Approximate Quantiles and the Order of the Stream

Sudipto Guha[*] Andrew McGregor
Department of Computer and Information Science
University of Pennsylvania
{sudipto,andrewm}@cis.upenn.edu

ABSTRACT

Recently, there has been an increased focus on modeling uncertainty by distributions. Suppose we wish to compute a function of a stream whose elements are samples drawn independently from some distribution. The distribution is unknown, but the order in which the samples are presented to us will not be completely adversarial. In this paper, we investigate the importance of the ordering of a data stream, without making any assumptions about the actual distribution of the data. Using quantiles as an example application, we show that we can design provably better algorithms, and settle several open questions on the impact of order on streams. With the recent impetus in the investigation of models for sensor networks, we believe that our approach will allow the construction of novel and significantly improved algorithms.

Categories and Subject Descriptors

F.2 [**Analysis of Algorithms & Problem Complexity**]

General Terms

Algorithms, Design, Performance, Theory

Keywords

quantiles, data streams, random order, adversarial order

1. INTRODUCTION

Over the last decade streaming has gained significant currency as a feasible paradigm for situations involving large data. Although significant progress has been made in developing algorithms for data streams, much remains to be understood. Consider the following example.

EXAMPLE 1. *In the model-driven data acquisition scenario described in [7] we want to create a quantile based summary of the distribution of data for each of a number of sensor nodes. Given a fixed space bound S we have two choices—either to store a set of S samples or use a streaming algorithm using space S but drawing more samples. It appears reasonable to expect that in the latter, as we draw more samples, we should get better estimates. But, in the case of computing an estimate of the median, even for one sensor node, the existing algorithms find an element of rank $(\frac{1}{2} \pm \epsilon)n$ where $\epsilon = 1/S$ (ignoring polylogarithmic factors). Suppose we now doubled the number of observations, these algorithms would still return an item of "relative rank" $(\frac{1}{2} \pm \epsilon)$, i.e. the accuracy would not improve. This is counter-intuitive, if we observe more data, we should have more accurate estimates. In statistical terminology, the estimator is not* consistent, *i.e. increasing the sample size does not decrease the probability of being far from the quantity being estimated. On the other hand, if we store all the data, the estimator would be consistent. A natural question in this regard is, are there consistent estimators whose space use is significantly sublinear in the amount of data?*

The issue is one of modeling. In the above example, the known algorithms will try to provision for more data being input and settle for pessimistic estimates. In a sense, the benefit of using more data is being offset by the fact that there is more potential for an adversary to arrange for the data to be presented in a misleading fashion.

In general, there are two components to an instance of a data stream problem: the object $\mathcal{O}$ described by the elements in the stream, e.g., the sensor readings in the above example, and the order in which the elements arrive. Aggregate statistics of the data, e.g., mean, median, etc., are typically independent of the order of these elements in the stream. The importance of the order is the extent to which it makes it difficult to compute the desired function. When proving performance guarantees of streaming algorithms most of the existing literature assumes that $\mathcal{O}$ is *worst-case* and that the order of the elements is chosen *adversarially.*

These two, the worst case input, and the order in which it is presented, can be orthogonal issues depending on the scenario, e.g., as above. The distribution of values of the sensor may be worst case (i.e., it is not a nice distribution) but if we have reason to believe that data elements are samples independently drawn from some distribution, it is plausible to believe that the order of these data elements is random. In general it might be too much to assume that the order is entirely random. Rather, there may exist patterns or trends on a small scale but, on a larger scale things "appear" random. We may assume that these local aberrations are generated

[*]Supported in part by an Alfred P. Sloan Research Fellowship and by an NSF Award CCF-0430376.

by an adversary but that this adversary has limited power. We now describe such a scenario.

Example 2. *Consider a real time network monitoring setting where the data stream consists of TCP packets, gigabytes of which are flowing past in a short time. The statistical assumptions made by queuing theory to analyze TCP packets need not be perfect, but their success implies that it is unlikely that there is a "genie in the network" delivering all packets at exactly the wrong point of time. Even if such an adversary existed, constrained in time and space, the adversary will have difficulty producing a bad input, because that in itself requires substantial resources. The "adversary" in this process is the network itself. Some packets will be delayed because of race conditions and the behavior of the network. But all packets will not be affected in the worst possible way. In effect, we would see that the packets behave according to a distribution and in the aggregate scheme of things this distribution will be an accurate model. It is no surprise that one of the most important applications of streaming and approximate computation is to find "aggregate" properties of the data—we believe that the aggregate properties signify something about the characteristics of the input.*

In the previous two examples, we viewed the data in the stream as being stochastically generated. This gave rise to the belief that it was not ordered adversarially.[1] The following is a scenario in which we dictate the order of the stream. Naturally we can therefore ensure it is non-adversarial!

