Quantiles and Equidepth Histograms over Streams

Michael B. Greenwald[1] and Sanjeev Khanna[2]

[1] Arastra, Inc., 275 Middlefield Road, Menlo Park, CA 94025
`greenwald@cis.upenn.edu`
[2] University of Pennsylvania, Dept. of Computer and Info. Science, 3330 Walnut
Street, Philadelphia, PA 19104 `sanjeev@cis.upenn.edu`

1 Introduction

A quantile query over a set S of size n, takes as input a quantile $\phi, 0 < \phi \leq 1$, and returns a value $v \in S$, whose rank in the sorted S is ϕn. Computing the median, the 99-percentile, or the quartiles of a set are examples of quantile queries. Many database optimization problems involve approximate quantile computations over large data sets. Query optimizers use quantile estimates to estimate the size of intermediate results and choose an efficient plan among a set of competing plans. Load balancing in parallel databases can be done by using quantile estimates. Above all, quantile estimates can give a meaningful summary of a large data set using a very small memory footprint. For instance, given any data set, one can create a data structure containing 50 observations, that can answer any quantile query to within 1% precision in rank.

Based on the underlying application domain, a number of desirable properties can be identified for quantile computation. In this survey, we will focus on the following three properties: (a) space used by the algorithm; (b) guaranteed accuracy to within a pre-specified precision; and (c) number of passes made.

It is desirable to compute quantiles using the smallest memory footprint possible. We can achieve this by dynamically storing, at any point in time, only a summary of the data seen so far, and not the entire data set. The size and form of such summaries are determined by our *a priori* knowledge of the types of quantile queries we expect to be able to answer. We may know, in advance, that the client intends to ask for a single, specific, quantile. Such a single quantile summary, is parameterized in advance by the quantile, ϕ, and a desired precision ϵ. For any $0 < \phi \leq 1$, and $0 \leq \epsilon \leq 1$, an ϵ-approximate ϕ-quantile on a data set of size n, is any value v whose rank, $r^*(v)$, is guaranteed to lie between $n(\phi - \epsilon)$ and $n(\phi + \epsilon)$. For example, a .01-approximate .5-quantile is any value whose rank is within 1% of the median.

Alternatively, we may know that the client is interested in a range of equally-spaced quantile queries. In such cases we summarize the data by an *equi-depth histogram*. An equi-depth histogram is parameterized by a bucket size, ϕ, and a precision, ϵ. The client may request any, or all, ϕ-quantiles, that is, elements of ranks, $\phi n, 2\phi n, ..., n$. We say that $H(\phi, \epsilon)$ is an ϵ-*approximate equi-depth histogram* with bucket width ϕ if for any $i = 1$ to $1/\phi$, it returns a value $v_{i\phi}$ for the $i\phi$ quantile, where $n(i\phi - \epsilon) \leq r^*(v_{i\phi}) \leq n(i\phi + \epsilon)$.

Finally, we may have no prior knowledge of the anticipated queries. Such ϵ-*approximate quantile summaries* are parameterized only by a desired precision ϵ. We say that a quantile summary $Q(\epsilon)$ is ϵ-*approximate* if it can be used to answer *any* quantile query to within a precision of ϵn. There is a close relation between equi-depth histograms and quantile summaries. $Q(\epsilon)$ can serve as an equi-depth histogram $H(\phi, \epsilon)$ for any $0 < \phi \leq 1$. Conversely, an equi-depth histogram $H(\phi, \epsilon)$ is a ϕ-approximate quantile summary, provided only that $\epsilon \leq \phi/2$.

Organization: The rest of this chapter is organized as follows. In Section 2, we formally introduce the notion of an approximate quantile summary, and some simple operations that we will use to describe various algorithms for maintaining quantile summaries. Section 3 describes deterministic algorithms for exact selection and for computing approximate quantile summaries. These algorithms give worst-case deterministic guarantees on the accuracy of the quantile summary. In contrast, Section 4 describes algorithms with probabilistic guarantees on the accuracy of the summary.

2 Preliminaries

We will assume throughout that the data is presented on a read-only tape where the tape head moves to the right after each unit of time. Each move of the tape head reveals the next observation (element) in the sequence stored on the tape. For convenience, we will simply say that a new observation arrives after each unit of time. We will use n to denote both the number of observations (elements of the data sequence) that have been seen so far as well as the current time. Almost all results presented here concern algorithms that make a single pass on the data sequence. In a multi-pass algorithm, we assume that at the beginning of each pass, the tape head is reset to the leftmost cell on the tape. We assume that our algorithms operate in a RAM model of computation. The space $s(n)$ used by an algorithm is measured in terms of the maximum number of words used by an algorithm while processing an input sequence of length n. This model assumes that a single word can store $\max\{n, |v^*|\}$ where v^* is the observation with largest absolute value that appears in the data sequence.

The set-up as described above concerns an "insertion-only" model that assumes that an observation once presented is not removed at a later time

from the data sequence. This is referred to as the *cash register* model in the literature [7]. A more general setting is the *turnstile* model [17] that also allows for deletion of observations. In Section 5 we will consider algorithms for this more general setting as well. We note here that a simple modification of the model above can be used to capture the turnstile case: the ith cell on the read-only tape, contains both the ith element in the data sequence and an additional bit that indicates whether the element is being inserted or deleted.

An order-statistic query over a data set S takes as input an integer $r \in [1..|S|]$ and outputs an element of rank r in S. We say that the order-statistic query is answered with ϵ-accuracy if the output element is guaranteed to have rank within $r \pm \epsilon n$. For simplicity, we will assume throughout that ϵn is an integer. If $1/\epsilon$ is an integer, then this is easily enforced by batching observations $1/\epsilon$ at a time. If $1/\epsilon$ is not an integer, then let i be an integer such that $1/2^{i+1} < \epsilon < 1/2^i$. We can then replace the ϵ-accuracy requirement by $\epsilon' = 1/2^{i+1}$ which is within a factor of two of the original requirement.

2.1 Quantile Summary

Following [10], we define a *quantile summary* for a set S to be an ordered set $Q = \{q_1, q_2, ..., q_\ell\}$ along with two functions $\texttt{rmin}_Q$ and $\texttt{rmax}_Q$ such that

(i) $q_1 \leq q_2... \leq q_\ell$ and $q_i \in S$ for $1 \leq i \leq \ell$.
(ii) For $1 \leq i \leq \ell$, each q_i has rank at least $\texttt{rmin}_Q(q_i)$, and at most $\texttt{rmax}_Q(q_i)$ in S.
(iii) Finally, q_1 and q_ℓ are the smallest and the largest elements, respectively, in the set S, that is, $\texttt{rmin}_Q(q_1) = \texttt{rmax}_Q(q_1) = 1$, and $\texttt{rmin}_Q(q_\ell) = \texttt{rmax}_Q(q_\ell) = |S|$.

We will say that Q is a *relaxed quantile summary* if satisfies properties (i) and (ii) above, and the following relaxation of property (iii): $\texttt{rmax}_Q(q_1) \leq \epsilon|S|$ and $\texttt{rmin}_Q(q_\ell) \geq (1 - \epsilon)|S|$.

We say that a summary Q is an *ϵ-approximate quantile summary* for a set S if it can be used to answer any order statistic query over S with ϵ-accuracy. That is, it can be used to compute the desired order-statistic within a rank error of at most $\epsilon|S|$. The proposition below describes a sufficient condition on the function $\texttt{rmin}_Q$ and $\texttt{rmax}_Q$ to ensure an ϵ-approximate summary.

Proposition 1 ([10]) *Let Q be a relaxed quantile summary such that it satisfies the condition $\max_{1 \leq i < \ell}(\texttt{rmax}_Q(q_{i+1}) - \texttt{rmin}_Q(q_i)) \leq 2\epsilon|S|$. Then Q is an ϵ-approximate summary.*

Proof. Let $r = \lceil \phi|S| \rceil$. We will identify an index i such that $r - \epsilon|S| \leq \texttt{rmin}_Q(q_i)$ and $\texttt{rmax}_Q(q_i) \leq r + \epsilon|S|$. Clearly, such a value q_i approximates the ϕ-quantile to within the claimed error bounds. We now argue that such an index i must always exist.

Let $e = \max_i(\texttt{rmax}_Q(q_{i+1}) - \texttt{rmin}_Q(q_i))/2$. Consider first the case $r \geq |S| - e$. We have $\texttt{rmin}_Q(q_\ell) \geq (1 - \epsilon)|S|$, and therefore $i = \ell$ has the desired property. We now focus on the case $r < |S| - e$, and start by choosing the smallest index j such that $\texttt{rmax}_Q(q_j) > r + e$. If $j = 1$, then j is the desired index since $r + e < \texttt{rmax}_Q(q_1) \leq \epsilon|S|$. Otherwise, $j \geq 2$, and it follows that $r - e \leq \texttt{rmin}_Q(q_{j-1})$. If $r - e > \texttt{rmin}_Q(q_{j-1})$ then $\texttt{rmax}_Q(q_j) - \texttt{rmin}_Q(q_{j-1}) > 2e$; a contradiction since $e = \max_i(\texttt{rmax}_Q(q_{i+1}) - \texttt{rmin}_Q(q_i))/2$. By our choice of j, we have $\texttt{rmax}_Q(q_{j-1}) \leq r + e$. Thus $i = j - 1$ is an index i with the above described property.

In what follows, whenever we refer to a (relaxed) quantile summary as ϵ-approximate, we assume that it satisfies the conditions of Proposition 1.

2.2 Operations

We now describe two operations that produce new quantile summaries from existing summaries, and compute bounds on the precision of the resulting summaries.

The Combine Operation Let $Q' = \{x_1, x_2, ..., x_a\}$ and $Q'' = \{y_1, y_2, ..., y_b\}$ be two quantile summaries. The operation $\texttt{combine}(Q', Q'')$ produces a new quantile summary $Q = \{z_1, z_2, ..., z_{a+b}\}$ by simply sorting the union of the elements in two summaries, and defining new rank functions for each element as follows. W.l.o.g. assume that z_i corresponds to some element x_r in Q'. Let y_s be the largest element in Q'' that is not larger than x_r (y_s is undefined if no such element), and let y_t be the smallest element in Q'' that is not smaller than x_r (y_t is undefined if no such element). Then

$$\texttt{rmin}_Q(z_i) = \begin{cases} \texttt{rmin}_{Q'}(x_r) & \text{if } y_s \text{ undefined} \\ \texttt{rmin}_{Q'}(x_r) + \texttt{rmin}_{Q''}(y_s) & \text{otherwise} \end{cases}$$

$$\texttt{rmax}_Q(z_i) = \begin{cases} \texttt{rmax}_{Q'}(x_r) + \texttt{rmax}_{Q''}(y_s) & \text{if } y_t \text{ undefined} \\ \texttt{rmax}_{Q'}(x_r) + \texttt{rmax}_{Q''}(y_t) - 1 & \text{otherwise} \end{cases}$$

Lemma 1. *Let Q' be an ϵ'-approximate quantile summary for a multiset S', and let Q'' be an ϵ''-approximate quantile summary for a multiset S''. Then $\texttt{combine}(Q', Q'')$ produces an $\bar{\epsilon}$-approximate quantile summary Q for the multiset $S = S' \cup S''$ where $\bar{\epsilon} = \frac{n'\epsilon' + n''\epsilon''}{n' + n''} \leq \max\{\epsilon', \epsilon''\}$. Moreover, the number of elements in the combined summary is equal to the sum of the number of elements in Q' and Q''.*

Proof. Let n' and n'' respectively denote the number of observations covered by Q' and Q''. Consider any two consecutive elements z_i, z_{i+1} in Q. By Proposition 1, it suffices to show that $\texttt{rmax}_Q(z_{i+1}) - \texttt{rmin}_Q(z_i) \leq 2\bar{\epsilon}(n' + n'')$. We analyze two cases. First, z_i, z_{i+1} both come from a single summary, say elements x_r, x_{r+1} in Q'. Let y_s be the largest element in Q'' that is smaller than

x_r and let y_t be the smallest element in Q'' that is larger than x_{r+1}. Observe that if y_s and y_t are both defined, then they must be consecutive elements in Q''.

$$
\begin{aligned}
\mathtt{rmax}_Q(z_{i+1}) - \mathtt{rmin}_Q(z_i) \leq \\
[\mathtt{rmax}_{Q'}(x_{r+1}) + \mathtt{rmax}_{Q''}(y_t) - 1] \\
-[\mathtt{rmin}_{Q'}(x_r) + \mathtt{rmin}_{Q''}(y_s)] \\
\leq [\mathtt{rmax}_{Q'}(x_{r+1}) - \mathtt{rmin}_{Q'}(x_r)] + \\
[\mathtt{rmax}_{Q''}(y_t) - \mathtt{rmin}_{Q''}(y_s) - 1] \\
\leq 2(n'\epsilon' + n''\epsilon'') = 2\bar{\epsilon}(n' + n'').
\end{aligned}
$$

Otherwise, if only y_s is defined, then it must be the largest element in Q''; or if only y_t is defined, it must be the smallest element in Q''. A similar analysis can be applied for both these cases as well.

Next we consider the case when z_i and z_{i+1} come from different summaries, say, z_i corresponds to x_r in Q' and z_{i+1} corresponds to y_t in Q''. Then observe that x_r is the largest element smaller than y_t in Q' and that y_t is the smallest element larger than x_r in Q''. Moreover, x_{r+1} is the smallest element in Q' that is larger than y_t, and y_{t-1} is the largest element in Q'' that is smaller than x_r. Using these observations, we get

$$
\begin{aligned}
\mathtt{rmax}_Q(z_{i+1}) - \mathtt{rmin}_Q(z_i) \leq \\
[\mathtt{rmax}_{Q''}(y_t) + \mathtt{rmax}_{Q'}(x_{r+1}) - 1] \\
-[\mathtt{rmin}_{Q'}(x_r) + \mathtt{rmin}_{Q''}(y_{t-1})] \\
\leq [\mathtt{rmax}_{Q''}(y_t) - \mathtt{rmin}_{Q''}(y_{t-1})] + \\
[\mathtt{rmax}_{Q'}(x_{r+1}) - \mathtt{rmin}_{Q'}(x_r) - 1] \\
\leq 2(n'\epsilon' + n''\epsilon'') = 2\bar{\epsilon}(n' + n'').
\end{aligned}
$$

Corollary 1 *Let Q be a quantile summary produced by repeatedly applying the* combine *operation to an initial set of summaries $\{Q_1, Q_2, ..., Q_q\}$ such that Q_i is an ϵ_i-approximate summary. Then regardless of the sequence in which* combine *operations are applied, the resulting summary Q is guaranteed to be $(\max_{i=1}^q \epsilon_i)$-approximate.*

Proof. By induction on q. The base case of $q = 2$ follows from Lemma 1. Otherwise, $q > 2$, and we can partition the set of indices $I = \{1, 2, ..., q\}$ into two disjoint sets I_1 and I_2 such that Q is a result of the combine operation applied to summary Q' resulting from a repeated application of combine to $\{Q_i | i \in I_1\}$, and summary Q'' results from a repeated application of combine to $\{Q_i | i \in I_2\}$. By induction hypothesis, Q' is $\max_{i \in I_1} \epsilon_i$-approximate and Q'' is $\max_{i \in I_2} \epsilon_i$-approximate. By Lemma 1, then Q must be $\max_{i \in I_1 \cup I_2} \epsilon_i = \max_{i \in I} \epsilon_i$-approximate.

The Prune Operation The prune operation takes as input an ϵ'-approximate quantile summary Q' and a parameter K, and returns a new summary Q of size at most $K + 1$ such that Q is an $(\epsilon' + (1/(2K)))$-approximate quantile summary for S. Thus $\mathtt{prune}$ trades off slightly on accuracy for potentially much reduced space. We generate Q by querying Q' for elements of rank $1, |S|/K, 2|S|/K, ..., |S|$, and for each element $q_i \in Q$, we define $\mathtt{rmin}_Q(q_i) = \mathtt{rmin}_{Q'}(q_i)$, and $\mathtt{rmax}_Q(q_i) = \mathtt{rmax}_{Q'}(q_i)$.

