An Approximate L^1-Difference Algorithm for Massive Data Streams*

Joan Feigenbaum
AT&T Labs – Research
180 Park Avenue
Florham Park, NJ 07932 USA
jf@research.att.com

Sampath Kannan†
AT&T Labs – Research
180 Park Avenue
Florham Park, NJ 07932 USA
skannan@research.att.com

Martin Strauss
AT&T Labs – Research
180 Park Avenue
Florham Park, NJ 07932 USA
mstrauss@research.att.com

Mahesh Viswanathan‡
Computer and Information Sciences
University of Pennsylvania
Philadelphia, PA 19104 USA
maheshv@saul.cis.upenn.edu

September 14, 1999

Abstract

Massive data sets are increasingly important in a wide range of applications, including observational sciences, product marketing, and 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 $\{V_j\}$ 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 *polylog*(c), for all seeds s. Our L^1-difference algorithm can be viewed as a "sketching" algorithm, in the sense of [Broder, Charikar, Frieze, and Mitzenmacher, STOC '98, pp. 327-336], 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.

*This paper is an expanded version of the authors' 1999 IEEE Symp. on Foundations of Computer Science extended abstract with the same title

†On leave from the Univ. of Pennsylvania. Part of this work was done at the Univ. of Pennsylvania, supported by grants NSF CCR96-19910 and ARO DAAH04-95-1-0092.

‡Supported by grant ONR N00014-97-1-0505, MURI.

1 Introduction

Massive data sets are increasingly important in a wide range of applications, including observational sciences, product marketing, and 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. 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*.* 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,† 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 workspace, 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.

*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."

†A WorldNet gateway router now generates more that 10Gb of NetFlow data each day.

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

> Consider two data streams of length at most n, each representing the non-zero points on the graph of an integer-valued function on a domain of size n. Assume that the maximum value of either 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 space $O(\log(M)\log(n)\log(1/\delta)/\epsilon^2)$ and time $O(\log(n)\log\log(n) + \log(M)\log(1/\delta)/\epsilon^2)$ to process each item. The input streams may be interleaved in an arbitrary (adversarial) order.

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 ± 1-valued random variables is called *range-summable* if $\sum_{j=0}^{c-1} V_j(s)$ can be computed in time $polylog(c)$, for all seeds s. We construct range-summable families of random variables that are n^2*-bad 4-wise independent.*‡

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], but the efficiency of the range summation seems to be significantly worse than 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.* [BCFM98] on sketching and that of Alon *et al.* [AMS96, 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 [BCFM98].

2.1 The Streaming Model

As in [HRR98], a *data stream* is a sequence of data items $\sigma_1, \sigma_2, \ldots, \sigma_n$ such that, on each *pass* through the stream, the items are read once in increasing order of their indices. We assume the items σ_i come from a set of size M, so that each σ_i has size $\log M$. In our computational model, we assume that the input is one or more data streams. We focus on two resources—the *workspace* required in words and the *time to process* an item in the stream. An algorithm will typically also require pre- and post-processing time, but usually applications can afford more time for these tasks.

Definition 1 The complexity class PASST$(s(\delta,\epsilon,n,M), t(\delta,\epsilon,n,M))$ (to be read as "probably approximately correct streaming space complexity $s(\delta,\epsilon,n,M)$ and time complexity $t(\delta,\epsilon,n,M)$") contains those functions f for which one can output a random variable X such that $|X-f| < \epsilon f$ with probability at least $1-\delta$, and computation of X can be done by making a single pass over the data, using

‡The property of n^2-bad 4-wise independence is defined precisely in Section 3 below.

workspace at most $s(\delta, \epsilon, n, M)$ and taking time at most $t(\delta, \epsilon, n, M)$ to process each of the n items, each of which is in the range 0 to $M-1$.

If $s = t$, we also write PASST(s) for PASST(s, t). ∎

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

2.2 The Sketch Model

Sketches were used in [BCFM98] to check whether two documents are nearly duplicates. A sketch can also be regarded as a *synopsis data structure* [GM98].

Definition 2 The complexity class PAS$(s(\delta, \epsilon, n, M)))$ (to be read as "probably approximately correct sketch complexity $s(\delta, \epsilon, n, M)$") contains those functions $f : X \times X \to Z$ of two inputs for which there exists a set S of size 2^s, a randomized *sketch function* $h : X \to S$, and a randomized *reconstruction function* $\rho : S \times S \to Z$ such that, for all $x_1, x_2 \in X$, 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, 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 $f : X \times Y \to Z$ 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:

Theorem 3 *If* $f \in$ PAS$(s(\delta, \epsilon, n, M))$ *via sketch function* $h \in$ PASST$(s(\delta, \epsilon, n, M), t(\delta, \epsilon, n, M))$, *then* $f \in$ PASST$(2s(\delta, \epsilon, n/2, M), t(\delta, \epsilon, n/2, M))$.

