# A Near-Optimal Algorithm for L1-Difference

Jelani Nelson[†] David P. Woodruff[‡]

**Abstract**

We give the first $L_1$-sketching algorithm for integer vectors which produces nearly optimal sized sketches in nearly linear time. This answers the first open problem in the list of open problems from the 2006 IITK Workshop on Algorithms for Data Streams. Specifically, suppose Alice receives a vector $x \in \{-M, \ldots, M\}^n$ and Bob receives $y \in \{-M, \ldots, M\}^n$, and the two parties share randomness. Each party must output a short sketch of their vector such that a third party can later quickly recover a $(1 \pm \varepsilon)$-approximation to $||x - y||_1$ with 2/3 probability given only the sketches. We give a sketching algorithm which produces $O(\varepsilon^{-2} \log(1/\varepsilon) \log(nM))$-bit sketches in $O(n \log^2(nM))$ time, independent of $\varepsilon$. The previous best known sketching algorithm for $L_1$ is due to [Feigenbaum *et al.*, SICOMP 2002], which achieved the optimal sketch length of $O(\varepsilon^{-2} \log(nM))$ bits but had a running time of $O(n \log(nM)/\varepsilon^2)$. Notice that our running time is near-linear for every $\varepsilon$, whereas for sufficiently small values of $\varepsilon$, the running time of the previous algorithm can be as large as quadratic. Like their algorithm, our sketching procedure also yields a small-space, one-pass streaming algorithm which works even if the entries of $x, y$ are given in arbitrary order.

## 1 Introduction

Space and time-efficient processing of massive databases is a challenging and important task in applications such as observational sciences, product marketing, and monitoring large systems. Usually the data set is distributed across several network devices, each receiving a portion of the data as a stream. The devices must locally process their data, producing a small sketch, which can then be efficiently transmitted to other devices for further processing. Although much work has focused on producing sketches of minimal size for various problems, in practice the time efficiency to produce the sketches is just as important, if not more so, than the sketch size.

In the 2006 IITK Workshop on Algorithms for Data Streams, the first open question posed [15] was to find a space- *and* time-efficient algorithm for $L_1$-difference computation. Formally, there are two parties, Alice and Bob, who have vectors $x, y \in \{-M, \ldots, M\}^n$ and wish to compute sketches $s(x)$ and $s(y)$, respectively, so that a third party can quickly recover a value $Z$ with $(1-\varepsilon) \|x - y\|_1 \leq Z \leq (1+\varepsilon) \|x - y\|_1$. Here, $\|x - y\|_1 = \sum_{i=1}^n |x - y|_i$ denotes the $L_1$-norm of the vector $x - y$. The third party should succeed with probability at least 2/3 over the randomness of Alice and Bob (this probability can be amplified by repeating the process and taking the median).

The original motivation [11] for the $L_1$-difference problem is Internet-traffic monitoring. As packets travel through Cisco routers, the NetFlow software [23] produces summary statistics of

[1]MIT Computer Science and Artificial Intelligence Laboratory. `minilek@mit.edu`. Supported by a National Defense Science and Engineering Graduate (NDSEG) Fellowship. Much of this work was done while the author was at the IBM Almaden Research Center.

[2]IBM Almaden Research Center, 650 Harry Road, San Jose, CA, USA. `dpwoodru@us.ibm.com`.

groups of packets with the same source and destination IP address. Such a group of packets is known as a *flow*. At the end of a specified time period, a router assembles sets of values $(s, f_t(s))$, where $s$ is a source-destination pair, and $f_t(s)$ is the total number of bytes sent from the source to the destination in time period $t$. The $L_1$-difference between such sets assembled during different time periods or at different routers indicates differences in traffic patterns.

The ability to produce a short sketch summarizing the set of values allows a central control and storage facility to later efficiently approximate the $L_1$-difference between the sketches that it receives. The routers producing the sketches cannot predict which source-destination pairs they will receive, or in which order. Since the routers can transmit their sketches and updates to their sketches to the central processing facility in an arbitrarily interleaved manner, it is essential that the $L_1$-difference algorithm support arbitrary permutations of the assembled sets of values. Because of the huge size of the packet streams, it is also crucial that the computation time required to produce the sketches be as small as possible.

The first algorithm for this problem is due to Feigenbaum *et al.* [11], and achieves a sketch of size $O(\varepsilon^{-2}\log(nM))$ bits with each party requiring $O(n\log(nM)/\varepsilon^2)$ processing time (in word operations). Later, Indyk [16] generalized this to the problem of estimating the $L_1$-norm of a general data stream with an arbitrary number of updates to each coordinate. For the $L_1$-difference problem, the space of Indyk's method is worse than that of [11] by a factor of $\log(n)$, while the time complexity is similar. Recently, in [21] it was shown how to reduce the space complexity of Indyk's method, thereby matching the sketch size of [11]. However, the time complexity per stream update is $\Omega(\varepsilon^{-2})$. Also in [21], a space lower bound of $\Omega(\varepsilon^{-2}\log(nM))$ was shown for the $L_1$-difference problem for nearly the full range of interesting values for $\varepsilon$, thereby showing that the space complexity of the algorithms of [11, 21] are optimal.

While the space complexity for $L_1$-difference is settled, there are no non-trivial lower bounds for the time complexity. Notice that the $\varepsilon^{-2}$ factor in the processing time can be a severe drawback in practice, and can make the difference between setting the approximation quality to $\varepsilon = .1$ or to $\varepsilon = .01$. Indeed, in several previous works (see the references in Muthukrishnan's book [22], or in Indyk's course notes [17]), the main goal was to reduce the dependence on $\varepsilon$ in the space and/or time complexity. This raises the question, posed in the IITK workshop, as to whether this dependence on $\varepsilon$ can be improved for $L_1$-difference. As a first step, Cormode and Ganguly [14] show that if one increases the sketch size to $\varepsilon^{-3}\mathrm{polylog}(nM)$, then it is possible to achieve processing time $n \cdot \mathrm{polylog}(nM)$, thereby removing the dependence on $\varepsilon$. Their algorithm even works for $L_1$-norm estimation of general data streams, and not just for $L_1$-difference. However, for reasonable values of $nM$, the sketch size is dominated by the $\varepsilon^{-3}$ term, which may be prohibitive in practice, and is sub-optimal.

