The Space Complexity of Pass-Efficient Algorithms for Clustering

Kevin L. Chang*† Ravi Kannan*

Abstract

We present multiple pass streaming algorithms for a basic clustering problem for massive data sets. If our algorithm is allotted 2ℓ passes, it will produce an approximation with error at most ϵ using $\tilde{O}(k^3/\epsilon^{2/\ell})$ bits of memory, the most critical resource for streaming computation. We demonstrate that this tradeoff between passes and memory allotted is intrinsic to the problem and model of computation by proving lower bounds on the memory requirements of any ℓ pass *randomized* algorithm that are nearly matched by our upper bounds. To the best of our knowledge, this is the first time nearly matching bounds have been proved for such an exponential tradeoff for randomized computation.

In this problem, we are given a set of n points drawn randomly according to a mixture of k uniform distributions and wish to approximate the density function of the mixture. The points are placed in a datastream (possibly in adversarial order), which may only be read sequentially by the algorithm. We argue that this models, among others, the datastream produced by a national census of the incomes of all citizens.

1 Introduction

The streaming model of computation has received much attention in the recent literature. In this model, the input to an algorithm may only be read in a single, sequential pass over the data; no random access of the input is allowed. Furthermore, the amount of memory used by the algorithm must be small (typically $o(n)$, where n is the size of the datastream). The streaming model fits well with the constraints of computation with massive data sets, where the size of the input is much too large to fit into main memory. The data must be placed in secondary storage (for instance a tape, or disk that must be read in entire blocks), which may only be accessed sequentially.

Multiple passes were first discussed in the early datastream papers [1, 13, 19]. The restricted *pass-efficient* model has been proposed by Drineas and Kannan [8] as a more flexible version of the streaming model. Intuitively, the rigid restriction of the streaming model to a single pass over the data unnecessarily limits its potential utility. The pass-efficient model allows for multiple sequential passes over the input (ideally a small, perhaps constant, number of passes), and a small amount of extra space. While processing each element of the datastream, the algorithm may only use computing time that is independent of n, but after each pass, it is allowed more computing time (typically $o(n)$). In this model of computation, we are most concerned with two resources: *the number of passes* and *additional storage space (memory)*. The model has been applied to a two pass algorithm for computing a succinct, approximate representation of a matrix [8].

In this paper, we consider a new clustering problem. Our problem, *learning a mixture of k uniform distributions*, is roughly stated as follows: Given a set of points drawn from a mixture of distributions (to be defined precisely below), compute the clusters (i.e. density function of the mixture) from which the points were drawn. The main motivation for our problem arises from considerations in the analysis of census data. Census data can naturally be modeled as a mixture of uniform distributions. Consider for instance the data set consisting of the personal incomes of a sampled set of citizens or, as in the case of the United States census, in principle all citizens. The incomes of individuals engaged in any single profession are certainly not exactly the same, but rather are distributed over some range of incomes. A reasonable first step in modeling such distributions is to assume that the incomes in a given profession are uniformly distributed over some interval. Thus, suppose profession i corresponds to the range of incomes (a_i, b_i), and represents w_i proportion of the population. A sample drawn for the census can be viewed as distributed according to a mixture of uniform distributions in the following way: with probability w_i, the individual sampled belongs to profession i, in which case the individual's income will fall in the interval (a_i, b_i). The whole of the census data may be in adversarial order, since realistically we may make no assumptions about the ordering of the census data. For instance, the ordering of the data may be based on location of residence, or some arbitrary identification number, rather than in-

*Dept. of Computer Science, Yale University, New Haven, CT 06520, USA. Email: **[kchang, kannan]@cs.yale.edu.**

†Supported by NSF's ITR program under grant number 0331548.

come value (indeed, a census record consists of many attributes; certainly records in a datastream cannot be sorted by more than one attribute).

An interesting question to ask is: *assuming the census data is distributed as a mixture of uniform distributions, can we compute the distribution of incomes?* Since the amount of data in a census is very large, the pass-efficient model is appropriate.

More formally, a *mixture of k uniform distributions* is defined by k possibly intersecting intervals (a_i, b_i) for $i = 1, \ldots, k$ and a corresponding *mixing weight* $w_i \geq 0$ for each interval, such that $\sum w_i = 1$. We assume that the mixture is not degenerate: $-\infty < a_i < b_i < \infty$, for all i. A mixture of k uniform distributions is a probability distribution on $\mathbb{R}$ defined by choosing the ith interval with probability w_i, and then picking a point uniformly at random from the chosen interval.

Suppose that X is a set of n points randomly chosen from $\mathbb{R}$ according to a mixture of k-uniform distributions with density function F. X can be ordered (possibly by the adversary) to produce a sequence which constitutes the datastream. Our problem is then: Design a pass-efficient algorithm that approximates F from the datastream X.

Vapnik-Cervonenkis arguments show that a random sample of size $\tilde{O}(k^2/\epsilon^2)$ ($\tilde{O}(\cdot)$ and $\tilde{\Omega}(\cdot)$ denote asymptotic notation with polylog factors omitted) from X will have about the right number of points in any interval; thus intuitively clustering the sample should give us an ϵ approximation to F. Some care is needed to rigorously prove this (see Section 3 for precise statements). The two main results of our paper are **a)** a multiple pass algorithm whose space requirements drop off significantly with more passes allowed and **b)** a lowerbound that shows that our tradeoff between passes and memory usage is close to tight. To the best of our knowledge, this is the first paper with a lower bound on randomized computation that shows that the sharp tradeoff that we achieve (memory decreases by $\Omega(1/\ell)$ in the exponent) is tight. Previous work either only considered deterministic lower bounds in very limited models of computation, or lower bounds with memory scaling by $1/\ell$. We also generalize our multiple pass algorithm to learning mixtures of k linear distributions.

We now describe the general technique for our multiple pass algorithm. In a single pass, we partition the domain into a set of intervals, based on samples of the datastream. Our algorithm will then estimate F on each of these intervals separately. In a second pass, we count the number of points in X that lie in each of these intervals, and run subroutine **Constant?**, which determines whether or not F is (approximately) constant on each interval. If F is constant on an interval I, then the density of F can be estimated easily by the number of points of X that fall in the interval. If F is not constant on I, the algorithm recursively estimates the density in the interval using subsequent passes, in effect "zooming in" on these more troublesome intervals until we are satisfied with our approximation.

