{max}\rangle$;
4     $Y_\ell \leftarrow \infty$;
5   **end**

Theorem 1 (Space Complexity): The space complexity of the sketch for correlated estimation is

$$O\left(\frac{c_1^f(\log y_{max}) \cdot (\log f_{max})}{c_2^f\left(\frac{\epsilon}{2}\right)} \cdot len\right),$$

where

$$len = \left| sk_f\left(\frac{\epsilon}{2}, \frac{\delta}{4y_{max}(2 + \log f_{max})}, S\right)\right|$$

is the number of bits needed to store the sketch.

Proof: There are no more than $2 + \log f_{max}$ levels and in each level ℓ, S_ℓ stores no more than α buckets. Each bucket b contains $sk_f(v, \gamma, S(b))$. The space complexity is $\alpha(2 + \log f_{max})$ times the space complexity of sketch sk_f. Here we assume that the space taken by sk_f is larger than the space required to store $l(b)$ and $r(b)$. ∎

B. Algorithm Correctness

Let S denote the stream of tuples observed up to a given point. Suppose the required correlated aggregate is $f(S, c)$.

Algorithm 2: When an element (x, y) arrives

1   **if** *There is a bucket b in S_0 such that $y \in span(b)$* **then**
2   Update $k(b)$ by inserting x.
3   **end**
4   **else**
5   Initialize a bucket $b = \langle sk_f(v, \gamma, \{x\}), y, y\rangle$, and insert b into S_0 ;
6   **if** $|S_0| > \alpha$ **then**
7   Discard the bucket $b \in S_0$ with the largest value of $l(b)$, say b^*, and update $Y_0 \leftarrow \min\{Y_0, l(b^*)\}$.
8   **end**
9   **end**
   `// levels i > 0`
10   **for** i *from* 1 *to* ℓ_{max} **do**
11   **if** $Y_i \leq y$ **then**
12   return
13   **end**
14   Let b be the bucket in S_i such that b is a leaf and $y \in span(b)$;
15   **if** b *is open* **then**
16   Insert x into $k(b)$;
17   **if** $(est(k(b)) \geq 2^{i+1})$ *and* $(l(b) \neq r(b))$ **then**
18   close bucket b
19   **end**
20   **end**
21   **else**
22   Store two buckets b_1, b_2 in S_i, where b_1 is the left child of b and b_2 is the right child of b. Initialize $k(b_1) = k(b_2) = sk_f(\nu, \gamma, \phi)$.
23   If $y \in span(b_1)$ then insert x into $k(b_1)$. Otherwise insert x into $k(b_2)$;
       `/* Check for overflow      */`
24   **if** $|S_i| \geq \alpha$ **then**
25   Let b' be the bucket in S_i with the largest value of attribute $l()$.;
26   Discard b' from S_i;
27   Update $Y_i \leftarrow l(b')$.
28   **end**
29   **end**
30   **end**

Let A be the set $\{x_i | ((x_i, y_i) \in S) \wedge (y_i \leq c)\}$. We have $f(S, c) = f(A)$.

For level $\ell, 0 \leq \ell \leq \ell_{max}$, we define B_1^ℓ and B_2^ℓ as follows.

- Let B_1^ℓ denote the set of buckets b in level ℓ such that $span(b) \subseteq [0, c]$.
- Let B_2^ℓ denote the set of buckets b in level ℓ such that $span(b) \not\subset [0, c]$, but $span(b)$ has a non-empty intersection with $[0, c]$.

Note that for each level ℓ, B_1^ℓ and B_2^ℓ are uniquely determined once the query $f(S, c)$ is fixed. These do not depend on the actions of the algorithm. This is a critical property that we

---

Algorithm 3: When there is a query for $f(S, c)$

---
1 Let $\ell \in [0, \ldots, \ell_{max}]$ be the smallest level such that $Y_\ell > c$. If no such level exists, output FAIL.

2 **if** ℓ is 0 **then**

3   Answer the query using S_0 by summing over appropriate singletons

4 **end**

5 **else**

6   Let B_1^ℓ be the set of all buckets b in level ℓ such that $span(b) \subseteq [0, c]$;
     // Compose the Sketches

7   Let K be the composition of all sketches in $\{k(b) | b \in B_1^\ell\}$

8   Return $est(K)$, the estimate of $f(B_1^\ell)$ gotten using the sketch K.

9 **end**

---

use, which allows the choice of which buckets to use during estimation to be independent of the randomness in our data structures. Further, note that only a subset of B_1^ℓ and B_2^ℓ is actually stored in S_ℓ.

Consider any level $\ell, 0 \le \ell \le \ell_{max}$. For bucket b, recall that $S(b)$ denotes the set of stream items inserted into the bucket until the time of the query. For bucket $b \in S_\ell$, let $f(b)$ denote $f(S(b))$. Let $est_f(b)$ denote the estimate of $f(b)$ obtained using the sketch $k(b)$. If $S(b) = \phi$, then $f(b) = 0$ and $est_f(b) = 0$. Thus note that $f(b)$ and $est_f(b)$ are defined no matter whether b is a stored, discarded, or an empty bucket.

Further, for a set of buckets B in the same level, let $S(B) = \cup_{b \in B} S(b)$, and let $f(B) = f(S(B))$. Let $est_f(B)$ be the estimate for $f(B)$ obtained through the composition of all sketches in $\cup_{b \in B} k(b)$ (by property V, sketches can be composed with each other).

Definition 2: Bucket b is defined to be "good" if $(1 - \upsilon)f(b) \le est_f(b) \le (1 + \upsilon)f(b)$. Otherwise, b is defined to be "bad".

Let G denote the following event: *each bucket b in each level $0 \ldots \ell_{max}$ is good.*

Lemma 1:
$$\Pr[G] \ge 1 - \frac{\delta}{2}$$

Proof: For each bucket b, note that $est_f(b)$ is a (υ, γ)-estimator for $f(b)$. Thus, the probability that b is bad is no more than γ. Noting that there are less than $2y_{max}$ buckets in each level, and $\ell_{max} + 1$ levels in total, and applying a union bound, we get:

$$\Pr[\bar{G}] \le 2y_{max}(\ell_{max} + 1)\gamma = \frac{\delta}{2}$$
∎

Lemma 2: For any level ℓ, $S(B_1^\ell) \subseteq A \subseteq S(B_1^\ell \cup B_2^\ell)$

Proof: Every bucket $b \in B_1^\ell$ must satisfy $span(b) \in [0, c]$. Thus every element inserted into B_1^ℓ must belong in A. Hence $S(B_1^\ell) \subseteq A$. Each element in A has been inserted into some

bucket in level ℓ (it is possible that some of these buckets have been discarded). By the definitions of B_1^ℓ and B_2^ℓ, an element in A cannot be inserted into any bucket outside of $B_1^\ell \cup B_2^\ell$. Thus $A \subseteq S(B_1^\ell \cup B_2^\ell)$. ∎

Using Lemma 2 and Condition II on f, we get the following for any level ℓ:

$$f(B_1^\ell) \le f(A) \le f(B_1^\ell \cup B_2^\ell) \quad (1)$$

We claim that Algorithm 3 does not output FAIL in step 1.

Lemma 3: Conditioned on event G, Algorithm 3 does not output FAIL in step 1.