Example 3. *Consider a scenario where we are estimating the selectivity of operators or some aggregate property by sampling tuples from a database [5, 16, 15]. Can we expect that the* order *in which we receive the samples is random? The answer depends on the architecture one is dealing with, but in several situations the answer is likely yes, since we are controlling the sampling. Consider the "backing sample" architecture proposed by Gibbons, Matias, and Poosala in [10, 9] where a large sample is stored in the disk and that sample is used to periodically correct estimates of data. Since the backing sample is large, we can use a streaming algorithm to make a pass over this data. Since we are storing the sample we can ensure that the order is not adversarial. The data structures for backing sample use* hashes *to enable $O(1)$ lookup and updates—a good hash function implies that the samples (in different pages) are stored in a random order.*

Likewise, consider the pipelined hash join *scenario and assume we are interested in estimating aggregate statistics "on-the-fly" [18, 3, 2, 1], possibly to decide if we should switch to an alternative evaluation plan. The records are stored in a hash table and if we simply make a linear pass over the data and use a streaming algorithm, the data is uncorrelated across the buckets.*

In the next scenario, the semantics of the data make it a fair assumption that some fields in a database are uncorrelated. We note that several query optimizers use such assumptions.

[1]Note the connection with the problem of string compression. Arguments from Kolomogorov complexity prove that there do not exist string compression algorithms that work well on all strings. One of the main reasons for the success of compression algorithms has been a statistical view of input and not an adversarial view.

Example 4. *Consider a database of salaries where the primary key is the "name." If we are seeing tuples sorted in the primary key is it reasonable to expect that the "salary" field is ordered at random? By randomly ordered we do not mean uniformly distributed, but the salary in a Jane Smith tuple which we are looking at, is likely to be any salary in the salary distribution. The salaries could be distributed as Zipf, exponential, anything. This assumption of random order is likely true to a first order of approximation (in this particular case the assumption appears justified; in general the standing assumption has been that the tuples in the same page are correlated, while the correlation across pages are not as strong, see [5].) If we want to find the median salary of the people in our database, can we use the uncorrelatedness in order information effectively to devise better algorithms? Our algorithms would be "self-checking" in the sense that if any correlation is present, the algorithm would give a coarser guarantee, but it would be no worse than an algorithm that made no assumptions about the order.*

A common thread emerges from the examples—in many scenarios we can relax the assumptions about the order in which the data is seen by an algorithm. This gives rise to the following:

Question. *Can we use the fact that the source generating a data stream is a stochastic process to develop better algorithms? Can we model an adversary that accurately reflects the small scale local changes/correlations that arise in data? Can we develop algorithms that are resilient to such adversary and perform provably better than algorithms devised for all powerful adversaries?*

We answer the questions in the affirmative. We note that this is an extremely natural model and has not been well studied. The notion of a sequence of observations drawn at random exists in learning theory literature. But the primary focus there has not been devising space bounded streaming algorithms, but online algorithms with unbounded space.

1.1 Our contributions

1. We initiate the study of data streams under limited adversarial ordering, a notion we formally define in the next section. We focus on some of the problems raised in the examples and limit our technical discussion to quantile estimations and equidepth histograms. We show that there is significant algorithmic advantage to be derived from limiting the power of the adversary in data streams. As a consequence, for sensor networks and monitoring applications we expect the new class of algorithms to be significantly better and useful in practice.

2. We show that using polylogarithmic space, in a single pass, (where the size of stream is n) for any k, we can find an element of rank to within $k \pm O(k^{\frac{1}{2}+\varepsilon})$ for streams that are ordered by an adversary of limited power. This improves upon the best results known for fully adversarial streams which approximates the median by an element of rank $\frac{n}{2} \pm \epsilon n$ in polylog space. These result have natural implications for Equidepth histograms.

3. We resolve several open questions related to streams under completely random order. Kannan [19, 24] conjectured that a streaming algorithm based on a complete random order, would require only one extra pass in the adversarial model. We resolve this conjecture in the negative. We approximate the median in one pass to a level of accuracy which would require a polynomial blowup in space in the adversarial setting. It also follows that, in complete random order streams, using polylog space, we can find the exact median in $O(\log\log n)$ passes. This was conjectured by Munro and Paterson [23] and has been unresolved since 1980.

1.2 Related Work

Munro and Paterson considered sorting and selection with limited storage [23] in one of the earliest papers on the data stream model, i.e., 1980, close to Morris's work on counting the number of items in sublogarithmic space [22] and the work of Flajolet and Martin on probabilistic counting [8]. The problem has received a lot of attention in recent years starting from the work of Manku, Rajagopalan, and Lindsay [21, 20]. The authors of [21, 20] showed that we can find an element of rank $\frac{n}{2} \pm \epsilon \frac{n}{2}$ using $O(\frac{1}{\epsilon^2}\log n)$ space. This was improved to a deterministic $O(\frac{1}{\epsilon}\log n)$ space algorithm by Greenwald and Khanna [12]. This was extended to a model supporting deletions by Gilbert et al. [11]. Gupta and Zane [14] discussed the issues involved in approximating the kth smallest element in a stream by an element of rank $k \pm \epsilon k$. Exact selection is possible in polylog space if the algorithm may have $O(\log n)$ passes over the data [23].

The complete random permutation or random order model introduced in [23] has received little attention to date. Demaine, López-Ortiz, and Munro [6] considered the frequency estimation of internet packet streams assuming the packets arrive in a random order. Guha, McGregor, and Venkatasubramanian [13] make connections between random order model and various oracle models considered for property testing distributions.

2. DEFINITIONS AND PRELIMINARIES

We first define the notion of a limited adversarial ordering.

