Noname manuscript No.
(will be inserted by the editor)

A General Method for Estimating Correlated Aggregates Over a Data Stream

Srikanta Tirthapura · **David P. Woodruff**

the date of receipt and acceptance should be inserted later

Abstract On a stream $\mathscr{S}$ of two dimensional data items (x,y) where x is an item identifier and y is a numerical attribute, a correlated aggregate query $C(\sigma,AGG,\mathscr{S})$ asks to first apply a selection predicate σ along the y dimension, followed by an aggregation AGG along the 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 correlated aggregates. Our main result is a general method that reduces the estimation of a correlated aggregate AGG to the streaming computation of AGG over an entire stream, for an aggregate that satisfies certain conditions. This results in the first sublinear space algorithms for the correlated estimation of a large family of statistics, including frequency moments. Our experimental validation shows that the memory requirements of these algorithms are significantly smaller than existing linear storage solutions, and that these achieve a fast per-record processing time. We also study the setting when items have weights. In the case when weights can be negative, 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. We complement this with a small space algorithm which uses a logarithmic number of passes.

1 Introduction

Consider a stream of tuples $\mathscr{S}=(x_i,y_i),i=1\ldots n$, where x_i is an item identifier, and y_i is a numerical attribute. A correlated aggregate query $C(\sigma,AGG,\mathscr{S})$ 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 $\mathscr{S}$, followed by the aggregation. Precisely,

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

A preliminary version of this article appeared in Proceedings of the 28th IEEE International Conference on Data Engineering (ICDE 2012), pages 162-173.

Srikanta Tirthapura
Iowa State University, E-mail: snt@iastate.edu. Supported in part by NSF CNS-0834743, CNS-0831903.

David P. Woodruff
IBM Almaden, E-mail: dpwoodru@us.ibm.com

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

We consider online processing of a data stream to support a query for a correlated aggregate, in cases where the selection predicate σ is not completely specified when data is being observed. A traditional streaming aggregation task, which we henceforth call "whole stream aggregation" asks for aggregates such as a frequency moments, quantiles, or heavy hitters over the entire stream, and a number of works have dealt with estimating these using limited memory (see, e.g., [1,18,21,24,23]). The major difference between whole stream aggregation and correlated aggregation is that in case of correlated aggregation, the scope of aggregation is restricted to only those items which satisfy the selection predicate. A summary for correlated aggregation must be prepared to answer queries over a substream that will be known only at query time, depending on the parameters supplied to the selection predicate. Correlated aggregates arise naturally in analytics on multi-dimensional streaming data, and space-efficient methods for implementing such aggregates are important in streaming analytics systems such as IBM Infosphere Streams.

A summary for correlated aggregation allows very flexible interrogation of a data stream. For example, consider a stream of IP flow records output by a network router equipped with Cisco's Netflow [28]. Suppose that we are interested in two attributes per flow record, the destination address, and the size (number of bytes) of the flow. Using a summary for correlated aggregate *AGG*, along with a whole stream quantile summary for the "size" dimension (any of the well known stream quantile summaries can be used here, including [21,27]), 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 flows whose size was more than the median flow size. (3) If the answer to the above was "interesting" 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 stream summary for correlated aggregates can be used for certain types of "drill down" queries into large streams. To get this flexibility, it is crucial that the selection predicate has parameters that can be specified at query time. Obviously, there is a limit to how much flexibility is allowed at query time. For instance, if an arbitrary selection predicate is allowed at query time, it will be possible to reconstruct the entire input stream using the summary, which implies that the summary cannot be small anymore.

We focus on selection predicates of the form $\sigma = (y \geq c)$ or $\sigma = (y \leq c)$ where c is specified at query time. We note this is still a broad and useful class of predicates; for instance it is possible to perform the sequence of queries described above, where the user can iteratively adapt her future queries based on the results of previous queries. This is a powerful tool for data analytics in applications such as network management or sensor data management.

1.1 Contributions

We study both small-space algorithms and space lower bounds for summarizing data streams for correlated aggregate queries.

<table>
<tr><td>General Aggregation Function</td><td colspan="2">Reduction to whole stream aggregation for any function that satisfies Conditions I to V (Section 2)</td></tr>
<tr><td rowspan="3">Frequency Moments, F_k, $k \geq 0$</td><td>Positive Weights Only</td><td>Sublinear Space Algorithms in a Single Pass (Section 3)</td></tr>
<tr><td rowspan="2">Positive and Negative Weights</td><td>Linear Space Lower Bound, constant passes (Section 4.1)</td></tr>
<tr><td>Sublinear Space Algorithm, logarithmic passes (Section 4.2)</td></tr>
</table>

Fig. 1: An overview of our results.

General Method for Correlated Aggregation. We present a general method for correlated aggregation for an aggregation function that satisfies a certain set of properties. For any such aggregation function *AGG*, we show how to reduce the construction of a sketch for correlated aggregation of *AGG* to the construction of a sketch for whole stream aggregation of *AGG*. This reduction allows us to use previously known streaming aggregation algorithms as a "black box" in constructing sketches for correlated aggregation.

We use this method to construct small space summaries for 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 two decades, starting with the work of Alon, Matias and Szegedy [1]. See, e.g., the references in [3]. Our algorithms are the first small-space (in fact, the first sub-linear in m space) algorithms for estimating correlated frequency moments with provable guarantees on the relative error. Our memory requirements are optimal up to factors that are logarithmic in m and the error probability δ, and factors polynomial in the relative error ε.

We also present the first space-efficient algorithms for correlated estimation 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 also give a technique for achieving a low amortized update time.

