

## AN APPROXIMATE $L^1$ -DIFFERENCE ALGORITHM FOR MASSIVE DATA STREAMS\*

JOAN FEIGENBAUM<sup>†</sup>, SAMPATH KANNAN<sup>‡</sup>, MARTIN J. STRAUSS<sup>§</sup>, AND MAHESH VISWANATHAN<sup>¶</sup>

**Abstract.** Massive data sets are increasingly important in a wide range of applications, including observational sciences, product marketing, and the monitoring and operations of large systems. In network operations, raw data typically arrive in *streams*, and decisions must be made by algorithms that make one pass over each stream, throw much of the raw data away, and produce “synopses” or “sketches” for further processing. Moreover, network-generated massive data sets are often *distributed*: Several different, physically separated network elements may receive or generate data streams that, together, comprise one logical data set; to be of use in operations, the streams must be analyzed locally and their synopses sent to a central operations facility. The enormous scale, distributed nature, and one-pass processing requirement on the data sets of interest must be addressed with new algorithmic techniques.

We present one fundamental new technique here: a space-efficient, one-pass algorithm for approximating the  $L^1$ -difference  $\sum_i |a_i - b_i|$  between two functions, when the function values  $a_i$  and  $b_i$  are given as data streams, and their order is chosen by an adversary. Our main technical innovation, which may be of interest outside the realm of massive data stream algorithmics, is a method of constructing families  $\{V_j(s)\}$  of limited-independence random variables that are *range-summable*, by which we mean that  $\sum_{j=0}^{c-1} V_j(s)$  is computable in time  $\text{polylog}(c)$  for all seeds  $s$ . Our  $L^1$ -difference algorithm can be viewed as a “sketching” algorithm, in the sense of [Broder et al., *J. Comput. System Sci.*, 60 (2000), pp. 630–659], and our technique performs better than that of Broder et al. when used to approximate the symmetric difference of two sets with small symmetric difference.

**Key words.** streaming algorithms, distance approximation

**AMS subject classifications.** 68W20, 68W25

**PII.** S0097539799361701

**1. Introduction.** Massive data sets are increasingly important in a wide range of applications, including observational sciences, product marketing, and the monitoring and operations of large systems. In network operations, raw data typically arrive in *streams*, and decisions must be made by algorithms that make one pass over each stream, throw much of the raw data away, and produce “synopses” or “sketches” for further processing. Moreover, network-generated massive data sets are often *distributed*: Several different, physically separated network elements may receive or generate data streams that, together, comprise one logical data set; to be of use in operations, the streams must be analyzed locally and their synopses sent to a central

---

\*Received by the editors September 14, 1999; accepted for publication (in revised form) July 30, 2002; published electronically December 11, 2002. An extended abstract of this paper appeared in *Proceedings of the 1999 IEEE Symposium on Foundations of Computer Science*.

<http://www.siam.org/journals/sicomp/32-1/36170.html>

<sup>†</sup>Department of Computer Science, Yale University, New Haven, CT 06520-8285 (feigenbaum@cs.yale.edu). Most of this work was done while this author was a member of the Information Sciences Research Center of AT&T Labs in Florham Park, NJ.

<sup>‡</sup>Computer and Information Science Department, University of Pennsylvania, Philadelphia, PA 19104-6389 (kannan@cis.upenn.edu). Part of this work was done while this author was visiting AT&T Labs in Florham Park, NJ. This author was supported by grants NSF CCR98-20885 and ARO DAAG55-98-1-0393.

<sup>§</sup>AT&T Labs—Research, 180 Park Avenue, Florham Park, NJ 07932 (mstrauss@research.att.com).

<sup>¶</sup>Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801 (vmahesh@cs.uiuc.edu). This work was done while this author was a Ph.D. student at the University of Pennsylvania and was supported by grant ONR N00014-97-1-0505, MURI.

operations facility. The enormous scale, distributed nature, and one-pass processing requirement on the data sets of interest must be addressed with new algorithmic techniques.

We present one fundamental new technique here: a space-efficient, one-pass algorithm for approximating the  $L^1$ -difference  $\sum_i |a_i - b_i|$  between two functions, when the function values  $a_i$  and  $b_i$  are given as data streams, and their order is chosen by an adversary. This algorithm fits naturally into a toolkit for Internet-traffic monitoring. For example, Cisco routers can now be instrumented with the NetFlow feature [CN98]. As packets travel through the router, the NetFlow software produces summary statistics on each *flow*.<sup>1</sup> Three of the fields in the flow records are source IP-address, destination IP-address, and total number of bytes of data in the flow. At the end of a day (or a week, or an hour, depending on what the appropriate monitoring interval is and how much local storage is available), the router (or, more accurately, a computer that has been “hooked up” to the router for monitoring purposes) can assemble a set of values  $(x, f_t(x))$ , where  $x$  is a source-destination pair, and  $f_t(x)$  is the total number of bytes sent from the source to the destination during a time interval  $t$ . The  $L^1$ -difference between two such functions assembled during different intervals or at different routers is a good indication of the extent to which traffic patterns differ.

Our algorithm allows the routers and a central control and storage facility to compute  $L^1$ -differences efficiently under a variety of constraints. First, a router may want the  $L^1$ -difference between  $f_t$  and  $f_{t+1}$ . The router can store a small “sketch” of  $f_t$ , throw out all other information about  $f_t$ , and still be able to approximate  $\|f_t - f_{t+1}\|_1$  from the sketch of  $f_t$  and (a sketch of)  $f_{t+1}$ .

The functions  $f_t^{(i)}$  assembled at each of several remote routers  $R_i$  at time  $t$  may be sent to a central tape-storage facility  $C$ . As the data are written to tape,  $C$  may want to compute the  $L^1$ -difference between  $f_t^{(1)}$  and  $f_t^{(2)}$ , but this computation presents several challenges. First, each router  $R_i$  should transmit its statistical data when  $R_i$ 's load is low and the  $R_i$ - $C$  paths have extra capacity; therefore, the data may arrive at  $C$  from the  $R_i$ 's in an arbitrarily interleaved manner. Also, typically the  $x$ 's for which  $f(x) \neq 0$  constitute a small fraction of all  $x$ 's; thus,  $R_i$  should only transmit  $(x, f_t^{(i)}(x))$  when  $f_t^{(i)}(x) \neq 0$ . The set of transmitted  $x$ 's is not predictable by  $C$ . Finally, because of the huge size of these streams,<sup>2</sup> the central facility will not want to buffer them in the course of writing them to tape (and cannot read from one part of the tape while writing to another), and telling  $R_i$  to pause is not always possible. Nevertheless, our algorithm supports approximating the  $L^1$ -difference between  $f_t^{(1)}$  and  $f_t^{(2)}$  at  $C$ , because it requires little work space, requires little time to process each incoming item, and can process in one pass all the values of both functions  $\{(x, f_t^{(1)}(x))\} \cup \{(x, f_t^{(2)}(x))\}$  in any permutation.

Our  $L^1$ -difference algorithm achieves the following performance:

Consider two data streams of length at most  $n$ , each representing the nonzero points on the graph of an integer-valued function on a domain of size  $n$ . Assume that the maximum value of either

<sup>1</sup>Roughly speaking, a “flow” is a semantically coherent sequence of packets sent by the source and reassembled and interpreted at the destination. Any precise definition of “flow” would have to depend on the application(s) that the source and destination processes were using to produce and interpret the packets. From the router’s point of view, a flow is just a set of packets with the same source and destination IP-addresses whose arrival times at the routers are close enough, for a tunable definition of “close.”

<sup>2</sup>In 1999, a WorldNet gateway router generated more than 10Gb of NetFlow data each day.

function on this domain is  $M$ . Then a one-pass streaming algorithm can compute with probability  $1 - \delta$  an approximation  $A$  to the  $L^1$ -difference  $B$  of the two functions such that  $|A - B| \leq \epsilon B$ , using total space  $O(\log(Mn) \log(1/\delta)/\epsilon^2)$  and  $O(\log^{O(1)}(Mn) \log(1/\delta)/\epsilon^2)$  time to process each item. The data streams may be interleaved in an arbitrary (adversarial) order. Here space usage is measured in number of bits and time in number of bit operations.

The main technical innovation used in this algorithm is a limited-independence random-variable construction that may prove useful in other contexts:

A family  $\{V_j(s)\}$  of uniform  $\pm 1$ -valued random variables is called *range-summable* if  $\sum_{j=0}^{c-1} V_j(s)$  can be computed in time  $\text{polylog}(c)$  for all seeds  $s$ . We construct range-summable families of random variables that are  $n^2$ -bad 4-wise independent.<sup>3</sup>

The property of  $n^2$ -bad 4-wise independence suffices for the time- and space-bounds on our algorithm. One can construct a truly 4-wise (in fact, 7-wise) independent range-summable family of random variables based on second-order Reed–Muller codes ([RS99]; for details about second-order Reed–Muller codes, see [MS77]), but the efficiency of the range summation seems to be significantly worse than it is in our construction.

The rest of this paper is organized as follows. In section 2, we give precise statements of our “streaming” model of computation and complexity measures for streaming and sketching algorithms. In section 3, we present our main technical results. Section 4 explains the relationship of our algorithm to other recent work, including that of Broder et al. [BCFM00] on sketching and that of Alon, Matias, and Szegedy [AMS99] and Alon et al. [AGMS99] on frequency moments.

**2. Models of computation.** Our model is closely related to that of Henzinger, Raghavan, and Rajagopalan [HRR98]. We also describe a related sketch model that has been used, e.g., in [BCFM00].

**2.1. The streaming model.** As in [HRR98], a *data stream* is a sequence of data items  $\sigma_1, \sigma_2, \dots, \sigma_n$  such that, on each *pass* through the stream, the items are read once in increasing order of their indices. We assume the items  $\sigma_i$  come from a set of size  $M$ , so that each  $\sigma_i$  has size  $\log M$ . In our computational model, we assume that the input stream consists of one or more data streams. We focus on two resources—the *work space* required in bits and the *time to process* each item in the stream. An algorithm will typically also require pre- and postprocessing time, but usually applications can afford more time for these tasks. For the algorithms in this paper, the pre- and postprocessing time is comparable to the per-item time and is not considered further.

