Fast Manhattan Sketches in Data Streams

Jelani Nelson*
MIT CSAIL
32 Vassar Street
Cambridge, MA 02139
minilek@mit.edu

David P. Woodruff
IBM Almaden Research Center
650 Harry Road
San Jose, CA 95120
dpwoodru@us.ibm.com

ABSTRACT

The ℓ_1-distance, also known as the Manhattan or taxicab distance, between two vectors x, y in $\mathbb{R}^n$ is $\sum_{i=1}^n |x_i - y_i|$. Approximating this distance is a fundamental primitive on massive databases, with applications to clustering, nearest neighbor search, network monitoring, regression, sampling, and support vector machines. We give the first 1-pass streaming algorithm for this problem in the turnstile model with $O^*(\varepsilon^{-2})$ space and $O^*(1)$ update time. The O^* notation hides polylogarithmic factors in ε, n, and the precision required to store vector entries. All previous algorithms either required $\Omega(\varepsilon^{-3})$ space or $\Omega(\varepsilon^{-2})$ update time and/or could not work in the turnstile model (i.e., support an arbitrary number of updates to each coordinate). Our bounds are optimal up to $O^*(1)$ factors.

Categories and Subject Descriptors: F.2.0 [Analysis of Algorithms and Problem Complexity]: General; H.2.8 [Database Management]: Database applications

General Terms: Algorithms, Theory

Keywords: streaming, sketching, clustering, data mining

1. INTRODUCTION

Recent years have witnessed an explosive growth in the amount of available data. Data stream algorithms have become a quintessential tool for analyzing such data. These algorithms have found diverse applications, such as large scale data processing and data warehousing [4, 5, 12, 15, 16, 18, 19, 22, 29, 30], machine learning [40], network monitoring [11, 13, 14, 23, 37, 50], and sensor networks and compressed sensing [6, 28].

A key ingredient in all these applications is a distance measure between data. In nearest neighbor applications, a database of points is compared to a query point to find the nearest match [40]. In clustering, classification, and kernels, e.g., those used for support vector machines (SVM), given a matrix of points, all pairwise distances between the points are computed [40]. In network traffic analysis and denial of service detection, global flow statistics computed using NetFlow software are compared at different times via a distance metric [9, 23]. Seemingly unrelated applications, such as the ability to sample an item in a tabular database proportional to its weight, i.e., to sample from the forward distribution [20], or to sample from the output of a SQL Join [8], require a distance estimation primitive for proper functionality [43].

One of the most robust measures of distance is the ℓ_1-distance, also known as the Manhattan or taxicab distance. The main reason is that it is less sensitive to outliers. Given vectors $x, y \in \mathbb{R}^n$, the ℓ_1-distance is defined as $\|x - y\|_1 \stackrel{\text{def}}{=} \sum_{i=1}^n |x_i - y_i|$. This measure, which also equals twice the total variation distance, is often used in statistical applications for comparing empirical distributions, for which it is more meaningful and natural than Euclidean distance, see, e.g., [32]. It also has a natural interpretation for comparing multisets, whereas Euclidean distance does not. Other applications of ℓ_1 include clustering [24, 36], regression [10, 38, 49, 51] (and with applications to time sequences [21, 38]), Internet-traffic monitoring [23], and similarity search [2]. As stated in [1], in the context of certain nearest-neighbor search problems, "the Manhattan distance metric is consistently more preferable than the Euclidean distance metric for high dimensional data mining applications". It may also support faster indexing for similarity search [57]; in that paper the authors claim their scheme is "up to 10 times faster" when using the ℓ_1-distance in multimodal search than when using Euclidean distance.

Another application is to estimating cascaded norms of a tabular database, i.e. we first compute the ℓ_p norm on a list of attributes of a record, then sum these values up over records. This problem is known as $\ell_1(\ell_p)$ estimation. As discussed in [34], an example application is in the processing of financial data. In a stock market, changes in stock prices are recorded continuously using a quantity $r_{\log}$ known as logarithmic return on investment. To compute the average historical volatility of the stock market from the data, we segment the data by stock, compute the variance of the $r_{\log}$ values for each stock, and then average these over all stocks. This corresponds to an $\ell_1(\ell_2)$ computation (normalized by a constant). As a subroutine for computing $\ell_1(\ell_2)$, the best known algorithms use a routine for ℓ_1-estimation [34].

*Supported by a National Defense Science and Engineering Graduate (NDSEG) Fellowship, and in part by the Center for Massive Data Algorithmics (MADALGO) - a center of the Danish National Research Foundation. Part of this work was done while the author was at the IBM Almaden Research Center.

PODS'10, June 6–11, 2010, Indianapolis, Indiana, USA.

Paper	Space	Update Time	Model
[23]	$O^*(\varepsilon^{-2})$	$O^*(\varepsilon^{-2})$	≤ 2 updates per coordinate
[31, 35, 39]	$O^*(\varepsilon^{-2})$	$O^*(\varepsilon^{-2})$	unrestricted updates
[40]	$O^*(\varepsilon^{-2})$	$O^*(1)$	restricted inputs
[26]	$O^*(\varepsilon^{-3})$	$O^*(1)$	unrestricted updates
[44]	$O^*(\varepsilon^{-2})$	$O^*(1)$	≤ 2 updates per coordinate
this work	$O^*(\varepsilon^{-2})$	$O^*(1)$	unrestricted updates

Figure 1: Chronological comparison of our contribution to previous works on ℓ_1-estimation in data streams.

Despite its many features, understanding the computational properties of the ℓ_1-distance has proven to be quite challenging. In many of the applications above, solutions with Euclidean distance were discovered years before those using the ℓ_1-distance. This is true of linear regression (the singular value decomposition (SVD) versus the later [10]), norm estimation in a data stream ([3] versus the later [31]), private two-party distance approximation ([33] versus the later [42, 56]), and subspace approximation (the SVD versus the later [24]). This difference also occurs in computational geometry, for which clustering a set of points based on minimizing the average squared distance to the center (the 1-means problem) can be solved by a simple formula, whereas minimizing the average distance to the center (the 1-median problem) has no closed form (though solutions exist based on Weiszfeld's algorithm [54] or semidefinite programming [45]). The 1-median is desired because of its robustness [41].

In this paper we study the problem of estimating the ℓ_1-distance in the most general turnstile model of data streaming. Formally, given a total of m updates (positive or negative) to an n-dimensional vector x, we maintain a succinct summary, or *sketch*, of what we have seen so that at any point in time we can output an estimate $E(x)$ so that with high probability, $(1-\varepsilon)\|x\|_1 \leq E(x) \leq (1+\varepsilon)\|x\|_1$, where $\varepsilon > 0$ is a tunable approximation parameter. Here, an update has the form (i, v), meaning that the value v should be added to coordinate i. We assume that v is an integer (this is without loss of generality by scaling), and that $|v| \leq M$, where M is a parameter. We stress that the updates may be interleaved and presented in an arbitrary order. Of interest is the amount of memory to store the sketch, the amount of time to process a coordinate update, and the amount of time to output an estimate upon request. The problem of ℓ_1-sketching has generated excitement in the theory community; see the first open question in the 2006 IITK Workshop on Algorithms for Data Streams [47].

Our Main Contribution. We demonstrate a 1-pass algorithm using $\varepsilon^{-2}\mathrm{polylog}(nmM)$ space for ℓ_1-estimation in data streams with $\mathrm{polylog}(nmM)$ update time, and reporting time $\varepsilon^{-2}\mathrm{polylog}(nmM)$.

Our algorithm is the first to be simultaneously optimal in both the space and the update time up to $\mathrm{polylog}(nmM)$ factors. All previous algorithms (see Section 1.1) either required at least $\varepsilon^{-3}\mathrm{polylog}(nmM)$ bits of space, or at least ε^{-2} update time. As ε can be arbitrarily small, our result can provide a substantial benefit over previous work. In light of known lower bounds, our space and time complexity are optimal up to $\mathrm{polylog}(nmM)$ factors.

Notation: In the remainder of the paper, for a function f we use the notation $O^*(f)$ to denote a function $g = O(f \cdot \mathrm{polylog}(nmM/\varepsilon))$. We similarly define Θ^* and Ω^*.

Our improvements result in corresponding gains for the aforementioned applications. Examples include the scan for nearest neighbor search, for which to obtain sketches of size $O^*(\varepsilon^{-2})$, we reduce the preprocessing time from $O(nd\varepsilon^{-2})$ to $O^*(nd)$. We also shave an ε^{-2} factor in the time for computing all pairwise ℓ_1-distances, in the update time for sampling from the forward distribution, in the time for comparing two collections of traffic-flow summaries, and in the time for estimating cascaded norms.

1.1 Previous Work

A full comparison of our work with previous work is given in Figure 1. We omit the reporting time, since it is the same as the space up to an $O^*(1)$ factor.

The first sublinear-space algorithm for estimating ℓ_1 in a single pass is due to Feigenbaum *et al* [23], who achieve $O^*(\varepsilon^{-2})$ space and $O^*(\varepsilon^{-2})$ update time. There are two drawbacks of this work, the first being that it only works if each coordinate is updated at most twice in the data stream. For some applications this is unrealistic, as in many of the examples above the underlying distribution is skewed, i.e., there are a few coordinates that appear many times. This is especially true when collecting network traffic data, since the underlying vector is indexed by source-destination pairs ("flows"), and one expects each flow to receive multiple packets. The other drawback is the update time.

Subsequent work for sketching ℓ_1 includes the algorithm of Indyk [31], with followup work by Li [39] and a space-optimal variant by Kane and the current authors [35]. As in [23], these algorithms achieve optimal or near-optimal space. They can additionally handle an arbitrary number of updates to each coordinate, eliminating the first drawback above. The downside is that these algorithms require $\Omega^*(\varepsilon^{-2})$ time to process each stream update, which is slow for small error parameters. Note that for some applications, the setting of ε may simply be determined by wishing to obtain the smallest approximation error possible under the sole constraint that the sketching algorithm fits in memory. However, if for example a device has 2^{32} words of memory available, setting ε based on the constraint $\Theta^*(\varepsilon^{-2}) = 2^{32}$ would make the streaming problem intractable due to intolerably slow performance with an update time of $\Omega^*(\varepsilon^{-2})$.

