Estimating Simple Functions on the Union of Data Streams

Phillip B. Gibbons
Information Sciences Research Center
Bell Laboratories
Murray Hill, NJ 07974
gibbons@research.bell-labs.com

Srikanta Tirthapura
Computer Science Department
Brown University
Providence, RI 02912
snt@cs.brown.edu

ABSTRACT

Massive data sets often arise as physically distributed, parallel data streams. We present algorithms for estimating simple functions on the union of such data streams, while using only logarithmic space per stream. Each processor observes only its own stream, and communicates with the other processors only after observing its entire stream. This models the set-up in current network monitoring products. Our algorithms employ a novel *coordinated sampling* technique to extract a sample of the union; this sample can be used to estimate aggregate functions on the union. The technique can also be used to estimate aggregate functions over the distinct "labels" in one or more data streams, e.g., to determine the zeroth frequency moment (i.e., the number of distinct labels) in one or more data streams. Our space and time bounds are the best known for these problems, and our logarithmic space bounds for *coordinated* sampling contrast with polynomial lower bounds for *independent* sampling. We relate our distributed streams model to previously studied non-distributed (i.e., merged) streams models, presenting tight bounds on the gap between the distributed and merged models for deterministic algorithms.

1. INTRODUCTION

This paper considers the following distributed setting for massive data sets. There are two parallel streams $A = \{a_1, a_2, \dots, a_n\}$ and $B = \{b_1, b_2, \dots, b_n\}$ of data items. Alice observes the $\{a_i\}$ in order, and Bob observes the $\{b_i\}$ in order. Alice and Bob each have some limited amount of workspace, $w \ll n$ bits, to be used while observing their respective streams. After observing their streams, Alice and Bob send the contents of their workspace to a Referee, who estimates a function F on A and B, without seeing either stream. Note that Alice and Bob are not allowed to communicate with each other directly. An example function is given in Figure 1.

We are interested in minimizing: (1) the total workspace used by Alice and Bob, and (2) the time taken by Alice and Bob to process each data item. More generally, we have $t \geq 2$ parties, each observing their respective data streams, and a Referee estimating a function on the streams. We denote this setting as the *distributed streams* model.

Input to Alice: $A = \{a_1, \dots, a_n\}$, a stream of bits.
Input to Bob: $B = \{b_1, \dots, b_n\}$, a stream of bits.
Referee to estimate: $U(A, B) = \sum_{i=1}^{n}(a_i \vee b_i)$.

Application: The bits correspond to the characteristic vectors of two sets A^* and B^* over the same domain of size n. The function $U(A, B) = |A^* \cup B^*|$.

Figure 1: The union function U

This model is motivated by the monitoring and characterization of Internet traffic. Monitoring devices in the network observe a stream of packets. Each device has a small workspace in which to store information on its observed stream, and the contents are periodically sent to a central data analyzer, in order to compute aggregated statistics on the streams. This set-up is used, e.g., in Lucent's Interpret-Net and Cisco's NetFlow network monitoring products.

The distributed streams model combines features of both streaming models (e.g., [4, 8, 9, 11, 13, 14, 16, 18]) and communication complexity models (e.g., [20, 21]). As in streaming models (but not communication complexity models), the data is observed once, and we seek to minimize the workspace and the time to process each data item. However, more in line with *simultaneous, 1-round* communication complexity models [20, 23] and sketch models (e.g., [3, 5, 8]), the input is shared among multiple parties who communicate only by sending a message to a referee, who then computes/estimates the function.

Often, it is important to consider aggregate functions on the *union* of the data streams. For example, the same logical stream of data from a source to a destination in a virtual private network is frequently split among multiple paths in the network, in order to improve throughput and reliability. Network monitoring devices along each path observe only the divided traffic. Yet, to understand the logical stream, the characterization must be done on the *union* (not the sum) of the constituent streams. More generally, a natural partitioning of the data items exists, e.g., by (discretized) time, by source/destination, etc., such that items within the same partition must be specially aggregated across all streams. For example, we would like to compute the number of distinct IP destinations across all streams.

In this paper, we study the problem of estimating simple functions on the union of data streams. We show how a technique we call *coordinated 1-sampling* can be effectively used to estimate simple monotone functions in the distributed streams model.

We demonstrate the technique by focusing first on the union function U in Figure 1, in which each data item a_i and b_i is a single bit and $U(A,B) = \sum_{i=1}^{n}(a_i \vee b_i)$ is the number of 1's in the bitwise OR of the two streams. Exact computation of U would solve the set disjointness problem, and hence requires $\Omega(n)$ workspace, even for randomized algorithms [21]. Because for massive data streams, linear workspace is unacceptably large (see, e.g., [13]), we instead seek to *approximate* U to within a small relative error. Our goal is to obtain an (ϵ,δ)-approximation scheme, defined as follows:

DEFINITION 1. *An (ϵ,δ)-approximation scheme for a quantity X is a (randomized) procedure that, given any positive $\epsilon < 1$ and $\delta < 1$, computes an estimate $\hat{X}$ of X that is within a relative error of ϵ with probability at least $1-\delta$, i.e.,* $\mathbf{Pr}\left\{|\hat{X} - X| \leq \epsilon X\right\} \geq 1-\delta$.

We present two (ϵ,δ)-approximation schemes for U that use $O(\frac{\log(1/\delta)\log n}{\epsilon^2})$ workspace. The first is for the *public coins* setting [20], in which Alice and Bob have access to the same random string of unbiased and fully independent bits. This scheme is the simpler of the two, and requires only constant time to process each data item. Note that in practice, it may suffice to simulate the public coins in the algorithm by having Alice and Bob use the same pseudo random number generator, with the same starting seed of $\Theta(\log n)$ bits. We will refer to this model as the *distributed streams model with public coins.*

Our second approximation scheme is for a *stored coins* setting, in which Alice and Bob have a shared string of unbiased and fully independent random bits, but these bits must be stored at Alice and Bob prior to observing their inputs. The space to store these bits must be accounted for in their workspace bound. This scheme is more involved, as we need to provide an explicit construction by each party of the needed random bits from a small random string, and analyze the scheme under the limited independence of the constructed bits. The time to process each data item is dominated by the time required to perform $O(\log(1/\delta))$ multiplications over a finite field of $\Theta(\log n)$ bits. We will refer to this model as the *distributed streams model with stored coins.* Previous works on streaming models (e.g., [4, 8, 9, 11, 16, 18]) have studied settings with stored coins. Stored coins differ from private coins (e.g., as studied in communication complexity [20, 22, 23]) in that the same random string can be stored in all parties.

We show that our sampling technique has wider applicability in streaming models by using it in an (ϵ,δ)-approximation scheme for F_0 (the number of distinct values) of a sequence. Given a sequence $A = \{a_1, \ldots, a_n\}$ where each item a_i is an integer value in $[1..m]$, we present an (ϵ,δ)-approximation scheme for F_0 that uses only $O(\frac{\log(1/\delta)\log m}{\epsilon^2})$ bits, and the time required to process each item is dominated by the time required to perform $O(\log(1/\delta))$ multiplications over a finite field of $\Theta(\log m)$ bits. We note that this is an interesting result in itself because of the importance of the F_0 function in database optimization (see, e.g., [6, 15, 26]) and in internet traffic analysis, e.g., the number of distinct web pages requested, or the number of distinct visitors to a website. This problem has been studied in [4, 10] and elsewhere, but we do not know of an (ϵ,δ)-approximation scheme for F_0 whose bounds match the bounds we obtain.

Our logarithmic space bounds for union and for F_0 using *coordinated* sampling contrast with $\Omega(\sqrt{n})$ and $\Omega(\sqrt{m})$ lower bounds we present for union and F_0, respectively, using *independent* sampling on each stream.