Lemma 2. *Let Q' be an ϵ'-approximate quantile summary for a multiset S. Then $\mathtt{prune}(Q', K)$ produces an $(\epsilon' + 1/(2K))$-approximate quantile summary Q for S containing at most $K + 1$ elements.*

Proof. For any pair of consecutive elements q_i, q_{i+1} in Q, $\mathtt{rmax}_Q(q_{i+1}) - \mathtt{rmin}_Q(q_i) \leq (\frac{1}{K} + 2\epsilon')|S|$. By Proposition 1, it follows that Q must be $(\epsilon' + 1/(2K))$-approximate.

3 Deterministic Algorithms

In this section, we will develop a unified framework that captures many of the known deterministic algorithms for computing approximate quantile summaries. This framework appeared in the work of Manku, Rajagopalan, and Lindsay [14], and we refer to it as the MRL framework. We show various earlier approaches for computing approximate quantile summaries are all captured by this framework, and the best-possible algorithm in this framework computes an ϵ-approximate quantile summary using $O(\log^2(\epsilon n)/\epsilon)$ space. We then present an algorithm due to Greenwald and Khanna [10] that deviates from this framework and reduces the space needed to $O(\log(\epsilon n)/\epsilon)$. This is the current best known bound on the space needed for computing an ϵ-approximate quantile summary. We start with some classical results on exact algorithms for selection.

3.1 Exact Selection

In a natural but a restricted model of computation, Munro and Patterson [16] established almost tight bounds on deterministic selection with bounded space. In their model, the only operation that is allowed on the underlying elements is a pairwise comparison. At any time, the summary stores a subset of the elements in the data stream. They considered multi-pass algorithms and showed that any *comparison-based* algorithm that solves the selection problem in p passes requires $\Omega(n^{1/p})$ space. Moreover, there is a simple algorithm that can solve the selection problem in p passes using only $O(n^{1/p}(\log n)^{2-2/p})$ space. We will sketch here the proofs of both these results. We start with the lower bound result.

We focus on the problem of determining the median element using space s. Fix any deterministic algorithm and let us consider the first pass made by

the algorithm. Without any loss of generality, we may assume that the first s elements seen by the algorithm get stored in the summary Q. Now each time an element x is brought into Q, some element y is evicted from the summary Q. Let $U(y)$ denote the set of elements that were evicted from Q to make room for element y directly or indirectly. An element z is *indirectly evicted* by an element y in Q if the element y' evicted to make room for y directly or indirectly evicted the element z. Clearly, x will never get compared to any elements in $U(y)$ or the element y. We set $U(x) = U(y) \cup \{y\}$. The adversary now ensures that x is indistinguishable from any element in $U(x)$ with respect to the elements seen so far. Let $z_1, z_2, ..., z_s$ be the elements in Q after the first $n/2$ elements have been seen. Then $\sum_{i=1}^{s} |U(z_i)| = n/2 - s$, and by the pigeonhole principle, there exists an element $z_j \in Q$ such that $|z_j \cup U(z_j)| \geq n/(2s)$. The adversary now adjusts the values of the remaining $n/2$ elements so as to ensure that the median element for the entire sequence is the median element of the set $U(z_j) \cup \{z_j\}$. Thus after one pass, the problem size reduces by a factor of $2s$ at most. For the algorithm to succeed in p passes, we must have $(2s)^p \geq n$, that is, $s = \Omega(n^{1/p})$.

Theorem 1. *[16] Any p-pass comparison-based algorithm to solve the selection problem on a stream of n elements requires $\Omega(n^{1/p})$ space.*

Recently, Guha and McGregor [12], using techniques from communication complexity, have shown that *any p-pass algorithm to solve the selection problem on a stream of n elements requires $\Omega(n^{1/p}/p^6)$ bits of space.*

The Munro and Patterson algorithm that almost achieves the space bound given in Theorem 1 proceeds as follows. The algorithm maintains at all times a left and a right "filter" such that the desired element is guaranteed to lie between them. At the beginning, the left filter is assumed to be $-\infty$ and the right filter is assumed to be $+\infty$. Starting with an initial bound of n candidate elements contained between the left and the right filters, the algorithm in each pass gradually tightens the gap between the filters until the final pass where it is guaranteed to be less than s. The final pass is then used to determine the exact rank of the filters and retain all candidates in between to output the appropriate answer to the selection problem. The key property that is at the core of their algorithm is as follows.

Lemma 3. *If at the beginning of a pass, there are at most k elements that can lie between the left and the right filters, then at the end of the pass, this number reduces to $O((k \log^2 k)/s)$.*

Thus each pass of the algorithm may be viewed as an approximate selection step, with each step refining the range of the approximation achieved by the preceding step. We describe the precise algorithmic procedure to achieve this in the next subsection. Assuming the lemma, it is easy to see that by choosing $s = \Theta(n^{1/p}(\log n)^{2-2/p})$, we can ensure that after the ith pass, the number of candidate elements between the filters reduces to at most $n^{\frac{p-i}{p}} \log^{\frac{2i}{p}} n$. Setting

$i = p - 1$, ensures that the number of candidate elements in the pth pass is at most $n^{1/p}(\log n)^{2-2/p}$.

3.2 MRL Framework for ϵ-approximate Quantile Summaries

A natural way to construct quantile summaries of large quantities of data is by merging several summaries of smaller quantities of data. Manku, Rajagopalan, and Lindsay [14] noted that all one-pass approximate quantile algorithms prior to their work (most notably, [16, 1]) fit this pattern. They defined a framework, refered from here on as the *MRL framework*, in which all algorithms can be expressed in terms of two basic operations on existing quantile summaries: NEW and COLLAPSE. Each algorithm in the framework builds a quantile summary by applying these operations to members of a set of smaller, fixed sized, quantile summaries. These fixed size summaries are referred to as *buffers*. A buffer is a quantile summary of size k that summarizes a certain number of observations. When a buffer summarizes k' observations we define the *weight* of the buffer to be $\lceil \frac{k'}{k} \rceil$.

NEW fills a buffer with k new observations from the input stream (we assume that n is always an integral multiple of k). COLLAPSE takes a set of buffers as input and returns a single buffer summarizing all the input buffers.

Each algorithm in the framework is parameterized by b, the total number of buffers, and k, the number of entries per buffer, needed to summarize a sample of size n to precision ϵ, as well as a *policy* that determines when to apply NEW and COLLAPSE. Further, the authors of [14] proposed a new algorithm that improved upon the space bk needed to summarize n observations to a given precision ϵ. In light of more recent work, it is illuminating to recast the MRL framework in terms of $\mathtt{rmin}_Q$ and $\mathtt{rmax}_Q$ for each entry in the buffer.

A buffer of weight w in the MRL framework is a quantile summary of kw observations, where k is the number of (sorted) entries in the buffer. Each entry in the buffer consists of a single value that represents w observations in the original data stream. We associate a level, l, with each buffer. Buffers created by NEW have a level of 0, and buffers created by COLLAPSE have a level, l', that is one greater than the maximum l of the constituent buffers.

NEW takes the next k observations from the input stream, sorts them in ascending order, and stores them in a buffer, setting the weight of this new buffer to 1. This buffer can reproduce the entire sequence of k observations and therefore $\mathtt{rmin}$ and $\mathtt{rmax}$ of the ith element both equal i, and the buffer has precision that can satisfy even $\epsilon = 0$.

COLLAPSE summarizes a set of α buffers, $B_1, B_2, \ldots, B_\alpha$, with a single buffer B, by first calling $\mathtt{combine}(B_1, B_2, \ldots, B_\alpha)$, and then[3] $\mathtt{prune}(B, k-1)$. The weight of this new buffer is $\sum_j w_j$.

---

[3] The prune phase of COLLAPSE in the MRL paper differs very slightly from our `prune`.

Lemma 4. *Let B be a buffer created by invoking* COLLAPSE *on a set of α buffers, $B_1, B_2, \ldots, B_\alpha$, where each buffer B_i has weight w_i and precision ϵ_i. Then B has precision $\leq 1/(2k-2) + \max_i\{\epsilon_i\}$.*

Proof. By Corollary 1, repeated application of `combine` creates a temporary summary, Q, with precision $\max_i\{\epsilon_i\}$ and αk entries. By Lemma 2, $B = $ `prune`$(Q, k-1)$ produces a summary with an ϵ that is $1/(2k-2)$ more than the precision of Q.

We can view the execution of an algorithm in this framework as a tree. Each node represents the creation of a new buffer of size k: leaves represent NEW operations and internal nodes represent COLLAPSE. Figure 1 represents an example of such a tree. The number next to each node specifies the weight of the resulting buffer. The level, l, of a buffer represents its height in this tree.

Lemma 4 shows that each COLLAPSE operation adds at most $1/(2k)$ to the precision of the buffer. It follows from repeated application of Lemma 4 that a buffer of level l has a precision of $l/2k$. Similarly, we can relate the precision of the final summary to the height of the tree.

Corollary 2 *Let $h(n)$ denote the maximum height of the algorithm tree on an input stream of n elements. Then the final summary produced by the algorithm is $(h(n)/(2k))$-approximate.*

We now apply the lemmas to several different algorithms, in order to compute the space requirements for a given precision and a given number of observations we wish to summarize.

The Munro-Patterson Algorithm

The Munro-Patterson algorithm [16] initially allocates b empty buffers. After k new observations arrive, if an empty buffer exists, then NEW is invoked. If no empty buffer exists, then it creates an empty buffer by calling COLLAPSE on two buffers of equal weight. Figure 1 represents the Munro-Patterson algorithm for small values of b.

Let $h = \lceil \log(n/k) \rceil$. Since the algorithm merges at each step buffers of equal weight, it follows that the resulting tree is a balanced binary tree of height h where the leaves represent k observations each, and each internal node corresponds to $k2^i$ observations for some integer $1 \leq i \leq h$. The number of available buffers b must satisfy the constraint $b \geq h$ since if $n = k2^h - 1$, the resulting summary requires h buffers with distinct weights of $1, 2, 4, \ldots$ etc. By Corollary 2, the resulting summary is guaranteed to be $h/2k$-approximate. Given a desired precision ϵ, we need to satisfy $h/2k \leq \epsilon$. It is easy to verify that choosing $k = \lceil (\log(2\epsilon n))/(2\epsilon) \rceil$ satisfies the precision requirement. Thus the total space used by this algorithm is $bk = O(\log^2(\epsilon n)/\epsilon)$.
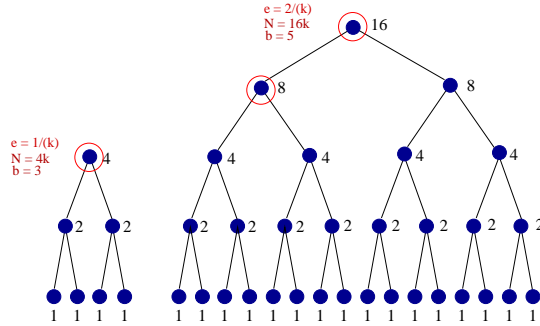
Fig. 1. Tree representations of Munro-Patterson algorithm for $b = 3$, and 5. Note that the final state of the algorithm under a root consists of the pair of buffers that are the children of the root. The shape and height of a tree depends only on b, and is independent of k, but the precision, ϵ, and the number of observations summarized, n, are both functions of k.

The Alsabti-Ranka-Singh Algorithm

The Alsabti-Ranka-Singh Algorithm [1] allocates b buffers and divides them, equally, into two classes. The first $b/2$ buffers are reserved for the leaves of the tree. Each group of $kb/2$ observations are collected into $b/2$ buffers using NEW, and then COLLAPSEed into a single buffer from the second class. This process is repeated $b/2$ times, resulting in $b/2$ buffers with weight $b/2$ as children of the root. After the last such operation, the $b/2$ leaf buffers are discarded.

The depth of the Alsabti-Ranka-Singh tree is always 2, so by Corollary 2, $k \geq 1/\epsilon$. We need $k * (b/2)^2 \geq n$ to cover all the observations. Given that b increases coverage quadratically and k only linearly, it is most efficient to choose the largest b and smallest k that satisfy the above constraints if we wish to minimize bk. The smallest k is $1/\epsilon$, hence $(b/2)^2 \geq \epsilon n$, so $b \geq \sqrt{\epsilon n/4}$, and $bk = O(\sqrt{n/\epsilon})$.

The Manku-Rajagopalan-Lindsay Algorithm

It is natural to try to devise the best algorithm possible within the MRL framework. It is easy to see that, for a given b and k, the more leaves an algorithm tree has, the more observations it summarizes. Also, from Corollary 2, the shallower the tree, the more precise the summary is. Clearly, for a fixed b it is best to construct the shallowest and widest tree possible, in order to summarize the most observations with the finest precision.

However, both algorithms presented above are inefficient in this light. For example, Alsabti-Ranka-Singh is not as wide as possible. After the algorithm fills the first $b/2$ buffers, it invokes COLLAPSE, leaving all buffers empty except for one buffer with precision $1/(2k)$ summarizing $bk/2$ observations. However,
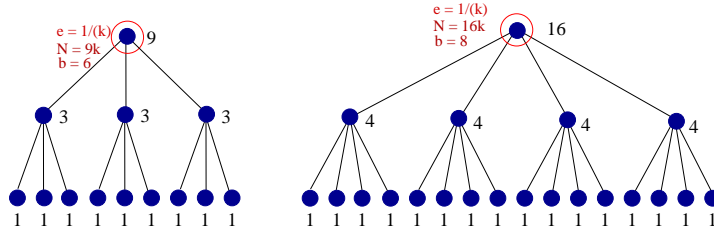
Fig. 2. Tree representation of Alsabti-Ranka-Singh algorithm. For any specific choice of b_1 and b_2, for $b_1 \neq b_2$, the tree for $b = b_1$ is not a subtree of $b = b_2$. The precision, ϵ, and the number of observations summarized, n, are both functions of k.

there is no need for it to call COLLAPSE at that point — there are $b/2$ empty buffers remaining. If it deferred calling COLLAPSE until after filling all b buffers, the results would again be all buffers empty except for one buffer with precision $1/(2k)$, but this time summarizing bk observations. Even worse, after $b/2$ calls to COLLAPSE, Alsabti-Ranka-Singh discards the $b/2$ "leaf buffers", although if it kept those buffers, and continued collecting, it could keep on collecting, roughly, a factor of $b/2$ *times* as many observations with no loss of precision.

The Munro-Patterson algorithm *does* use empty buffers greedily. However, it is not as shallow as possible. Munro-Patterson requires a tree of height $\log \beta$ to combine β buffers, because it only COLLAPSEs pairs of buffers at a time, instead of combining the entire set at once. Had Munro-Patterson called COLLAPSE on the entire set in a single operation, it would end with a buffer with $\log \beta/(2k)$ higher precision (there is a loss of precision of $1/(2k)$ for each call to COLLAPSE).

The new Manku-Rajagopalan-Lindsay (MRL) algorithm [14] aims to use the buffers as efficiently as possible - to build the shallowest, widest tree it can for a fixed b. The MRL algorithm never discards buffers; it uses any buffers that are available to record new observations. The basic approach taken by MRL is to keep the algorithm tree as wide as possible. It achieves this by labeling each buffer B_j with a level L_j, which denotes its height (see Figure 3). Let l denote the smallest value of L_j for all existing, full, buffers. The MRL policy is to allocate new buffers at level 0 until the buffer pool is exhausted, and then to call COLLAPSE on all buffers of level l. More specifically MRL considers two cases:

- Empty buffers exist. Call NEW on each and assign level 0 to them.
- No empty buffers exist. Call COLLAPSE on all buffers of level l and assign the output buffer a level L of $l + 1$.

