

Streaming Algorithms for Data in Motion

M. Hoffmann¹, S. Muthukrishnan^{2*}, and Rajeev Raman¹

¹ Department of Computer Science, University of Leicester, Leicester LE1 7RH, UK.
{m.hoffmann,r.raman}@cs.le.ac.uk

² Division of Computer and Information Sciences, Rutgers University, Piscataway,
NJ 08854-8019, USA. muthu@cs.rutgers.edu

Abstract. We propose two new data stream models: the *reset* model and the *delta* model, motivated by applications to databases, and to tracking the location of spatial points.

We present algorithms for several problems that fit within the stream constraint of polylogarithmic space and time. These include tracking the “extent” of the points and L_p sampling.

1 Introduction

The area of data stream algorithms is the study of algorithmic problems in applications where the data arrives at extremely high speed. Further, the memory available is quite small, so the entire data can not be stored. Thus algorithms have to solve data analyses problems to the best extent they can with sublinear—often polylogarithmic—space, and fast per-item processing. Under these constraints, computing most significant functions on the data stream is provably impossible. So, algorithms typically tend to produce an approximation and are often randomized. Such algorithms have been developed for a number of data analyses problems including estimation of various norms [8, 28], clustering [23, 11, 24], histograms [21], quantiles [22], heavy-hitters [33] etc.

The quintessential application where this arises is in processing internet traffic logs such as the headers on IP packets and IP sessions. Since packets are forwarded at blistering speeds in the internet, these traffic logs are generated at very high speed; memory working at this speed within the routing elements is quite scarce, and the logs have to be processed in real time to solve number of queries involved in monitoring the internet for security breaches, accounting, billing etc. This application has become fairly well developed, with tutorials and a number of research papers in networking conferences [38, 16, 32], specialized database systems aimed at such applications discussed in workshops, tutorials and a number of papers in database conferences [1, 37, 9], and fundamental algorithmic and complexity-theoretic issues being studied in the theoretical computer science community [8, 6, 35].

Recently, there is growing interest in *other* data streams. In particular, an emerging area is one of *spatial* streams. These are location-specific streams. For

* Supported by NSF EIQ 0087022, NSF ITR 0220280 and NSF EIA 02-05116.

example, consider a vast collection of moving objects. Over time, their locations change and one wants to track various functions of their locations. GPS (Global Positioning System) enables tracking locations of devices over large scale. Wireless service providers are looking to use this or other infrastructure to enable location-aware services, which is a burgeoning industry [3]. Other industries are actively engaged in developing applications based on tracking the location of various objects: delivery trucks (eg. [2]), University campus buses [4], cars, stocks of goods, biosensors [5], astronomical studies via satellite and ground-based systems [36], etc.

We propose two new models. Both models have relevance to existing scenarios (e.g. databases) as well as spatial streams. We study a fundamental problems, e.g., accurately estimating the “extent” of moving objects, under both models and design space- and time-efficient algorithms for these problems. As in the study of standard data streams [9, 35], exact calculation of most functions is impossible in our new model, and algorithms tend to be probabilistic and approximate. Our algorithms rely on new versions of techniques used commonly in streaming problems.

1.1 The reset model.

The first model is the *reset* model, which is as follows. There is a vector $\mathbf{d} = (d_1, \dots, d_n)$ in \mathfrak{R}^n , where n is assumed to be very large. We assume that $d_i = 0$ for $i = 1, \dots, n$ initially, and consider the following model of updates:

Reset model: The updates appear in a stream (i, x) , implying that $d_i \leftarrow x$.

For spatial data, each element of the vector may be a point in \mathfrak{R}^d , and an update may reset one or more coordinates of a given point.

Remark 1. In contrast, in existing data stream models (e.g. the *turnstile* model [35, p9]), an update (i, x) implies $d_i \leftarrow d_i + x$. So, instead of increments as in previous models, an update resets data values.

The reset model also differs from the *dynamic geometric* model considered in e.g. [27] processes a stream of intermixed operations $INS(p)$ and $DEL(q)$, for points p and q . One cannot reduce the reset model to the dynamic geometric model, as simulating a reset by a DEL followed by an INS gives information about the previous location of the point that is not readily available to the algorithm in the reset model (recall that \mathbf{x} and \mathbf{y} are not stored explicitly).