In this paper we show how to achieve a near-optimal $O(\varepsilon^{-2}\log(1/\varepsilon)\log(nM))$ sketch size, while simultaneously achieving a near linear $O(n\log^2(nM))$ processing time, independent of $\varepsilon$. Notice our space is only a factor of $\log(1/\varepsilon)$ more than the lower bound. The time for a third party to recover a $(1\pm\varepsilon)$-approximation to the $L_1$-difference, given the sketches, is nearly linear in the sketch size. Furthermore, our sketching procedure naturally can be implemented as a one-pass streaming algorithm over an adversarial ordering of the coordinates of $x, y$ (this was also true of previous algorithms). Thus, up to small factors, we resolve the first open question of [15]. We henceforth describe our sketching procedure as a streaming algorithm.

While in [14] and [15] it is suggested to use the techniques of [5] and [19] for estimating $L_p$ norms, $p > 2$, which are themselves based on estimating coordinates of heavy weight individually and removing them, we do not follow this approach. Moreover, the approach of [11] is to embed $L_1$-difference into $L_2^2$, then use the AMS sketch [1] and range-summable hash functions they design to reduce the processing time. We do not follow this approach either.

Instead, our first idea is to embed the $L_1$-difference problem into $L_0$, the number of non-zero coordinates of the underlying vector (in this case $x - y$) presented as data stream. Such an embedding has been used before, for example, in lower bounding the space complexity of estimating $L_0$ in a data stream [18]. Suppose for simplicity $x_i, y_i \geq 0$ for all $i \in [n]$. Here the idea is for Alice to treat her input $x_i$ as a set of distinct items $M(i-1)+1, \ldots, M(i-1)+x_i$, while Bob treats his input $y_i$ as a set of distinct items $M(i-1)+1, \ldots, M(i-1)+y_i$. Then the size of the set-difference of these two sets is $|x_i - y_i|$. Thus, if Alice inserts all of the set elements corresponding to her coordinates as insertions into an $L_0$-algorithm, while Bob inserts all of his elements as deletions, the $L_0$-value in the resulting stream equals $\|x - y\|_1$. A recent space-efficient algorithm for estimating $L_0$ with deletions is given in [21].

The problem with directly reducing to $L_0$ is that, while the resulting space complexity is small, the processing time can be as large as $O(nM)$ since we must insert each set element into the $L_0$-algorithm. We overcome this by developing a range-efficient $L_0$ algorithm, i.e. an algorithm which allows updates to ranges at a time, which works for streams coming out of our reduction by exploiting the structure of ranges we update (all updated ranges are of length at most $M$ and start at an index of the form $M(i-1)+1$). We note that range-efficient $L_0$ algorithms have been developed before [2, 25], but those algorithms do not allow deletions and thus do not suffice for our purposes.

At a high level, our algorithm works by sub-sampling by powers of 2 the universe $[nM]$ arising out of our reduction to $L_0$. At each level we keep a data structure of size $O(\varepsilon^{-2}\log(1/\varepsilon))$ to summarize the items that are sub-sampled at that level. We also maintain a data structure on the side to handle the case when $L_0$ is small, and we in parallel obtain a constant-factor approximation $R$ of the $L_1$-difference using [11]. At the stream's end, we give our estimate of the $L_1$-difference based on the summary data structure living at the level where the expected number of universe elements sub-sampled is $\Theta(1/\varepsilon^2)$ (we can determine this level knowing $R$). As is the case in many previous streaming algorithms, the sub-sampling of the stream can be implemented using pairwise-independent hash functions. This allows us to use a subroutine developed by Pavan and Tirthapura [25] for quickly counting the number of universe elements that are sub-sampled at each of the $\log(nM)$ levels. Given these counts, our summary data structures are such that we can update each one efficiently.

Our summary data structure at a given level maintains $(x' - y')H$, where $H$ is the parity-check matrix of a linear error-correcting code, and $x', y'$ are the vectors derived from $x, y$ by sub-sampling at that level. When promised that $x', y'$ differ on few coordinates, we can treat $x' - y'$ as a corruption of the encoding of the 0 codeword then attempt to decode to recover the "error" $x' - y'$. The decoding succeeds as long as the minimum distance of the code is sufficiently high. This idea of using error-correcting codes to sketch vectors whose distance is promised to be small is known in the error-correcting codes literature as *syndrome decoding* [28]. Aside from error-correction, syndrome decoding has also found uses in cryptography: in the work of [4, 26] to give a two-party protocol for agreeing on a shared secret key when communicating over a noisy channel, and in the work of [10] as part of a private two-party communication protocol for computing the Hamming distance between two bitstrings.

For efficiency reasons, our implementation of the summary data structure is mostly inspired by work of Feigenbaum *et al.* [10]. Given that $x', y'$ differ on at most $k$ coordinates, they use the parity-check matrix of a Reed-Solomon code of minimum distance $O(k)$. Decoding can then be done in time $O(k^2 + k \cdot \text{polylog}(k)\log(n))$ using an algorithm of Dodis *et al.* [9]. In our application $k = \Theta(\varepsilon^{-2})$, and thus this recovery procedure is too slow for our purposes. To remedy this, we first hash the indices of $x', y'$ into $O(\varepsilon^{-2}/\log(1/\varepsilon))$ buckets with an $O(\log(1/\varepsilon))$-wise independent hash function, then in each bucket we keep the product of the difference vector, restricted to the indices mapped to that bucket, with the parity check matrix. With constant probability, no bucket receives more than $O(\log(1/\varepsilon))$ indices where $x', y'$ differ. We can thus use a Reed-Solomon code with minimum distance only $O(\log(1/\varepsilon))$, making the algorithm of [9] fast enough for our purposes.

We note that our summary data structure in each level is in fact a $k$-set structure, as defined by Ganguly [13], that can be used to return a set of $k$ items undergoing insertions and deletions in a data stream. While Ganguly's $k$-set structure uses near-optimal space and has fast update time, it only works in the strict turnstile model (i.e., it requires that each coordinate of $z = x - y$ is non-negative, in which case the $L_1$-difference problem has a trivial solution: maintain an $O(\log(nM))$-bit counter). This is due to the algorithm's reliance on the identity $(\sum_{i=1}^n i \cdot z_i)^2 = (\sum_{i=1}^n z_i)(\sum_{i=1}^n i^2 \cdot z_i)$, for which there is no analogue outside the strict turnstile setting. Using certain modifications inspired by the Count-Min sketch [8] it may be possible to implement his algorithm in the turnstile model, though the resulting space and time would be sub-optimal. In a different work, Ganguly and Majumder [12] design a deterministic $k$-set structure based on Vandermonde matrices, but the space required of this structure is sub-optimal.