A crucial part of the algorithm is subroutine **Constant?**, which solves a problem of independent interest: determining in a single pass over X whether or not F is approximately constant on some interval. For any $\beta > 0$, **Constant?** uses only $O((\log(1/\beta) + \log k)\log(1/\delta))$ bits of memory, provided that $n = \tilde{\Omega}(k^5/\beta^5)$, and determines whether F is within β in L^1 distance of the uniform distribution. **Constant?** first slightly perturbs the data in X so that each point is drawn from an η-*smoothed distribution*, F_η, which is very close to F in variational distance. F_η has the useful property that it is constant on intervals that form a regular partition of the domain. We then would like to compute the number of samples of X in each of the $1/\eta$ subintervals; we look upon this as a vector v with $1/\eta$ components. $1/\eta$ will be too large for us to store the entire vector; however, using the known algorithm in [14], we find the projection of v into a random low dimensional subspace, from which we can glean a good approximation to $||v||_1$. Pseudorandom generators for small space computation are then used to compress the projection matrix (these generators do not require the existence of one-way functions).

In order to prove lower bounds for multiple pass algorithms, we appeal to known results for the communication complexity of the GT (Greater Than) function, which determines for two inputs a, b whether $a > b$. Specifically, we show that any ℓ pass algorithm that solves our problem will induce a $(2\ell-1)$-round protocol for the GT function. Lower bounds on the $(2\ell-1)$-round communication complexity of GT will then provide lower bounds for the memory usage of ℓ pass streaming algorithms.

1.1 Our Results

The L^1 *distance* with respect to the usual Lebesgue measure between measurable functions f and g is defined as $\int_{\mathbb{R}} |f - g|$. Given $\epsilon > 0$ and $\delta > 0$, we describe various randomized pass-efficient algorithms that find with probability at least $1-\delta$ a function G that is an approximation to F, with error measured by L^1 distance, $\int_{\mathbb{R}} |F - G|$:

1. In Section 3, a one pass algorithm with error at most ϵ that requires $\tilde{O}(k^2/\epsilon^2)$ bits of memory.

2. In Section 4, an at most 2ℓ pass algorithm with error at most ϵ using $\tilde{O}(k^3/\epsilon^{2/\ell})$ space for any integer $\ell > 0$, provided that $n = \tilde{\Omega}(1.25^\ell k^6/\epsilon^6)$.

Alternatively, we can view this as an algorithm with error at most ϵ^ℓ that uses $\tilde{O}(k^3/\epsilon^2 + \ell/\epsilon)$ bits of memory, provided that $n = \tilde{\Omega}(1.25^\ell k^6/\epsilon^{6\ell})$.

3. In Section 5, we consider a slight generalization of our main learning problem and prove a lower bound of $\Omega\left(\left(\frac{1}{2\epsilon}\right)^{1/(2\ell-1)} c^{-2\ell+1}\right)$ for the bits of memory needed by any ℓ pass, *randomized* algorithm that solves this new problem, for some constant c. The algorithm from Section 4 can be modified to solve this generalized problem. Thus, we modify the multiple pass algorithm to provide an upper bound of $\tilde{O}(1/\epsilon^{4/\ell})$ on bits needed for an ℓ pass algorithm when ℓ is even, for the generalized problem.

4. In Sections 6 and 7, we generalize our multiple pass algorithm to algorithms for *learning a mixture of k linear distributions* and *learning a mixture of k two dimensional uniform distributions*. If w is an upper bound on the maximum density of the mixture, we present a 2ℓ pass algorithm that uses $\tilde{O}(k^3/\epsilon^{2/\ell})$ bits of memory for the former problem and a $4\ell+1$ pass algorithm using $\tilde{O}(k^4/\epsilon^{3/\ell})$ bits of memory for the latter.

These results show that making more passes over the data allows our algorithms to use less space, thus trading one resource in the pass-efficient model for the other. This feature demonstrates the potential for multiple passes in designing a streaming algorithm with flexible performance guarantees on space needed and passes used; the algorithm can be tailored to the specific needs and limitations of a given application and system configuration.

Some details and proofs have been omitted, but will appear in the journal version.

1.2 Related Work: Lower bounds for multi-pass algorithms.

Papers with space lower bounds for multiple pass algorithms include [9, 13]. These papers consider streamed graphs and problems of the nature: find all nodes at distance exactly k from a particular vertex. The former paper considers lower bounds for deterministic algorithms. The latter paper contains a lower bound of $\Omega(nk/\ell)$ on randomized computation, as well as a nearly matching upper bound. Bar-Yossef *et al.* [3] have proved lower bounds for multiple pass algorithms approximating frequency moments, which are, however, independent of the number of passes taken. Matching upper bounds are provided in [1, 15].

A more comparable work is that of Munro and Paterson[19], who proved a lower bound of $\Omega(n^{1/\ell})$ for the number of storage locations needed for an ℓ pass deterministic algorithm for median finding, and a nearly matching upper bound. The model of computation is more limited, since it is deterministic and assumes that only input elements may be stored (not some sort of sketch).

Streaming clustering algorithms. One pass streaming algorithms have been designed for a wide array of clustering problems. In contrast to our approximation schemes, these algorithms give constant factor (or weaker) approximations; for our purposes and motivations, such coarse approximations may not be acceptable. Charikar *et al.* [4] gave a streaming algorithm that achieved a constant factor approximation for k-center, using $O(k)$ extra space. Also, Charikar *et al.* [5] gave a streaming algorithm that achieved a constant factor approximation for k-median, using $O(k \log^2 n)$ extra space.

Histogram problems. Another related problem that has been studied in the theory and database literature is the problem of histogram construction [6, 10, 16, 17]. Generally, these problems are similar to ours, but there are some major distinctions. First, our paradigm and algorithm generalize very naturally to mixtures of other distributions; as we show in this paper, our algorithm is easily adapted to two dimensions, and to piecewise linear density functions. Histograms are only comparable to the one dimensional uniform case.

Guha *et al.* [12] designed one pass algorithms for the histogram construction problem. Gilbert *et al.* [11] improved on this, designing one pass algorithms for the maintenance problem, which is more general than our uniform problem (but not the linear or two dimensional cases), since it does not assume the input comes from a mixture. However, we exploit the extra mixture structure to give us the multiple pass aspect of the algorithm to reduce the amount of memory. Thus, in contrast to our space usage of $\tilde{O}(k^3/\epsilon^{2/\ell})$ and $O(\log k \log(1/\epsilon))$ time per data element, their work uses much more time per update and space (both polynomial in $(1/\epsilon, k, \log n)$), where n is the size of the datastream.

Mixtures of Gaussians Algorithms for learning mixtures of Gaussian distributions in high dimension have been studied previously [2, 7], but these are not streaming or pass efficient.