The first approach to bypass the $\Omega^*(\varepsilon^{-2})$ time barrier is due to Li *et al* [40], and is based on the concept of sparse Cauchy sketches (for detail on Cauchy sketches, see below). This approach though has the drawback that it only works if the input data satisfies certain uniformity conditions, whereas the algorithms above work for worst-case inputs. The first algorithm to break the $\Omega^*(\varepsilon^{-2})$ time barrier for worst-case inputs while using nontrivially small space is that of Ganguly and Cormode [26]. The authors handle an arbitrary number of updates to each coordinate, and achieve $O^*(1)$ update time. This, however, comes at the cost of an $\Omega^*(\varepsilon^{-3})$ requirement on the space. We also obtain an incomparable result in an unpublished manuscript [44], which although achieving $O^*(1)$ update time and $O^*(\varepsilon^{-2})$ space (improving upon [23]), only works if each coordinate is updated at most twice in the stream. The main question left open in this line of work is whether one could obtain $O^*(1)$

update time and $O^*(\varepsilon^{-2})$ space for an arbitrary number of updates per coordinate. Such an algorithm would match known lower bounds [35, 55] up to $O^*(1)$ factors.

We remark that the work of Ganguly and Cormode [26] only appears to be correct under the assumption of the existence of a truly random hash function, as their use of Nisan's generator (Appendix B in their paper) increases their update time by a factor of $\Omega^*(\varepsilon^{-2})$. However, it appears that one can fix their argument using additional methods [25].

1.2 Our Techniques

Given the large body of prior work on this problem, the simplicity of our solution is quite surprising. In [26] it is suggested that using the Cauchy sketches of Li (particularly, his *geometric mean estimator*) would require $\Omega^*(\varepsilon^{-2})$ update time. The authors then used multi-level sketches, incurring an extra $\Omega^*(\varepsilon^{-1})$ factor in the space. Contrary to the intuition given in [26], we achieve $O^*(1)$ update time by using Cauchy sketches (and particularly, Li's geometric mean estimator)! However, to achieve our result we first need to preprocess and partition the data; details follow.

The notion of a Cauchy sketch was pioneered by Indyk [31] and refined by Li [39]. Given a vector x, the sketch is a collection of counters $Y_j = \sum_{i=1}^n x_i C_{i,j}$ for $j = 1, \ldots, k$, where the $C_{i,j}$ are standard Cauchy random variables with probability density function $\mu(y) = \frac{1}{\pi(1+y^2)}$. The $C_{i,j}$ are generated pseudorandomly using Nisan's pseudorandom generator (PRG) [46]. By the 1-stability of the Cauchy distribution, Y_j is also distributed as a standard Cauchy random variable, scaled by $\|x\|_1$. Li shows that there is a constant $c_k > 0$ so that for any $k \geq 3$, if $Y_1, \ldots, Y_k$ are independent Cauchy sketches, then the geometric mean estimator

$$\mathrm{Est}_{\mathrm{GM}} = c_k \cdot (|Y_1| \cdot |Y_2| \cdots |Y_k|)^{1/k},$$

has an expected value $\mathbf{E}[\mathrm{Est}_{\mathrm{GM}}] = \|x\|_1$ and a variance of $\mathbf{Var}[\mathrm{Est}_{\mathrm{GM}}] = \Theta(\|x\|_1^2/k)$. The space and time complexity of maintaining the Y_j in a data stream are $O^*(k)$, and by linearity, can be computed in a single pass. By Chebyshev's inequality, for $k = \Theta(\varepsilon^{-2})$ one obtains a $(1 \pm \varepsilon)$-approximation to $\|x\|_1$ with constant probability, which can be amplified by taking the median of independent repetitions. While the space needed is $O^*(\varepsilon^{-2})$, unfortunately so is the update time.

Our starting point is the following idea. Suppose we randomly partition the coordinates into $\Theta(\varepsilon^{-2})$ buckets. In each bucket we maintain Li's estimator but only with parameter $k = 3$. Given an update to a coordinate i, it lands in a unique bucket, and the contents of this bucket can be updated in $O^*(1)$ time. Using $\Theta(\varepsilon^{-2})$ buckets, the space is also $O^*(\varepsilon^{-2})$. One is then faced with the following temptation: letting G_i be the estimate returned by Li's procedure in bucket i for $k = 3$, output $G = \sum_{i=1}^r G_i$. From the properties of the G_i, this is correct in expectation.

The main wrinkle is that $\mathbf{Var}[G]$ can be as large as $\Omega(\|x\|_1^2)$, which is not good enough. To see that this can happen, suppose x contains only a single non-zero coordinate $x_1 = 1$. In the bucket containing x_1, the value G of Li's estimator is the geometric mean of 3 standard Cauchy random variables. By the above, $\mathbf{Var}[G] = \Theta(\|x\|_1^2/k) = \Theta(\|x\|_1^2)$.

Note though in the above example, x_1 contributed a large fraction of the ℓ_1 mass of x (in fact, all of it). Our main idea then is the following. A *ϕ-heavy hitter* of the vector x is a coordinate i for which $|x_i| \geq \phi \cdot \|x\|_1$. Algorithms for finding heavy hitters, also known as iceberg queries, have been extensively studied in the database community [4, 5, 7, 17, 22, 29, 30], and we can use such algorithms in our algorithm. Set $\phi = \varepsilon^2$. We remove every ϕ-heavy hitter from x, estimate the contribution of these heavy coordinates separately, then use the bucketing above on the remaining coordinates. We show that this reduces $\mathbf{Var}[G]$ to $O(\|x_{\mathrm{tail}}\|_2^2)$, where x_{tail} is the vector obtained from x by removing the heavy hitters. A calculation shows that $O(\|x_{\mathrm{tail}}\|_2^2) = O(\varepsilon^2\|x\|_1^2)$, which is good enough to argue that $\|x_{\mathrm{tail}}\|_1$ can be estimated to within an additive $\varepsilon\|x\|_1$ with constant probability. This idea can be implemented in a single pass.

The main remaining hurdle is estimating $\|x_{\mathrm{head}}\|_1$, the contribution to $\|x\|_1$ from the heavy hitters. Using current techniques, we could, say, use the CountMin sketch [17] to estimate the value of each ε^2-heavy hitter up to an additive $\varepsilon^3\|x\|_1$. Summing the estimates gives $\|x_{\mathrm{head}}\|_1$ up to an additive $\varepsilon\|x\|_1$. This, however, requires $\Omega^*(\varepsilon^{-3})$ space, which we cannot afford. We instead design a new subroutine, Filter, which estimates the sum of the absolute values of the heavy hitters, i.e., the value $\|x_{\mathrm{head}}\|_1$, up to an additive $\varepsilon\|x\|_1$, without guaranteeing an accurate frequency estimate to any *individual* heavy hitter. This relaxed guarantee is sufficient for correctness of our overall algorithm, and is implementable in $O^*(\varepsilon^{-2})$ space.

Other technical complications arise due to the fact that the partitioning is not truly random, nor is the randomness used by Li's estimator. We use a family of functions of Pagh and Pagh [48] that is close to an $O(\varepsilon^{-2})$-wise independent family, but doesn't suffer the $O(\varepsilon^{-2})$ evaluation time required of functions in such families (e.g., $O(\varepsilon^{-2})$-degree polynomial evaluation). These functions can be evaluated in constant time. The caveat is that the correctness analysis needs more attention. Also, naïvely implementing Li's algorithm requires $\Omega(n)$ space to store the Cauchy random variables assigned to coordinates (recall each coordinate can be updated multiple times in an arbitrary order). The randomness can be reduced using Nisan's PRG, but previous applications of such a PRG resulted in time proportional to the space of the algorithm, which here would be $\Omega^*(\varepsilon^{-2})$. We show that for our algorithm, Nisan's PRG can be applied in a way that only adds $O^*(1)$ to the update time.

At a high level, our algorithm is inspired by works of Charikar *et al* [7] and Thorup and Zhang [52], who give implementations of the AMS sketch [3] for ℓ_2-estimation with fast update time. Similarly to our algorithm, they hashed each index to a bucket then performed an unbiased ℓ_2^2 estimator in the bucket, but unlike in our algorithm, they did not have to handle heavy hitters separately.

2. PRELIMINARIES

Our algorithm operates in the following model. A vector x of length n is initialized to 0, and it is updated in a stream of m updates from the set $[n] \times \{-M, \ldots, M\}$. An update (i, v) corresponds to the change $x_i \leftarrow x_i + v$. In this work, we are interested in computing a $(1 \pm \varepsilon)$-approximation to $\|x\|_1 = \sum_{i=1}^n |x_i|$ for some given parameter $\varepsilon > 0$. All space bounds in 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). We differentiate between *update time*, which is the time to process a stream update, and *reporting time*, which is the time required to output an answer. Each machine word is assumed to be $\Omega(\log(nmM/\varepsilon))$ bits so that we can index each vector and do arithmetic on vector entries and the input approximation parameter in constant time.

Throughout this document, for integer z we use $[z]$ to denote the set $\{1,\dots,z\}$. For reals A, B, we use $A \pm B$ to denote some value in the interval $[A-B, A+B]$. Whenever we discuss a frequency x_i, we are referring to that frequency at the stream's end. We also assume $\|x\|_1 \neq 0$ without loss of generality (note $\|x\|_1 = 0$ iff $\|x\|_2 = 0$, and the latter can be detected with arbitrarily large constant probability in $O(\log(nmM))$ space and $O(1)$ update and reporting time by, say, the AMS sketch [3]), and that $\varepsilon < \varepsilon_0$ for some fixed constant ε_0.

3. ℓ_1 STREAMING ALGORITHM

In this section we describe and analyze our algorithm for $(1 \pm \varepsilon)$-approximating $\|x\|_1$. As discussed in Section 1.2, the algorithm works by estimating the contribution to ℓ_1 from the heavy hitters and non heavy hitters separately, then summing these estimates.

A "ϕ-heavy hitter" is an index i such that that $|x_i| \geq \phi\|x\|_1$. We use a known heavy hitter algorithm for the turnstile model of streaming (the model we are currently operating in) to identify the ε^2-heavy hitters. Given this information, we use a subroutine Filter (described in Section 3.1) to estimate the contribution of these heavy hitters to ℓ_1 up to an additive error of $\varepsilon\|x\|_1$. This takes care of the contribution from heavy hitters.