2.3 Arithmetic and Bit Complexity

Often one will run a streaming algorithm on a stream of n items of size $\log M$ on a computer with word size at least $\max(\log M, \log n)$. We assume that the following operations can be performed in constant time on words:

- Copy x into y
- Shift the bits of x one place to the left or one place to the right.

[§] This is not always possible, *e.g.*, not if $f(x, y)$ is the x'th bit of y.

- Perform the bitwise AND, OR, or XOR of x and y.
- Add x and y or subtract x from y.
- Assign to z the number of 1's among the bits of x.

We call such a model an *arithmetic* model and give complexity bounds in it. These operations all take at most linear time in a bit model; thus a machine that performs such operations bit by bit will run more slowly by a factor of $\max(\log M, \log n)$. Multiplication over a finite field may take more than $\log n$ time in a bit model; we use this operation but do *not* assume that it can be performed in constant time.

3 The L^1 Difference of Functions

3.1 An Approach to Solving the Problem

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 $(i, b_i, -1)$. 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. It is important that tuples of the form $(i, 0, \pm 1)$ not contribute to the size of the input. 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 to make 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 ± 1-valued random variables $R_{i,j}, j = 0, 1, \ldots, (M-1)$ 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 desired properties for 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 ± 1-valued random variables with seed s (chosen at random from some set of seeds, S) 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.*,

$$\forall \text{ distinct } j_1, j_2, j_3, \forall a, b, c \in \{+1, -1\} \Pr_s[V_{j_1}(s) = a | V_{j_2}(s) = b \wedge V_{j_3}(s) = c] = \Pr_s[V_{j_1}(s) = a]$$

.

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

3. For all $a < b$,

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

∎

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.* [AMS96]. 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 $V_j(s)$, $0 \le j < M$, such that, for all $c \le M$, one can compute $\sum_{j=0}^{c-1} V_j(s)$ quickly.

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 $(\log(M) - 1)^{\text{st}}$ 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 $\hat{H}$ 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 Z_2 of s with the j'th column of $\hat{H}$. Let i_k denote the k'th bit of the binary expansion of i, starting from zero. Define $f(i)$ by

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

¶Here and henceforth we will actually assume that M is a power of 4 to simplify the exposition.

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

$$0111\,1000\,1000\,1000\;\; 1000\,0111\,0111\,0111\;\; 1000\,0111\,0111\,0111\;\; 1000\,0111\,0111\,0111\ldots,$$

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 π. Finally, put $V_j = (-1)^{(s\cdot\hat{H}_j)+f(j)}$.

Proposition 5 *The quantity $\sum_{j=0}^{c-1} V_j(s)$ can be computed in time polylogarithmic in 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 $\begin{pmatrix} \hat{H}_{\log c} \\ 0 \end{pmatrix}$, 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

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

Also, note that the first M bits of p have the form $p_{\log M} = p_{\log M-2}\overline{p_{\log M-2}p_{\log M-2}p_{\log M-2}}$. 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 $s \cdot \begin{bmatrix} 1\cdots 1 \\ H_{(\log M-2)} \\ b_1\cdots b_1 \\ b_2\cdots b_2 \end{bmatrix}$. Then $f_{b_1b_2} = f'$ or $f_{b_1b_2} = 1 - f'$, 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 $f_{b_1b_2}$'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 $\sum_{j=q4^r}^{(q+1)4^r-1} V_j$.

Finally, if c is not a power of 4, write the interval $\{0, \ldots, (c-1)\} = [0, c)$ 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 $\log(c)$ time, not $\log^2(c)$ time, in an arithmetic model. 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 $\sum_{j=0}^{15} V_j$, then calls 2 deep for $\sum_{j=16}^{19} V_j$, then 1 deep for each of V_{20} and V_{21}. Better would be to compute V_{20} directly, use this value to compute V_{21} and $\sum_{j=16}^{19} V_j$ (note that V_{16} is easy to compute from V_{20}), and finally use $\sum_{j=16}^{19} V_j$ to compute $\sum_{j=0}^{15} V_j$.

Altogether, this requires time $O(\log(c))$ in an arithmetic model and, in any case, $\log^{O(1)}(c)$ time in a bit-complexity model. ∎

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

Proposition 6 *For all $a < b$, we have*

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

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, since, 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.