**DEFINITION 1.** *The complexity class  $\text{PASST}(s(\delta, \epsilon, n, M), t(\delta, \epsilon, n, M))$  (to be read as “probably approximately correct streaming space complexity  $O(s(\delta, \epsilon, n, M))$  and time complexity  $O(t(\delta, \epsilon, n, M))$ ”) contains those functions  $f$  on domain  $X^n$ , where  $|X| = M$ , for which one can output a random variable  $R$  such that  $|R - f| < \epsilon f$  with probability at least  $1 - \delta$ , and computation of  $R$  can be done by making a single pass over an instance  $x \in X^n$ , presented in a stream, using total workspace  $O(s(\delta, \epsilon, n, M))$  and taking time  $O(t(\delta, \epsilon, n, M))$  to process each item.*

*If  $s = t$ , we also write  $\text{PASST}(s)$  for  $\text{PASST}(s, t)$ .*

<sup>3</sup>The property of  $n^2$ -bad 4-wise independence is defined precisely in section 3 below.

We will also abuse notation and write  $A \in \text{PASST}(s, t)$  to indicate that an algorithm  $A$  for  $f$  witnesses that  $f \in \text{PASST}(s, t)$ .

Thus  $f$  is a function of a single input that has  $n$  *elements* or *components*. We allow the input elements of  $f$  to be presented in any order in the stream; thus, an input item will be of the form “the  $j$ th input element value is  $a_j$ .” For example, a fragment of the stream representing  $f$  might look like  $\cdots (5, 2)(3, 7)(7, 4)(2, 6) \cdots$ , and this is interpreted as  $f(5) = 2$ ,  $f(3) = 7$ , etc. Note that the input to  $f$  is considered to be static—in a properly formed input stream, at most one item specifies the value of  $a_j$ . Thus the length of the input stream is  $n$  (items) for our algorithms.<sup>4</sup> Other variants of input streams are possible, in which input values may change (repeatedly) throughout the stream, or in which the input comes in a nonarbitrary order (e.g., in sorted order or random order). We do not consider these variations in this paper.

**2.2. The sketch model.** Sketches were used in [BCFM00] to check whether two documents are nearly duplicates. A sketch can also be regarded as a *synopsis data structure* [GM99].

**DEFINITION 2.** *Let  $X$  be a set containing at most  $M$  items. The complexity class  $\text{PAS}(s(\delta, \epsilon, n, M))$  (to be read as “probably approximately correct sketch complexity  $s(\delta, \epsilon, n, M)$ ”) contains those functions  $f : X^n \times X^n \rightarrow Z$  of two inputs for which there exists a set  $S$  of size  $2^{O(s)}$ , a randomized sketch function  $h : X^n \rightarrow S$ , and a randomized reconstruction function  $\rho : S \times S \rightarrow Z$  such that, for all  $x_1, x_2 \in X^n$ , with probability at least  $1 - \delta$ ,  $|\rho(h(x_1), h(x_2)) - f(x_1, x_2)| < \epsilon f(x_1, x_2)$ .*

By “randomized function” of  $k$  inputs, we mean a function of  $k+1$  variables. The first input is distinguished as the source of randomness. It is not necessary that, for all settings of the last  $k$  inputs and for most settings of the first input, the function outputs the same value.

Note that we can also define the sketch complexity of a function  $f : X \times Y \rightarrow Z$  for  $X \neq Y$ . There may be two different sketch functions involved.

There are connections between the sketch model and the streaming model. Let  $XY$  denote the set of concatenations of  $x \in X$  with  $y \in Y$ . It has been noted in [KN97] and elsewhere that a function on  $XY$  with low streaming complexity also has low one-round communication complexity (regarded as a function on  $X \times Y$ ), because it suffices to communicate the memory contents of the hypothesized streaming algorithm after reading the  $X$  part of the input. Sometimes one can also produce a low-sketch-complexity algorithm from an algorithm with low streaming complexity. Our main result is an example.

Also, in practice, it may be useful for the sketch function  $h$  to have low streaming complexity. If the set  $X$  is large enough to warrant sketching, then it may also warrant processing by an efficient streaming algorithm.

Formally, we have the following.

**THEOREM 3.** *If  $f \in \text{PAS}(s(\delta, \epsilon, n, M))$  via sketch function*

$$h \in \text{PASST}(s(\delta, \epsilon, n, M), t(\delta, \epsilon, n, M)),$$

*then  $f \in \text{PASST}(2s(\delta, \epsilon, 2n, M), t(\delta, \epsilon, 2n, M))$ , where we identify  $f : X^n \times X^n \rightarrow Z$  with  $f : X^{2n} \rightarrow Z$  in the natural way.*

<sup>4</sup>It turns out, however, that our algorithms will work if, by convention, we define  $a_j$  to be zero if no stream item specifies the value of  $a_j$ . Thus the length of the input stream may be considerably less than  $n$ . Our streaming algorithm for the  $L^1$ -distance between two vectors actually, at each point during the stream, can approximate the  $L^1$ -distance between the vectors seen thus far, regarding unseen inputs as zero.

We will state our time bounds in terms of  $\text{field}(D)$ , the time necessary to perform a single arithmetic operation in a field of size  $2^D$ . Naïve field-arithmetic algorithms guarantee that  $\text{field}(D) = O(D^2)$ .

**3. The  $L^1$ -difference of functions.**

**3.1. Our approach.** We consider the following problem. The input stream is a sequence of tuples of the form  $(i, a_i, +1)$  or  $(i, b_i, -1)$  such that, for each  $i$  in the universe  $[n]$ , there is at most one tuple of the form  $(i, a_i, +1)$  and at most one tuple of the form  $(i, b_i, -1)$ , and  $a_i$  and  $b_i$  are nonnegative integers. If there is no tuple of the form  $(i, a_i, +1)$ , then define  $a_i$  to be zero for our analysis, and similarly for  $b_i$ . Also note that, in general, a small-space streaming algorithm cannot know for which  $i$ 's the tuple  $(i, a_i, +1)$  does not appear. The goal is to approximate the value of  $F_1 = \sum |a_i - b_i|$  to within  $\pm\epsilon F_1$ , with probability at least  $1 - \delta$ .

Let  $M$  be an upper bound on  $a_i$  and  $b_i$ . We assume that  $n$  and  $M$  are known in advance; in section 3.7, we discuss small modifications that can be made when either of these is not known in advance.

We first present an intuitive exposition of the algorithm. Suppose that, for each type  $i$ , we can define a family of  $M$   $\pm 1$ -valued random variables  $R_{i,j}, j = 0, 1, \dots, (M - 1)$ , with independence properties to be specified later. When we encounter a tuple of the form  $(i, a_i, +1)$ , we add  $\sum_{j=0}^{a_i-1} R_{i,j}$  to a running sum  $z$ , and when we encounter a tuple of the form  $(i, b_i, -1)$ , we subtract  $\sum_{j=0}^{b_i-1} R_{i,j}$  from  $z$ . The overall effect on  $z$  is to cancel the first  $\min(a_i, b_i)$  random variables leaving the sum of the remaining  $|a_i - b_i|$  random variables. Finally, consider  $z^2$ . There are exactly  $\sum_{i=1}^n |a_i - b_i|$  terms that are squares of random variables, and these terms contribute exactly the desired quantity  $F_1$  to  $z^2$ . If the cross terms  $R_{i,j}R_{k,l}$  with  $\{i, j\} \neq \{k, l\}$  contribute very little, then  $z^2$  is a good approximation to  $F_1$ .

Pairwise independence of the random variables in question will ensure that the expected contribution from these cross terms is 0, and 4-wise independence will ensure that the variance is small, thus ensuring that the cross terms contribute very little with high probability. Therefore, we would ideally like our random variables to be 4-wise independent. In addition, as seen above, we want to be able to compute sums of the form  $\sum_{j=0}^c R_{i,j}$  efficiently. In order to compute these sums very efficiently, our construction produces random-variable families that deviate slightly from 4-wise independence.

We now develop a more formal treatment of the above. We start with the definition that captures the properties desired of the family of random variables corresponding to one type  $i$ . We will show how to construct random variables that satisfy this definition. Later, we extend this to show how to construct random-variable families to handle more than one type.

**3.2. Construction of random variable families.**

**DEFINITION 4.** A family  $\{V_j(s)\}$  of uniform  $\pm 1$ -valued random variables with seed  $s$  (chosen at random from some set  $S$  of seeds) is called range-summable,  $n^2$ -bad 4-wise independent if the following properties are satisfied:

1. The family  $\{V_j(s)\}$  is 3-wise independent, i.e., for all distinct  $j_1, j_2, j_3$ , for all  $a, b, c \in \{+1, -1\}$ ,

$$\Pr_s[V_{j_1}(s) = a | V_{j_2}(s) = b \wedge V_{j_3}(s) = c] = \Pr_s[V_{j_1}(s) = a].$$

2. For all  $s$ ,  $\sum_{j=0}^{c-1} V_j(s)$  can be computed in time polylogarithmic in  $c$ .

3. For all  $a < b$ ,

$$E \left[ \left( \sum_{j=a}^{b-1} V_j(s) \right)^4 \right] = O((b-a)^2).$$

In property 3 and in similar expressions throughout the rest of this paper, the expectation is computed over  $s$ .

Note that even for 4-wise independent random variables, the sum in property 3 is  $\Theta((b-a)^2)$  because of terms of the form  $V_j^2(s)V_k^2(s)$ . Thus, property 3 does not represent a significant weakening of 4-wise independence. On the other hand, we do not know of a construction using 4-wise independent random variables that matches ours in efficiency with regard to property 2.

We now describe our construction. This is the main technical innovation of our paper. It is also a significant point of departure from the work on frequency moments by Alon et al. [AMS99]. The relationship between our algorithm and the frequency-moment algorithms is explained in section 4.

We will construct a single family of  $M$  random variables  $V_j(s)$ ,  $0 \leq j < M$ , such that, for all  $c \leq M$ , one can compute  $\sum_{j=0}^{c-1} V_j(s)$  quickly. In the discussion that follows,  $\oplus$  represents boolean exclusive-or, and  $\vee$  represents boolean or. Logarithms in this paper are always to the base 2.