We maintain, in parallel, $R = \Theta(1/\varepsilon^2)$ "buckets" B_i, which then allow us to estimate the contribution from non heavy hitters. Each index in $[n]$ is hashed to exactly one bucket $i \in [R]$. The ith bucket keeps track of the dot product of x, restricted to those indices hashed to i, with three random Cauchy vectors, then applies a known unbiased estimator of ℓ_1 due to Li [39] (the "geometric mean estimator") to estimate the ℓ_1 norm of x restricted to indices hashed to i. We then sum up (some scaling of) the estimates from the buckets not containing any ε^2-heavy hitters. The value of the summed estimates turns out to be approximately correct in expectation. Then, using that the summed estimates only come from buckets without heavy hitters, we are able to show that the variance is also fairly small, which then allows us to show that our estimation of the contribution from the non heavy hitters is correct up to $\varepsilon\|x\|_1$ with large probability.

We first describe Filter, in Section 3.1, then give the final algorithm in Section 3.2.

3.1 The Filter data structure: estimating the contribution from heavy hitters

In this section, we assume we know a subset $L \subseteq [n]$ of indices i so that (1) for all i for which $|x_i| \geq \varepsilon^2\|x\|_1$, $i \in L$, and (2) for all $i \in L$, $|x_i| \geq (\varepsilon^2/2)\|x\|_1$. Note this implies $|L| \leq 2/\varepsilon^2$. Furthermore, we suppose we know $\mathrm{sign}(x_i)$ for each $i \in L$. Throughout this section, we let x_{head} denote the vector x projected onto coordinates in L, so that $\sum_{i\in L}|x_i| = \|x_{\mathrm{head}}\|_1$. The culmination of this section is Theorem 3, which shows that we can obtain an estimate $\Phi = \|x_{\mathrm{head}}\|_1 \pm \varepsilon\|x\|_1$ in small space with large probability, via a subroutine we dub Filter.

We now proceed to give the details. We require the following uniform hash family construction given in [48].

THEOREM 1 (PAGH AND PAGH [48, THEOREM 1.1]). *Let $S \subseteq U = [u]$ be a set of $z > 1$ elements, and let $V = [v]$, with $1 < v \leq u$. Suppose the machine word size is $\Omega(\log(u))$. For any constant $c > 0$ there is a word RAM algorithm that, using time $\log(z)\log^{O(1)}(v)$ and $O(\log(z) + \log\log(u))$ bits of space, selects a family $\mathcal{H}$ of functions from U to V (independent of S) such that:*

1. *With probability $1 - O(1/z^c)$, $\mathcal{H}$ is z-wise independent when restricted to S.*
2. *Any $h \in \mathcal{H}$ can be represented by a RAM data structure using $O(z\log(v))$ bits of space, and h can be evaluated in constant time after an initialization step taking $O(z)$ time.*

We define the BasicFilter data structure as follows. Choose a random sign vector $\sigma \in \{-1,1\}^n$ from a 4-wise independent family. Put $r = \lceil 27/\varepsilon^2 \rceil$. We choose a hash function $h : [n] \to [r]$ at random from a family $\mathcal{H}$ constructed randomly as in Theorem 1 with $u = n, v = z = r, c = 1$. Note $|L| + 1 < z$. We also initialize r counters $b_1, \dots, b_r$ to 0. Given an update of the form (i, v), add $\sigma(i) \cdot v$ to $b_{h(i)}$.

We define the Filter data structure as follows. Initialize $s = \lceil \log_3(1/\varepsilon^2) \rceil + 3$ independent copies of the BasicFilter data structure. Given an update (i, v), perform the update described above to each of the copies of BasicFilter. We think of this data structure as an $s \times r$ matrix of counters $D_{i,j}$, $i \in [s]$ and $j \in [r]$. We let σ^i denote the sign vector σ in the i-th independent instantiation of BasicFilter, and similarly define h^i and $\mathcal{H}^i$. Notice that the space complexity of Filter is $O(\varepsilon^{-2}\log(1/\varepsilon)\log(mM) + \log(1/\varepsilon)\log\log n)$. The update time is $O(\log(1/\varepsilon))$.

For each $w \in L$ for which $h^i(w) = j$, say a count $D_{i,j}$ is *good for* w if for all $y \in L\backslash\{w\}$, $h^i(y) \neq j$. Since h^i is $|L|$-wise independent when restricted to L with probability at least $1 - 1/r$, we have that $\mathbf{Pr}[D_{i,j} \text{ is good for } w] \geq (1 - 1/r) \cdot (1 - (|L| - 1)/r) \geq 2/3$, where the second inequality holds for $\varepsilon \leq 1$. It follows that since Filter is the concatenation of s independent copies of BasicFilter,

$$\mathbf{Pr}[\forall w \in L,\ \exists i \in [s] \text{ for which } D_{i,h^i(w)} \text{ is good for } w] \geq 1 - |L| \cdot \left(\frac{1}{3^s}\right) > \frac{9}{10}. \quad (1)$$

Let $\mathcal{E}$ be the event of Eq. (1).

We define the following estimator Φ of $\|x_{\mathrm{head}}\|_1$ given the data in the Filter structure, together with the list L. We also assume $\mathcal{E}$ holds, else our estimator is not well-defined. For each $w \in L$, let $i(w)$ be the smallest i for which $D_{i,h^i(w)}$ is good for w, and let $j(w) = h^{i(w)}(w)$. Our estimator is then

$$\Phi = \sum_{w\in L} \mathrm{sign}(x_w) \cdot \sigma^{i(w)}(w) \cdot D_{i(w),j(w)}.$$

Note our Filter data structure is quite similar to the CountSketch structure of [7], but with universal hashing replaced by uniform hashing, and with a different estimation procedure.

LEMMA 2. $\mathbf{E}[\Phi \mid \mathcal{E}] = \|x_{\mathrm{head}}\|_1$ *and* $\mathbf{Var}[\Phi \mid \mathcal{E}] \leq 2\varepsilon^2\|x\|_1^2/9$.

PROOF. By linearity of expectation,

$$\mathbf{E}[\Phi \mid \mathcal{E}] = \sum_{w\in L} \mathbf{E}[\mathrm{sign}(x_w) \cdot \sigma^{i(w)}(w) \cdot D_{i(w),j(w)} \mid \mathcal{E}].$$

Fix a $w \in L$, and for notational convenience let $i = i(w)$ and $j = j(w)$. For each $y \in [n]$, set $\Gamma(y) = 1$ if $h^i(y) = j$, and set $\Gamma(y) = 0$ otherwise. Then

$$\begin{aligned}\mathbf{E}_{\sigma^i,h^i}&[\mathrm{sign}(x_w)\cdot\sigma^i(w)\cdot D_{i,j} \mid \mathcal{E}] \\ &= \sum_y \mathbf{E}_{\sigma^i,h^i}[\mathrm{sign}(x_w)x_y\Gamma(y)\sigma^i(y)\sigma^i(w) \mid \mathcal{E}].\end{aligned}$$

Consider any fixing of h^i subject to the occurrence of $\mathcal{E}$, and notice that σ^i is independent of h^i. Since σ^i is 4-wise independent, it follows that

$$\begin{aligned}\mathbf{E}_{\sigma^i}&[\mathrm{sign}(x_w)\sigma^i(w)D_{i,j} \mid h^i] \\ &= \mathbf{E}_{\sigma^i,h^i}[\mathrm{sign}(x_w)x_w\Gamma(w)\sigma^i(w)\sigma^i(w)] = |x_w|, \qquad (2)\end{aligned}$$

and hence

$$\mathbf{E}[\Phi \mid \mathcal{E}] = \sum_{w\in L}|x_w| = \|x_{\mathrm{head}}\|_1.$$

We now bound $\mathbf{Var}[\Phi \mid \mathcal{E}] = \mathbf{E}[\Phi^2 \mid \mathcal{E}] - \mathbf{E}^2[\Phi \mid \mathcal{E}]$, or equivalently,

$$\begin{aligned}\mathbf{Var}[\Phi \mid \mathcal{E}] &= -\|x_{\mathrm{head}}\|_1^2 \\ &\quad + \sum_{w,y\in L}\mathbf{E}[\mathrm{sign}(x_w)\mathrm{sign}(x_y)\sigma^{i(w)}(w)\sigma^{i(y)}(y) \\ &\qquad\qquad \times D_{i(w),j(w)}D_{i(y),j(y)} \mid \mathcal{E}] \\ &= -\|x_{\mathrm{head}}\|_1^2 + \sum_{w\in L}\mathbf{E}[D_{i(w),j(w)}^2 \mid \mathcal{E}] \\ &\quad + \sum_{w\neq y\in L}\mathbf{E}[\mathrm{sign}(x_w)\mathrm{sign}(x_y)\sigma^{i(w)}(w)\sigma^{i(y)}(y) \\ &\qquad\qquad \times D_{i(w),j(w)}D_{i(y),j(y)} \mid \mathcal{E}]\end{aligned}$$

We first bound $\sum_{w\in L}\mathbf{E}[D_{i(w),j(w)}^2 \mid \mathcal{E}]$. Fix a $w \in L$, and for notational convenience, put $i = i(w)$ and $j = j(w)$. Then,

$$\begin{aligned}\mathbf{E}[D_{i,j}^2 \mid \mathcal{E}] &= \sum_{y,y'}\mathbf{E}[x_yx_{y'}\Gamma(y)\Gamma(y')\sigma^i(y)\sigma^i(y') \mid \mathcal{E}] \\ &= \sum_y \mathbf{E}[x_y^2\cdot\Gamma(y) \mid \mathcal{E}] \\ &= \sum_y x_y^2\cdot\mathbf{Pr}[h^i(y) = j \mid \mathcal{E}],\end{aligned}$$

where the second equality follows from the fact that σ^i is 4-wise independent and independent of $\mathcal{E}$. Note $\mathbf{Pr}[h^i(y) = j \mid \mathcal{E}] = 0$ for any $y \in (L\setminus\{w\})$, and $\mathbf{Pr}[h^i(w) = j \mid \mathcal{E}] = 1$ by definition.