Secondly, 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 because

$$\begin{aligned} V_{j_1}(s)V_{j_2}(s)V_{j_3}(s)V_{j_4}(s) &= (-1)^{(s\cdot\hat{H}_{j_1})+f(j_1)} \cdot (-1)^{(s\cdot\hat{H}_{j_2})+f(j_2)} \cdot (-1)^{(s\cdot\hat{H}_{j_3})+f(j_3)} \cdot (-1)^{(s\cdot\hat{H}_{j_4})+f(j_4)} \\ &= (-1)^{f(j_1)+f(j_2)+f(j_3)+f(j_4)}. \qquad (2) \end{aligned}$$

Line (2) 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

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

for some constant K. From Theorem 9 (which we prove next), we can see that $K \le 4$ and so the proposition holds. ∎

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 $U(4a, 4b) \le 16U(a, b)$.

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*. Of the 64 possibilities making the columns dependent, 24 (= 4!) are odd and 40 are even. Note that an odd tuple has the additional characteristic that there are an odd number of i's for which $(j_i)_0 \vee (j_i)_1 = 1$. Given an odd tuple (j_1, j_2, j_3, j_4) pair it with (j'_1, j'_2, j'_3, j'_4) such that j'_i and j_i agree on all but the least two significant bits, and (j'_1, j'_2, j'_3, j'_4) has an even configuration of the two least significant bits. If (j_1, j_2, j_3, j_4) is a tuple having one of the 16 other 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 $U(4a, 4b) \le 16U(a, b)$. ∎

We now give a simple argument that $U(a, b) \le 78(b - a)^2$.

Theorem 8 $U(a, b) \le 78(b - a)^2$.

Proof. Given a and b, find r with $a, b \le 4^r$. Let α be the smallest multiple of 4 that is at least a and let β 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 at most 24 ways to assign roles to j_1, j_2, j_3, and j_4. There are at most 6 ways to choose $j_1 \in [a, \alpha) \cup [\beta, b)$. There are at most $(b-a)^2/2$ 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. This gives $72(b-a)^2$ tuples altogether. Thus we conclude that

$$\begin{aligned}
U(a,b) &\leq U(4\lfloor a/4\rfloor, 4\lfloor b/4\rfloor) + 72(b-a)^2 \\
&= 16U(\lfloor a/4\rfloor, \lfloor b/4\rfloor) + 72(b-a)^2 \\
&\leq 16U(4\lfloor\lfloor a/4\rfloor/4\rfloor, 4\lfloor\lfloor b/4\rfloor/4\rfloor) + 72(\lfloor b/4\rfloor - \lfloor a/4\rfloor)^2 + 72(b-a)^2 \\
&= 16^2U(\lfloor\lfloor a/4\rfloor/4\rfloor, \lfloor\lfloor b/4\rfloor/4\rfloor) + 72(\lfloor b/4\rfloor - \lfloor a/4\rfloor)^2 + 72(b-a)^2 \\
&\vdots \\
&\leq 16^{\lfloor \log_4(b-a)\rfloor} + 72(b-a)^2\left[1 + 1/16 + 1/16^2 \cdots\right] \\
&\leq (b-a)^2 + 72(b-a)^2(16/15) \\
&\leq 78(b-a)^2.
\end{aligned}$$

∎

We now give a more involved analysis that lets us improve the bound to $U(a,b) \leq 4(b-a)^2$.

Theorem 9 *$U(a,b) \leq 4(b-a)^2$.*

Proof. Define $A(a,b) \triangleq \frac{U(a,b)}{(b-a)^2}$. Notice that it is sufficient to show that $\sup_{a,b} A(a,b) \leq 4$. 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 $b - a \geq 16$. 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. (Since $b-a \geq 16 \geq 4$, it follows that $a \leq a' \leq b' \leq b$.) 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 $U(a',b') \leq 16U(a'/4, b'/4) = A(a'/4, b'/4)(b'-a')^2$, by induction. We now count the number of unpaired tuples having at least one column in $[a,a') \cup [b',b)$.

There are at most six tuples such that $j_1 = j_2 = j_3 = j_4$ in $[a,a') \cup [b',b)$. There are at most $36(b-a)$ tuples such that two of the columns are in $[a,a') \cup [b',b)$ and two of the columns are identical and equal to some other value. (A factor $\binom{4}{2} = 6$ is needed to assign the four columns to the 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 pick j_2 and j_3. Suppose that the least significant bit in which j_2 and j_3 differ is bit k. Let α be the least multiple of $2^{k'+1}$ that is at least a, and let β be the greatest multiple of $2^{k'+1}$ that is at most b, where

$$k' = \begin{cases} k-1; & k \text{ odd} \\ k+1; & k \text{ even.} \end{cases}$$

If $\alpha \leq j_2, j_3 < \beta$, then we can form j_2' and j_3' by toggling the k''th bit of j_2 and j_3 and pair (j_1, j_2, j_3, j_4) with (j_1, j_2', j_3', j_4). We then have

- $a \leq \alpha \leq j_2', j_3' < \beta \leq b$
- (j_1, j_2', j_3', j_4) is a dependent tuple

- (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 (1) corresponding to bits k and k' for each of $f(j_3')$ and $f(j_3)$ is odd 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 $f(j_3')$.