DEFINITION 1 (t-BOUNDED ADVERSARY). *Given a stream of elements $x_1, \ldots x_n$ we consider an adversary upstream of our algorithm that can reorder the elements subject to having limited memory to do this reordering. Specifically, a t-*bounded adversary *is an adversary that can only delay at most t elements at a time and therefore can ensure that the stream received is any stream of the form $x_{\sigma(1)}, \ldots x_{\sigma(n)}$ where σ is any permutation such that, for all $i \in [n]$, $|\{j \in [n] : j < i \text{ and } \sigma(i) < \sigma(j)\}| \le t$. E.g. with $t = 2$ the stream 1, 2, 3, 4, 5, 6, 7, 8, 9 can become 3, 2, 1, 6, 5, 4, 9, 8, 7 or 3, 4, 5, 6, 7, 8, 9, 1, 2 but not 9, 8, 7, 6, 5, 4, 3, 2, 1. A stream is t-random if it is generated by a t-bounded adversary acting on a stream whose order is uniformly chosen from all possible orderings.*

Therefore a fully adversarially ordered stream is $(n-1)$-random and a totally randomly ordered stream is 0-random. We now make the following definition regarding sub-streams of the data stream.

DEFINITION 2. *Rank of an item x in a set $\mathcal{S}$ is defined as* RANK$[x, \mathcal{S}] = |\{x' | x' \le x, x' \in \mathcal{S}\}|$. *Given any (j', a, b) define $\Gamma(j', a, b) = \{x_j | j > j', a \le x_j \le b\}$.*

The following lemma follows immediately from the above definitions.

LEMMA 1. *Consider a t-random stream and $\Gamma(j', a, b)$. A length w contiguous sub-stream S' of $\Gamma(j', a, b)$ is of the form $(A \cup B) \setminus C$ where A is a random size w subset of $\Gamma(j', a, b)$ and $B \subset \Gamma(j', a, b), C \subset A$ have size $|B| = |C| \le t$.*

3. ONE PASS APPROXIMATE SELECTION

In this section we show how to perform approximate selection in a t-random stream. We will present the algorithm assuming the exact value of the length of the stream, n, is known in advance. In the following sections we will show that this assumption is not necessary and also improve the bounds when trying to select low rank elements.

Our algorithm proceeds in "phases" each composed of two distinct "sub-phases." In each phase we aim to narrow the area of our search such that, subsequently, we are only looking in a sub-stream of values in a specific range. In general, the relative rank of the element we look for in the sub-stream will not be the same as the rank of the element we were initially looking for. So we need to also determine the rank of the element we are now looking for. See Fig. 1 for the algorithm.

LEMMA 2. *At the end of the sampling phase, whp, we return S that includes $u, v \in \Gamma(i)$ such that*[2] $k(i) - \epsilon\Gamma(i) \le$ RANK$(u, \Gamma(i)) \le k(i) \le$ RANK$(v, \Gamma(i)) \le k(i) + \epsilon\Gamma(i)$.

PROOF. We prove there exists, with probability $> 1 - 2/n$, an element $u \in S$ with the required properties. The proof for v is identical. Assume $k(i) \ge \epsilon\Gamma(i)$ otherwise LB_i suffices. Consider the set S' of the first $48(\log n + t)/\epsilon$ elements of $\Gamma(i)$. Let X be the set of elements $x \in S'$ such that $k(i) - \epsilon\Gamma(i) \le$ RANK$(x, \Gamma(i)) \le k(i)$. Now $E(X) \ge \epsilon|S'| - t$ and, using the Chernoff bound,

$$\begin{aligned} & \Pr\left(|X| < \frac{\epsilon}{2}|S'|\right) \\ = \;& \Pr\left(|X| < (1 - 1/4)E(X)\right) \\ \le \;& \exp(-(48\log n + 47t)/48) \\ \le \;& 1/n \end{aligned}$$

Then $\Pr\left(X \cap S = \emptyset \middle| |X| > \frac{\epsilon}{2}|S'|\right) \le (1-\epsilon/2)^{2\log n/\epsilon} \le 1/n$. □

We now ascertain that our estimate GAP(i) for $\Gamma(i)$ is sufficiently accurate.

LEMMA 3 (MIND THE GAP). *Whp,* $|$GAP$(i) - \Gamma(i)| \le c_3(s' + t + \sqrt{n\log n})$ *where* $c_3 = \frac{8}{\sqrt{c_1}(1-4\epsilon)}$.

PROOF. The proof is by induction. The base case, $i = 1$, follows since $|\Gamma(1)| = n$ and we are assuming that we know n exactly. (As noted, we will preempt this assumption in the next section.) The induction hypothesis is that $|$GAP$(i) - \Gamma(i)| \le c_3(s' + t + \sqrt{n\log n})$.

Define $\Delta_C(i) = C(i, \ell+1) - C(i, \ell)$ and let μ be the number of items between $y_{i\ell}$ and $y_{i(\ell+1)}$ in S'. Note that $\mu \le s'$. It follows that,

$$\begin{aligned} & |\Gamma(s(i), y_{i\ell}, y_{i(\ell+1)})| \\ = \;& \mu + \Delta_C(i) + |\Gamma(s(i+1), y_{i\ell}, y_{i(\ell+1)})| \\ = \;& \mu + \Delta_C(i) + \Gamma(i+1)\ , \end{aligned}$$

[2]Note that here and henceforth we abuse notation and denote $|\Gamma(i)|$ as $\Gamma(i)$.