If level l contains only 2 buffers, then COLLAPSE frees only a single buffer which NEW assigns level 0. When that buffer is filled, it is the only buffer at level 0. Calling COLLAPSE on a single buffer merely increments the level

without modifying the buffer. This will continue until the buffer is promoted to level l, where other buffers exist. Thus MRL treats a third case specially:

- Precisely one empty buffer exists. Call NEW on it and assign it level l.
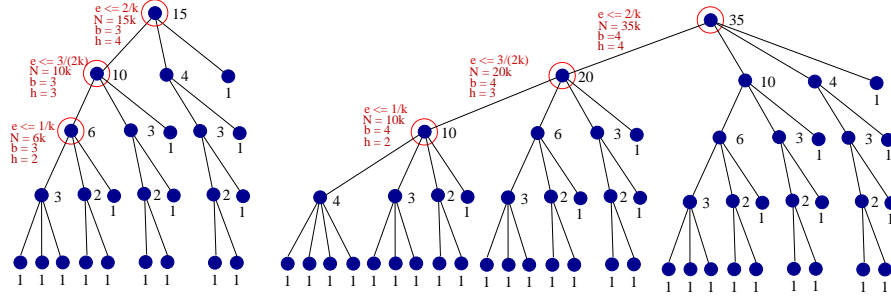

Fig. 3. Tree representation of Manku-Rajagopalan-Lindsay algorithm.

Proposition 2 *In the tree representing the* COLLAPSE*s and* NEW*s in an MRL algorithm with b buffers, the number of leaves in a subtree of height h, $L(b, h)$, is* $\begin{pmatrix} b + h - 2 \\ h - 1 \end{pmatrix}$.

Proof. We will prove by induction on h that $L(b, h) = \begin{pmatrix} (b - 1) + (h - 1) \\ h - 1 \end{pmatrix}$.

For $h = 1$ the tree is a single node, a leaf. So, for all $b, L(b, 1) = 1 = \begin{pmatrix} b - 1 \\ 0 \end{pmatrix}$.

Assume that for all $h' < h$, for all b, that $L(b, h') = \begin{pmatrix} (b - 1) + (h' - 1) \\ h' - 1 \end{pmatrix}$.

$L(b, h)$ is equal to $\sum_{i=1}^{b} L(i, h - 1)$. To see this, note that we build the hth level by finishing a tree of $(b, h - 1)$, then collapsing it all into 1 buffer. Now we have $b - 1$ buffers left over to build another tree of height $h - 1$. When we finish, we collapse *that* into a single buffer, and start over building a tree of height $h - 1$ with $b - 2$ buffers, and so on, until we are left with only 1 buffer, which we fill. At that point we have no free buffers left and so we collapse all b buffers into the single buffer that is the root at height h. By the induction hypothesis we know that $L(b, h - 1) = \begin{pmatrix} (b - 1) + (h - 2) \\ h - 2 \end{pmatrix}$. Therefore $L(b, h) = \sum_{i=1}^{b} \begin{pmatrix} (i - 1) + (h - 2) \\ h - 2 \end{pmatrix}$, or $L(b, h) = \sum_{i=0}^{b-1} \begin{pmatrix} i + (h - 2) \\ h - 2 \end{pmatrix}$. But

by summation on the upper index, we have $L(b,h) = \sum_{i=0}^{b-1} \left(\begin{array}{c} (i + (h-2)) \\ h-2 \end{array} \right) = \left(\begin{array}{c} (b-1) + 1 + h - 2 \\ h-1 \end{array} \right) = \left(\begin{array}{c} (b-1) + (h-1) \\ h-1 \end{array} \right).$

The leaf buffers must include all n observations, so by Proposition 2, $b, k,$ and h must be chosen such that $kL(b,h) = k \left(\begin{array}{c} b+h-2 \\ h-1 \end{array} \right) \geq n.$ The summary must be ϵ-approximate, so by Corollary 2, $h/(2k) \geq \epsilon'$. We must choose $b, k,$ and h to satisfy these constraints while minimizing bk. Increasing h (up to the point we would violate the precision requirement) increases space-efficiency because larger h covers more observations without increasing the memory footprint, bk. The largest h that bounds the precision of the summary to be within ϵ, is $h = 2\epsilon k$.

Now that h can be computed as a function of k and ϵ, we can focus on choosing the best values of b and k to minimize the space bk required. We first show that if a pair b, k is *space-efficient* — meaning that no other pair b', k' could cover more observations in the same space bk — then $k = O(b/\epsilon)$.

The number of observations covered by the MRL algorithm for a pair b, k is $kL(b, 2\epsilon k)$. $L(b, h)$ is symmetric in b and h (e.g. $L(b, h) = L(h, b)$). The symmetry of L implies that $k \geq b/(2\epsilon)$ (else we could have used our space more efficiently by swapping b and $2\epsilon k$ ($b' = 2\epsilon k, k' = b/2\epsilon$) yielding the same values of $L(b', 2\epsilon k')$. $b'k' = bk$ implying that our space requirements were equivalent, but the larger value of k' would mean that we cover more observations (as long as $\epsilon < .5$). Consequently, we would never choose $k < b/(2\epsilon)$ if we were trying to minimize space.)

On the other hand, if we choose k too large relative to b, we would again use space inefficiently. Assume, for contradiction, that a space-efficient b, k existed, such that $k > 15b/\epsilon$. But, if so, we could more efficiently choose $k' = k/2$ and $b' = 2b$, using the same space but covering more observations. b' and k' would cover more observations because $k/2 \left(\begin{array}{c} 2b + \epsilon k \\ \epsilon k \end{array} \right) > k \left(\begin{array}{c} b + 2\epsilon k \\ 2\epsilon k \end{array} \right)$, when $k > 15b/\epsilon$. It follows that if a pair b, k is space-efficient, then k is bounded by $b/(2\epsilon) \leq k \leq 15b/\epsilon$.

If $k = O(b/\epsilon)$, then we need only find the minimum b such that $\frac{b}{2\epsilon} \left(\begin{array}{c} 2b - 2 \\ b - 1 \end{array} \right) \approx n.$ Roughly, $b = O(\log(2\epsilon n))$, and $k = O(\frac{1}{2\epsilon})\log(2\epsilon n)$, so $bk = O((1/(2\epsilon))\log^2(2\epsilon n)).$

3.3 The GK Algorithm

The new MRL algorithm was designed to be the best possible algorithm within the MRL framework. Nevertheless, it still suffers from some inefficiencies. For a given memory requirement, bk, the precision is reduced (that is, ϵ is increased) by three factors. First, each time COLLAPSE is called, combining the α buffers

together increases the gap between `rmin` and `rmax` of elements in that buffer by the sum of the gaps between the individual `rmin` and `rmax` in *all* the buffers — because algorithms do not maintain any information that can allow them to recover how the deleted entries in each buffer may have been interleaved. Second, each COLLAPSE invokes the `prune` operation, which increases ϵ by $1/(2k)$. Finally, as is true for all algorithms in the MRL framework, MRL keeps no per-entry information about `rmin` and `rmax` for individual entries. Rather, we must assume that every entry in a buffer has the worst `rmax−rmin`.

We next describe an algorithm due to Greenwald and Khanna [10] that overcomes some of these drawbacks by not using `combine` and `prune`. It yields an ϵ-approximate quantile summary using only $O((\log \epsilon n)/\epsilon)$ space.

The GK Summary Data Structure

At any point in time n, GK maintains a summary data structure $\mathsf{Q}_{\mathsf{GK}}(n)$ that consists of an ordered sequence of tuples which correspond to a subset of the observations seen thus far. For each observation v in Q_{GK}, we maintain implicit bounds on the minimum and the maximum possible rank of the observation v among the first n observations. Let $\mathtt{rmin}_{\mathsf{GK}}(v)$ and $\mathtt{rmax}_{\mathsf{GK}}(v)$ denote respectively the lower and upper bounds on the rank of v among the observations seen so far. Specifically, Q_{GK} consists of tuples $t_0, t_1, ..., t_{s-1}$ where each tuple $t_i = (v_i, g_i, \Delta_i)$ consists of three components: (i) a value v_i that corresponds to one of the elements in the data sequence seen thus far, (ii) the value g_i equals $\mathtt{rmin}_{\mathsf{GK}}(v_i) - \mathtt{rmin}_{\mathsf{GK}}(v_{i-1})$ (for $i = 0$, $g_i = 0$), and (iii) Δ_i equals $\mathtt{rmax}_{\mathsf{GK}}(v_i) - \mathtt{rmin}_{\mathsf{GK}}(v_i)$. Note that $v_0 \leq v_1 \leq ... \leq v_{s-1}$. We ensure that, at all times, the maximum and the minimum values are part of the summary. In other words, v_0 and v_{s-1} always correspond to the minimum and the maximum elements seen so far. It is easy to see that $\mathtt{rmin}_{\mathsf{GK}}(v_i) = \sum_{j \leq i} g_j$ and $\mathtt{rmax}_{\mathsf{GK}}(v_i) = \sum_{j \leq i} g_j + \Delta_i$. Thus $g_i + \Delta_i - 1$ is an upper bound on the *total* number of observations that may have fallen between v_{i-1} and v_i. Finally, observe that $\sum_i g_i$ equals n, the total number of observations seen so far.

Answering Quantile Queries: A summary of the above form can be used in a straightforward manner to provide ϵ-approximate answers to quantile queries. Proposition 1 forms the basis of our approach, and the following is an immediate corollary.

Corollary 3 *If at any time n, the summary $\mathsf{Q}_{\mathsf{GK}}(n)$ satisfies the property that $\max_i(g_i + \Delta_i) \leq 2\epsilon n$, then we can answer any ϕ-quantile query to within an ϵn precision.*

Overview: At a high level, our algorithm for maintaining the quantile summary proceeds as follows. Whenever the algorithm sees a new observation, it inserts in the summary a tuple corresponding to this observation. Periodically, the algorithm performs a sweep over the summary to "merge" some of the tuples into their neighbors so as to free up space. The heart of the

algorithm is in this merge phase where we maintain several conditions that allow us to bound the space used by Q_{GK} at any time. We next develop some basic concepts that are needed to precisely describe these conditions.

Tuple Capacities: When a new tuple t_i is added to the summary at time n, we set its g_i value to be 1 and its Δ_i value[4] to be $\lfloor 2\epsilon n \rfloor - 1$. All summary operations maintain the property that the Δ_i value never changes. By Corollary 3, it suffices to ensure that at all times greater than $1/2\epsilon$, $\max_i(g_i + \Delta_i) \leq 2\epsilon n$. (For times earlier than $1/2\epsilon$ the summary preserves every observation, the error is always zero, and $\Delta_i = 0$ for all i.) Motivated by this consideration, we define the *capacity* of a tuple t_i at any time n', denoted by $\mathsf{cap}(t_i, n')$, to be $\lfloor 2\epsilon n' \rfloor - \Delta_i$. Thus the capacity of a tuple increases over time. An individual tuple is said to be *full* at time n' if $g_i + \Delta_i = \lfloor 2\epsilon n' \rfloor$. The *capacity* of an individual tuple is, therefore, the maximum number of observations that can be counted by g_i before the tuple becomes full.

Bands: Tuples with higher capacity correspond to values whose ranks are known with higher precision. Intuitively, high capacity tuples are more valuable than lower capacity tuples and our merge rules will favor elimination of lower capacity tuples by merging them into larger capacity ones. However, we will find it convenient to not differentiate among tuples whose capacities are within a small multiplicative factor of one another. We thus group tuples into geometric classes referred to as *bands* where roughly speaking, a tuple t_i is in a band α if $\mathsf{cap}(t_i, n) \approx 2^\alpha$. Since capacities increase over time, the band of a tuple increases over time. We will find it convenient to ensure the following stability property in assigning bands: if at some time n, we have $\mathsf{band}(t_i, n) = \mathsf{band}(t_j, n)$ then for all times $n' \geq n$, we have $\mathsf{band}(t_i, n') = \mathsf{band}(t_j, n')$. We thus use a slightly more technical definition of bands. Let $p = \lfloor 2\epsilon n \rfloor$ and $\hat{\alpha} = \lceil \log_2 p \rceil$. Then we say $\mathsf{band}(t_i, n)$ is α if

$$2^{\alpha-1} + (p \bmod 2^{\alpha-1}) \leq \mathsf{cap}(t_i, n) < 2^\alpha + (p \bmod 2^\alpha).$$

If the Δ value of tuple t_i is p, then we say $\mathsf{band}(t_i, n)$ is 0. It follows from the definition of band that at all times the first $1/(2\epsilon)$ observations, with $\Delta = 0$, are alone in $\mathsf{band}_{\hat{\alpha}}$.

We will denote by $\mathsf{band}(t_i, n)$ the band of tuple t_i at time n, and by $\mathsf{band}_\alpha(n)$ all tuples (or equivalently, the capacities associated with these tuples) that have a band value of α. The terms $(p \bmod 2^{\alpha-1})$ and $(p \bmod 2^\alpha)$ above ensure the following stability property: once a pair of tuples is in the same band, they stay together from there on even as band boundaries are modified. At the same time, these terms do not change the underlying geo-

---

[4] In practice , we set Δ more tightly by inserting $(v, 1, g_i + \Delta_i - 1)$ as the tuple immediately preceding t_{i+1}. It is easy to see that if Corollary 3 is satisfied before insertion, it remains true after insertion, and that $\lfloor 2\epsilon n \rfloor - 1$ is an upper bound on the value of Δ_i. For the purpose of our analysis, we always assume the worst-case insertion value of $\Delta_i = \lfloor 2\epsilon n \rfloor - 1$.

metric grouping in any fundamental manner; $\mathsf{band}_\alpha(n)$ contains either $2^{\alpha-1}$ or 2^α distinct capacity values.

Proposition 3 *1. At any point in time n and for any α, $\hat{\alpha} > \alpha \geq 1$, the number of distinct capacity values that can belong to $\mathsf{band}_\alpha(n)$ is either $2^{\alpha-1}$ or 2^α.*

2. If at some time n, any two tuples t_i, t_j are in the same band, then for all times $n' \geq n$, this holds true.

3. At any point in time, n, and for any α, $\hat{\alpha} \geq \alpha \geq 0$, the number of distinct capacity values that can belong to $\mathsf{band}_\alpha(n)$ is $\leq 2^\alpha$.

Proof. To see (1), if $p \bmod 2^\alpha < 2^{\alpha-1}$, then $p \bmod 2^\alpha = p \bmod 2^{\alpha-1}$, and $\mathsf{band}_\alpha(n)$ contains $2^\alpha - 2^{\alpha-1} = 2^{\alpha-1}$ distinct values of capacity. If $p \bmod 2^\alpha \geq 2^{\alpha-1}$, then $p \bmod 2^\alpha = 2^{\alpha-1} + (p \bmod 2^{\alpha-1})$, and $\mathsf{band}_\alpha(n)$ contains 2^α distinct capacity values.

To see (2), consider any $\mathsf{band}_\alpha(n)$. Each time p increases by 1, if $p \bmod 2^\alpha \notin \{0, 2^{\alpha-1}\}$, then both $p \bmod 2^{\alpha-1}$ and $p \bmod 2^\alpha$ increase by 1 and thus the range of $\mathsf{band}_\alpha(n)$ shifts by 1. At the same time, capacity of each tuple changes by 1 and thus the set of tuples that belong to $\mathsf{band}_\alpha(n)$ stays unchanged.

Now suppose when p increases by 1, $p \bmod 2^\alpha = 2^{\alpha-1}$. It is easy to verify that for any value of p, there is at most one value of α that satisfies the equation $p \bmod 2^\alpha = 2^{\alpha-1}$. Then the left boundary of the band α decreases by $2^{\alpha-1} - 1$ while the right boundary increases by 1. The resulting band now captures all tuples that belonged to old bands $(\alpha - 1)$ and α. Also, for all bands β where $1 \leq \beta < \alpha$ (and hence $p \bmod 2^\beta = 0$), the range of the band changes from $[2^\beta - 1, 2^{\beta+1} - 1)$ to $[2^{\beta-1}, 2^\beta)$. Thus all tuples in the old band β now belong together to the new band $\beta + 1$. Finally, all bands $\gamma > \alpha$ satisfy $p \bmod 2^\gamma \notin \{0, 2^{\gamma-1}\}$, and hence continue to capture the same set of tuples as observed above.