Motivations. The reset model has two independent motivations. First, there is a selftuning approach to selectivity estimation in databases [7]. The query processor has a “summary” that describes some (not necessarily the current) state of the data distribution in the database. It selects query plans for execution using the summary; but when a query is executed, as a by-product, it gets to know the actual values the query intersects. The query optimizer needs to use this feedback to update its knowledge of the data distribution. Each time it knows

an actual value, it corresponds to a reset. The goal is for the query optimizer to estimate various parameters of interest on the data distribution under this stream of resets. This precisely corresponds to the reset model of data streams. The second motivation arises in streams of locations. For example, in traffic sensor information systems [34], the stream corresponds to location-dependent queries from clients that are passed on to one of several servers by mediators. Each server gets a subset of clients; a given client may go to multiple servers before being assigned to one of them again. Servers need to track statistics on the locations of clients assigned to them. At each server, a new location-dependent query for a client is precisely a reset operation and they need to tracking statistics on their location corresponds using “summaries”. We have provided only high level view of how reset model of streaming fits both these applications. More details are in [7, 34].

We now describe our results in the reset and delta models. In what follows, an (ϵ, δ) -estimator of a quantity x is a random variable \hat{x} such that $\Pr[(1 - \epsilon)x \leq \hat{x} \leq (1 + \epsilon)x] \geq 1 - \delta$.

Results. The reset model is significantly more restrictive than known data stream models, so it presents significant algorithmic challenges. Indeed, for the reset model, only very basic techniques appear to be useful, such as random sampling, using which several problems such as approximate quantiles or the 1-median problem can be solved easily (see e.g. [29]). We obtain results in this model by assuming that the update stream is *monotone*, i.e. an update always causes d_i to increase.

Let \mathbf{d} in \mathfrak{R}^n be updated as in the reset model. We consider two fundamental problems—that of estimating $\|\mathbf{d}\|_p$ for some $p \geq 0$, and L_p sampling, i.e. choosing an element from $\{1, \dots, n\}$ such that i is chosen with probability proportional to $|d_i|^p$. If $p = 0$ then the problem reduces to calculating the L_0 norm of a stream which can be solved as in [12]. In fact, we can assume that $p = 1$, and that d_i is always non-negative, without loss of generality. Specifically, let $1 \geq \delta, \epsilon > 0$ be two constants specified by the user. We give an algorithm that, if the update stream is monotone:

- returns on request an (ϵ, δ) estimator for $\|\mathbf{d}\|_1$,
- for any $k, 1 \leq k \leq n$, returns on request an (ϵ, δ) estimator for $\mathbf{d}^{(k)}$, the sum of the k largest elements in \mathbf{d} .
- returns on request an integer i from $\{1, \dots, n\}$ with probability \tilde{p}_i where $\frac{|d_i|}{\|\mathbf{d}\|_1(1+\epsilon)} - \epsilon/n \leq \tilde{p}_i \leq \frac{(1+\epsilon)|d_i|}{\|\mathbf{d}\|_1}$.

In case the algorithm is presented with a non-monotone update sequence, it (in essence) ignores any non-monotone updates, and may therefore return an incorrect answer. However, in case the update sequence is ‘significantly’ non-monotone, this can be detected, in the following sense:

- Let d_i^* be the maximum value attained by d_i over the sequence of updates, and let $\mathbf{d}^* = (d_1^*, \dots, d_n^*)$. If the update stream is non-monotone, with probability $1 - \delta$, we detect non-monotonicity before $\|\mathbf{d}\|_1(1 + \epsilon)^3 \leq \|\mathbf{d}^*\|_1$.

The algorithm uses $\text{polylog}(n)$ space, and $\text{polylog}(n)$ time to process each update and query. The approach is similar to one used in [14], but we consider a wider range of problems than were considered there.

We make a few remarks about the results. The problem of computing the L_∞ norm of \mathbf{d} (the maximum element) is known to be hard in the turnstile model, even if values are increasing monotonically. However, this problem is trivial in the monotone reset model. Conversely, the problem of computing the sum of the values in \mathbf{d} is trivial in the turnstile model, but requires a little effort in the reset model. It is not known how to solve the problem of L_p sampling, or to estimate the sum of the k -largest elements (clearly impossible for $k = 1$), in the turnstile model.