We show how to extend our scheme to the scenario studied in [8, 18] and elsewhere in which data items are (label,value) pairs. Unlike previous work on data streams, our sampling technique obtains a random sample of the (distinct) labels in a stream (or in the union of streams) of a desired target size β in $O(\beta)$ workspace, and could be used to estimate other functions which are well-estimated with a sample of size β. Moreover, for labels in the sample that are in more than one stream, the referee learns the respective values from each stream. Thus we can estimate aggregate functions over the values associated with distinct labels in one or more data streams, e.g., the variance in request sizes averaged over all distinct destinations. As epitomized by the TPC benchmarks [25], which are the primary industry benchmarks for large scale query processing, many queries and reports seek aggregates of values over the distinct "labels" or *groups* of a data set.

Next, we consider the previously-studied streams model in which there is only one party, who observes both streams, and the streams are interleaved in an arbitrary order by an adversary (e.g., [8, 11, 18]). We consider the obvious generalization of this model to $t > 2$ streams, and refer to the model as the *merged streams* model. We present a comparison of the relative power of the distributed and merged streams models. We show that for any function f, the deterministic merged stream complexity (i.e., space bound) is within a factor of t of the deterministic t-party distributed stream complexity, and that this gap is existentially tight. We show similar results for both exact computation of f and for approximating f to within either an absolute or relative error ϵ. It follows that deterministic merged streams algorithms can be designed assuming that the streams are not interleaved, at a penalty of at most t (which is viewed as a small constant). The comparison for randomized complexities remains open.

The remainder of the paper is organized as follows. Section 2 presents preliminary results and comparisons with previous related work. Section 3 presents our coordinated 1-sampling technique and its application in computing the union function U. Extensions to the basic method and its applications are discussed in Section 4. Our results comparing distributed and merged stream models are in Section 5, with conclusions following in Section 6.

2. PRELIMINARIES AND RELATED WORK

In this section, we first discuss work on related models and then present some preliminary results for both the union function and for the number of distinct values.

Data streams and sketches. Previous work on data streams has studied:

- approximating a function such as the kth frequency moment on a *single stream* observed by a *single party* (e.g., [3, 4, 12, 16]), and
- approximating a function such as the L^p-difference on

two streams observed by a *single party* (e.g., [8, 9, 11, 18]).

As indicated above, in these models, the single party observes either a single stream or two arbitrarily-interleaved streams, and then estimates a given function f on the data (f is known prior to observing the streams). The goal is to design an (ϵ, δ)-approximation scheme for f that minimizes the workspace used and the time to process each item in the sequence. The single party, two streams model can obviously be generalized to $t > 2$ streams, where the interleaving of the streams is controlled by an adversary. As indicated above we refer to this model as the *merged streams* model. The small-space data structures maintained by a party are called *synopses* or *sketches*.

A related, more offline model is the *sketch model* [3, 5, 8]. Each of t parties observes a distinct data set, which it processes offline (i.e., it has random access to its entire data set and it can use as much time and space as needed). Each party, however, must produce a small-space synopsis or sketch, and only these sketches can be used to compute/estimate a given function f. Previous work on the sketch model (e.g., [3, 5, 8, 18, 19]) has studied approximation schemes for functions such as the set resemblance, the frequency moments, the L^p-difference, and the (database) join size.

Frameworks for studying data synopses, beyond the data stream and sketch models, were presented in [13], along with a survey of results.

As has been observed in many of these works, algorithms designed for one of the above models are often suited for other such models as well. The same can be said for the distributed streams model. For example, an (ϵ, δ)-approximation scheme for a function f in the distributed streams model trivially implies an (ϵ, δ)-approximation scheme for f in the sketch model, with the same space bound. On the other hand, algorithms designed for a single party are sometimes ill-suited for the distributed streams model, e.g., the kth frequency moment ($k \geq 3$) algorithm of [4] requires counting at Bob the number of occurrences of labels known only to Alice. This paper presents several results relating the complexity of various streams models.

Communication complexity. In communication complexity models [21], unlike in streams models, the parties have *unlimited* time and space with which to process their respective inputs. One-way communication complexity results can often be related to the merged streams model, whereas simultaneous, 1-round communication complexity results can often be related to the distributed streams model. Both public/common and private coin communication complexity models have been studied (e.g., [22, 23]). For streaming models and their applications, the dichotomy is between public coins and stored (common) coins; this dichotomy does not arise in communication complexity, due to the absence of workspace constraints.

Parallel and distributed algorithms. Unlike traditional work in parallel and distributed algorithms (which is not targeted for the network monitoring environment), in the distributed streams model the parties/processors communicate only before and after reading their input. Moreover, each processor has limited (sublinear) workspace and is permitted to read its input only once.

In the remainder of this section, we will focus on two specific functions considered in this paper: the union function and the number of distinct values.

Union. In the distributed streams model, $\sum_i a_i$ and $\sum_i b_i$ can be trivially computed exactly and deterministically by Alice and Bob, respectively, using $\log n$ bits each. Thus $\frac{4}{3}$ times the maximum of these two is an approximation for the union, with space complexity $\log n$ and relative error $\epsilon = \frac{1}{3}$. Thus our focus will be on (ϵ, δ)-approximation schemes for $\epsilon \ll \frac{1}{3}$.

For $t = 2$ streams, the identity $2|\sum_i (a_i \vee b_i)| = |\sum_i a_i| + |\sum_i b_i| + |\sum_i (a_i \bigtriangleup b_i)|$ can be used to obtain an (ϵ, δ)-approximation scheme for U for the distributed streams model with stored coins from the (ϵ, δ)-approximation scheme for the size of the symmetric difference, $|\sum_i (a_i \bigtriangleup b_i)|$, in [8]. The symmetric difference algorithm is presented for the merged streams model in [8], but trivially extends to the distributed streams model with stored coins. The resulting scheme uses the same space bound as the scheme we present, but the time to process each data item is worse: it is dominated by the time required to perform $O(\log(1/\delta)/\epsilon^2)$ multiplications over a finite field of $\Theta(\log n)$ bits.[1] Our algorithm uses completely different techniques and is considerably simpler. Moreover, while our scheme extends trivially to more than two streams, it is not known how to extend the scheme in [8] to estimate the union function U of $t > 2$ streams.

A crucial factor in our approach is that Alice and Bob's sampling is *coordinated*:

THEOREM 1. *Consider an algorithm for the distributed streams model in which Alice and Bob each send the referee a random sample of (the 1-bits in) their observed stream, where the sampling at Alice is independent of the sampling at Bob. Then any (possibly randomized) estimator for U by the referee that, with probability $\frac{2}{3}$, is within $\epsilon < \frac{1}{3}$ relative error requires $\Omega(\sqrt{n})$ sample sizes.*

PROOF. Consider an input in which Alice and Bob each have $\frac{n}{2}$ 1-bits and $\frac{n}{2}$ 0-bits. In scenario 1, there is no overlap in the positions of the 1-bits, so that $U = \frac{n}{2} + \frac{n}{2} = n$. In scenario 2, there is a complete overlap in the positions of the 1-bits, so that $U = \frac{n}{2}$. We assume that Alice and Bob both send the referee an independent sample of the positions of their 1-bits: extending the proof to handle a mix of 0-bits and 1-bits in a sample is straightforward (and it does not improve the estimator). In scenario 1, there is obviously no overlap in the positions sent to the referee. But also in scenario 2, there is only a very small probability of any overlap, when only $o(\sqrt{n})$ positions are sent by Alice and Bob to the referee. Thus, with high probability the referee cannot distinguish between scenarios 1 and 2. Therefore, the referee must have a deterministic or randomized procedure for outputting an estimate, which either at most half the time outputs an estimator $\geq \frac{2n}{3}$, or at most half the time outputs an estimator $\leq \frac{2n}{3}$. In the former case (latter case), it is within $\frac{1}{3}$ relative error for scenario 1 (scenario 2, respectively) at most half the time. □

Thus *uncoordinated* sampling requires $\Omega(\sqrt{n})$ workspace.