General Streaming Models and Single-Pass Lower Bounds. We next consider the case where stream items can have an associated positive or negative integer weight. Allowing negative weights is useful for analyzing the symmetric difference of two data streams, since 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.

Each stream element is a 3-tuple (x_i, y_i, z_i) where z_i specifies the weight of the item, and x_i, y_i are as before. 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 provides an accurate estimate of correlated aggregates in a single pass must use memory linear in the stream size*.

This is to be contrasted with whole stream estimation of frequency moments, where sublinear space single pass algorithms are known even in the presence of positive and negative weights on items [22].

Multipass Algorithms. 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 stored on a medium such as a tape, where it is possible to perform efficient sequential scans through the data using a machine whose working memory is very small when compared with the data size.

In the multipass 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.

Aggregation Over Asynchronous Streams. A closely related problem to correlated aggregation is that of streaming aggregation over a sliding window in the case when the elements of the stream arrive in an asynchronous (out-of-order) fashion. 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 stream elements are not observed by the processor in the 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 [15]. There is a straightforward reduction from the problem of aggregating over asynchronous streams to that of computing correlated aggregates, and vice-versa [31,6]. Hence, all of our results for (single-pass estimation) of 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 small-space algorithms for aggregation over asynchronous streams for a wide class of statistics.

Experimental Results. We present results of experiments evaluating 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 stream. Moreover, for a given accuracy requirement, the size of the sketch remains nearly constant and does not increase much with the stream size. Hence, these sketches are an effective way to summarize extremely large data sets and streams.

1.2 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. [10, 11,9]. Single pass computation of summaries for correlated aggregates on streams was first considered by Gehrke, Korn, and Srivastava [17], 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. [2] 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, Tirthapura, and Busch [31,30] considered the computation of sum and median over a sliding window of an asynchronous stream, and presented summaries that allowed the estimation of the sum and median within a small *relative* error with high probability. The above results also yield summaries for correlated sum with relative error guarantees, with the same space bounds. Cormode, Korn, and Tirthapura [12] presented algorithms for correlated quantiles and heavy hitters (frequent elements). The space complexity of these was subsequently improved by Chan et al. [7]. Further results on aggregate computation over an asynchronous stream include [13,14].

Significant prior work (for example, [15,4,19]) 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 correlated aggregates as we consider here – the problem is simpler since the data that is to be aggregated is always a contiguous subsequence, and in particular, a suffix of the entire stream. This structure allows one to design data structures (as in [15, 4,19]) that construct summaries over different fixed subsequences of the stream. A query is answered through constructing the union over a few such data structures. However, with correlated aggregates, the data to be aggregated may not appear as a contiguous subsequence of the stream, and hence the techniques used for synchronous sliding windows do not work.

Roadmap: We present a general method for estimating correlated aggregates in Section 2, and its application to frequency moments in Section 3. In Section 4 we present results on deletions, lower bounds, and multipass algorithms, and finally in Section 5 we present experimental results.

2 A General Method for Estimating Correlated Aggregates

We first consider a general method for estimating correlated aggregates for any 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, we use the union of sets to imply 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 c, 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 space-efficient estimation of $f(T,c)$ to space-efficient estimation of $f(R)$. We use the following definition of an (ε, δ) estimator.

Definition 1 Given parameters ε, δ, where $0 < \varepsilon < 1$, and $0 < \delta < 1$, an (ε, δ) estimator for a number Y is a random variable X such that with probability at least $1 - \delta$, $(1-\varepsilon)Y \leq X \leq (1+\varepsilon)Y$

In the following description, we use the term "sketching function" to denote a compression function on the input set with certain properties. More precisely, we say that f has a sketching function $sk_f()$ that takes three parameters υ, γ, R, where $0 < \upsilon < 1$, $0 < \gamma <$, and R is a multiset.

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 admit 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 [22]. In these examples, the sketching function is obtained by taking random linear combinations of the input.

We require the following conditions from f. 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 output is sensitive to small changes, the easier it is estimate correlated aggregates

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 $\varepsilon<1$, there exists a function $c_2^f(\varepsilon)$ with the following properties. For two sets A and B such that $B\subseteq A$, if $f(B)\leq c_2^f(\varepsilon)\cdot f(A)$, then $f(A-B)\geq(1-\varepsilon)f(A)$.
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 $sk_f^{cor}(\varepsilon,\delta,T)$ for estimating the correlated aggregate $f(T,c)$ with the following properties:

A. Using $sk_f^{cor}(\varepsilon,\delta,T)$, 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 $sk_f^{cor}(\gamma,\varepsilon,T)$, it is possible to construct $sk_f^{cor}(\gamma,\varepsilon,T\cup\{(x,y)\})$.

2.1 Algorithm Description

Let f_{max} denote an upper bound on the value 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:

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

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 $2y_{max}-1$ 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, $k(b) = sk_f(\upsilon, \gamma, S(b))$ 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 ← null set φ; Y_0 ← ∞;
2 for ℓ from 1 to ℓ_max do
3     S_ℓ is a set with a single element ⟨sk_f(·,·,φ), 0, y_max⟩;
4     Y_ℓ ← ∞;
```

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

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

where

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

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 $sk_f(\upsilon, \gamma, S(b))$. 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 $r(b)$.

2.2 Algorithm Correctness

Let S denote the stream of tuples observed so far. Suppose the required correlated aggregate is $f(S,c)$. 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 $[0,c]$.

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

if *There is a bucket b in S_0 such that $y \in span(b)$* **then**
  Update $k(b)$ by inserting x.
else
  Initialize a bucket $b = \langle sk_f(\upsilon,\gamma,\{x\}), y, y\rangle$, and insert b into S_0 ;
  if $|S_0| > \alpha$ **then**
    Discard the bucket $b \in S_0$ with the largest value of $l(b)$, say b^*, and update $Y_0 \leftarrow \min\{Y_0, l(b^*)\}$.
`// levels` $i > 0$
for *i from 1 to ℓ_{max}* **do**
  if $Y_i \leq y$ **then**
    return
  Let b be the bucket in S_i such that b is a leaf and $y \in span(b)$;
  if *b is open* **then**
    Insert x into $k(b)$;
    if $(est(k(b)) \geq 2^{i+1})$ *and* $(l(b) \neq r(b))$ **then**
      close bucket b
  else
    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 $k(b_1) = k(b_2) = sk_f(\nu,\gamma,\phi)$.
    If $y \in span(b_1)$ then insert x into $k(b_1)$. Otherwise insert x into $k(b_2)$;
    `/* Check for overflow */`
    if $|S_i| \geq \alpha$ **then**
      Let b' be the bucket in S_i with the largest value of attribute $l()$.;
      Discard b' from S_i;
      Update $Y_i \leftarrow l(b')$.

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

Let $\ell \in [0,\ldots,\ell_{max}]$ be the smallest level such that $Y_\ell > c$. If no such level exists, output FAIL.
if *ℓ is 0* **then**
  Answer the query using S_0 by summing over appropriate singletons
else
  Let B_1^ℓ be the set of all buckets b in level ℓ such that $span(b) \subseteq [0,c]$;
  `// Compose the Sketches`
  Let K be the composition of all sketches in $\{k(b) | b \in B_1^\ell\}$
  Return $est(K)$, the estimate of $f(B_1^\ell)$ gotten using the sketch K.

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 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 $\ell, 0 \leq \ell \leq \ell_{max}$. 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 $f(B) = f(S(B))$. 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 together).

Definition 2 Bucket b is defined to be "good" if $(1-\upsilon)f(b) \leq est_f(b) \leq (1+\upsilon)f(b)$. 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

$$\Pr[G] \geq 1 - \frac{\delta}{2}$$

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:

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

□

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:

$$f(B_1^\ell) \leq f(A) \leq f(B_1^\ell \cup B_2^\ell) \tag{1}$$

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

Lemma 3 *Conditioned on 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 $est(k(b)) \geq 2^{\ell_{max}+1}$. But $2^{\ell_{max}+1} > 2f_{\max}$, which means that $est(k(b))$ does not provide a $(1+\varepsilon)$-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 $\ell^* \geq 1$ and G is true, then*

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

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 $b \in B_2^{\ell^*}$, $est_f(b) \leq 2^{\ell^*+1}$ Because G is true, for every bucket $b \in B_2^{\ell^*}$, b is good, so that $f(b) \leq \frac{2^{\ell^*+1}}{1-\upsilon}$.

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:

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

Since $\upsilon \leq 1/2$, we get the desired result. □

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

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

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 $|I| \geq (\alpha-1)/2 - \log y_{max}$. Since G is true, we have that for any bucket $b \in I$, $f(b) \geq \frac{2^{\ell^*-1}}{1+\upsilon}$,

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 $1-\delta$:*

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

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 $\mathscr{E}_1$ 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:

$$\Pr[\mathscr{E}_1] \geq 1-\gamma \tag{2}$$

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

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

From Lemmas 4 and 5:

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

where we have substituted the value of α.

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

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

Using Property IV, we get the following:

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

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

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

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

To complete the proof of the theorem, note that

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

□

3 Frequency Moments F_k

3.1 $F_k, k \geq 2$

We first 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 $F_k(B) \leq (\varepsilon/(3k))^k F_k(A)$, then $F_k(A \cup B) \leq (1+\varepsilon)F_k(A)$.*

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

$$\begin{aligned}
&F_k(A\cup B) = \sum_{i=1}^n (a_i+b_i)^k \\
&= F_k(A)+F_k(B)+\sum_{i=1}^n\sum_{j=1}^{k-1}\binom{k}{j}a_i^j b_i^{k-j} \\
&= F_k(A)+F_k(B)+\sum_{j=1}^{k-1}\binom{k}{j}\sum_{i=1}^n a_i^j b_i^{k-j} \\
&\leq F_k(A)+F_k(B)+\sum_{j=1}^{j-1}\binom{k}{j}\left(\sum_{i=1}^n (a_i^j)^{k/j}\right)^{\frac{j}{k}}\left(\sum_{i=1}^n (b_i^{k-j})^{k/(k-j)}\right)^{\frac{k-j}{k}} \\
&= F_k(A)+F_k(B)+\sum_{j=1}^{k-1}\binom{k}{j}F_k(A)^{\frac{j}{k}}F_k(B)^{\frac{k-j}{k}} \\
&\leq F_k(A)+F_k(B)+\sum_{j=1}^{k-1}\binom{k}{j}F_k(A)\left(\frac{\varepsilon}{3k}\right)^{k-j} \\
&\leq (1+\varepsilon/3)F_k(A)+F_k(A)\sum_{j=1}^{k-1}\binom{k}{j}(\varepsilon/(3k))^{k-j} \\
&\leq (1+\varepsilon/3)F_k(A)+F_k(A)(1+\varepsilon/(3k))^k - F_k(A) \\
&\leq (1+\varepsilon/3)F_k(A)+F_k(A)(1+2\varepsilon/3) - F_k(A) \\
&\leq (1+\varepsilon)F_k(A),
\end{aligned}$$

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 $F_k(C) \leq (\varepsilon/(9k))^k F_k(D)$, 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

$$\begin{aligned}
F_k(D-C) &\geq F_k(D)/2^k - F_k(C) \\
&\geq ((9k/\varepsilon)^k(1/2^k)^k - 1)F_k(C) \\