Selection Algorithm:

1. Let $c_1 = 1/10, c_2 = (1/c_1 + 9\frac{8}{\sqrt{c_1(1-\epsilon)}}), s' = 48\epsilon^{-1}(\log n + t)$ and $s = 2\epsilon^{-1}\log n$.
2. Each phase i starts corresponding to the $s(i)$th item in the stream. Initially $s(1) = 1$.
3. We maintain two bounds LB_i, and UB_i which are bounds on the range of elements in which we are interested. Initially $LB_1 = -\infty$ and $UB_1 = \infty$.
4. We maintain a value $k(i)$ which indicates the rank of the element we are searching for in the set $\Gamma(i) = \Gamma(s(i), LB_i, UB_i)$. Initially $k(1) = k$.
5. We maintain an estimate, $\textsc{Gap}(i)$, of $|\Gamma(i)|$. Initially $\textsc{Gap}(1) = n$.
6. Phase i:
   (a) If $\textsc{Gap}(i) \le \frac{1}{c_1}(\sqrt{n\log n} + t + s')$ output LB_i and UB_i. Both of them are good approximations for the element of rank k in the original stream and we stop.
   (b) *Sample* sub-phase: From the next s' elements, call them S', we sample s elements. Call the sampled elements $y_{i1} \le y_{i2} \le \ldots \le y_{is}$. Let $y_{i0} = LB_i$ and $y_{i(s+1)} = UB_i$. Call the set $S = \{y_{i\ell}\}_{0\le\ell\le s+1}$.
   (c) *Estimate* sub-phase: Maintain $s+1$ counters $(C(i,\ell))_{1\le\ell\le s+1}$. For the next $\alpha(i) = c_1\textsc{Gap}(i)$ elements, if an element is smaller than $y_{i\ell}$ then increment $C(i,\ell)$.
   (d) If for some $y_{i\ell}$ we have $\left|\frac{1}{c_1}C(i,\ell) - k(i)\right| \le c_2\sqrt{n\log n}$ we have no more phases.
   (e) Set $LB_{i+1} = y_{i\ell}$ and $UB_{i+1} = y_{i(\ell+1)}$ where $y_{i\ell}, y_{i(\ell+1)}$ s.t. $\frac{1}{c_1}C(i,\ell) \le k(i) \le \frac{1}{c_1}C(i,\ell+1)$.
   (f) Set $k(i+1) = (1-c_1)\left(k(i) - \frac{C(i,\ell)}{c_1}\right)$ and $\textsc{Gap}(i+1) = (C(i,\ell+1) - C(i,\ell))\left(\frac{\textsc{Gap}(i)}{\alpha(i)} - 1\right)$.

Figure 1: The Selection Algorithm

since there are $\Delta_C(i) + \mu$ elements in the stream starting at $s(i)$ and ending at $s(i+1)$ whose values are between $y_{i\ell}$ and $y_{i(\ell+1)}$. Now, using the Chernoff bound and Lemma 1 we obtain,

$$\Pr\left(\left|\frac{\Delta_C(i)}{\alpha(i)} - \frac{\Delta_C(i) + \Gamma(i+1)}{\Gamma(i)}\right| > \frac{2t}{\alpha(i)} + \frac{s'}{\Gamma(i)} + 8\sqrt{\frac{\log n}{\alpha(i)}}\right) \le \frac{1}{n} \quad .$$

Therefore, with probability at least $1 - 1/n$,

$$\begin{aligned} & 8\sqrt{\frac{\log n}{\alpha(i)}} + \frac{2t}{\alpha(i)} + \frac{s'}{\Gamma(i)} \\ \ge\ & \left|\frac{\Delta_C(i)}{\alpha(i)} - \frac{\Delta_C(i) + \Gamma(i+1)}{\Gamma(i)}\right| \\ =\ & \frac{1}{\Gamma(i)}\left|\Delta_C(i)\left(\frac{\Gamma(i)}{\alpha(i)} - 1\right) - \Gamma(i+1)\right| \\ \ge\ & \frac{1}{\Gamma(i)}\left|\Delta_C(i)\left(\frac{\textsc{Gap}(i)}{\alpha(i)} - 1\right) - \Gamma(i+1)\right| \\ & -\frac{\Delta_C(i)}{\Gamma(i)}\frac{|\textsc{Gap}(i) - \Gamma(i)\ |}{\alpha(i)} \quad . \end{aligned}$$

The last inequality follows by the triangle inequality. Substituting $\textsc{Gap}(i+1) = \Delta_C(i)\left(\frac{\textsc{Gap}(i)}{\alpha(i)} - 1\right)$ and rearranging gives,

$$\begin{aligned} & |\textsc{Gap}(i+1) - \Gamma(i+1)| \\ \le\ & s' + \frac{2t\Gamma(i)}{\alpha(i)} + \Gamma(i)8\sqrt{\frac{\log n}{\alpha(i)}} + \frac{\Delta_C(i)}{\alpha(i)}|\textsc{Gap}(i) - \Gamma(i)\ | \\ \le\ & s' + \frac{2t\textsc{Gap}(i)}{\alpha(i)} + 8\textsc{Gap}(i)\sqrt{\frac{\log n}{\alpha(i)}} \\ & +|\textsc{Gap}(i) - \Gamma(i)\ |\left(\frac{2t}{\alpha(i)} + \frac{\Delta_C(i)}{\alpha(i)} + 8\sqrt{\frac{\log n}{\alpha(i)}}\right) \\ \le\ & s' + \frac{2t}{c_1} + \frac{8}{\sqrt{c_1}}\sqrt{n\log n} \\ & +c_3(s' + t + \sqrt{n\log n})(\epsilon + \epsilon + 2\epsilon) \quad . \end{aligned}$$

The last line follows since $\Delta_C(i)/\alpha(i) \le \epsilon$ by the Chernoff bound and $\alpha(i) \ge \sqrt{n\log n} + t + s'$. Therefore $c_3 = \frac{8}{\sqrt{c_1}(1-4\epsilon)}$ is sufficient to ensure $|\textsc{Gap}(i+1) - \Gamma(i+1)| \le c_3(s' + t + \sqrt{n\log n})$. $\square$

Observe that we do not run out of elements before finding an element whose rank appears to be close to the desired rank. The following lemma quantifies how the error accumulates with each phase.

