Approximation and Streaming Algorithms for Histogram Construction Problems

SUDIPTO GUHA
University of Pennsylvania
NICK KOUDAS
University of Toronto
and
KYUSEOK SHIM
Seoul National University

Histograms and related synopsis structures are popular techniques for approximating data distributions. These have been successful in query optimization and a variety of applications, including approximate querying, similarity searching, and data mining, to name a few. Histograms were a few of the earliest synopsis structures proposed and continue to be used widely. The histogram construction problem is to construct the best histogram restricted to a space bound that reflects the data distribution most accurately under a given error measure.

The histograms are used as quick and easy estimates. Thus, a slight loss of accuracy, compared to the optimal histogram under the given error measure, can be offset by fast histogram construction algorithms. A natural question arises in this context: Can we find a fast near optimal approximation algorithm for the histogram construction problem? In this article, we give the first linear time $(1+\epsilon)$-factor approximation algorithms (for any $\epsilon > 0$) for a large number of histogram construction

problems including the use of piecewise small degree polynomials to approximate data, workloads, etc. Several of our algorithms extend to data streams.

Using synthetic and real-life data sets, we demonstrate that in many scenarios the approximate histograms are almost identical to optimal histograms in quality and are significantly faster to construct.

Categories and Subject Descriptors: F.2 [**Theory of Computation**]: Analysis of Algorithms; G.1.2 [**Numerical Analysis**]: Approximation; H.2 [**Information Systems**]: Database Management

General Terms: Algorithms, Performance, Theory

Additional Key Words and Phrases: Data Streams, histograms, approximation algorithm

1. INTRODUCTION

Obtaining fast and accurate synopsis of data distributions is a central problem in database query optimization. Given a query the optimizer tries to determine the cost of various alternative query plans based on estimates [Selinger et al. 1979]. Histograms were one of the early techniques proposed in this context to approximate the data distributions [Kooi 1980; Muralikrishna and DeWitt 1988]. More recently histograms have been used in a broad range of topics, for example, approximate query answering [Acharya et al. 1999], mining time series data [Keogh et al. 2002] and curve simplification [Bertolotto and Egenhofer 1999] among many others. There is a broad taxonomy of histograms which we will not be able to cover in this article; the interested readers should refer to Ioannidis [2003]. In this article, we will focus on serial histograms [Ioannidis 1993; Ioannidis and Poosala 1995] where disjoint intervals of the domain are grouped together and define a bucket. Each bucket is represented by a single value. Thus a histogram defines a piecewise constant approximation of the data. Given a query that asks the data value x_i at i, the value (say $\hat{x}_i$) corresponding to the bucket containing i is returned as an answer.

The objective of a histogram construction algorithm is to find a histogram with at most B buckets which minimizes a suitable function of the errors. One of the most common error measures used in histogram construction is $\sum_i (x_i - \hat{x}_i)^2$, which is also known as the V-Optimal measure. This was first introduced by Poosala et al. [1996]. However, since then several proposals have been introduced optimizing different error measures, for example, range queries [Koudas et al. 2000; Gilbert et al. 2001; Guha et al. 2002b], relative error [Garofalakis and Gibbons 2002; Guha et al. 2004], to name a few. Each one of these has unique benefits relevant to its domain of application. However, the V-Optimal measure continues to be widely popular. This measure is also important mathematically since it is the square of the ℓ_2-distance between the original distribution and the distribution specified by the synopsis. Also this error measure has been frequently used to optimize other synopses such as wavelets and discrete Fourier transforms. In this article, we use the popular V-Optimal measure as a running example to illustrate our ideas. However, the discussion applies to a broad array of error measures and we indicate a few examples of such generalizations.

As mentioned, histograms are not the only synopsis structures used. Wavelets [Matias et al. 1998; Gilbert et al. 2001; Garofalakis and Gibbons 2002; Guha and Harb 2006] and quantile summaries [Manku et al. 1998; Greenwald

and Khanna 2001] have been used widely as well. There is a broad literature available on these topics—the references in the above papers contain appropriate pointers, and we omit further discussion.

In an early paper, Jagadish et al. [1998] gave an $O(n^2 B)$ algorithm for constructing the best V-Optimal histogram. This algorithm is based on dynamic programming which generalizes nicely to a wide variety of error measures as well. The quadratic running time is undesirable for large datasets. Jagadish et al. [1998] also gave an approximation algorithm that runs in time $O(n^2 B/\ell)$ and uses $(B + \ell)$ buckets while guaranteeing the quality of the $(B+\ell)$-bucket histogram constructed is no worse than that of the best B-bucket histogram. However, even for extra $O(B)$ buckets, the running time remains quadratic.