Suppose, without much loss of generality, that  $M$  is a power of 2. Let  $H_{(\log M)}$  be the matrix with  $M$  columns and  $\log M$  rows such that the  $j$ th column is the binary expansion of  $j$ . For example,

$$H_{(3)} = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

Let  $\hat{H}_{(\log M)}$  be formed from  $H_{(\log M)}$  by adding a row of 1's at the top.

$$\hat{H}_{(3)} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

We will index the  $\log M + 1$  rows of  $\hat{H}$  starting with  $-1$  for the row of all 1's, then 0 for the row consisting of the  $2^0$ -bits of the binary expansions, and continue consecutively up to the  $(\log(M) - 1)$ st row. We will left multiply  $\hat{H}$  by a seed  $s$  of length  $\log M + 1$  and use the same indexing scheme for bits of  $s$  as for rows of  $\hat{H}$ . We will also refer to the last bit of  $s$  and the last row of  $\hat{H}$ , where "last" means  $(\log M - 1)$ st, as the "most significant."

Given a seed  $s \in \{0, 1\}^{\log M + 1}$ , let  $s \cdot \hat{H}_j$  denote the inner product over  $\mathbb{Z}_2$  of  $s$  with the  $j$ th column of  $\hat{H}$ . Let  $i_k$  denote the coefficient of  $2^k$  in the binary expansion of  $i$ . Define  $f(i)$  by<sup>5</sup>

$$(3.1) \quad f(i) = (i_0 \vee i_1) \oplus (i_2 \vee i_3) \oplus \cdots \oplus (i_{\log M - 2} \vee i_{\log M - 1}).$$

<sup>5</sup>Here and henceforth, we will actually assume that  $M$  is a power of 4 to simplify the exposition.

Thus, the sequence  $p$  of values  $f(i)$ ,  $i = 0, 1, 2, \dots$ , is

0111 1000 1000 1000 1000 0111 0111 0111 1000 0111 0111 0111 1000 0111 0111 0111  $\dots$

and can be obtained as the string  $p_{\log M}$  by starting with  $p_0 = 0$  and putting  $p_{k+2} = p_k \overline{p_k} \overline{p_k} \overline{p_k}$ , where  $\overline{\pi}$  denotes the bitwise negation of the pattern  $\pi$ . Finally, put  $V_j(s) = (-1)^{(s \cdot \hat{H}_j) + f(j)}$ .

PROPOSITION 5. *The quantity  $\sum_{j=0}^{c-1} V_j(s)$  can be computed in time  $O(\log(c))$ .*

*Proof.* First assume that  $c$  is a power of 4. If  $c < M$ , then the first  $c$  columns of  $\hat{H}_{\log M}$  have the form  $(\hat{H}_0^{\log c})$ , and we can reduce our problem to one in which we truncate  $s$  to include only the first  $1 + \log c$  bits. We may thus assume that  $c = M$ . Then  $\hat{H}_{(\log M)}$  is given recursively by

$$\hat{H}_{(\log M)} = \begin{bmatrix} 1 \dots 1 & 1 \dots 1 & 1 \dots 1 & 1 \dots 1 \\ H_{(\log M-2)} & H_{(\log M-2)} & H_{(\log M-2)} & H_{(\log M-2)} \\ 0 \dots 0 & 1 \dots 1 & 0 \dots 0 & 1 \dots 1 \\ 0 \dots 0 & 0 \dots 0 & 1 \dots 1 & 1 \dots 1 \end{bmatrix}.$$

Also, note that the first  $M$  bits of  $p$  have the form

$$p_{\log M} = p_{\log M-2} \overline{p_{\log M-2}} \overline{p_{\log M-2}} \overline{p_{\log M-2}}.$$

Let  $s'$  be a string of length  $\log M - 2$  that is equal to  $s$  without the  $-1$ st bit and without the two most significant bits, and let  $f'$  denote the fraction of 1's in  $s' \cdot H_{(\log M-2)}$ . Also, for bits  $b_1, b_2$ , let  $f_{b_1 b_2}$  denote the fraction of 1's in

$$s \cdot \begin{bmatrix} 1 \dots 1 \\ H_{(\log M-2)} \\ b_1 \dots b_1 \\ b_2 \dots b_2 \end{bmatrix}.$$

Then  $f_{b_1 b_2} = f'$  or  $f_{b_1 b_2} = 1 - f'$ , depending on  $b_1, b_2$ , and the three bits of  $s$  dropped from  $s'$  (namely,  $-1$ ,  $\log M - 2$ , and  $\log M - 1$ ). Recursively compute  $f'$ , and use the value to compute all the  $f_{b_1 b_2}$ 's and, from that, the number of 1's in  $\sum_{j=0}^{c-1} V_j(s)$ . This procedure requires recursive calls of depth that is logarithmic in  $c$ .

Similarly, one can compute  $\sum_{j=q4^r}^{(q+1)4^r-1} V_j(s)$ .

Finally, if  $c$  is not a power of 4, write the interval  $\{0, \dots, (c-1)\} = [0, c)$  as the disjoint union of at most  $O(\log(c))$  intervals, each of the form  $[q4^r, (q+1)4^r)$ . Use the above technique to compute the fraction of  $V$ 's equal to 1 over each subinterval, and then combine. If one is careful to perform the procedure bottom up, the entire procedure requires just  $\log(c)$  recursive calls, not  $\log^2(c)$  calls. For example, suppose  $c = 22$ . Write  $[0, 22)$  as  $[0, 16) \cup [16, 20) \cup [20, 21) \cup [21, 22)$ . A naïve way to proceed would be to perform recursive calls 3 deep to compute  $\sum_{j=0}^{15} V_j(s)$ , then calls 2 deep for  $\sum_{j=16}^{19} V_j(s)$ , then 1 deep for each of  $V_{20}(s)$  and  $V_{21}(s)$ . It is better to proceed as follows: compute  $V_{20}(s)$  directly (by taking the dot product of the first  $O(\log(c))$  bits of  $s$  with the first  $O(\log(c))$  rows of column 20 in  $\hat{H}_{\log(M)}$ , then adding  $f(20)$ ); use this value to compute  $V_{21}(s)$  and  $V_{16}(s)$  (each of these computations requires looking at just  $O(1)$  bits of  $s$ —in this case,  $V_{21}(s)$  is the sum of  $V_{20}(s)$  and the  $2^0$  bit of  $s$ , and  $V_{16}(s)$  is the sum of  $V_{20}(s)$  and the  $2^2$  bit of  $s$ ); then use  $V_{16}(s)$  to compute  $\sum_{j=16}^{19} V_j(s)$ ; and finally use  $\sum_{j=16}^{19} V_j(s)$  to compute  $\sum_{j=0}^3 V_j(s)$  and, from that,

$\sum_{j=0}^{15} V_j(s)$ . For  $j < c$ , computing the value of a single  $V_j(s)$  takes time  $O(\log(c))$ , and the overhead in each recursive call takes constant time. Thus, altogether, computing a range sum of  $V$ 's requires time  $O(\log(c))$ .  $\square$

We now show that this construction yields a family of random variables that is  $n^2$ -bad 4-wise independent. The fact that  $\{V_j(s)\}$  is three-wise independent is shown in [AS92].

PROPOSITION 6. *For all  $a < b$ , we have*

$$E \left[ \left( \sum_{j=a}^{b-1} V_j(s) \right)^4 \right] \leq 5(b-a)^2.$$

*Proof.* First, note that, for some tuples  $(j_1, j_2, j_3, j_4)$ , columns  $j_1, j_2, j_3$ , and  $j_4$  of  $\hat{H}$  are independent. These tuples do not contribute to the expectation on the left of the inequality, because for each desired outcome  $(v_1, v_2, v_3, v_4)$ , the sets

$$S_{(v_1, v_2, v_3, v_4)} = \{s : (V_{j_1}(s), V_{j_2}(s), V_{j_3}(s), V_{j_4}(s)) = (v_1, v_2, v_3, v_4)\}$$

have the same size by linear algebra.

Second, observe that, because any three columns of  $\hat{H}$  are independent, if the columns  $\hat{H}_{j_1}, \hat{H}_{j_2}, \hat{H}_{j_3}$ , and  $\hat{H}_{j_4}$  are dependent, then their mod 2 sum is zero. Thus a dependent tuple has one of 3 basic forms—all four columns are identical; there are two pairs of distinct columns; or all four columns are distinct. In the case of dependent tuples, the seed  $s$  is irrelevant to the product  $V_{j_1}(s)V_{j_2}(s)V_{j_3}(s)V_{j_4}(s)$  because

$$\begin{aligned} (3.2) \quad & V_{j_1}(s)V_{j_2}(s)V_{j_3}(s)V_{j_4}(s) \\ &= (-1)^{(s \cdot \hat{H}_{j_1}) + f(j_1)} \cdot (-1)^{(s \cdot \hat{H}_{j_2}) + f(j_2)} \cdot (-1)^{(s \cdot \hat{H}_{j_3}) + f(j_3)} \cdot (-1)^{(s \cdot \hat{H}_{j_4}) + f(j_4)} \\ (3.3) \quad &= (-1)^{f(j_1) + f(j_2) + f(j_3) + f(j_4)}. \end{aligned}$$

Line (3.3) follows from the fact that the columns  $\hat{H}_{j_1}, \hat{H}_{j_2}, \hat{H}_{j_3}$ , and  $\hat{H}_{j_4}$  sum to zero. Thus it is sufficient to show that

$$U(a, b) \triangleq \sum_{\substack{a \leq j_1, j_2, j_3, j_4 < b \\ j_1 \oplus j_2 \oplus j_3 \oplus j_4 = 0}} (-1)^{f(j_1) + f(j_2) + f(j_3) + f(j_4)} \leq K(b-a)^2$$

for some constant  $K$ . From Theorem 9 below, we can see that  $K \leq 5$ , and thus the proposition holds.  $\square$

We shall now provide upper bounds for the quantity  $U(a, b)$  defined in the proof of Proposition 6. We will give two bounds for  $U(a, b)$ —a simply derived though poor bound (Theorem 8) and a more tediously obtained but much tighter bound (Theorem 9). Before presenting these bounds, we first prove a lemma that is used later in the proofs.