Proof: Consider ℓ_{max}. We claim that $Y_{\ell_{max}} > c$ if event G occurs. Observe that $Y_{\ell_{max}}$ is initialized to ∞ in Algorithm 1. Its value can only change if the root b of $S_{\ell_{max}}$ closes. For this to happen, we must have $est(k(b)) \ge 2^{\ell_{max}+1}$. But $2^{\ell_{max}+1} > 2f_{\max}$, which means that $est(k(b))$ does not provide a $(1 + \epsilon)$-approximation. This contradicts the occurrence of event G. Hence, $Y_{\ell_{max}} > c$ and so Algorithm 3 does not output FAIL in step 1. ∎

Let ℓ^* denote the level used by Algorithm 3 to answer the query $f(S, c)$.

Lemma 4: If $\ell^* \ge 1$ and G is true, then

$$f(B_2^{\ell^*}) \le c_1^f (\log y_{max}) 2^{\ell^*+2}$$

Proof: First, we note that there can be no singleton buckets in $B_2^{\ell^*}$ by definition of B_2^ℓ for a level ℓ. Thus, for each bucket $b \in B_2^{\ell^*}$, $est_f(b) \le 2^{\ell^*+1}$ Because G is true, for every bucket $b \in B_2^{\ell^*}$, b is good, so that $f(b) \le \frac{2^{\ell^*+1}}{1-\upsilon}$.

Next, note that there are no more than $\log y_{max}$ buckets in $B_2^{\ell^*}$, since there can be only one dyadic interval of a given size that intersects $[0, c]$ but is not completely contained within $[0, c]$.

From Property III. we have:

$$f(B_2^{\ell^*}) = f(\cup_{b \in B_2^{\ell^*}} S(b)) \le c_1^f (\log y_{max}) \cdot \frac{2^{\ell^*+1}}{1-\upsilon}$$

Since $\upsilon \le 1/2$, we get the desired result. ∎

Lemma 5: If $\ell^* \ge 1$ and G is true, then:

$$f(A) \ge \alpha 2^{\ell^*-4}$$

Proof: Since the algorithm used level ℓ^* for answering the query, it must be the case that there are buckets in S_{ℓ^*-1} that had an intersection with $[0, c]$ but were discarded from the data structure. It follows that there are at most $\log y_{max}$ buckets $b \in S_{\ell^*-1}$ such that $span(b) \not\subset [0, c]$. For the remaining buckets $b \in S_{\ell^*-1}$, it must be true that $span(b) \subset [0, c]$. If we view S_{ℓ^*-1} as a binary tree with α nodes, according to the ordering between the different dyadic intervals, then S_{ℓ^*-1} must have $(\alpha - 1)/2$ internal nodes.

Suppose I denoted the set of buckets in $b \in S_{\ell^*-1}$ such that b is an internal node, and $span(b) \subset [0, c]$. Thus $|I| \ge (\alpha - 1)/2 - \log y_{max}$. Since G is true, we have that for any bucket $b \in I$, $f(b) \ge \frac{2^{\ell^*-1}}{1+\upsilon}$,

Using property II repeatedly, we get:

$$f(A) \geq f(I) \geq \frac{|I| 2^{\ell^*-1}}{1+\upsilon}$$

Using $\upsilon < 1$, and for an appropriately large value of α, we have $((\alpha-1)/2 - \log y_{max}) \geq \alpha/4$. Combining the above, we get the following:

$$f(A) \geq \frac{\alpha 2^{\ell^*}}{2 \cdot 4 \cdot 2} = 2^{\ell^*-4}\alpha$$

∎

Theorem 2: When presented with a query for $f(S,c)$, let est denote the estimate returned by the algorithm. Then, with probability at least $1-\delta$:

$$(1-\epsilon)f(S,c) \leq est \leq (1+\epsilon)f(S,c)$$

Proof: If $\ell^* = 0$, then all elements $(x,y) \in S$ such that $y \leq c$ are stored in S_0. In this case, the theorem follows by the definition of event G and Lemma 1.

Otherwise, we have $est = est_f(B_1^{\ell^*})$, and $f(S,c) = f(A)$. First, note that in level ℓ^*, none of the buckets in $B_1^{\ell^*}$ have been discarded. Thus each bucket $b \in B_1^{\ell^*}$ is either empty or is stored. Thus, it is possible to execute line 7 in Algorithm 3 correctly to construct a sketch of $S(B_1^{\ell^*})$. From property (b) of sketching functions, we get a sketch $sk(\upsilon, \gamma, S(B_1^{\ell^*}))$.

Let $\mathcal{E}_1$ denote the event $(1-\upsilon)f(B_1^{\ell^*}) \leq est_f(B_1^{\ell^*}) \leq (1+\upsilon)f(B_1^{\ell^*})$

Thus, we have:

$$\Pr[\mathcal{E}_1] \geq 1 - \gamma \qquad (2)$$

In the following, we condition on both $\mathcal{E}_1$ and G occurring. From Equation 1, we have:

$$f(A) \leq f(B_1^{\ell^*} \cup B_2^{\ell^*}) \qquad (3)$$

From Lemmas 4 and 5:

$$\frac{f(B_2^{\ell^*})}{f(A)} \leq \frac{c_1^f(\log y_{max})2^{\ell^*+2}}{\alpha 2^{\ell^*-4}} \leq \frac{c_1^f(\log y_{max})2^6}{\alpha} \leq c_2^f\left(\frac{\epsilon}{2}\right) \qquad (4)$$

where we have substituted the value of α.

Since $(A - B_1^{\ell^*}) \subseteq B_2^{\ell^*}$, we have the following:

$$\frac{f(A - B_1^{\ell^*})}{f(A)} \leq c_2^f\left(\frac{\epsilon}{2}\right) \qquad (5)$$

Using Property IV, we get the following:

$$f(B_1^{\ell^*}) = f(A - (A - B_1^{\ell^*})) \geq \left(1 - \frac{\epsilon}{2}\right)f(A) \qquad (6)$$

Conditioned on $\mathcal{E}_1$ and G both being true, we have:

$$est_f(B_1^{\ell^*}) \geq (1-\epsilon/2)(1-\upsilon)f(A) \geq (1-\epsilon)f(A) \qquad (7)$$

This proves that conditioned on G and $\mathcal{E}_1$, the estimate returned is never too small. For the other direction, we note that conditioned on $\mathcal{E}_1$ being true:
$est_f(B_1^{\ell^*}) \leq (1+\upsilon)f(B_1^{\ell^*}) \leq (1+\upsilon)f(A) \leq (1+\epsilon)f(A)$
where we have used $f(B_1^{\ell^*}) \leq f(A)$, and $\upsilon < \epsilon$.

To complete the proof of the theorem, note that

$$
\begin{aligned}
\Pr[G \wedge \mathcal{E}_1] &= 1 - \Pr[\bar{G} \vee \bar{\mathcal{E}}_1] \\
&\geq 1 - \Pr[\bar{G}] - \Pr[\bar{\mathcal{E}}_1] \\
&\geq 1 - \frac{\delta}{2} - \gamma \quad \text{using Lemma 1 and Eqn 2} \\
&\geq 1 - \delta \quad \text{using } \gamma < \delta/2
\end{aligned}
$$

∎

C. Frequency Moments F_k

In this section, we show how the general technique that we presented can yield a data structure for the correlated estimation of the frequency moments F_k, $k \geq 2$.