Recall that the histograms (or all synopsis structures) provide "rough" estimates for the cost of operators. A natural question arises in this context: "since the end use of a histogram is to approximate a data distribution, can we find a linear-time near-optimal approximation to the best histogram?" In this article, we provide such algorithms. The algorithms allow a graceful trade-off between the (guaranteed) quality of the histogram and the construction time. More specifically, we provide several approximation schemes where given a precision parameter $\epsilon > 0$, the algorithms return solutions which are at most $(1+\epsilon)$ times worse than the optimal solution. (See Hochbaum [1996] and Vazirani [2001] for discussion on approximation schemes.) We note that the discussion of the "best" error measure is orthogonal to the goal of this article.

Our contribution

—We give the first one pass linear time $(1+\epsilon)$-approximation algorithm for the histogram construction problem for a wide variety of error measures. In the context of V-Optimal histograms, our results provide the best-known bounds for approximation algorithms. Our article subsumes (as well as improves) our previous paper Guha et al. [2001] and improves upon Guha et al. [2004] (which discusses relative error). We also improve the results in and Guha and Koudas [2002] and Guha et al. [2002a] although some of the issues raised in those papers are incomparable to the question of the "best (approximate) histogram construction algorithm". We provide a table of the published results in Table I. The time and space complexities in the table are reported for V-optimal histograms. We also indicate if the algorithm generalizes to the broad class of error measures we consider here. The table is explained in more detail in Section 1.1.

—Our algorithms extend to data streams. The streaming model we consider in this article assumes that the data items x_i are presented one at a time in an increasing order of i. Thus, for time series and analogous applications, these algorithms are one-pass stream algorithms. The "Y/N" in Table I indicates that the result can be applied to sliding window data streams because of its (linear) space complexity. However, such an algorithm is really an offline algorithm.

—We provide a general framework that extends to a broad class of error measures, including those considered by Jagadish et al. [1998], for example,

Table I. Summary of Results on Similar Problems

Article	Error	Stream	Factor	TIME (V-OPT)	Space
[Jagadish et al. 1998]	g	No	Opt	$O(n^2 B)$	$O(nB)$
[Guha 2005]	g	Y/N	Opt	$O(n^2 B)$	$O(n)$
[Guha et al. 2001]	g	Yes	$(1 + \epsilon)$	$O(nB\tau)$	$O(B\tau)$
[Guha and Koudas 2002]	g	Y/N	$(1 + \epsilon)$	$O(n + B\tau^2 \log n)$	$O(n + B\tau)$
[Gilbert et al. 2002]	V,ℓ_1	Yes	$(1 + \epsilon)$	$n\tau^{O(1)}$	$\tau^{O(1)}$
[Guha et al. 2002a]	V	Yes	$(1 + \epsilon)$	$O(n + B\tau^2 \frac{1}{\epsilon^2} \log^2 \frac{1}{\epsilon})$	$O(\frac{B\tau}{\epsilon} \log \frac{1}{\epsilon})$
[Guha et al. 2004]	g	Yes	$(1 + \epsilon)$	$O(n + B\tau^2 \log \tau)$	$O(B\tau^2 \log n)$
This article	g	Y/N	$(1 + \epsilon)$	$O(n + B^3 \log^2 n + \tau B^2/\epsilon)$	$O(n + B^2/\epsilon)$
	g	Yes		$O(n + M\tau)$	$O(B\tau + M)$

The table follows a decreasing order of running time (for the V-Optimal measure) with the exception of the result of Gilbert et al. 2002, which applies to a more general streaming algorithm, see text for more details. The symbol 'g' stands for a general class of error measures, 'V' denotes the algorithm is applicable to the V-Optimal error measure only. $\tau = \min\{B\epsilon^{-1} \log n, n\}$ usually $\tau \ll n$ and $M = B(B\epsilon^{-1} \log \tau + \log n) \log \tau) \ll B\tau \log \tau < B\tau \log n$. The sequence of improvements in the above table shows how the effects of the factors B, ϵ^{-1} and $\log n$ on the running time can be separated. Each of B, ϵ^{-1} and $\log n$ can be easily be ~ 10 (B will likely be larger) and separating their dependence leads to significant speedup.

workloads. In particular, we summarize the main techniques in a theorem which can be used to design approximate histogram algorithms for alternate measures. We consider several examples, (i) approximation by piecewise linear segments (as well as degree-d polynomials) (ii) the χ^2-test error function (this was proposed, among others, by Donjerkovic et al. [1999] in defining dynamic compressed histograms), (iii) sum of absolute errors, proposed in Poosala et al. [1996] and Matias et al. [1998].