LEMMA 4. *Whp,* $\textsc{Rank}(y_{i\ell}, \Gamma(i)) = \frac{C(i,\ell)}{c_1} \pm c_4(\sqrt{n\log n} + t + s')$ *where* $c_4 = (\frac{8}{c_1} + 9c_3)$.

PROOF. In each phase we estimate $\hat{y}_i = \textsc{Rank}(y_{i\ell}, \Gamma(i))$ as $\frac{1}{c_1}C(i,\ell)$. Using Lemma 1 and the Chernoff bound we can see that with high probability,

$$\left|\frac{\hat{y}_i}{\Gamma(i)}\alpha(i) - C(i,\ell)\right| \leq 8\sqrt{\alpha(i)\log n} + t + s' \ ,$$

where s' is an upper bound on the number of elements less than $y_{i\ell}$ in S'. Therefore,

$$\begin{aligned}
& \left|\hat{y}_i - \frac{1}{c_1}C(i,\ell)\right| \\
\leq\ & \left|\hat{y}_i - \frac{\Gamma(i)}{\alpha(i)}C(i,\ell)\right| + \left|\frac{\Gamma(i)}{\alpha(i)}C(i,\ell) - \frac{1}{c_1}C(i,\ell)\right| \\
\leq\ & \frac{\Gamma(i)}{\alpha(i)}(8\sqrt{\alpha(i)\log n} + t + s') \\
& + \left|\frac{\Gamma(i)}{\alpha(i)}C(i,\ell) - \frac{\textsc{Gap}(i)}{\alpha(i)}C(i,\ell)\right| \\
\leq\ & \frac{|\textsc{Gap}(i)| + |\textsc{Gap}(i) - \Gamma(i)|}{\alpha(i)}(8\sqrt{\alpha(i)\log n} + t + s') \\
& + |\Gamma(i) - \textsc{Gap}(i)| \\
\leq\ & \left(\frac{1}{c_1} + c_3\frac{\sqrt{n\log n} + t + s'}{\alpha(i)}\right)(8\sqrt{\alpha(i)\log n} + t + s') \\
& + c_3(\sqrt{n\log n} + t + s') \\
\leq\ & \left(\frac{8}{c_1} + 9c_3\right)(\sqrt{n\log n} + t + s') \ ,
\end{aligned}$$

where the last two inequalities use Lemma 3 and the fact that $\alpha(i) \geq \sqrt{n\log n} + t + s'$ respectively. □

THEOREM 1. *If we know the exact length of the stream, then given any k, in a single pass over a random stream we can find an element of rank $k \pm O(n^{\epsilon'}(\sqrt{n} + t + s'))$ with high probability using polylog space. The constant depends exponentially on ϵ'.*

PROOF. Let x be the value returned be the algorithm. Let $\hat{k}(i) = \textsc{Rank}(x, \Gamma(i))$. Whp, the number of elements between x and $y_{i\ell}$ in the $\alpha(i)$ elements we saw in phase i is $c_1(\hat{k}(i) - \hat{y}_i) \pm O(\sqrt{n\log n} + t + s')$. Therefore,

$$\hat{k}(i) = \hat{k}(i+1) + \hat{y}_i + c_1(\hat{k}(i) - \hat{y}_i) \pm O(\sqrt{n\log n} + t + s') \ . \quad (1)$$

Lemma 4 implies that $\hat{y}_i = \frac{1}{c_1}C(i,\ell) \pm O(\sqrt{n\log n} + t + s')$ and by definition $k(i+1) = (1-c_1)(k(i) - \frac{1}{c_1}C(i,\ell))$. Therefore from Eq. 1 we deduce,

$$\begin{aligned}
& (1-c_1)|\hat{k}(i) - k(i))| \\
=\ & |\hat{k}(i+1) - (1-c_1)k(i) + (1-c_1)\hat{y}_i| \\
& \pm O(\sqrt{n\log n} + t + s') \\
=\ & |\hat{k}(i+1) - k(i+1)| \pm O(\sqrt{n\log n} + t + s') \ .
\end{aligned}$$

Let the algorithm terminate in phase p. Observe that since the number of elements decrease by a factor of at least 2ϵ, $p \leq (\log n)/\log(1/2\epsilon)$. If $\epsilon < \frac{1}{2}(1-c_1)^{1/\epsilon'}$, then the above recurrence relation yields,

$$\begin{aligned}
|\hat{k}(1) - k(1)| \ \leq\ & O\left(\frac{\sqrt{n\log n} + t/\epsilon}{(1-c_1)^p}\right) \\
=\ & O\left(n^{\frac{\log\frac{1}{1-c_1}}{\log(1/2\epsilon)}}(\sqrt{n\log n} + t + s')\right) \\
\leq\ & O\left(n^{\epsilon'}(\sqrt{n} + t + s')\right) \ .
\end{aligned}$$

□

Note: We can alter the algorithm to continue estimating the ranks in $\Gamma(i)$ of all the elements returned in the sample at the ith phase. This allows us to "self–correct" in the sense that we will always output elements with ranks that sandwich the desired rank.