1.2 Delta Model

Let $\mathbf{x}, \mathbf{y} \in \mathfrak{R}^n$; initially, $\mathbf{x} = \mathbf{y} = \mathbf{0}$. The vectors are updated as follows:

Delta model: The updates appear in a stream $(i, \Delta x, \Delta y)$, implying that $x_i \leftarrow x_i + \Delta x$ and $y_i \leftarrow y_i + \Delta y$.

Again, \mathbf{x}, \mathbf{y} hold the coordinates of n points in \mathfrak{R}^2 . This model is a direct generalisation of the classical turnstile model to two dimensions. As in Remark 1, this model is different from the geometric stream models considered in the literature. Although we are aware of no “physically” spatial application where the delta model is currently appropriate¹ this model is also motivated by database contexts, where several of the myriad motivations of the (one-dimensional) turnstile model carry over.

Results. We consider the problem of computing the “extent” of the point set S given by \mathbf{x} and \mathbf{y} . Let \mathbf{d} be the vector of L_p distances of S relative to some centre $c = (c_x, c_y)$, which is specified as part of the query. The objective is to be able to estimate $\|\mathbf{d}\|_q$ for some q . If $p = q$, the problem is equivalent to that of estimating $\|\mathbf{x} - c_x\|_p + \|\mathbf{y} - c_y\|_p$, where \mathbf{x} and \mathbf{y} are the vectors of x - and y - coordinates respectively. Thus, the problem factors trivially into two 1-D L_p norm estimations. The truly interesting case is when $p = 2$ and $q = 1$; this corresponds to computing the sum of the (Euclidean) distance of the points from a given centre. We show:

- For any fixed $\epsilon > 0$ and $\delta > 0$, we give a data structure that, given any $c = (c_x, c_y)$, returns an (ϵ, δ) -estimator of $\|\mathbf{d}\|_1$, where \mathbf{d} is the vector of L_2 distances of points in S from c .

The result assumes that the choice of c is independent of the random choices made by the algorithm (more precisely, that the operation sequence is generated

¹ However, one can easily conceive of devices that can report the change in position since the last report, without being aware of their current position.

by an oblivious adversary). The algorithm uses $\text{polylog}(n)$ space, and $\text{polylog}(n)$ time to process each update and query.

In the remainder of the paper, we give the results on the reset and delta model in Section 2 and Section 3, and conclude with some open problems.

2 Reset model

Let \mathbf{d} in \mathfrak{R}^n be updated as in the reset model. Recall that an update stream is *monotone* if an update always causes d_i to increase. We show:

Lemma 1. *For any constants $1 \geq \delta, \epsilon > 0$ there is an algorithm that returns on request an (ϵ, δ) estimator \tilde{d} of $\|\mathbf{d}\|_1$; furthermore, given an integer k , $1 \leq k \leq n$, the algorithm can also return an (ϵ, δ) -estimator \tilde{d} of $\mathbf{d}^{(k)}$. Both these results assume that the update stream is monotone; the algorithm ignores non-monotone operations.*

The algorithm takes $O(\tau s)$ time to process an update and uses $O(\sigma s)$ space, where τ and σ are the time and space required to return an $(\epsilon', \delta/(2(s+1)))$ -estimator of the number of distinct elements in a stream of items from $\{1, \dots, n\}$, where $\epsilon' = \epsilon/3$ and $s = \lceil 1 + \log_{1+\epsilon'}(n/(\epsilon')^2) \rceil$.

Proof. We conceptually create an infinite sequence of buckets, $\dots, B_{-1}, B_0, B_1, \dots$, where the i -th bucket has *weight* $w_i = (1 + \epsilon')^i$ and $\epsilon' = \epsilon/3$. At any given time, only a contiguous sequence of buckets B_{r-s}, \dots, B_r is *active*. Let n_i denote the number of distinct elements from $\{1, \dots, n\}$ that are placed into B_i . Associated with each active bucket B_i is the assumed data structure that returns an $(\epsilon', \delta/(2(s+1)))$ -estimator \tilde{n}_i of the n_i . Activating a bucket means allocating memory for such a data structure and appropriately initializing it. For any d , we let $\rho(d)$ be the integer k such that $\sum_{j=-\infty}^{k-1} w_j < d \leq \sum_{j=-\infty}^k w_j$.

