Quantiles and Equidepth Histograms over Streams

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

[1] Bell Labs, Lucent Technologies, Rm. 2C-424, 600-700 Mountain Avenue P.O. Box 636, Murray Hill, NJ 07974 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 left-most 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* case in the literature [8]. A more general setting is the *turnstile* case [16] that also allows for deletion of observations. 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], a *quantile summary* for a set S is 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 $\overline{\epsilon}$-approximate quantile summary Q for the multiset $S = S' \cup S''$ where $\overline{\epsilon} = \frac{n'\epsilon' + n''\epsilon''}{n' + n''} \leq \max\{\epsilon', \epsilon''\}$. 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 $\texttt{rmax}_Q(z_{i+1}) - \texttt{rmin}_Q(z_i) \leq 2\overline{\epsilon}(n' + n'')$. 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 $Q^{''}$.

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

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

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

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

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

Proof. For any pair of consecutive elements q_i, q_{i+1} in Q, $\texttt{rmax}_Q(q_{i+1}) - \texttt{rmin}_Q(q_i) \leq (\frac{1}{B} + 2\epsilon')|S|$. By Proposition 1, it follows that Q must be $(\epsilon' + 1/(2B))$-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 [13], 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 [15] 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 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 $\Omega(n^{1/p}(\log n)^{2-2/p})$ space. The proofs of both these results are elementary and we establish them both here. 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 pigeonhole principle, there exists an element z_j 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. *[15] Any p-pass algorithm to solve the selection problem on a stream of n elements requires $\Omega(n^{1/p})$ space.*

The Munro and Patterson algorithm that almost achieves this space bound 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 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 [13] noted that all one-pass approximate quantile algorithms prior to their work (most notably, [15, 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 [13] 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 $r_{min}()$ and $r_{max}()$ 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 `rmin` and `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 $\texttt{combine}(B_1, B_2, \ldots, B_\alpha)$, and then[3] $\texttt{prune}(B, k-1)$. The weight of this new buffer is $\sum_j w_j$.

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 $\leq 1/(2k-2) + \max_i\{\epsilon_i\}$.*

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

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, $B = \texttt{prune}(Q, k-1)$ 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.

Munro-Patterson Algorithm

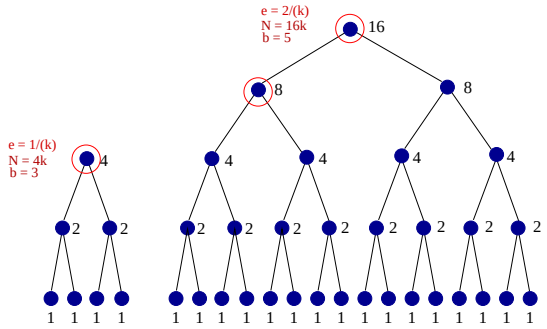

Fig. 1. Tree representations of Munro-Patterson algorithm for $b = 3$, and 5. Note that the final state of the algorithm under a root consists of the pair of buffers that are the children of the root. The shape and height of a tree depends only on b, and is independent of k, but the precision, ϵ, and the number of observations summarized, n, are both functions of k.

The Munro-Patterson algorithm [15] 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 $1, 2, 4, ...$ 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 $bk = O(\log^2(\epsilon n)/\epsilon)$.

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 $bk = O(\sqrt{n/\epsilon})$.

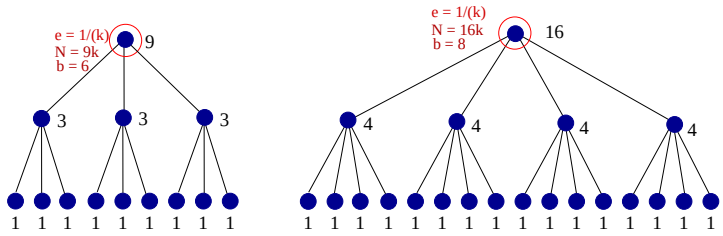

Fig. 2. Tree representation of Alsabti-Ranka-Singh algorithm. For any specific choice of b_1 and b_2, for $b_1 \neq b_2$, the tree for $b = b_1$ is not a subtree of $b = b_2$. The precision, ϵ, and the number of observations summarized, n, are both functions of k.

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, 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 [13] 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.

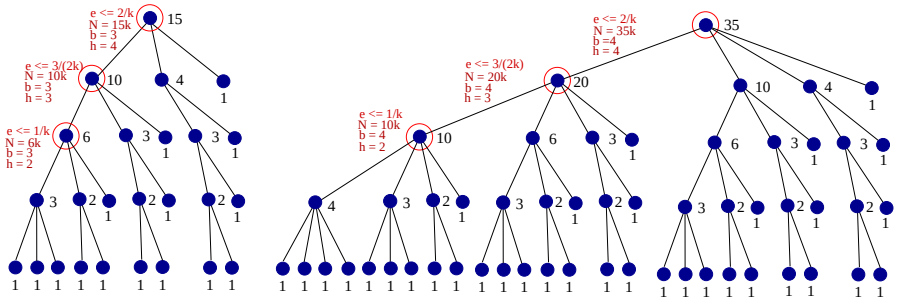

Fig. 3. Tree representation of Manku-Rajagopalan-Lindsay algorithm.

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* $\binom{b+h-2}{h-1}$.

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

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

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

$L(b,h)$ 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 $L(b,h-1) = \binom{(b-1)+(h-2)}{h-2}$. Therefore $L(b,h) = \sum_{i=1}^{b} \binom{(i-1)+(h-2)}{h-2}$, or $L(b,h) = \sum_{i=0}^{b-1} \binom{(i+(h-2)}{h-2}$. But by summation on the upper index, we have $L(b,h) = \sum_{i=0}^{b-1} \binom{(i+(h-2)}{h-2} = \binom{(b-1)+1+h-2}{h-1} = \binom{(b-1)+(h-1)}{h-1}$.