Our fast sketching algorithm for $L_1$-difference improves the running time of an algorithm of Jayram and the second author [20] by a $\Theta(\varepsilon^{-2})$ factor for estimating $L_1(L_2)$ of a matrix $A$, defined as the sum of Euclidean lengths of the rows of $A$. As $L_1$-difference is a basic primitive, we believe our algorithm is likely to have many further applications.

## 2 Preliminaries

All space bounds mentioned throughout this paper are in bits, and all logarithms are base 2, unless explicitly stated otherwise. Running times are measured as the number of standard machine word operations (integer arithmetic, bitwise operations, and bitshifts). Each machine word is assumed to be $\Omega(\log(nM))$ bits so that we can index each vector and do arithmetic on vector entries in constant time. Also, for integer $A$, $[A]$ denotes the set $\{1, \ldots, A\}$.

We now formally define the model in which our sketching procedure runs. Alice receives $x \in \{-M, \ldots, M\}^n$, and Bob receives $y \in \{-M, \ldots, M\}^n$. Both parties have access to a shared source of randomness and must, respectively, output bit-strings $s(x)$ and $s(y)$. The requirement is that a third party can, given access to only $s(x)$ and $s(y)$, compute a value $Z$ such that $\mathbf{Pr}[|Z - ||x-y||_1| > \varepsilon ||x-y||_1] \leq 1/3$ (recall $||x-y||_1 \stackrel{\text{def}}{=} \sum_{i=1}^n |x_i - y_i|$). The probability is over the randomness shared by Alice and Bob, and the value $\varepsilon \in (0, 1]$ is a parameter given to all parties. The goal is to minimize the lengths of $s(x)$ and $s(y)$, as well as the amount of time Alice and Bob each take to compute them. Without loss of generality, throughout this document we assume $x_i, y_i \geq 0$ for all $i$. This promise can be enforced by increasing all coordinates of $x, y$ by $M$, which does not alter $||x - y||_1$. Doing so increases the upper bound on coordinate entries by a factor of two, but this alters our algorithm's running time and resulting sketch size by subconstant factors.

Since we present our sketching algorithm as a streaming algorithm in Section 3, we now introduce some streaming notation. We consider a vector $f = (f_1, f_2, \ldots, f_n)$ that is updated in a stream as follows. The stream has exactly $2n$ updates $(i_1, v_1), \ldots, (i_{2n}, v_{2n}) \in [n] \times \{-M, \ldots, M\}$. Each update $(i, v)$ corresponds to the action $f_i \leftarrow f_i + v$. For each $j \in [n]$, there are exactly two stream updates $(i, v)$ with $i = j$. If these two stream updates are $(i_{z_1}, v_{z_1}), (i_{z_2}, v_{z_2})$, then at most one of $v_{z_1}, v_{z_2}$ is negative, and at most one is positive. The nonnegative update corresponds to adding $x_i$ to $f_i$, and the nonpositive update corresponds to subtracting $y_i$ from $f_i$ (recall we assumed $x_i, y_i \geq 0$). We make no restriction on the possible values for $z_1$ and $z_2$. That is, our algorithm functions correctly even if the stream presents us with an adversarial permutation of the $2n$ coordinates $x_1, \ldots, x_n, y_1, \ldots, y_n$. At the end of the stream $||f||_1 = ||x - y||_1$, so our streaming algorithm must approximate $||f||_1$. For Alice and Bob to use our streaming algorithm for sketching, Alice runs the algorithm with updates $(i, x_i)$ for each $1 \leq i \leq n$, and Bob separately runs the algorithm (using the same random bits) with updates $(i, y_i)$. The sketches they produce are simply the contents of the algorithm's memory at the end of the stream. It is a consequence of how our algorithm works that these sketches can be efficiently combined by a third party to approximate $||f||_1$.

# 3 Main Streaming Algorithm

Throughout this section we assume $\varepsilon \geq 1/\sqrt{n}$. Otherwise, we can compute $||f||_1$ exactly by keeping the entire vector in memory using $O(n \log M) = O(\varepsilon^{-2} \log M)$ space with constant update time.

## 3.1 Handling Small $L_1$

We give a subroutine TwoLeveLEstimator, described in Figure 1, to compute $L_1$ exactly when promised that $L_1 \leq k$ for some parameter $k \geq 2$. We assume that integers polynomially large in $k$ fit in a machine word, which will be true in our use of this subroutine later.

In Figure 1, we assume we have already calculated a prime $p$ satisfying

$$C \leq p \leq 2C, \; C = 4 \cdot (5 \lceil \log k \rceil + 24)^2 \cdot \lceil k / \log k \rceil + 1 \tag{1}$$

(the choice of $p$ will be justified later), along with a generator $g$ for the multiplicative group $\mathbb{F}_p^*$. We also precalculate logarithm tables $T_1, T_2$ such that $T_1[i] = g^i \bmod p$ and $T_2[x] = \mathrm{dlog}(x)$, where $0 \leq i \leq p-2$ and $1 \leq x \leq p-1$. Here $\mathrm{dlog}(x)$ is the discrete logarithm of $x$ (i.e. the $i \in \mathrm{GF}(p)$ such that $g^i \equiv x \bmod p$).

The subroutine TwoLevelEstimator makes calls to the following algorithm given in [9].

**Theorem 1** (Dodis *et al.* [9, Lemma E.1]). *Let $p$ be prime and $r = (r_x)_{x \in \mathbb{F}_p^*}$ have at most $s$ non-zero entries ($2s + 1 < p$). Given $\sum_{x \in \mathbb{F}_p^*} r_x x^i$ for $i \in [2s]$, there is an algorithm to recover $\{(x, r_x) | r_x \neq 0\}$ which uses $O(s^2 + s(\log s)(\log \log s)(\log p))$ field operations over* $\mathrm{GF}(p)$. ■

The proof of correctness of TwoLevelEstimator relies in part on the following lemma.

**Lemma 2** (Bellare and Rompel [3, Lemma 2.3]). *Let $X_i \in [0, 1]$, $1 \leq i \leq n$, be $t$-wise independent for $t \geq 4$ an even integer, $X = \sum_{i=1}^n X_i$, and $A > 0$. Then $\mathbf{Pr}[|X - \mathbf{E}[X]| \geq A] \leq 8 \left(\frac{t\mathbf{E}[X] + t^2}{A^2}\right)^{t/2}$.* ■

**Theorem 3.** *Ignoring the space to store the hash functions $h_1, h_2$ and tables $T_1, T_2$, the algorithm* TwoLevelEstimator *uses $O(k \log k)$ bits of space. The hash functions $h_1, h_2$ and tables $T_1, T_2$ require an additional $O((\log k)(\log n) + k \log^2 k)$ bits. The time to process a stream update is $O(\log k)$.*

Subroutine TwoLevelEstimator:

`// Compute` $||f||_1$ `exactly when promised` $||f||_1 \le k$. `The value` $p$ `is as in Eq.` (1).