2 Preliminaries

We first give an alternate description of the structure of the probability density function induced by a mixture of uniform distributions.

LEMMA 2.1. *Given a mixture of k uniform distributions $(a_i, b_i), w_i$, let F be the induced probability density function on $\mathbb{R}$. F (except possibly at at most $2k$ points) can be represented as a collection of at most $2k-1$ dis-*

joint intervals, with a constant value on each interval.

Since the above characterization of F holds except on a set of measure zero, it will suffice for our algorithm to learn F with this representation. We will be somewhat lax in our language, and say that a set of intervals partitions some larger interval I, even if there are a finite number of points of I that are not included in any interval of the partition.

DEFINITION 2.1. *The representation of a mixture of k uniform intervals as a set of at most $2k-1$ pairs, $(x_i, x_{i+1}), h_i$, for $i = 0, \ldots, m-1 \leq 2k$ such that the density of the mixture is a constant h_i on the interval (x_i, x_{i+1}) will be called the* step function representation *of the mixture. We call each interval (x_i, x_{i+1}) a* step *of the mixture.*

Implicit in this definition is the fact that the m steps form a partition of the interval (x_0, x_m). The *support* of a mixture given by step function representation $(x_i, x_{i+1}), h_i$ consisting of m steps is the interval (x_0, x_m). A *jump* of a step function is a point x where the density changes (i.e. an endpoint shared by two different steps of the mixture).

We will work exclusively with the step function representation of the mixture.

3 A Single Pass Algorithm

In this section, we sketch a single pass algorithm that with probability $1-\delta$ will learn the density function of a mixture of k uniform intervals within L^1 distance ϵ using memory $\tilde{O}(k^2/\epsilon^2)$. The algorithm is a divide and conquer algorithm reminiscent of merge sort.

In a single pass, we choose $m = \tilde{O}(k^2/\epsilon^2)$ points uniformly at random and store them in memory. The algorithm divides the sample into two halves and recursively approximates the density function F in each half. The approximation is given as a partition of the domain into intervals on which the density function is estimated as constant. We then apply a *merging* procedure that decides whether or not the last interval of one half can be merged with the first interval of the other half. We omit the algorithm due to space constraints.

THEOREM 3.1. *There exists a one pass algorithm that will learn a mixture of k uniform intervals to within L^1 distance ϵ with probability at least $1-\delta$, using at most $\tilde{O}\left(\frac{k^2}{\epsilon^2}\log(1/\delta)\right)$ bits of memory.*

4 Multiple Passes

In this section, we show that additional passes can significantly reduce the amount of memory needed to achieve the same accuracy. Given a mixture of k uniform distributions with density function F and an integer ℓ, algorithm **SmallRam** can approximate F to within L^1 distance ϵ with 2ℓ passes, using at most $\tilde{O}(k^3/\epsilon^{2/\ell})$ memory, provided that $n = \tilde{\Omega}((1.25^\ell k^6)/\epsilon^6)$. We first give an algorithm with accuracy ϵ^ℓ using space at most $\tilde{O}(k^3/\epsilon^2 + \ell/\epsilon)$.

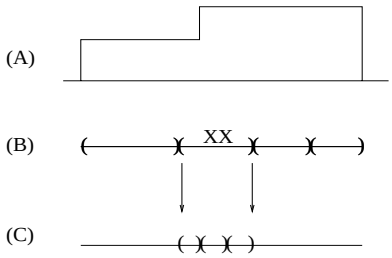

Figure 1: Zooming. (A) A step distribution. (B) Intervals created in Step 3 of some call to **Estimate**. One of them contains a jump and has been marked. (C) **Estimate** is recursively called on the marked interval.

SmallRam makes use of a subroutine called **Constant?**. **Constant?** is a single pass algorithm that will determine whether or not a datastream contains points drawn from a distribution with a constant density function. In Section 4.2, we describe the algorithm and prove the following theorem about its performance:

THEOREM 4.1. *Let H be a mixture of at most k uniform distributions. Given a datastream X consisting of $|X| = \tilde{O}(\frac{k^5}{\beta^5}\log(1/\delta))$ samples drawn from H, with probability $1-\delta$ single pass algorithm **Constant?** will* **accept** *if H is uniform, and will* **reject** *if H is not within L^1 distance β of uniform. **Constant?** uses at most $O((\log(1/\beta) + \log k)\log(1/\delta))$ bits of memory.*

4.1 The Algorithm Subroutine **Estimate** draws a random sample S of size $|S| = \tilde{O}(k^2/\epsilon^2)$ from the input stream. It sorts S and partitions it into $O(k/\epsilon)$ intervals. **Estimate** then calls **Constant?** on each of the intervals; **Constant?** uses the entire datastream X to determine if the distribution is constant on the interval. If F is close to constant on an interval, we can output the interval as a step of the final output G with a very accurate constant density estimate, again using the large set of samples, X, read via the datastream. However, if F is far away from constant on the interval, then estimating its density by a constant is too coarse; thus we repeat the process by recursively calling **Estimate**. This recursive call can be thought of as "zooming in on the jumps." See Figure 1.

Given a sequence of points X and an interval J, we define $X|_J$ to be the subsequence of X that consists of points that fall in J. $X|_J$ can be considered a

datastream; a pass over X can simulate a pass over $X|_J$.

Algorithm **SmallRam** input: datastream X of samples from distribution with density F, such that $n = |X| = \Omega\left(\frac{1.25^\ell k^6}{\epsilon^{6\ell}} \cdot \ell \cdot \log\left(\frac{k\ell}{\delta\epsilon}\right)\right)$

1. initialize $p \leftarrow 1$, $J \leftarrow$ Support(F) (Note that J can be found in the first pass of the algorithm).
2. Call **Estimate** on p, J.

Subroutine **Estimate**: input: interval J, and level p.

1. Make a fresh pass, extracting a set S of size $m = \Theta\left(\frac{8k^2}{\epsilon^2}\log\left(\frac{\ell k}{\epsilon\delta}\right)\right)$ from $X|_J$, and store it in memory.
2. Sort the samples in S in ascending order. If $p = 1$, let $q \leftarrow \frac{9\epsilon}{20k}m$; otherwise let $q \leftarrow (9/10)\cdot\epsilon m$.
3. Partition J into $m/q = 20k/(9\epsilon)$ or $10/(9\epsilon)$ disjoint intervals J_i^p, such that $|S \cap J_i^p| = q$.
4. Make an additional pass to count the exact number of points of X, the entire datastream, that fall in each of the J_i^p's.
5. Also in this pass, call subroutine **Constant?** on $X|_{J_i^p}$, for each of the m/q intervals J_i^p in parallel, with error parameter $\epsilon^\ell/2k$. Mark those intervals J_i^p that **Constant?** rejects. Let M^p be the set of marked intervals.
6. If interval J_i^p is not marked in the previous step, output it as a step of G with density $|X \cap J_i^p|/(n\text{length}(J_i^p))$.
7. If $p < \ell$, for each $J_i^p \in M^p$, run **Estimate** on $J \leftarrow J_i^p$, $p \leftarrow p+1$, in parallel with all other level $p+1$ copies of **Estimate**.