Thus we need only consider j_2 and j_3 such that j_2 is less than α or greater than β. If $\alpha - a \leq 2^{k'}$ then clearly there are at most $2^{k'}$ ways to choose $j_2 \in [a, \alpha)$. Otherwise, if $\alpha - a > 2^{k'}$, then the set $[a, \alpha - 2^{k'}) \cup [a + 2^{k'}, \alpha)$ is closed under toggling the k''th bit; so, if j_2 and j_3 are both in $[a, \alpha - 2^{k'}) \cup [a + 2^{k'}, \alpha) \cup [\alpha, \beta)$, then we can pair j_2 and j_3 (and similarly at the b end). The set of remaining possibilities for j_2 and j_3 is $[\alpha - 2^{k'}, a + 2^{k'}) \subseteq [\alpha - 2^{k'}, \alpha)$, which has size at most $2^{k'}$. Thus, whether or not $a \leq \alpha - 2^{k'}$ and whether or not $b \geq \beta + 2^{k'}$, there are at most $2^{k'}$ possibilities for j_2 in $[\alpha - 2^{k'}, a + 2^{k'})$ and another $2^{k'}$ possibilities in $[b - 2^{k'}, \beta + 2^{k'})$, so we get $2^{k'+1} \leq 2^{k+2}$ possibilities for j_2 in total. The $k+1$ least significant bits of j_3 are determined (the k least significant are the same as in j_2 and bit k is opposite); so there are at most $\left\lceil (b-a)/2^{k+1} \right\rceil \leq \frac{17}{16}(b-a)/2^{k+1}$ ways to choose j_3. (Note that, because $b - a \geq 16$, we have $(b - a + 1) \leq \frac{17}{16}(b-a)$.) Thus there are at most $(17/8)(b-a)$ ways to pick j_2 and j_3. After that, j_4 is determined. Altogether, for this k, there are at most $306(b-a) = 24 \cdot 6 \cdot (17/8)(b-a)$ ways to pick a dependent unpaired tuple with all columns different, and at most $36(b-a) + 6 \leq 37(b-a)$ ways to pick a tuple with a repeated column, such that all columns are in $[a, b)$ and some column is in $[a, a') \cup [b', b)$, for a total of $343(b-a)$.

We need to sum over all k such that $2^k \leq (b-a)$. Thus, we get that

$$\begin{aligned}
U(a,b) &\leq U(a', b') + 343(b-a)\log_2(b-a) \\
&\leq 16U(a'/4, b'/4) + 343(b-a)\log_2(b-a) \\
&= 16A(a'/4, b'/4)(b'/4 - a'/4)^2 + 343(b-a)\log_2(b-a) \\
&= A(a'/4, b'/4)(b'-a')^2 + 343(b-a)\log_2(b-a) \\
&\leq A(a'/4, b'/4)(b-a)^2 + 343(b-a)\log_2(b-a),
\end{aligned}$$

and so

$$A(a,b) \leq A(\lceil a/4 \rceil, \lfloor b/4 \rfloor) + 343\frac{\log_2(b-a)}{(b-a)}.$$

Let

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

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

$$D_i \leq \begin{cases} D_{i-1} + 343 \cdot \frac{2i}{4^i} & i \geq C \\ M_C & i < C, \end{cases} \tag{3}$$

where $M_C = \max(D_0, \ldots, D_{C-1})$ 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) has a solution

$$\begin{aligned}
D_i &= M_C + 343\sum_{j=C}^{i} \frac{2j}{4^{-j}} \\
&\leq M_C + 686\frac{C + 1/3}{3 \cdot 4^{C-1}},
\end{aligned}$$

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

$$D_i \le M_6 + 1.42,$$

whence, for all a, b, $A(a, b) \le M_6 + 1.42$.

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* $M_C = \max_{b-a\le 4^C=2^{2C}} A(a,b)$ *is at most*

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

Thus, we can take M_C *to be* M'_C.

Proof. Suppose (a, b) is a pair with $b - a \le 2^{2C}$ 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 $a, b \le 2^r$, 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 $a, b \le 3 \cdot 2^r$ then $U(a, b) = U(3 \cdot 2^r - b, 3 \cdot 2^r - a)$.