Distinct counting. Consider the zeroth frequency moment (F_0) of a sequence of n items in $[1..m]$, where $m \leq n$. Estimating this function has been studied in the context

[1]Note that the time per item is critical in practice, due to the extremely high network traffic rate.

of a single stream for both public coins [7, 10, 26] and stored coins [4]. These previous algorithms trivially extend to the distributed streams model, but the space and/or time bounds for an (ϵ, δ)-approximation scheme for F_0 are worse than our algorithm's bounds. The best previous bounds are due to Cohen [7], which matches our space bound, but its time bound is $\Theta(\frac{1}{\epsilon^2})$ worse, and its guarantees are only for the public coins model. The previous algorithms use probabilistic counting instead of sampling: only the estimate is obtained, not a sample that can be used for multiple estimation problems.

Note that computing the union U is simply a special case of computing F_0, in which all items *within* a stream are distinct. Thus the $\Omega(\sqrt{n})$ lower bound on *uncoordinated* sampling (Theorem 1) extends to F_0. More precisely, it extends to an $\Omega(\sqrt{m})$ bound, which is $\Omega(\sqrt{n})$ whenever m is $\Theta(n)$. Whereas, as discussed above, estimating U becomes nontrivial only for $\epsilon < \frac{1}{3}$, the presence of duplicates makes estimating F_0 to within *any* constant factor nontrivial, even for a single stream.

3. COORDINATED 1-SAMPLING

Our algorithm for estimating U is based on the following random sampling procedure. Given a p between 0 and 1, Alice and Bob decide on a randomly chosen subset $S(p)$ of $\{1, \ldots, n\}$ as the positions at which to sample their bit streams. $S(p)$ is created by selecting each number in $[1..n]$ independently with probability p. Both Alice and Bob know $S(p)$. Alice and Bob sample their streams at positions specified by $S(p)$ and store only those positions where 1-bits occur. Alice collects $\{i | i \in S(p) \wedge a_i = 1\}$ and Bob $\{i | i \in S(p) \wedge b_i = 1\}$. Our estimate for U is $Y = \frac{1}{p}\sum_{i \in S(p)}(a_i \vee b_i)$. By Chernoff bounds,

$$\mathbf{Pr}\{Y \notin (1-\epsilon, 1+\epsilon)U\} < 2e^{-Up\epsilon^2/2} \qquad (1)$$

By storing only the 1-bits, the expected space used is $O(U \cdot p \log n)$, because the positions of an expected Up items have to be stored (instead of np items), and storing each of them takes $\log n$ bits.

If we choose the sampling probability p such that the product $U \cdot p$ is about $\frac{\log(1/\delta)}{\epsilon^2}$, then we are assured of getting a good estimate with high probability and moreover, the space complexity is low. Specifically, setting $U \cdot p = \frac{2\log(2/\delta)}{\epsilon^2}$ implies by equation 1 that $\mathbf{Pr}\{Y \notin (1-\epsilon, 1+\epsilon)U\} < \delta$. The problem, of course, is that Alice and Bob do not know U and hence cannot decide on the right value of p beforehand. To solve this, we adapt the sampling probability as the algorithm proceeds. Each party starts off by sampling with probability 1. If the current sampling probability at a party causes the stored sample to become too large, then the probability is halved and the sampling proceeds. This way, we ensure that the space taken by the sample is never too large, and also, as we prove later, the accuracy of our estimate meets the required criteria. Two questions arise here:

1. How does a party switch to a lower probability of sampling? When we decide to do that, some bits of the stream have already flown by, and we cannot observe them anymore. We must ensure that these bits are not needed in our sample. To ensure this, we select $S(p/2)$ from $S(p)$ by including each element with probability $\frac{1}{2}$. Thus all the sample points required by the new sample, $S(p/2)$, up through the current data stream item are already there in the stored sample associated with $S(p)$.

2. What if the two parties are sampling their respective streams at different probabilities and hence, at different positions? This is entirely possible since each party makes the decision to change the sampling probability independently of the other, based on the number of 1-bits in their stream in the selected positions. But we know the following: if Alice ends at probability p_A and Bob at p_B, and say $p_A \leq p_B$, then $S(p_A) \subseteq S(p_B)$. So, the referee can determine a sample of the sequence $\{a_i \vee b_i\}$ resulting from a sampling probability of p_A. A key property we are exploiting here is that U is monotone, so that when Alice targets $p \cdot \sum a_i \approx \frac{\log(1/\delta)}{\epsilon^2}$ and Bob targets $p \cdot \sum b_i \approx \frac{\log(1/\delta)}{\epsilon^2}$, we have that $p \cdot U \approx r\frac{\log(1/\delta)}{\epsilon^2}$ for $1 \leq r \leq 2$.

3.1 Public Coins Scheme

We now present our algorithm for public coins. We assume that Alice and Bob have access to a hash function $e : \{1, 2, \ldots, n\} \to \{1, 2, \ldots, d = \log n\}$ such that for each $i \in \{1, 2, \ldots, n\}$, $e(i)$ is a random variable distributed as $\mathbf{Pr}\{e(i) = j\} = 1/2^j, j = 1 \ldots d-1$. We further assume that these random variables are i.i.d.[2] In Section 3.2, we show how to relax these assumptions, at a cost to the processing time per bit. The random sample at probability $1/2^l$ is the set of indices which have been hashed to a value greater than or equal to l, i.e., $S(1/2^l) = \{i | e(i) \geq l\}$.

The algorithm is depicted in Figure 2, presenting the steps for Alice, Bob, and the referee. We describe the algorithm for Alice. (The algorithm for Bob is symmetric.) Let $\alpha = \frac{\log(3/\delta)}{\epsilon^2}$, and let $c = 84$, a constant determined from the analysis[3]; $c \cdot \alpha$ is the bound on the sample size. At any stage in the execution, Alice is at a particular "level" l which is related to her current sampling probability. At level l, she is sampling with a probability of $\frac{1}{2^l}$. She maintains the set $\{i | (a_i = 1) \wedge (i \in S(1/2^l))\}$, thus implicitly maintaining the value of a_i for every position in $S(1/2^l)$. She starts at level 0; the level may increase as the computation proceeds. S is her sample. When $|S|$ overflows the sample size bound, Alice increases her level by 1 and discards sample points that do not belong to the current level. After she is done observing her data stream, Alice sends her level ℓ and her sample S to the referee. After receiving the level and sample from both Alice and Bob, the referee computes the estimate.

An example is given in Figure 3, for two data streams of $n = 16$ bits. In this example, Alice and Bob each have space to store 4 integers (each integer is a position in the bit stream). Note that Alice and Bob can also store the hash value $e(i)$ for each stored position i, increasing the workspace by only a $o(1)$ factor.

THEOREM 2. *For any positive $\epsilon \leq 1$ and $\delta \leq 1$, the above algorithm is an (ϵ, δ)-approximation scheme for U, the size of the union of two bit streams, in the distributed streams model with public coins. The algorithm uses $O(1)$ worst case expected time to process each item, and a total of $O(\frac{\log(1/\delta)\log n}{\epsilon^2})$ bits of workspace.*

[2]Technically, we need to access the ith $\log n$ bit word w in the public random string of uniform bits, and then compute $e(i)$ as the largest j such that the j most significant bits of w are all 0.