   If $p = \ell$, output each interval $J_i^p \in M^p$ as a step of G with density 0.

The order in which this recursive algorithm computes each call to **Estimate** has a natural structure defined by the level of the call (defined by its parameter p): All level p copies of **Estimate** are run in parallel, then all level $p+1$ copies are run in parallel, and so forth. All copies of **Estimate** running in parallel may read the datastream simultaneously, thus limiting the total number of passes per level to two. The total number of passes is therefore 2ℓ.

LEMMA 4.1. *Let J_i^p be an interval created in Step 3 of a level p call to* **Estimate**. *With probability at least $1 - \delta\epsilon/(25k^2\ell)$, J_i^p contains at most $\epsilon^p/2k$ proportion of the total weight of F, and at least $(8/10)^p\epsilon^p/2k$ proportion of the total weight. That is,*

$$\left(\frac{8}{10}\right)^p \cdot \frac{\epsilon^p}{2k} \le \int_{J_i^p} F \le \frac{\epsilon^p}{2k}.$$

The lemma follows from induction and VC bounds.

COROLLARY 4.1. *Let J_i^p be some interval created in step 3 of* **Estimate**, *and suppose $\epsilon < 8/10$. With probability at least $1 - \delta\epsilon/(20k^2\ell)$, the datastream X satisfies*

$$|X \cap J_i^p| = \Omega\left(\frac{k^5}{\epsilon^{5\ell}} \cdot \ell \cdot \log\left(\frac{k\ell}{\delta\epsilon}\right)\right). \tag{4.1}$$

Proof. Follows from the Chernoff bound.

LEMMA 4.2. *With probability $1 - \delta/2$, the following is true for all levels p: the aggregate number of intervals marked in level p calls to* **Estimate** *is at most $2k - 1$.*

The idea behind this Lemma is that an interval may only be marked if it contains a jump in the distribution, and there are at most $2k - 1$ jumps.

We summarize all the events that occurred in order to derive the $1 - \delta/2$ bound from the previous lemma:

COROLLARY 4.2. *With probability at least $1 - \delta/2$, the following are true:*

1. *Any level p interval contains at least $(8/10)^p\epsilon^p/2k$ proportion of the weight of F and at most $\epsilon^p/2k$.*
2. *Equation (4.1) holds for all intervals created in Step 3 of* **Estimate**.
3. *There are at most $2k-1$ calls to* **Estimate** *at any level.*
4. *No calls to **Constant?** fail (i.e., the low probability event that **Constant?** doesn't work correctly does not occur).*

We now prove the main theorem of this section.

THEOREM 4.2. *With probability at least $1 - \delta$, **SmallRam** will compute an approximation to F within L^1 distance ϵ^ℓ of F, using at most 2ℓ passes and $\tilde{O}(k^3/\epsilon^2 + \ell/\epsilon)$ bits of memory.*

Proof. We condition this proof on Corollary 4.2 occurring, which has probability at least $1 - \delta/2$.

We first bound the amount of memory used. At any level of the computation, we are running at most $2k - 1$ parallel copies of **Estimate**. Since each copy

uses at most $O(m)$ bits of memory, the total amount of memory used is at most $2k$ times this amount:

$$2k \cdot O\left(\frac{4k^2}{\epsilon^2}\log\left(\frac{\ell k}{\epsilon\delta}\right)\right).$$

We now prove that the algorithm outputs a good approximation to F. After all ℓ levels of the algorithm, we have density estimates for all of Support(F).

The intervals that define the steps of G are of two types:

1. Intervals that had their density estimated in Step 6 of some call to **Estimate**. A type (1) interval J_i^p can further be classified into one of two cases:
   (a) F does not contain a point in J_i^p where there is a jump from one step of F to another, and thus F is constant on J_i^p.
   (b) F does contain such a jump in J_i^p.
2. Intervals marked in level ℓ that had their density estimated in Step 7 of a level ℓ call to **Estimate**.

A bound for the error from type (1), case (a) intervals

Suppose J_i^p belongs to case (a). It follows from the Chernoff bound and Corollary 4.2, item 2, that with probability at least $1-\delta\epsilon/(32k^2\ell)$,

$$\left|\frac{|X\cap J_i^p|}{n} - \int_{J_i^p} F\right| \le \frac{\epsilon^\ell}{4k}\int_{J_i^p} F.$$

This inequality then implies that

$$\begin{aligned}\frac{\epsilon^\ell}{4k}\int_{J_i^p} F \ge \left|\frac{|X\cap J_i^p|}{n} - \int_{J_i^p} F\right| &= \left|\int_{J_i^p} G - \int_{J_i^p} F\right| \\ &= \int_{J_i^p} |F-G|,\end{aligned}$$

where the last equality follows from the fact that F and G are both constant on J_i^p.

Let Γ be the set of all type (1) case (a) intervals. Since there are at most $((10/9)4\ell k^2/\epsilon)$ intervals in Γ, the probability that the above inequality holds for all of them is at least $1-\delta/4$. Then, the total error induced by all such intervals is at most:

$$\sum_{J_i^p\in\Gamma}\int_{J_i^p}|F-G| \le \frac{\epsilon^\ell}{4k}\sum_{J_i^p\in\Gamma}\int_{J_i^p} F \le \frac{\epsilon^\ell}{4k}.$$

A bound for the error from type (1), case (b) intervals

Suppose that J_i^p belongs to case (b). It follows from the Chernoff bound and Corollary 4.2, item 2, that with probability $1-\delta\epsilon/(32k^2\ell)$,

$$\begin{aligned}\left|\frac{|X\cap J_i^p|}{n\text{length}(J_i^p)} - \frac{\int_{J_i^p} F}{\text{length}(J_i^p)}\right| &\le \frac{\epsilon^\ell}{8k^2\text{length}(J_i^p)}\int_{J_i^p} F \\ &\le \frac{\epsilon^\ell}{8k^2\text{length}(J_i^p)}.\end{aligned}$$

Define $\bar{F}|_{J_i^p} = \frac{1}{\text{length}(J_i^p)}\int_{J_i^p} F$ to be the *average density* of F on J_i^p. Since J_i^p was not marked, we know that subroutine **Constant?** accepted when run on J_i^p. This means that F is at most $\epsilon^\ell/2k$ in L^1 distance from constant on J_i^p:

$$\int_{J_i^P}\left|F-\bar{F}|_{J_i^p}\right| \le \frac{\epsilon^\ell}{2k}.$$

Combining the two inequalities above gives us

$$\frac{\epsilon^\ell}{2k}+\frac{\epsilon^\ell}{8k^2} \ge \int_{J_i^p}\left|F - \frac{|X\cap J_i^p|}{n\text{length}(J_i^p)}\right| = \int_{J_i^p}|F-G|.$$

If t_1 is the number of case (b) intervals, the total error induced by all case (b) intervals is at most $(\epsilon^\ell/2k + \epsilon^\ell/8k^2)\cdot t_1$, with probability at least $1-\delta/4$.