&\geq (3k/\varepsilon)^k F_k(C)
\end{aligned}$$

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

Theorem 3 *For parameters $0 < \varepsilon < 1$ 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 $n^{1-2/k} poly(\varepsilon^{-1} \log(n/\delta))$.*

Proof From Lemma 6, we have $c_1^{F_k}(j) = j^k$. From Lemma 8, we have $c_2^{F_k}(\varepsilon) = (\varepsilon/(9k))^k$. Using these in Theorem 1, we get $c_1^{F_k}(\log y_{max}) = (\log y_{max})^k$, and $c_2^f(\varepsilon/2) = (\varepsilon/(18k))^k$. Using the sketches for F_2 from [1] and for $F_k, k > 2$ from [22], we get the above result. □

Remark 1 The space can be improved to $r^{1-2/k} poly(\varepsilon^{-1} \log(n/\delta))$, where r is the number of distinct x_i-values in the stream [22]. 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 $0 < \varepsilon < 1$ 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 $O(\varepsilon^{-4}(\log(1/\delta) + \log y_{max})(\log^2 y_{max})(\log^2 f_{max}))$ 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 $O((\log f_{max})(1/\varepsilon^2)\log(1/\delta))$ bits [1]. From the proof of Theorem 3, we have $c_1^{F_2}(j) = j^2$, and $c_2^{F_2}(\varepsilon) = (\varepsilon/18)^2$. Using the above in Theorem 1, we get the space to be $O(\varepsilon^{-4} \log^2 f_{max} \log^2 y_{max}(\log 1/\delta + \log y_{max}))$ 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 $O(\varepsilon^{-2} \log^2 y_{\max})$ buckets, each holding a sketch of $O(\varepsilon^{-2} \log f_{max}(\log 1/\delta + \log y_{max}))$ bits. We process a batch of $O(\varepsilon^{-2} \log^2 y_{\max})$ updates at once. We first sort the batch in order of non-decreasing y-coordinate. This can be done in $O(\varepsilon^{-2} \log^2 y_{\max}(\log 1/\varepsilon + \log\log y_{\max}))$ 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 [29], 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 $O(\varepsilon^{-2} \log^2 y_{\max}(\log 1/\delta + \log y_{max}))$. The time to update all the S_i is $O(\log f_{max})$ times this. So the amortized time is $O(\log f_{max}(\log 1/\delta + \log y_{max}))$. □

3.2 F_0: 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. 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 [20]. 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 [20]. The algorithm in [20] 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 [20] 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 [20], each level was a simple FIFO (first-in-first-out) queue.

Our algorithm takes advantage of the fact that the basic data structure in [20] 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 [20], 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 [16], as elaborated by Datar et al. [15]. We are however, not aware of any previous work applying these ideas to the context of correlated aggregates, or associated experimental results.

Theorem 4 *Given parameters $0 < \varepsilon < 1$ 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 $O(\log m + \log y_{max})\frac{\log m}{\varepsilon^2}\log 1/\delta$ 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 $|\{x|((x,y) \in S) \wedge (y \leq y_0)\}|$*

3.3 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 [5].

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, $0 < \varepsilon < \phi < 1$, 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 $|\{(x_i,y_i) \mid x_i = x \wedge y_i \leq c\}|^2 \leq (\phi-\varepsilon)F_2(c)$. 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 $(\varepsilon/10)\cdot 2^i$. See, e.g., the COUNTSKETCH algorithm of [8] for such an algorithm. To estimate the correlated F_2-heavy hitters, for each item we obtain an additive $(\varepsilon/10)\cdot F_2(c)$ 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 $\varepsilon/10$ 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 in [5].

4 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., [26].

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 $g : \{-n, -n+1, -n+2, \ldots, n-1, n\} \to \{0, 1, 2, \ldots, poly(n)\}$ 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, small-space approximation algorithm for this problem in the turnstile model (see Theorem 7).

4.1 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 [26], 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 5 *([25]) The t-round randomized communication complexity of* GREATER-THAN *is $\Omega(r^{1/t})$.*

Theorem 6 *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 $0 \leq \tau \leq y_{\max}$.

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 $1 \leq i \leq y_{\max}$. 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 $1 \leq i \leq y_{\max}$. 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 6 that ALG must use $\Omega(y_{\max}^{1/(2t-1)}/\log y_{\max}) = y_{\max}^{\Omega(1/t)}/\log y_{\max}$ bits of space.

4.2 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 12. We divide the interval

$[0, y_{\max}]$ of y-values into positions $p(0), p(1), p(2), \ldots, p(r)$, for some $r = O(\varepsilon^{-1}\log(nm))$. Ideally, the $p(i)$ are such that the $\hat{f}_{p(i)}$ satisfy

$$(1-\varepsilon)\cdot(1+\varepsilon)^i \leq f_{p(i)} \leq (1+\varepsilon)\cdot(1+\varepsilon)^i.$$

Given the output of MULTIPASS and a query τ, the QUERY-RESPONSE algorithm could then find the largest value of i for which $p(i) \leq \tau$ and output $(1+\varepsilon)^i$. Since the $p(i)$ represent (approximate) jumps in the function value by powers of $(1+\varepsilon)$, 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 $(1-\varepsilon)\cdot(1+\varepsilon)^i \leq f_j \leq (1+\varepsilon)\cdot(1+\varepsilon)^i$, 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

$$(1-\varepsilon)\cdot(1+\varepsilon)^i \leq f_{p(i)} \text{ and } f_{p(i)-1} \leq (1+\varepsilon)^i.$$

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 $p(i) \leq \tau$. It then outputs $(1+\varepsilon)^i$. Notice that if $p(i-\ell) = p(i-\ell+1) = \cdots = p(i)$, we indeed need to output $(1+\varepsilon)^i$, so it is important to find the largest i for which $p(i) \leq \tau$.

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 $f \leq \hat{f} \leq (1+\varepsilon)f$ 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 𝒜 be a classical streaming algorithm for (ε,δ')-approximating f, where δ' = δ/(y_max+1).
2  Fix the random string of 𝒜 for the rest of this algorithm.
3  In the first pass (ε,δ')-approximate f_{y_max} using 𝒜, obtaining estimate f̂_{y_max}.
4  Set r = ⌈log_{1+ε} f̂_{y_max}⌉.
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 𝒜, obtaining estimate f̂_{p(i)}.
10         If f̂_{p(i)} > (1+ε)^i, then p(i) ← p(i) − (y_max+1)/2^j, else p(i) ← p(i) + (y_max+1)/2^j.
11     If f̂_{p(i)} < (1+ε)^i, then p(i) ← p(i) + 1.
12 Output p(0), p(1), p(2), ..., p(r).
```