3.1 Generalizing to Unknown Stream Lengths

The algorithm in the previous section assumed that we know the precise value of n, the length of the stream. As this is not usually the case we now discuss a way around this assumption. First we argue that, for our purposes, it is sufficient to only look at half of the stream.

LEMMA 5. *Assume the stream is t-random. Let the set of values in the entire stream be S and let S' be the values in a contiguous sub-stream of length $\tilde{n} \geq n/2$. Then whp, the $\tilde{k}$th smallest element of $\tilde{S}$ has rank $k = \frac{\tilde{k}}{\tilde{n}}n \pm 2(\sqrt{8\tilde{k}\log n} + 4t)$.*

PROOF. Let $a = \tilde{k}/4\tilde{n}$. Let the elements in S be $v_1 \leq v_2 \leq \ldots \leq v_n$. Let $X = |\{v_1, \ldots v_{an+b}\} \cup S'|$ and $Y = |\{v_1, \ldots v_{an-b-1}\} \cup S'|$. Let X' and Y' be random variables distributed as $\mathbf{Bin}(\tilde{n}, a + b/n)$ and $\mathbf{Bin}(\tilde{n}, a - (b+1)/n)$ respectively. If $b = 2(\sqrt{8\tilde{k}\log n} + 4t)$ then the probability that the element of rank $a\tilde{n}$ in S' has rank in S outside the range $[an - b, an + b]$ is less than,

$$\begin{aligned}
& \Pr(X < a\tilde{n}) + \Pr(Y > a\tilde{n}) \\
\leq\ & \Pr\left(X' < a\tilde{n} + t\right) + \Pr\left(Y' > a\tilde{n} - t\right) \\
\leq\ & \Pr\left(X' < E\left(X'\right) + t - b/2\right) \\
& + \Pr\left(Y' > E\left(Y'\right) - t + b/2\right) \\
\leq\ & 2\exp\left(-(b/2 - t)^2/(3(a\tilde{n} + b))\right) \leq 1/n \ .
\end{aligned}$$

□

To get around not knowing n we make multiple instantiations of the algorithm presented in the previous section. Each instantiation corresponds to a guess of n. Let $\beta = 1.5$. Instantiation i guesses a length of $\lceil 4\beta^i \rceil - \lfloor \beta^i \rfloor + 1$ and is run on the stream starting with the $\lfloor \beta^i \rfloor$th data item and ending with the $\lceil 4\beta^i \rceil$th data item. We remember the result of the algorithm until the $2(\lceil 4\beta^i \rceil - \lfloor \beta^i \rfloor + 1)$th element arrives. We say the instantiation has been canceled at this point.

LEMMA 6. *At any given time there are only a constant number of instantiations. Furthermore, whenever the stream terminates, at least one instantiation has run on a sub-stream of at least half the total length.*

PROOF. Consider the tth element of the data stream. By this point there have been $O(\log_\beta t)$ instantiations made. However, $\Omega(\log_\beta t/6)$ instantiations have been canceled. Hence $O(\log_\beta t - \log_\beta t/6) = O(1)$ instantiations are running. We now show that there always exists a "good" instantiation, i.e. one which has been running on at least half the stream. The ith instantiation gives a useful result if the length of the stream $n \in U_i = \{\lfloor 4\beta^i \rfloor + 1, \ldots, 2(\lceil 4\beta^i \rceil - \lfloor \beta^i \rfloor + 1)]$. But $\bigcup_{i \geq 0} U_i = \mathbb{N} \setminus \{0,1,2,3,4\}$ since for all $i > 1$, $\lfloor 4\beta^i + 1 \rfloor \leq 2(\lceil 4\beta^{i-1} \rceil - \lfloor \beta^{i-1} \rfloor + 1)$. □

We can therefore generalize Theorem 1 as follows,

THEOREM 2. *Whp, given any k, in a single pass over a random stream we can find an element of rank $k \pm O(n^{1/2+\epsilon'})$.*

3.2 Approximate Selection for Low Rank Items

We now consider the problem of finding elements with small rank, i.e. $k = o(n)$. Assume that k, the rank of the element to be selected, is at least $\epsilon^{-1} \log n$ since otherwise storing the $\epsilon^{-1} \log n$ smallest elements is sufficient for exact selection. The algorithm is in two parts: 1) In the first half of the stream we find an element u which we will prove has rank, in the entire stream, between k and $10k$. To choose u we sample $(n \log n)/k$ elements and store the $(10 \log n)$ smallest elements from this sample. Let u be the $(10 \log n)$th smallest element. 2) In the second half of the stream we run the algorithm in Section 3 with $UB_1 = u$.