To process an update (i, d) , we determine $r_i = \rho(d)$ and distinguish three cases:

- If $r_i < r - s$ we update no buckets.
- If $r - s \leq r_i \leq r$ we place the integer i into *all* buckets B_{r-s}, \dots, B_{r_i} .
- If $r_i > r$, we deactivate buckets $B_{r-s}, \dots, B_{r_i-s-1}$, activate any inactive buckets among B_{r_i-s}, \dots, B_r , place the integer i into all these buckets, and finally set $r := r_i$.

To estimate $\|\mathbf{d}\|_1$, we proceed as follows. For $j = r - s, \dots, r$ we obtain an estimate \tilde{n}_j of the number of distinct values placed in B_j and return $\tilde{d} = \sum_{j=r-s}^r \tilde{n}_j \cdot w_j$. We now calculate the error in this estimate.

First note that by definition, $\sum_{j=-\infty}^{r_i-1} w_j \leq d_i$, and so $\sum_{j=-\infty}^{r_i} w_j \leq (1 + \epsilon')d_i$. Since $\tilde{n}_i \leq (1 + \epsilon')n_i$ for all $i = r - s, \dots, r$, it follows that with probability at least $1 - \delta/2$:

$$\tilde{d} = \sum_{j=r-s}^r \tilde{n}_j \cdot w_j \leq (1 + \epsilon') \sum_{j=r-s}^r n_j \cdot w_j$$

$$\begin{aligned} &\leq (1 + \epsilon')^2 \sum_{j=r-s}^r d_j \leq (1 + \epsilon/3)^2 \|\mathbf{d}\|_1 \\ &\leq (1 + \epsilon) \|\mathbf{d}\|_1 \end{aligned}$$

To lower-bound the estimate, take $s = \lceil 1 + \log_{1+\epsilon'}(n/(\epsilon')^2) \rceil$, and note that:

$$\sum_{j=r-s}^{r_i} w_j \geq d_i - \sum_{j=-\infty}^{r-s-1} w_j \geq d_i - d_{max}/(1 + \epsilon')^s \geq d_i - \epsilon' d_{max}/(n(1 + \epsilon'))$$

Summing over i , we get that:

$$\sum_{j=r-s}^r n_j \cdot w_j \geq \sum_{i=1}^n d_i - (\epsilon')^2 d_{max}/(1 + \epsilon') \geq (1 - \epsilon') \sum_{i=1}^n d_i$$

As with the upper bound, we note that the probability that \tilde{n}_i is not a significant underestimate is at least $(1 - \delta/2)$. This gives the desired probability bound.

Finally, we now describe how to compute the sum of the top k elements in \mathbf{d} , denoted here by $\mathbf{d}^{(k)}$. We omit the analysis of the failure probability, which is as above. For integer x , $r - s \leq x \leq r$, define $f(x) = k \sum_{i=r-s}^x w_i + \sum_{i=x+1}^r n_i w_i$. Let t be the largest index i such that $n_i \geq k$. Arguing as above, $(1 - \epsilon')\mathbf{d}^{(k)} \leq f(t) \leq (1 + \epsilon')\mathbf{d}^{(k)}$, as $f(t)$ is precisely how $\mathbf{d}^{(k)}$ is represented within the buckets. Now let $\tilde{f}(x) = k \sum_{i=r-s}^x w_i + \sum_{i=x+1}^r \tilde{n}_i w_i$. The algorithm chooses the largest index t' such that $\tilde{n}_{t'} \geq k$ and returns $\tilde{f}(t')$ as the estimator. We now bound the error $|f(t) - \tilde{f}(t')|$, considering first the case $t' \geq t$.

Since $\tilde{n}_{t'} \geq k$, except with negligible probability, we have $n_{t'} \geq k/(1 + \epsilon')$. Thus, $k/(1 + \epsilon) \leq n_{t'} \leq \dots \leq n_{t+1} < k$, and:

$$\begin{aligned} |f(t) - \tilde{f}(t')| &\leq \sum_{i=t+1}^r w_i |\tilde{n}_i - n_i| + \sum_{i=t+1}^{t'} w_i (k - n_i) \\ &\leq \sum_{i=t+1}^r \epsilon' n_i w_i + \sum_{i=t+1}^{t'} \epsilon' w_i n_i \leq \epsilon' f(t) \end{aligned}$$