Theorem 7 *Algorithm* MULTIPASS *is an $O(\log y_{\max})$-pass, $O(\varepsilon^{-1}s(f,n,\varepsilon,\delta/(y_{\max}+1))\log(nm))$-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 12. It remains to argue correctness. We condition on the event $\mathscr{E}$ that, for every $\tau \in \{0,1,2,\ldots,y_{\max}\}$, algorithm $\mathscr{A}$ outputs an estimate $\hat{f}_\tau$ with $f_\tau \leq \hat{f}_\tau \leq (1+\varepsilon)f_\tau$. Since $\mathscr{A}$ fails with probability at most δ', it follows that $\Pr[\mathscr{E}] \geq 1-\delta$.

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

We first would like to argue that $f_{p(i)} \geq (1-\varepsilon)(1+\varepsilon)^i$. 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 $\hat{f}_{p(i)+1} > (1+\varepsilon)^i$. Now, once Step 12 is reached, if $\hat{f}_{p(i)} < (1+\varepsilon)^i$, then $p(i)$ is replaced with $p(i)+1$, and so after Step 12 we are guaranteed that $\hat{f}_{p(i)} \geq (1+\varepsilon)^i$. This means that

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

Second, we would like to argue that $f_{p(i)-1} \leq (1+\varepsilon)^i$. 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 $\hat{f}_{p(i)-1} \leq (1+\varepsilon)^i$, Now, once Step 12 is reached, if $\hat{f}_{p(i)} \geq (1+\varepsilon)^i$, then $p(i)-1$ remains the same. Otherwise, $p(i)-1$ is replaced with $p(i)$, and so after Step 12 we have $\hat{f}_{p(i)-1} < (1+\varepsilon)^i$. Hence, in either case,

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

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

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

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

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

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

Remark 2 For the case of $f = F_2$ and $m, n, y_{\max}$ being polynomially related, we get an $O(\varepsilon^{-3}\log^3 n)$-space $O(\log n)$-pass algorithm for $(\varepsilon, 1/n)$-approximation given a query point τ. This improves the $O(\varepsilon^{-4}\log^5 n)$-space complexity of our 1-pass algorithm for estimating F_2 with $1/n$ error probability, at the cost of additional passes.

5 Experimental Evaluation

5.1 Correlated F_2, Second Frequency Moment

Setup. We implemented our algorithm for correlated F_2 estimation in Python, with the goal of evaluating the scaling behavior of our algorithms for relatively large datasets. We used the following three datasets for our experiments on correlated F_2. (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 [29]. This variant gives a better update time than the original algorithm of Alon *et al.*.

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 2, 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. For all datasets that we considered, 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 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 3 (for $\varepsilon = 0.15$), Figure 4 (for $\varepsilon = 0.2$), and Figure 5 (for $\varepsilon = 0.25$). 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. 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).

5.2 Correlated F_0, Number of Distinct Elements

We implemented the algorithm for F_0 in Python. In addition to the three datasets described in Section 5.1, 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 with the datasets that were used for F_2 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 F_2.

The variation of the sketch size with ε is shown in Figure 6. 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

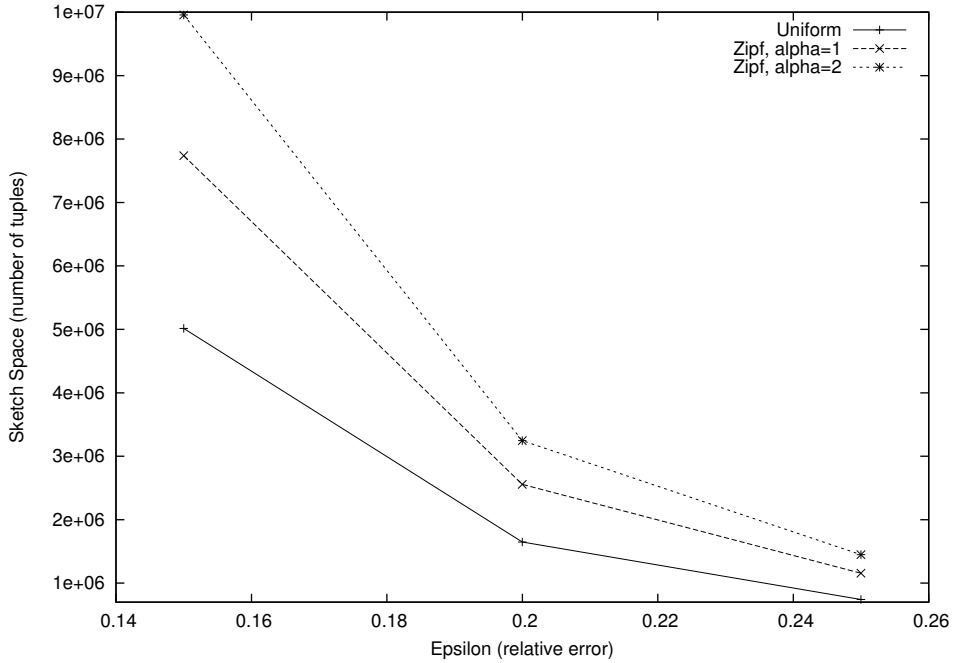

Fig. 2: F_2: Space taken by the sketch for versus relative error ε. The stream size is 40 million, for all datasets.