Fact 1: (Hölder's Inequality) For vectors a and b of the same dimension, and any integer $k \geq 1$, $\langle a, b \rangle \leq \|a\|_k \cdot \|b\|_{k/(k-1)}$.

Lemma 6: For sets $S_i, i = 1 \ldots j$, if $F_k(S_i) \leq \beta$ for each $i = 1 \ldots j$, then $F_k(\cup_{i=1}^j S_i) \leq j^k \beta$.

Proof: Fact 1 on j-dimensional vectors a and b implies that $|\langle a, b \rangle|^k \leq \|a\|_k^k \cdot \|b\|_{k/(k-1)}^k$. Setting $b = (1, 1, \ldots, 1)$, it follows that $(a_1 + \cdots + a_j)^k \leq j^{k-1}(a_1^k + \cdots + a_j^k)$. Hence, it follows that $F_k(\cup_{i=1}^j \cup S_i) \leq j^{k-1} \sum_{i=1}^j F_k(S_i) \leq j^k \beta$. ∎

Lemma 7: If $F_k(B) \leq (\epsilon/(3k))^k F_k(A)$, then $F_k(A \cup B) \leq (1+\epsilon)F_k(A)$.

Proof: Suppose A and B have support on $\{1, 2, \ldots, n\}$. Let a and b be the characteristic vectors of sets A and B, respectively. Using Fact 1, we have

$$
\begin{aligned}
&F_k(A \cup B) \\
&= \sum_{i=1}^n (a_i + b_i)^k \\
&= F_k(A) + F_k(B) + \sum_{i=1}^n \sum_{j=1}^{k-1} \binom{k}{j} a_i^j b_i^{k-j} \\
&= F_k(A) + F_k(B) + \sum_{j=1}^{k-1} \binom{k}{j} \sum_{i=1}^n a_i^j b_i^{k-j} \\
&\leq F_k(A) + F_k(B) + \sum_{j=1}^{j-1} \\
&\quad \binom{k}{j} \left(\sum_{i=1}^n (a_i^j)^{k/j}\right)^{\frac{j}{k}} \left(\sum_{i=1}^n (b_i^{k-j})^{k/(k-j)}\right)^{\frac{k-j}{k}} \\
&= F_k(A) + F_k(B) + \sum_{j=1}^{k-1} \binom{k}{j} F_k(A)^{\frac{j}{k}} F_k(B)^{\frac{k-j}{k}} \\
&\leq F_k(A) + F_k(B) + \sum_{j=1}^{k-1} \binom{k}{j} F_k(A) \left(\frac{\epsilon}{3k}\right)^{k-j} \\
&\leq (1+\epsilon/3)F_k(A) + F_k(A) \sum_{j=1}^{k-1} \binom{k}{j} (\epsilon/(3k))^{k-j}
\end{aligned}
$$

$$\leq (1+\epsilon/3)F_k(A) + F_k(A)(1+\epsilon/(3k))^k - F_k(A)$$
$$\leq (1+\epsilon/3)F_k(A) + F_k(A)(1+2\epsilon/3) - F_k(A)$$
$$\leq (1+\epsilon)F_k(A),$$

where we used that $(1+x)^y \leq e^{xy}$ for all x and y, and $e^z \leq 1+2z$ for $z \leq 1/2$. This completes the proof. ∎

Lemma 8: If $C \subset D$, and $F_k(C) \leq (\epsilon/(9k))^k F_k(D)$, then $F_k(D-C) \geq F_k(D)$.

Proof: We know that for any two sets A and B, $F_k(A \cup B) \leq 2^k(F_k(A) + F_k(B))$.

$$F_k(D) = F_k((D-C) \cup (C)$$
$$\leq 2^k(F_k(D-C) + F_k(C))$$

which leads to

$$F_k(D-C) \geq F_k(D)/2^k - F_k(C)$$
$$\geq ((9k/\epsilon)^k (1/2^k)^k - 1)F_k(C)$$
$$\geq (3k/\epsilon)^k F_k(C)$$

Thus, $F_k(C) \leq (\epsilon/3k)^k F_k(D-C)$. Applying Lemma 7, we get $F_k(C \cup (D-C)) \leq (1+\epsilon)F_k(D-C)$. Thus, $F_k(D-C) \geq F_k(D)/(1+\epsilon) \geq (1-\epsilon)F_k(D)$. ∎

Theorem 3: For parameters $0 < \epsilon < 1$ and $0 < \delta < 1$, there is a sketch for an (ϵ, δ)-estimation of the correlated aggregate F_k on a stream of tuples of total length n, using space $n^{1-2/k}poly(\epsilon^{-1}\log(n/\delta))$.

Proof: From Lemma 6, we have $c_1^{F_k}(j) = j^k$. From Lemma 8, we have $c_2^{F_k}(\epsilon) = (\epsilon/(9k))^k$. Using these in Theorem 1, we get $c_1^{F_k}(\log y_{max}) = (\log y_{max})^k$, and $c_2^f(\epsilon/2) = (\epsilon/(18k))^k$. Using the sketches for F_2 from [1] and for $F_k, k > 2$ from [10], we get the above result. ∎

Remark 4: The space can be improved to $r^{1-2/k}poly(\epsilon^{-1}\log(n/\delta))$, where r is the number of distinct x_i-values in the stream [10]. In the worst-case, though, r could be $\Theta(n)$.

We make the dependence more explicit for the case of F_2.

Lemma 9: For parameters $0 < \epsilon < 1$ and $0 < \delta < 1$, there is a sketch for (ϵ, δ) error correlated estimation of F_2 on a stream of tuples of total length n, using space $O(\epsilon^{-4}(\log(1/\delta) + \log y_{max})(\log^2 y_{max})(\log^2 f_{max}))$ bits.

The amortized update time is $O(\log f_{max} \cdot \log y_{max})$.

Proof: The space taken by a sketch for an (ϵ, δ) estimator for F_2 on a stream is $O((\log f_{max})(1/\epsilon^2)\log(1/\delta))$ bits [1]. From the proof of Theorem 3, we have $c_1^{F_2}(j) = j^2$, and $c_2^{F_2}(\epsilon) = (\epsilon/18)^2$.

Using the above in Theorem 1, we get the space to be $O(\epsilon^{-4}\log^2 f_{max} \log^2 y_{max}(\log 1/\delta + \log y_{max}))$ bits.

To get $O(\log f_{max}(\log 1/\delta + \log y_{max}))$ amortized processing time, observe that there are $O(\log f_{max})$ data structures S_i, each containing $O(\epsilon^{-2}\log^2 y_{\max})$ buckets, each holding a sketch of $O(\epsilon^{-2}\log f_{max}(\log 1/\delta + \log y_{max}))$ bits.

We process a batch of $O(\epsilon^{-2}\log^2 y_{\max})$ updates at once. We first sort the batch in order of non-decreasing y-coordinate. This can be done in $O(\epsilon^{-2}\log^2 y_{\max}(\log 1/\epsilon + \log\log y_{\max}))$ time. Then we do the following for each S_i. We perform a pre-order traversal of the buckets in S_i and we update the appropriate buckets. Importantly, each bucket maintains an update-efficient AMS sketch due to Thorup and Zhang [22], which can be updated in time $O(\log 1/\delta + \log y_{max})$. Since our updates are sorted in increasing y-value and the list is represented as a pre-order traversal, the total time to update S_i is $O(\epsilon^{-2}\log^2 y_{\max}(\log 1/\delta + \log y_{max}))$. The time to update all the S_i is $O(\log f_{max})$ times this. So the amortized time is $O(\log f_{max}(\log 1/\delta + \log y_{max}))$. ∎

D. Other Useful Statistics

While many aggregation function satisfy the properties described above, some important ones do not. However, in many important remaining cases, these aggregation functions are related to aggregation functions that do satisfy these properties, and the mere fact that they are related in the appropriate way enables efficient estimation of the corresponding correlated aggregate. The idea is similar in spirit to work by Braverman, Gelles and Ostrovsky [23].