If $t > t'$ then for $i = t' + 1, \dots, t$, $n_i \geq k > \tilde{n}_i \geq (1 - \epsilon')n_i$. Thus:

$$\begin{aligned} |f(t) - \tilde{f}(t')| &\leq \sum_{i=t'+1}^r w_i |\tilde{n}_i - k| + \sum_{i=t'}^t w_i (k - \tilde{n}_i) \\ &\leq \sum_{i=t'+1}^r \epsilon' n_i w_i + \sum_{i=t'+1}^{t'} \epsilon' w_i n_i \leq \epsilon' f(t) \end{aligned}$$

Thus, we have that $(1 - \epsilon')^2 \mathbf{d}^{(k)} \leq \tilde{f}(t') \leq (1 + \epsilon')^2 \mathbf{d}^{(k)}$, and the result follows as above. \square

For non-monotone sequences, let d_i^* be the maximum value attained by d_i over the sequence of updates, and let $\mathbf{d}^* = (d_1^*, \dots, d_n^*)$. We now show:

Theorem 1. *For any constants $1 \geq \delta, \epsilon > 0$ there is an algorithm that, if the update stream is monotone:*

- (a) *returns on request an (ϵ, δ) -estimator of $\|\mathbf{d}\|_1$,*
- (b) *for any $k, 1 \leq k \leq n$, returns on request an (ϵ, δ) -estimator of $\mathbf{d}^{(k)}$ and*
- (c) *returns on request an element i from $\{1, \dots, n\}$ with probability \tilde{p}_i where*

$$\frac{|d_i|}{\|\mathbf{d}\|_1(1+\epsilon)} - \epsilon/n \leq \tilde{p}_i \leq \frac{(1+\epsilon)|d_i|}{\|\mathbf{d}\|_1}.$$

Non-monotone updates are ignored by the algorithm. Furthermore,

- (d) *if the update stream is non-monotone, the algorithm, with probability at least $1 - \delta$ detects the non-monotonicity before $\|\mathbf{d}\|_1(1 + \epsilon)^3 \leq \|\mathbf{d}^*\|_1$, where \mathbf{d}^* is as above.*

The algorithm uses polylog(n) space and time to process an update.

Proof. (Sketch) Lemma 1 essentially proves (a) and (b); it only remains to note that either the Flajolet-Martin algorithm or its refinements [18, 10] or the algorithm of [12, Theorem 4] estimates the number of distinct items in a stream in at most polylog(n) space and time.

We now outline a solution to (c). Ideally, for every active bucket B_i we would choose a random representative, such that each (distinct) element in B_i is chosen with probability $1/n_i$ (and update the representative when a bucket is updated). When asked for a sample from \mathbf{d} , we would return one of the representatives, choosing the representative of bucket i with probability proportional to $n_i w_i$. Then, the probability of selecting i is (taking $r_i = \rho(d_i)$):

$$\sum_{j=r-s}^{r_i} \frac{1}{n_j} \frac{n_j w_j}{\sum_{j=r-s}^r n_j w_j} = \frac{\sum_{j=r-s}^{r_i} w_j}{\sum_{j=r-s}^r n_j w_j}.$$

As before, the numerator is essentially d_i and the denominator is essentially $\|\mathbf{d}\|_1$. Other sources of errors are that we are able only to sample a bucket representative with a distribution that is arbitrarily close to uniform (using approximately min-wise independent permutations, see [15]), and that we must work with probabilistic estimates \tilde{n}_j rather than n_j . These are handled as in Lemma 1.

Finally, we consider (d), and indicate how to detect non-monotonicity. After a non-monotonic update, the updated element may continue to exist incorrectly in a (higher-numbered) bucket. We let n_i^* and n_i denote the number of elements placed in B_i by the algorithm, and the number of elements that ‘ought’ to be in B_i , respectively. If $\|\mathbf{d}\|_1(1 + \epsilon)^3 < \|\mathbf{d}^*\|_1$, where \mathbf{d}^* is as above, then $(1 + \epsilon) \sum_{j=r-s}^r n_i w_i \leq \sum_{j=r-s}^r n_i^* w_i$. Thus, there is at least one bucket B_i for which $(1 + \epsilon)n_i w_i \leq n_i^* w_i$. This can only happen if at least $\epsilon n_i^*/(1 + \epsilon)$ of the n_i^* elements that were ever in B_i were updated non-monotonically. This can

be detected with the required probability by keeping a test sample of $m = O(\log \delta / (1 - \epsilon / (1 + 2\epsilon)))$ elements chosen approximately uniformly at random from the n_i^* elements in B_i , together with their current values, and checking to see if any of the elements in the test sample are updated non-monotonically. (The sampling is done as in [15].) \square