Now consider a coordinate $y \notin L$. For $S \subseteq [n]$ let $\mathcal{F}_S^i$ be the event that $\mathcal{H}^i$ is $|S|$-wise independent when restricted to S. By Bayes' rule,

$$\begin{aligned}\mathbf{Pr}[\mathcal{F}_{L\cup\{y\}}^i \mid \mathcal{E}] &= \frac{\mathbf{Pr}[\mathcal{E} \mid \mathcal{F}_{L\cup\{y\}}^i]\cdot\mathbf{Pr}[\mathcal{F}_{L\cup\{y\}}^i]}{\mathbf{Pr}[\mathcal{E}]} \\ &= \frac{\left(\mathbf{Pr}[\mathcal{E}] - \mathbf{Pr}[\mathcal{E} \mid \neg\mathcal{F}_{L\cup\{y\}}^i]\cdot\mathbf{Pr}[\neg\mathcal{F}_{L\cup\{y\}}^i]\right)}{\mathbf{Pr}[\mathcal{E}]} \\ &\geq \frac{\mathbf{Pr}[\mathcal{E}] - \frac{1}{r}}{\mathbf{Pr}[\mathcal{E}]} \\ &\geq 1 - \frac{10}{9r}\end{aligned}$$

Conditioned on $\mathcal{F}_{L\cup\{y\}}^i$, the value $h^i(y)$ is uniformly random even given the images of all members in L under h^i. Thus, $\mathbf{Pr}[h^i(y) = j \mid \mathcal{E}] \leq 10/(9r) + 1/r < 3/r$. Since the bucket is good for w, the total contribution of such y to $\mathbf{E}[D_{i,j}^2 \mid \mathcal{E}]$ is at most $3\cdot\|x_{\mathrm{tail}}\|_2^2/r$, where x_{tail} is the vector x with the coordinates in L removed. Then $\|x_{\mathrm{tail}}\|_2^2$ is maximized when there are ε^{-2} coordinates each of magnitude $\varepsilon^2\|x\|_1$. In this case $\|x_{\mathrm{tail}}\|_2^2 = \varepsilon^2\|x\|_1^2$.

Hence,

$$\mathbf{E}[D_{i,j}^2 \mid \mathcal{E}] \leq x_w^2 + 3\varepsilon^2\|x\|_1^2/r \leq x_w^2 + \varepsilon^4\|x\|_1^2/9.$$

As $|L| \leq 2\varepsilon^{-2}$, it follows that

$$\sum_{w\in L}\mathbf{E}[D_{i(w),j(w)}^2 \mid \mathcal{E}] \leq 2\varepsilon^2\|x\|_1^2/9 + \sum_{w\in L}x_w^2.$$

Finally, we turn to bounding

$$\begin{aligned}\sum_{w\neq y\in L}&\mathbf{E}[\mathrm{sign}(x_w)\mathrm{sign}(x_y)\sigma^{i(w)}(w)\sigma^{i(y)}(y) \\ &\qquad \times D_{i(w),j(w)}D_{i(y),j(y)} \mid \mathcal{E}].\end{aligned}$$

Fix distinct $w, y \in L$. Note that $(i(w), j(w)) \neq (i(y), j(y))$ conditioned on $\mathcal{E}$ occurring.

Suppose first that $i(w) \neq i(y)$. Then

$$\begin{aligned}&\mathbf{E}[\mathrm{sign}(x_w)\mathrm{sign}(x_y)\sigma^{i(w)}(w)\sigma^{i(y)}(y)D_{i(w),j(w)}D_{i(y),j(y)} \mid \mathcal{E}] \\ &= \mathbf{E}[\mathrm{sign}(x_w)\sigma^{i(w)}D_{i(w),j(w)} \mid \mathcal{E}] \\ &\quad \times \mathbf{E}[\mathrm{sign}(x_y)\sigma^{i(y)}D_{i(y),j(y)} \mid \mathcal{E}] \\ &= |x_w|\cdot|x_y|\end{aligned}$$

since it holds for any fixed $h^{i(w)}, h^{i(y)}$, where the final equality follows from Eq. (2).

Now suppose that $i(w) = i(y)$. Let $i = i(w) = i(y)$ for notational convenience. Define the indicator random variable $\Gamma^w(z) = 1$ if $h^i(z) = j(w)$, and similarly let $\Gamma^y(z) = 1$ if $h^i(z) = j(y)$. Then we can expand the expression $\mathbf{E}[\mathrm{sign}(x_w)\mathrm{sign}(x_y)\sigma^{i(w)}(w)\sigma^{i(y)}(y)D_{i(w),j(w)}D_{i(y),j(y)} \mid \mathcal{E}]$ using the definition of $D_{i(w),j(w)}$ and $D_{i(y),j(y)}$ as

$$\begin{aligned}\sum_{z,z'}&\mathbf{E}[\mathrm{sign}(x_w)\mathrm{sign}(x_y)x_zx_{z'}\Gamma^w(z)\Gamma^y(z')\sigma^i(z)\sigma^i(z') \\ &\qquad \times \sigma^i(w)\sigma^i(y) \mid \mathcal{E}].\end{aligned}$$

We fix z and z' and analyze a summand of the form

$$\begin{aligned}\mathbf{E}&[\mathrm{sign}(x_w)\mathrm{sign}(x_y)x_zx_{z'}\Gamma^w(z)\Gamma^y(z') \\ &\qquad \times \sigma^i(z)\sigma^i(z')\sigma^i(w)\sigma^i(y) \mid \mathcal{E}].\end{aligned}$$

Consider any fixing of h^i subject to the occurrence of $\mathcal{E}$, and recall that σ^i is independent of h^i. Since σ^i is 4-wise independent and a sign vector, it follows that this summand vanishes unless $\{z, z'\} = \{w, y\}$. Moreover, since $\Gamma^w(y) = \Gamma^y(w) = 0$, while $\Gamma^w(w) = \Gamma^y(y) = 1$, we must have $z = w$ and $z' = y$. In this case,

$$\begin{aligned}\mathbf{E}&[\mathrm{sign}(x_w)\mathrm{sign}(x_y)x_zx_{z'}\Gamma^w(z)\Gamma^y(z') \\ &\times \sigma^i(z)\sigma^i(z')\sigma^i(w)\sigma^i(y) \mid h^i] = |x_w|\cdot|x_y|.\end{aligned}$$

Hence, the total contribution of all distinct $w, y \in L$ to $\mathbf{Var}[\Phi \mid \mathcal{E}]$ is at most $\sum_{w\neq y\in L}|x_w|\cdot|x_y|$.

Combining our bounds, it follows that

$$\begin{aligned}\mathbf{Var}[\Phi \mid \mathcal{E}] &\le -\|x_{\mathrm{head}}\|_1^2 + 2\varepsilon^2\|x\|_1^2/9 \\ &\quad + \sum_{w\in L} x_w^2 + \sum_{w\ne y\in L} |x_w|\cdot|x_y| \\ &= -\|x_{\mathrm{head}}\|_1^2 + 2\varepsilon^2\|x\|_1^2/9 + \left(\sum_{w\in L}|x_w|\right)^2 \\ &= -\|x_{\mathrm{head}}\|_1^2 + 2\varepsilon^2\|x\|_1^2/9 + \|x_{\mathrm{head}}\|_1^2 \\ &= 2\varepsilon^2\|x\|_1^2/9.\end{aligned}$$

This completes the proof of the lemma. □

By Chebyshev's inequality, Lemma 2 implies

$$\begin{aligned}\mathbf{Pr}\left[|\Phi - \|x_{\mathrm{head}}\|_1| \ge \varepsilon\|x\|_1 \mid \mathcal{E}\right] &\le \frac{\mathbf{Var}[\Phi \mid \mathcal{E}]}{\varepsilon^2\|x\|_1^2} \\ &\le \frac{2\varepsilon^2\|x\|_1^2}{9\varepsilon^2\|x\|_1^2} = \frac{2}{9},\end{aligned}$$

and thus,

$$\mathbf{Pr}\left[(|\Phi - \|x_{\mathrm{head}}\|_1| \le \varepsilon\|x\|_1) \wedge \mathcal{E}\right] \ge \left(\frac{7}{9}\right)\cdot\left(\frac{9}{10}\right) = \frac{7}{10}.$$

We summarize our findings with the following theorem.

THEOREM 3. *Suppose we have a set $L \subseteq [n]$ of indices j so that (1) for all j for which $|x_j| \ge \varepsilon^2\|x\|_1$, $j \in L$, and (2) for all $j \in L$, $|x_j| \ge (\varepsilon^2/2)\|x\|_1$. Further, suppose we know $\mathrm{sign}(x_j)$ for all $j \in L$.*

Then, there is a 1-pass algorithm, Filter*, which outputs an estimate Φ for which with probability at least $7/10$, $|\Phi - \|x_{\mathrm{head}}\|_1| \le \varepsilon\|x\|_1$. The space complexity of the algorithm is $O(\varepsilon^{-2}\log(1/\varepsilon)\log(mM) + \log(1/\varepsilon)\log\log n)$. The update time is $O(\log(1/\varepsilon))$, and the reporting time is $O(\varepsilon^{-2}\log(1/\varepsilon))$.*

3.2 The final algorithm

We now analyze our final algorithm for $(1\pm\varepsilon)$-approximating $\|x\|_1$, which was outlined in the beginning of Section 3. The full details of the algorithm are in Figure 2.

Before giving our algorithm and analysis, we define the ℓ_1 heavy hitters problem.

DEFINITION 4. *Let $0 < \gamma < \phi$ and $\delta > 0$ be given. In the ℓ_1 heavy hitters problem, with probability at least $1-\delta$ we must output a list $L \subseteq [n]$ such that*

1. *For all i with $|x_i| \ge \phi\|x\|_1$, $i \in L$.*
2. *For all $i \in L$, $|x_i| > (\phi-\gamma)\|x\|_1$.*
3. *For each $i \in L$, an estimate $\hat{x}_i$ is provided such that $|\hat{x}_i - x_i| < \gamma\|x\|_1$.*

Note that for $\gamma \le \phi/2$, the last two items above imply we can determine $\mathrm{sign}(x_i)$ for $i \in L$. For a generic algorithm solving the ℓ_1 heavy hitters problem, we use $\mathsf{HHUpdate}(\phi)$, $\mathsf{HHReport}(\phi)$, and $\mathsf{HHSpace}(\phi)$ to denote update time, reporting time, and space, respectively, with parameter ϕ and $\gamma = \phi/2$, $\delta = 1/20$.