1. Set $t = 2\lceil \log k \rceil + 12$ and $s = 2t + \lceil \log k \rceil$. Pick a random $h_1 : [n] \to [\lceil k/\log k \rceil]$ from a $t$-wise independent hash family and a random $h_2 : [n] \to [p-1]$ from a pairwise independent family.
2. For each $j \in [\lceil k/\log k \rceil]$ maintain $2s$ counters $X_1^j, \ldots, X_{2s}^j$ modulo $p$, initialized to 0.
3. Upon seeing stream update $(i, v)$, increment $X_z^{h_1(i)}$ by $v \cdot (h_2(i))^z$ for $z \in [2s]$.
4. At the stream's end, for each $j \in [\lceil k/\log k \rceil]$, attempt to recover the non-zero entries of an $s$-sparse vector $f_j = ((f_j)_x)_{x \in \mathbb{F}_p^*}$ satisfying $\sum_{x \in \mathbb{F}_p^*} ((f_j)_x) x^z = X_z^j$ for each $z \in [2s]$ using Theorem 1.
5. Define $\sigma : \mathrm{GF}(p) \to \mathbb{Z}$ to be such that $\sigma(\alpha)$ equals $\alpha$ if $\alpha \le p/2$, and equals $\alpha - p$ otherwise. Output $\sum_{j=1}^{\lceil k/\log k \rceil} \sum_{(f_j)_x \neq 0} |\sigma((f_j)_x)|$.

Figure 1: TwoLevelEstimator subroutine pseudocode

*If* $L_1 \le k$*, the final output value of* TwoLevelEstimator *equals* $L_1$ *exactly with probability at least* 3/4.

**Proof.** Aside from storing $h_1, h_2, T_1, T_2$, the number of counters is $2s \lceil k/\log k \rceil = O(k)$, each of size $O(\log p) = O(\log k)$ bits, totaling $O(k \log k)$ bits. The space to store $h_1$ is $O((\log k)(\log n))$, and the space to store $h_2$ is $O(\log n)$ [7]. The tables $T_1, T_2$ each have $p - 1 = O(k \log k)$ entries, each requiring $O(\log p) = O(\log k)$ bits. Processing a stream update requires evaluating $h_1, h_2$, taking $O(\log k)$ time and $O(1)$ time, respectively [7].

As for update time, each stream token requires updating $2s = O(\log k)$ counters (Step 3). Each counter update can be done in constant time with the help of table lookup since $(h_2(i))^z = g^{z \cdot \mathrm{dlog}(h_2(i))} = T_1[(z \cdot T_2[h_2(i)]) \bmod (p-1)]$.

We now analyze correctness. Define $I = \{i \in [n] : f_i \neq 0 \text{ at the stream's end}\}$. Note $|I| \le L_1 \le k$. For $j \in [\lceil k/\log k \rceil]$, define the random variable $Z_j = |h_1^{-1}(j) \cap I|$. We now define two events.

Let $\mathcal{Q}$ be the event that $Z_j \le s = 2t + \lceil \log k \rceil$ for all $j \in [\lceil k/\log k \rceil]$.

Let $\mathcal{Q}'$ be the event that there do not exist distinct $i, i' \in I$ with both $h_1(i) = h_1(i')$ and $h_2(i) = h_2(i')$.

We first argue that, conditioned on both $\mathcal{Q}, \mathcal{Q}'$ holding, the output of TwoLevelEstimator is correct. Note $p - 1 \ge 4s^2 \lceil k/\log k \rceil \ge 100k \log k$ (recall the definition of $s$ in Step 1 of Figure 1). If $\mathcal{Q}'$ occurs, $|h_2^{-1}(i) \cap h_1^{-1}(j) \cap I| \le 1$ for all $i \in [p-1]$ and $j \in [\lceil k/\log k \rceil]$. One can then view $X_z^j$ as holding $\sum_{x \in \mathbb{F}_p^*} (r_j)_x x^z$, where $(r_j)_x$ is the frequency (modulo $p$) of the unique element in the set $h_2^{-1}(i) \cap h_1^{-1}(j) \cap I$ (or 0 if that set is empty). Conditioned on $\mathcal{Q}$, every $r_j$ is $s$-sparse, so we correctly recover $r_j$ in Step 4 by Theorem 1 since $2s + 1 = 5\lceil \log k \rceil + 13 < 100k \lceil \log k \rceil < p$. Note that $p$ is strictly greater than twice the absolute value of the largest frequency since $L_1 \le k$, and thus negative frequencies are strictly above $p/2$ in $\mathrm{GF}(p)$, and positive frequencies are strictly below $p/2$. Thus, given that the $r_j$ are correctly recovered, $\sigma$ correctly recovers the actual frequencies in

Step 5, implying correctness of the final output.

Now we proceed to lower bound $\mathbf{Pr}[\mathcal{Q} \wedge \mathcal{Q}']$. First we show $\mathcal{Q}$ occurs with probability at least $7/8$. If we let $Z_{j,i}$ indicate $h_1(i) = j$, then note the random variables $\{Z_{j,i}\}_{i \in I}$ are $t$-wise independent and $Z_j = \sum_{i \in I} Z_{j,i}$. Also, $\mathbf{E}[Z_j] = |I| / \lceil k/\log k \rceil \leq \log k$. Noting

$$\mathbf{Pr}[\neg \mathcal{Q}] \leq \mathbf{Pr}[|Z_j - \mathbf{E}[Z_j]| \geq 2t]$$

then setting $A = 2t$ and applying Lemma 2,

$$\begin{aligned} \mathbf{Pr}[|Z_j - \mathbf{E}[Z_j]| \geq 2t] &\leq 8 \left( \frac{t\mathbf{E}[Z_j] + t^2}{(2t)^2} \right)^{t/2} \\ &\leq 8 \left( \frac{2t^2}{4t^2} \right)^{\log k + 6} \leq \frac{1}{8k} \end{aligned}$$

since $\mathbf{E}[Z_j] \leq t$. A union bound implies $\mathbf{Pr}[\mathcal{Q}] \geq 7/8$.