LEMMA 7.  $U(4a, 4b) \leq 16U(a, b)$ .

*Proof.* Let  $(j_1, j_2, j_3, j_4)$  be a dependent tuple in  $[4a, 4b]^4$ . Consider the two least significant bits of the  $j$ 's. We will say that the tuple is *odd* if no two of its members have the same pair of least significant bits; otherwise, we will say that the tuple is *even*. There are 64 possibilities making the columns dependent, because we can choose two bits from each of the first three columns arbitrarily, and this forces a unique choice of the bits from the last column. Of these, 24 = 4! are odd and 40 are even. (The 40 even tuples arise from 4 tuples in which all columns have identical bits and 36 tuples



in which the 4 columns are paired in one of 6 ways, and the two pairs are given two distinct values out of the 4 possible in one of 6 ways.)

Note that a dependent tuple is odd if and only if there are an odd number of  $i$ 's for which  $(j_i)_0 \vee (j_i)_1 = 1$ . Thus, if  $(j_1, j_2, j_3, j_4)$  is an odd dependent tuple and  $(j'_1, j'_2, j'_3, j'_4)$  is an even dependent tuple where  $j'_i$  agrees with  $j_i$  on all bits except possibly the two least significant, then the contributions of these two tuples to the  $U(4a, 4b)$  cancel out. Therefore, given an odd tuple  $(j_1, j_2, j_3, j_4)$ , pair it with  $(j'_1, j'_2, j'_3, j'_4)$  as above. Because  $4a$  and  $4b$  are multiples of 4,  $(j'_1, j'_2, j'_3, j'_4)$  will be in the correct range. If  $(j_1, j_2, j_3, j_4)$  is a tuple having one of the 16 other (even) configurations of the two least significant bits, attempt to pair it inductively with  $(j'_1, j'_2, j'_3, j'_4)$  such that  $j_i$  and  $j'_i$  have the same two least significant bits. Thus  $U(4a, 4b) \leq 16U(a, b)$ .  $\square$

We now give a simple argument that  $U(a, b) \leq 27(b-a)^2$ .

**THEOREM 8.**  $U(a, b) \leq 27(b-a)^2$ .

*Proof.* Given  $a$  and  $b$ , find  $r$  with  $a, b \leq 4^r$ . Let  $\alpha$  be the smallest multiple of 4 that is at least  $a$  and let  $\beta$  be the largest multiple of 4 that is at most  $b$ . Then  $U(a, b)$  is at most  $U(\alpha, \beta)$  plus the number of tuples having at least one column in  $[a, \alpha) \cup [\beta, b)$ . We will handle the first term inductively; we now count the number of tuples having at least one column in  $[a, \alpha) \cup [\beta, b)$ . First, there are 4 ways to choose one of  $j_1, j_2, j_3$ , and  $j_4$  to be in  $[a, \alpha) \cup [\beta, b)$ . (Having paid the factor 4, we now call this column  $j_1$ .) There are at most 6 ways to choose  $j_1 \in [a, \alpha) \cup [\beta, b)$ . There are at most  $(b-a)^2$  ways to choose  $j_2$  and  $j_3$  in  $[a, b)$ . Finally, once  $j_1, j_2$ , and  $j_3$  are fixed, there is at most one way to choose  $j_4$  to make  $(j_1, j_2, j_3, j_4)$  dependent. (Note that  $j_2, j_3$ , and  $j_4$  play symmetric roles.) This gives  $24(b-a)^2$  tuples altogether. Thus we conclude that

$$\begin{aligned} U(a, b) &\leq U(4 \lceil a/4 \rceil, 4 \lfloor b/4 \rfloor) + 24(b-a)^2 \\ &= 16U(\lceil a/4 \rceil, \lfloor b/4 \rfloor) + 24(b-a)^2 \\ &\leq 16U(4 \lceil \lceil a/4 \rceil / 4 \rceil, 4 \lfloor \lfloor b/4 \rfloor / 4 \rfloor) + 24(\lfloor b/4 \rfloor - \lceil a/4 \rceil)^2 + 24(b-a)^2 \\ &= 16^2 U(\lceil \lceil \lceil a/4 \rceil / 4 \rceil \rceil, \lfloor \lfloor \lfloor b/4 \rfloor / 4 \rfloor \rfloor) + 24(\lfloor b/4 \rfloor - \lceil a/4 \rceil)^2 + 24(b-a)^2 \\ &\quad \vdots \\ &\leq 16^{\lfloor \log_4(b-a) \rfloor} + 24(b-a)^2 [1 + 1/16 + 1/16^2 \dots] \\ &\leq (b-a)^2 + 24(b-a)^2(16/15) \\ &\leq 27(b-a)^2, \end{aligned}$$

as desired.  $\square$

We now give a more involved analysis that lets us improve the bound.

**THEOREM 9.**  $U(a, b) \leq 5(b-a)^2$ .

*Proof.* Define  $A(a, b)$  to be  $\frac{U(a, b)}{(b-a)^2}$ . Note that it is sufficient to show that  $\sup_{a, b} A(a, b) \leq 5$ . We will give a recurrence for  $A(a, b)$  and discuss a computer search over  $a, b$  with  $b-a$  small that yields a bound better than immediately available from the recurrence.

We first assume that  $b-a \geq 16$ . Let  $a'$  be the smallest multiple of 4 that is at least  $a$ , and let  $b'$  be the greatest multiple of 4 that is at most  $b$ . (Because  $b-a \geq 16$ , it follows that  $a \leq a' < b' \leq b$ .) The number of unpaired tuples in  $[a, b)^4$  is at most the number of unpaired tuples in  $[a', b')^4$  plus the number of unpaired tuples having at least one column in  $[a, a') \cup [b', b)$ . The number of unpaired tuples in  $[a', b')^4$  is  $U(a', b') \leq 16U(a'/4, b'/4) = A(a'/4, b'/4)(b'-a')^2$ . We now count the number of unpaired tuples having at least one column in  $[a, a') \cup [b', b)$ .

There are at most  $36(b-a)$  tuples such that two of the columns are identical and in  $[a, a'] \cup [b', b)$ , and the two other columns are identical. (The four columns may or may not all be equal. Note that a factor  $\max\binom{4}{1}, \binom{4}{2} = 6$  is needed to assign the four columns to the one or two values.) We now count the tuples whose columns are all different. Pick an assignment of roles for the columns, which contributes a factor 24. There are at most 6 ways to pick  $j_1$  in  $[a, a'] \cup [b', b)$ . Next we will consider the choices of the pair  $j_2$  and  $j_3$ , which in turn will determine  $j_4$  uniquely. We will argue that, for most pairs  $(j_2, j_3)$ , by making local changes to  $j_2$  and  $j_3$ , we can produce another tuple of columns  $(j_1, j'_2, j'_3, j_4)$  that cancels out with  $(j_1, j_2, j_3, j_4)$ . The main difficulty will be to ensure that the columns  $j'_2$  and  $j'_3$  are in the correct range.

The local change strategy is as follows: Let  $k$  be the index of the least significant bit on which  $j_2$  and  $j_3$  disagree. Let  $k'$  be the index of the “mate” of  $k$ , i.e., the bit that is “or”ed with the  $k$ th bit in the computation of  $f(j_2)$  or  $f(j_3)$ . The columns  $j'_2$  and  $j'_3$  will be obtained by toggling the  $k'$ th bit of  $j_2$  and  $j_3$ , respectively. We have to check that the tuples  $(j_1, j_2, j_3, j_4)$  and  $(j_1, j'_2, j'_3, j_4)$  have opposite parity. To see this, assume without loss of generality that the  $k$ th bits of  $j_2$  and  $j_3$  are 0 and 1, respectively. Then the disjunction in expansion (3.1) corresponding to bits  $k$  and  $k'$  for each of  $f(j'_3)$  and  $f(j_3)$  is 1, because of the 1 in bit  $k$ , but the  $k$ - $k'$  disjunction for  $f(j'_2)$  and  $f(j_2)$  differ, because the  $k$ th bits are zero, but the  $k'$ th bits differ. All the other disjunctions are the same in  $f(j_2)$  as  $f(j'_2)$  and in  $f(j_3)$  as  $f(j'_3)$ . Note also that  $(j_1, j'_2, j'_3, j_4)$  is a dependent tuple whenever  $(j_1, j_2, j_3, j_4)$  is a dependent tuple.

Next we have to determine the conditions for  $j'_2$  and  $j'_3$  to be between  $a$  and  $b$ . We will consider the situation in which one of these columns falls below the lower bound  $a$  and appeal to symmetry for the situation in which one column is at least  $b$ .

For two columns  $x, y$ , let  $eq(x, y) = \ell$  if the most significant bit in which they differ has index  $\ell$ .