—Finally, we demonstrate the effectiveness of the approximation schemes using synthetic and real life data sets. Since the overall algorithmic technique is the same for different error measures, we only report on the performance of approximate V-Optimal histograms. The results confirm that the approximation algorithms are an attractive tool for constructing accurate histograms faster.

Organization. In Section 2, we present definitions and reviews of previous work which are necessary for the remainder of the article. In Section 3, we present our algorithms and analyses. In Section 4, we summarize the central properties of the approximation technique and demonstrate its use in the context of three new examples. In Section 5, we present the results of an experimental evaluation. Section 6 concludes the article.

1.1 Related Work and Discussion

Histograms are not the only synopsis structures used. See Gibbons and Matias [1999], Matias et al. [1998], Gilbert et al. [2001], and Garofalakis and Gibbons [2002] for excellent overviews of other synopsis techniques. For other histograms, for example, range queries, several surveys including a

retrospective exist [Ioannidis 2003; Muthukrishnan and Strauss 2004]. We focus on the results mentioned in Table I.

For frequency histogram construction where x_i is the frequency of item i, all the algorithms except that in Gilbert et al. [2002] require the frequency histogram to be computed as a preprocessing step. This requires at least one extra pass, and space versus pass tradeoff results exist. We omit the discussion in interest of space.

The main thrust of Guha and Koudas [2002] was fast construction of approximate histograms for sliding window streams and time series. The main issue that arises in such scenarios is that the computation for a time window on the first n items, over the interval $[1, n]$, may be useless for the next n items, that is, over the interval $[2, n+1]$. The main question we addressed in that article was: "can a data structure be maintained in the context of sliding window streams such that near optimal histogram representations can be computed on-demand efficiently ?" We showed that we can maintain a data structure that requires $O(1)$ update time and allows the construction of an approximate histogram (whenever required) in $O(B^3 \epsilon^{-2} \log^3 n)$ time. This avoided the $O(nB^2 \epsilon^{-1} \log n)$ time or more expensive algorithms that were previously known. This was a significant improvement (for reasonable B, ϵ) and quite useful if the histograms were not constructed too often. In hindsight, the same algorithm gives a better offline algorithm for the original histogram construction problem itself—the histogram is constructed only once! Sliding window stream problems, along with histogram construction, have been investigated further in Datar et al. [2002].

Notice that the algorithms in Guha et al. [2002a, 2004] and Section 3.5 in this article assume that we see all the data before we attempt to compute any histogram. This is different from the algorithm in Guha et al. [2001] (Section 3.2) where a B-bucket histogram is maintained at all times. Note that none of these are online algorithms since there is no notion of *irrevocable commitment*, see Borodin and El-Yaniv [1998].

The algorithm of Gilbert et al. [2002] applies to a more general model of streaming. It shows that collecting a number of suitable wavelet coefficients gives us a robust histogram—whose error does not decrease if we add a few extra buckets. It then uses the robust histogram to construct a histogram with B buckets. The algorithm uses sketches or distance preserving embeddings along the lines of Alon et al. [1999], Feigenbaum et al. [2002], and Indyk [2000] to collect the wavelet coefficients. The algorithm in Thaper et al. [2002] uses a variant of the algorithm in Gilbert et al. [2002] for multidimensional histogram synopsis. Guha et al. [2002a] show that the construction of robust histograms is significantly easier in a simpler model of streaming.

2. PRELIMINARIES

2.1 Problem Statement

Let $X = x_1, \ldots, x_n$ be a finite data sequence. The histogram construction problem is defined as follows: given some space constraint B, create and store a

compact representation H_B of the data sequence using at most B storage, such that H_B is optimal under some notion of error $E_X(H_B)$ defined between the data sequence and H_B. The typical histogram representation collapses the values x_i in a sequence of consecutive points i where $i \in [s_r, e_r]$ (i.e., $s_r \le i \le e_r$) into a single value h_r, thus forming a bucket b_r, that is, $b_r = (s_r, e_r, h_r)$. The histogram H_B is used to answer queries about the value at a point i where $1 \le i \le n$. For $s_r \le i \le e_r$, we estimate x_i by h_r. The histogram uses at most B buckets which cover the entire interval $[1, n]$, and saves space by storing only $O(B)$ numbers instead of n values. Since h_r is an estimate for the values in bucket b_r for the query at a point i for $s_r \le i \le e_r$, we incur an error $h_r - x_i$. The error $E_X(H_B)$ of the histogram H_B is defined as a function of these point errors.