LEMMA 7. *Whp, the rank of u in the entire stream is $20k \pm 2(6k + \sqrt{128k \log n} + 4t)$.*

PROOF. We first prove that the rank of u in the first half of the stream is $10k \pm 8k$. Let the elements in the first half of the stream be $S_1 = v_1 \leq v_2 \leq \ldots \leq v_{n/2}$. Let $w = (n \log n)/k$. Let the sample of w elements from the first half of the stream be S'. Let $X = |\{v_1, \ldots, v_{10k+b}\} \cap S'|$ and $Y = |\{v_1, \ldots, v_{10k-b-1}\} \cap S'|$. Then $E(X) = (10k + b)2w/n = 10 \log n + (b \log n)/k$ and $E(Y) = (10k - b - 1)2w/n = 10 \log n - ((b + 1) \log n)/k$. If $b = 8k$ then probability that the $10 \log n$ th smallest element of S' has rank (in S_1) outside the range $[10k - b, 10k + b]$ is less than,

$$\begin{aligned} & \Pr(X < 10 \log n) + \Pr(Y > 10 \log n) \\ \leq \quad & \Pr\left(X < E(X) - \frac{b \log n}{k}\right) \\ & \quad + \Pr\left(Y > E(Y) + \frac{b \log n}{k}\right) \\ \leq \quad & 2 \exp\left(-(b \log n/k)^2 / (3(10 \log n + (b \log n)/k))\right) \\ \leq \quad & 1/n \ . \end{aligned}$$

Then by appealing to Lemma 5 we get the result claimed. □

THEOREM 3. *Whp, given any k, in a single pass over a random stream we can find an element of rank $k \pm O(k^{1/2+\epsilon'})$.*

4. CONSEQUENCES

4.1 Exact Selection in Multiple Passes

A natural question arises from the one pass polylog space $\pm O(n^{1/2+\epsilon'})$ approximation. Observe that before the start of the stream the element sought after could have been one of n elements. After the pass we have an element whose rank is within $O(n^{1/2+\epsilon'})$ of the element which we were seeking. If we view the number of candidate elements, that number followed a recursion $n \to n^{1/2+\epsilon'}$. Thus after $O(\log \log n)$ passes we can expect to arrive at a point where we have $O(\log^6 n)$ candidates and we solve the problem by brute memory at that point. The idea as stated does not *immediately* work. In the first pass we are looking at a random permutation; but after the first pass how do we ensure that the random permutation property—particularly since we have already looked at the numbers, how do we ensure that the numbers are random? The core of the argument is the following lemma. It is crucial, but is almost proven by straightforward inspection.

LEMMA 8. *Consider the elements S between LB_{r+1}, UB_{r+1}, after the pass. Let the input order be π. Consider another order π' where all elements of S are permuted among themselves, but all elements not in S are not touched. Then the algorithm will also end up in the same values for LB_{r+1}, UB_{r+1} with input π' instead of π.*

From the above we can easily see that the probability of observing a particular permutation of S which is conditioned on LB_{r+1}, UB_{r+1}, is exactly the same as observing any other permutation of S. Therefore conditioned on LB_{r+1}, UB_{r+1} the numbers between the bounds are in random order. Therefore we can bootstrap at this point, and the two stage algorithm will have an approximation guarantee of $O(n^{1/4+\epsilon'})$. Repeating this process $\Theta(\log \log n)$ times we get the following:

THEOREM 4. *Given any k, in $O(\log \log n)$ passes over a random stream we can find an element of rank k with a high probability in polylog space.*

4.2 Equi-Depth Histograms

Equidepth Histograms use quantiles to approximates the data. The B buckets are defined by the B-quantiles of the data, i.e., the items of rank $n/(B+1), \ldots, nB/(B+1)$. One of the measures of goodness proposed by Gibbons, Matias, and Poosala [10] in this context is by how much does the true rank of the ith bucket boundaries differ from $in/(B+1)$—in particular they consider $1/n$ times the square–root of the average sum of squares of the deviation, i.e., if the error in rank is ϵ_i then the measure is $\mu = n^{-1}\sqrt{B^{-1} \sum_i \epsilon_i^2}$. The authors of [10] show that the above measure can be made smaller than any constant $\epsilon > 0$ but did not decrease as n was increased. As a consequence of a better quantile approximation, we can show that in their setting, which allows random order, the $\epsilon_i \leq n^{\frac{1}{2}+\varepsilon}$ and therefore $\mu' = O(1/n^{\frac{1}{2}-\varepsilon})$ and this tends to zero as we increase the size of the backing sample.

5. LOWER BOUNDS

We now present a lower bound for approximate median finding in the adversarial model. It is generalization of a result from [17, 23] and involves a reductions from communication complexity results. The significance of this lower bound is that it shows a strict separation between the random order streaming model and the adversarial order streaming model.

THEOREM 5. *Finding the median of an adversarially ordered stream in a single pass requires $\Omega(n)$ space. Finding an n^δ approximate median of an adversarially ordered stream requires $\Omega(n^{1-\delta})$ space.*

PROOF. The proof uses a reduction from INDEXING.

> Let Alice have a length n binary string $x = x_1 \ldots x_n$ unknown to Bob and Bob has an index $j \in [n]$ unknown to Alice. Bob wishes to learn the value of x_j. Then, if only one way communication is permitted from Alice to Bob, if Bob is to learn x_j with probability at least $1 - 1/100$, Alice must send $\Omega(n)$ bits.