Because we have already paid a factor of 24 for the assignment of roles, choose roles such that  $eq(a, j_2) \geq eq(a, j_3)$ . Suppose all columns are  $r$  bits long. There are at most  $2^{r-\ell}$  vectors  $j_2$  for which  $eq(a, j_2)$  is  $\ell$ . For each such vector  $j_2$ , there are at most  $\lceil (b-a)/2^{r-\ell-1} \rceil - 1 \leq (b-a)/2^{r-\ell-1}$  vectors  $j_3$  distinct from  $j_2$  that agree with  $j_2$  on the least significant  $r-\ell-1$  bits. (Such  $(j_2, j_3)$  pairs may require a toggling of the first  $\ell$  bits that could cause  $j'_2$  or  $j'_3$  to drop below  $a$ .) Thus the total number of problem pairs with respect to the lower bound is at most  $2(b-a)$  for each choice of  $\ell$ . Over all choices of  $\ell$  this number is at most  $2(b-a)\log(b-a)$ . By symmetry, the number of problem pairs with respect to  $b$  is also at most  $2(b-a)\log(b-a)$ . Thus there are at most  $4(b-a)\log(b-a)$  ways to pick pairs  $j_2$  and  $j_3$  that do not cancel out. Combining with the 24 ways of assigning roles and the 6 ways of picking  $j_1$  we find that there are at most  $576(b-a)\log(b-a)$  tuples that do not get canceled. Thus, including the dependent tuples with repeated columns (at most  $36(b-a)$ ), we get

$$\begin{aligned} U(a, b) &\leq U(a', b') + 576(b-a)\log(b-a) + 36(b-a) \\ &\leq U(a', b') + 585(b-a)\log(b-a) \\ &\leq 16U(a'/4, b'/4) + 585(b-a)\log(b-a) \\ &= 16A(a'/4, b'/4)(b'/4 - a'/4)^2 + 585(b-a)\log(b-a) \\ &= A(a'/4, b'/4)(b' - a')^2 + 585(b-a)\log(b-a) \\ &\leq A(a'/4, b'/4)(b-a)^2 + 585(b-a)\log(b-a), \end{aligned}$$

and so

$$A(a, b) \leq A(\lceil a/4 \rceil, \lfloor b/4 \rfloor) + 585 \frac{\log(b-a)}{(b-a)}.$$

Let

$$D_i = \sup_{4^i < (b-a) \leq 4^{i+1}} A(a, b).$$

For each  $C \geq 2$ , we have the recurrence

$$(3.4) \quad D_i \leq \begin{cases} D_{i-1} + 585 \cdot \frac{2^i}{4^i}, & i \geq C, \\ M_C, & i < C, \end{cases}$$

where  $M_C = \max(D_0, \dots, D_{C-1})$  is a bound on  $A(a, b)$  over  $4^i < b - a \leq 4^{i+1}$  for  $i < C$ , i.e.,  $b - a \leq 4^C$ . We want to find a minimal solution. We will discuss below how we establish  $M_C$  precisely using an exhaustive computer search.

Recurrence (3.4) has a solution

$$\begin{aligned} D_i &= M_C + 585 \sum_{j=C}^i \frac{2^j}{4^j} \\ &\leq M_C + 2 \cdot 585 \frac{C + 1/3}{3 \cdot 4^{C-1}}, \end{aligned}$$

where the empty sum is taken to be zero. If we put  $C = 6$ , we get

$$D_i \leq M_6 + 2.413,$$

whence, for all  $a, b$ ,  $A(a, b) \leq M_6 + 2.413$ .

It remains to evaluate  $M_6$ . We first show that it is sufficient to consider a finite number of pairs  $\{a, b\}$  even though the definition of  $M_6$  requires it to be a supremum over an infinite number of pairs. We then use a computer search to find  $M_6$ .

CLAIM 10. *The value  $M_C = \max_{b-a \leq 4^C = 2^{2C}} A(a, b)$  is at most*

$$M'_C = \max_{\substack{a \leq 2^{2C-1} \\ b \leq a + 2^{2C}}} A(a, b).$$

*Proof.* Suppose  $(a, b)$  is a pair with  $b - a \leq 2^{2C}$  but  $a > 2^{2C-1}$ . We produce  $a', b'$  with  $a' < a$  and  $b' < b$  such that  $b' - a' = b - a$  and  $A(a', b') = A(a, b)$ . The claim follows.

First, we show that if  $a, b \leq 2^r$ , then  $U(a, b) = U(2^r - b, 2^r - a)$ . Given a tuple  $(j_1, j_2, j_3, j_4) \in [a, b]^4$ , write each  $j$  with  $r$  bits, padding with leading zeros if necessary. Form  $j'_i = 2^r - 1 - j_i$  by negating all the bits in  $j_i$ . This procedure toggles the parity of the  $k$ - $k'$  disjunct in the expansion of  $f(j)$  when the  $k$ - $k'$  bits are 00 or 11; for each  $k$ , in a dependent tuple, there are an even number of columns that are 00 or 11 in bits  $k$  and  $k'$  and an even number of columns that are 01 or 10 there. It follows that  $(j_1, j_2, j_3, j_4)$  and  $(j'_1, j'_2, j'_3, j'_4)$  have the same parity. Note also that this mapping is a bijection from  $[a, b]$  to  $[2^r - b, 2^r - a]$ . From this we can conclude that  $U(a, b) = U(2^r - b, 2^r - a)$ . Similarly, if  $a, b \leq 3 \cdot 2^r$ , then  $U(a, b) = U(3 \cdot 2^r - b, 3 \cdot 2^r - a)$ .

Finally:

- if  $2^{2C-1} < a \leq 2^{2C}$ , then
  - if  $2^{2C-1} < b \leq 2^{2C}$ , then put  $(a', b') = (2^{2C} - b, 2^{2C} - a)$ ;
  - if  $2^{2C} < b \leq 3 \cdot 2^{2C-1}$ , then put  $(a', b') = (3 \cdot 2^{2C-1} - b, 3 \cdot 2^{2C-1} - a)$ ;

- if  $3 \cdot 2^{2C-1} < b$ , then note that  $b < a + 2^{2C} \leq 2^{2C+1}$ . Put  $(a', b') = (2^{2C+1} - b, 2^{2C+1} - a)$ .
- if  $2^{2C} < a$ , then find  $q > 2C$  with  $2^q < a \leq 2^{q+1}$ ;
  - if  $b \leq 2^{q+1}$ , then  $2^q < a < b \leq 2^{q+1}$ . Put  $(a', b') = (2^{q+1} - b, 2^{q+1} - a)$ ;
  - otherwise,  $2^{q+1} < b \leq a + 2^{2C} \leq 2^{q+1} + 2^q = 3 \cdot 2^q$ . Put  $(a', b') = (3 \cdot 2^q - b, 3 \cdot 2^q - a)$ .

In all cases,  $a' < a$ ,  $b' < b$ , and  $b' - a' = b - a$ .  $\square$

Thus, if we are interested in  $M_6$ , i.e.,  $(a, b)$  with  $b - a \leq 4096$ , we need only consider  $a \leq 2048$ . A computer search was done for  $A(a, b)$  over  $a \leq 2048$  and  $b \leq a + 4096$ , and the maximum is  $2.55334$ . Thus  $A(a, b) \leq 2.56 + 2.413 < 5$ .  $\square$

**3.3. The algorithm.** Recall that for the overall algorithm, we will need to generate a family of random variables for each of the different types. It would be ideal to make these families  $n$ -wise independent, but that would require storing a seed for each of the  $n$  types, which is infeasible. Therefore, we will use short *master seeds* to generate  $n$  different seeds that are 4-wise independent and, from these, compute the  $n$  families of random variables that we will use to get an estimate of  $F_1$ . It will be necessary to repeat this process to achieve the specified values of  $\epsilon$  and  $\delta$ .

For each  $k$ ,  $1 \leq k \leq 3 \log(1/\delta)$  and for each  $\ell$ ,  $1 \leq \ell \leq 8A/\epsilon^2$  (where  $A = 10$  will be justified later), choose a *master seed*  $S_{k,\ell}$  and use  $S_{k,\ell}$  to define a 4-wise independent family  $\{s_{i,k,\ell}\}$  of  $n$  seeds, each of length  $\log M + 1$ . Each seed  $s_{i,k,\ell}$  in turn defines a range-summable,  $n^2$ -bad 4-wise independent family  $\{V_{i,j,k,\ell}\}$  of  $M$  uniform  $\pm 1$ -valued random variables, where  $V_{i,j,k,\ell} \stackrel{\Delta}{=} V_j(s_{i,k,\ell})$ .

We can use any standard construction to define a family of seeds from a master seed. For example, we can use the construction based on BCH codes in [AS92]. Another construction is one in which the master seed is used to define the coefficients of a random degree-3 univariate polynomial over a sufficiently large finite field. We will describe and use this more elementary construction.

Let  $D = \max(\log M + 1, \log n)$ . Choose  $F = GF_{2^D}$  as the finite field. Fix a representation for the elements of  $F$  as bit strings of length  $D$ . Choose a master seed  $S_{k,\ell}$  of length  $4D$  bits uniformly at random, and view these bits as coefficients  $a_3, a_2, a_1, a_0$  of a degree-3 polynomial  $a(x) \in F[x]$ . Now define the  $i$ th seed,  $s_{i,k,\ell} = a(i)$ . It is immediate from basic algebra that these seeds are 4-wise independent and that the individual seeds can be computed in a constant number of field operations over the field  $F$ .

A final point of concern is whether the use of a master seed to generate individual seeds impacts the analysis of the last subsection. There we assumed that the seed for a single family of random variables was chosen uniformly at random among strings of a fixed length. In our construction here, when the master seed is chosen uniformly at random from strings of the correct length, each seed is also distributed uniformly at random, and hence the analysis of the previous subsection still applies.

A more formal, high-level description of the algorithm is given in Figure 1.

**3.4. Correctness.** The proof in this section that the algorithm described in Figure 1 is correct closely follows the one given in [AMS99] for the correctness of their algorithm (see section 4.3).

**THEOREM 11.** *The algorithm described in Figure 1 outputs a random variable  $W = \text{median}_k \text{avg}_\ell Z_{k,\ell}^2$  such that  $|W - F_1| < \epsilon F_1$  with probability at least  $1 - \delta$ .*

---

 ALGORITHM L1( $\langle\langle i, c_i, \theta_i \rangle\rangle$ ).

Initialize:

For  $k = 1$  to  $3 \log(1/\delta)$  do  
   For  $\ell = 1$  to  $(8 \cdot A)/\epsilon^2$  do  
     //For any  $A \geq 10$  —see (3.8) and the end of section 3.2  
     {  $Z_{k,\ell} = 0$   
       pick a master seed  $S_{k,\ell}$  from the  $(k, \ell)$ th sample space }  
     // This implicitly defines  $s_{i,k,\ell}$  for  $0 \leq i < n$  and  
     // in turn implicitly defines  $V_{i,j,k,\ell}$  for  $0 \leq i < n$  and  $0 \leq j < M$ .

For each tuple  $(i, c_i, \theta_i)$  in the input stream do

For  $k = 1$  to  $3 \log(1/\delta)$  do  
   For  $\ell = 1$  to  $(8 \cdot A)/\epsilon^2$  do  
      $Z_{k,\ell} += \theta_i \sum_{j=0}^{c_i-1} V_{i,j,k,\ell}$

Output  $\text{median}_k \text{avg}_\ell Z_{k,\ell}^2$ .