There exist a couple of solutions to the ℓ_1 heavy hitters problem in the turnstile model. The work of [17] gives an algorithm with $\mathsf{HHSpace}(\phi) = O(\phi^{-1}\log(mM)\log(n))$, $\mathsf{HHUpdate}(\phi) = O(\log(n))$, and with $\mathsf{HHReport}(\phi) = O(n\log(n))$, and [27] gives an algorithm with

$$\mathsf{HHSpace}(\phi) = O(\phi^{-1}\log(\phi n)\log\log(\phi n)\log(1/\phi)\log(mM)),$$

and with $\mathsf{HHUpdate}(\phi) = O(\log(\phi n)\log\log(n)\log(1/\phi))$, and $\mathsf{HHReport}(\phi) = O(\phi^{-1}\log(\phi n)\log\log(\phi n)\log(1/\phi))$.

Also, the following theorem follows from Lemma 2.2 of [39] (with $k = 3$ in their notation). In Theorem 5 (and in Figure 2), the *Cauchy distribution* is a continuous probability distribution defined by its density function $\mu(x) = (\pi(1+x^2))^{-1}$. One can generate a Cauchy random variable X by setting $X = \tan(\pi U/2)$ for U a random variable uniform in $[0,1]$. Of course, to actually implement our algorithm (or that of Theorem 5) one can only afford to store these random variables to some finite precision; this is discussed in Remark 9.

THEOREM 5. *For an integer $n > 0$, let $A_1[j], \ldots, A_n[j]$ be $3n$ independent Cauchy random variables for $j = 1,2,3$. Let $x \in \mathbb{R}^n$ be arbitrary. Then given $C_j = \sum_{i=1}^n A_i[j]\cdot x_i$ for $j = 1,2,3$, the estimator*

$$\mathrm{Est}_{\mathrm{GM}} = \mathrm{Est}_{\mathrm{GM}}(C_1,C_2,C_3) = \frac{8\sqrt{3}}{9}\cdot\sqrt[3]{|C_1|\cdot|C_2|\cdot|C_3|}$$

satisfies the following two properties:

1. $\mathbf{E}[\mathrm{Est}_{\mathrm{GM}}] = \|x\|_1$.
2. $\mathbf{Var}[\mathrm{Est}_{\mathrm{GM}}] = \frac{19}{8}\cdot\|x\|_1^2$.

We show in Theorem 6 that our algorithm outputs $(1\pm O(\varepsilon))\|x\|_1$ with probability at least $3/5$. Note this error term can be made ε by running the algorithm with ε' being ε times a sufficiently small constant. Also, the success probability can be boosted to $1-\delta$ by running $O(\log(1/\delta))$ instantiations of the algorithm in parallel and returning the median output across all instantiations.

THEOREM 6. *The algorithm of Figure 2 outputs $(1\pm O(\varepsilon))\|x\|_1$ with probability at least $3/5$.*

PROOF. Throughout this proof we use A to denote the $3n$-tuple $(A_1[1],\ldots,A_n[1],\ldots,A_1[3],\ldots,A_n[3])$, and for $S \subseteq [n]$ we let $\mathcal{F}_S$ be the event that the hash family $\mathcal{H}$ we randomly select in Step 3 via Theorem 1 is $|S|$-wise independent when restricted to S. For an event $\mathcal{E}$, we let $\mathbf{1}_{\mathcal{E}}$ denote the indicator random variable for $\mathcal{E}$. We also use x_{head} to denote x projected onto the coordinates in L, and x_{tail} to denote the remaining coordinates. Note $\|x\|_1 = \|x_{\mathrm{head}}\|_1 + \|x_{\mathrm{tail}}\|_1$.

We first prove the following lemma. The proof requires some care since h is not always a uniform hash function on small sets, but is only so on any particular (small) set with large probability.

LEMMA 7. *Conditioned on the randomness of* HH *of Figure 2,*

$$\mathbf{E}_{A,h}\left[\frac{R}{|I|}\cdot\sum_{j\in I}\widetilde{L}_1(j)\right] = (1\pm O(\varepsilon))\|x_{\mathrm{tail}}\|_1.$$

1. Run an instantiation F of Filter, and HH of an ℓ_1 heavy hitters algorithm with $\phi = \varepsilon^2, \gamma = \phi/2, \delta = 1/20$.
2. Initialize $3R = 3 \cdot (4/\varepsilon^2)$ counters $B_1[j], \ldots, B_R[j]$ to 0 for $j = 1, 2, 3$.
3. Pick a random hash function $h : [n] \to [R]$ as in Theorem 1 with $z = R$ and $c = 2$.
4. Select independent Cauchy random variables $A_1[j], \ldots, A_n[j]$ for $j = 1, 2, 3$.
5. **Update(i, v):** Feed update (i, v) to both F and HH.
   $B_{h(i)}[j] \leftarrow B_{h(i)}[j] + A_i[j] \cdot v$ for $j = 1, 2, 3$.
6. **Estimator:** Let L be the list output by HH, and let $I = [R] \backslash h(L)$.
   Let Φ be the output of F, using the list L from the previous step.
   Let $\widetilde{L}_1(j) = \mathrm{Est}_{\mathrm{GM}}(B_j[1], B_j[2], B_j[3])$ as in Theorem 5.
   Output $\Phi + \frac{R}{|I|} \cdot \sum_{j \in I} \widetilde{L}_1(j)$.

Figure 2: Final algorithm for ℓ_1-estimation. Step 4 is derandomized in Section 4.

PROOF. For $\rho = 1 - \mathbf{Pr}[\mathcal{F}_L]$,

$$\begin{aligned}
&\mathbf{E}_{A,h}\left[\frac{R}{|I|} \cdot \sum_{j \in I} \widetilde{L}_1(j)\right] \\
&= \mathbf{E}_{A,h}\left[\frac{R}{|I|} \cdot \sum_{j \in I} \widetilde{L}_1(j) \;\middle|\; \mathcal{F}_L\right] \cdot \mathbf{Pr}[\mathcal{F}_L] \\
&\quad + \mathbf{E}_{A,h}\left[\frac{R}{|I|} \cdot \sum_{j \in I} \widetilde{L}_1(j) \;\middle|\; \neg\mathcal{F}_L\right] \cdot \mathbf{Pr}[\neg\mathcal{F}_L] \\
&= (1-\rho)\mathbf{E}_{A,h}\left[\frac{R}{|I|} \cdot \sum_{j \in I} \widetilde{L}_1(j) \;\middle|\; \mathcal{F}_L\right] \\
&\quad + \rho \cdot \mathbf{E}_A\left[\mathbf{E}_h\left[\frac{R}{|I|} \cdot \sum_{j \in I} \widetilde{L}_1(j) \;\middle|\; \neg\mathcal{F}_L\right]\right] \\
&= (1-\rho)\mathbf{E}_{A,h}\left[\frac{R}{|I|} \cdot \sum_{j \in I} \widetilde{L}_1(j) \;\middle|\; \mathcal{F}_L\right] \pm (\rho \cdot R)\|x_{\mathrm{tail}}\|_1
\end{aligned} \tag{3}$$

by Theorem 1 and Theorem 5. We now compute the above expectation conditioned on I. Let $\mathcal{E}_{I'}$ be the event $I = I'$ for an arbitrary I'. Then,

$$\begin{aligned}
&\mathbf{E}_{A,h}\left[\frac{R}{|I'|} \cdot \sum_{j \in I'} \widetilde{L}_1(j) \;\middle|\; \mathcal{F}_L, \mathcal{E}_{I'}\right] \\
&= \frac{R}{|I'|} \cdot \sum_{j \in I'} \mathbf{E}_{A,h}\left[\widetilde{L}_1(j) \;\middle|\; \mathcal{F}_L, \mathcal{E}_{I'}\right] \\
&= \frac{R}{|I'|} \cdot \sum_{j \in I'} \mathbf{E}_h\left[\sum_{i \notin L} \mathbf{1}_{h(i)=j} \cdot |x|_i \;\middle|\; \mathcal{F}_L, \mathcal{E}_{I'}\right] \\
&= \frac{R}{|I'|} \cdot \sum_{j \in I'} \sum_{i \notin L} |x|_i \cdot \mathbf{Pr}_h\left[h(i) = j \mid \mathcal{F}_L, \mathcal{E}_{I'}\right] \\
&= \frac{R}{|I'|} \cdot \sum_{j \in I'} \sum_{i \notin L} |x|_i \cdot \frac{\mathbf{Pr}_h\left[(h(i) = j) \wedge \mathcal{E}_{I'} \mid \mathcal{F}_L\right]}{\mathbf{Pr}[\mathcal{E}_{I'} \mid \mathcal{F}_L]}
\end{aligned} \tag{4}$$

Now we note

$$\begin{aligned}
&\mathbf{Pr}_h\left[(h(i) = j) \wedge \mathcal{E}_{I'} \mid \mathcal{F}_L\right] \\
&= \mathbf{Pr}_h\left[(h(i) = j) \wedge \mathcal{E}_{I'} \mid \mathcal{F}_{L\cup\{i\}}, \mathcal{F}_L\right] \cdot \mathbf{Pr}[\mathcal{F}_{L\cup\{i\}} \mid \mathcal{F}_L] \\
&\quad + \mathbf{Pr}_h\left[(h(i) = j) \wedge \mathcal{E}_{I'} \mid \neg\mathcal{F}_{L\cup\{i\}}, \mathcal{F}_L\right] \cdot \mathbf{Pr}[\neg\mathcal{F}_{L\cup\{i\}} \mid \mathcal{F}_L] \\
&= \mathbf{Pr}_h\left[h(i) = j \mid \mathcal{F}_{L\cup\{i\}}, \mathcal{F}_L\right] \cdot \mathbf{Pr}_h\left[\mathcal{E}_{I'} \mid \mathcal{F}_L, \mathcal{F}_{L\cup\{i\}}, h(i) = j\right] \\
&\qquad \times \mathbf{Pr}[\mathcal{F}_{L\cup\{i\}} \mid \mathcal{F}_L] \\
&\quad + \mathbf{Pr}[\neg\mathcal{F}_{L\cup\{i\}} \mid \mathcal{F}_L] \cdot \mathbf{Pr}[\mathcal{E}_{I'} \mid \mathcal{F}_L, \neg\mathcal{F}_{L\cup\{i\}}] \\
&\qquad \times \mathbf{Pr}[h(i) = j \mid \neg\mathcal{F}_{L\cup\{i\}}, \mathcal{F}_L, \mathcal{E}_{I'}]
\end{aligned}$$