Thus once a pair of tuples is present in the same band, their bands never diverge again.

(3) holds for both band_0 and $\mathsf{band}_{\hat{\alpha}}$ — they both contain precisely one distinct capacity value. For all other values of α, (3) follows directly from (1).

A Tree Representation: In order to decide on how tuples are merged in order to *compress* the summary, we create an *ordered* tree structure, referred to as the *quantile tree*, whose nodes correspond to the tuples in the summary. The tree structure creates a hierarchy based on tuple capacities and proximity. Specifically, the quantile tree $\mathsf{T}(n)$ at time n is created from the summary $\mathsf{Q}_{\mathsf{GK}}(n) = \langle t_0, t_1, ..., t_{s-1} \rangle$ as follows. $\mathsf{T}(n)$ contains a node V_i for each t_i along with a special root node R. The parent of a node V_i is the node V_j such that j is the least index greater than i with $\mathsf{band}(t_j, n) > \mathsf{band}(t_i, n)$. If no such index exists, then the node R is set to be the parent. The children of each node are ordered as they appear in the summary. All children (and all descendants) of a given node V_i correspond to tuples that have capacities smaller than that of
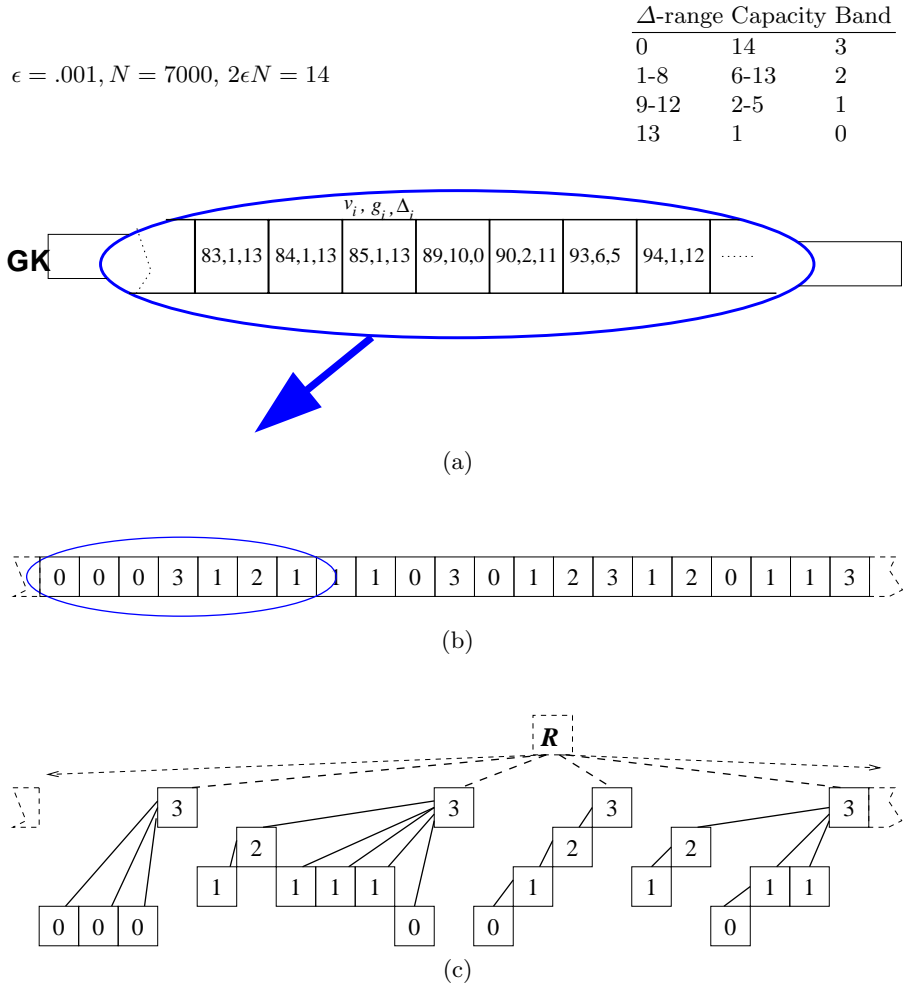
Δ-range	Capacity	Band
0	14	3
1-8	6-13	2
9-12	2-5	1
13	1	0

$\epsilon = .001, N = 7000, 2\epsilon N = 14$

(a)

(b)

(c)

Fig. 4. (a) Tuple representation. (b) Tuples labeled only with band numbers. (c) Corresponding tree representation.

tuple t_i. The relationship between $\mathsf{Q_{GK}}(n)$ and $\mathsf{T}(n)$ is represented pictorially in Figure 4. We next highlight two useful properties of $\mathsf{T}(n)$.

Proposition 4 *Each node V_i in a quantile tree $\mathsf{T}(n)$ satisfies the following two properties:*

1. *The children of each node V_i in $\mathsf{T}(n)$ are always arranged in non-increasing order of band in $\mathsf{Q_{GK}}(n)$.*
2. *The tuples corresponding to V_i and the set of all descendants of V_i in $\mathsf{T}(n)$ form a contiguous segment in $\mathsf{Q_{GK}}(n)$.*

Proof. To see property (1), consider any two children V_j and $V_{j'}$ of V_i with $j < j'$. Then if $\mathsf{band}(t_{j'}, n) > \mathsf{band}(t_j, n)$, $V_{j'}$ and not V_i would be the parent of node V_j in $\mathsf{T}(n)$.

We establish property (2) using proof by contradiction. Consider a node V_i that violates the property. Let k be the largest integer such that V_k is a descendant of V_i, and let j be the largest index less than k such that V_j is a descendant of V_i while V_{j+1} is not. Also, let $j < \ell < k$ be the largest integer such that V_ℓ is not a descendant of V_i. Clearly, V_i must be the parent of V_k, and some node V_x where $x > \ell$ must be the parent of V_j.

If $\mathsf{band}(t_\ell, n) < \mathsf{band}(t_k, n)$, then one of the nodes $V_{\ell+1}, ..., V_k$ must be the parent of V_ℓ – a contradiction since each of these nodes is a descendant of V_i by our choice of ℓ. Otherwise, we have $\mathsf{band}(t_\ell, n) \geq \mathsf{band}(t_k, n)$. Now if $\mathsf{band}(t_\ell, n) \geq \mathsf{band}(t_i, n)$, then V_j's parent is some node $V_{x'}$ with $x' \leq \ell$. So it must be that $\mathsf{band}(t_k, n) \leq \mathsf{band}(t_\ell, n) < \mathsf{band}(t_i, n)$. Consider the node V_y that is the parent of V_ℓ in $\mathsf{T}(n)$. By our choice of V_ℓ, we have $k < y < i$. Since $\mathsf{band}(V_y, n) > \mathsf{band}(V_\ell, n)$, it must be that V_y is the parent of V_k and not V_i. A contradiction!

Operations

We now describe the various operations that we perform on our summary data structure. We start with a description of external operations:

External Operations

QUANTILE(ϕ) To compute an ϵ-approximate ϕ-quantile from the summary $\mathsf{Q_{GK}}(n)$ after n observations, compute the rank, $r = \lceil \phi(n-1) \rceil$. Find i such that both $r - \mathtt{rmin_{GK}}(v_i) \leq \epsilon n$ and $\mathtt{rmax_{GK}}(v_i) - r \leq \epsilon n$ and return v_i.

INSERT(v) Find the smallest i, such that $v_{i-1} \leq v < v_i$, and insert the tuple $(v, 1, \lfloor 2\epsilon n \rfloor - 1)$, between t_{i-1} and t_i. Increment s. As a special case, if v is the new minimum or the maximum observation seen, then insert $(v, 1, 0)$.

INSERT(v) maintains correct relationships between g_i, Δ_i, $\mathtt{rmin_{GK}}(v_i)$ and $\mathtt{rmax_{GK}}(v_i)$. Consider that if v is inserted before v_i, the value of $\mathtt{rmin_{GK}}(v)$ may be as small as $\mathtt{rmin_{GK}}(v_{i-1}) + 1$, and hence $g_i = 1$. Similarly, $\mathtt{rmax_{GK}}(v)$ may be as large as the *current* $\mathtt{rmax_{GK}}(v_i)$, which in turn is bounded to be within $\lfloor 2\epsilon n \rfloor$ of $\mathtt{rmin_{GK}}(v_{i-1})$. Note that $\mathtt{rmin_{GK}}(v_i)$ and $\mathtt{rmax_{GK}}(v_i)$ increase by 1 after insertion.

Internal Operations

COMPRESS() The operation COMPRESS repeatedly attempts to find a suitable segment of adjacent tuples in the summary $\mathsf{Q_{GK}}(n)$ and merges them into the neighboring tuple (i.e. the tuple that succeeds them in the summary). We first describe how the summary is updated when for any

```
COMPRESS()
  for i from s − 2 to 0 do
    if ((band(t_i, n) ≤ band(t_{i+1}, n)) &&
       (g_i^* + g_{i+1} + Δ_{i+1} < 2ϵn)) then
      g_{i+1} = g_{i+1} + g_i^*;
      Remove t_i and all its descendants;
    end if
  end for
end COMPRESS
```

Fig. 5. Pseudocode for COMPRESS

$1 < x \leq y$, tuples $t_x, ..., t_y$ are merged together into the neighboring tuple t_{y+1}. We replace g_{y+1} by $\sum_{j=x}^{y+1} g_j$ and Δ_{y+1} remains unchanged. It is easy to verify that this operation correctly maintains $\mathtt{rmin_{GK}}(t_{y+1})$ and $\mathtt{rmax_{GK}}(t_{y+1})$ values for all the tuples in the summary. Deletion of $t_x, ..., t_y$ does not alter the $\mathtt{rmin_{GK}}()$ and $\mathtt{rmax_{GK}}()$ values for any of the remaining tuples and the merge operation above precisely maintains this property. COMPRESS chooses the segments to be merged in a specific manner, namely, it considers only those segments that correspond to a tuple t_i and all its descendants in the tree $\mathsf{T}(n)$. By Lemma 4 we know that t_i and all its descendants corresponds to a segment of adjacent tuples $t_{i-a}, t_{i-a+1}, ..., t_i$ in $\mathsf{Q_{GK}}(n)$. Let g_i^* denote the sum of g-values of the tuple t_i and *all its descendants* in $\mathsf{T}(n)$, that is, $g_i^* = \sum_{j=i-a}^{i} g_j$. To maintain the ϵ-approximate guarantee for the summary, we ensure that a merge is done only if $g_i^* + g_{i+1} + \Delta_{i+1} \leq 2\epsilon n$. Finally, COMPRESS ensures that we always merge into tuples of comparable or better capacity. The operation COMPRESS terminates if and only if there are no segments that satisfy the conditions above. Figure 5 describes an efficient implementation of the COMPRESS operation.

Note that since COMPRESS never alters the Δ value of surviving tuples, it follows that Δ_i of any quantile entry remains unchanged once it has been inserted.

COMPRESS inspects tuples from right (highest index) to left. Therefore, it first combines children (and their entire subtree of descendants) into parents. It combines siblings only when no more children can be combined into the parent.

Analysis

The INSERT as well as COMPRESS operations always ensure that $g_i + \Delta_i \leq 2\epsilon n$. As n always increases, it is easy to see that the data structure above maintains an ϵ-approximate quantile summary at each point in time. We will

> **Initial State**
> $Q_{GK} \leftarrow \emptyset$; $s = 0$; $n = 0$.
> **Algorithm**
> *To add the $(n + 1)st$ observation, v, to summary* $Q_{GK}(n)$:
>
>   **if** $(n \equiv 0 \bmod \frac{1}{2\epsilon})$ **then**
>   COMPRESS();
>   **end if**
>   INSERT(v);
> $n = n + 1$;

Fig. 6. Pseudo-code for the algorithm

now establish that the total number of tuples in the summary Q_{GK} after n observations have been seen is bounded by $(11/2\epsilon) \log(2\epsilon n)$.

We start by defining a notion of coverage. We say that a tuple t in the quantile summary Q_{GK} *covers* an observation v at any time n if either the tuple for v has been directly merged into t_i or a tuple t that covered v has been merged into t_i. Moreover, a tuple always covers itself. It is easy to see that the total number of observations covered by t_i is exactly given by $g_i = g_i(n)$. The lemmas below highlight some useful properties concerning coverage of observations by various tuples.

Lemma 5. *At no point in time, a tuple t with a band value of α covers a tuple t' which if it were alive, would have a band value strictly greater than α.*

Proof. Note that the band of a tuple at any time n is completely determined by its Δ value. Since the Δ value never changes once a tuple is created, the notion of band value of a tuple is well-defined even if the tuple no longer exists in the summary.

Now suppose at some time n, the event described in the lemma occurs. The COMPRESS subroutine never merges a tuple t_i into an adjacent tuple t_{i+1} if the band of t_i is greater than the band of t_{i+1}. Thus the only way in which this event can occur is if it at some earlier time $m < n$, we had $\mathsf{band}(t_i, m) \leq \mathsf{band}(t_{i+1}, m)$, and at the current time n, we have $\mathsf{band}(t_i, n) > \mathsf{band}(t_{i+1}, n)$. Consider first the case when $\mathsf{band}(t_i, m) = \mathsf{band}(t_{i+1}, m)$. By Proposition 3, it can not be the case that at some later time n, $\mathsf{band}(t_i, n) \neq \mathsf{band}(t_{i+1}, n)$. Now consider the case when $\mathsf{band}(t_i, m) < \mathsf{band}(t_{i+1}, m)$. Then $\Delta_i > \Delta_{i+1}$, and hence $\mathsf{band}(t_i, n) \leq \mathsf{band}(t_{i+1}, n)$ for all n.

Lemma 6. *At any point in time n, and for any integer α, the total number of observations covered cumulatively by all tuples with band values in $[0..\alpha]$ is bounded by $2^\alpha/\epsilon$.*

Proof. By Proposition 3, each $\mathsf{band}_\beta(n)$ contains at most 2^β distinct values of Δ. There are no more than $1/2\epsilon$ observations with any given Δ, so at most $2^\beta/2\epsilon$ observations were inserted with $\Delta \in \mathsf{band}_\beta$. By Lemma 5, no observations from bands $> \alpha$ will be covered by a node from α. Therefore the nodes in question can cover, at most, the total number of observations from all bands $\leq \alpha$. Summing over all $\beta \leq \alpha$ yields an upper bound of $2^{\alpha+1}/2\epsilon = 2^\alpha/\epsilon$.

The next lemma shows that for any given band value α, only a small number of nodes can have a child with that band value.

Lemma 7. *At any time n and for any given α, there are at most $3/2\epsilon$ nodes in $\mathsf{T}(n)$ that have a child with band value of α. In other words, there are at most $3/2\epsilon$ parents of nodes from $\mathsf{band}_\alpha(n)$.*

Proof. Let $m_{\min}$ and $m_{\max}$, respectively denote the earliest and the latest times at which an observation in $\mathsf{band}_\alpha(n)$ could be seen. It is easy to verify that

$$m_{\min} = \frac{2\epsilon n - 2^\alpha - (2\epsilon n \bmod 2^\alpha)}{2\epsilon} \quad \text{and} \quad m_{\max} = \frac{2\epsilon n - 2^{\alpha-1} - (2\epsilon n \bmod 2^{\alpha-1})}{2\epsilon}.$$

Thus, any parent of a node in $\mathsf{band}_\alpha(n)$ must have $\Delta_i < 2\epsilon m_{\min}$.

Fix a parent node V_i with at least one child in $\mathsf{band}_\alpha(n)$ and let V_j be the rightmost such child. Denote by m_j the time at which the observation corresponding to V_j was seen.

We will show that at least a $(2\epsilon/3)$-fraction of all observations that arrived after time $m_{\min}$ can be uniquely mapped to the pair(V_i, V_j). This in turn implies that no more than $3/2\epsilon$ such V_i's can exist, thus establishing the lemma. The main idea underlying our proof is that the fact that COMPRESS did not merge V_j into V_i implies there must be a large number of observations that can be associated with the parent-child pair (V_i, V_j).