Remark 2. In contrast, the problem under the general case when updates are not monotonic is provably impossible to solve under streaming constraints (e.g. polylog space). We can reduce the problem to that estimating $|A| - |A \cap B|$ for two sets A and B given in standard turnstile model, which in turn, can be shown to have a nearly linear space lower bound for $1 + \epsilon$ approximation by known lower bounds in Communication Complexity. We omit showing this process here because it is quite standard once the problem of estimating $|A| - |A \cap B|$ is given as the key.

3 Delta model

Recall that in this model, a set S of n 2-D points is represented by two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$. We consider the following problem:

- Let \mathbf{d} be the vector of L_2 distances of the points of S relative to some centre $c = (c_x, c_y)$ (the centre is specified as part of the query). The objective is to be able to estimate $\|\mathbf{d}\|_1$.

We begin by considering a simple 1-D version of the problem. Consider a vector \mathbf{x} of size n that contains x -coordinates, and suppose that we process a sequence of updates $(i, \Delta x)$, which sets $x_i \leftarrow x_i + \Delta x$. Then, we have that:

Proposition 1. *For any fixed $\epsilon > 0$ and $\delta > 0$, there is a data structure that processes each update in poly-log time, uses poly-log space, and in poly-log time, given any c , returns an (ϵ, δ) -estimator of $\sum_{i=1}^n |x_i - c|$.*

Proof. We essentially observe that the standard algorithm [28] for estimating $\|\mathbf{x}\|_1$ also computes the above quantity. The algorithm maintains a *sketch* of \mathbf{x} , which is simply a sequence of values $\mathbf{x} \cdot \mathbf{r}_i$, for $i = 1, \dots, k$, and where \mathbf{r}_i is a pseudo-random vector drawn from an *1-stable* distribution. It can be shown that the (normalised) median of $\mathbf{x} \cdot \mathbf{r}_i$, for $i = 1, \dots, k$, for sufficiently large k is an (ϵ, δ) -estimator of $\|\mathbf{x}\|_1$. In order to estimate $\|\mathbf{x} - \mathbf{c}\|_1$, where $\mathbf{c} = (c, c, \dots, c)$, we simply compute a sketch of $\mathbf{x} - \mathbf{c}$, by computing $\mathbf{x} \cdot \mathbf{r}_i - \mathbf{c} \cdot \mathbf{r}_i$, for $i = 1, \dots, k$. This can be done in $O(1)$ time per sketch element provided the sum of the values in \mathbf{r}_i is known. Since \mathbf{r}_i is not stored explicitly, this is not trivial, but can be done by using so-called *range-summable* random variables [21, Lemma 2].

The 2-D version of the problem can be reduced to two 1-D problems by considering the projections of the points on to two distinct axes. Estimating the extent based on two projections, however, would yield a $(\sqrt{2} + \epsilon)$ -approximation in the worst case. Considering the projections of the points along several axes allows us to greatly improve the accuracy:

Lemma 2. Let ℓ be a line segment of length x . Consider a collection \mathcal{C} of $k \geq 1$ lines passing through the origin, at angles $(\pi i)/k$ for $i = 0, \dots, k-1$. Then, if s is the sum of the lengths of the projections of ℓ on all lines in \mathcal{C} , then $x(1 - \Theta(1/k)) \leq s\pi/(2k) \leq x(1 + \Theta(1/k))$.

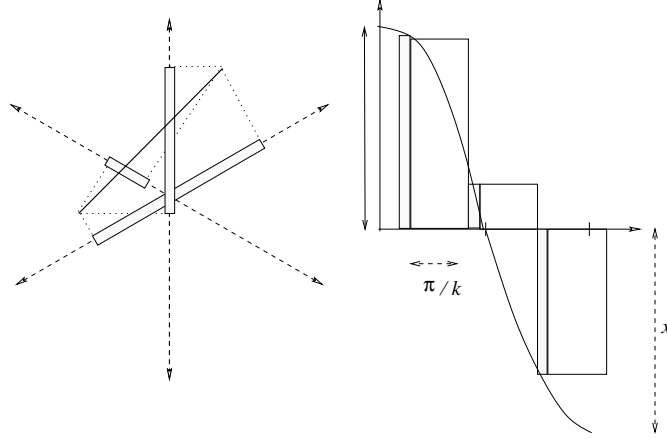


Fig. 1. Approximation of a line by its projections.