Now we analyze $\mathbf{Pr}[\mathcal{Q}' | \mathcal{Q}]$. Let $Y_{i,i'}$ be a random variable indicating $h_2(i) = h_2(i')$ and define the random variable $Y = \sum_{(i,i') \in \binom{I}{2}, h_1(i) = h_1(i')} Y_{i,i'}$. Note $\mathcal{Q}'$ is simply the event that $Y = 0$. We have

$$\begin{aligned} \mathbf{E}[Y] &= \sum_{j=1}^{\lceil k/\log k \rceil} \mathbf{E}\left[ \sum_{(i,i') \in \binom{h_1^{-1}(j) \cap I}{2}} \mathbf{Pr}[h_2(i) = h_2(i')] \right] \\ &\leq \sum_{j=1}^{\lceil k/\log k \rceil} \frac{\mathbf{E}[|h_1^{-1}(j) \cap I|^2]/2}{p - 1} \\ &\leq \sum_{j=1}^{\lceil k/\log k \rceil} \frac{\mathbf{E}[|h_1^{-1}(j) \cap I|^2]/2}{4s^2 \lceil k/\log k \rceil} \end{aligned}$$

where the expectation on the right side of the first equality is over the random choice of $h_1$, and the probability is over the random choice of $h_2$. The first inequality holds by pairwise independence of $h_2$. Conditioned on $\mathcal{Q}$, $|h_1^{-1}(j) \cap I| \leq s$ for all $j$ so that $\mathbf{E}[Y|\mathcal{Q}] \leq 1/8$, implying $\mathbf{Pr}[\mathcal{Q}'|\mathcal{Q}] = 1 - \mathbf{Pr}[Y \geq 1|\mathcal{Q}] \geq 7/8$ by Markov's Inequality.

In total, we have $\mathbf{Pr}[\mathcal{Q} \wedge \mathcal{Q}'] = \mathbf{Pr}[\mathcal{Q}] \cdot \mathbf{Pr}[\mathcal{Q}'|\mathcal{Q}] \geq (7/8)^2 > 3/4$, and the claim is proven. ■

**Remark 4.** *In Step 1 of Figure 1, we twice pick a hash function $h : [a] \to [b]$ from an $m$-wise independent family for some integers $m$ and $a \neq b$ (namely, when picking $h_1$ and $h_2$). However, known constructions [7] have $a = b$, with $a$ a prime power. This is easily circumvented. When we desire an $h$ with unequal domain size $a$ and range size $b$, we can pick a prime $\ell \geq 2 \cdot \max\{a, b\}$ then pick an $m$-wise independent hash function $h' : [\ell] \to [\ell]$ and define $h(x) \stackrel{\text{def}}{=} (h'(x) \bmod b) + 1$. The family of such $h$ is still $m$-wise independent, and by choice of $\ell$, no range value is more than twice more likely than any other, which suffices for our application with a slight worsening of constant factors.*

The following theorem analyzes the pre-processing and post-processing complexity of TwoLevelEstimator.

**Theorem 5.** *Ignoring the time needed to find the prime $\ell$ in Remark 4, the pre-processing time of* TwoLevelEstimator *before seeing the stream is $O(k \log k)$, and the post-processing time is $O(k \log k \log \log k \log \log \log k)$.*

**Proof.** We first discuss the pre-processing time. It is known that the prime $p$ and generator $g$ for $\mathbb{F}_p^*$ can be found in time $\mathrm{polylog}(C) = \mathrm{polylog}(k)$ (see the proof of Theorem 4 in [6]). Once we have $p, g$, filling in $T_1, T_2$ takes $O(p) = O(k \log k)$ time, which dominates the pre-processing time. The time to allocate the $O(k)$ counters $X_z^j$ is just $O(k)$.

The post-processing work is done in Steps 4 and 5 in Figure 1. For Step 4, there are $O(k/\log k)$ values of $j$, for each of which we run the algorithm of Theorem 1 with $s = O(\log k)$ and $p = O(k \log k)$, thus requiring a total of $O(k \log k \log \log k \log \log \log k)$ field operations over $\mathrm{GF}(p)$. Since we precalculate the table $T_2$, we can do all $\mathrm{GF}(p)$ operations in constant time, including division. In Step 5 we need to sum the absolute values of $O(\log k)$ non-zero entries of $O(k/\log k)$ vectors $f_j$, taking time $O(k)$. ■

**Remark 6.** *Our subroutine* TwoLevelEstimator *uses the fact that since $L_1 \leq k$ and $f$ is an integer vector, it must be the case that $L_0 \leq k$. From here, what we develop is a $k$*-set structure *as defined by Ganguly [13], which is a data structure that allows one to recover the $k$-sparse vector $f$. In fact, any $k$-set structure operating in the turnstile model (i.e., where some $f_i$ can be negative) would have sufficed in place of* TwoLevelEstimator. *We develop our particular subroutine since previous approaches were either less space-efficient or did not work in the turnstile setting [12, 13]. We remark that at the cost of an extra $O(\log^2 k)$ factor in space, but with the benefit of only $O(1)$ post-processing time, one can replace* TwoLevelEstimator *with an alternative scheme. Namely, for each $j \in [\lceil k/\log k \rceil]$, attempt to perfectly the hash the $O(\log k)$ coordinates contributing to $||f||_1$ mapped to $j$ under $h_1$ by pairwise independently hashing into $O(\log^2 k)$ counters, succeeding with constant probability. Each counter holds frequency sums modulo $p$. By repeating $r = \Theta(\log k)$ times and taking the maximum sum of counter absolute values over any of the $r$ trials, we succeed in finding the sum of frequency absolute values of items mapping to $j$ under $h_1$ with probability $1 - 1/\mathrm{poly}(k)$. Thus by a union bound, we recover $||f||_1$ with probability $99/100$ by summing up over all $j$. The estimate of $||f||_1$ can be maintained on the fly during updates to give $O(1)$ post-processing, and updates still take only $O(\log k)$ time.*

## 3.2 The Full Algorithm

Our full algorithm requires, in part, a constant factor approximation to the $L_1$-difference. To obtain this, we can use the algorithm of Feigenbaum *et al.* [11] with $\varepsilon$ a constant.