Finally:

- If $2^{2C-1} < a \le 2^{2C}$, then
  - If $2^{2C-1} < b \le 2^{2C}$, then put $(a', b') = (2^{2C} - b, 2^{2C} - a)$.
  - If $2^{2C} < b \le 3 \cdot 2^{2C-1}$, 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 $b < a + 2^{2C} \le 2^{2C+1}$. Put $(a', b') = (2^{2C+1} - b, 2^{2C+1} - a)$.
- If $2^{2C} < a$, then find $q > 2C$ with $2^q < a \le 2^{q+1}$.
  - If $b \le 2^{q+1}$, then $2^q < a < b \le 2^{q+1}$. Put $(a', b') = (2^{q+1} - b, 2^{q+1} - a)$.
  - Otherwise, $2^{q+1} < b \le a + 2^{2C} \le 2^{q+1} + 2^q = 3 \cdot 2^q$. Put $(a', b') = (3 \cdot 2^q - b, 3 \cdot 2^q - a)$.

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

Thus, if we are interested in M_6, *i.e.*, (a, b) with $b - a \le 4096$, we need only consider $a \le 2048$. A computer search was done for $A(a, b)$ over $a \le 2048$ and $b \le a + 4096$, and the maximum is 2.55334. Thus $A(a, b) \le 2.56 + 1.42 \le 4$. ∎

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 completely independent but this 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 ϵ and δ.

For each k, $1 \leq k \leq 4\log(1/\delta)$, and each ℓ, $1 \leq \ell \leq 72/\epsilon^2$, 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 ± 1-valued random variables.

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 where 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 amongst 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 [AMS96] 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* $1 - \delta$.

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

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

We shall now compute $E[Z_{k\ell}^2]$ and $E[Z_{k\ell}^4]$, for each k, ℓ. 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.

$$E[Z_{k,\ell}^2] \quad = \quad E\left[\left(\sum_i \sum_{j=a_i}^{b_i - 1} V_{i,j}\right)^2\right]$$

Figure 1: High level L^1 algorithm

Algorithm L1($\langle(i, c_i, \theta_i)\rangle$)

Initialize:
    For $k = 1$ to $4\log(1/\delta)$ do
        For $\ell = 1$ to $(8 \cdot A)/\epsilon^2$ do
            // any $A \geq A'$ will work for A' known to be between 7.5 and 9.
            { $Z_{k,\ell} = 0$
            pick a master seed $S_{k,\ell}$ from the (k, ℓ)'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, θ_i) in the input stream do
    For $k = 1$ to $4\log(1/\delta)$ 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 $\text{median}_k \text{avg}_\ell Z_{k,\ell}^2$.

$$\begin{aligned}
&= E\left[\left(\sum_{m=1}^{F_1} \pm V_m\right)^2\right] && (4)\\
&= \sum_{m=1}^{F_1} E[(\pm V_m)^2] + 2 \sum_{1 \leq m < m' < F_1} E[(\pm V_m)(\pm V_{m'})] \\
&= \sum_{m=1}^{F_1} 1 && (5)\\
&= F_1,
\end{aligned}$$

where, in line (4), we have relabeled the indices of V and in line (5) we used the pairwise independence of V_m and $V_{m'}$.

Next, consider

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

By 3-wise independence and the fact that $E[V^{\text{odd}}]=0$, the only terms with non-vanishing 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 latter 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 three-wise independence of the V's, and X and Y are independent by four-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

$$\begin{aligned} E[Z_{k,\ell}^4] &\le F_1 + 6F_1(F_1-1) + \sum_i E\left[\left(\sum_{j=a_i}^{b_i-1} V_{i,j}\right)^4\right] \\ &\le 6F_1^2 + \sum_i 4(b_i-a_i)^2 \qquad (6)\\ &\le 10F_1^2. \end{aligned}$$

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

Thus

$$\mathrm{Var}(Z_{k,\ell}^2) = E[Z_{k,\ell}^4] - E^2[Z_{k,\ell}^2] \le A\cdot F_1^2,$$

for $A=9$. Now, put $Y_k = \frac{\epsilon^2}{8\cdot A}\sum_{1\le\ell\le(8\cdot A)/\epsilon^2} Z_{k,\ell}^2$. Then $\mathrm{Var}(Y_k)\le\frac{\epsilon^2}{8}F_1^2$. By Chebyshev's inequality,

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

Put $W = \mathrm{median}_k Y_k$. Then $|W-F_1|>\epsilon F_1$ only if we have $|Y_k-F_1|>\epsilon F_1$ for half of the k's. By Chernoff's inequality, the probability of this is at most δ.