We first claim that $g_j^*(n) + \sum_{k=j+1}^{i-1} g_k(n) \geq g_{i-1}^*(n)$. If $j = i - 1$, it is trivially true. Otherwise, the tuple t_{i-1} is distinct from t_j, and since V_j is a child of V_i (and not V_{i-1}), we know that $\mathsf{band}(t_{i-1}, n) \leq \mathsf{band}(t_j, n)$. Thus no tuple among $t_1, t_2, ..., t_j$ could be a descendant of t_{i-1}. Therefore, $\sum_{k=j+1}^{i-1} g_k(n) \geq g_{i-1}^*(n)$ and the claim follows.

Now since COMPRESS did not merge V_j into V_i, it must be the case that $g_{i-1}^*(n) + g_i(n) + \Delta_i > 2\epsilon n$. Using the claim above, we can conclude that $g_j^*(n) + \sum_{k=j+1}^{i-1} g_k(n) + g_i(n) + \Delta_i > 2\epsilon n$. Also, at time m_j, we had $g_i(m_j) + \Delta_i < 2\epsilon m_j$. Since m_j is at most $m_{\max}$, it must be that

$$g_j^*(n) + \sum_{k=j+1}^{i-1} g_k(n) + (g_i(n) - g_i(m_j)) > 2\epsilon(n - m_{\max}).$$

Finally observe that for any other such parent-child pair $V_{i'}$ and $V_{j'}$, the observations counted above by (V_j, V_i) and $(V_{j'}, V_{i'})$ are distinct. Since there

are at most $n - m_{\min}$ total observations that arrived after $m_{\min}$, we can bound the total number of such pairs by

$$\frac{n - m_{\min}}{2\epsilon(n - m_{\max})}$$

which can be verified to be at most $3/2\epsilon$.

We say that adjacent tuples (t_{i-1}, t_i) constitute a *full pair of tuples* at time n', if $g_{i-1} + g_i + \Delta_i > \lfloor 2\epsilon n' \rfloor$. Given such a full pair of tuples, we say that the tuple t_{i-1} is a *left partner* and t_i is a *right partner* in this full pair.

Lemma 8. *At any time n and for any given α, there are at most $4/\epsilon$ tuples from $\mathsf{band}_\alpha(n)$ that are right partners in a full pair of tuples.*

Proof. Let X be the set of tuples in $\mathsf{band}_\alpha(n)$ that participate as a right partner in some full pair. We first consider the case when tuples in X form a single contiguous segment in $\mathsf{Q_{GK}}(n)$. Let $t_i, ..., t_{i+p-1}$ be a maximal contiguous segment of $\mathsf{band}_\alpha(n)$ tuples in $\mathsf{Q_{GK}}(n)$. Since these tuples are alive in $\mathsf{Q_{GK}}(n)$, it must be the case that

$$g^*_{j-1} + g_j + \Delta_j > 2\epsilon n \qquad\qquad i \le j < i + p.$$

Adding over all j, we get

$$\sum_{j=i}^{i+p-1} g^*_{j-1} + \sum_{j=i}^{i+p-1} g_j + \sum_{j=i}^{i+p-1} \Delta_j > 2p\epsilon n.$$

In particular, we can conclude that

$$2\sum_{j=i-1}^{i+p-1} g^*_j + \sum_{j=i}^{i+p-1} \Delta_j > 2p\epsilon n.$$

The first term in the LHS of the above inequality counts twice the number of observations covered by nodes in $\mathsf{band}_\alpha(n)$ or by one of its descendants in the tree $\mathsf{T}(n)$. Using Lemma 6, this sum can be bounded by $2(2^\alpha/\epsilon)$. The second term can be bounded by $p(2\epsilon n - 2^{\alpha-1})$ since the largest possible Δ value for a tuple with a band value of α or less is $(2\epsilon n - 2^{\alpha-1})$. Substituting these bounds, we get

$$\frac{2^{\alpha+1}}{\epsilon} + p(2\epsilon n - 2^{\alpha-1}) > 2p\epsilon n$$

Simplifying above, we get $p < 4/\epsilon$ as claimed by the lemma. Finally, the same argument applies when nodes in X induce multiple segments in $\mathsf{Q_{GK}}(n)$; we simply consider the above summation over all such segments.

Lemma 9. *At any time n and for any given α, the maximum number of tuples possible from each $\mathsf{band}_\alpha(n)$ is $11/2\epsilon$.*

Proof. By Lemma 8 we know that the number of $\mathsf{band}_\alpha(n)$ nodes that are right partners in some full pair can be bounded by $4/\epsilon$. Any other $\mathsf{band}_\alpha(n)$ node either does not participate in any full pair or occurs only as a left partner. We first claim that each parent of a $\mathsf{band}_\alpha(n)$ node can have at most one such node in $\mathsf{band}_\alpha(n)$. To see this, observe that if a pair of non-full adjacent tuples t_i, t_{i+1}, where $t_{i+1} \in \mathsf{band}_\alpha(n)$, is not merged then it must be because $\mathsf{band}(t_i, n)$ is greater than α. But Proposition 4 tells us that this event can occur only once for any α, and therefore, V_{i+1} must be the unique $\mathsf{band}_\alpha(n)$ child of its parent that does not participate in a full pair. It is also easy to verify that for each parent node, at most one $\mathsf{band}_\alpha(n)$ tuple can participate only as a left partner in a full pair. Finally, observe that only one of the above two events can occur for each parent node. By Lemma 7, there are at most $3/2\epsilon$ parents of such nodes, and thus the total number of $\mathsf{band}_\alpha(n)$ nodes can be bounded by $11/2\epsilon$.

Theorem 2. *At any time n, the total number of tuples stored in $\mathsf{Q}_{\mathsf{GK}}(n)$ is at most $(11/2\epsilon)\log(2\epsilon n)$.*

Proof. There are at most $1 + \lfloor \log 2\epsilon n \rfloor$ bands at time n. There can be at most $3/2\epsilon$ total tuples in $\mathsf{Q}_{\mathsf{GK}}(n)$ from bands 0 and 1. For the remaining bands, Lemma 9 bounds the maximum number of tuples in each band. The result follows.

4 Randomized Algorithms

We present here two distinct approaches for using randomization to reduce the space needed. The first approach essentially samples the input elements, and presents the sample as input to a deterministic algorithm. The second approach uses hashing to randomly cluster the input elements, thus reducing the number of distinct input elements seen by the quantile summary. The sampling-based approaches presented here work only for the cash-register model. The hashing-based approach, on the other hand, works in the more general turnstile model that allows for deletion of elements. However, this latter approach requires that we know the number of elements in the input sequence in advance.

4.1 Sampling-based Approaches

Sampling of elements offers a simple and effective way to reduce the space needed to create quantile summaries. In particular, the space needed can be made independent of the size of the data stream, if we are willing to settle for a probabilistic guarantee on the precision of the summary generated. The

idea is to draw a random sample from the input, and run a deterministic algorithm on the sample to generate the quantile summary. The size of the sample depends only on the probabilistic guarantee and the desired precision for the summary. The following lemma from Manku *et al* [14] serves as a basis for this approach.

Lemma 10 ([14]). *Let $\epsilon, \delta \in (0, 1)$, and let S be a set of n elements. There exists an integer $p = \Theta\left(\frac{1}{\epsilon^2} \log\left(\frac{1}{\epsilon\delta}\right)\right)$ such that if we sample p elements from S uniformly at random, and create an $\epsilon/2$-approximate quantile summary Q on the sample, then Q is an ϵ-approximate summary for S with probability at least $1 - \delta$.*

If the length of the input sequence is known in advance. we can easily draw a sample of size p as required above, and maintain an $\epsilon/2$-approximate quantile summary Q on the sample. Total space used by this approach is $O(\frac{1}{\epsilon} \log(\epsilon p))$, giving us the following theorem.

Theorem 3. *For any $\epsilon, \delta \in (0, 1)$, we can compute with probability at least $1 - \delta$, an ϵ-approximate quantile summary for a sequence of n elements using $O\left(\frac{1}{\epsilon} \log(\frac{1}{\epsilon}) + \frac{1}{\epsilon} \log\log\left(\frac{1}{\epsilon\delta}\right)\right)$ space, assuming the sequence size n is known in advance.*

When the length of the input sequence is not known apriori, one approach is to use a technique called *reservoir sampling* (discussed in detail in another chapter in this handbook) that maintains a uniform sample at all times. However, the elements in the sample are constantly being replaced as the length of the input sequence increases, and thus the quantile summary cannot be constructed incrementally. The sample must be stored explicitly and the observations can be fed to the deterministic algorithm only when we stop, and are certain the elements in the sample will not be replaced. Since the sample size dominates the space needed in this case, we get the following theorem.

Theorem 4. *For any $\epsilon, \delta \in (0, 1)$, we can compute with probability at least $1 - \delta$, an ϵ-approximate quantile summary for a sequence of n elements using $O\left(\frac{1}{\epsilon^2} \log\left(\frac{1}{\epsilon\delta}\right)\right)$ space.*

Manku *et al* [15], used a non-uniform sampling approach to get around the large space requirements imposed by the reservoir sampling. We state their main result below and refer the reader to the paper for more details.

Theorem 5 ([15]). *For any $\epsilon, \delta \in (0, 1)$, we can compute with probability at least $1 - \delta$, an ϵ-approximate quantile summary for a sequence of n elements using $O\left(\frac{1}{\epsilon} \log^2(\frac{1}{\epsilon}) + \frac{1}{\epsilon} \log^2 \log\left(\frac{1}{\epsilon\delta}\right)\right)$ space.*

4.2 The Count-Min Algorithm

The Count-Min (CM) algorithm [4] is a randomized approach for maintaining quantiles when the universe size is known in advance. Suppose all elements are drawn from from a universe $U = \{1, 2, ..., M\}$ of size M. The CM algorithm uses $O(\frac{1}{\epsilon}(\log^2 M)(\log(\frac{\log M}{\epsilon \delta})))$ space to answer any quantile query with ϵ-accuracy with probability at least $1 - \delta$. This is in contrast to the $O(\frac{1}{\epsilon}\log(\epsilon n))$ space used by the deterministic GK algorithm. The two space bounds are incomparable in the sense that their relative quality depends on the relation between n and M. In addition to being quite simple, the main strength of the CM algorithm is that it works in the more general turnstile model, provided all element counts are non-negative throughout its execution. We start with a description of the basic data structure maintained by the CM algorithm and then describe how the data structure is adapted to handle quantile queries. A key concept underlying the CM data structure is that of universal hash families.

Universal Hash Families: For any positive integer m, a family $\mathcal{H} = \{h_1, ..., h_k\}$ of hash functions where each $h_i : U \rightarrow [1..m]$ is a *universal hash family* if for any two distinct elements $x, y \in U$, we have

$$\Pr_{h_i \in \mathcal{H}}[h_i(x) = h_i(y)] \leq 1/m.$$

The set of all possible hash functions $h : U \rightarrow [1..m]$ is easily seen to be a hash family but the number of hash functions in this family is exponentially large. A beautiful result of Carter and Wegman [3] shows that there exist universal hash families with only $O(M^2)$ hash functions that can be constructed in polynomial-time. Moreover, any function in the family can be described completely using $O(\log M)$ bits.

We are now ready to describe the basic CM data structure.

Basic CM Data Structure: An (ϵ_0, δ_0) CM data structure consists of a $p \times q$ table T where $p = \lceil \ln(1/\delta_0) \rceil$ and $q = \lceil e/\epsilon_0 \rceil$, and a universal hash family $\mathcal{H}$ such that each $h \in \mathcal{H}$ is a function $h : U \rightarrow [1..q]$. We associate with each row $i \in [1..p]$, a hash function h_i chosen uniformly at random from the hash family $\mathcal{H}$. The table is initialized to all zeroes at the beginning. Whenever an update (x, c_x) arrives for some $x \in U$, we modify for each $1 \leq i \leq p$:

$$T[i, h_i(x)] = T[i, h_i(x)] + c_x.$$

At any point in time t, let $\mathbf{C}(t) = (C_1, ..., C_M)$ where C_x denotes the sum $\{\sum c_x \mid (x, c_x) \text{ arrived before time } t\}$. When the time t is clear from context, we will simply use $\mathbf{C}$.

Given a query for C_x, the CM data structure outputs the estimate $\hat{C}_x = \min_{1 \leq i \leq p} T[i, h_i(x)]$. The lemma below gives useful properties of the estimate $\hat{C}(x)$.

Lemma 11. *Let $x \in U$ be any fixed element. Then at any time t, with probability at least $1 - \delta_0$:*

$$C_x \leq \hat{C}_x \leq C_x + \epsilon_0 \|\mathbf{C}\|_1.$$

Proof. Recall that by assumption, $C_y \geq 0$ for all $y \in U$ at all times t. It is then easy to see that $\hat{C}_x \geq C_x$ at all times since each update (x, c_x) leads to increment of $T[i, h_i(x)]$ by c_x for each $1 \leq i \leq p$. In addition, any element y such that $h_i(y) = h_i(x)$ may contribute to $T[i, h_i(x)]$ as well but this contribution is guaranteed to be non-negative by our assumption.

We now bound the probability that $\hat{C}_x > C_x + \epsilon_0 \|\mathbf{C}\|_1$ at any time t. Fix an element $x \in U$ and an $i \in [1..p]$. We start by analyzing the probability of the event that $T[i, h_i(x)] > C_x + \epsilon_0 \|\mathbf{C}\|_1$. Let $Z_i(x)$ be a random variable that is defined to be $|\{\sum_{y \in U \setminus \{x\}} C_y \mid h_i(y) = h_i(x)\}|$. Since h_i is drawn uniformly at random from a universal hash family, we have

$$E[Z_i(x)] = \sum_{y \in U} \Pr[h_i(y) = h_i(x)] C_y \leq \frac{\sum_{y \in U \setminus \{x\}} C_y}{q} \leq \frac{\epsilon_0 \|\mathbf{C}\|_1}{e}.$$

Then by Markov's inequality, we have that

$$\Pr\left[T[i, h_i(x)] > C_x + \epsilon_0 \|\mathbf{C}\|_1\right] \leq \Pr\left[T[i, h_i(x)] > \epsilon_0 \|\mathbf{C}\|_1\right] \leq \frac{1}{e}.$$

Thus

$$\Pr\left[\min_{1 \leq i \leq p} T[i, h_i(x)]\right] > C_x + \epsilon_0 \|\mathbf{C}\|_1 \leq \left(\frac{1}{e}\right)^{\ln(1/\delta_0)} \leq \delta_0.$$

CM Data Structure for Quantile Queries: In order to support quantile queries, we need to modify the basic CM data structure to support range queries. A range query $R[\ell, r]$ specifies two elements $\ell, r \in U$ and asks for $\sum_{\ell \leq x \leq r} C_x$. Suppose we are given a data structure that can answer with probability at least $1 - \delta$ every range query to within an additive error of at most $\epsilon \|\mathbf{C}\|_1$. Then this data structure can be used to answer any ϕ-quantile query with ϵ-accuracy with probability at least $1 - \delta$. The idea is to perform a binary search for the smallest element $r \in U$ such that $\hat{R}[1, r] \geq \phi \|\mathbf{C}\|$. We output the element r as the ϕ-quantile. Clearly, it is an ϵ-accurate ϕ-quantile with probability at least $1 - \delta$.