Problem 1 (*Optimal Histograms*). Given a sequence X of length n, a number of buckets B, find H_B to minimize $E_X(H_B)$ under the given error function E.

Jagadish et al. [1998] gave a general technique to compute the optimum histogram in $O(n^2 B)$ time and $O(Bn)$ space for several measures. However, the resulting histogram is used to *approximate* the data, and it is natural question to ask: why should we spend quadratic time to construct such an approximation? Can we construct a histogram which is nearly optimal in time linear in the size of input data? Can we also make the approximation "tunable", that is, allow faster running times if a less accurate histogram suffices for the application at hand? These lead to the following problem formulation:

Problem 2 (($1 + \epsilon$)-*approximate Histograms*). Given a sequence X of length n, a number of buckets B, and a precision parameter $\epsilon > 0$, find H_B with $E_X(H_B)$ at most $(1 + \epsilon) \min_H E_X(H)$ where the minimization is taken over all histograms H with B buckets.

Note. We assume that the input values are integers in the range $[-R, R]$. All algorithms and techniques in this paper extend to reals by simple scaling provided the minimum non-zero error of a bucket can be bounded. This is true for any input with bounded precision. Typically the histograms are used to describe frequency counts and the frequencies are integers at most the size of the data. We use this fact and assume $\log R = O(\log n)$.

We will present an $O(n)$ time algorithm for the above approximate histogram construction problem, which applies to a wide variety of error measures. This is the best possible result since we have to look at every data point in the input, requiring $\Omega(n)$ time for any algorithm. As mentioned earlier, we will focus on the V-Optimal measure as a running example of the technique.

2.2 The V-Optimal Measure

The most common definition of total error is the sum of squares of the errors at every point i. The resulting optimal histogram is the well-known *V-Optimal* histogram [Poosala et al. 1996].

Since the intervals corresponding to the buckets do not overlap and every point belongs to exactly one bucket, we can express the total error as a sum of

bucket errors. The total error for a histogram H with buckets $b_1, \ldots, b_B$ is the sum over all bucket errors $\sum_r \text{SQERROR}(b_r)$. The error SQERROR for the bucket b_r defined by the interval $[s_r, e_r]$ and representative h_r is:

$$\text{SQERROR}(b_r) = \sum_{i=s_r}^{e_r} (x_i - h_r)^2. \tag{1}$$

The above error is minimized when $h_r = \frac{1}{e_r - s_r + 1} \sum_{i=s_r}^{e_r} x_i$ (i.e., the mean of the values x_i for i in the bucket). After an easy simplification, we have:

$$\text{SQERROR}(s_r, e_r) = \sum_{\ell=s_r}^{e_r} x_\ell^2 - \frac{1}{e_r - s_r + 1} \left(\sum_{\ell=s_r}^{e_r} x_\ell \right)^2. \tag{2}$$

2.3 V-Optimal Histogram Construction

We now review the V-optimal histogram construction algorithm of Jagadish et al. [1998]. In this problem, given a sequence of n numbers $x_1, \ldots, x_n$, we seek to partition the index set $\{1..n\}$ into B intervals (or buckets) minimizing the sum of the squared errors in approximating each data point j for $1 \leq j \leq n$. From the previous section, we know that the data points within each bucket are represented by their mean value, and the total error is the sum of the errors of each bucket. A basic observation is that if the last bucket contains the data points indexed by $[i+1, n]$ in the optimal histogram, then the rest of the buckets must form an optimal histogram with $B-1$ buckets for $[1, i]$. If this condition is not true, then the cost of the solution can be decreased by taking the optimal histogram with $(B-1)$ buckets for $[1, i]$ and the last bucket defined on the points in $[i+1, n]$, which contradicts the optimality of the original solution. Thus, if we have found the best possible $(B-1)$ bucket histogram approximating $[1, i]$ for all i; we try all of the $n-1$ values of i to find the best i and compute the best B bucket approximation for $[1, n]$. Before we proceed further, we need the following definition:

Definition 1. Let $\text{TERR}[i, k]$ be the best (minimum) error achieved by a k-bucket histogram representing the interval $[1, i]$. Note that the optimum histogram construction problem is to find a histogram with error $\text{TERR}[n, B]$.