We can compute the correlated F_2-heavy hitters, as well as the rarity (defined below) by relating these quantities to F_2 and F_0, respectively.

For example, in the correlated F_2-heavy hitters problem with y-bound of c and parameters ϵ, ϕ, $0 < \epsilon < \phi < 1$, letting $F_2(c)$ denote the correlated F_2-aggregate with y-bound of c, then we wish to return all x for which $|\{(x_i, y_i) \mid x_i = x \wedge y_i \leq c\}|^2 \geq \phi F_2(c)$, and no x for which $|\{(x_i, y_i) \mid x_i = x \wedge y_i \leq c\}|^2 \leq (\phi - \epsilon)F_2(c)$. To do this, we use the same data structures S_i as used for estimating the correlated aggregate F_2. However, for each S_i and each bucket in S_i we additionally maintain an algorithm for estimating the squared frequency of each item inserted into the bucket up to an additive $(\epsilon/10) \cdot 2^i$. See, e.g., the COUNTSKETCH algorithm of [24] for such an algorithm. To estimate the correlated F_2-heavy hitters, for each item we obtain an additive $(\epsilon/10) \cdot F_2(c)$ approximation to its squared frequency by summing up the estimates provided for it over the different buckets contained in $[0, c]$ in the data structure S_i used for estimating $F_2(c)$. Since only an $\epsilon/10$ fraction of $F_2(c)$ does not occur in such buckets, we obtain the list of all heavy hitters this way, and no spurious ones.

In the rarity problem, the problem is to estimate the fraction of distinct items which occur exactly once in the multi-set. The ideas for estimating rarity are similar, where we maintain the same data structures S_i for estimating the correlated aggregate F_0, but in each bucket maintain data structures for estimating the rarity of items inserted into that bucket. This is similar to ideas of [23]; we omit the details.

E. Number of Distinct Elements

The number of distinct elements in a stream, also known as the zeroth frequency moment F_0, is a fundamental and widely studied statistic of the stream. In this section, we consider the correlated estimation of the number of distinct elements in a stream. Consider a stream of (x, y) tuples, where $x \in \{1, \ldots, m\}$ and $y \in \{1, y_{max}\}$. The goal is to estimate, given

a parameter c at query time, the value $|\{x|((x,y) \in S) \wedge (y \leq c)\}|$

Our algorithm is an adaptation of the algorithm for estimating the number of distinct elements within a sliding window of a data stream, due to Gibbons and Tirthapura [25]. Similar to their algorithm, our algorithm for correlated estimation of F_0 is based on "distinct sampling", or sampling based on the hash values of the item identifiers. We maintain multiple samples, $S_0, S_1, \ldots, S_k$, where $k = \log m$. Suppose that for simplicity, we have a hash function h that maps elements in $\{1, \ldots, m\}$ to the real interval $[0, 1]$. This assumption of needing such a powerful hash function can be removed, as shown in [25].

The algorithm in [25] proceeds as follows. Stream items are placed in these samples S_i in the following manner. (A)Each item (x, y) is placed in S_0. (B)For $i > 0$, an item (x, y) is placed in level i iff $h(x) <= \frac{1}{2^i}$. Note that if an item x is placed in level i, it must have been placed in level $i - 1$ also.

Since each level has a limited space budget, say α, we also need a way to discard elements from each level. Our algorithm differs from [25] in the following aspect of how to discard elements from each level. *For correlated aggregates, we maintain in S_i only those items (x, y) that (1)have an x value that is sampled into S_i, and (2)have the smallest y values among all the elements sampled into S_i.* In other words, it is a priority queue using the y values as the weights, whereas in [25], each level was a simple FIFO (first-in-first-out) queue.

Our algorithm takes advantage of the fact that the basic data structure in [25] is a sample, and it is easy to maintain a sample that is solely dependent on the values of the items that were inserted into the sample, and independent of the *order* in which items were inserted. It can be shown that the above scheme of retaining those elements with a smaller value of y, when combined with the sampling scheme in [25], yields an (ϵ, δ) estimator for the correlated distinct counts. We omit the proof and a detailed description of the algorithm. We however, present results on the experimental performance of this data structure, showing that it is very viable in practice.

We note that other methods for estimating distinct elements may also be adapted to work here, such as the variant of the algorithm due to Flajolet and Martin [26], as elaborated by Datar et al. [11]. We are however, not aware of any previous work applying these ideas to the context of correlated aggregates, or associated experimental results.

Theorem 5: Given parameters $0 < \epsilon < 1$ and $0 < \delta < 1$, there is a streaming algorithm that can maintain a summary of a stream of tuples (x, y), where $x \in \{1, \ldots, m\}$ and $y \in \{1, y_{max}\}$ such that (1)The space of the summary is $O(\log m + \log y_{max}) \frac{\log m}{\epsilon^2} \log 1/\delta$ bits (2)The summary can be updated online as stream elements arrive, and (3)Given a query y_0, the summary can return an (ϵ, δ)-estimator of $|\{x|((x,y) \in S) \wedge (y \leq y_0)\}|$

III. DELETIONS IN A STREAM

While in many application settings we see a data stream of insertions, in other settings it is important to consider deletions, or more generally, pairs (x_i, y_i) together with an integer weight which may be positive or negative. For instance, x_i and y_i may represent the first two attribute values of a record, and for a given application, we may not be interested in the remaining attributes. If there are many records with the same first two attribute values, one can represent all such records with a single positive integer weight followed by the pair (x_i, y_i).

Allowing negative integer weights allows for analyzing attributes which occur in the symmetric difference of two datasets. Indeed, suppose the records in each dataset are represented by a positive integer weight together with a pair of attribute values. We can include all records from the first dataset in the data stream. Then we can negate the weights of all records from the second dataset and append these to the data stream. The absolute value of the sum of the weights of a pair of attribute values in the stream represents the number of times the pair occurs in the symmetric difference of the datasets.

This data stream model with positive and negative weights is referred to as the *turnstile model* in the data stream literature; see, e.g., [27].

In the turnstile model our upper bounds no longer hold. This is not an artifact of our algorithm or analysis, as we now show an impossibility result in this setting. In fact, we show a lower bound assuming the weights are restricted to come from the set $\{1, -1\}$, i.e., when we see a record (x_i, y_i) we also see a label "insert" or "delete", corresponding to weight 1 or weight -1, respectively.