A bound for the error from type (2) intervals

Each interval of type (2) was created at a level ℓ call to **Constant?**. Thus by Corollary 4.2, item 1, each contains at most $\epsilon^\ell/2k$ proportion of the weight of F. Estimating F by 0 on an interval of type (2) will induce an error of at most $\epsilon^\ell/2k$. The total error induced by type (2) intervals is at most $\epsilon^\ell/2k\cdot t_2$, where t_2 is the number of type 2 intervals.

A bound for the total error

Since type (1) case (b) and type (2) intervals only occur at jumps in steps of F, $t_1+t_2 \le 2k-1$. Thus, with probability at least $1-\delta$, the total error is at most

$$\begin{aligned}\frac{\epsilon^\ell}{4k}+(t_1+t_2)\left(\frac{\epsilon^\ell}{2k}+\frac{\epsilon^\ell}{8k^2}\right) &\le \frac{\epsilon^\ell}{4k}+(2k-1)\left(\frac{\epsilon^\ell}{2k}+\frac{\epsilon^\ell}{8k^2}\right) \\ &\le \frac{\epsilon^\ell}{4k}+\frac{(2k-1)\epsilon^\ell}{2k}+\frac{\epsilon^\ell}{4k} = \epsilon^\ell.\end{aligned}$$

If we analyze the algorithm with $\epsilon^{1/\ell}$ in lieu of ϵ, we get an ϵ approximation requiring $\tilde{O}(k^3/\epsilon^{2/\ell})$ bits of memory (note the ℓ/ϵ term disappears).

COROLLARY 4.3. *There exists an algorithm that, with probability at least $1-\delta$, will compute an approximation to F within L^1 distance ϵ of F, using at most 2ℓ passes and $\tilde{O}(k^3/\epsilon^{2/\ell})$ bits of memory.*

4.2 Testing intervals for uniformity: Proof of Theorem 4.1. We now describe the single pass subroutine **Constant?** used by **SmallRam**. Given a datastream of n samples from a mixture of k uniform distributions, with H as its density function, **Constant?** will accept H if it is constant, and reject if it is not within β in L^1 distance of constant.

By suitably scaling and translating the input, we may assume without loss of generality that the support of H is exactly the interval $[0,1]$. Thus, our subroutine will determine whether $\int_0^1 |H-1| \leq \beta$ or not.

4.2.1 Preliminaries: Approximating the ℓ_1 length of a vector given as a stream of dynamic updates. Consider a datastream consisting of pairs of the form (i,a) where $i \in [n]$ and $a \in \{-M,\ldots,M\}$. We define the quantity $L_1(S) = \left(\sum_{i\in[n]} \left|\sum_{(i,a)\in S} a\right|\right)$. Note that $L_1(S)$ is the ℓ_1 length of the n dimensional vector given by $a_i = \sum_{(i,a)} a$.

FACT 4.1. *[14] There is an algorithm which estimates $L_1(S)$ up to a factor of $1/2$ with probability $1-\delta$ and uses $O(\log M \log(1/\delta))$ bits of memory, $O(\log M \log(n/\delta))$ random bits, and $O(\log n/\delta)$ arithmetic operations per pair (i,a).*

The η-smoothed distribution of H. For ease of exposition of the proof of Theorem 4.1, we introduce the *η-smoothed distribution* of H, which is a smoothed version of H.

DEFINITION 4.1. *Given a mixture of k uniform distributions that defines a probability density H and a number $0 < \eta \leq 1$, define the* η-smoothed distribution *to be the following distribution, with density H_η. Let $i \leq 1/\eta - 1$ be the integer such that $x \in (i\eta,(i+1)\eta]$. Then*

$$H_\eta(x) = \frac{1}{\eta}\int_{i\eta}^{(i+1)\eta} H(y)dy.$$

It is immediate that H_η is a step distribution consisting of at most $1/\eta$ steps. Thus, let h_i be the constant density of H_η on each of the intervals $(i\eta,(i+1)\eta)$. Let $\alpha_i = |h_i - 1|$ be the distance from 1 of the density of H_η on the ith interval. Our interest in H_η lies in the following useful properties.

LEMMA 4.3. *If H is uniform, then H_η is also uniform.*

LEMMA 4.4. *Let $\eta < \beta/5k$. Let $I = (x_i, x_{i+1}), w_i$ be a step of H, and suppose that*

$$|w_i length(I) - length(I)| > \beta/2k.$$

For some step I' of H_η, we have $|H_\eta(x) - 1| > \beta/2k$ for $x \in I'$.

COROLLARY 4.4. *If $\eta \leq \beta/5k$ and H is not within L^1 distance β of uniform, there exists a step of H_η such that $\alpha_i \geq \beta/2k$.*

The two lemmas establish the connection between H and H_η. An algorithm that rejects if there is at least one step i such that $\alpha_i > \beta/2k$, and that accepts if H_η is uniform (i.e. all $\alpha_i = 0$) will be sufficient for the guarantee of Theorem 4.1.

4.2.2 The algorithm and its proof of correctness First, we define a number of random variables based on the datastream that can be used to estimate α_i.

1. Let $N_i = |X \cap (i\eta,(i+1)\eta)|$ be the number of points of X that fall into the interval $(i\eta,(i+1)\eta)$. It is important to note that X consists of samples of H, but that $\mathbf{E}[N_i/n\eta] = h_i$.
2. Let $\hat{\alpha_i} = \left|\frac{1}{\eta}\frac{N_i}{n} - 1\right|$ be an estimate of α_i based on the datastream. Note that $\mathbf{E}[\hat{\alpha_i}] = \alpha_i$.