A dynamic programming algorithm follows from the above which is presented in Figure 1. To compute the error of the bucket $[i + 1, \ldots, j]$ that is given by Eq. (2) we maintain two arrays SUM and SQSUM, such that,

$$\text{SUM}[1, i] = \sum_{\ell=1}^{i} x_\ell \quad \text{SQSUM}[1, i] = \sum_{\ell=1}^{i} x_\ell^2.$$

We can now compute the error of the bucket $\text{SQERROR}(i, j)$ in $O(1)$ time since the partial sums in Eq. (2) reduce to

$$\sum_{\ell=i+1}^{j} x_\ell = \text{SUM}[1, j] - \text{SUM}[1, i-1] \quad \sum_{\ell=i+1}^{j} x_\ell^2 = \text{SQSUM}[1, j] - \text{SQSUM}[1, i-1].$$

```
Procedure V-OPT()
begin
1.      Sum[1, 1] := v₁
2.      Sqsum[1, 1] := v₁²
3.      for i := 2 to n do {
4.          Sum[1, i] := Sum[1, i − 1] + vᵢ
5.          Sqsum[1, i] := Sqsum[1, i − 1] + vᵢ²
6.      }
7.      for j := 1 to n do {
8.          Terr[j, k] := ∞
9.          for k := 2 to B do
10.             for i := 1 to j-1 do
11.                 Terr[j, k] := min (Terr[j, k], Terr[i, k − 1] + Sqerror(i + 1, j))
12.     }
end
```

Fig. 1.   V-Optimal histogram algorithm.

Computing each entry of $\text{Terr}[j,k]$ requires $O(n)$ time. The algorithm runs in $O(n^2 B)$ time because we have to compute $O(nB)$ entries of $\text{Terr}[j, k]$.

3. APPROXIMATION ALGORITHMS

Histograms are typically used to approximate a distribution. As mentioned earlier, because the end use of a histogram is to approximate data, it may be desirable to construct an almost optimal histogram faster than the quadratic (in n) algorithm. However the notion of "almost optimal" has to be precise for the histogram not to lose its descriptive power. We use the notion of *approximation schemes*, that is, given a precision parameter $\epsilon > 0$, the approximation algorithm will return a histogram whose error is $(1 + \epsilon)$ times the error of the optimal histogram. Thus, if we desire a 1% approximation to the optimal histogram, we would set $\epsilon = 0.01$. The running time of the approximation algorithm will dependent on ϵ^{-1} along with other parameters of the problem (namely B, n). Thus, approximation schemes allow us to have a graceful tradeoff between the accuracy of the solution and the construction time. We provide fully polynomial approximation schemes.[1] There is a large literature concerning the notions of approximation [Hochbaum 1996; Vazirani 2001] and we omit further discussion. From a practical consideration, a worst case $(1 + \epsilon)$ approximation guarantee gives us a more organized starting point to develop a heuristic. We first provide a simple $(1 + \epsilon)$ approximation algorithm in Section 3.2. This algorithm runs in $O(nB^2\epsilon^{-1} \log n)$ time and $O(B^2\epsilon^{-1} \log n)$ space. Subsequently we improve the algorithm to run in linear time. But before proceeding to the algorithms, let us focus on why we can expect subquadratic approximation algorithms.

3.1 Intuition and Challenges

Consider the *VOPT* algorithm in Figure 1. We begin with the following:

OBSERVATION 1. $\text{Sqerror}(i + 1, j)$ *and* $\text{Terr}[i, k − 1]$ *are non-increasing and non-decreasing (both are non-negative) functions respectively over the values of* i.

---

[1]The running time is a polynomial in n, $\frac{1}{\epsilon}$, B as opposed to $n^{\frac{1}{\epsilon}}$.
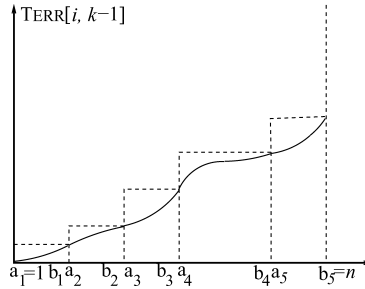
Fig. 2.   Approximating $\textsc{Terr}[i, k - 1]$ by a histogram.