[3]We have not attempted to minimize the constants in our analysis.

```
Alice: Initialization: $l \leftarrow 0$, $S \leftarrow \emptyset$.
Upon receiving $a_i$:
    • If ($a_i = 1$) and ($e(i) \geq l$) then    // If belongs in $S$
        // Store the positions of the 1's in $S$. Position $i$ is
        // in the sample until the value of $l$ exceeds $e(i)$.
        - $S \leftarrow S \cup \{(i, e(i))\}$
    • If $|S| > c\alpha$ then    // If the sample is now too large
        // Shift to a higher level, i.e., a lower probability
        - $l \leftarrow l + 1$
        // Discard sample points that no longer belong
        - $S \leftarrow S - \{(i, l-1)\}$
Finally: Send $l$ and $S$ to the referee.

Bob: Symmetric to Alice.

Referee: The referee receives $l_A$, $S_A$ from Alice and $l_B$, $S_B$ from Bob. From these, the referee knows $a_i$ for all $i$ in $S(1/2^{l_A})$ and $b_i$ for all $i$ in $S(1/2^{l_B})$. Let $l^* = \max(l_A, l_B)$. The referee outputs $2^{l^*} \cdot \sum_{i \in S(1/2^{l^*})}(a_i \vee b_i)$.
```

Figure 2: Public coins algorithm.

PROOF. *Estimation guarantees:* We omit the proof of the approximation guarantees for this model, since the proof is similar in spirit, but simpler, than the proof for the stored coins model given in Section 3.2.

Time complexity: Upon receiving a bit a_i, Alice stores i if $e(i)$ is greater than the current level, l. If storing the bit requires Alice to move to a higher level, then she incurs the cost of removing all the items in the sample that do not belong to the next level. The expected number of level changes that any item $a_i = 1$ survives is two. Amortized over n bits, this gives an amortized expected cost of three operations, one of them being the computation of e. By applying "lazy discarding" when changing levels (as discussed in more detail for our stored coins scheme), so that once $|S|$ reaches $c\alpha$, it remains there, the amortized bound becomes an expected bound for worst case inputs.

Space complexity: Alice and Bob each store up to $c\alpha$ pairs of $(i, e(i))$. Hence, the space complexity is $O(\alpha \log n)$, which is $O(\frac{\log(1/\delta)\log n}{\epsilon^2})$. Recall that in the public coins model, we do not account for the space required to store the shared random bits. □

3.2 Stored Coins Scheme

The stored coins algorithm is identical to the public coins algorithm except for the following differences: We can no longer afford (in terms of workspace) an all powerful hash function e that generates fully independent random variables. Instead, we settle for one that generates random variables that are pairwise independent. But in doing so, we lose the high probability approximation guarantee of the public coins algorithm: an instance of the algorithm now gives us an estimate with ϵ relative error only with probability bounded below by $\frac{2}{3}$. In order to get the error probability down to δ, we take the median of the values computed by $\beta = 48\log(\frac{1}{\delta})$ parallel instances of the algorithm. Interestingly, the threshold sample size at which the algorithm shifts to a higher level is smaller than in the public coins algorithm by a factor of $\Theta(\log(1/\delta)$. Specifically, it is c/ϵ^2, for $c = 36$, a constant determined by the analysis, and hence

position	1	2	3	4	5	6	7	8
A	0	1	1	0	1	0	1	1
B	1	1	0	0	1	1	0	1
U	1	1	1	0	1	1	1	1

position	9	10	11	12	13	14	15	16
A	0	0	0	1	0	1	0	1
B	0	0	1	0	0	0	0	1
U	0	0	1	1	0	1	0	1

(a) The data streams for Alice and Bob, as well as the bitwise contribution to U. Note that $\sum_i a_i = 8$, $\sum_i b_i = 7$, and $U(A, B) = 11$.

level	positions to be sampled
0	all
1	{2,4,5,9,11,12,14,15,16}
2	{4,5,11,12}
3	{5,11}
4	{5}

(b) Positions selected to be sampled at each level, according to the hash function e. (Levels 3 and 4 will not be needed.)

Alice

bits received	action	current sample	level
a_1 to a_7	sample	{2,3,5,7}	0
a_8	change levels	{2,5}	1
a_9 to a_{15}	sample	{2,5,12,14}	1
a_{16}	change levels	{5,12}	2
send {5,12}, level=2 to the referee			

Bob

bits received	action	current sample	level
b_1 to b_7	sample	{1,2,5,6}	0
b_8	change levels	{2,5}	1
b_9 to b_{16}	sample	{2,5,11,16}	1
send {2,5,11,16}, level=1 to the referee			

(c) The computation by Alice and Bob. The *current sample* column refers to the sample *after* receiving and processing all the bits in the *bits received* column on the left. When the size of the sample exceeds 4, Alice (Bob) changes to the next level and discards sample points that do not belong at this new level. Note that only the 1's are stored in the sample.

Referee

1. Receives messages from Alice and Bob. The referee level l is the max of the two levels, i.e., $l = 2$.
2. Subsamples Bob's sample to level 2. Bob's new sample is $S_B = \{5, 11\}$, level=2. Alice's is $S_A = \{5, 12\}$, level=2.
3. Outputs $|S_A \cup S_B| \cdot 2^l = 12$ as the estimate.

(d) The computation by the referee.

Figure 3: An example computation.

we use $\alpha = 1/\epsilon^2$.

Let e_k denote the hash function used by instance k of the algorithm. For every k, e_k is a mapping from $\{1 \dots n\}$ to $\{0 \dots d = \log n\}$, such that for $0 \le l \le (d-1)$, $\mathbf{Pr}\{e_k(i) = l\} = \frac{1}{2^{l+1}}$. $e_k(i)$ is computed as follows: we consider the numbers $\{1 \dots n\}$ as members of the field $G = GF(2^d)$. In a preprocessing step, we choose q_k and r_k uniformly and independently at random from G and store them with Alice and Bob. In order to compute $e_k(i)$, Alice (Bob) computes $x = q_k \cdot i + r_k$, all operations being performed in G. We represent x as a d-bit vector and then $e_k(i)$ is the largest j such that the j most significant bits of x are zero (i.e., $j = d - \lfloor \log x \rfloor - 1$). Clearly, $e_k(i) \in [0..d]$. The two properties of e_k that we use are: (1) x is distributed uniformly over G. Hence the probability that $e_k(i)$ equals l (where $l < d$) is exactly $\frac{1}{2^{l+1}}$. (2) The mapping is pairwise independent, i.e., for distinct i and j, $\mathbf{Pr}\{(e_k(i) = k_1) \wedge (e_k(j) = k_2)\} = \mathbf{Pr}\{e_k(i) = k_1\} \cdot \mathbf{Pr}\{e_k(j) = k_2\}$.

THEOREM 3. *For any positive $\epsilon \le 1$ and $\delta \le 1$, the above algorithm is an (ϵ, δ)-approximation scheme for U, the size of the union of the two bit streams, in the distributed streams model with stored coins. The algorithm uses a worst case expected $O(\log(1/\delta))$ finite field operations to process each item, and a total of $O(\frac{\log(1/\delta)\log n}{\epsilon^2})$ bits of workspace.*

PROOF. *Estimation guarantees:* The detailed proof is presented below.

Time complexity: We first analyze the time taken to process a bit by an instance, A_k, of the algorithm at Alice. For each 1-bit a_i, A_k does the following: (1) Compute $e_k(i)$ and determine whether the item should be stored. (2) If there is no space to store the item, then move to higher level and evict those that do not belong at the higher level.

We store the sample S_k as an array $S_k[1..d]$ of linked lists, with $S_k[i]$ containing all those elements j for which $e_k(j) = i$, i.e., those which "die" at level i. Upon receiving a bit, it is inserted into the appropriate list (if it is a 1). When the algorithm needs to move to the next level, it simply deletes all the elements that are in the list corresponding to the current level. Because every element is inserted and deleted exactly once from this data structure, the total work done over all the n elements is $O(n)$ plus the time for the computation of $e_k(i), i = 1, \dots, n$. If we assume that we can multiply and add two $\log n$ bit numbers in $GF(n)$ in constant time, then this would give us an amortized constant time per bit for A_k. Hence the time complexity of this algorithm is $O(\beta)$ amortized, which is $O(\log(1/\delta))$.