∎

3.5 Cost

Theorem 12 *There is an implementation of algorithm L1 (in Figure 1) that is in*

$$\mathrm{PASST}\left(\log(M)\log(n)\log(1/\delta)/\epsilon^2,\ \log(n)\log\log(n)+\log(M)\log(1/\delta)/\epsilon^2\right).$$

If the input tuples come in the order $(0, a_0, +1), (1, a_1, +1), \ldots, (0, b_0, -1), (1, b_1, -1) \ldots$*, (or, more generally, if the a tuples come in the same order as the b tuples), then another implementation of algorithm L1 runs in*

$$\mathrm{PASST}\left(\log(M)\log(n)\log(1/\delta)/\epsilon^2,\ \log(M)\log(1/\delta)/\epsilon^2\right).$$

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 (± 1)-valued random variables $V_{i,j,k,\ell}$

The space to store the counters is $O((\log(M) + \log(n))\log(1/\delta)/\epsilon^2)$. By our construction, for each k, ℓ, 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 for all the master seeds to have $O(\log(1/\delta)/\epsilon^2)$ independent parallel repetitions. This is the same is the same space requirement as for the counter storage.

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 i^2 and i^3. These computations can be performed in time $O(D\log D)$ (using, for example, an FFT-based algorithm). After this, $s_{i,k,\ell}$ can be computed in an additional $O(D\log D)$ time for an overall time of $O(D\log D)$. From $s_{i,k,\ell}$, we need to compute a range sum $\sum_{j=a}^{b-1} V_j(s_{i,k,\ell})$, which can be computed in $O(\log M)$. Thus the overall complexity is

$$O(D\log D + \log(M)\log(1/\delta)/\epsilon^2)$$

Finally, we consider the restricted order. Suppose we are guaranteed that the tuples occur in the order

$$(0, a_0, +1), (1, a_1, +1), \ldots, (0, b_0, -1), (1, b_1, -1) \ldots,$$

or just that the a tuples occur in the same order as the b tuples. In this case we can compute $a(i) = a_3 i^3 + a_2 i^2 + a_1 i + a_0$ from $a(i-1)$ in constant time by some simple massaging of the above expression to express it as a linear combination of powers of $i - 1$.

Altogether, the processing time is

$$O(\log(M)\log(1/\delta)/\epsilon^2)$$

per item in an arithmetic model. ∎

3.6 Optimality

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

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

A similar result holds in the streaming model.

Proof. We reduce the set disjointness problem to this 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 $L^1 \in \mathrm{PAS}(\log^\alpha(n)/\epsilon^{1-\beta})$. Put $\epsilon = 1/(2n)$. 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

$$\log^\alpha(n)/\epsilon^{1-\beta} \leq \log^\alpha(n)(2n)^{1-\beta},$$

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. ∎

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. 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, θ) with c bigger than the maximum C encountered before, we simply do the following. For each k, ℓ, we pick $(\log c - \log C)$ new master seeds $S_{k,\ell}$ of length $O(\log n)$. We use the master seeds to extend randomly 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 penalty if n is not known in advance. Initialize n to 2. Start reading the stream, performing the calculation for this n. As the stream is read, maintain $N_T = \max_{1 \leq t \leq T} i_t$, where T is the current position in the stream and i_t is the first component of the t'th item (i_t, c_{i_t}, θ_t).

If, at some point T, we read a tuple (i, c_i, θ) with $i > N_T$, then, for each k, ℓ, we prepare a new master seed for a family of i^2 4-wise independent seeds. The union of all seeds is still 4-wise independent. (In fact, for any collection $\mathcal{O}$ of old seeds, the distribution on any set of new seeds is fully independent of any setting of $\mathcal{O}$.) We store both the old and the new master seeds. At the end, we will have a final value of n, and, along the way, we will have constructed master seeds for families of size $n, n^{1/2}, n^{1/4}, n^{1/8}, \ldots$. These require storage space $\log(n), \frac{1}{2}\log(n), \frac{1}{4}\log(n) \ldots$, and, thus, the total storage is $O(\log(n))$, ignoring the factor $\log(M)\log(1/\delta)/\epsilon^2$ that is the same whether or not we know n in advance. The average processing time per item increases by $o(1)$, but the maximum processing time per item increases by $\log n$.

4 Related Work

4.1 Relationship with Sketch Algorithms

In [BCFM98], the authors consider the problem of detecting near-duplicate web pages. For their and our purposes, 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 minimum of $h(A \cup B)$ is in $h(A \cap B)$ is exactly $r(A, B)$. Some of the relevant techniques in [BCFM98] were used, earlier, in [C94, BGMZ97].