Now note a couple things. First, if $\mathcal{F}_{L\cup\{i\}}$ occurs, then $\mathcal{E}_{I'}$ is independent of the event $h(i) = j$. Also, if $\mathcal{F}_L$ occurs, then $\mathcal{E}_{I'}$ is independent of $\mathcal{F}_{L\cup\{i\}}$. Thus, the above equals

$$\begin{aligned}
&\mathbf{Pr}_h\left[h(i) = j \mid \mathcal{F}_{L\cup\{i\}}\right] \cdot \mathbf{Pr}_h\left[\mathcal{E}_{I'} \mid \mathcal{F}_L\right] \cdot \mathbf{Pr}[\mathcal{F}_{L\cup\{i\}} \mid \mathcal{F}_L] \\
&\quad + \mathbf{Pr}[\neg\mathcal{F}_{L\cup\{i\}} \mid \mathcal{F}_L] \cdot \mathbf{Pr}[\mathcal{E}_{I'} \mid \mathcal{F}_L] \\
&\qquad \times \mathbf{Pr}[h(i) = j \mid \neg\mathcal{F}_{L\cup\{i\}}, \mathcal{F}_L, \mathcal{E}_{I'}]
\end{aligned}$$

Note $\mathbf{Pr}[\neg\mathcal{F}_{L\cup\{i\}} \mid \mathcal{F}_L] \leq \mathbf{Pr}[\neg\mathcal{F}_{L\cup\{i\}}]/\mathbf{Pr}[\mathcal{F}_L] = \rho'_i/(1-\rho)$ for $\rho'_i = 1 - \mathbf{Pr}[\mathcal{F}_{L\cup\{i\}}]$. Also, $\mathbf{Pr}[\mathcal{F}_{L\cup\{i\}} \mid \mathcal{F}_L] \geq \mathbf{Pr}[\mathcal{F}_{L\cup\{i\}}]$ since $\mathbf{Pr}[\mathcal{F}_{L\cup\{i\}}]$ is a weighted average of $\mathbf{Pr}[\mathcal{F}_{L\cup\{i\}} \mid \mathcal{F}_L]$ and $\mathbf{Pr}[\mathcal{F}_{L\cup\{i\}} \mid \neg\mathcal{F}_L]$, and the latter is 0. Thus, for some $\rho''_i \in [0, \rho'_i]$ Eq. (4) is

$$\begin{aligned}
&\frac{R}{|I'|} \cdot \sum_{j \in I'} \sum_{i \notin L} |x|_i \cdot \left(\frac{1 - \rho''_i}{R} \pm \frac{\rho'_i}{1-\rho}\right) \\
&\qquad = \|x_{\mathrm{tail}}\|_1 - \sum_{i \notin L} \rho''_i |x|_i \pm \left(\frac{\max_i \rho'_i}{1-\rho}\right) \cdot R \cdot \|x_{\mathrm{tail}}\|_1
\end{aligned}$$

By our setting of $c = 2$ when picking the hash family of Theorem 1 in Step 3, we have $\rho, \rho'_i, \rho''_i = O(\varepsilon^3)$ for all i, and thus $\rho'_i/(1-\rho) \cdot R = O(\varepsilon)$, implying the above is $(1 \pm O(\varepsilon))\|x_{\mathrm{tail}}\|_1$. Plugging this in Eq. (3) then shows that the desired expectation is $(1 \pm O(\varepsilon))\|x_{\mathrm{tail}}\|_1$. $\square$

We now bound the expected variance of $(R/|I|) \cdot \sum_{j \in I} \widetilde{L}_1(j)$.

LEMMA 8. *Conditioned on* HH *being correct,*

$$\mathbf{E}_h\left[\mathbf{Var}_A\left[\frac{R}{|I|} \cdot \sum_{j \in I} \widetilde{L}_1(j)\right]\right] = O(\varepsilon^2 \cdot \|x\|_1^2).$$

PROOF. For any fixed h, $R/|I|$ is determined, and the

$\widetilde{L}_1(j)$ are pairwise independent. Thus, for fixed h,

$$\mathbf{Var}_A\left[\frac{R}{|I|}\cdot\sum_{j\in I}\widetilde{L}_1(j)\right]=\left(\frac{R}{|I|}\right)^2\cdot\sum_{j\in I}\mathbf{Var}_A[\widetilde{L}_1(j)]$$

First observe that since $|I|\ge R-|L|\ge 2/\varepsilon^2$, for any choice of h we have that $R/|I|\le 2$. Thus, up to a constant factor, the expectation we are attempting to compute is

$$\mathbf{E}_h\left[\mathbf{Var}_A\left[\sum_{j\in I}\widetilde{L}_1(j)\right]\right].$$

For notational convenience, say $\widetilde{L}_1(j)=0$ if $j\notin I$. Now,

$$\begin{aligned}\mathbf{E}_h\left[\mathbf{Var}_A\left[\sum_{j\in I}\widetilde{L}_1(j)\right]\right]&=\mathbf{E}_h\left[\sum_{j=1}^{R}\mathbf{1}_{j\in I}\cdot\mathbf{Var}_A\left[\widetilde{L}_1(j)\right]\right]\\&=\sum_{j=1}^{R}\mathbf{E}_h\left[\mathbf{1}_{j\in I}\cdot\mathbf{Var}_A\left[\widetilde{L}_1(j)\right]\right]\\&\le\sum_{j=1}^{R}\mathbf{E}_h\left[\mathbf{Var}_A\left[\widetilde{L}_1(j)\right]\ \middle|\ j\in I\right]\\&=\sum_{j=1}^{R}\mathbf{E}_h\left[\frac{19}{8}\cdot\left(\sum_{i\notin L}\mathbf{1}_{h(i)=j}\cdot|x_i|\right)^2\ \middle|\ j\in I\right],\end{aligned}$$

which equals

$$\frac{19}{8}\cdot\left(\sum_{j=1}^{R}\sum_{i\notin L}|x_i|^2\cdot\mathbf{Pr}_h[h(i)=j\mid j\in I]+\sum_{j=1}^{R}\sum_{\substack{i\neq i'\\ i,i'\notin L}}|x_i||x_{i'}|\cdot\mathbf{Pr}_h[(h(i)=j)\wedge(h(i')=j)\mid j\in I]\right)\tag{5}$$

Now consider the quantity $\mathbf{Pr}_h[h(i)=j\mid j\in I]$. Then

$$\begin{aligned}&\mathbf{Pr}_h[h(i)=j\mid j\in I]\\&\quad=\mathbf{Pr}_h[h(i)=j\mid\mathcal{F}_{L\cup\{i\}},j\in I]\cdot\mathbf{Pr}_h[\mathcal{F}_{L\cup\{i\}}\mid j\in I]\\&\qquad+\mathbf{Pr}_h[h(i)=j\mid\neg\mathcal{F}_{L\cup\{i\}},j\in I]\\&\qquad\quad\times\mathbf{Pr}_h[\neg\mathcal{F}_{L\cup\{i\}}\mid j\in I]\\&\quad\le\mathbf{Pr}_h[h(i)=j\mid\mathcal{F}_{L\cup\{i\}},j\in I]+\mathbf{Pr}_h[\neg\mathcal{F}_{L\cup\{i\}}\mid j\in I]\\&\quad=\frac{1}{R}+\mathbf{Pr}_h[\neg\mathcal{F}_{L\cup\{i\}}\mid j\in I]\end{aligned}$$

Then by Bayes' theorem, the above is at most

$$\begin{aligned}\frac{1}{R}+\frac{\mathbf{Pr}_h[\neg\mathcal{F}_{L\cup\{i\}}]}{\mathbf{Pr}_h[j\in I]}&\le\frac{1}{R}+\frac{\mathbf{Pr}_h[\neg\mathcal{F}_{L\cup\{i\}}]}{\mathbf{Pr}_h[j\in I\mid\mathcal{F}_L]\cdot\mathbf{Pr}[\mathcal{F}_L]}\\&=\frac{1}{R}+\frac{\mathbf{Pr}_h[\neg\mathcal{F}_{L\cup\{i\}}]}{(1-\frac{1}{R})^{|L|}\cdot\mathbf{Pr}[\mathcal{F}_L]}\\&\le\frac{1}{R}+\frac{\mathbf{Pr}_h[\neg\mathcal{F}_{L\cup\{i\}}]}{\left(1-\frac{|L|}{R}\right)\cdot\mathbf{Pr}[\mathcal{F}_L]}\end{aligned}$$

Note $|L|/R\le 1/2$. Also, by choice of c,z in the application of Theorem 1 in Step 3, $\mathbf{Pr}[\mathcal{F}_L]=1-O(\varepsilon)$ and $\mathbf{Pr}[\neg\mathcal{F}_{L\cup\{i\}}]=O(1/R^2)$. Thus overall,

$$\mathbf{Pr}_h[h(i)=j\mid j\in I]=O(1/R).$$

An essentially identical calculation, but conditioning on $\mathcal{F}_{L\cup\{i,i'\}}$ instead of $\mathcal{F}_{L\cup\{i\}}$, gives that

$$\mathbf{Pr}_h[(h(i)=j)\wedge(h(i')=j)\mid j\in I]=O(1/R^2).$$

Combining these bounds with Eq. (5), we have that the expected variance we are aiming to compute is

$$O(\|x_{\mathrm{tail}}\|_2^2+\|x_{\mathrm{tail}}\|_1^2/R).$$

The second summand is $O(\varepsilon^2\|x\|_1^2)$. For the first summand, conditioned on HH being correct, every $|x_i|$ for $i\notin L$ has $|x_i|\le\varepsilon^2\|x\|_1$. Under this constraint, $\|x_{\mathrm{tail}}\|_2^2$ is maximized when there are exactly $1/\varepsilon^2$ coordinates $i\notin L$ each with $|x_i|=\varepsilon^2\|x\|_1$, in which case $\|x_{\mathrm{tail}}\|_2^2=\varepsilon^2\|x\|_1^2$. □