We can use lazy discarding when changing levels, so that once $|S_k|$ reaches $c\alpha$, we discard stored items from discarded levels one at a time as needed to make room for a new item. The linked lists by level enable the algorithm to discard the next item to be discarded in constant expected time. It is constant *expected* time instead of constant *deterministic* time because when all the items in a level have been discarded, we need to find the next level with items to discard, and it is possible that we first have to skip over some empty levels. Before sending their samples to the referee, Alice and Bob discard any remaining stored items from discarded levels. Note that even for the worst case input streams, the algorithm is constant expected time per instance per bit.

Space complexity: Each of the β instances of the algorithm stores up to $c\alpha$ positions at a time, where each position is between 1 and n. Storing a hash function e_k requires $\log n$ space, since we need to store q_k and r_k, both of which are members of $GF(n)$. Thus, each instance of the algorithm needs space $O(c\alpha \log n + \log n)$, which is $O(\frac{\log n}{\epsilon^2})$. Since there are β instances, the space needed at each of Alice and Bob is $O(\frac{\log(1/\delta)\log n}{\epsilon^2})$. □

Proof of the estimation guarantees. If we could ensure that Alice and Bob stop at the correct level, then the desired estimation guarantees would follow from Chernoff bounds. However, Alice and Bob decide when to stop changing levels based on the outcome of random trials during the course of observing their streams, and hence may stop at incorrect levels, and make correspondingly bad estimates. Thus, a more careful proof is needed.

Let 1_A denote the set $\{i | a_i = 1\}$, i.e., the positions of the 1-bits in Alice's stream. Similarly, let $1_B = \{i | b_i = 1\}$, and let $1_U = \{i | a_i = 1 \vee b_i = 1\}$. The referee is trying to estimate the size of 1_U. The random variable computed by the kth instance of the algorithm, z_k, is described with the help of the following process: We place the numbers $1 \dots n$ in "levels" $\{0 \dots d\}$ by placing i in every level from 0 through (and including) $e_k(i)$. For $l \in [0..d]$ and $i \in [1..n]$, we define random variables X_{li}, as follows. $X_{li} = 1$ if i was placed in level l and 0 otherwise. For every level $l \in [0..d]$, we define $X_l = \sum_{i \in 1_U} X_{li}$ and $X_l^A = \sum_{i \in 1_A} X_{li}$ and $X_l^B = \sum_{i \in 1_B} X_{li}$. Note that X_l is greater than or equal to both X_l^A and X_l^B.

Here is how the algorithm fits into this process: Alice stops at level l_A, where l_A is the lowest numbered level l such that $X_l^A < c\alpha$. Similarly, Bob stops at level l_B. If l_A or l_B equals d, the algorithm quits. Otherwise it returns the estimate $z_k = 2^f \cdot X_f$ where $f = \max\{l_A, l_B\}$. For every $l \in [0..d-1]$, we define level l to be *bad* if $2^l X_l \notin [(1-\epsilon)U, (1+\epsilon)U]$. Let B_i denote the event that level i is bad. Let S_i denote the event that the algorithm stops in level i (i.e., f equals i).

LEMMA 1. *The probability that the kth instance of the algorithm fails to produce an estimate which is within a factor ϵ of U is less than $\frac{1}{3}$.*

PROOF. The kth instance fails to do so in the following cases:

- l_A or l_B equals d, i.e., $f = \max\{l_A, l_B\} = d$ and the algorithm quits, or more likely,
- f is less than d, but f is a bad level, i.e., for some level k, the events S_k and B_k are both true.

Let P denote the probability that the algorithm fails. P is less than the sum of the probabilities of the above events. Thus $P \le \mathbf{Pr}\{2^f X_f \notin ((1-\epsilon)U, (1+\epsilon)U)\} + \mathbf{Pr}\{S_d\} = \sum_{i=0}^{d-1} \mathbf{Pr}\{S_i \wedge B_i\} + \mathbf{Pr}\{S_d\}$

Let level l denote the first level such that $E[X_l] < C\alpha$, for $C = 24$, a constant determined from the analysis. Note that l is less than d, because $E[X_d] = \frac{U}{2^d} = \frac{U}{n} \le 1 < C\alpha$. We split the above sum as follows:

$$\begin{aligned}
P &= \sum_{i=0}^{l} \mathbf{Pr}\{S_i \wedge B_i\} + \sum_{i=l+1}^{d-1} \mathbf{Pr}\{S_i \wedge B_i\} + \mathbf{Pr}\{S_d\} \\
&\le \sum_{i=0}^{l} \mathbf{Pr}\{B_i\} + \sum_{i=l+1}^{d-1} \mathbf{Pr}\{S_i\} + \mathbf{Pr}\{S_d\} \\
&= \sum_{i=0}^{l} \mathbf{Pr}\{B_i\} + \sum_{i=l+1}^{d} \mathbf{Pr}\{S_i\}
\end{aligned}$$

The idea here is that the first few levels (until l) are likely to have good estimates and it is unlikely that we stop in a later level ($l+1$ onwards). Note that if $U \le c\alpha$, the algorithm collects all the 1-bits in both streams and hence there is no error. So assume that $U > c\alpha$.

To complete the proof, we prove a series of four lemmas (Lemmas 3-6 below). We first show that for $i = 0, \ldots, d-1$, we have $E[X_i] = \frac{U}{2^i}$ and $\mathrm{Var}[X_i] = \frac{U}{2^i}\left(1 - \frac{1}{2^i}\right)$. Then we use Chebyshev's inequality to show that $\mathbf{Pr}\{B_i\} < \frac{2^i}{U\epsilon^2}$. Thus $\sum_{i=0}^{l} \mathbf{Pr}\{B_i\} < \frac{2^{l+1}}{U\epsilon^2}$. Because l is the first level such that $\frac{U}{2^l} < C\alpha$ and $c > C$, we have that $\frac{U}{2^l} \ge \frac{C}{2\epsilon^2}$. Thus $\sum_{i=0}^{l} \mathbf{Pr}\{B_i\} < \frac{4}{C} = \frac{1}{6}$. Next, we observe that $\sum_{i=l+1}^{d} \mathbf{Pr}\{S_i\} = \mathbf{Pr}\{X_l > c\alpha\}$. Then we use Chebyshev's again and the fact that $E[X_l] < C\alpha$ to show that this is less than $\frac{C\epsilon^2}{(c-C)^2} < \frac{1}{6}$. Thus, $P < \frac{1}{6} + \frac{1}{6} = \frac{1}{3}$, completing the proof of Lemma 1. □

Using Lemma 1 and Chernoff bounds, we prove the following lemma, which completes the proof of Theorem 3:

LEMMA 2. *The median of the set* $\{z_k | k = 1, \ldots, \beta\}$ *is within an* ϵ *relative error of* U *with probability* $\ge 1 - \delta$.

PROOF. The median fails to be an (ϵ, δ) estimator of U if more than $\beta/2$ instances of the algorithm fail. We know from Lemma 1 that the probability of each instance failing is less than 1/3. By Chernoff bounds, the probability the algorithm fails is less than $\exp(-\beta/48)$, i.e., less than δ, because $\beta = 48 \log(1/\delta)$. □

Details of the four lemmas. We conclude this section by stating and proving the four lemmas used to complete the proof of Lemma 1.