Our results on computing the L^1 difference between two functions can be viewed as a sketch algorithm. The sketch function h takes as input (the graph of) a single function and performs the algorithm of Section 3.1, getting a set $\{Z_{k,\ell}\}$ of random variables. 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 $\{0, \ldots, n-1\}$ to $\{0, \ldots, M-1\}$ is in*

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

∎

In particular, the L^1 difference (or L^2 difference) of two characteristic functions χ_A and χ_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 $O(\log(m)\log(n)\log(1/\delta)/\epsilon^2)$, 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 of two sets from a universe of size n is in*

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

∎

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 [BCFM98]. First note that $|A|$, $|B|$, and $|A| + |B|$ are trivial in the sketch model. Next, an additive approximation of r yields an approximation of $(1 + r)$ with small relative error; thus, $|A \cup B| = (|A| + |B|)/(1 + r)$ can be approximated with small relative error using the techniques of [BCFM98] 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 [BCFM98] and as $|A \cap B| = (|A| + |B| - |A\Delta B|)/2$ by our methods. Finally, the techniques of [BCFM98] only approximate $1 - r$ additively, and, if $1 - r$ is smaller than the error δ of approximation (*i.e.,* if $|A\Delta B| < \delta|A \cup B|$), then the techniques of [BCFM98], which approximate $|A\Delta B|$ as $|A\Delta B| = (|A| + |B|)\frac{1-r}{1+r}$, do not perform well,[‖] 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 (but note that A and A-complement may have different sizes).

4.2 Approximating Sizes of Supports and the Zeroth Frequency Moment

In this section, we briefly consider three variants.

Let

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

[‖] The results of [BCFM98] are the best possible in their original context, which is somewhat different than our context.

Table 1: Relative-error approximability via sketches

	$\|A\|$, $\|B\|$	$\|A \cap B\|$	$\|A \cup B\|$	$\|A \Delta B\|$
Here	trivial	iff large	yes	yes
[BCFM98]	trivial	iff large	yes	iff large

Note that these are all generalizations of $F_0 = |\{i : a_i \neq 0\}|$, which was studied in [AMS96]. 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 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 put

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

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 put

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

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 to that of approximating $F_0^{20\%}$.

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

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

and put

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

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 $F_0^{21\%} = |\{i : (a_i > 1.21b_i) \vee (b_i > 1.21a_i)\}|$, since 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 $F_0^{\tau} = |\{i : (a_i > \tau b_i) \vee (b_i > \tau a_i)\}|$,

Theorem 16 *We have*

1. $F_0^{\neq} \in \mathrm{PASST}(\log(M)\log(n)\log(1/\delta)/\epsilon^2)$.
2. $F_0^{\neq 0} \in \mathrm{PASST}(\log(M)\log(n)\log(1/\delta)/\epsilon^2)$.
3. *For all fixed* $\epsilon < 1$, $\delta < 1/4$, *and* $M > 1/\tau + 2$, *and for any* $f = o(n)$, *we have* $F_0^{\tau} \notin \mathrm{PASST}(f(n))$.

Also,

1. $F_0^{\neq} \in \mathrm{PAS}(\log(M)\log(n)\log(1/\delta)/\epsilon^2)$.
2. $F_0^{\neq 0} \in \mathrm{PAS}(\log(M)\log(n)\log(1/\delta)/\epsilon^2)$.
3. *For all fixed* $\epsilon < 1$, $\delta < 1/4$, *and* $M > 1/\tau + 2$, *and for any* $f = o(n)$, *we have* $F_0^{\tau} \notin \mathrm{PAS}(f(n))$.

■

4.3 Approximating the L^2 Difference and the Second Frequency Moment

In [AMS96], the authors consider the following problem. The input is a sequence of elements from $[n] = \{0, \ldots, n-1\}$. An element $i \in [n]$ may occur many times; we let a_i denote the number of times i occurs in the sequence. Now the length of the sequence is some m unrelated to n. (This notation attaches the same semantics to n, a_i, and b_i as Section 3 does, namely, n is the number of different types of items and our goal is to approximate $\sum_{i=0}^{n-1} |a_i - b_i|^p$. Syntactically, the parameters (n, m) here play the roles of (M, n) in Definition 1, since, in the input format of this section, there are m items, each of which is a number between 0 and $n-1$.)

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 m and is therefore trivial to compute, but other frequency moments are non-trivial. Alon, Matias, and Szegedy [AMS96] give a variety of upper and lower bounds for frequency moments. In particular, for $F_2 = \sum a_i^2$, they show

Theorem 17 ([AMS96])

$$F_2 \in \mathrm{PASST}((\log n + \log m)\log(1/\delta)/\epsilon^2,\ \log(n)\log\log(n) + \log(1/\delta)/\epsilon^2).$$

(Here $F_2 = \sum a_i^2$, where a_i is the number of occurrences of i in the input stream.)