We now describe how the basic CM data structure can be modified to support range queries. For clarity of exposition, we will assume without any loss of generality that $M = 2^u$ for some integer u. We will define a collection of CM data structures, say, $\mathrm{CM}_0, \mathrm{CM}_1, ..., \mathrm{CM}_u$ such that CM_i can answer any range query of the form $R[j2^i + 1, (j + 1)2^i]$ with an additive error of at most $\frac{\epsilon \|\mathbf{C}\|_1}{(u+1)}$. Then to answer a range query $R[1, r]$ for any $r \in [1..M]$, we consider the binary representation of r. Let $i_1 > i_2 > ... > i_b$ denote the bit

positions with a 1 in the representation. In response to the query $R[1, r]$, we return

$$\hat{R}[1,r] = \hat{R}[1,2^{i_1}] + \hat{R}[2^{i_1}+1, 2^{i_1}+2^{i_2}] + ... + \hat{R}[2^{i_1}+...+2^{i_{b-1}}+1, 2^{i_1}+...+2^{i_{b-1}}+2^{i_b}]$$

where $\hat{R}[2^{i_1} + ... + 2^{i_{j-1}} + 1, 2^{i_1} + ... + 2^{i_{j-1}} + 2^{i_j}]$ is the value returned by CM_{i_j} in response to the query $R[2^{i_1} + ... + 2^{i_{j-1}} + 1, 2^{i_1} + ... + 2^{i_{j-1}} + 2^{i_j}]$. Since each term in the RHS has an additive error of at most $\frac{\epsilon\|\mathbf{C}\|_1}{(u+1)}$, we know that

$$R[1,r] \leq \hat{R}[1,r] \leq R[1,r] + \epsilon\|\mathbf{C}\|_1.$$

The Data Structure CM_i**:** It now remains to describe the data structure CM_i for $0 \leq i \leq u$. Fix an $i \in [0..u]$, and let $u_i = u - i$. Define $U_i = \{x_{i,1}, ..., x_{i,2^{u_i}}\}$ to be the universe underlying the data structure CM_i. The element $x_{i,j} \in U_i$ serves as the *unique representative* for all elements in U that lie in the range $[j2^i + 1, (j+1)2^i]$. Thus each element in U is covered by a unique element in U_i. The data structure CM_i is an (ϵ_0, δ_0) CM data structure over U_i where $\epsilon_0 = \frac{\epsilon}{(u+1)}$ and $\delta_0 = \frac{\delta}{(u+1)}$. Whenever an update (x, c_x) arrives for $x \in U_i$, we simply add c_x to the unique representative $x_{i,j} \in U_i$ that covers x.

The following is an immediate corollary of Lemma 11.

Corollary 1. *Let $j \in [1..2^{u-i})$ be a fixed integer. The data structure CM_i can be used to answer the query $R[j2^i + 1, (j + 1)2^i]$ within an additive error of $\epsilon_0\|\mathbf{C}\|_1$ with probability at least $1 - \delta_0$.*

To answer a range query $R[1..r]$, we aggregate answers from up to $(u + 1)$ queries (to the data structures $CM_0, CM_1, ..., CM_u$), each with an additive error of $\epsilon_0\|\mathbf{C}\|_1$ with probability at least $1 - \delta_0$. Using union bounds, we can thus conclude that with probability at least $1 - (u + 1)\delta_0 = 1 - \delta$, the total error is bounded by $(u + 1)\epsilon_0\|\mathbf{C}\|_1 = \epsilon\|\mathbf{C}\|_1$.

Application to Quantile Queries: In order to answer every ϕ-quantile query to within ϵ-accuracy, it suffices to be able to answer ϕ-quantile queries for ϕ-values restricted to be in the set $\{\epsilon/2, \epsilon, 3\epsilon/2, ...\}$ with $\epsilon/2$-accuracy. Given any arbitrary ϕ-quantile query, we can answer it by querying for a ϕ'-quantile and returning the answer, where

$$\phi' = \lceil\frac{\phi}{(\epsilon/2)}\rceil(\epsilon/2).$$

It is easy to see that any $(\epsilon/2)$-accurate answer to the ϕ'-quantile is an ϵ-accurate answer to the ϕ-quantile query.

In order to answer every quantile query to within an additive error of $\epsilon\|\mathbf{C}\|_1$ with probability at least $1 - \delta$, each CM_i data structure is created with

suitably chosen parameters ϵ_0 and δ_0. Since any single range query requires aggregating together at most $(u + 1)$ answers, and there are $2/\epsilon$ quantile queries overall, it suffices to set

$$\epsilon_0 = \frac{\epsilon}{(u+1)} \quad \text{and} \quad \delta_0 = \frac{\delta}{(u+1)} \cdot \frac{2}{\epsilon}.$$

The space used by each CM_i data structure for this choice of parameters is $O\left(\frac{1}{\epsilon}(\log M)(\log(\frac{u}{\epsilon\delta}))\right)$. Hence the overall space used by this approach is $O\left(\frac{1}{\epsilon}(\log^2 M)(\log(\frac{\log M}{\epsilon\delta}))\right)$.

The CM algorithm strongly utilizes the knowledge of the universe size. In absence of deletions, stronger space bounds can be obtained by exploiting the knowledge of the universe size. For instance, the *q-digest* summary of Shrivastava et al [18] described in Section 5.3 is an ϵ-approximate quantile summary that uses only $O(\frac{1}{\epsilon}\log M)$ space. Moreover, the precision guarantee of a q-digest is deterministic. However, the strength of the CM algorithm is in its ability to handle deletions. To see another interesting example of an algorithm that uses randomization to handle deletions, the reader is referred to the RSS algorithm [8, 9].

5 Other Models

So far we have considered deterministic algorithms with absolute guarantees and randomized algorithms with probabilistic guarantees mainly in the setting of the *cash register* model [7]. However, quantile computations can be considered under different streaming models. We have seen in Section 4.2 that the cash register model can be extended to the *turnstile* model [17], in which the stream can include both insertions *and* deletions of observations. We can also consider settings in which the complete dataset is accessible, at a cost, allowing us to perform multiple (expensive) passes. Settings in which other features of this model have been varied have been studied as well. For example, one may know the types of queries in advance [15], or exploit prior knowledge of the precision and range of the data values [8, 4, 18].

While algorithms from two different settings cannot be directly compared, it is still worth understanding how they may be related. In this section we will briefly consider a small sample of alternative models where the ideas presented in this chapter are directly applicable — either used as black box components in other algorithms, or adapted to a new setting with relatively minor modifications — and compare the modified algorithms to other algorithms in the literature. In each setting, we first briefly present an algorithm that follows naturally from the ideas presented in this chapter, then present an algorithm from the literature specifically designed for the new setting. Some of these models will be covered in more depth in later chapters in this book.

5.1 Deletions

In many database applications, a summary is stored with large data sets. For queries in which an approximate answer is sufficient, the query can be cheaply executed over the summary rather than over the entire, large, dataset. In such cases, we need to *maintain* the summary in the face of operations on the underlying data set. This setting differs from our earlier model in an important way: both insertion and deletion operations may be performed on the underlying set. We focus here on *well-formed inputs* where each deletion corresponds uniquely to an earlier insertion. When deletions are possible, the size of the dataset can grow and shrink. The parameter n can no longer denote both the number of observations that have been seen so far and the current time. In the turnstile model we will, instead, denote the current time by t and let $n = n(t)$ denote the current number of elements in the data set, and $m = m(t)$ will denote $\max_{1 \leq i \leq t} n(i)$.

The difficulty in guaranteeing precise responses to quantile queries in the turnstile model lies in recovering information once it has been discarded from the summary. In particular, when there are only insertions, the error allowed in the ranks of elements in the summary grows monotonically. But when deletions occur, we may need to greatly refine the rank information for existing elements in the summary. For instance, if we insert n elements in a set, then the allowed error in the rank of any observation is ϵn. But now if we delete all but $1/\epsilon$ of the observations, then the ϵ-approximate property now requires us to know the rank and value of each remaining element in the set exactly!

However, we can deal with similarly useful, but more tractable, problems by investigating slightly more relaxed settings. Gibbons, Matias, and Poosala [5, 6] were the first to present an algorithm to maintain a form of quantile summary in the face of deletions in the case where multiple passes over the data set are possible, although expensive. In situations where a second pass is impossible, we relax the requirement that we return a value v that is (currently) in S. In this latter setting we can gain some further traction by weakening the guarantees we offer. Deterministic algorithms can temper their precision guarantees as a function of the input pattern (performing better on "easy" input patterns, and worse on "hard" patterns), and randomized algorithms can offer probabilistic, rather than absolute, guarantees.

Deterministic Algorithms with input dependent guarantees

We first extend the GK [10] algorithm in a natural way to support a DELETE(v) operation. We will see that for certain input sequences we can maintain a guarantee of ϵ-precision in our responses to quantile queries — even in the face of deletions.

DELETE(v) Find the smallest i, such that $v_{i-1} \leq v < v_i$. (Note that i may be 1 if the minimum element was already deleted). To delete v

we must update $\texttt{rmax}_{\mathsf{GK}}(v_j)$ and $\texttt{rmin}_{\mathsf{GK}}(v_j)$ for all observations stored in the summary. For all $j > i$, $\texttt{rmin}_{\mathsf{GK}}(v_j)$ and $\texttt{rmax}_{\mathsf{GK}}(v_j)$ are reduced by 1. Further, we know that $\texttt{rmax}_{\mathsf{GK}}(v_{j-1}) < \texttt{rmax}_{\mathsf{GK}}(v_j)$. If our estimate of $\texttt{rmax}_{\mathsf{GK}}((v_j))$ is reduced, such that $\texttt{rmax}_{\mathsf{GK}}(v_{j-1}) = \texttt{rmax}_{\mathsf{GK}}(v_j)$, then decrement $\texttt{rmax}_{\mathsf{GK}}(v_{j-1})$ by 1. Deletion of an observation covered by v_i is implemented by simply decrementing g_i. This decrements all $\texttt{rmin}_{\mathsf{GK}}(v_j)$ and $\texttt{rmax}_{\mathsf{GK}}(v_j)$, for $j > i$, as required. The pseudo-code in Figure 7 that decrements Δ_{i-1} maintains the invariant that $\texttt{rmax}_{\mathsf{GK}}(v_{j-1}) < \texttt{rmax}_{\mathsf{GK}}(v_j)$. Finally, v_i is removed from the summary if $g_i = 0$ and i was not one of the extreme ranking elements ($i = 0$, or $i = s - 1$).

```
DELETE(v)
    g_i = g_i − 1
    if (g_i = 0) and
        ((i ≠ s − 1) or (i ≠ 0)) then
        Remove t_i from Q
    end if
    for j = (i − 1) to 0
        if (Δ_j ≥ (g_{j+1} + Δ_{j+1}))
            then Δ_j = Δ_j − 1
            else break
        end if
    end for
```

Fig. 7. Pseudocode for DELETE

Because we do not delete v_i until *all* observations covered by the tuple are also deleted, it is no longer the case that all $v_i \in \mathsf{Q}_{\mathsf{GK}}$ are members of the underlying set.

At time t, a GK summary $\mathsf{Q}_{\mathsf{GK}}(\epsilon)$ will never delete a tuple if the resulting gap would exceed $2\epsilon n(t)$. By Proposition 1 the precision of the resulting summary is the maximum, over all i, of $(\texttt{rmax}_{\mathsf{Q}_{\mathsf{GK}}(\epsilon)}(v_{i+1}) - \texttt{rmin}_{\mathsf{Q}_{\mathsf{GK}}(\epsilon)}(v_i))/(2n(t))$. In the simple setting without deletions, the actual precision of $\mathsf{Q}_{\mathsf{GK}}(\epsilon)$ is always $\leq \epsilon$. Unfortunately it is impossible to bound this precision in the face of arbitrary input in the setting of the turnstile model. In particular, after deletions, $\mathsf{Q}_{\mathsf{GK}}(\epsilon')$ may not have precision ϵ' in the turnstile model. However, in many cases, application-specific behavior can allow us to bound the maximum $\texttt{rmax}_{\mathsf{GK}}(v_{i+1}) - \texttt{rmin}_{\mathsf{GK}}(v_i)$ and hence construct summaries with guaranteed ϵ precision.

Example: bounded deletions

Perhaps the simplest example of application-specific behavior is when we know, in advance, some bound on the the impact of deletions on the size of the data set. Let $\alpha < 1$ denote a known fixed lower bound, such that for any time t, $n(t) > \alpha m(t)$. If $\mathsf{Q}_{\mathsf{GK}}(\epsilon')$ is an ϵ'-approximate quantile summary, then COMPRESS will never delete a tuple if the resulting gap would exceed $2\epsilon' n(t)$. Given that at all time $t, n(t) \leq m(t)$, we know that for all tuples $\mathtt{rmax}_{\mathsf{GK}}(v_{i+1}) - \mathtt{rmin}_{\mathsf{GK}}(v_i) \leq 2\epsilon' m(t)$, and the precision at time t is then bounded above by $\epsilon' m(t)/n(t)$. $n(t) > \alpha m(t)$, and therefore the precision $\epsilon' m(t)/n(t) \leq \epsilon'/\alpha$. Consequently, if we choose $\epsilon' = \alpha\epsilon$, then a summary $\mathsf{Q}_{\mathsf{GK}}(\epsilon')$ has precision ϵ even after deletions. This loose specification is too broad to derive specific bounds on the size of our data structures. We proceed now to a concrete example in which analysis of the application-specific behavior allows us to demonstrate stronger limits on the size and precision of the resulting summary.

Example: session data

The AT&T network monitors the distribution of the duration of *active* calls over time [8] through the collection of Call Detail Records (CDRs). The start-time of each call is inserted into the quantile summary when the call begins. When the call ends it is no longer active and the start-time is deleted from the summary. At any time there is one observation in the set for each active call, and the duration of the call is simply the difference between the current time and the stored start time of the call. (The median duration is the current time minus the start time of the median observation in the set.)

We first show that the size of the GK summary structure will be constant.

Session data arrives at the summary in order of start time. Consequently, new observations are monotonically increasing (although deletions may occur in arbitrary order). Each new observation is a new maximum, and we know its rank exactly.

Proposition 5 *If the input sequence is monotonically increasing, the data structure $\mathsf{Q}_{\mathsf{GK}}(\epsilon)$ uses only $O(1/\epsilon)$ space.*

Proof. The exact rank of each newly arriving observation is always known. Hence, the Δ value of each tuple in the summary is always 0. It then follows that all tuples are siblings in the tree representation. Consequently, after we run COMPRESS, each tuple except the leftmost tuple is a right partner of a full tuple pair, and $g_{i-1} + g_i + \Delta_i > 2\epsilon n(t)$ in $\mathsf{Q}_{\mathsf{GK}}(\epsilon)$. Thus if there are $(k + 1)$ tuples in $\mathsf{Q}_{\mathsf{GK}}(\epsilon)$, then summing over the k full tuple pairs we get $\sum_{i=1}^{k}(g_{i-1} + g_i + \Delta_i) \geq k(2\epsilon n(t))$. Since $\sum \Delta_i = 0$ and $\sum g_i \leq n(t)$, we get $2n(t) \geq k(2\epsilon n(t))$, and $k \leq \frac{1}{\epsilon}$. Since we run COMPRESS after $2/\epsilon$ new observations are added, $\mathsf{Q}_{\mathsf{GK}}(\epsilon)$ contains $O(1/\epsilon)$ tuples at all times t, regardless of the size of $n(t)$.

Proposition 5 tells us only about the size of $\mathsf{Q_{GK}}$. By Proposition 1, $\mathsf{Q_{GK}}$ will have precision ϵ' at time t if the difference in rank between any two consecutive stored tuples, $\mathtt{rmax_{GK}}(v_{i+1}) - \mathtt{rmin_{GK}}(v_i)$, is $\leq 2\epsilon'n(t)$. In our example, the expected difference in rank follows from the observation that phone calls, (for example the trace data from [8]), are typically modeled as having exponentially distributed lifetimes.

Proposition 6 *For sessions with exponentially distributed lifetimes and monotonically increasing arrivals, for a given ϵ, the expected difference in rank between consecutive tuples in $\mathsf{Q_{GK}}(\epsilon)$ at time t is $\leq 2\epsilon E(n(t))$, where $E(n(t))$ is the expected number of elements in S at time t.*

Proof. Fix any i. Let t' denote the most recent time at which COMPRESS deleted any tuples that lay between v_i and v_{i+1}. Let Δr denote the difference $(\mathtt{rmax_{GK}}(v_{i+1}) - \mathtt{rmin_{GK}}(v_i))$ at time t'. Δr must have been $\leq 2\epsilon n(t')$. Let $d = d(t', t)$ denote the total number of deletions that occurred between times t' and t. Then $n(t) \geq n(t') - d$.