Proof. Viewing the question as a continuous one, we can approximate the sum by $x \int_0^\pi |\cos(\theta)| d\theta = 2x$. As \cos has bounded slope, we have that $|s\pi/k - 2x| = O(x/k)$ (see Figure 1). The lemma follows.

We conclude:

Theorem 2. For any fixed $\epsilon > 0$ and $\delta > 0$, there is a data structure that processes each update in poly-log time, uses poly-log space, and in poly-log time, given any $c = (c_x, c_y)$, returns an ϵ, δ -estimator of $\|\mathbf{d}\|_1$, where \mathbf{d} contains the L_2 distances from c to points in S .

Proof. For k a sufficiently large constant, we consider a collection of k lines through the origin as above. Let ℓ be a line in this collection, and project all the points (x_i, y_i) onto ℓ . Let t_i be the distance of the projected point from the origin. We use Proposition 1 to maintain a sketch of the vector (t_1, \dots, t_n) ; this allows us also to project a given centre c on to ℓ and use the sketch to estimate the total distance $\tau_\ell(c)$ from the projection of the centre. But $\tau_\ell(c)$ is nothing but the sum of the projections of the distance from c to the points; by Lemma 2, we can sum all estimates of $\tau_\ell(c)$ over ℓ , multiply this by $\pi/2k$, and obtain the desired ϵ, δ estimator.

4 Conclusions and Open Problems

We have proposed a new model for processing high speed location streams, and presented algorithms for the problem of tracking the “extent”, i.e., l_p norm of the distances of the points from a center. For the reset model, the assumption is that the input sequence is *monotone*; we obtain further algorithms for L_p sampling and computing the sum of the k largest elements. All algorithms work under ‘streaming’ constraints, namely, they use poly-logarithmic space and time.

The problem of processing location streams does not have the immediacy of applications that processing IP network streams has, at this point. But the earliest streaming algorithms were developed around 1996 [8] when the immediacy of IP networking application was not clear. Our model is challenging and we hope it will prove to be a rich source of problems as new applications with location streams become prevalent. Many natural open problems are obvious in the reset models, including the fundamental geometric problems such as estimating convex hulls, etc. Particularly interesting is the formulation of suitable notions of monotonicity that may be needed to make these problems tractable.

Acknowledgements. We thank Graham Cormode for several enlightening discussions, and to Christian Söhler for bringing [29] to our attention.

References

1. DIMACS Workshop on Managing and Processing Data Streams, FCRC 2003. <http://www.research.att.com/conf/mpds2003/>
2. <http://www.interfleet.com/>
3. <http://www.lbszone.com/> <http://gislounge.com/ll/lbs.shtml>
<http://www.lbsportal.com/>
4. <http://www.whereismybus.com/>
5. <http://dimacs.rutgers.edu/Workshops/WGDeliberate/FinalReport5-20-02.doc>.
6. DIMACS Working Group on Streaming Data Analysis. <http://dimacs.rutgers.edu/Workshops/StreamingII/>
7. A. Abounaga and S. Chaudhuri. Self-tuning histograms: Building histograms without looking at the data. *Proc. SIGMOD*, 1999.
8. N. Alon, Y. Matias and M. Szegedy. The space complexity of approximating the frequency moments. *Proc. ACM STOC*, 20–29, 1996.
9. B. Babcock, S. Babu, M. Datar, R. Motwani and J. Widom. Models and issues in data stream systems. *ACM PODS*, 2002, 1–16.
10. Z. Bar-Yossef, T. S. Jayram, R. Kumar, D. Sivakumar, L. Trevisan. Counting distinct elements in a data stream. In *Proc. RANDOM 2002*, LNCS 2483, pp. 1–10, 2002.
11. M. Charikar, L. O’Callaghan and R. Panigrahy. Better streaming algorithms for clustering problems. *ACM STOC*, 2003.
12. G. Cormode, M. Datar, P. Indyk and S. Muthukrishnan. Comparing data streams using Hamming norms (How to zero in). *IEEE Trans. Knowledge and Data Engineering*, **15** (2003), pp. 529–541.
13. G. Cormode and S. Muthukrishnan. Radial Histograms. DIMACS TR 2003.