In this result and the following corollary, the time cost of $\log n \log\log n$ can be reduced to 1 on a machine that can perform finite field multiplication on elements of size $\log n$ in constant time. It can also be reduced under suitable restrictions on the input order.

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 workspace of the algorithm of Theorem 17 may be found in [AMS96].

Proof. [sketch]

For each k, $1 \le k \le O(\log(1/\delta))$ and for each ℓ, $1 \le \ell \le O(1/\epsilon^2)$, let $\{v_{k\ell}[i]\}_i$ be a set of 4-wise independent random variables. Output $\mathrm{median}_k \mathrm{avg}_\ell \left(\sum a_i v_{k\ell}[i]\right)^2$. ∎

Consider now a generalization of the input allowing signed examples. 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 F_k denote $\sum |a_i - b_i|^k$.

We have obtained the following corollary. It was obtained independently by Alon *et al.* [AGMS99].

Corollary 18 $F_2 \in \mathrm{PASST}((\log n + \log m)\log(1/\delta)/\epsilon^2,\ \log(n)\log\log(n) + \log(1/\delta)/\epsilon^2)$*. (Here* $F_2 = \sum(a_i - b_i)^2$*, where* a_i *is the number of positive occurrences of* i *in the input and* b_i *is the number of negative occurrences of* i *in the input.)*

Proof. [sketch] With k, ℓ, and $v_{k\ell}[i]$ as above, output $\mathrm{median}_k \mathrm{avg}_\ell \left(\sum(a_i - b_i) v_{k\ell}[i]\right)^2$. ∎

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 F_1 is non-trivial. In fact, we don't know how to compute F_1 in the input representation of Corollary 18.

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, θ_i) into c_i occurrences of (θ, 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.

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 [AMS96] improved this result by substituting a practically available family of hash functions. [AMS96] 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 [GM98].

Acknowledgment

We thank S. Muthukrishnan for helpful discussions.

References

[AMS96] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *Proc. of 28'th Symp. on the Theory of Computing*, ACM Press, New York, pages 20–29, 1996. Expanded version to appear in *J. Comput. System Sci.*

[AGMS99] N. Alon, P. Gibbons, Y. Matias, and M. Szegedy. Tracking Join and Self-Join Sizes in Limited Storage. In *Proc. of the 18'th Symp. on Principles of Database Systems*, ACM Press, New York, pages 10–20, 1999.

[AS92] N. Alon and J. Spencer. *The Probabilistic Method.* John Wiley and Sons, New York, 1992.

[BCFM98] A. Broder, M. Charikar, A. Frieze, and M. Mitzenmacher. Min-wise independent permutations. In *Proc. of the 30'th Symp. on the Theory of Computing*, ACM Press, New York, pages 327–336, 1998.

[BGMZ97] A. Broder, S. Glassman, M. Manasse, and G. Zweig. Syntactic Clustering of the Web. In *Proc. Sixth Int'l. World Wide Web Conference*, World Wide Web Consortium, Cambridge, pages 391–404, 1997. Available as `http://decweb.ethz.ch/WWW6/Technical/Paper205/Paper205.html`.

[CN98] Cisco NetFlow, 1998. `http://www.cisco.com/warp/public/732/netflow/`.

[C94] E. Cohen. Size-estimation framework with applications to transitive closure and reachability. In *J. Comput. System Sci.*, 55:441–453, 1997. (Special issue of selected papers from 1994 *IEEE Symp. on Foundations of Computer Science.*)

[FM83] P. Flajolet and G. N. Martin. Probabilistic Counting. In *Proc. 24'th Foundations of Computer Science Conference*, IEEE Computer Society, Los Alamitos, pages 76–82, 1983.

[GM98] P. Gibbons and Y. Matias. Synopsis Data Structures for Massive Data Sets. To appear in *Proc. 1998 DIMACS Workshop on External Memory Algorithms.* DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, Providence. Abstract in *Proc. Tenth Symposium on Discrete Algorithms*, ACM Press, New York and Society for Industrial and Applied Mathematics, Philadelphia, pages S909–910, 1999.

[HRR98] M. Rauch Henzinger, P. Raghavan, and S. Rajagopalan. Computing on data streams. Technical Report 1998-011, Digital Equipment Corporation Systems Research Center, May 1998.

[KN97] E. Kushilevitz and N. Nisan. *Communication Complexity.* Cambridge University Press, 1997.

[MS77] F. J. MacWilliams and N. J. A. Sloane. *The Theory of Error-Correcting Codes.* North Holland Mathematical Library, Vol. 16, North Holland, New York, 1977.

[RS99] E. Rains and N. J. A. Sloane. Private communication. For details about second order Reed-Muller codes, see [MS77].