Consider a correlated aggregate function f of the following form. For each (x_i, y_i) seen in the stream, $1 \leq i \leq n$, we assume $x_i \in [m]$ and $y_i \in \{0, 1, \ldots, y_{\max}\}$. Then for $j \in [m]$ and $\tau \in \{0, 1, \ldots, y_{\max}\}$, let $j(\tau)$ equal the sum, over i, of the weights assigned to records of the form (x_i, y), where $x_i = j$ and $0 \leq y \leq \tau$. We consider functions of the form $f_\tau = \sum_{j=1}^{m} g(j(\tau))$, where $g : \{-n, -n+1, -n+2, \ldots, n-1, n\} \rightarrow \{0, 1, 2, \ldots, poly(n)\}$ is a non-negative function with $g(k) = 0$ iff $k = 0$, and $poly(n)$ is some positive polynomial. A query specifies an index $\tau \in \{0, 1, \ldots, y_{\max}\}$ and then requires an (ϵ, δ)-approximation to $f_\tau(x)$. This class of functions contains all frequency moments F_k studied in earlier sections.

We show an impossibility result even if we allow *multiple passes* over the data stream. Namely, we show that any t-pass algorithm for approximating f_τ, for any τ given at query time, up to a constant factor and with constant probability, requires $y_{\max}^{\Omega(1/t)} / \log y_{\max}$ bits of space, even when $m = 2$ and $n = O(y_{\max})$. Hence, deletions cause estimation to be significantly harder, even if allowed multiple passes.

We match our lower bound by giving an $O(\log y_{\max})$-pass, $O(\epsilon^{-1} \cdot \log y_{\max} \cdot s(f, n, \epsilon/30, \delta/(y_{\max}+1)))$-space algorithm for this problem in the turnstile model, where $s(f, n, \epsilon, \delta)$ is the space complexity of a 1-pass algorithm which (ϵ, δ)-approximates the function f on a stream of length at most n.

A. Lower Bound

Our lower bound comes from a two-party communication problem between players, denoted Alice and Bob. While communication complexity is often used to prove streaming lower bounds [27], we have not seen the communication problem we use, the GREATER-THAN problem, used to establish multi-pass lower bounds.

Definition 3: In the two-party GREATER-THAN communication problem between Alice and Bob, Alice has a number $a \in [2^r]$, Bob has $b \in [2^r]$, and they want to know if $a > b$.

Definition 4: The t-round randomized communication complexity of a problem is the minimum, over all randomized protocols for computing a function which on every input fail with probability at most $1/3$ (over the protocol's random coin tosses), of the maximum number of bits exchanged by the two parties, subject to the constraint that there are at most t messages exchanged.

Theorem 6: ([28]) The t-round randomized communication complexity of GREATER-THAN is $\Omega(r^{1/t})$.

Theorem 7: Any t-pass randomized algorithm ALG for estimating a function f of the above form (together with the associated function g) up to a constant factor with constant probability for a τ given at query time, must use $y_{\max}^{\Omega(1/t)}/\log y_{\max}$ bits of memory, even if $m = 2$ and $n = O(y_{\max})$.

Proof: By increasing the space complexity of ALG by an $O(\log y_{\max})$ factor, e.g., using independent repetition and taking the median of outputs, we can assume that ALG is correct for all $0 \le \tau \le y_{\max}$.

We reduce from the GREATER-THAN communication problem on input strings of length $y_{\max}$. Letting $a_1, \ldots, a_{y_{\max}}$ be the binary representation of Alice's input a, where a_1 is the most significant bit, she inserts the values $(1 + a_i, i)$ with weight 1 into the stream, where $1 \le i \le y_{\max}$. She feeds this stream to ALG. After completion, she sends the state of ALG to Bob who inserts the values $(1 + b_i, i)$ with weight -1 into the stream, for $1 \le i \le y_{\max}$. Bob then sends the state of ALG back to Alice, who continues the computation of ALG on the stream she has created. If ALG uses t passes, this results in a $(2t - 1)$-round communication protocol. Observe that the stream length $n = 2y_{\max}$.

At the end of the $(2t - 1)$-st round, ALG is queried on $\tau = 0, 1, 2, \ldots, y_{\max}$. Let τ be the smallest non-zero index returned by ALG for which the estimate to f_τ is positive. If $b_\tau = 1$, Bob declares that $b > a$, otherwise he declares that $a > b$. If for all τ the estimate to f_τ is 0, Bob declares $b = a$. Correctness follows from the two facts (1) the statement $a > b$ is equivalent to having the first index τ at which a and b disagree in their binary representation satisfying $a_\tau = 1$ while $b_\tau = 0$, and (2) $g(k) = 0$ iff $k = 0$. It follows from Theorem 7 that ALG must use $\Omega(y_{\max}^{1/(2t-1)}/\log y_{\max}) = y_{\max}^{\Omega(1/t)}/\log y_{\max}$ bits of space. ∎

B. Multipass Upper Bound

We give an $O(\log y_{\max})$-pass upper bound for computing correlated aggregates f of the form above, showing that the fact that our lower bound becomes trivial when $t =$

$O(\log y_{\max})$ is no coincidence. Our algorithm MULTIPASS is given in Figure 14. We divide the interval $[0, y_{\max}]$ of y-values into positions $p(0), p(1), p(2), \ldots, p(r)$, for some $r = O(\epsilon^{-1} \log(nm))$. Ideally, the $p(i)$ are such that the $\hat{f}_{p(i)}$ satisfy

$$(1 - \epsilon) \cdot (1 + \epsilon)^i \le f_{p(i)} \le (1 + \epsilon) \cdot (1 + \epsilon)^i.$$

Given the output of MULTIPASS and a query τ, the QUERY-RESPONSE algorithm could then find the largest value of i for which $p(i) \le \tau$ and output $(1 + \epsilon)^i$. Since the $p(i)$ represent (approximate) jumps in the function value by powers of $(1+\epsilon)$, it would follow that we obtain a good approximation to f_τ.

Unfortunately, the above guarantee on $f_{p(i)}$ is impossible as there may be no index j for which $(1 - \epsilon) \cdot (1 + \epsilon)^i \le f_j \le (1 + \epsilon) \cdot (1 + \epsilon)^i$, e.g., if there exist indices k for which $f_k \gg f_{k-1}$. We instead impose the requirement that $p(i)$ is an index for which

$$(1 - \epsilon) \cdot (1 + \epsilon)^i \le f_{p(i)} \quad \text{and} \quad f_{p(i)-1} \le (1 + \epsilon)^i.$$

Notice that we may have $p(i) = p(i + 1)$ in some cases. Given the output of MULTIPASS and a query τ, the QUERY-RESPONSE algorithm first finds the largest value of i for which $p(i) \le \tau$. It then outputs $(1 + \epsilon)^i$. Notice that if $p(i - \ell) = p(i - \ell + 1) = \cdots = p(i)$, we indeed need to output $(1 + \epsilon)^i$, so it is important to find the largest i for which $p(i) \le \tau$.

In the description of MULTIPASS, when we say that an algorithm (ϵ, δ)-approximates a function f, we use this to mean it outputs a number $\hat{f}$ with $f \le \hat{f} \le (1 + \epsilon)f$ with probability at least $1 - \delta$. This is a one-sided estimator, which can be constructed from any two-sided estimator by scaling.

Algorithm 4: Our $O(\log y_{\max})$-pass MULTIPASS protocol for estimating f. Without loss of generality, $y_{\max} + 1$ is assumed to be a power of 2.

1 Let $\mathcal{A}$ be a classical streaming algorithm for (ϵ, δ')-approximating f, where $\delta' = \delta/(y_{\max} + 1)$.