The exponential distribution is "memoryless" — all calls are equally likely to terminate within a given interval. Therefore, if the probability that a call terminates within the interval $[t', t]$ is p, then the expected value of the total number of deletions, d, during $[t', t]$ is $pn(t')$. Similarly, the expected value of the number of deletions falling between v_i and v_{i+1} during $[t', t]$ is $p\Delta r$. At time t the expected value of $(\mathtt{rmax_{GK}}(v_{i+1}) - \mathtt{rmin_{GK}}(v_i))$ is $(1 - p)\Delta r$. The expected value of $n(t)$ is $\geq (1 - p)n(t')$. Given that $\Delta r \leq 2\epsilon n(t')$, we have $(1 - p)\Delta r \leq 2\epsilon(1 - p)n(t') \leq 2\epsilon n(t)$.

By Proposition 5, the size of $\mathsf{Q_{GK}}(\epsilon)$ will be $O(1/\epsilon)$.

To be more concrete, in the CDR trace data described in [8], the RSS summary has $\epsilon = 0.1$ precision, and has a maximum memory footprint of 11K bytes. Assuming that we store 12 bytes per tuple, the GK algorithm, in the same memory footprint, should be able to store more than 900 tuples. We expect the typical gap between stored tuples to be commensurate with a precision of roughly .0011 — about 100 times more accurate than the RSS summary. This demonstrates the potential payoff of domain-specific analysis, but it is important to recall that this analysis provides no guarantees in the general case.

Probabilistic guarantees across all inputs

The GK algorithm above exploited structure that was specific to the given example. However, more adversarial cases are easy to imagine. An adversary, for example, can delete all observations except for those that lie between two consecutive tuples in the summary. In such cases either there can be no precision guarantee (we will not be able to return even a single observation from the data set), or else the original "summary" must store every observation — providing no reduction in space. Thus, when details of the application are not

known, algorithms such as GK are unsatisfactory. Even when the expected behavior of an application is known, there may be a chance that the input sequence is unexpectedly adversarial.

Fortunately, there is a much better approach than aiming for absolute guarantees in the face of deletions. Randomized algorithms that summarize the number of observations within a range of values (c.f. RSS [8] or CM [4]) can give probabilistic guarantees on precision and upper bounds on space for *any* application, without requiring case by case analysis. The Count-Min algorithm is described in Section 4.2. Assuming that observations can take on any one of M values, then CM summarizes a sample in space $O\left(\frac{1}{\epsilon}(\log^2 M)(\log(\frac{\log M}{\epsilon\delta}))\right)$. This summary is ϵ-approximate with probability at least $1 - \delta$.

5.2 Sliding Window Model

In some applications, we need to compute order statistics over the W most recent observations, rather than over the entire stream. When W is small enough to fit into memory, it suffices to store the last W observations in a circular buffer and compute the statistics exactly. However, when W is itself very large, then we must *summarize* a *sliding window* of the most recent W elements of the stream. A sliding window is a case where observations are being deleted in a systematic fashion based on the time of arrival. The chief difficulty in such sliding window quantile summaries compared to the summaries over entire streams is that, as the window slides, we need to remove observations from the summary — as in the turnstile model. However, unlike the strict turnstile model where the deletions are delivered to the summary from an external source (and, we presume, are properly paired to an undeleted input), we do not have a complete ordered record of observations, and hence do not know what value needs to be deleted at any given time. On the other hand, deletions in the sliding window model are not arbitrarily distributed; they have a nice structure that can be exploited.

Sliding windows may be either *fixed* or *variable* size. In a fixed sliding window of size W, each newly arrived observation (after the first W arrivals) is paired with a deletion of the oldest observation in the window. Thus, in the steady state, the window always covers precisely W elements. Variable sized sliding windows decouple the arrivals from deletions — at any point in time *either* a new observation arrives or the oldest observation is deleted. A long string of arrivals increases the size of the window; a long string of deletions can reduce the size of the window. (We may sometimes exploit an upper bound on the size of the variable window, if such a bound is known in advance).

Fixed Size Windows

We can implement a trivial fixed sliding window summary with precision ϵ by dividing the input stream into blocks of $\epsilon W/2$ consecutive observations.

We summarize each block with an $\epsilon/2$ precision quantile summary. The block summarizing the most recent data is *under construction*. We add each new arrival to it until it contains $\epsilon W/2$ observations, and is considered *complete*. Once the block is complete it is no longer modified[5] We store only the summaries of the last $2/\epsilon$ blocks. When a single observation in a block exits the window (it is "deleted"), we mark that block *expired*, and do not include it in our combined summary. These block summaries cover at most the last W, and at least the last $W - \epsilon W/2$, observations. By Corollary 1, the combined summary has a precision of $\epsilon/2$. If the most recent block has only just been started, and covers a very few observations, then our combined summary is missing up to $\epsilon W/2$ observations. In the worst case all the missing observations have values that lie between our current estimate and the true ϕ quantile of interest. Even so, they could increase the error in rank by at most $\epsilon W/2$, keeping the total error below ϵW, ensuring that our combined sliding window summary has precision ϵ. The summary for each individual block uses $O(\frac{1}{\epsilon}\log(\epsilon^2 W))$ space, and the aggregate uses $O(\frac{1}{\epsilon^2}\log(\epsilon^2 W))$ space.

Arasu and Manku [2] employ the same basic structure of maintaining a set of summaries over fixed size windows in the input stream, but use a more sophisticated approach improves upon the space bound achieved by the simple algorithm above. Their algorithm suffers a blowup of only a factor of $O(\log(1/\epsilon))$ for maintaining a summary over a window of size W, compared to a blowup of $\Omega(1/\epsilon)$ in the simple implementation. When ϵ is small (say .001), this improvement is significant.
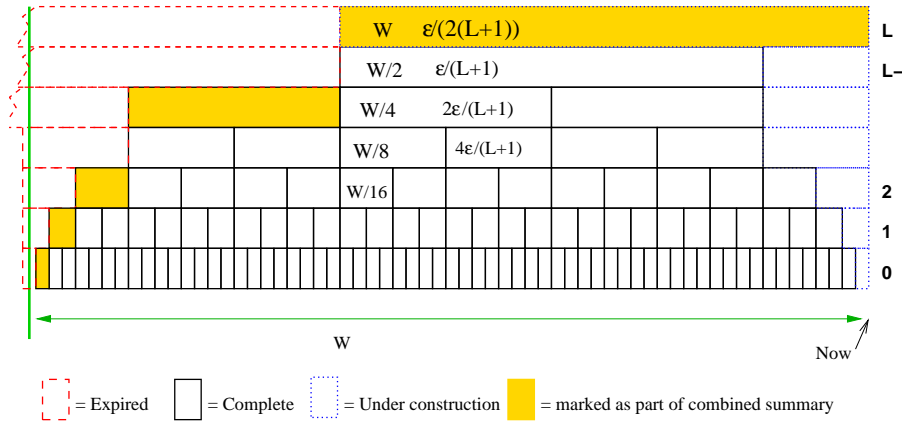

Fig. 8. Graphical representation of levels in Arasu-Manku Algorithm.

---

[5] Lin et al [13] use a variant of this simple approach, but construct $\epsilon W/4$ size blocks, and call `prune` on completed blocks.

Arasu and Manku use a data structure with $L+1$ levels where each level covers the stream by blocks of geometrically increasing sizes. At each point in time, exactly one block in each of the $L+1$ levels is under construction. Each block is constructed using the GK algorithm until it is complete. It may seem that summarizing the entire window in $L+1$ different ways, with $L+1$ sets of blocks, would increase the required space, but, in fact, the total space requirement is *reduced* due to two basic observations.

First, once a block is complete, we can call `prune` to reduce the required space to $O(1/\epsilon)$. Second, for a given precision, larger blocks summarize the data stream more efficiently than smaller blocks — each stored tuple covers more observations. On the other hand, small blocks result in fewer lost observations when we discard the oldest block. The Arasu-Manku algorithm summarizes the window by combining non-overlapping blocks of *different* sizes. It summarizes most of the window, efficiently, using large blocks with fine precision (saving space due to the large block size). We can summarize the tail end of the window using coarser precision smaller blocks (saving space by the coarse precision), and bounding the number of lost observations at the very end of the window to the size of the smallest block we use.

Specifically, [2] divides the input stream into blocks in $L+1$ different ways, where $L = \log_2 (4/\epsilon)$. Each decomposition is called a *level*, and the levels are labeled 0 through L. As we go up each level the block size, denoted by N_ℓ, doubles to $2^\ell \epsilon W/4$, and the precision, denoted by ϵ_ℓ, is halved to $\frac{2^{(L-\ell)}\epsilon}{2(L+1)}$.

Proposition 7 *The Arasu Manku window algorithm can summarize a fixed window of the W most recent observations, using at most $L+1$ non-overlapping blocks from its summary structure.*

Proof. The most recent observations are covered by the level L block currently under construction. For any $0 \leq \ell \leq L$, let η_ℓ denote the number of observations that need to be "covered" by blocks from level 0 through ℓ. We start by defining η_{L-1} to be the number of observations not covered by a level L block; clearly $\eta_{L-1} < W$. If $\eta_{L-1} \geq W/2$, then we can include a level $L-1$ block of size $W/2$ to cover $W/2$ additional observations, and set $\eta_{L-2} = \eta_{L-1} - W/2 < W/2$. Otherwise, $\eta_{L-2} = \eta_{L-1} < W/2$. It is easy to see $\eta_{L-\ell} < W2^{-\ell+1}$, and consequently at most one block from each level will be used. There are $L+1$ levels.

Lemma 12. *The Arasu Manku window algorithm implements an ϵ-approximate quantile-summary of a fixed window of the W most recent observations.*

Proof. Let A denote the combined summary of non-overlapping blocks. From the analysis in the proof of Lemma 1 the maximum gap between consecutive stored observations in A is $\sum_{\ell=0}^{L} \hat{i}_l N_l \epsilon_\ell$, where $\hat{i}_\ell$ is 1 if a block from level ℓ is present in A, and 0 if it is not. But

$$N_\ell \epsilon_l = 2^l \epsilon \left(\frac{W}{4}\right) 2^{L-\ell} \frac{\epsilon}{2(L+1)} = \epsilon \left(\frac{W}{4}\right) 2^L \frac{\epsilon}{2(L+1)},$$

a value that is independent of l. Since $2^L = 4/\epsilon$, this can be simplified to $\frac{\epsilon W}{2(L+1)}$. Consequently, the maximum gap is simply $\frac{\epsilon W}{2(L+1)}$ times the number of blocks used in A. By Proposition 7 there are at most $L + 1$ blocks in A.

Therefore, the maximum gap is at most $(L + 1)\frac{\epsilon W}{2(L+1)}$, or $\epsilon W/2$.

An upper bound on the number of unsummarized observations in the window is just the smallest block size in the summary, namely $N_0 = \epsilon W/4$. Combining this with the precision of A allows us to answer order-statistic queries with precision $3\epsilon/4$ (which is less than ϵ).

Lemma 13. *The Arasu Manku algorithm can answer quantile queries with ϵ precision over a fixed window of W elements in $O(\frac{1}{\epsilon} \log \frac{1}{\epsilon} \log W)$ space.*

Proof. There are at most $2^{L-\ell}$ complete blocks at each level, and those are pruned to $O(1/\epsilon_\ell)$ space. There are $L = O(\log 1/\epsilon)$ levels; the complete blocks therefore use

$$\sum_{\ell=1}^{L} \left(\frac{2^{L-l}}{\epsilon_\ell} \right) = \sum_{\ell=1}^{L} \left(\frac{2^{L-\ell}(2(2L+2))}{\epsilon 2^{L-\ell}} \right) = \frac{2L(2L+2)}{\epsilon} = O\left(\frac{L^2}{\epsilon} \right).$$

So the aggregate space used by the completed blocks at all levels is $O(\frac{1}{\epsilon} \log^2 \frac{1}{\epsilon})$.

At each level ℓ, there is also at most one block that is still under construction. At its largest, just before it is completed, that block uses $O(\frac{1}{\epsilon_l} \log \epsilon_\ell N_\ell)$ space in the worst case. Expanding ϵ_ℓ (also, as noted above, $\epsilon_\ell N_\ell = \frac{\epsilon W}{2(L+1)}$) yields that the space used by a level ℓ block in construction is $O(2^{\ell-L}\frac{2(L+1)}{\epsilon} \log \frac{\epsilon W}{2(L+1)})$. Summing over all levels ℓ gives us a geometric series whose sum is

$$O\left(\frac{2(L+1)}{\epsilon} \log \frac{\epsilon W}{2(L+1)} \right) = O\left(\frac{2(\log \frac{4}{\epsilon} + 1)}{\epsilon} \log \frac{\epsilon W}{2(L+1)} \right) = O\left(\frac{1}{\epsilon} \log \frac{1}{\epsilon} \log W \right).$$

The combined space is just the sum of the completed blocks and the blocks under construction, namely $O(\frac{1}{\epsilon} \log \frac{1}{\epsilon} \log W + \frac{1}{\epsilon} \log^2 \frac{1}{\epsilon})$, which can be simplified to $O(\frac{1}{\epsilon} \log \frac{1}{\epsilon} \log W)$ since the case when $W < \frac{1}{\epsilon}$ can be trivially solved using $O(\frac{1}{\epsilon})$ space.

Variable Size Windows

Although the algorithms described above assume that W is fixed, they can be extended in a straightforward way to handle variable sized windows. At any point in time we assume that the maximum window size is some W. If the actual window size differs from W by more than a factor of 2, then we alter our assumed window size to $2W$ or $W/2$, and update our data structures accordingly.

We first consider the case of a sliding window whose size has grown. We increase our assumed window size to $2W$. That this is generally possible follows

from the calculation of the maximum gap between the minimum and maximum rank of two consecutive stored tuples in the combined summary. If we have an ϵ-approximate summary of a window of size W, then the maximum such gap is of size $2\epsilon W$. If the window size is increased to $2W$, then that data structure can answer queries to a precision of $\epsilon/2$.

In particular, for the trivial fixed window algorithm described earlier, as the window size grows, we do not alter the block size or the precision per block. We continue to add enough blocks of size $\epsilon W/2$ to summarize our entire window. When the number of blocks increases to $4/\epsilon$ (corresponding to a window size of $2W$), we simply merge every adjacent pair of blocks into a single block of size ϵW.

Extending the Arasu-Manku summary to accomodate a window that has grown beyond W is just as easy. Recall that the number of levels, L, is a function of ϵ and not W. So as W changes, the number of levels remain constant. However, a block at level l is now twice the size as before. We already have blocks of the required size of the new level l in the old level $l + 1$ summary. Fortunately, those blocks are twice as precise as needed. To proceed with a new W of twice the size, then, we simply discard the level 0 blocks, and rename each level l as level $l - 1$. To construct level L, the highest level, we simply duplicate the old level L. Although the nominal size of those blocks were of size W, the *expired* block is of no consequence, and the block that is *under construction* is truncated to the current window size — which is guaranteed to be less than or equal to $W - 1$.

We now consider the case of a sliding window whose size has shrunk significantly. It is easy to see that we can accommodate any size window smaller than W by accepting a space blowup of at most a factor of $O(\log W)$. We can treat any fixed window summary as a black box, and simultaneously maintain summaries for window sizes $W, W/2, W/4, ...,$ and so on. Each time the actual window size halves, we can discard the summary over the largest window.

Lin et al [13] consider another variant of the sliding window model. In their "n of N" model, we once again summarize a fixed window consisting of the W latest observations. However a query for the ϕ-quantile can be qualified by any integer $w \leq W$, such that the returned value must be the observation with rank $w\phi$ of the most recent w observations. It should be clear that the variable window extension to the Arasu Manku algorithm must also be able to answer such queries with precision ϵ. If not, the algorithm would not be able to accurately answer subsequent quantile queries after the window shrank by $W - w$ consecutive deletions.