We now wrap up the proof of correctness of our full algorithm in Figure 2 as follows. We condition on the event $\mathcal{E}_{\mathsf{HH}}$ that HH succeeds, i.e. satisfies the three conditions of Definition 4. Given this, we then condition on the event $\mathcal{E}_{\mathsf{F}}$ that F succeeds as defined by Theorem 3, i.e. that

$$\Phi=\|x_{\mathrm{head}}\|_1\pm\varepsilon\|x\|_1.$$

Next we look at the quantity

$$X=\frac{R}{|I|}\cdot\sum_{j\in I}\widetilde{L}_1(j).$$

By Lemma 7, $\mathbf{E}[X]$, even conditioned on the randomness used by HH to determine L, is $(1\pm O(\varepsilon))\|x_{\mathrm{tail}}\|_1$. Also, conditioned on $\mathcal{E}_{\mathsf{HH}}$, the expected value of $\mathbf{Var}[X]$ for a random h is $O(\varepsilon^2\|x\|_1^2)$. Since $\mathbf{Var}[X]$ is always nonnegative, Markov's bound applies, and we have that $\mathbf{Var}[X]=O(\varepsilon^2\|x\|_1^2)$ with probability at least 19/20 (over the randomness in selecting h). Thus, by Chebyshev's inequality,

$$\mathbf{Pr}_{A,h}[|X-\mathbf{E}[X]|>t\varepsilon\|x\|_1\mid\mathcal{E}_{\mathsf{HH}}]<\frac{1}{20}+O(1/t^2),\tag{6}$$

which can be made at most 1/15 by setting t a sufficiently large constant. Call the event in Eq. (6) $\mathcal{F}$. Then, as long as $\mathcal{E}_{\mathsf{HH}},\mathcal{E}_{\mathsf{F}},\mathcal{F}$ occur, we have that our final estimate of $\|x\|_1$ is

$$(1+O(\varepsilon))\|x_{\mathrm{tail}}\|_1+\|x_{\mathrm{head}}\|_1\pm O(\varepsilon\|x\|_1)=(1\pm O(\varepsilon))\|x\|_1$$

as desired. Our probability of correctness is then at least

$$\begin{aligned}\mathbf{Pr}[\mathcal{E}_{\mathsf{HH}}\wedge\mathcal{E}_{\mathsf{F}}\wedge\mathcal{F}]&\ge\mathbf{Pr}[\mathcal{E}_{\mathsf{F}}\wedge\mathcal{F}\mid\mathcal{E}_{\mathsf{HH}}]\cdot\mathbf{Pr}[\mathcal{E}_{\mathsf{HH}}]\\&=\mathbf{Pr}[\mathcal{E}_{\mathsf{F}}\mid\mathcal{E}_{\mathsf{HH}}]\cdot\mathbf{Pr}[\mathcal{F}\mid\mathcal{E}_{\mathsf{HH}}]\cdot\mathbf{Pr}[\mathcal{E}_{\mathsf{HH}}]\\&\ge\frac{7}{10}\cdot\frac{14}{15}\cdot\frac{19}{20}\\&>3/5\end{aligned}$$

REMARK 9. *It is known from previous work (see [31, Claim 3]) that each $A_i[j]$ can be be maintained up to only $O(\log(n/\varepsilon))$ bits of precision, and requires the same amount of randomness to generate, to preserve the probability of correctness to within an arbitrarily small constant. Then, note that the counters $B_i[j]$ each only consume $O(\log(nmM/\varepsilon))$ bits of storage.*

Given Remark 9, we have the following theorem.

THEOREM 10. *Ignoring the space to store the $A_i[j]$, the algorithm of Figure 2 requires space $O((\varepsilon^{-2}\log(nmM/\varepsilon)+\log\log(n))\log(1/\varepsilon))+\mathsf{HHSpace}(\varepsilon^2)$. The update and reporting times are, respectively, $O(\log(1/\varepsilon))+\mathsf{HHUpdate}(\varepsilon^2)$, and $O(\varepsilon^{-2}\log(1/\varepsilon))+\mathsf{HHReport}(\varepsilon^2)$.*

PROOF. Ignoring F and HH, the update time is $O(1)$ to compute h, and $O(1)$ to update the corresponding $B_{h(i)}$. Also ignoring F and HH, the space required is $O(\varepsilon^{-2}\log(nmM/\varepsilon))$ to store all the $B_i[j]$ (Remark 9), and $O(\varepsilon^{-2}\log(1/\varepsilon) + \log\log(n))$ bits to store h and randomly select the hash family it comes from (Theorem 1). The time to compute the final line in the estimator, given L and ignoring the time to compute Φ, is $O(1/\varepsilon^2)$.

The bounds stated above then take into account the complexities of F and HH. $\square$

4. DERANDOMIZING THE FINAL ALGORITHM

Observe that a naïve implementation of storing the entire tuple A in Figure 2 requires $\Omega(n\log(n/\varepsilon))$ bits. Considering our goal is to have a small-space algorithm, this is clearly not affordable.

As it turns out, using a now standard technique in streaming algorithms, originally pioneered by Indyk [31], one can avoid storing the tuple A explicitly. This is accomplished by generating A from a short, truly random seed which is then stretched out by a pseudorandom generator against space-bounded computation such as Nisan's PRG [46]. In Indyk's original argument, he used Nisan's PRG to show that his entire algorithm was fooled by using the PRG to stretch a short seed of length $O(\varepsilon^{-2}\log(n/\varepsilon)\log(nmM/\varepsilon))$ to generate $\Theta(n/\varepsilon^2)$ Cauchy random variables. However, for fooling his algorithm, this derandomization step used $\Omega(1/\varepsilon^2)$ time during each stream update to generate the necessary Cauchy random variables from the seed. Given that our goal is to have fast update time, we cannot afford this. We show that to derandomize our algorithm, Nisan's PRG can be applied in such a way that the time to apply the PRG to the seed to retrieve any $A_i[j]$ is small. We now give the details.

First, recall the definition of a finite state machine (FSM). An FSM M is parametrized by a tuple $(T_{\rm init}, S, \Gamma, n)$. The FSM M is always in some "state", which is just a string $x \in \{0,1\}^S$, and it starts in the state $T_{\rm init}$. The parameter Γ is a function mapping $\{0,1\}^S \times \{0,1\}^n \to \{0,1\}^S$. We also abuse notation and for $x \in (\{0,1\}^n)^r$ for r a positive integer, we use $\Gamma(T,x)$ to denote $\Gamma(\ldots(\Gamma(\Gamma(T,x_1),x_2),\ldots),x_r)$. Note that given a distribution $\mathcal{D}$ over $(\{0,1\}^n)^r$, there is an implied distribution $M(\mathcal{D})$ over $\{0,1\}^S$ obtained as $\Gamma(T_{\rm init},\mathcal{D})$.

DEFINITION 11. *Let t be a positive integer. For $\mathcal{D},\mathcal{D}'$ two distributions on $\{0,1\}^t$, we define the* total variation distance *$\Delta(\mathcal{D},\mathcal{D}')$ by*

$$\Delta(\mathcal{D},\mathcal{D}') = \max_{T\subset\{0,1\}^t} |\mathbf{Pr}_{X\leftarrow\mathcal{D}}[X\in T] - \mathbf{Pr}_{Y\leftarrow\mathcal{D}'}[Y\in T]|.$$

THEOREM 12 (NISAN [46]). *Let U^t denote the uniform distribution on $\{0,1\}^t$. For any positive integers r, n, and for some $S = \Theta(n)$, there exists a function $G_{\rm nisan} : \{0,1\}^s \to (\{0,1\}^n)^r$ with $s = O(S\log(r))$ such that for* any *FSM $M = (T_{\rm init}, S, \Gamma, n)$,*

$$\Delta(M((U^n)^r), M(G_{\rm nisan}(U^s))) \le 2^{-S}.$$

Furthermore, for any $x \in \{0,1\}^s$ and $i \in [r]$, computing the n-bit block $G_{\rm nisan}(x)_i$ requires $O(S\log(r))$ space and $O(\log(r))$ arithmetic operations on $O(S)$-bit words.

Before finally describing how Theorem 12 fits into a derandomization of Figure 2, we state the following standard lemma (see for example [53, Lemma 5.3]).

LEMMA 13. *If $X_1,\ldots,X_m$ are independent and $Y_1,\ldots,Y_m$ are independent, then*

$$\Delta(X_1\times\cdots\times X_m, Y_1\times\cdots\times Y_m) \le \sum_{i=1}^m \Delta(X_i,Y_i).$$

Now, the derandomization of Figure 2 is as follows. Condition on all the randomness in Figure 2 except for A. Recall we have $R = \Theta(1/\varepsilon^2)$ "buckets" B_u. Each bucket contains three counters, which is a sum of at most n Cauchy random variables, each weighted by at most mM. Given the precision required to store A (Remark 9), the three counters in B_u in total consume $S' = O(\log(nmM/\varepsilon))$ bits of space. Consider the FSM M_u which has 2^S states for $S = S' + \log(n)$, representing the state of the three counters together with an index $i_{\rm cur} \in [n]$ that starts at 0. Define t as the number of uniform random bits required to generate each $A_i[j]$, so that $t = O(\log(nmM/\varepsilon))$ by Remark 9. Note $t = \Theta(S)$. Consider the transition function $\Gamma : \{0,1\}^{3t} \to \{0,1\}^S$ defined as follows: upon being fed $(A_i[1], A_i[2], A_i[3])$ (or more precisely, the $3t$ uniform random bits used to generate this tuple), increment $i_{\rm cur}$ then add $A_i[j]\cdot x_i$ to each $B_u[j]$, for i being the $(i_{\rm cur})$th index $i \in [n]$ such that $h(i) = u$. Now, note that if one feeds the $(A_i[1], A_i[2], A_i[3])$ for which $h(i) = u$ to M_u, sorted by i, then the state of M_u corresponds exactly to the state of bucket B_u in our algorithm.