The random variable $\hat{\alpha_i}$ gives an estimate of α_i, the quantity we are interested in. However, in a single pass we do not have enough space to compute and store $\hat{\alpha_i}$ for all $1/\eta$ values of i. The proof of the next lemma uses Indyk's algorithm to estimate $||\hat{\alpha}||_1$ using a small amount of space. From the value of $||\hat{\alpha}||_1$, we can glean our desired information: a small value of $||\hat{\alpha}||_1$ implies that all $\hat{\alpha_i}$ are small, whereas a large value implies that at least one $\hat{\alpha_i}$ is large.

LEMMA 4.5. *In a single pass over X we can compute an estimate ζ such that $(1/2)||\hat{\alpha}||_1 \leq \zeta \leq (3/2)||\hat{\alpha}||_1$ with probability at least $1-\delta$, using space at most $O(\log(\eta n)\log(1/\delta))$.*

Proof. First consider the construction of a datastream S_X derived from X:

1. For each element $x \in X$, let $j_x \in [1/\eta]$ be the integer such that $x \in (j_x\eta,(j_x+1)\eta)$. We append $(j_x,1)$ to S_X.
2. We append to S_X the $1/\eta$ elements $(i,-\eta n)$, for all $i \in [1/\eta]$.

Clearly Indyk's algorithm on input S_X can be simulated in a single pass over X. We have:

$$L_1(S_X) = \sum_{i=0}^{1/\eta} |N_i - \eta n| = \eta n \sum_i |\frac{1}{\eta}\frac{N_i}{n} - 1| = \eta n||\hat{\alpha}||_1.$$

Thus, in a single pass we can derive an estimate ζ of $||\hat{\alpha}||_1$ such that $(1/2)||\hat{\alpha}||_1 \leq \zeta \leq (3/2)||\hat{\alpha}||_1$ with Indyk's algorithm using space at most $O(\log(\eta n)\log(1/\delta))$.

We now formally describe the algorithm:

Constant? input: datastream X such that

$$|X| = \tilde{O}\left(\left(\frac{k}{\beta}\right)^2 \frac{1}{\eta^3}\log\frac{1}{\delta}\right) = \tilde{O}\left(\frac{k^5}{\beta^5}\log\frac{1}{\delta}\right)$$

1. Set $\eta \leftarrow \beta/(5k)$.
2. Simulate Indyk's algorithm on X, as described in Lemma 4.5. Let ζ be the estimate of $||\hat{\alpha}||_1$ output by the algorithm.
3. **accept** if $\zeta \leq \beta/5k$. Otherwise, **reject**.

Proof. (of Theorem 4.1) Since $n = \tilde{O}(k^2/(\beta^2\eta^3))$, an application of Hoeffding's bound implies that with probability at least $1-\delta$, for all $1/\eta = O(k/\beta)$ choices of i simultaneously: if $h_i \leq 2$ then $|N_i - h_i\eta n| \leq \frac{\beta\eta}{40k}\sqrt{w_i}\eta n$, which implies that

$$|\hat{\alpha_i} - \alpha_i| \leq \frac{\beta\eta}{20k}.$$

If $h_i \geq 2$ then $|N_i - h_i\eta n| \leq \frac{\beta\eta}{20k}\sqrt{w_i}\eta n$, which implies that

$$\hat{\alpha_i} \geq \frac{1}{2} \geq \frac{\beta}{k}.$$

We prove the correctness of **Constant?** by considering two cases:

Case 1: H_η is uniform. Then $h_i = 1$ for all i, and $\alpha_i = 0$ for all i. Thus, $\hat{\alpha_i} \leq \frac{1}{20k}\beta\eta$ for all i. By Lemma 4.5, this implies that $\zeta \leq \frac{3}{2}||\hat{\alpha}||_1 \leq \beta/5k$; therefore **Constant?** will accept H_η. Appealing to Lemma 4.3 it follows that **Constant?** will accept if H is uniform.

Case 2: There exists at least one h_i such that $|h_i - 1| \geq \beta/2k$. Then $||\hat{\alpha}||_1 \geq \beta/2k - \frac{1}{20k}\beta\eta > \beta/(2.5k)$. By Lemma 4.5, this implies that $\zeta \geq \frac{1}{2}||\hat{\alpha}||_1 > \beta/5k$; therefore **Constant?** will reject H_η. Appealing to Corollary 4.4, it follows that **Constant?** will reject if H is not within L^1 distance β of uniform.

5 Matching bounds on memory usage of randomized algorithms

In this section, we consider lower bounds on the amount of memory needed by any ℓ pass **randomized** algorithm for a generalization of our learning problem. We then sketch a pass-efficient algorithm for this problem.

Problem 5.1. (Generalized Learning Problem) Let F be the density function of a mixture of at most $1/\epsilon$ uniform distributions on $[0,1]$. Let $t \in [0,1]$ be the largest number such that F is a step distribution with at most k steps on $[0,t]$. Given a datastream X consisting of sufficiently many iid samples from F, with probability at least $1-\delta$, find a function G and a number $t' > t$ such that $\int_0^{t'} |F - G| < \epsilon$.

Intuitively, the problem is to learn the density function of the first k steps of a distribution containing $1/\epsilon$ steps.

We use known lower bounds for the communication complexity of r-round protocols to prove that any ℓ pass algorithm needs at least $\Omega\left(\left(\frac{1}{2\epsilon}\right)^{1/(2\ell-1)} c^{-2\ell+1}\right)$ bits of memory to solve this problem for the $k=3$ case. We then generalize our algorithms for learning mixtures of k-uniform distributions to solving the generalized learning problem. These algorithms will provide an upper bound of $\tilde{O}(k^3/\epsilon^{4/\ell})$ on the number of bits of memory needed by an ℓ pass algorithms, for even ℓ.

5.1 A Lower Bound Consider the following communication problem: Alice and Bob are given vectors $a, b \in \{0,1\}^n$ respectively and wish to compute $f(a,b)$ for some Boolean function f.

An r-round protocol is a protocol with the restriction that Alice and Bob may exchange at most r messages, where Alice must send the first message and only one of the players need output the answer. The *r-round probabilistic communication complexity of f*, denoted by $R^r(f)$, is the number of bits in the largest message, minimized over all r-round protocols that output f correctly with probability at least $2/3$.