LEMMA 3. *For a given* l, *the random variables* $\{X_{li} | 1 \le i \le n\}$ *are pairwise independent.*

PROOF. For distinct i, j, $\mathbf{Pr}\{(X_{li} = 1) \wedge (X_{lj} = 1)\} = \mathbf{Pr}\{(e(i) \ge l) \wedge (e(j) \ge l)\} = \mathbf{Pr}\{e(i) \ge l\} \cdot \mathbf{Pr}\{e(j) \ge l\} = \mathbf{Pr}\{X_{li} = 1\} \cdot \mathbf{Pr}\{X_{lj} = 1\}$, because $e(i)$ and $e(j)$ are pairwise independent. Similar proofs for the other cases show that the X_{li}'s are pairwise independent. □

LEMMA 4. *For* $l \in [0..d-1]$, $E[X_l] = \frac{U}{2^l}$ *and* $Var[X_l] = \frac{U}{2^l}\left(1 - \frac{1}{2^l}\right)$.

PROOF.
$$E[X_l] = E[\sum_{i \in 1_U} X_{li}] = \sum_{i \in 1_U} E[X_{li}] = |1_U| \cdot E[X_{li}]$$
where we have used linearity of expectation. $E[X_{li}]$ is the probability that i made it to level l, which is $\mathbf{Pr}\{e_k(i) \ge l\}$, i.e., $\frac{1}{2^l}$. Since $|1_U|$ is U, we have $E[X_l] = \frac{U}{2^l}$.

Since the X_{ki}'s are pairwise independent (Lemma 3), the variance of the sum is the sum of variances.
$$\mathrm{Var}[X_l] = \sum_{i \in 1_U} \mathrm{Var}[X_{li}] = |1_U| \cdot \mathrm{Var}[X_{li}] = \frac{U}{2^l}\left(1 - \frac{1}{2^l}\right)$$
□

LEMMA 5. $\sum_{i=0}^{l} \mathbf{Pr}\{B_i\} < 1/6$

PROOF. Let μ_k and σ_k denote the mean and standard deviation of X_k respectively. Then,
$$\mathbf{Pr}\{B_k\} = \mathbf{Pr}\{|X_k - \mu_k| \ge \epsilon\mu_k\}$$
Using Chebyshev's inequality, $\mathbf{Pr}\{|X_k - \mu_k| \ge t\sigma_k\} \le \frac{1}{t^2}$ and substituting $t = \frac{\epsilon\mu_k}{\sigma_k}$, we get:
$$\mathbf{Pr}\{B_k\} \le \frac{\sigma_k^2}{\epsilon^2\mu_k^2} \tag{2}$$
Substituting values of μ_k and σ_k^2 from Lemma 4 in equation 2, we get
$$\mathbf{Pr}\{B_k\} \le \frac{2^k}{U\epsilon^2}(1 - \frac{1}{2^k}) < \frac{2^k}{U\epsilon^2}$$
$$\sum_{i=0}^{l} \mathbf{Pr}\{B_i\} < \sum_{i=0}^{l} \frac{2^k}{U\epsilon^2} = \frac{2^{l+1} - 1}{U\epsilon^2} < \frac{2^{l+1}}{U\epsilon^2}$$
Because l is the first level such that $\frac{F}{2^l} < C\alpha$, we have $\frac{U}{2^{l-1}} \ge C\alpha$, i.e., $\frac{U}{2^l} \ge \frac{C}{2\epsilon^2}$. (Recall that we have assumed $U > c\alpha$.) Thus, $\sum_{i=0}^{l} \mathbf{Pr}\{B_i\} < \frac{4}{C}$.

We choose $C = 24$ and Lemma 5 is proved. □

LEMMA 6. $\sum_{i=l+1}^{d} \mathbf{Pr}\{S_i\} < 1/6$

PROOF. The sum $\sum_{i=l+1}^{d} \mathbf{Pr}\{S_i\}$ is the probability that the algorithm stops in level $l+1$ or greater (because the S_i's are mutually exclusive). This is exactly the probability that at level l, we still have more than $c\alpha$ 1's, i.e., the probability that $X_l \ge c\alpha$.
$$\begin{aligned}\mathbf{Pr}\{X_l \ge c\alpha\} &= \mathbf{Pr}\{X_l - \mu_l \ge c\alpha - \mu_l\} \\ &\le \mathbf{Pr}\{X_l - \mu_l \ge c\alpha - C\alpha\} \\ &\le \frac{\sigma_l^2}{(c-C)^2\alpha^2}\end{aligned}$$
The second inequality holds because $\mu_l < C\alpha$, while the third inequality holds by Chebyshev's inequality.

From Lemma 4 we have $\sigma_l^2 = U(\frac{1}{2^l})(1 - \frac{1}{2^l}) \le \frac{U}{2^l}$. Since $E[X_l] < C\alpha$, we have $\frac{U}{2^l} < C\alpha$. Using this in the above expression, we have:
$$\begin{aligned}\mathbf{Pr}\{X_l \ge c\alpha\} &< \frac{C}{(c-C)^2\alpha} = \frac{C\epsilon^2}{(c-C)^2} \\ &= \frac{\epsilon^2}{6} \quad \text{by choosing} \quad c = 36, C = 24 \\ &< \frac{1}{6} \quad \text{since} \quad \epsilon < 1\end{aligned}$$
□

4. EXTENSIONS AND APPLICATIONS

In this section, we highlight a few of the extensions and applications of coordinated 1-sampling.

More general scenarios. The public coins and stored coins algorithms of Section 3 can be extended in the following three ways:

1. The same algorithms can be used for $t > 2$ parties.

2. The same algorithms can be used when the bits do not arrive in the same order to Alice and Bob. In this scenario, Alice observes pairs (i, a_i) and Bob observes pairs (j, b_j), where an adversary controls the order of the pairs in each data stream.

3. The same algorithms can be used when only the 1-bits appear in the data stream: Alice and Bob each observe an unordered sequence of the positions of the 1-bits in their respective streams. The time bounds in Theorems 2 and 3 are per observed bit. The space bounds have a factor that is a logarithm of the domain size for position numbers.

In all cases, the algorithms obtain the same approximation guarantees, the same per item time bounds, and the same per stream space bounds as before. Let A_p (A_s) be the public coins (stored coins, respectively) algorithm from Section 3 generalized to $t \geq 2$ parties. Then, for example, we have the following corollary to Theorems 2 and 3:

COROLLARY 1. *Consider t sets from a domain of size m, where each data stream consists of the elements in a set arriving in an order controlled by an adversary. For any positive $\epsilon \leq 1$ and $\delta \leq 1$, A_p (A_s) is an (ϵ, δ)-approximation scheme for the union of these t sets, in the distributed streams model with public coins (stored coins, respectively). The algorithm uses worst case expected $O(1)$ time ($O(\log(1/\delta))$ finite field operations, respectively) to process each item, and $O(\frac{\log(1/\delta)\log n}{\epsilon^2})$ bits of workspace per stream.*

Unlike previous work on data streams, our coordinated 1-sampling approach obtains a random sample of the union, of a target size ρ in $O(\rho)$ workspace, along with a scaling factor.

Estimating F_0 and related functions. The approach can be extended to obtain a random sample of the distinct values ("labels") in the data streams, even when a label can appear multiple times within a stream. To estimate the number of distinct labels F_0, we must ensure that each label is stored at most once in the sample collected at a party and we need some additional data structures for this. We store the at most $c\alpha$ labels present in the sample in a hash table of size $\Theta(\alpha)$, using the label as a key. (This is in addition to storing each label i in the linked list for $e(i)$.) This enables constant expected look-up time. More generally, we can store in the hash table entry for each label in the sample an accumulation value for that label relevant to the function being estimated, e.g., the number of occurrences of the label within the stream. (We could also store $e(i)$, if desired.) Maintaining a distinct labels sample in the presence of new data is useful for approximate query answering systems for data warehouses, such as the Aqua system [1, 2].