By Theorem 12, if rather than defining A by $3tr$ truly random bits (for $r = n$) we instead define it by stretching a seed of length $s = O(S\log(n)) = O(\log(nmM/\varepsilon)\log(n))$ via $G_{\rm nisan}$, we have that the distribution on the state of B_u at the end of the stream changes by at most a total variation distance of 2^{-S}. Now, suppose we use R independent seeds to generate different A vectors in each of the R buckets. Note that since each index $i \in [n]$ is hashed to exactly one bucket, the $A_i[j]$ across each bucket need not be consistent to preserve the behavior of our algorithm. Then for U^t being the uniform distribution on $\{0,1\}^t$, we have

$$\Delta\Big(M_1(U^{3t})^r \times\cdots\times M_R(U^{3t})^r,\\ M_1(G_{\rm nisan}(U^s))\times\cdots\times M_R(G_{\rm nisan}(U^s))\Big) \le R\cdot 2^{-S}$$

by Lemma 13. By increasing S by a constant factor, we can ensure $R\cdot 2^{-S}$ is an arbitrarily small constant δ. Now, note that the product measure on the output distributions of the M_u corresponds exactly to the state of our entire algorithm at the end of the stream. Thus, if we consider T to be the set of states $(B_1,\ldots,B_R)$ for which our algorithm outputs a value $(1\pm\varepsilon)\|x\|_1$ (i.e., is correct), by definition of total variation distance (Definition 11), we have that the probability of correctness of our algorithm changes by at most an additive δ when using Nisan's PRG instead of uniform randomness. Noting that storing R independent seeds just takes Rs space, and that the time required to extract any $A_i[j]$ from a seed requires $O(\log(n))$ time by Theorem 12, we have the following theorem.

THEOREM 14. *Including the space and time complexities of storing and accessing the $A_i[j]$, the algorithm of Figure 2 can be implemented with an additive $O(\varepsilon^{-2}\log(nmM/\varepsilon)\log(n))$ increase to the space, additive $O(\log(n))$ increase to the update time, and no change to the reporting time, compared with the bounds given in Theorem 10.*

5. REFERENCES

[1] Charu C. Aggarwal, Alexander Hinneburg, and Daniel A. Keim. On the surprising behavior of distance metrics in high dimensional spaces. In *ICDT*, pages 420–434, 2001.

[2] Rakesh Agrawal, King-Ip Lin, Harpreet S. Sawhney, and Kyuseok Shim. Fast similarity search in the presence of noise, scaling, and translation in time-series databases. In *VLDB*, pages 490–501, 1995.

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

[4] Radu Berinde, Graham Cormode, Piotr Indyk, and Martin J. Strauss. Space-optimal heavy hitters with strong error bounds. In *PODS*, pages 157–166, 2009.

[5] Kevin S. Beyer and Raghu Ramakrishnan. Bottom-up computation of sparse and iceberg cubes. In *SIGMOD Conference*, pages 359–370, 1999.

[6] Emmanuel J. Candès, Justin Romberg, and Terence Tao. Stable signal recovery from incomplete and inaccurate measurements. *Communications on Pure and Applied Mathematics*, 59(8), 2006.

[7] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *Proceedings of the 29th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 693–703, 2002.

[8] Surajit Chaudhuri, Rajeev Motwani, and Vivek R. Narasayya. On random sampling over joins. In *SIGMOD Conference*, pages 263–274, 1999.

[9] Cisco NetFlow. `http://www.cisco.com/go/netflow`.

[10] Kenneth L. Clarkson. Subgradient and sampling algorithms for l1 regression. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2005.

[11] Edith Cohen, Nick G. Duffield, Haim Kaplan, Carsten Lund, and Mikkel Thorup. Algorithms and estimators for accurate summarization of internet traffic. In *Internet Measurement Comference*, pages 265–278, 2007.

[12] Graham Cormode, Mayur Datar, Piotr Indyk, and S. Muthukrishnan. Comparing data streams using hamming norms (how to zero in). *IEEE Trans. Knowl. Data Eng.*, 15(3):529–540, 2003.

[13] Graham Cormode and Minos N. Garofalakis. Sketching streams through the net: Distributed approximate query tracking. In *VLDB*, pages 13–24, 2005.

[14] Graham Cormode, Piotr Indyk, Nick Koudas, and S. Muthukrishnan. Fast mining of massive tabular data via approximate distance computations. In *ICDE*, pages 605–, 2002.

[15] Graham Cormode, Flip Korn, S. Muthukrishnan, and Divesh Srivastava. Space- and time-efficient deterministic algorithms for biased quantiles over data streams. In *PODS*, pages 263–272, 2006.

[16] Graham Cormode, Flip Korn, and Srikanta Tirthapura. Time-decaying aggregates in out-of-order streams. In *PODS*, pages 89–98, 2008.

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

[18] Graham Cormode and S. Muthukrishnan. Space efficient mining of multigraph streams. In *Proceedings of the 24th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 271–282, 2005.

[19] Graham Cormode and S. Muthukrishnan. What's hot and what's not: tracking most frequent items dynamically. *ACM Trans. Database Syst.*, 30(1):249–278, 2005.

[20] Graham Cormode, S. Muthukrishnan, and Irina Rozenbaum. Summarizing and mining inverse distributions on data streams via dynamic inverse sampling. In *VLDB*, pages 25–36, 2005.

[21] Yadolah Dodge. *L1-Statistical Procedures and Related Topics*. Institute for Mathematical Statistics, 1997.

[22] Min Fang, Narayanan Shivakumar, Hector Garcia-Molina, Rajeev Motwani, and Jeffrey D. Ullman. Computing iceberg queries efficiently. In *VLDB*, pages 299–310, 1998.

[23] 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.

[24] Dan Feldman, Morteza Monemizadeh, Christian Sohler, and David P. Woodruff. Coresets and sketches for high dimensional subspace problems. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), to appear*, 2010.

[25] Sumit Ganguly. personal communication.

[26] 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.

[27] Sumit Ganguly, Abhayendra N. Singh, and Satyam Shankar. Finding frequent items over general update streams. In *Proceedings of the 20th International Conference on Scientific and Statistical Database Management (SSDBM)*, pages 204–221, 2008.

[28] Anna C. Gilbert, Martin J. Strauss, Joel A. Tropp, and Roman Vershynin. One sketch for all: fast algorithms for compressed sensing. In *STOC*, pages 237–246, 2007.

[29] Jiawei Han, Jian Pei, Guozhu Dong, and Ke Wang. Efficient computation of iceberg cubes with complex measures. In *SIGMOD Conference*, pages 1–12, 2001.

[30] John Hershberger, Nisheeth Shrivastava, Subhash Suri, and Csaba D. Tóth. Space complexity of hierarchical heavy hitters in multi-dimensional data streams. In *Proceedings of the 24th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 338–347, 2005.

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

[32] Piotr Indyk and Andrew McGregor. Declaring independence via the sketching of sketches. In *SODA*, pages 737–745, 2008.

[33] Piotr Indyk and David P. Woodruff. Polylogarithmic private approximations and efficient matching. In *TCC*, pages 245–264, 2006.

[34] T. S. Jayram and David P. Woodruff. The data stream space complexity of cascaded norms. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 765–774, 2009.

[35] Daniel M. Kane, Jelani Nelson, and David P. Woodruff. On the exact space complexity of sketching and streaming small norms. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1161–1178, 2010.

[36] Khaled Labib and V. Rao Vemuri. A hardware-based clustering approach for anomaly detection, 2006.

[37] Wing Cheong Lau, Murali S. Kodialam, T. V. Lakshman, and H. Jonathan Chao. Datalite: a distributed architecture for traffic analysis via light-weight traffic digest. In *BROADNETS*, pages 622–630, 2007.

[38] Kenneth D. Lawrence and Jeffrey L. Arthur. *Robust Regression*. Dekker, 1990.

[39] Ping Li. Estimators and tail bounds for dimension reduction in l_p $(0 < p \leq 2)$ using stable random projections. In *Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 10–19, 2008.

[40] Ping Li, Trevor Hastie, and Kenneth Ward Church. Very sparse random projections. In *KDD*, pages 287–296, 2006.

[41] Hendrik P. Lopuhaä and Peter J. Rousseeuw. Breakdown points of affine equivalent estimators of multivarite location and covariance matrices. *Annals of Statistics*, 19(1):229–248, 1991.

[42] Andre Madeira and S. Muthukrishnan. Functionally private approximation for negligibly-biased estimators. In *Proceedings of the 29th International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, 2009.

[43] Morteza Monemizadeh and David P. Woodruff. 1-pass relative-error l_p sampling with applications. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), to appear*, 2010.

[44] Jelani Nelson and David P. Woodruff. A near-optimal algorithm for L1-difference. *CoRR*, abs/0904.2027, 2009.

[45] Jiawang Nie, Pablo A. Parillo, and Bernd Sturmfels. Semidefinite representation of the k-ellipse. *Algorithms in Algebraic Geometry, IMA Volumes in Mathematics and its Applications*, 146:117–132, 2008.

[46] Noam Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, 1992.

[47] 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`.

[48] Anna Pagh and Rasmus Pagh. Uniform hashing in constant time and linear space. *SIAM J. Comput.*, 38(1):85–96, 2008.

[49] Peter J. Rousseeuw and Annick M. Lerow. *Robust Regression and Outlier Detection*. John Wiley, 1987.

[50] Robert T. Schweller, Zhichun Li, Yan Chen, Yan Gao, Ashish Gupta, Yin Zhang, Peter A. Dinda, Ming-Yang Kao, and Gokhan Memik. Reversible sketches: enabling monitoring and analysis over high-speed data streams. *IEEE/ACM Trans. Netw.*, 15(5):1059–1072, 2007.

[51] Nicholas D. Sidiropoulos and Rasmus Bro. Mathematical programming algorithms for regression-based non-linear filtering in $\mathbb{R}^n$. *IEEE Transactions on Signal Processing*, pages 771–782, 1999.

[52] Mikkel Thorup and Yin Zhang. Tabulation based 4-universal hashing with applications to second moment estimation. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 615–624, 2004.

[53] Salil P. Vadhan. Pseudorandomness II. Manuscript. `http://people.seas.harvard.edu/~salil/cs225/spring09/lecnotes/FnTTCS-vol2.pdf`.

[54] Endre V. Weiszfeld. Sur le point pour lequel la somme des distances de n points donnes est minimum. *Tohoku Math*, 43:355–386, 1937.

[55] David P. Woodruff. Optimal space lower bounds for all frequency moments. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 167–175, 2004.

[56] David P. Woodruff. Private approximations from sampling in a data stream. Manuscript, 2010.

[57] Byoung-Kee Yi and Christos Faloutsos. Fast time sequence indexing for arbitrary Lp norms. In *VLDB*, pages 385–394, 2000.