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 7, for $\varepsilon = 1$. 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.

References

1. N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58(1):137–147, 1999.
2. 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*, 15(3):569–572, 2003.
3. A. Andoni, R. Krauthgamer, and K. Onak. Streaming algorithms via precision sampling. In *Proceedings Foundations of Computer Science (FOCS)*, pages 363–372, 2011.
4. V. Braverman and R. Ostrovsky. Effective computations on sliding windows. *SIAM Journal on Computing.*, 39(6):2113–2131, 2010.
5. Vladimir Braverman, Ran Gelles, and Rafail Ostrovsky. How to catch l_2-heavy-hitters on sliding windows. In *Proceedings 19th International Conference on Computing and Combinatorics (COCOON)*, pages 638–650, 2013.

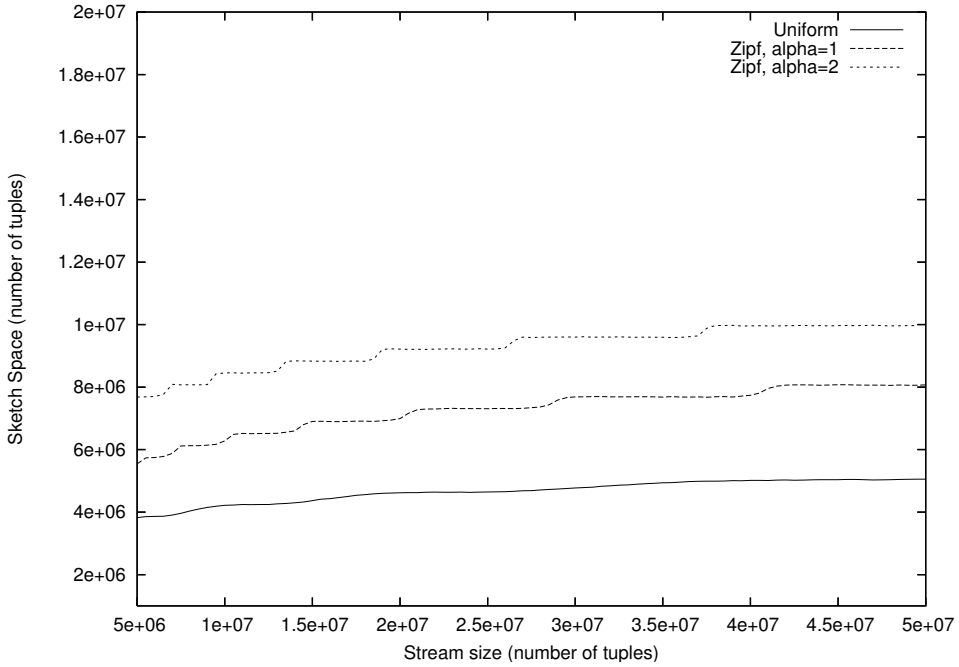

Fig. 3: F_2: Space taken by the sketch versus stream size n, $\varepsilon = 0.15$.

6. C. Busch and S. Tirthapura. A deterministic algorithm for summarizing asynchronous streams over a sliding window. In *24th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 465–476, 2007.
7. Ho-Leung Chan, Tak Wah Lam, Lap-Kei Lee, and Hing-Fung Ting. Approximating frequent items in asynchronous data stream over a sliding window. In *Workshop on Approximation and Online Algorithms (WAOA)*, pages 49–61, 2009.
8. M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. *Theor. Comput. Sci.*, 312(1):3–15, 2004.
9. D. Chatziantoniou. Ad Hoc OLAP: Expression and Evaluation. In *Proceedings of the 15th International Conference on Data Engineering (ICDE)*, page 250, 1999.
10. D. Chatziantoniou, M. O. Akinde, T. Johnson, and S. Kim. The MD-join: An Operator for Complex OLAP. In *Proceedings of the 17th International Conference on Data Engineering (ICDE)*, pages 524–533, 2001.
11. D. Chatziantoniou and K. A. Ross. Querying Multiple Features of Groups in Relational Databases. In *Proceedings of 22th International Conference on Very Large Data Bases (VLDB)*, pages 295–306, 1996.
12. G. Cormode, F. Korn, and S. Tirthapura. Time-decaying aggregates in out-of-order streams. In *Proceedings of the 24th International Conference on Data Engineering (ICDE)*, pages 1379–1381, 2008.
13. G. Cormode, S. Tirthapura, and B. Xu. Time-decaying sketches for sensor data aggregation. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 215–224, 2007.
14. G. Cormode, S. Tirthapura, and B. Xu. Time-decaying sketches for robust aggregation of sensor data. *SIAM Journal on Computing*, 39(4):1309–1339, 2009.
15. M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. *SIAM Journal on Computing*, 31(6):1794–1813, 2002.
16. P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences*, 31:182–209, 1985.
17. J. Gehrke, F. Korn, and D. Srivastava. On computing correlated aggregates over continual data streams. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 13–24, 2001.