Average interarrival gap. Our relative error approximation of U permits a relative error approximation of the average interarrival gap I in the union of multiple streams, for the common case where time is discretized, i.e.,

$$I = \frac{\text{number of time slots}}{\text{size of union}}$$

Set resemblance. The *set resemblance* $r(A, B)$ of two sets A and B is the size of their intersection divided by the size of their union. Set resemblance is used by Alta Vista to eliminate near-duplicate pages from search results [5]. Coordinated 1-sampling can be used to approximate the set resemblance, using the identity:

$$r(A, B) = \frac{|A| + |B|}{|A \cup B|} - 1$$

We match the asymptotic space bounds and approximation guarantees of the best previous approaches (e.g., [5, 7, 17]), while improving the time bounds by a factor of $\Theta(1/\epsilon^2)$. Note that for set resemblance, the guarantees are in terms of an absolute and not a relative error. This is unavoidable, due to known lower bounds [8].

General values. Our algorithm can also be generalized for the case when the a_i's and b_i's are not binary, but are integers in the range $\{0, \ldots, M-1\}$, and the function to be estimated is $g(A, B) = \sum_{i=1}^{n} \max\{a_i, b_i\}$. We can obtain the same asymptotic space bound as the boolean case (as long as M is at most polynomial in n), at the cost of the time bound, as follows. Let $Y(x, M)$ (where $0 \leq x < M$) be the unary representation of x, with enough zeros added to the right to make the length of the representation exactly M. For example, $Y(5, 8) = 11111000$. We observe that $U(Y(x, M), Y(y, M)) = \max(x, y)$, where $x, y < M$. Hence, we can treat the integer stream $A = \{a_i | i = 1, \ldots, n\}$ as a bit stream $Y(A)$ formed by concatenating the bit streams $\{Y(a_i, M) | i = 1, \ldots, n\}$ and similarly for B, and then apply the binary input algorithm.

5. COMPARISON OF MODELS

In this section we compare the power of different streaming models on distributed data sets, and use ideas from communication complexity to derive relationships between these models for deterministic algorithms.

The *deterministic merged stream complexity* of a function f, denoted by $DM_t(f)$, is the minimum workspace required by a deterministic algorithm to compute (exactly) the function $f(X_1, \ldots, X_t)$ when t streams $X_1, \ldots, X_t$ are observed by a single party, but interleaved arbitrarily. Similarly, the *deterministic distributed stream complexity* of f, denoted by $DD_t(f)$ is the minimum workspace required by a deterministic algorithm to compute (exactly) the function $f(X_1, \ldots, X_t)$ in the distributed streams model. In both the distributed and merged stream models, arbitrary pre- and post-processing resources (both space and time) are permitted, and only the resources used while observing the streams are counted towards the complexity. Note that in the distributed streams model, the space complexity is the *sum* of the workspace used at all the parties.

THEOREM 4. *For any $t \geq 1$ and any function f, (1) $DM_t(f) \leq DD_t(f)$; (2) $DD_t(f) \leq t \cdot DM_t(f)$; and (3) the factor of t bound is existentially tight, i.e., there exists a function g for which $DD_t(g) = t \cdot DM_t(g)$.*

PROOF. (1) In the merged streams model, we can simulate the independent parties of a distributed streams protocol, within the same space bounds.

(2) The proof adapts techniques from communication complexity [21]. Given any protocol P_m in the merged stream model which can compute f in space s, we will show a protocol P_d in the distributed stream model which can compute f in space $t \cdot s$. Suppose all of stream X_1 was given to

P_m before any item of another stream. Let $h_1(X_1)$ denote the "synopsis" of X_1 computed by P_m after it has seen X_1. Similarly, define $h_2(X_2), \ldots, h_t(X_t)$. In P_d, the ith party computes $h_i(X_i)$. The space complexity of this algorithm is $t \cdot s$. Consider the "truth table" of f, which is a table of size $\prod_{i=1}^{t} D_i$, where $D_1 \times \cdots \times D_t$ is the domain of f, and the ith dimension is labeled by each $x \in D_i$. The fact that $h_i(x)$ can be stored in space s means that we can partition the rows along the ith dimension into 2^s classes such that if y and z are in the same class (i.e., $h_i(y) = h_i(z)$) then for any other inputs, $f(\ldots, x_i = y, \ldots) = f(\ldots, x_i = z, \ldots)$. It is clear that if we know the classes to which each x_i belongs, then the value of f is uniquely determined. Thus, because the referee receives $h_i(x_i)$ for all i, he can compute $f(x_1, \ldots, x_t)$.

(3) Let g be the (exact) Hamming distance between $t = 2$ sequences, A and B, of n bits each. (This argument can be generalized to $t > 2$.) An item of a stream is a pair (i, v), denoting that v is the value of the ith bit in the corresponding sequence; an adversary controls the order of the pairs. In the merged streams model, we use an array $r[1..n]$ of bits, initialized to zero. Upon receiving (i, v) in A or in B, we set $r[i] = r[i] \oplus v$. We compute $\sum_{i=1}^{n} r[i]$ in a post-processing step, and output the result. Thus $DM_2(g) \leq n$.

We now show that $DD_2(g) \geq 2 \cdot n$. If we use $< 2n$ space, then at least one of the entities (say Alice) uses space less than n bits. Because there are 2^n different bit sequences of length n, there exist two distinct sequences, say x and y, for which Alice has the same workspace contents and hence sends the same message to the referee. Consider two instances of the problem, one in which $A = B = x$ and the other in which $A = y$ and $B = x$. The messages sent by both Alice and Bob to the referee are the same in both cases and the referee will return the same value, which is a contradiction. □

We next compare the complexity of merged streams vs. deterministic streams for deterministic approximation algorithms. For $\epsilon > 0$, we define the *ϵ-approximate merged stream complexity* of a function f to be the minimum workspace required by a deterministic algorithm to compute the function $f(X_1, \ldots, X_t)$ within an error of ϵ. There are two versions of this, one with absolute error of ϵ which we denote by $DM_t^{\epsilon,abs}(f)$ and one with relative error of ϵ, denoted by $DM_t^{\epsilon,rel}(f)$. We define the distributed stream analogs of these similarly, and denote them by $DD_t^{\epsilon,abs}(f)$ and $DD_t^{\epsilon,rel}(f)$.

THEOREM 5. *For $t \geq 1$ and any function f, we have (1)$DM_t^{\epsilon,abs}(f) \leq DD_t^{\epsilon,abs}(f)$ and (2)$DD_t^{(t\epsilon),abs}(f) \leq t \cdot DM_t^{\epsilon,abs}(f)$.*

PROOF. We prove (2) since the proof of (1) is clear. We are given a merged stream protocol P_m which computes f within an absolute error of ϵ in space $s = DM_t^{\epsilon,abs}(f)$. We describe the proof for the case of $t = 2$; this can be readily generalized for $t > 2$.

The proof is a generalization of the proof in part (2) of Theorem 4, and again adapts techniques from communication complexity [21]. Suppose all of stream X_1 was given to P_m before any item of another stream. Let $h_1(X_1)$ denote the synopsis of X_1 computed by P_m after it has seen X_1. Similarly, define $h_2(X_2)$. The distributed streams protocol is simple: Alice computes $h_1(X_1)$, Bob computes $h_2(X_2)$, and the referee computes a function $g(h_1(X_1), h_2(X_2))$, explained below. The protocol is shown to have error less than 2ϵ. It is clear that the space required is $2s$.

As above, we can think of f as a table, which has a row for each value of X_1 and a column for each value of X_2. We can divide the rows of f into 2^s "row-classes", such that for two points in the table (r_1, c) and (r_2, c) if r_1 and r_2 belong to the same row-class (i.e., $h_1(r_1) = h_1(r_2)$), then the protocol returns the same value for both the points. Since the protocol's error is at most ϵ, it follows that $|f(r_1, c) - f(r_2, c)| \leq 2\epsilon$. Similarly, we can divide the columns also into "column-classes" such that for two points (r, c_1) and (r, c_2), if c_1 and c_2 belong to the same column-class (i.e., $h_2(c_1) = h_2(c_2)$), then $|f(r, c_1) - f(r, c_2)| \leq 2\epsilon$.