**Theorem 7** (Feigenbaum *et al.* [11, Theorem 12])**.** *There is a one-pass streaming algorithm for $(1 \pm \varepsilon)$-approximating the $L_1$-difference using $O(\varepsilon^{-2} \log(nM))$ space with update time $O(\varepsilon^{-2} \log(nM))$, and succeeding with probability at least $19/20$.* ■

**Remark 8.** *It is stated in [11] that the update time in Theorem 7 is $O(\varepsilon^{-2}\mathrm{field}(\log(nM)))$, where $\mathrm{field}(D)$ is the time to do arithmetic over $\mathrm{GF}(2^D)$ (not including division). Section 2.2 of [11] points out that $\mathrm{field}(D) = O(D^2)$ naïvely. In fact, it suffices for the purposes of their algorithm to work over $\mathrm{GF}(2^D)$ for the smallest $D \geq \log(nM)$ such that $D = 2 \cdot 3^\ell$, in which case a highly explicit irreducible polynomial of degree $D$ over $\mathbb{F}_2[x]$ (namely $x^D + x^{D/2} + 1$ [27, Theorem 1.1.28]) can be used to perform $\mathrm{GF}(2^D)$ arithmetic in time $O(D)$ in the word RAM model without any additional pre-processing space or time.*

Main Algorithm L1-DIFF:

1. Set $\varepsilon' = \varepsilon/8$.
2. Pick a random hash function $h : [q] \to [q]$ from a pairwise independent family so that $h(x) = ax + b \bmod q$ for some prime $q \in [2nM, 4nM]$ and $a, b \in \mathrm{GF}(q)$.
3. Initialize instantiations $\mathrm{TLE}_1, \ldots, \mathrm{TLE}_{\lceil \log((\varepsilon')^2 nM) \rceil}$ of TWOLEVELESTIMATOR with $k = \lceil 4/(\varepsilon')^2 \rceil$. All instantiations share the same prime $p$, generator $g$, hash functions $h_1, h_2$, and logarithm tables $T_1, T_2$.
4. Upon seeing stream update $(i, v)$, let $v_j$ be the output of the algorithm from Theorem 9 with inputs $a, b$ as in Step 2, $c = c_j = 2^{\lfloor \log q \rfloor - j}$, $d = d_j = 2^{\lfloor \log q \rfloor - j + 1} - 1$, $x = (i-1)M + 1$, $r = |v| - 1$, and $m = q$. Feed the update $(i, \mathrm{sgn}(v) \cdot v_j)$ to $\mathrm{TLE}_j$ for $j = 1, \ldots, \lceil \log((\varepsilon')^2 nM) \rceil$. Let $R_j$ be the output of $\mathrm{TLE}_j$.
5. Run an instantiation TLE of TWOLEVELESTIMATOR in parallel with $k = \lceil 1/(\varepsilon')^2 \rceil$ which receives all updates, using the same $h_1, h_2, p, g, T_1, T_2$ of Step 2. Let its output be $R$.
6. Run the algorithm of Theorem 7 in parallel with error parameter $1/3$ to obtain a value $R' \in [L_1/2, L_1]$.
7. If $R' \leq \lceil 1/(\varepsilon')^2 \rceil$, output $R$. Otherwise, output $q \cdot 2^{\lceil \log((\varepsilon')^2 R') \rceil - \lfloor \log q \rfloor} R_{\lceil \log((\varepsilon')^2 R') \rceil}$.

Figure 2: L1-DIFF pseudocode

We also make use of the following algorithm due to Pavan and Tirthapura [25].

**Theorem 9** (Pavan and Tirthapura [25, Theorem 2])**.** *Let $a, b, c, d, x, r, m$ be integers fitting in a machine word with $m > 0$ and $a, b, c, d \in \{0, \ldots, m-1\}$. There is an algorithm to calculate $|\{i \ : \ (a \cdot (x+i) + b \bmod m) \in [c, d],\ 0 \leq i \leq r\}|$ in time $O(\log(\min(a, r)))$ using $O(\log(r \cdot m))$ space.* ■

Our main algorithm, which we call L1-DIFF, is described in Figure 2. Both in Figure 2 and in the proof of Theorem 10, sgn denotes the function which takes as input a real number $x$ and outputs $-1$ if $x$ is negative, and 1 otherwise.

**Theorem 10.** *The algorithm* L1-DIFF *has update time $O(\log(\varepsilon^2 nM) \log(M/\varepsilon))$ and the space used is $O(\varepsilon^{-2} \log(1/\varepsilon) \log(\varepsilon^{-2} nM))$. Pre-processing requires* $\mathrm{polylog}(nM) + O(\varepsilon^{-2} \log(1/\varepsilon) \log(\varepsilon^{-2} nM))$ *time. Time $O(\varepsilon^{-2} \log(1/\varepsilon) \log\log(1/\varepsilon) \log\log\log(1/\varepsilon))$ is needed for post-processing. The output is $(1 \pm \varepsilon) L_1$ with probability at least $2/3$.*

**Proof.** The hash function $h$ requires $O(\log(nM))$ space. There are $O(\log(\varepsilon^2 nM))$ instantiations of TWOLEVELESTIMATOR (Steps 2 and 4), each with $k = O(\varepsilon^{-2})$, taking a total of $O(\varepsilon^{-2} \log(1/\varepsilon) \log(\varepsilon^2 nM))$ space by Theorem 3. The hash functions $h_1, h_2$ and tables $T_1, T_2$ take $O(\log(1/\varepsilon) \log(n) + \varepsilon^{-2} \log^2(1/\varepsilon)) = O(\varepsilon^{-2} \log(1/\varepsilon) \log n)$ space, also by Theorem 3 (recall we assume $\varepsilon \geq 1/\sqrt{n}$). Step 6 requires only $O(\log(nM))$ space by Theorem 7, since the algorithm is run with error parameter $1/3$.

As for running time, in Step 3 we call the algorithm of Theorem 9 $O(\log(\varepsilon^2 nM))$ times, each time with $a < q$ and $r \leq M$, thus taking a total of $O(\log(\varepsilon^2 nM)\log(\min(q,M))) = O(\log(\varepsilon^2 nM)\log M)$ time. We must also feed the necessary update to each $\mathrm{TLE}_j$, each time taking $O(\log(1/\varepsilon))$ time by Theorem 3. Updating every $\mathrm{TLE}_j$ thus takes time $O(\log(\varepsilon^2 nM)\log(1/\varepsilon))$.

In pre-processing we need to pick a prime $q$ in the desired range, which can be accomplished by picking numbers at random and testing primality; the expected time is $\mathrm{polylog}(nM)$. We also need to prepare $h_1, h_2, T_1, T_2$ and all the TwoLevelEstimator instantiations, which takes $O(\varepsilon^{-2}\log(1/\varepsilon)\log(\varepsilon^2 nM))$ time by Theorem 5, in addition to the $\mathrm{polylog}(n)$ time required to find an appropriate prime $\ell$ as described in Remark 4. The pre-processing time for Step 6 is $O(1)$ (see Figure 1 of [11]).