---

 FIG. 1. High level  $L^1$  algorithm.

*Proof.* Note that for each  $j < \min(a_i, b_i)$ , both  $V_{i,j,k,\ell}$  and  $-V_{i,j,k,\ell}$  are added to  $Z_{k\ell}$ , and for  $j \geq \max(a_i, b_i)$ , neither  $V_{i,j,k,\ell}$  nor  $-V_{i,j,k,\ell}$  is added. Thus

$$Z_{k\ell} = \sum_i \sum_{\min(a_i, b_i) \leq j < \max(a_i, b_i)} \pm V_{i,j,k,\ell}.$$

We shall now compute  $E[Z_{k\ell}^2]$  and  $E[Z_{k\ell}^4]$  for each  $k, \ell$ . We shall use the convention that  $\sum_{a \leq i < b} = -\sum_{b \leq i < a}$  if  $b < a$ . For notational convenience, we let  $V_{i,j}$  denote  $V_{i,j,k,\ell}$  in the analysis below.

$$\begin{aligned}
 (3.5) \quad E[Z_{k,\ell}^2] &= E \left[ \left( \sum_i \sum_{j=a_i}^{b_i-1} V_{i,j} \right)^2 \right] \\
 &= E \left[ \left( \sum_{m=1}^{F_1} \pm V_m \right)^2 \right] \\
 &= \sum_{m=1}^{F_1} E[(\pm V_m)^2] + 2 \sum_{1 \leq m < m' < F_1} E[(\pm V_m)(\pm V_{m'})]
 \end{aligned}$$

$$\begin{aligned}
 (3.6) \quad &= \sum_{m=1}^{F_1} 1 \\
 &= F_1,
 \end{aligned}$$

where, in line (3.5), we have relabeled the indices of  $V$ , and, in line (3.6), we have used the pairwise independence of  $V_m$  and  $V_{m'}$  and the fact that the expectation of each of these random variables is 0.

Next, consider

$$E[Z_{k,\ell}^4] = E \left[ \sum_{0 \leq i_1, i_2, i_3, i_4 < n} \sum_{\substack{a_{i_1} \leq j_1 < b_{i_1} \\ a_{i_2} \leq j_2 < b_{i_2} \\ a_{i_3} \leq j_3 < b_{i_3} \\ a_{i_4} \leq j_4 < b_{i_4}}} V_{i_1, j_1} V_{i_2, j_2} V_{i_3, j_3} V_{i_4, j_4} \right].$$

By 3-wise independence and the fact that  $E[V^t] = 0$  for odd  $t$ , the only terms with nonvanishing expectation are of the form  $V_{i,j}^4$  (of which there are  $F_1$  terms),  $V_{i,j}^2 V_{i',j'}^2$  for  $(i,j) \neq (i',j')$  (of which there are  $\binom{4}{2} F_1(F_1-1)$  terms), and  $V_{i_1, j_1} V_{i_2, j_2} V_{i_3, j_3} V_{i_4, j_4}$  for  $(i_1, j_1), (i_2, j_2), (i_3, j_3), (i_4, j_4)$  all different. Suppose, in the third case, that  $i_1, i_2, i_3, i_4$  are not all the same. Let  $X = \prod_{i_m=i_1} V_{i_m, j_m}$  and  $Y = \prod_{i_m \neq i_1} V_{i_m, j_m}$ . Then  $E[X] = 0$  by 3-wise independence of the  $V$ 's, and  $X$  and  $Y$  are independent by 4-wise independence of the seeds  $s_{i,k,\ell}$ . Therefore, if  $(i_1, j_1), (i_2, j_2), (i_3, j_3), (i_4, j_4)$  are all different and  $i_1, i_2, i_3, i_4$  are not all the same,

$$E[V_{i_1, j_1} V_{i_2, j_2} V_{i_3, j_3} V_{i_4, j_4}] = E[XY] = 0.$$

Thus we have

$$\begin{aligned} E[Z_{k,\ell}^4] &\leq F_1 + 6F_1(F_1 - 1) + \sum_i E \left[ \left( \sum_{j=a_i}^{b_i-1} V_{i,j} \right)^4 \right] \\ (3.7) \quad &\leq 6F_1^2 + \sum_i 5(b_i - a_i)^2 \\ &\leq 11F_1^2. \end{aligned}$$

In line (3.7), we used Proposition 6, which shows that our construction of random variables is  $n^2$ -bad 4-wise independent, with constant 5.

Thus

$$(3.8) \quad \text{Var}(Z_{k,\ell}^2) = E[Z_{k,\ell}^4] - E^2[Z_{k,\ell}^2] \leq A \cdot F_1^2$$

for  $A = 10$ . Now, put  $Y_k = \frac{\epsilon^2}{8 \cdot A} \sum_{1 \leq \ell \leq (8 \cdot A)/\epsilon^2} Z_{k,\ell}^2$ . Then  $\text{Var}(Y_k) \leq \frac{\epsilon^2}{8} F_1^2$ . By Chebyshev's inequality,

$$\begin{aligned} \Pr(|Y_k - F_1| > \epsilon F_1) &\leq \frac{\text{Var}(Y_k)}{\epsilon^2 F_1^2} \\ &\leq 1/8. \end{aligned}$$

Put  $W = \text{median}_k Y_k$ . Then  $|W - F_1| > \epsilon F_1$  only if we have  $|Y_k - F_1| > \epsilon F_1$  for at least half of the  $k$ 's. Let  $A_k = 1$  if  $|Y_k - F_1| > \epsilon F_1$  and  $A_k = 0$  otherwise; so, for all  $k$ ,  $E[A_k] \leq 1/8$ . Put  $m = 3 \log(1/\delta)$  and  $A = \sum_{k=1}^m A_k$ ; then  $E[A] \leq m/8$ . By Chernoff's inequality, the probability that  $A \geq m/2 \geq (1+3)E[A]$  is at most

$$\begin{aligned} \left[ \frac{e^3}{(1+3)^{(1+3)}} \right]^{m/8} &\approx 1.374^{-m} \\ &\leq 2^{-m/3} \\ &\leq \delta. \end{aligned}$$

The result follows.  $\square$

**3.5. Cost.**

THEOREM 12. *There is an implementation of Algorithm L1 (in Figure 1) that is in*

$$\text{PASST}(\log(Mn) \log(1/\delta)/\epsilon^2, \text{field}(\log(Mn)) \log(1/\delta)/\epsilon^2).$$

*Proof.* The algorithm stores

- $\log(1/\delta)/\epsilon^2$  random variables  $Z_{k,\ell}$  (called counters) whose values are at most  $Mn$ ;
- master seeds, specifying the seeds and, through the seeds, the values of the  $\pm 1$ -valued random variables  $V_{i,j,k,\ell}$ .

The space to store the counters is  $O(\log(Mn) \log(1/\delta)/\epsilon^2)$ . By our construction, for each  $k, \ell$ , we need  $O(\max(\log M, \log n)) = O(D)$  bits of master seed, and so we would need  $O(D \log(1/\delta)/\epsilon^2)$  bits of storage to store all the master seeds. This is asymptotically the same as the space requirement for storing the counters.

We now consider the cost of processing a single item  $(i, c_i, \pm 1)$ . First, one has to produce the seeds  $s_{i,k,\ell}$  from the master seeds  $S_{k,\ell}$ . For each  $i$ , this involves computing a degree-3 polynomial over  $GF(2^D)$ , which takes time  $O(\text{field}(D))$ . From  $s_{i,k,\ell}$ , we need to compute a range sum  $\sum_{j=0}^{b-1} V_j(s_{i,k,\ell})$ , which can be computed in  $O(\log M)$ . Thus, the overall time complexity is  $O((\text{field}(D) + \log(M)) \log(1/\delta)/\epsilon^2)$ . Under the reasonable assumption that field arithmetic takes at least linear time, the first term dominates, and the complexity is as claimed above.  $\square$

**3.6. Optimality.** Our algorithm is quite efficient in the parameters  $n, M$ , and  $\delta$ , but it requires space quadratic in  $1/\epsilon$ . We now show that for some nontrivial settings of  $M$ , for all large settings of  $n$ , and for all small settings of  $\delta$ , any sketching algorithm that approximates the  $L^1$ -difference to within  $\epsilon$  requires space close to  $1/\epsilon$ . Thus, our algorithm uses space within a polynomial of optimal.

THEOREM 13. *Fix  $M = 2$ . For sufficiently small  $\delta$ , and for any (large)  $\alpha$  and any (small)  $\beta > 0$ , the  $L^1$ -difference problem is not in  $\text{PAS}(\log^\alpha(n)/\epsilon^{1-\beta})$ .*

*A similar result holds in the streaming model.*

*Proof.* We reduce the set-disjointness problem to the  $L^1$ -difference problem.

Recall the set-disjointness problem of communication complexity [KN97]. Alice has a string  $x$ ,  $|x| = n$ , and Bob has a string  $y$ ,  $|y| = n$ , and they want to determine whether there exists a position  $i$  such that the  $i$ th bit of  $x$  and the  $i$ th bit of  $y$  are both 1. Any protocol for this problem, even a randomized protocol, requires  $\Omega(n)$  bits of communication, even under the restriction that there is at most one bit  $i$  with  $x_i = y_i = 1$ . Finally, note that an efficient sketching or streaming algorithm directly yields a protocol with low communication complexity.