For the function $\mathrm{GT}_n : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ defined by $\mathrm{GT}_n(a,b) = 1$ iff $a \geq b$, a lower bound of $R^r(\mathrm{GT}_n) = \Omega(n^{1/r}c^{-r})$, for some fixed constant c, and a nearly matching upper bound of $R^r(\mathrm{GT}_n) = O(n^{1/r}\log n)$ are known [18]. We will use $R^r(\mathrm{GT}_n)$ to prove lower bounds for multiple pass algorithms:

Theorem 5.1. *Any ℓ pass randomized algorithm that solves the Generalized Learning Problem for $k=3$ with probability at least $2/3$ and error ϵ requires at least $\Omega\left(\left(\frac{1}{2\epsilon}\right)^{1/(2\ell-1)} c^{-2\ell+1}\right)$ bits of memory, for some fixed constant c.*

Proof. Fix $k=3, \epsilon, m = 1/(2\epsilon)$, and an ℓ pass algorithm A that solves the Generalized Learning Problem with probability $2/3$ and that uses at most $M(A)$ bits of memory. We will prove that A induces a $2\ell-1$-round protocol for GT_m that uses at most $M(A)$ bits of communication per message. This will prove that $M(A) \geq R^{2\ell-1}(\mathrm{GT}_m) = \Omega(1/(2\epsilon)^{2\ell-1}c^{-2\ell+1})$.

Suppose that Alice and Bob are given vectors $a, b \in \{0,1\}^m$. Alice will construct a probability distribution μ_a^A. μ_a^A will be constructed as a piecewise constant function on each of the $2m$ intervals $(i/m, (i+1/2)/m]$ and $((i+1/2)/m, (i+1)/m]$ as follows: Set $\mu_a^A(x) =$

$2a_{m-i}$ for $x \in (i/m, (i+1/2)/m]$ and $\mu_a^A(x) = 2(1 - a_{m-i})$ for $x \in ((i+1/2)/m, (i+1)/m]$.

Bob will construct a similar function μ_b^B, but "reversed": $\mu_b^B(x) = 2(1-b_{m-i})$ for $x \in (i/m, (i+1/2)/m]$ and $\mu_b^B(x) = 2b_{m-i}$ for $x \in ((i+1/2)/m, (i+1)/m]$.

Alice (Bob) generates an arbitrarily large sample of points drawn iid according to μ_a (μ_b) of arbitrary precision and places them in a datastream X_a (X_b). We require that $|X_a| = |X_b|$. The datastream formed by appending X_b to X_a, denoted by $X_a \circ X_b$, can be viewed as having been drawn from the distribution defined by density function $\mu = \frac{\mu_a^A}{2} + \frac{\mu_b^B}{2}$. Note that μ is a step function with at most $1/\epsilon$ steps.

Claim: If Bob is given the solution to the Generalized Learning Problem on datastream $X = X_a \circ X_b$ with parameters ϵ, $k = 3$, he can compute $\mathrm{GT}_m(a, b)$.

Bob has in his possession the vector b and the output of the algorithm: a number t' such that on $(0, t')$, μ is comprised of at least 3 steps, and a function μ_{approx} such that $\int_0^{t'} |\mu - \mu_{\text{approx}}| < \epsilon$.

In order to compute $\mathrm{GT}_m(a, b)$, all Bob needs to do is learn the bits a_i for $i \geq j^*$, where j^* is the highest order bit for which a_i and b_i differ. Note that μ is a step function consisting of 3 steps on the interval $(0, (j^*+1)/m)$: It has a constant value of 1 on $(0, j^*/m)$ since $a_i = b_i$ for bits $i > j^*$. If $a_{j^*} = 1$ (and correspondingly $b_{j^*} = 0$) then it has a constant value of 2 on $(j^*/m, (j^*+1/2)/m)$ and a value of 0 on $((j^*+1/2)/m, (j^*+1)/m)$. If $a_{j^*} = 0$ (and correspondingly $b_{j^*} = 1$), the situation is reversed. Note that $t' \geq (j^*+1)/m$

Bob can compute $\mathrm{GT}_m(a, b)$ in the following manner. He finds (by enumeration) a vector $\tilde{a} \in \{0,1\}^m$ that induces the probability density $\tilde{\mu} = \frac{\mu_{\tilde{a}}^A}{2} + \frac{\mu_b^B}{2}$, such that $\int_0^{t'} |\tilde{\mu} - \mu_{\text{approx}}| \leq \epsilon$. Bob then outputs 1 iff $\tilde{a} \geq b$.

The correctness of this scheme follows from the observation that $\tilde{a}_i = a_i$ for all bits $i \geq j^*$. Suppose this were not the case, that $\tilde{a}$ differed from a by a bit a_i for $i \geq j^*$ then $\int_0^t |\tilde{\mu} - \mu| \geq 2\epsilon$, but since Bob is guaranteed that $\int_0^{t'} |\mu - \mu_{\text{approx}}| < \epsilon$, it follows from the triangle inequality that $\int_0^t |\tilde{\mu} - \mu_{\text{approx}}| \geq \epsilon$.

The $2\ell - 1$-round protocol for computing $\mathrm{GT}_m(a, b)$ is as follows: Alice simulates the first pass of algorithm A on X_a to find the contents of memory at this intermediate point of the pass. She sends these $M(A)$ bits of memory to Bob. Bob continues the simulation of the first pass of A on X_b and sends his $M(A)$ bits to Alice. They simulate each pass of A in this fashion until all ℓ passes have been completed, sending a total of $2\ell - 1$ messages of size at most $M(A)$.

After the completion of the communication, Bob will have the output of A run on datastream $X_a \circ X_b$, from which he can determine $\mathrm{GT}_m(a, b)$.

5.2 An Upper Bound Using the full power of **Constant?** (it can decide whether or not a mixture of $1/\epsilon$ uniform distributions is within ϵ of uniform using at most $O(\log(1/\epsilon)\log(1/\delta))$ bits of memory), we can strengthen **SmallRam**:

THEOREM 5.2. *For even ℓ, there exists an ℓ pass algorithm that can solve the Generalized Learning Problem with probability 2/3 and error ϵ using at most $\tilde{O}(k^3/\epsilon^{4/\ell})$ bits of memory.*

For **SmallRam** to learn a mixture of k uniform intervals using $\tilde{O}(k^3/\epsilon^{4/\ell})$ bits of memory, it was crucial that at each level the algorithm only zooms in on $2k-1$ intervals (the intervals that contain jumps). If we want to learn the first $k+1$ steps of a mixture of $1/\epsilon$ uniform distribution using the same amount of memory, it is only necessary to zoom in on the first $k+1$ subintervals that contain a jump (rather than all $1/\epsilon$ intervals).