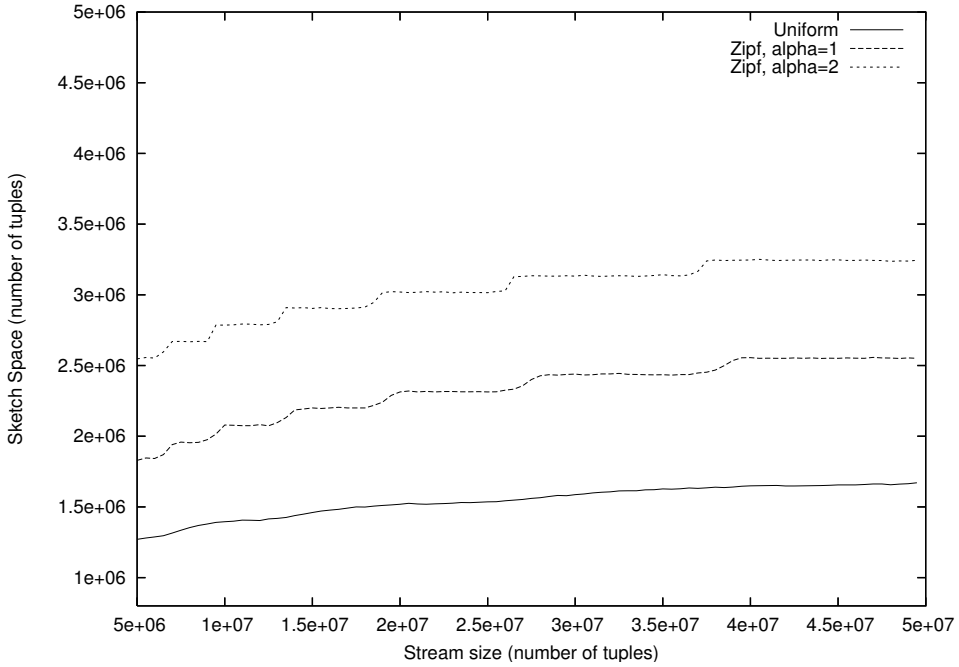

Fig. 4: F_2: Space taken by the sketch versus stream size n, $\varepsilon = 0.2$.

18. P. Gibbons and S. Tirthapura. Estimating simple functions on the union of data streams. In *Proceedings ACM Symp. on Parallel Algorithms and Architectures (SPAA)*, pages 281–291, 2001.
19. P. Gibbons and S. Tirthapura. Distributed streams algorithms for sliding windows. In *Proceedings of the Fourteenth Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 63–72, 2002.
20. P. Gibbons and S. Tirthapura. Distributed streams algorithms for sliding windows. *Theory of Computing Systems*, 37:457–478, 2004.
21. M. Greenwald and S. Khanna. Space efficient online computation of quantile summaries. In *Proceedings ACM International Conference on Management of Data (SIGMOD)*, pages 58–66, 2001.
22. 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)*, pages 202–208, 2005.
23. D. M. Kane, J. Nelson, and D. P. Woodruff. An optimal algorithm for the distinct elements problem. In *Proceedings of the Twenty-Ninth ACM Symposium on Principles of Database Systems (PODS)*, pages 41–52, 2010.
24. L.K. Lee and H.F. Ting. A simpler and more efficient deterministic scheme for finding frequent items over sliding windows. In *Proceedings of the Twenty-Fifth ACM Symposium on Principles of Database Systems (PODS)*, pages 290–297, 2006.
25. P. B. Miltersen, N. Nisan, S. Safra, and A. Wigderson. On data structures and asymmetric communication complexity. *J. Comput. Syst. Sci.*, 57(1):37–49, 1998.
26. S. Muthukrishnan. *Data Streams: Algorithms and Applications*. Foundations and Trends in Theoretical Computer Science. Now Publishers, August 2005.
27. N. Shrivastava, C. Buragohain, D. Agrawal, and S. Suri. Medians and beyond: new aggregation techniques for sensor networks. In *Proceedings of the 2nd ACM International Conference on Embedded Networked Sensor Systems (SenSys)*, 2004.
28. Cisco Systems. Cisco IOS Netflow. `http://www.cisco.com/web/go/netflow`.
29. 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)*, pages 615–624, 2004.