Suppose  $L^1 \in \text{PAS}(\log^\alpha(n)/\epsilon^{1-\beta})$ . Put  $\epsilon = 1/(3n)$ . Let  $(a, b)$  be an instance of set disjointness; so, for all  $i$ ,  $a_i, b_i \leq M = 2$ . Note that the symmetric difference of  $a$  and  $b$  as sets is the  $L^1$ -difference of  $a$  and  $b$  as functions. By hypothesis one can, with high probability, approximate the symmetric difference  $|a\Delta b|$  to within  $\epsilon|a\Delta b| < 1/2$  (whence one can, with high probability, compute  $|a\Delta b|$  exactly) using space at most

$$\log^\alpha(n)/\epsilon^{1-\beta} = \log^\alpha(n)(3n)^{1-\beta},$$

i.e., at most  $o(n)$ . From the exact symmetric difference, one can compute the set intersection size as  $(|a| + |b| - |a\Delta b|)/2$  with high probability. This is a contradiction.  $\square$

**3.7. Algorithm for unknown parameters.** If  $M$  is not known in advance, we won't know in advance how many random variables to construct (or, equivalently, how many bits are needed for each seed). Note that, because of the recursive construction of  $H$  and  $p$ , this is not a problem. If at any point we encounter a tuple  $(i_0, c, \theta)$  with  $c$  bigger than the maximum  $C$  encountered before, we simply do the following. For each  $k, \ell$ , we pick  $(\log c - \log C)$  new random bits, which we associate with the master seed  $S_{k,\ell}$ . Use the new random bits to extend randomly (when needed) each of the  $n$  seeds  $s_{i,k,\ell}$  to length  $\lceil \log c \rceil + 1$ . We also virtually form larger matrices  $\hat{H}$  and pattern  $p$  without actually instantiating them.

It is also possible to run the algorithm with a constant factor space penalty if  $n$  is not known in advance. Initialize  $n$  to 2. Pick a field size appropriate for this  $n$  and for the known (or so far encountered) value of  $M$ . Let  $F_1$  be this field and  $f_1$  be its size. Start reading the stream, performing calculation in  $F_1$ . At an arbitrary point in the stream, suppose we are using a field  $F_c$  of size  $f_c$ .

If we now read a tuple  $(i, c_i, \theta)$  where  $i$  is too big to handle in  $F_c$ , we compute the smallest  $q$  such that a degree  $q$  extension of  $F_c$  is sufficient to handle  $i$ , and for each  $k, \ell$ , we prepare a new master seed in this extension which we denote by  $F_{c+1}$ . Note that  $f_{c+1} = f_c^q$ . In other words, for each  $k, \ell$ , we generate at random the coefficients of a degree-3 polynomial in  $F_{c+1}$ . The new master seeds and field are to be used for all types which could not be handled in  $F_c$  but can be handled in  $F_{c+1}$ .

By keeping track of all field sizes we use and all master seeds for each of these field sizes, when we encounter a new type we can easily figure out the field size needed to handle this type using the correct seeds. The union of all seeds is still 4-wise independent. At the end, we will have a final value of  $n$  (the maximum type), and, along the way, in the worst case, we will have constructed master seeds for families of size  $\nu, \nu^{1/2}, \nu^{1/4}, \nu^{1/8}, \dots$ , where  $\nu^{1/2} < n \leq \nu$ . For each  $k, \ell$ , storing these master seeds and successive fields requires storage space  $\log(\nu), \frac{1}{2} \log(\nu), \frac{1}{4} \log(\nu) \dots$ , and, thus, the storage for the master seeds is  $O(\max(\log M, \log \nu)) = O(\max(\log M, \log n))$ . The total space for all the master seeds and fields is  $O((\max(\log M, \log n)) \log(1/\delta)/\epsilon^2)$ . The space required for storing the counters remains  $O(\log(Mn) \log(1/\delta)/\epsilon^2)$ , and so the overall space is  $O(\log(Mn) \log(1/\delta)/\epsilon^2)$ . The average processing time per item increases by  $o(1) \cdot \text{field}(\log(Mn))$ , but preparing the last new collection of master seeds takes time  $\log(\nu) \log(1/\delta)/\epsilon^2$ , and this time represents an (acceptable) additive increase in the maximum per-item time. Note that the amortized per-item time is asymptotically the same as for the case when  $n$  is known in advance.

## 4. Related work.

**4.1. Relationship with sketch algorithms.** In [BCFM00], the authors consider the problem of detecting near-duplicate web pages. For their purpose and ours, a web page is a subset of a large universe, and two web pages  $A$  and  $B$  are near-duplicates if  $r(A, B) = \frac{|A \cap B|}{|A \cup B|}$  is large. They present an algorithm that computes a small fixed-size “sketch” of each web page such that, with high probability,  $r(A, B)$  can be approximated to within additive error given the two sketches. A central technique is based on the observation that, under a random injection  $h$  of the universe into the integers, the probability that the minimal element of  $h(A \cup B)$  is in  $h(A \cap B)$  is exactly  $r(A, B)$ . (In practice, the injections come from a small sample space; for the purpose of our comparison, we can consider truly random injections.) Some of the relevant techniques in [BCFM00] were used earlier in [C97, BGMZ97].

Our results on computing the  $L^1$ -difference between two functions can be viewed



TABLE 1  
Relative-error approximability via sketches.

	$ A ,  B $	$ A \cap B $	$ A \cup B $	$ A \Delta B $
Here	trivial	iff large	yes	yes
[BCFM00]	trivial	iff large	yes	iff large

as a sketch algorithm. The sketch function  $h$  takes as input the graph of a single function and performs the algorithm of section 3, getting a set  $\{Z_{k,\ell}\}$  of random variables. (Note that the same master seeds must be used for all sketches.) To reconstruct the  $L^1$ -difference from two sketches  $\{Z_{k,\ell}\}, \{Z'_{k,\ell}\}$ , compute  $\rho(\{Z_{k,\ell}\}, \{Z'_{k,\ell}\}) = \text{median}_k \text{avg}_\ell (Z_{k,\ell} - Z'_{k,\ell})^2$ .

THEOREM 14. *The  $L^1$ -difference of two functions from  $\{0, \dots, n-1\}$  to  $\{0, \dots, M-1\}$  is in*

$$\text{PAS}(\log(Mn) \log(1/\delta)/\epsilon^2).$$

In particular, the  $L^1$ -difference (or  $L^2$ -difference) of two characteristic functions  $\chi_A$  and  $\chi_B$  is the size of the symmetric difference  $|A \Delta B|$ ; we've shown how to approximate it to within small relative error with high probability. The size of the sketch is  $O(\log(Mn) \log(1/\delta)/\epsilon^2)$ , the space bound of the streaming algorithm. Finally, note that computation of these sketches can be performed in the streaming model, which is sometimes an advantage both theoretically and in practice.

COROLLARY 15. *The symmetric difference between two sets from a universe of size  $n$  is in*

$$\text{PAS}(\log(n) \log(1/\delta)/\epsilon^2).$$

One can now ask which cells of the  $A$ - $B$  Venn diagram can be approximated as functions of  $(A, B)$  in the sketch model using our techniques and using the techniques of [BCFM00]. First note that  $|A|$ ,  $|B|$ , and  $|A| + |B|$  are trivial in the sketch model. Next, an additive approximation of  $r = r(A, B)$  yields an approximation of  $(1+r)$  and, thus, of  $1/(1+r)$  with small relative error; thus,  $|A \cup B| = (|A| + |B|)/(1+r)$  can be approximated with small relative error using the techniques of [BCFM00] or with small relative error as  $(|A| + |B| + |A \Delta B|)/2$  using our techniques. In general, one cannot approximate  $|A \cap B|$  with small relative error, even using randomness [KN97], but, if  $|A \cap B|$  is sufficiently large compared with  $|A \cup B|$ , the intersection can be approximated as  $|A \cap B| = (|A| + |B|)r/(1+r)$  by [BCFM00] and as  $|A \cap B| = (|A| + |B| - |A \Delta B|)/2$  by our methods. Finally, the techniques of [BCFM00] only approximate  $1-r$  additively, and, if  $1-r$  is smaller than the error  $\epsilon$  of approximation (i.e., if  $|A \Delta B| < \epsilon|A \cup B|$ ), then the techniques of [BCFM00], which approximate  $|A \Delta B|$  as  $|A \Delta B| = (|A| + |B|)^{\frac{1-r}{1+r}}$ , do not perform well,<sup>6</sup> but our technique approximates  $|A \Delta B|$  with small relative error regardless of the size of  $|A \Delta B|$ .

This information is summarized in Table 1. Other cells in the Venn diagram reduce to these results, e.g., by complementing  $A$  or  $B$ . (Note that  $A$  and  $A$ -complement may have different sizes.)

<sup>6</sup>The results of [BCFM00] are the best possible in their original context, which is somewhat different from our context.

**4.2. Approximating sizes of supports and the zeroth frequency moment.** In this section, we briefly consider three variants.

Let

$$\begin{aligned} F_0^{\neq} &= |\{i : a_i \neq b_i\}|, \\ F_0^{\neq 0} &= |\{i : (a_i = 0 \wedge b_i \neq 0) \vee (a_i \neq 0 \wedge b_i = 0)\}|, \\ F_0^{20\%} &= |\{i : (a_i > 1.2b_i) \vee (b_i > 1.2a_i)\}|. \end{aligned}$$

Note that these are all generalizations of  $F_0 = |\{i : a_i \neq 0\}|$ , which was studied in [AMS99]. We will show that  $F_0^{\neq}$  and  $F_0^{\neq 0}$  can be approximated, but  $F_0^{20\%}$  cannot be approximated. We do this by using reductions under which

$$\text{PASST}(\log(M) \log(n) \log(1/\delta)/\epsilon^2)$$

is closed.

To approximate  $F_0^{\neq}$ , put

$$A_{i,x} = \begin{cases} 1 & a_i = x, \\ 0 & \text{otherwise} \end{cases}$$

and

$$B_{i,x} = \begin{cases} 1 & b_i = x, \\ 0 & \text{otherwise.} \end{cases}$$

Then  $\frac{1}{2} \sum_{i,x} |A_{i,x} - B_{i,x}| = F_0^{\neq}$ , where the sum is over  $0 \leq i < n$  and  $0 \leq x < M$ . We can approximate this  $L^1$ -difference.

To approximate  $F_0^{\neq 0}$ , put

$$A_i = \begin{cases} 1 & a_i > 0, \\ 0 & \text{otherwise} \end{cases}$$

and

$$B_i = \begin{cases} 1 & b_i > 0, \\ 0 & \text{otherwise.} \end{cases}$$

Then  $\sum_i |A_i - B_i| = F_0^{\neq 0}$ . This is used in section 4.1.

Finally, consider  $F_0^{20\%}$ . We reduce the set-disjointness problem (restricted to inputs with intersection size at most one) to that of approximating  $F_0^{20\%}$ .