Assume that $n-1$ is a multiple of 2. Consider an instance (x, j) of INDEXING where $x = x_1 x_2 \ldots x_{(n-1)/2}$ is a length $(n-1)/2$ binary string and $j \in [(n-1)/2]$. Suppose there exists a streaming algorithm $\mathcal{A}$ that finds the median of a length n stream with space $S(n)$. The $\mathcal{A}$ gives rise to a protocol for INDEXING as follows: Alice simulates $\mathcal{A}$ on $\{2i + x_i : i \in [(n-1)/2]\}$. She then transmits the memory state of $\mathcal{A}$ to Bob. Bob initializes $\mathcal{A}$ with this memory state and continues running the algorithm on $(n+1)/2 - j$ copies of 0 and then j copies of $(n-1)$. Notice that the least significant bit of the median of this set of values equals x_j. Hence $S(n) = \Omega(n)$.

To further prove a lower bound for approximate median finding we use an idea similar to that of *streaming reductions* that was introduced by Bar-Yossef et al. [4]. Consider an instance of exact median finding in a stream S of length n. From above, this requires $\Omega(n)$ space. We reduce this problem to finding an approximate median in an induced string S' formed by repeating each element of S, b times. S' is a length nb stream. Let $\mathcal{A}$ be an algorithm that finds an $|S'|^\delta$ approximate median. Now an $(nb)^\delta$-approximate median of S' takes the same value as the exact median of S if $b \geq (nb)^\delta$ i.e. if $b \geq n^{\delta/(1-\delta)}$. In this case the stream length is $|S'| = n^{1/(1-\delta)}$ and we know that we require $\Omega(n)$ space to find the approximate median. The theorem follows. $\square$

6. ACKNOWLEDGEMENTS

We thank Anupam Gupta for suggesting that we consider the case when k is $o(n)$.

7. REFERENCES

[1] D. J. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. B. Zdonik. Aurora: a new model and architecture for data stream management. *VLDB J.*, 12(2):120–139, 2003.

[2] A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, R. Motwani, I. Nishizawa, U. Srivastava, D. Thomas, R. Varma, and J. Widom. Stream: The Stanford stream data manager. *IEEE Data Engineering Bulletin*, 26(1):19–26, 2003.

[3] R. Avnur and J. M. Hellerstein. Eddies: Continuously adaptive query processing. *Proc. of SIGMOD*, pages 261–272, 2000.

[4] Z. Bar-Yossef, T. S. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan. Counting distinct elements in a data stream. *Proc. of RANDOM*, pages 1–10, 2002.

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

[6] E. D. Demaine, A. López-Ortiz, and J. I. Munro. Frequency estimation of internet packet streams with limited space. In R. H. Möhring and R. Raman, editors, *ESA*, volume 2461 of *Lecture Notes in Computer Science*, pages 348–360. Springer, 2002.

[7] A. Deshpande, C. Guestrin, S. Madden, J. M. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. *Proc. of VLDB Conference (Best Paper)*, pages 588–599, 2004.

[8] P. Flajolet and G. Martin. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences*, 31:182–209, 1985.

[9] P. Gibbons and Y. Matias. Synopsis data structures for massive data sets. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science: Special Issue on External emory Algorithms and Visualization*, A:39–70, 1999.

[10] P. B. Gibbons, Y. Matias, and V. Poosala. Fast incremental maintenance of approximate histograms. *ACM Trans. Database Syst.*, 27(3):261–298, 2002.

[11] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. How to summarize the universe: Dynamic maintenance of quantiles. In *Proc. 28th International Conference on Very Large Data Bases*, pages 454–465, 2002.

[12] M. Greenwald and S. Khanna. Space-efficient online computation of quantile summaries. *SIGMOD Conference*, 2001.

[13] S. Guha, A. McGregor, and S. Venkatasubramanian. Streaming and sublinear approximation of entropy and information distances. In *SODA*, pages 733–742. ACM Press, 2006.

[14] A. Gupta and F. Zane. Counting inversions in lists. *Proc. of SODA*, pages 253–254, 2003.

[15] P. Haas and A. Swami. Sequantial Sampling Procedures for Query Size Estimation. *Proc. of ACM SIGMOD, San Diego, CA*, pages 341–350, June 1992.

[16] P. J. Haas, J. F. Naughton, S. Seshadri, and L. Stokes. Sampling-based estimation of the number of distinct values of an attribute. In *VLDB*, pages 311–322, 1995.

[17] M. R. Henzinger, P. Raghavan, and S. Rajagopalan. Computing on data streams. *Technical Report 1998-001, DEC Systems Research Center*, 1998.

[18] Z. G. Ives, A. Y. Halevy, and D. S. Weld. Adapting to source properties in processing data integration queries. *Proc. of SIGMOD*, pages 395–406, 2004.

[19] S. Kannan. Open problems in streaming. *PPT Slides, (request source)*.

[20] G. S. Manku, S. Rajagopalan, and B. Lindsay. Random sampling techniques for space efficient online computation of order statistics of large datasets. *Proc. of SIGMOD*, pages 251–262, 1999.

[21] G. S. Manku, S. Rajagopalan, and B. G. Lindsay. Approximate medians and other quantiles in one pass and with limited memory. In *SIGMOD*, pages 426–435, 1998.

[22] R. Morris. Counting large numbers of events in small registers. *CACM*, 21(10):840–842, 1978.

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

[24] S. Muthukrishnan. Data streams: Algorithms and applications. *Survey available on request at* `muthu@research.att.com`, 2003.