5.3 Distributed Quantile Summaries

The summary algorithms for streaming data described so far consider a setting in which the data can be observed at a centralized location. In such a setting, the GK algorithm is more space-efficient than algorithms in the `combine` and `prune` family.

However, in some settings the input stream is not observable at a single location. The aggregate quantile summary over a data set must then be computed by merging summaries over each of the subsets comprising that total set. For example, it may be desirable to split an input stream across different nodes in a large cluster, in order to process the input stream in parallel. Alternatively, in sensor networks whose nodes are organized into a tree, nodes may summarize the data in their subtree in order to avoid the communication costs of sending individual sensor readings to the root. In such cases, each site produces a summary of a subset of the stream and then passes the summary to another location where it is merged with summaries of other subsets until the entire stream is summarized. Algorithms that are built out of `combine` and `prune` can be extended naturally to such settings.

Nodes in sensor networks are typically resource-scarce. In particular, they have very limited memory and must conserve power. Summary algorithms over sensor networks must therefore optimize transmission cost between nodes (communication costs are the dominant drain on power). Nodes in sensor networks typically send their sensor readings up to the root, through a tree-shaped topology. In order to reduce the transmission cost, each node may aggregate the data from their children and summarize it before passing it on. In this book, Chapter VI.1 ("Sensor Networks", by Madden) describes sensor networks in more detail. Section 1.3 of [11] discusses the relation between streaming data and sensor networks. We briefly discuss here two summary algorithms over sensor networks (more details are available in the corresponding papers). Both Greenwald and Khanna [11] and Shrivastava et al. [18] support efficient ϵ-approximate quantile queries over sensor networks. Both can be characterized as applications of `combine`. One ([11]) uses `prune` to manage communication costs, while the other ([18]) uses a variant of COMPRESS.

A general algorithm using `combine` and `prune`

A simple algorithm to build quantile summaries over sensor networks collects the summaries of all the children of a node, and applies `combine` to produce a single summary that is sent to the parent. The operation `prune` is then applied to reduce the size of the data transmitted up the tree. Unfortunately, each application of `prune` to reduce the input to a buffer of size K can result in a loss of precision of up to $1/(2K)$. If the network topology is a tree of large depth, then either the aggregate loss of precision is too high, or else K must be very large and little reduction of communication costs can be achieved. Consequently a slightly more complex strategy is required.

The general approach of the algorithms in [11] is to decouple the combining tree from the sensor network routing tree. The parent of a node in the combining tree may not be the immediate parent in the routing tree – rather the parent may be any ancestor at an arbitrary height up the tree. Physical nodes in the sensor network pass all summaries up through their parents in the routing tree until they hit the node that is considered the parent in the

combining tree. The combining tree, and not the routing topology, controls the number of `combine` and `prune` operations executed by the algorithm. Intuitively, the goal of this algorithm is to build the widest, shallowest, such tree subject to minimizing the worst case amount of data sent from a child to its parent in the underlying physical topology. The paper shows that, regardless of topology, an $\epsilon-$approximate quantile summary can be constructed over a sensor network with n nodes using a maximum per-node transmission cost of $O(\log^2 n/\epsilon)$. Let h, the "height" of the sensor network, denote the number of hops from the root of the network to the farthest leaf node. If h is known, then an embedding with a maximum per-node transmission cost of $O((\log n \log \frac{h}{\epsilon})/\epsilon)$ is achievable. Finally, if h is known to be smaller than $\log n$, then the maximum per-node transmission cost can be bounded by $O(h/\epsilon)$.

Optimized algorithm when the range of values is known

Shrivastava et al [18] study a slightly different setting in which observations can only take on integer values in the range $[1..M]$, where M is known in advance. Further, there is no requirement that the value v, returned by a quantile query, must be an element of S. Their algorithm is therefore able to pursue a different strategy. Rather than calling `prune`, which may lose precision on every call, it calls a variant of COMPRESS, which reduces the size of the summary as much as it can, while still preserving the precision. Thus their COMPRESS' operation, which we will describe shortly, can be performed after each call to `combine`, at each node in the network.

The basic strategy is to construct a summary Q, called a *q-digest*, that consists of a sparse binary tree over the data range $1..M$. Each node b in the tree, referred to as a "bucket", maintains a count $b.g$ that represents observations that fell between the minimum and maximum value of the bucket. Each bucket has two children, covering the lower and upper halves of its range. To keep the summary small, COMPRESS' moves observations in underpopulated buckets up the tree, to levels where larger ranges can be covered less precisely by fewer buckets. Buckets with a count of 0 are elided.

The `combine` operation when applied to a pair of such summaries simply sums the counts in corresponding buckets. Insertion of a new sensor value v into a summary Q is implemented by `combine`(Q, Q'), where Q' is a new summary consisting of only the single bucket $[v]$ with count $= 1$. COMPRESS' is called after each `combine` operation.

The precise behavior of COMPRESS' is a depth-first tree-walk, starting at the leaves. It checks to see whether a bucket can be merged into its parent. When visiting a bucket (other than the root, which has no sibling or parent), it sums up the count of observations in the bucket, its sibling bucket, and its parent's bucket. If the bucket has a sum $\leq \lfloor \frac{\epsilon n}{\log M} \rfloor$, then COMPRESS' deletes the bucket, and adds the count to its parent.

Proposition 8 *The maximum count in any non-leaf node in* Q *is* $\frac{\epsilon n}{\log M}$.

g=1

g=2

g=1

g=2 g=2 g=2 g=1 g=2 g=1

g=6 g=4 g=1 g=2   g=2   g=1 g=2

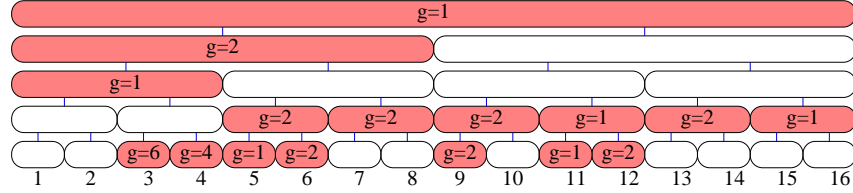1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Fig. 9. An example q-digest, Q. Each non-empty bucket is labeled by g, the count of observations within that bucket. Q has $M = 16, n = 32$, and $\epsilon = .25$. It follows from these values that if the aggregate count in two siblings and their parent is $\leq (32 \times (.25))/4 = 2$, then COMPRESS' will delete the two children and merge them into their parent. If n were 48, then the allowed aggregate count would be 3. In such a case, [1..4] could be merged into [1..8], and both [13..14] and [15..16] could be merged together into [13..16]. [9..10] and [11..12] would be merged into [9..12]. This would open up the subtree under [9..12]. After the next insertion (assuming it did not lie in the range [9..12]), then the 2 observations in [9] would be moved up into [9..10], and both [11] and [12] merged into [11..12].

Proof. Non-leaf nodes only cover new observations by either COMPRESS' deleting their children, or by `combine` taking two q-digests and merging two corresponding buckets. The first case trivially maintains the bound, because COMPRESS' will never delete a pair of children if the sum of the children and their parent exceeds $\frac{\epsilon n}{\log M}$. In the second case, when `combine` combines two q-digests containing n_1 and n_2 observations respectively, the combined digest contains $n = n_1 + n_2$ observations. Prior to `combine` the two corresponding buckets contained fewer than $\epsilon n_1 / \log(M)$ and $\epsilon n_2 / \log(M)$ observations, respectively. After summing they contain fewer than $\frac{\epsilon(n_1 + n_2)}{\log M} = \frac{\epsilon(n)}{\log M}$ observations, and the proposition holds.

We can use Q to answer quantile queries over S. The minimum rank of a value v in the data set S is computed by adding 1 to the cumulative counts in all of the buckets that contain only values less than v. The maximum rank of v is computed by adding to the minimum rank, the sum of the counts in all non-leaf buckets whose minimum value is less than v, but whose maximum value is greater than or equal to v. For example, in Figure 9, the minimum rank of the value 5 is 12, because 5 could be the first value after the 6 instances of 3, the 4 instances of 4, and the single observation that lies somewhere within [1,4]. The two observations within [1,8] and the single observation within [1,16] *may* be values that are greater than or equal to 5, and hence cannot be counted in the minimum possible rank. On the other hand, there *may* be values that precede 5, and hence could increase the rank of 5 to be as high as 15.

Theorem 6. *A q-digest* $Q(\epsilon)$ *summarizing a dataset* S *is an* ϵ-*approximate quantile summary using at most* $3 \log(M)/\epsilon$ *buckets, regardless of the size of* $|S|$.

We first establish the upper bound on the size of Q. Let k denote the number of surviving non-zero buckets in Q. For $b \in \mathsf{Q}$, let $b.g$ be the count of observations in b, and let $s(b)$ denote the sibling of bucket b in Q, and $p(b)$ denote its parent in Q. Every non-empty bucket b obeys $b.g + s(b).g + p(b).g > \frac{\epsilon n}{\log M}$. Each bucket can appear at most once as a left sibling, once as a right sibling, and at most once as a parent. Summing this inequality over all k surviving buckets, we get $3n \geq \sum_{b \in \mathsf{Q}}(b.g + s(b).g + p(b).g) > k\frac{\epsilon n}{\log M}$. Therefore, $3\log(M)/\epsilon > k$.

We next show that $\mathsf{Q}(\epsilon)$ is an ϵ-approximate quantile summary. We observe that at most one bucket at each level of the tree can overlap the leaf bucket containing v. The height of the tree (excluding the leaves) is $\log M$, and by Proposition 8, each non-leaf bucket contains at most $\frac{\epsilon n}{\log M}$ observations, so the cumulative gap between minimum and maximum rank is at most ϵn. The biggest gap between the minimum rank of a stored value v in Q and the maximum rank of its successor, v', is therefore $2\epsilon n$, and by Proposition 1 Q is an ϵ-approximate quantile summary.

6 Concluding Remarks

We presented here a broad range of algorithmic ideas for computing quantile summaries of data streams using small space. We highlighted connections among these ideas, and how techniques developed for one setting sometimes naturally lend themselves to a seemingly different setting. While the past decade has seen significant advances in space-efficient computation of quantile summaries, some fundamental questions remain unresolved. For instance, in the cash-register model, it is not known if the space bound of $O((\log(\epsilon n)/\epsilon)$ achieved by the GK algorithm [10] on a stream of length n is the best possible for any deterministic algorithm. When the elements are known to be in the range of $[1..M]$ for some positive integer M, is the $O((\log(M)/\epsilon)$ bound achieved by the q-digest algorithm [18] optimal? In either setting, only a trivial lower bound of $\Omega(1/\epsilon)$ on space is known. Similarly, when randomization is allowed, what is the best possible dependence of space needed on ϵ and δ? It appears that progress on these questions would require significant new ideas that may help advance our understanding of space-bounded computation as a whole.

References

1. Khaled Alsabti, Sanjay Ranka, and Vineet Singh. A one-pass algorithm for accurately estimating quantiles for disk-resident data. In Matthias Jarke et al., editors, *Proceedings of the Twenty-third International Conference on Very Large Data Bases*, pages 346–355, Los Altos, CA 94022, USA, August 26–29 1997. Morgan Kaufmann Publishers. Athens, Greece.

2. Arvind Arasu and Gurmeet Singh Manku. Approximate counts and quantiles over sliding windows. In *Proceedings of the 23rd ACM Symposium on Principles of Database Systems (PODS 2004)*, pages 286–296, June 14–16 2004. Paris, France.
3. J. Lawrence Carter and Mark N. Wegman. Universal classes of hash functions. In Matthias Jarke et al., editors, *Proceedings of the Ninth Annual ACM Symposium on Theory of Computing*, pages 106–112, Los Altos, CA 94022, USA, 1977. ACM. Colorado, USA.
4. Graham Cormode and S. Muthukrishnan. An improved data stream summary: the Count-Min sketch and its applications. In *Proceedings of Latin American Theoretical Informatics (LATIN '04)*, 2004.
5. Phillip B. Gibbons, Yossi Matias, and Viswanath Poosala. Fast incremental maintenance of approximate histograms. In Matthias Jarke, Michael J. Carey, Klaus R. Dittrich, Frederick H. Lochovsky, Pericles Loucopoulos, and Manfred A. Jeusfeld, editors, *Proc. 23rd Int. Conf. Very Large Data Bases, VLDB*, pages 466–475. Morgan Kaufmann, August 25–27 1997.
6. Phillip B. Gibbons, Yossi Matias, and Viswanath Poosala. Fast incremental maintenance of approximate histograms. *ACM Transactions on Database Systems*, 27(3):261–298, September 2003.
7. Anna C. Gilbert, Yannis Kotidis, S. Muthukrishnan, and Martin J. Strauss. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. In Peter M. G. Apers, Paolo Atzeni, Stefano Ceri, Stefano Paraboschi, Kotagiri Ramamohanarao, and Richard T. Snodgrass, editors, *Proceedings of the 27th Intl. Conf. Very Large Data Bases, VLDB*, pages 79–88, September 11-14 2001. Rome, Italy.
8. Anna C. Gilbert, Yannis Kotidis, S. Muthukrishnan, and Martin J. Strauss. How to summarize the universe: Dynamic maintenance of quantiles. In *Proceedings of the 28th Intl. Conf. Very Large Data Bases, VLDB*, pages 454–465, August 2002. Hong Kong, China.
9. Anna C. Gilbert, Yannis Kotidis, S. Muthukrishnan, and Martin J. Strauss. Domain-driven data synopses for dynamic quantiles. *IEEE Transactions on Knowledge and Data Engineering*, 17(7):927–938, July 2005.
10. Michael Greenwald and Sanjeev Khanna. Space-efficient online computation of quantile summaries. In *Proceedings of the 2001 ACM SIGMOD Intl. Conference on Management of Data*, pages 58–66, May 2001.
11. Michael B. Greenwald and Sanjeev Khanna. Power-conserving computation of order-statistics over sensor networks. In *Proceedings of the 23rd ACM Symposium on Principles of Database Systems (PODS 2004)*, pages 275–285, June 14–16 2004. Paris, France.
12. S. Guha and A. McGregor. Lower Bounds for Quantile Estimation in Random-Order and Multi-Pass Streaming. *International Colloquium on Automata, Languages and Programming*, 2007.
13. Xuemin Lin, Hongjun Lu, Jian Xu, and Jeffrey Xu Yu. Continuously maintaining quantile summaries of the most recent n elements over a data stream. In *Proceedings of the 20th International Conference on Data Engineering (ICDE04)*, pages 362–374. IEEE Computer Society, March 30 – April 2 2004. Boston, MA.
14. Gurmeet Singh Manku, Sridhar Rajagopalan, and Bruce G. Lindsay. Approximate medians and other quantiles in one pass and with limited memory. *SIGMOD Record (ACM Special Interest Group on Management of Data) SIGMOD '98*, 27(2):426–435, June 1998. Seattle, WA.

15. Gurmeet Singh Manku, Sridhar Rajagopalan, and Bruce G. Lindsay. Random sampling techniques for space efficient online computation of order statistics of large datasets. *SIGMOD Record (ACM Special Interest Group on Management of Data), SIGMOD '99*, 28(2):251–262, June 1999. Philadelphia, PA.

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

17. S. Muthukrishnan. Data streams: Algorithms and applications, 2003. Unpublished report, (invited talk at SODA03), available at `http://athos.ruthers.edu/~muthu/stream-1-1.ps`.

18. Nisheeth Shrivastava, Chiranjeeb Buragohain, Divy Agrawal, and Subhash Suri. Medians and beyond: New aggregation techniques for sensor networks. In *Proceedings of the 2nd ACM Conference on Embedded Network Sensor Systems (SenSys '04)*, pages 239–249, Nov 3-5 2004. Baltimore, MD.