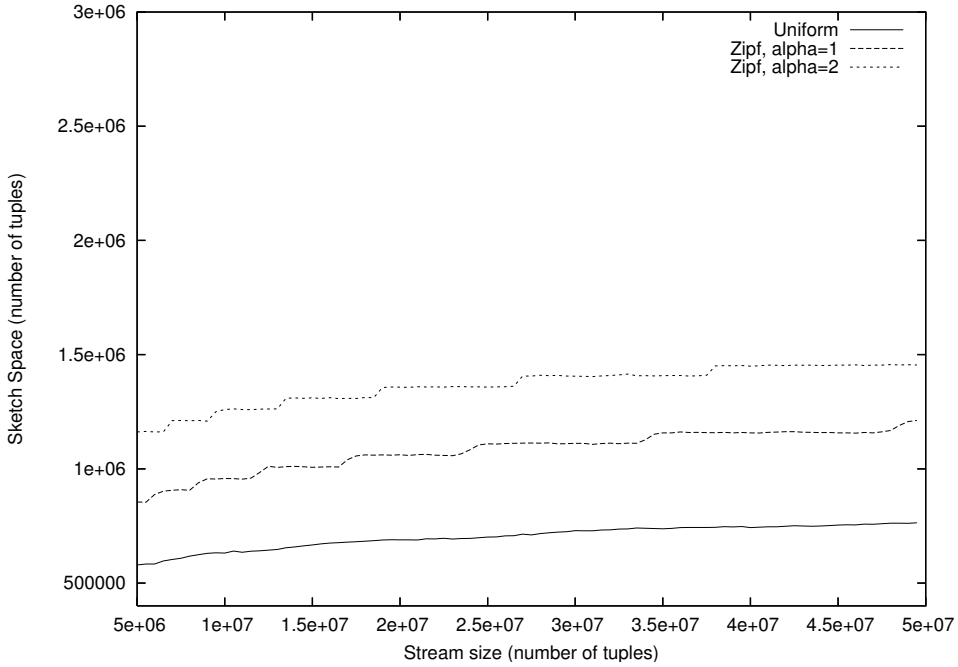

Fig. 5: F_2: Space taken by the sketch versus stream size n, $\varepsilon = 0.25$.

30. S. Tirthapura, B. Xu, and C. Busch. Sketching asynchronous streams over a sliding window. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 82–91, 2006.
31. B. Xu, S. Tirthapura, and C. Busch. Sketching asynchronous data streams over sliding windows. *Distributed Computing*, 20(5):359–374, 2008.

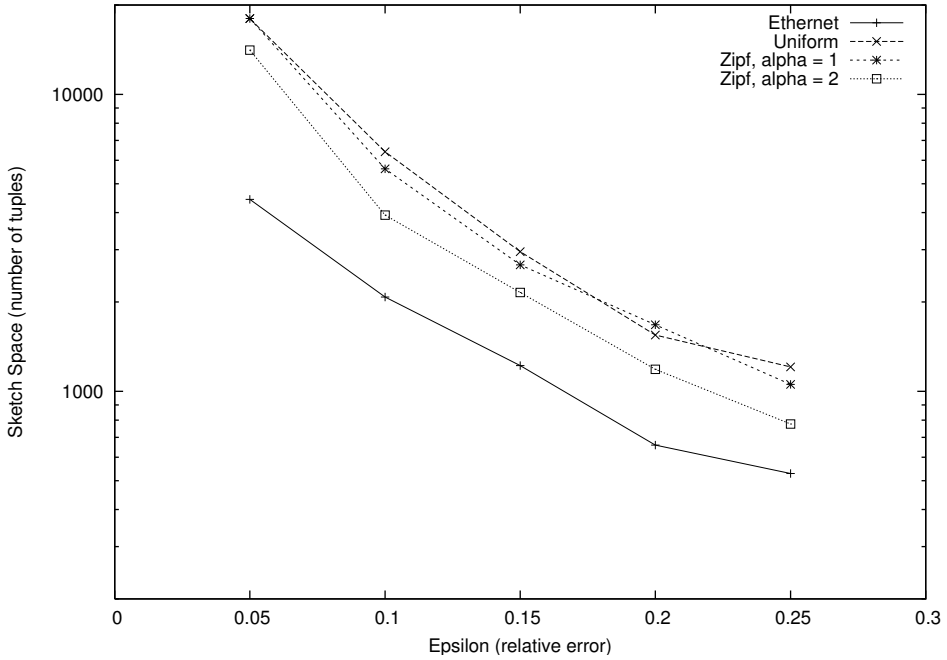

Fig. 6: F_0: Space taken by the sketch versus relative error ε. The stream size is 2×10^6, for all datasets.

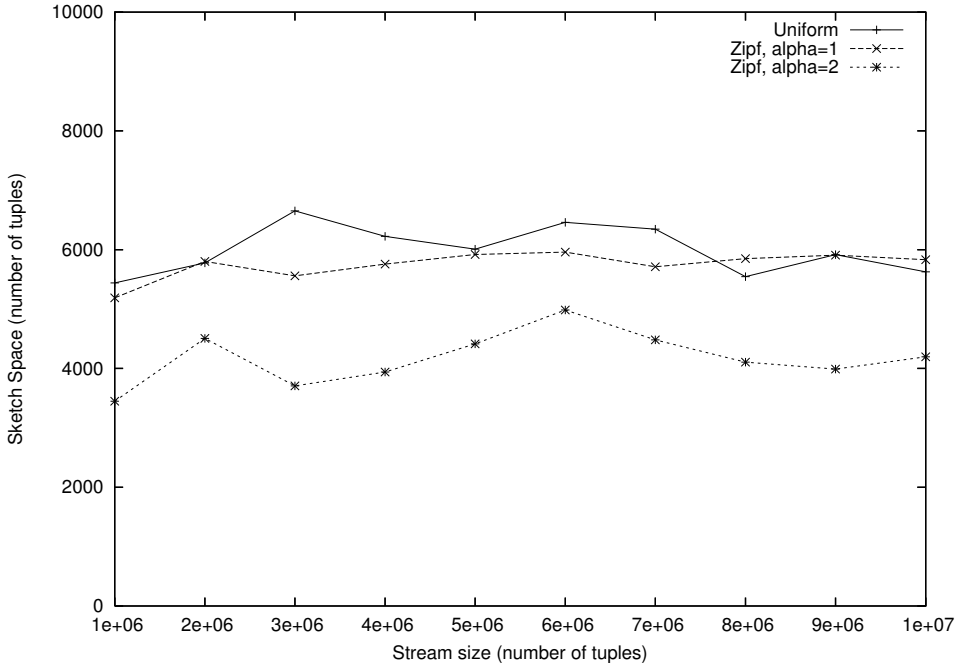

Fig. 7: F_0: Space taken by the sketch versus stream size n, for the Uniform and Zipfian datasets. $\varepsilon = 0.1$.