14. G. Cormode and S. Muthukrishnan. Estimating dominance norms of multiple data streams. In *Proc. ESA 2003*, LNCS 2832, pp. 148–160, 2003.
15. M. Datar and S. Muthukrishnan Estimating Rarity and Similarity over Data Stream Windows *Proc. ESA 2002*, LNCS 2461, 323–334, 2002.
16. C. Estan, S. Savage and G. Varghese. Automatically inferring patterns of resource consumption in network traffic. SIGCOMM 2003.
17. J. Feigenbaum, S. Kannan and J. Ziang. Computing diameter in the streaming and sliding window models. *Manuscript*, 2002.
18. P. Flajolet and G. Martin. Probabilistic counting algorithms for database applications. *JCSS*, **31**, 1985, pp. 182–209.
19. P. Gibbons and Y. Matias. Synopsis data structures. *Proc. SODA*, 1999, 909–910.
20. A. Gilbert, Y. Kotidis, S. Muthukrishnan and M. Strauss. Surfing wavelets on streams: One pass summaries for approximate aggregate queries. *VLDB Journal*, 79–88, 2001.
21. A. C. Gilbert, S. Guha, P. Indyk, P. Indyk, Y. Kotidis, S. Muthukrishnan and M. J. Strauss. Fast, small-space algorithms for approximate histogram maintenance. *Proceedings 34th ACM STOC*, 389–398, 2002.
22. M. Greenwald and S. Khanna. Space-efficient online computation of quantile summaries. *Proc. ACM SIGMOD*, 2001.
23. S. Guha, N. Mishra, R. Motwani and L. O’Callaghan. Clustering data streams. *IEEE FOCS*, 2000, 359–366.
24. S. Har-Peled and S. Mazumdar. On Coresets for k -Means and k -Median Clustering. *Proc. 36th ACM STOC*, 2004, pp. 291–300.
25. M. Henzinger, P. Raghavan and S. Rajagopalan. Computing on data stream. Technical Note 1998-011. Digital systems research center, Palo Alto, May 1998.
26. J. Hershberger and S. Suri Convex hulls and related problems on data streams. Proc. MPDS 2003.
27. P. Indyk. Algorithms for dynamic geometric problems over data streams Proc. *Annual ACM Symposium on Theory of Computing (STOC)*, 2004, pp. 373–380.
28. P. Indyk. Stable distributions, pseudorandom generators, embeddings and data stream computation. *IEEE FOCS* 2000, 189–197.
29. P. Indyk and M. Thorup. Unpublished manuscript, 2001.
30. R. Jana, T. Johnson, S. Muthukrishnan and A. Vitaletti. Location based services in a wireless WAN using cellular digital packet data (CDPD). *MobiDE 2001*: 74-80
31. F. Korn, S. Muthukrishnan and D. Srivastava. Reverse nearest neighbor aggregates over data streams. *Proc. VLDB*, 2002.
32. B. Krishnamurthy, S. Sen, Y. Zhang and Y. Chen. Sketch-based change detection: methods, evaluation and applications. Proc. Internet Measurement Conference (IMC), 2003.
33. G. Manku and R. Motwani. Approximate frequency counts over data streams. *Proc. VLDB*, 2002, 346–357.
34. S. Madden and M. Franklin. Fjording the stream: An architecture for queries over streaming sensor data. *Proc. ICDE*, 2002.
35. S. Muthukrishnan. Data Streams: Algorithms and Applications. <http://www.cs.rutgers.edu/muthu/stream-1-1-1.ps>
36. J. Bates. Talk at NAS meeting on Statistics and Massive Data. http://www7.nationalacademies.org/bms/Massive_Data_Workshop.html.
37. Querying and mining data streams: you only get one look. Tutorial at SIGMOD 2002, VLDB 2002, etc. See <http://www.bell-labs.com/user/minos/tutorial.html>.
38. G. Varghese. Detecting packet patterns at high speeds. Tutorial at SIGCOMM 2002.