2 Fix the random string of $\mathcal{A}$ for the rest of this algorithm.

3 In the first pass (ϵ, δ')-approximate $f_{y_{\max}}$ using $\mathcal{A}$, obtaining estimate $\hat{f}_{y_{\max}}$.

4 Set $r = \lceil \log_{1+\epsilon} \hat{f}_{y_{\max}} \rceil$.

5 In parallel **for** i *from* 0 *to* r **do**

6 $p(i) = (y_{\max} - 1)/2$.

7   *binary search procedure*

8   **for** j *from* 2 *to* $\log y_{\max}$ **do**

9 (ϵ, δ')-approximate $f_{p(i)}$ using $\mathcal{A}$, obtaining estimate $\hat{f}_{p(i)}$.

10   If $\hat{f}_{p(i)} > (1 + \epsilon)^i$, then $p(i) \leftarrow p(i) - (y_{\max} + 1)/2^j$, else $p(i) \leftarrow p(i) + (y_{\max} + 1)/2^j$.

11   **end**

12   If $\hat{f}_{p(i)} < (1 + \epsilon)^i$, then $p(i) \leftarrow p(i) + 1$.

13 **end**

14 Output $p(0), p(1), p(2), \ldots, p(r)$.

Theorem 8: Algorithm MULTIPASS is an $O(\log y_{\max})$-pass, $O(\epsilon^{-1}s(f, n, \epsilon, \delta/(y_{\max} + 1)) \log(nm))$-space algorithm for which, given a query τ, the QUERY-RESPONSE algorithm outputs an (ϵ, δ)-approximation to f_τ.

Proof: The pass and space complexity of MULTIPASS follow immediately from the description of the algorithm given in Figure 14. It remains to argue correctness. We condition on the event $\mathcal{E}$ that, for every $\tau \in \{0, 1, 2, \ldots, y_{\max}\}$, algorithm $\mathcal{A}$ outputs an estimate $\hat{f}_\tau$ with $f_\tau \leq \hat{f}_\tau \leq (1+\epsilon)f_\tau$. Since $\mathcal{A}$ fails with probability at most δ', it follows that $\Pr[\mathcal{E}] \geq 1 - \delta$.

Consider $p(i)$ for some $i \in \{0, 1, 2, \ldots, r\}$.

We first would like to argue that $f_{p(i)} \geq (1 - \epsilon)(1 + \epsilon)^i$. Consider the behavior of the algorithm before Step 12 is reached. At some point the nearest common ancestor a in the tree on leaves $\{0, 1, 2, \ldots, r\}$ of $p(i)$ and $p(i) + 1$ was considered, and the algorithm branched to the left. This means $\hat{f}_{p(i)+1} > (1 + \epsilon)^i$. Now, once Step 12 is reached, if $\hat{f}_{p(i)} < (1 + \epsilon)^i$, then $p(i)$ is replaced with $p(i) + 1$, and so after Step 12 we are guaranteed that $\hat{f}_{p(i)} \geq (1 + \epsilon)^i$. This means that

$$f_{p(i)} \geq \frac{\hat{f}_{p(i)}}{1 + \epsilon} \geq \frac{(1 + \epsilon)^i}{1 + \epsilon} \geq (1 - \epsilon) \cdot (1 + \epsilon)^i.$$

Second, we would like to argue that $f_{p(i)-1} \leq (1+\epsilon)^i$. Consider the behavior of the algorithm before Step 12 is reached. At some point the nearest ancestor a in the tree on leaves $\{0, 1, 2, \ldots, r\}$ of $p(i) - 1$ and $p(i)$ was considered, and the algorithm branched to the right. This means $\hat{f}_{p(i)-1} \leq (1+\epsilon)^i$, Now, once Step 12 is reached, if $\hat{f}_{p(i)} \geq (1+\epsilon)^i$, then $p(i) - 1$ remains the same. Otherwise, $p(i) - 1$ is replaced with $p(i)$, and so after Step 12 we have $\hat{f}_{p(i)-1} < (1 + \epsilon)^i$. Hence, in either case,

$$f_{p(i)-1} \leq \hat{f}_{p(i)-1} \leq (1 + \epsilon)^i.$$

Given a query τ, the QUERY-RESPONSE algorithm first finds the largest value of i for which $p(i) \leq \tau$. It then outputs $(1 + \epsilon)^i$. Since $\tau \geq p(i)$, we have

$$f_\tau \geq f_{p(i)} \geq (1 - \epsilon)(1 + \epsilon)^i.$$

Since $p(i) + 1 > \tau$, we have

$$f_\tau \leq f_{p(i)+1} \leq (1 + \epsilon)^{i+1}.$$

It follows that the output of $(1+\epsilon)^i$ is a $(1+\epsilon)$-approximation, as desired. ∎

Remark 9: For the case of $f = F_2$ and $m, n, y_{\max}$ being polynomially related, we get an $O(\epsilon^{-3} \log^3 n)$-space $O(\log n)$-pass algorithm for $(\epsilon, 1/n)$-approximation given a query point τ. This improves the $O(\epsilon^{-4} \log^5 n)$-space complexity of our 1-pass algorithm for estimating F_2 with $1/n$ error probability, at the cost of additional passes.

IV. EXPERIMENTAL EVALUATION

A. Correlated F_2, Second Frequency Moment

Setup: We implemented our algorithm for correlated F_2 estimation in Python. Our goal was to evaluate the scaling behavior of our algorithms for relatively large datasets.

We used the following three datasets for our experiments. (1)The *Uniform* data set, which is a sequence of tuples (x, y) where x is generated uniformly at random from the set $\{0, \ldots, 500000\}$ and y is generated uniformly at random from the set $\{0, \ldots, 1000000\}$. This maximum size of this dataset is 50 million. (2)The Zipfian data set, with $\alpha = 1$. Here the x values are generated according to the Zipfian distribution with parameter $\alpha = 1$, from the domain $\{0, \ldots, 500000\}$, and the y values are generated uniformly at random from the set $\{0, \ldots, 1000000\}$. The maximum size of this dataset is 50 million. (3)The Zipfian data set as described above, with α set to 2.

For the sketch for F_2, we used a variant of the algorithm due to Alon *et al.*[1], based on the idea of Thorup and Zhang [22]. This variant gives much better update time than the original algorithm of Alon *et al.*.

Our first observation was that for all the datasets, the relative error of the algorithm was almost always within the desired approximation error ϵ, for $\delta < 0.2$.

Space Usage as a function of ϵ: We measured the space consumption of the algorithm in terms of the number of tuples stored by it (the storage for each tuple is a constant number of bytes). The space depends on a number of factors, including the values of δ, the value of f_{max} (since f_{max} determines the maximum size of the data structure at each level) and more critically, on ϵ.

In Figure 1, the space taken for the summary for F_2 is plotted as a function of ϵ. This is shown for all the data sets described above, with each dataset of size 40 million tuples. We note that the space taken by the sketch increases rapidly with decreasing ϵ, and the rate of the growth is similar for all four datasets. Further, the rate of growth of space with respect to ϵ remains similar for all data sets.

Space Usage as a function of the stream size: We next analyze the space taken by the sketch as a function of the stream size. The results are shown in Figure 2 (for $\epsilon = 0.15$), Figure 3 (for $\epsilon = 0.2$), and Figure 4 (for $\epsilon = 0.25$). The good news is that in all cases, as predicted by theory, the space taken by the sketch does not change much, and increases only slightly as the stream size increases. This shows that the space savings of this algorithm is much larger with streams that are larger in size.