In post-processing we need to recover the estimate $R'$ from Step 6, which takes $O(1)$ time, then recover an estimate from some TwoLevelEstimator instantiation, so the time is as claimed. In post-processing, to save time one should not run Steps 4 and 5 of TwoLevelEstimator in Figure 1 except at the instantiation whose output is used in Step 7.

Now we analyze correctness. Let $\mathcal{Q}$ be the event that $R' \in [L_1/2, L_1]$. We proceed by a case analysis.

For the first case, suppose $L_1 \leq \lceil 1/(\varepsilon')^2 \rceil$. Then, TLE computes $L_1$ exactly with probability at least $3/4$ by Theorem 3, and hence overall we output $L_1$ exactly with probability at least $(19/20)\cdot(3/4) > 2/3$.

Now, suppose $L_1 > \lceil 1/(\varepsilon')^2 \rceil$. In analyzing this case, it helps to view L1-Diff as actually computing $L_0(f') \stackrel{\mathrm{def}}{=} |\{i : f'_i \neq 0\}|$, where we consider an $nM$-dimensional vector $f'$ that is being updated as follows: when receiving an update $(i,v)$ in the stream, we conceptually view this update as being $|v|$ updates $((i-1)M+1, \mathrm{sgn}(v)), \ldots, ((i-1)M+|v|, \mathrm{sgn}(v))$ to the vector $f'$. Here, the vector $f'$ is initialized to $\vec{0}$. Note that at the stream's end, $L_0(f') = ||f||_1$.

Let $f'^j$ denote the vector whose $i$th entry, $i \in [mM]$, is $f'_i$ if $h(i) \in [c_j, d_j]$ and 0 otherwise. That is, $f'^j$ receives stream updates only from items fed to $\mathrm{TLE}_j$. For $i \in [nM]$, let $X_{i,j}$ be a random variable indicating $h(i) \in [c_j, d_j]$, and let $X_j = \sum_{f'_i \neq 0} X_{i,j}$ so that $X_j = L_0(f'^j)$. Define $p_j \stackrel{\mathrm{def}}{=} (d_j - c_j + 1)/q = 2^{\lfloor \log q \rfloor - j}/q$ so that $\mathbf{E}[X_{i,j}] = p_j$. Thus, $\mathbf{E}[X_j] = p_j \cdot L_0(f')$. Note that $1/2 \leq 2^{\lfloor \log q \rfloor}/q \leq 1$. Conditioned on $\mathcal{Q}$, we have the inequalities

$$\frac{L_0(f')}{2^{\lceil \log((\varepsilon')^2 R') \rceil}} \leq \frac{L_0(f')}{(\varepsilon')^2 R'} \leq \frac{2}{(\varepsilon')^2}$$

and

$$\frac{L_0(f')}{2^{\lceil \log((\varepsilon')^2 R') \rceil}} \geq \frac{L_0(f')}{2(\varepsilon')^2 R'} \geq \frac{1}{2(\varepsilon')^2}$$

By the choice of $j = \lceil \log((\varepsilon')^2 R') \rceil$ in Step 7 of Figure 2, we thus have, assuming $\mathcal{Q}$ occurs,

$$\frac{16}{\varepsilon^2} = \frac{1}{4(\varepsilon')^2} \leq \mathbf{E}[X_j] \leq \frac{2}{(\varepsilon')^2}$$

since $\mathbf{E}[X_j] = p_j \cdot L_0(f') = (2^{\lfloor \log q \rfloor}/q)\cdot(L_0(f')/2^j)$.

Let $\mathcal{Q}'$ be the event that $|X_j - \mathbf{E}[X_j]| \leq \varepsilon \mathbf{E}[X_j]$. Applying Chebyshev's inequality,

$$\mathbf{Pr}[\mathcal{Q}'|\mathcal{Q}] \geq 1 - \frac{\mathbf{Var}[X_j]}{\varepsilon^2 \mathbf{E}^2[X_j]} \geq 1 - \frac{1}{\varepsilon^2 \mathbf{E}[X_j]} \geq \frac{15}{16}$$

The second inequality holds since $h$ is pairwise independent and $X_j$ is the sum of Bernoulli random variables, implying $\mathbf{Var}[X_j] = \sum_i \mathbf{Var}[X_{i,j}] \leq \sum_i \mathbf{E}[X_{i,j}] = \mathbf{E}[X_j]$. The last inequality holds by choice of $\varepsilon' = \varepsilon/8$.

Let $\mathcal{Q}''$ be the event that $\mathrm{TLE}_j$ outputs $X_j$ correctly. Now, conditioned on $\mathcal{Q} \wedge \mathcal{Q}'$, we have $X_j \leq 2(1+\varepsilon)/(\varepsilon')^2 \leq 4/(\varepsilon')^2$ since $\varepsilon \leq 1$. Thus by Theorem 3, $\mathbf{Pr}[\mathcal{Q}''|\mathcal{Q} \wedge \mathcal{Q}'] \geq 3/4$. Overall, we compute $L_1$ of the entire stream correctly with probability at least

$$\begin{aligned} \mathbf{Pr}[\mathcal{Q} \wedge \mathcal{Q}' \wedge \mathcal{Q}''] &= \mathbf{Pr}[\mathcal{Q}] \cdot \mathbf{Pr}[\mathcal{Q}'|\mathcal{Q}] \cdot \mathbf{Pr}[\mathcal{Q}''|\mathcal{Q} \wedge \mathcal{Q}'] \\ &\geq (19/20) \cdot (15/16) \cdot (3/4) > 2/3 \end{aligned}$$

■

Our streaming algorithm also gives a sketching procedure. This is because, as long as Alice and Bob share randomness, they can generate the same $h, h_1, h_2, p, g$ then separately apply the streaming algorithm to their vectors $x, y$. The sketch is then just the state of the streaming algorithm's data structures. Since each stream token causes only linear updates to counters, a third party can then take the counters from Bob's sketch and subtract them from Alice's, then do post-processing to recover the estimation of the $L_1$-difference. The running time for Alice and Bob to produce their sketches is the streaming algorithm's pre-processing time, plus $n$ times the update time. The time for the third party to obtain an approximation to $||x-y||_1$ is the time required to combine the sketches, plus the post-processing time. We thus have the following theorem.