6 Learning mixtures of linear distributions

Let F be the density function of a mixture of k linear distributions. We will need to make the assumption that the algorithm is given an upper bound on F, call it w: $F(x) \leq w$ for all x in the domain. As with mixtures of uniform distributions, we can describe F as a piecewise linear density function with at most $O(k)$ different pieces: $a_i x + b_i$ for $x \in (x_i, x_{i+1})$.

Let X be a datastream of points drawn from distribution H. There exists a single pass algorithm that will determine whether or not a distribution H is within β of linear, using at most $O((\log(1/\beta) + \log k + \log w)\log(1/\delta))$ bits of memory. A mixture of linear distributions can then be learned in the same way as uniform distributions, with only minor modifications.

THEOREM 6.1. *There exists a 2ℓ pass algorithm that, with probability at least $1-\delta$ can learn a mixture of k linear distributions within L^1 distance ϵ using at most $\tilde{O}(k^3/\epsilon^{2/\ell})$ bits of memory. The algorithm requires the datastream to have size $\tilde{\Omega}(k^5 w^5/\epsilon^5)$.*

7 Learning mixtures of uniform distributions in two dimensions

Let F be the density function of a mixture of k uniform distributions on axis-aligned rectangles in $(0,1) \times (0,1) \subset \mathbb{R}^2$. We call such a distribution a *mixture of k uniform(2) distributions*. F is thus given by k axis aligned rectangles, R_i, $i = 1, \ldots, k$, each with weight w_i. Note that each of these rectangles is defined by four boundary lines, for a total of $4k$ boundary lines.

We say that F is *vertical* on a rectangle $R = (0, a) \times (0, b)$ if it does not contain any horizontal boundary line. Let X be a datastream of points drawn from F. There exists a one pass algorithm that will determine if F is within β of vertical using at most $O((\log 1/\beta + \log k + \log w) \log(1/\delta))$ bits of memory.

Our $4\ell + 1$ pass algorithm for learning a mixture of k uniform(2) distributions is similar to the algorithm for the one dimensional case. As before, we organize the algorithm into ℓ pairs of passes. In the first pass of each pair, we partition the domain into rectangles of roughly equal weight. In the second pass of each pair, we test each of these horizontal rectangles for verticality: if the rectangle S is close to vertical, we can project the points of X that lie in S onto the horizontal axis and learn a mixture of $O(k)$ uniform distributions using **SmallRam** in 2ℓ passes, from which we can derive a good approximation to F on S. Otherwise, we zoom in on S in subsequent passes.

THEOREM 7.1. *Let F be the density function of a mixture of k uniform(2) distributions, such that $F(x) \leq w$ for all $x \in R$. Suppose X is a datastream drawn according to F, with a sufficient number of elements. There exists a $4\ell + 1$-pass algorithm that approximate F to within L^1 distance ϵ using at most $\tilde{O}(k^4/\epsilon^{3/\ell})$ bits of memory.*

8 Open Problems

Our learning problem is very flexible; possible extensions include algorithms for learning mixtures of uniform distributions in three or more dimensions, or mixtures of Gaussian distributions in constant dimension.

References

[1] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the 28th Annual ACM Symposium on the Theory of Computing*, pages 20–29, 1996.

[2] S. Arora and R. Kannan. Learning mixtures of arbitrary gaussians. In *Proceedings of the 33rd Annual ACM Symposium on the Theory of Computing*, pages 247–257, 2001.

[3] Z. Bar-Yossef, T. S. Jayram, R. Kumar, and D. Sivakumar. An information statistics approach to data stream and communication complexity. In *Proceedings of the 43rd IEEE Symposium on Foundations of Computer Science*, pages 209–218, 2002.

[4] M. Charikar, C. Chekuri, T. Feder, and R. Motwani. Incremental clustering and dynamic information retrieval. In *Proceedings of the 29th Annual ACM Symposium on the Theory of Computing*, pages 626–635, 1997.

[5] M. Charikar, L. O'Callaghan, and R. Panigrahy. Better streaming algorithms for clustering problems. In *Proceedings of the 35th Annual ACM Symposium on the Theory of Computing*, pages 30–39, 2003.

[6] S. Chaudhuri, R. Motwani, and V. R. Narasayya. Random sampling for histogram construction: How much is enough? In *SIGMOD Conference*, pages 436–447, 1998.

[7] S. DasGupta. Learning mixtures of gaussians. In *Proceedings of the 40th IEEE Symposium on Foundations of Computer Science*, pages 634–644, 1999.

[8] P. Drineas and R. Kannan. Pass efficient algorithm for large matrices. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 223–232, 2003.

[9] J. Feigenbaum, S. Kannan, A. McGregor, and S. Suri. Graph distances in the streaming model: The value of space. In *Proceedings of the 16th Symposium on Discrete Algorithms*, pages 745–754, 2005.

[10] P. B. Gibbons, Y. Matias, and V. Poosala. Fast incremental maintenance of approximate histograms. In *Proceedings of 23th International Conference on Very Large Data Bases*, pages 266–275, 1997.

[11] A. C. Gilbert, S. Guha, P. Indyk, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Fast, small-space algorithms for approximate histogram maintenance. In *Proceedings of the 34th Annual ACM Symposium on the Theory of Computing*, pages 389–398, 2002.

[12] S. Guha, N. Koudas, and K. Shim. Data-streams and histograms. In *Proceedings of the 33rd Annual ACM Symposium on the Theory of Computing*, pages 471–475, 2001.

[13] M. R. Henzinger, P. Raghavan, and S. Rajagopalan. Computing on data streams. *External memory algorithms*, pages 107–118, 1999.

[14] P. Indyk. Stable distributions, pseudorandom generators, embeddings, and data stream computation. In *Proceedings of the 41th IEEE Symposium on Foundations of Computer Science*, pages 189–197, 2000.

[15] P. indyk and D. Woodruff. Optimal approximations of the frequency moments of data streams. In *Proceedings of the 37th Annual ACM Symposium on the Theory of Computing*, pages 202–208, 2005.

[16] H. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. C. Sevcik, and T. Suel. Optimal histograms with quality guarantees. In *Proceedings of 24th International Conference on Very Large Data Bases*, pages 275–286, 1998.

[17] Y. Matias, J. S. Vitter, and M. Wang. Dynamic maintenance of wavelet-based histograms. In *Proceedings of 26th International Conference on Very Large Data Bases*, pages 101–110, 2000.

[18] P. B. Miltersen, N. Nisan, S. Safra, and A. Wigderson. On data structures and asymmetric communication complexity. *J. Comput. Syst. Sci.*, 57(1):37–49, 1998.

[19] J. I. Munro and M. Paterson. Selection and sorting with limited storage. *Theoretical Computer Science*, 12:315–323, 1980.