The time required for processing the stream was nearly the same for all the three datasets; processing 2 million elements took about 5 minutes on an Intel 8 CPU machine with 24GB of RAM (the simulation did not make use of parallelism). The processing rate can be improved by using the C/C++ language, and by using a more optimized implementation than ours. These experiments show that a reasonable processing rate can be achieved for the data structure for F_2, and that correlated query processing is indeed practical, and provides significant space savings, especially for large data streams (of the order of 10 million tuples or larger).
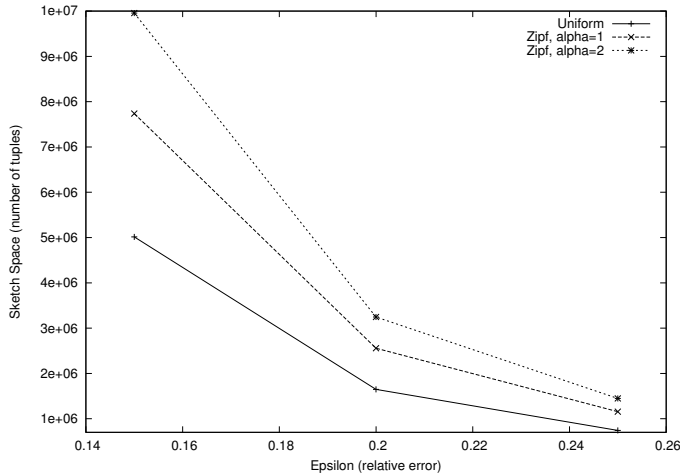
Fig. 1. F_2: Space taken by the sketch for versus relative error ϵ. The stream size is 40 million, for all datasets.
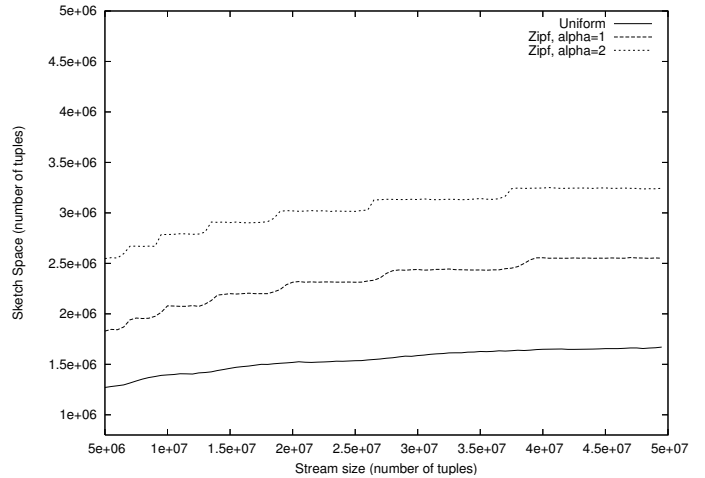

Fig. 3. F_2: Space taken by the sketch versus stream size n, $\epsilon = 0.2$.

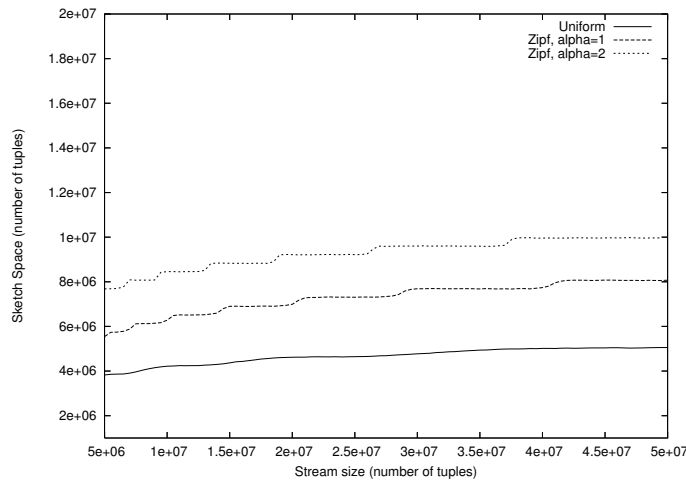
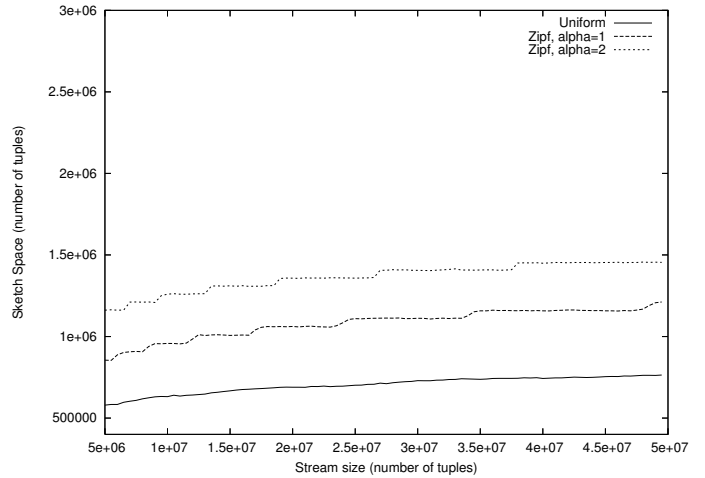Fig. 2. F_2: Space taken by the sketch versus stream size n, $\epsilon = 0.15$.

Fig. 4. F_2: Space taken by the sketch versus stream size n, $\epsilon = 0.25$.

B. Correlated F_0, Number of Distinct Elements

We implemented the algorithm for F_0 in Python. In addition to the three datasets described in F_2, we used an additional dataset derived from packet traces of Ethernet traffic on a LAN and a WAN (see http://ita.ee.lbl.gov/html/contrib/BC.html). The relevant data here is the number of bytes in the packet, and the timestamp on the packet (in milliseconds). We call this dataset the "Ethernet" dataset. This dataset has 2 million packets, and was constructed by taking two packet traces from the above source and combining them by interleaving them. We also considered the first 2 million tuples for the other three datasets.

Another difference is that in the Uniform and Zipfian datasets, the range of x values was made larger ($0\ldots 1000000$) than in the case of F_2, where the range of x values was $0\ldots 500000$. The reason for this change is that there are much simpler algorithms for correlated F_0 estimation when the domain size is small: simply maintain the list of all distinct elements seen so far along the x dimension, along with the

smallest value associated with it in the y dimension. Note that such a simplification is not (easily) possible for the case of F_2.

The variation of the sketch size with ϵ is shown in Figure 5. Note that while the sketch size decreases with increasing ϵ, the rate of decrease is not as fast as in the case of F_2. Further, note that the sketch size for comparable values of ϵ is much smaller than the sketch for correlated F_2. Another point is that the space taken by the sketch for the Ethernet dataset is significantly smaller than the sketch for the other datasets. This is due to the fact that the range of x values in the Ethernet dataset was much smaller ($0..2000$) than for the other datasets ($0...1000000$). The number of levels in the data structure is proportional to the logarithm of the number of possible values along the x dimension. Note that as explained above, the algorithm of choice for correlated F_0 estimation for the Ethernet-type datasets (where the x range is small) will be different from our sketch, as explained above. Our algorithm is useful for datasets where the x range is much larger.