**Theorem 11.** *Sharing* $\mathrm{polylog}(nM)$ *randomness, two parties Alice and Bob, holding vectors* $x, y \in \{-M, \ldots, M\}^n$*, respectively, can produce* $O(\varepsilon^{-2}\log(1/\varepsilon)\log(\varepsilon^2 nM))$*-bit sketches* $s(x), s(y)$ *such that a third party can recover* $||x-y||_1$ *to within* $(1 \pm \varepsilon)$ *with probability at least* $2/3$ *given only* $s(x), s(y)$*. Each of Alice and Bob use time* $O(n\log(\varepsilon^2 nM)\log(M/\varepsilon))$ *to produce their sketches. In* $O(\varepsilon^{-2}(\log(\varepsilon^2 nM) + \log(1/\varepsilon)\log\log(1/\varepsilon)\log\log\log(1/\varepsilon)))$ *time, the third party can recover* $||x-y||_1$ *to within a multiplicative factor of* $(1 \pm \varepsilon)$*.* ■

Note Alice and Bob's running time is always $O(n\log^2(nM))$ since $\varepsilon \geq 1/\sqrt{n}$.

**Remark 12.** *Though we assume Alice and Bob share randomness, to actually implement our algorithm in practice this randomness must be communicated at some point. We note that while the sketch length guaranteed by Theorem 11 is* $O(\varepsilon^{-2}\log(1/\varepsilon)\log(\varepsilon^2 nM))$ *bits, the required amount of shared randomness is* $\mathrm{polylog}(nM)$*, which for large enough* $\varepsilon$ *is larger than the sketch length. This is easily fixed though. Since the required randomness is only polynomially larger than the space used by the sketching algorithm (which is asymptotically equal to the sketch length), the two parties can use the Nisan-Zuckerman pseudorandom generator [24] to stretch a seed whose length is linear in the sketch length to a pseudorandom string of length* $\mathrm{polylog}(nM)$ *which still provides the guarantees of Theorem 11. Alice and Bob then only need to communicate this random seed.*

## Acknowledgments

We thank Avinatan Hassidim, Piotr Indyk, Yuval Ishai, and Swastik Kopparty for useful comments and discussions, and Milan Ružić for pointing out a tweak to an early version of our work which improved our space by a factor of $O(\log(1/\varepsilon))$. We thank Venkat Chandar for pointing out the reference [28], and Silvio Micali for pointing out [6].

## References


[1] Noga Alon, Yossi Matias, and Mario Szegedy. The Space Complexity of Approximating the Frequency Moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999.

[2] Ziv Bar-Yossef, Ravi Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 623–632, 2002.

[3] Mihir Bellare and John Rompel. Randomness-efficient oblivious sampling. In *Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 276–287, 1994.

[4] Charles H. Bennett, Gilles Brassard, and Jean-Marc Robert. Privacy amplification by public discussion. *SIAM J. Comput.*, 17(2):210–229, 1988.

[5] Lakshminath Bhuvanagiri, Sumit Ganguly, Deepanjan Kesh, and Chandan Saha. Simpler algorithm for estimating frequency moments of data streams. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 708–713, 2006.

[6] Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM J. Comput*, 13(4):850–864, 1984.

[7] J. Lawrence Carter and Mark N. Wegman. Universal classes of hash functions. *J. Comput. Syst. Sci.*, 18(2):143–154, 1979.

[8] Graham Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms*, 55(1):58–75, 2005.

[9] Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM J. Comput.*, 38(1):97–139, 2008.

[10] Joan Feigenbaum, Yuval Ishai, Tal Malkin, Kobbi Nissim, Martin J. Strauss, and Rebecca N. Wright. Secure multiparty computation of approximations. *ACM Transactions on Algorithms*, 2(3):435–472, 2006.

[11] Joan Feigenbaum, Sampath Kannan, Martin Strauss, and Mahesh Viswanathan. An approximate L1-difference algorithm for massive data streams. *SIAM J. Comput.*, 32(1):131–151, 2002.

[12] S. Ganguly and A. Majumder. Deterministic $k$-set structure. In *Proceedings of the 25th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, 2006.

[13] Sumit Ganguly. Counting distinct items over update streams. *Theor. Comput. Sci.*, 378(3):211–222, 2007.

[14] Sumit Ganguly and Graham Cormode. On estimating frequency moments of data streams. In *Proceedings of the 11th International Workshop on Randomization and Computation (RANDOM)*, pages 479–493, 2007.

[15] Open Problems in Data Streams and Related Topics. IITK Workshop on Algorithms for Data Streams, 2006. `http://www.cse.iitk.ac.in/users/sganguly/data-stream-probs.pdf`.

[16] Piotr Indyk. Stable distributions, pseudorandom generators, embeddings, and data stream computation. *J. ACM*, 53(3):307–323, 2006.

[17] Piotr Indyk. Sketching, streaming and sublinear-space algorithms, 2007. Graduate course notes available at `http://stellar.mit.edu/S/course/6/fa07/6.895/`.

[18] Piotr Indyk and David P. Woodruff. Tight lower bounds for the distinct elements problem. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 283–, 2003.

[19] Piotr Indyk and David P. Woodruff. Optimal approximations of the frequency moments of data streams. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC)*, pages 202–208, 2005.

[20] T. S. Jayram and David Woodruff. The space complexity of cascaded stream aggregates. Manuscript, 2008.

[21] Daniel M. Kane, Jelani Nelson, and David P. Woodruff. Revisiting norm estimation in data streams. CoRR abs/0811.3648, 2008.

[22] S. Muthukrishnan. Data Streams: Algorithms and Applications. *Foundations and Trends in Theoretical Computer Science*, 1(2):117–236, 2005.

[23] CISCO Netflow, 1998. `http://www.cisco.com/warp/public/732/netflow`.

[24] Noam Nisan and David Zuckerman. Randomness is linear in space. *J. Comput. Syst. Sci.*, 52(1):43–52, 1996.

[25] A. Pavan and Srikanta Tirthapura. Range-efficient counting of distinct elements in a massive data stream. *SIAM J. Comput.*, 37(2):359–379, 2007.

[26] Adam Smith. Scrambling adversarial errors using few random bits, optimal information reconciliation, and better private codes. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 395–404, 2007.

[27] Jacobus Hendricus van Lint. *Introduction to coding theory.* Springer-Verlag, 3rd edition, 1999.

[28] Aaron D. Wyner. Recent Results in the Shannon Theory. *IEEE Trans. Inform. Theory*, 20(1):2–10, 1974.