Let  $(x, y)$  be an instance of set disjointness, and put

$$a_i = \begin{cases} 11 & x_i = 1, \\ 13 & \text{otherwise} \end{cases}$$

and

$$b_i = \begin{cases} 15 & y_i = 1, \\ 13 & \text{otherwise.} \end{cases}$$

Then  $a_i$  and  $b_i$  differ by at least 20% exactly in the 11-15 case, i.e., exactly when  $x_i = y_i = 1$ .

If we could output a number  $X$  such that  $|X - F_0^{20\%}| \leq \epsilon F_0^{20\%}$  for  $\epsilon < 1$  with probability  $1 - \delta$ , then we would be able to distinguish the situation  $F_0^{20\%} = 0$  from  $F_0^{20\%} = 1$  and in turn distinguish  $|x_i \cap y_i| = 0$  from  $|x_i \cap y_i| = 1$ , a contradiction.

Note that we cannot even output a random variable  $X$  satisfying the following apparently (but not actually) weaker condition:

$$F_0^{21\%}(1 - \epsilon) \leq X \leq F_0^{20\%}(1 + \epsilon),$$

with  $F_0^{21\%} = |\{i : (a_i > 1.21b_i) \vee (b_i > 1.21a_i)\}|$ , because, in fact,

$$\{i : (a_i > 1.21b_i) \vee (b_i > 1.21a_i)\} = \{i : (a_i > 1.20b_i) \vee (b_i > 1.20a_i)\},$$

so that  $F_0^{21\%} = F_0^{20\%}$ .

In summary, putting  $F_0^\tau = |\{i : (a_i > (1 + \tau)b_i) \vee (b_i > (1 + \tau)a_i)\}|$ , we obtain the following.

**THEOREM 16.** *We have*

1.  $F_0^\tau \in \text{PASST}(\log(Mn) \log(1/\delta)/\epsilon^2, \text{field}(\log(Mn)) \log(1/\delta)/\epsilon^2)$ ;
2.  $F_0^{\tau \neq 0} \in \text{PASST}(\log(Mn) \log(1/\delta)/\epsilon^2, \text{field}(\log(Mn)) \log(1/\delta)/\epsilon^2)$ ;
3. *for all  $\tau \in (0, 1)$ , all fixed  $\epsilon < 1$ ,  $\delta < 1/4$ , and  $M > 1/\tau + 2$ , and, for any*

$f = o(n)$ ,  $F_0^\tau \notin \text{PASST}(f(n))$ .

*Also,*

1.  $F_0^\tau \in \text{PAS}(\log(Mn) \log(1/\delta)/\epsilon^2, \text{field}(\log(Mn)) \log(1/\delta)/\epsilon^2)$ ;
2.  $F_0^{\tau \neq 0} \in \text{PAS}(\log(Mn) \log(1/\delta)/\epsilon^2, \text{field}(\log(Mn)) \log(1/\delta)/\epsilon^2)$ ;
3. *for all  $\tau \in (0, 1)$ , all fixed  $\epsilon < 1$ ,  $\delta < 1/4$ , and  $M > 1/\tau + 2$ , and, for any*

$f = o(n)$ ,  $F_0^\tau \notin \text{PAS}(f(n))$ .

**4.3. Approximating the  $L^2$ -difference and the second frequency moment.** In [AMS99], the authors consider the following problem. The input is a sequence of elements from  $[n] = \{0, \dots, n - 1\}$ . An element  $i \in [n]$  may occur many times. We let  $a_i$  denote the number of times  $i$  occurs. As above, assume that, for all  $i$ ,  $|a_i| \leq M$ .

The  $k$ th frequency moment  $F_k$  of the sequence is defined to be  $\sum a_i^k$ . Note that the first frequency moment  $F_1 = \sum a_i$  is just the length of the stream and is therefore trivial to compute, but other frequency moments are nontrivial. Alon, Matias, and Szegedy [AMS99] give a variety of upper and lower bounds for frequency moments. In particular, for  $F_2 = \sum a_i^2$ , they show the following.

**THEOREM 17** (see [AMS99]).

$$F_2 \in \text{PASST}((\log(Mn)) \log(1/\delta)/\epsilon^2, \text{field}(\log(Mn)) \log(1/\delta)/\epsilon^2).$$

We sketch the algorithm, without proof, in order to illustrate the previous work that is our point of departure. A full treatment of the correctness and work space of the algorithm of Theorem 17 may be found in [AMS99].

*Proof* (sketch). For each  $k$ ,  $1 \leq k \leq \Theta(\log(1/\delta))$ , and for each  $\ell$ ,  $1 \leq \ell \leq \Theta(1/\epsilon^2)$ , let  $\{v_{k\ell}[i]\}_i$  be a set of 4-wise independent  $\pm 1$ -valued random variables. Output

$$\text{median}_{k,\text{avg}_\ell} \left( \sum a_i v_{k\ell}[i] \right)^2. \quad \square$$

Consider now a generalization of the input allowing signed examples, which were also considered by Alon et al. in [AGMS99]. That is, each item in the sequence consists of a type  $i \in [n]$  and a sign  $\pm$ , and there may be many items of type  $i$  of each

sign. Denote by  $a_i$  the number of positive occurrences of  $i$  and by  $b_i$  the number of negative occurrences of  $i$ , and let  $L_k$  denote  $\sum |a_i - b_i|^k$ .

The following corollary was obtained independently by Alon et al. [AGMS99].

COROLLARY 18.  $L_2 \in \text{PASST}(\log(Mn) \log(1/\delta)/\epsilon^2, \text{field}(\log(Mn)) \log(1/\delta)/\epsilon^2)$ .

*Proof* (sketch). With  $k, \ell$ , and  $v_{k\ell}[i]$  as above, output

$$\text{median}_{k \text{ avg}_\ell} \left( \sum (a_i - b_i) v_{k\ell}[i] \right)^2. \quad \square$$

The algorithm of Corollary 18 can be used to approximate the  $L^2$ -difference between the two functions  $a$  and  $b$ .

Note that for signed input examples, computing the frequency moment  $L_1$  is nontrivial, modulo the special case of when  $a_i \geq b_i$  for all  $i$ .

For any  $p$ , the problem of computing the  $p$ th frequency moment and that of computing the corresponding  $L^p$ -difference of functions differs only in the representation of the input stream. Given an  $L^p$ -instance stream  $\langle (i, c_i, \theta_i) \rangle$ , one can expand each item  $(i, c_i, \theta_i)$  into  $c_i$  occurrences of  $(\theta, i)$  to get a frequency moment instance. Therefore a frequency moment algorithm for signed examples can be used to compute the  $L^p$ -difference of functions, but note that, in general, one pays a high cost in processing time, even just to read the input—the input has been expanded exponentially. The algorithm of Corollary 18 avoids this cost, because it is efficient in both input representations. Thus, the  $L^2$ -difference is in

$$\text{PASST}((\log(Mn)) \log(1/\delta)/\epsilon^2, \text{field}(\log(Mn)) \log(1/\delta)/\epsilon^2).$$

**4.4. Earlier work on probabilistic counting.** In [FM83], the authors give a small-space randomized algorithm that approximates the number of distinct elements in a stream. Their algorithm assumed the existence of certain ideal hash functions. Later, [AMS99] improved this result by substituting a practically available family of hash functions. [AMS99] also gives a variety of other results on approximating the frequency moments. Many results of this kind, some old and some new, are described in [GM99].

**Acknowledgments.** We thank S. Muthukrishnan for helpful discussions. We also thank an anonymous referee for a very careful reading of the paper.

#### REFERENCES

- [AMS99] N. ALON, Y. MATIAS, AND M. SZEGEDY, *The space complexity of approximating the frequency moments*, J. Comput. System Sci., 58 (1999), pp. 137–147.
- [AGMS99] N. ALON, P. GIBBONS, Y. MATIAS, AND M. SZEGEDY, *Tracking join and self-join sizes in limited storage*, in Proceedings of the 18th Symposium on Principles of Database Systems, ACM, New York, 1999, pp. 10–20.
- [AS92] N. ALON AND J. SPENCER, *The Probabilistic Method*, John Wiley, New York, 1992.
- [BCFM00] A. BRODER, M. CHARIKAR, A. FRIEZE, AND M. MITZENMACHER, *Min-wise independent permutations*, J. Comput. System Sci., 60 (2000) pp. 630–659.
- [BGMZ97] A. BRODER, S. GLASSMAN, M. MANASSE, AND G. ZWEIG, *Syntactic clustering of the Web*, in Proceedings of the Sixth International World Wide Web Conference, Santa Clara, CA, World Wide Web Consortium, Cambridge, MA, 1997, pp. 391–404. Also available online from <http://www.scope.gmd.de/info/www6/technical/paper205/paper205.html>.
- [CN98] CISCO NETFLOW, 1998. Available online from <http://www.cisco.com/warp/public/732/netflow/>.
- [C97] E. COHEN, *Size-estimation framework with applications to transitive closure and reachability*, J. Comput. System Sci., 55 (1997), pp. 441–453.

- [FM83] P. FLAJOLET AND G. N. MARTIN, *Probabilistic counting*, in Proceedings of the 24th Foundations of Computer Science Conference, IEEE Computer Society, Los Alamitos, 1983, pp. 76–82.
- [GM99] P. GIBBONS AND Y. MATIAS, *Synopsis Data Structures for Massive Data Sets*, in Eternal Memory Algorithms, DIMACS Ser. Discrete Math. Theoret. Comput. Sci. 50, AMS, Providence, RI, 1999, pp. 39–70.
- [HRR98] M. R. HENZINGER, P. RAGHAVAN, AND S. RAJAGOPALAN, *Computing on Data Streams*, Technical Report 1998-011, Digital Equipment Corporation Systems Research Center, Palo Alto, CA, 1998.
- [KN97] E. KUSHILEVITZ AND N. NISAN, *Communication Complexity*, Cambridge University Press, Cambridge, UK, 1997.
- [MS77] F. J. MACWILLIAMS AND N. J. A. SLOANE, *The Theory of Error-Correcting Codes*, North-Holland Math. Library 16, North Holland, New York, 1977.
- [RS99] E. RAINS AND N. J. A. SLOANE, *Private communication*.