The size of the sketch as a function of the stream size is shown in Figure 6, for $\epsilon = 1$. It can be seen that the sketch size hardly changes with the stream size. Note however, that for much smaller streams, the sketch will be smaller, since some of the data structures at different levels have not reached their maximum size yet. The results for other values of ϵ are similar, and are not shown here.
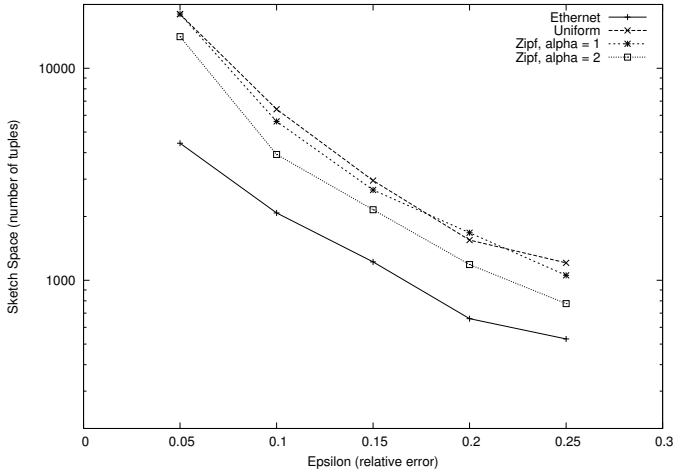

Fig. 5. F_0: Space taken by the sketch versus relative error ϵ. The stream size is 2×10^6, for all datasets.
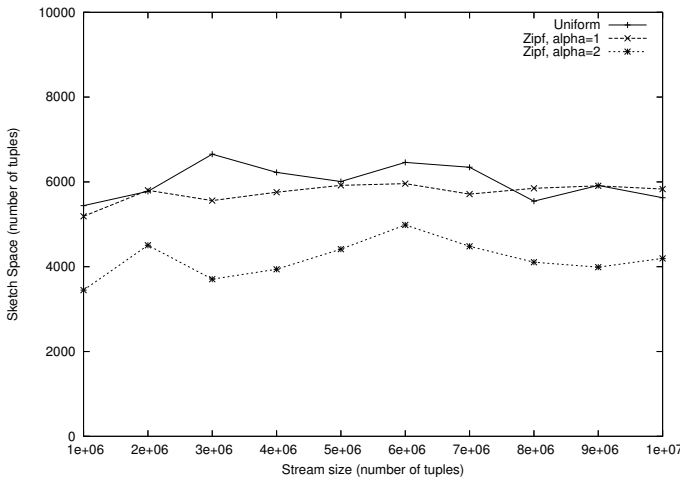

Fig. 6. F_0: Space taken by the sketch versus stream size n, for the Uniform and Zipfian datasets. $\epsilon = 0.1$.

REFERENCES

[1] N. Alon, Y. Matias, and M. Szegedy, "The space complexity of approximating the frequency moments," *Journal of Computer and System Sciences*, vol. 58, no. 1, pp. 137–147, 1999.

[2] M. Greenwald and S. Khanna, "Space efficient online computation of quantile summaries," in *Proc. ACM International Conference on Management of Data (SIGMOD)*, 2001, pp. 58–66.

[3] L. Lee and H. Ting, "A simpler and more efficient deterministic scheme for finding frequent items over sliding windows," in *Proceedings of ACM Principles of Database Systems*, 2006, pp. 290–297.

[4] D. M. Kane, J. Nelson, and D. P. Woodruff, "An optimal algorithm for the distinct elements problem," in *PODS*, 2010, pp. 41–52.

[5] IBM, "System s - stream computing at ibm research," IBM White Paper.

[6] C. Systems, "Cisco ios netflow," http://www.cisco.com/web/go/netflow.

[7] N. Shrivastava, C. Buragohain, D. Agrawal, and S. Suri, "Medians and beyond: new aggregation techniques for sensor networks," in *SenSys*, 2004.

[8] A. Andoni, R. Krauthgamer, and K. Onak, "Streaming algorithms from precision sampling," in *Foundations of Computer Science (FOCS)*, 2011.

[9] C. Busch and S. Tirthapura, "A deterministic algorithm for summarizing asynchronous streams over a sliding window," in *STACS*, 2007.

[10] P. Indyk and D. P. Woodruff, "Optimal approximations of the frequency moments of data streams," in *Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC)*, 2005, pp. 202–208.

[11] M. Datar, A. Gionis, P. Indyk, and R. Motwani, "Maintaining stream statistics over sliding windows," *SIAM Journal on Computing*, vol. 31, no. 6, pp. 1794–1813, 2002.

[12] B. Xu, S. Tirthapura, and C. Busch, "Sketching asynchronous data streams over sliding windows," *Distributed Computing*, vol. 20, no. 5, pp. 359–374, 2008.

[13] D. Chatziantoniou, M. O. Akinde, T. Johnson, and S. Kim, "The md-join: An operator for complex olap," in *ICDE*, 2001, pp. 524–533.

[14] D. Chatziantoniou and K. A. Ross, "Querying multiple features of groups in relational databases," in *VLDB*, 1996.

[15] D. Chatziantoniou, "Ad hoc olap: Expression and evaluation," in *ICDE*, 1999, p. 250.

[16] J. Gehrke, F. Korn, and D. Srivastava, "On computing correlated aggregates over continual data streams," in *SIGMOD*, 2001.

[17] R. Ananthakrishna, A. Das, J. Gehrke, F. Korn, S. Muthukrishnan, and D. Srivastava, "Efficient approximation of correlated sums on data streams," *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, no. 3, pp. 569–572, 2003.

[18] G. Cormode, F. Korn, and S. Tirthapura, "Time-decaying aggregates in out-of-order streams," in *ICDE*, 2008.

[19] H.-L. Chan, T. W. Lam, L.-K. Lee, and H.-F. Ting, "Approximating frequent items in asynchronous data stream over a sliding window," in *Workshop on Approximation and Online Algorithms (WAOA)*, 2009, pp. 49–61.

[20] V. Braverman and R. Ostrovsky, "Effective computations on sliding windows," *SIAM Journal on Computing.*, vol. 39, no. 6, pp. 2113–2131, 2010.

[21] P. Gibbons and S. Tirthapura, "Distributed streams algorithms for sliding windows," in *SPAA*, 2002.

[22] M. Thorup and Y. Zhang, "Tabulation based 4-universal hashing with applications to second moment estimation," in *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2004, pp. 615–624.

[23] V. Braverman, R. Gelles, and R. Ostrovsky, "How to catch l_2-heavy-hitters on sliding windows," *CoRR*, vol. abs/1012.3130, 2010.

[24] M. Charikar, K. Chen, and M. Farach-Colton, "Finding frequent items in data streams," *Theor. Comput. Sci.*, vol. 312, no. 1, pp. 3–15, 2004.

[25] P. Gibbons and S. Tirthapura, "Distributed streams algorithms for sliding windows," *Theory of Computing Systems*, vol. 37, pp. 457–478, 2004.

[26] P. Flajolet and G. N. Martin, "Probabilistic counting algorithms for data base applications," *Journal of Computer and System Sciences*, vol. 31, pp. 182–209, 1985.

[27] S. Muthukrishnan, *Data Streams: Algorithms and Applications*, ser. Foundations and Trends in Theoretical Computer Science. Now Publishers, August 2005.

[28] P. B. Miltersen, N. Nisan, S. Safra, and A. Wigderson, "On data structures and asymmetric communication complexity," *J. Comput. Syst. Sci.*, vol. 57, no. 1, pp. 37–49, 1998.