We now describe $g(x, y)$. Consider the rectangle $R(x, y)$ with rows $\{r | h_1(r) = x\}$ and columns $\{c | h_2(c) = y\}$. This is a set of the points all of which return the same value $g(x, y)$ in the distributed streams protocol. Let (x_m, y_m) denote the minimum element in this rectangle and (x_M, y_M) the maximum element. Then $g(x, y) = [f(x_M, y_M) + f(x_m, y_m)]/2$. We show that for any point in the rectangle, returning $g(x, y)$ will result in error at most 2ϵ. We know that $f(x_M, y_m) - f(x_m, y_m) \leq 2\epsilon$, because $h_1(x_m) = h_1(x_M)$. Similarly, $f(x_M, y_M) - f(x_M, y_m) \leq 2\epsilon$. Thus, $f(x_M, y_M) - f(x_m, y_m) \leq 4\epsilon$, i.e., the maximum and minimum elements of the rectangle are only 4ϵ apart. This implies that no element in the rectangle is farther than 2ϵ from the mean of these two elements, which is $g(x, y)$. Thus, the error of returning $g(x, y)$ for the whole rectangle is at most 2ϵ. □

THEOREM 6. *For the approximate relative error scenario, for any function f which is positive non-zero everywhere, $DD_2^{\epsilon',rel}(f) \leq 2 \cdot DM_2^{\epsilon,rel}(f)$, where $\epsilon' = \frac{2\epsilon}{1-\epsilon}$.*

PROOF. The proof is similar to the proof of Theorem 5, except that we use $g(x, y) = \sqrt{f(x_M, y_M) f(x_m, y_m)}$. □

6. CONCLUSIONS

This paper presented a simple technique, called coordinated 1-sampling, for estimating functions on the union of data streams. Although simple, this technique improves upon the best known space and/or time bounds for estimating various important functions on one or more data streams, and unlike the previous probabilistic counting approaches, it provides not just an estimate for one function, but a sample that can be used for estimating multiple functions. A careful analysis is presented of its approximation guarantees for the Union function. We also presented exponential gaps between coordinated and uncoordinated sampling for estimating both the Union and the number of distinct values F_0. Finally, we motivated and formulated the distributed streams model, and presented tight bounds comparing it to previously studied streams models for deterministic algorithms.

An open problem is to determine the complexity of merged streams vs. distributed streams for randomized algorithms in both the public coins and stored coins models. Kremer et al. [20] presented a proof that for any boolean function, the randomized simultaneous communication complexity with public coins is linear in the sum of the two one-way (Alice-to-Bob and Bob-to-Alice) communication complexities. However, we recently uncovered a flaw in their proof, which was subsequently confirmed by the authors [24]. Because of the connections between (a) merged streams and one-way communication and (b) distributed streams and simultaneous communication, any result relating communication models will likely lead to a similar result for the streams models, and perhaps vice-versa.

Acknowledgments. The second author would like to acknowledge Aravind Srinivasan and Eli Upfal for helpful discussions.

7. REFERENCES

[1] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. The Aqua approximate query answering system. In *Proc. ACM SIGMOD International Conf. on Management of Data*, pages 574–576, June 1999. Demo paper.

[2] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. Join synopses for approximate query answering. In *Proc. ACM SIGMOD International Conf. on Management of Data*, pages 275–286, June 1999.

[3] N. Alon, P. B. Gibbons, Y. Matias, and M. Szegedy. Tracking algorithms for join and self-join sizes. In *Proc. 18th ACM Symp. on Principles of Database Systems*, pages 1-11, May 1999. Full version to appear in JCSS special issue for PODS'99.

[4] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *Proc. 28th ACM Symp. on the Theory of Computing*, pages 20–29, May 1996. Full version to appear in JCSS special issue for STOC'96.

[5] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-wise independent permutations. In *Proc. 30th ACM Symp. on the Theory of Computing*, pages 327–336, May 1998. Full version to appear in JCSS special issue for STOC'98.

[6] M. Charikar, S. Chaudhuri, R. Motwani, and V. Narasayya. Towards estimation error guarantees for distinct values. In *Proc. 19th ACM Symp. on Principles of Database Systems*, pages 268–279, May 2000.

[7] E. Cohen. Size-estimation framework with applications to transitive closure and reachability. *J. of Computer and System Sciences*, 55(3):441–453, 1997.

[8] J. Feigenbaum, S. Kannan, M. Strauss, and M. Viswanathan. An approximate L^1-difference algorithm for massive data streams. In *Proc. 40th IEEE Symp. on Foundations of Computer Science*, pages 501–511, Oct. 1999.

[9] J. Feigenbaum, S. Kannan, M. Strauss, and M. Viswanathan. Testing and spot-checking of data streams. Technical report, AT&T Shannon Laboratories, Florham Park, NJ, July 1999.

[10] P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. *J. Computer and System Sciences*, 31:182–209, 1985.

[11] J. Fong and M. Strauss. An approximate L^p-difference algorithm for massive data streams. In *Proc. 17th Symp. on Theoretical Aspects of Computer Science, LNCS 1770*, pages 193–204. Springer, Feb. 2000.

[12] P. B. Gibbons and Y. Matias. New sampling-based summary statistics for improving approximate query answers. In *Proc. ACM SIGMOD International Conf. on Management of Data*, pages 331–342, June 1998.

[13] P. B. Gibbons and Y. Matias. Synopsis data structures for massive data sets. In J. M. Abello and J. S. Vitter, editors, *External Memory Algorithms*, pages 39–70. AMS, 1999. DIMACS: Series in Discrete Mathematics and Theoretical Computer Science, Vol. 50. A two page summary appeared as a short paper in SODA'99.

[14] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams. In *Proc. 41st IEEE Symp. on Foundations of Computer Science*, pages 359–366, Nov. 2000.

[15] P. J. Haas, J. F. Naughton, S. Seshadri, and L. Stokes. Sampling-based estimation of the number of distinct values of an attribute. In *Proc. 21st International Conf. on Very Large Data Bases*, pages 311–322, Sept. 1995.

[16] M. R. Henzinger, P. Raghavan, and S. Rajagopalan. Computing on data streams. Technical report, Digital Systems Research Center, Palo Alto, CA, May 1998.

[17] P. Indyk. A small approximately min-wise independent family of hash functions. Technical report, Stanford University, Palo Alto, CA, Nov. 1998.

[18] P. Indyk. Stable distributions, pseudorandom generators, embeddings and data stream computation. In *Proc. 41st IEEE Symp. on Foundations of Computer Science*, pages 189–197, Nov. 2000.

[19] P. Indyk, N. Koudas, and S. Muthukrishnan. Identifying representative trends in massive time series datasets using sketches. In *Proc. 26th International Conf. on Very Large Databases*, pages 363–372, Sept. 2000.

[20] I. Kremer, N. Nisan, and D. Ron. On randomized one-round communication complexity. *Computational Complexity*, 8(1):21–49, 1999. Preliminary version in STOC'95.

[21] E. Kushilevitz and N. Nisan. *Communication Complexity.* Cambridge University Press, Cambridge, UK, 1997.

[22] I. Newman. Private vs. common random bits in communication complexity. *Information Processing Letters*, 39:67–71, 1991.

[23] I. Newman and M. Szegedy. Public vs. private coin flips in one round communication games. In *Proc. 28th ACM Symp. on the Theory of Computing*, pages 561–570, May 1996.

[24] N. Nisan and D. Ron. Private communication, October-November 2000.

[25] Transaction processing performance council (TPC). *TPC Benchmarks*, 2000. URL: `www.tpc.org`.

[26] K.-Y. Whang, B. T. Vander-Zanden, and H. M. Taylor. A linear-time probabilistic counting algorithm for database applications. *ACM Transactions on Database Systems*, 15(2):208–229, 1990.