The simplest proof of the above is to observe that, as i increases, the solution that approximated the interval $[1, i + 1]$ using k buckets remains a valid approximation of $[1, i]$ using k buckets.[2] Further that same solution has error at most $\textsc{Terr}[i + 1, k]$ on $[1, i]$ because if we discount the contribution of $(i + 1)$ the error cannot increase. But the best possible solution for the subproblem of approximating $[1, i]$ with k buckets has error $\textsc{Terr}[i, k]$ and hence $\textsc{Terr}[i, k] \leq \textsc{Terr}[i + 1, k]$. The other property can be proved analogously.

In light of the above, a natural question arises: "because we are searching for the minimum of the sum of two functions, both nonnegative, one of which is non-increasing and the other non-decreasing, can we use a more effective search strategy instead of the for loop in lines (10)–(11) of Figure 1?" The answer, unfortunately, is no. Consider a set of nonnegative values $v_1, \ldots, v_n$; let $f(i) = \sum_{r=1}^{i} v_r$, and $g(i) = f(n) - f(i - 1)$. The function $f(i)$ and $g(i)$ are monotonically increasing and decreasing respectively. But finding the minimum of the sum of these two functions amounts to minimizing $f(n) + v_i$, or in other words minimizing v_i, which has a $\Omega(n)$ lower bound. Note that this does not rule out that over B levels, the cost of the searching can be amortized—but no such analysis exists to date. The interesting aspect of the example is that picking any i gives us a 2 approximation (since $f(n) + v_i \leq 2f(n)$ and the minimum is no smaller than $f(n)$). In essence, the searching can be reduced if we are willing to settle for approximation.

The central idea is that instead of storing the entire function $\textsc{Terr}[i, k - 1]$, we *approximate the function* by a staircase or a histogram as shown in Figure 2. The interval $[1, n]$ is broken down into τ intervals (a_i, b_i) to approximately represent the function with a histogram. We have $a_1 = 1$, $a_{i+1} = b_i + 1$, and $b_\tau = n$. Furthermore, the intervals are created such that the value of the function at the right hand boundary of an interval is at most a factor $(1 + \delta)$ times the value of the function at the left hand boundary.[3] The parameter δ will be fixed

---

[2]Note that we are not restricted to storing the mean of the values in a bucket as a representative. But storing the mean arises as a *natural* consequence of the optimization, any other choice is suboptimal, which is precisely the point we are making.

[3]For readers familiar with histogram construction literature, we are building EquiWidth histograms, but in the *exponent*. That is, the buckets correspond to equally spaced exponent values of $(1 + \delta)$. It is feasible to use actual EquiWidth histograms if we have an idea of the final error, and we use them in the subsequent algorithm AHIST-L-Δ. We use a combination of both EquiWidth and EquiWidth-in-exponent in algorithm AHIST-B.

Procedure AHIST-S()
begin
1.   Set up $(B - 1)$ lists $Q[k]$ to store the intervals
2.   SUM := SQSUM := 0
3.   **for** j := 1 **to** n **do** {
4.   SUM := SUM + v_j
5.   SQSUM := SQSUM + v_j^2
6.   **for** k := 2 **to** B **do** {
7.   APXERR$[j, k] = \infty$
8.   **for** i := each end point b of interval list for $(k - 1)$-th list $Q[k - 1]$ **do**
9.   // Recall APXERR$[j,1]$ = SQERROR$(1, j)$
10.   APXERR$[j, k]$ = min (APXERR$[j, k]$, APXERR$[i, k - 1]$ + SQERROR$(i + 1, j)$)
11.   // a_ℓ is the start index of the last interval in $Q[k]$
12.   // b_ℓ is the end index of the last interval in $Q[k]$
13.   **if** $(k \le B - 1$ **and** APXERR$[j,k] > (1+\delta)$APXERR$[a_\ell,k])$ {
14.   // Now, we have APXERR$[j, k]$, SUM = SUM$[j]$ and SQSUM = SQSUM$[j]$
15. $a_{\ell+1} := b_{\ell+1} := j$
16.   Insert a new interval $[a_{\ell+1}, b_{\ell+1}$,APXERR$[j,k]$,SUM,SQSUM$]$ to $Q[k]$
17.   }
18.   **else**
19.   Set b_ℓ to j
20.   }
21.   }
end

Fig. 3.   The algorithm AHIST-S.

to be $\frac{\epsilon}{2B}$ with $\epsilon < 1$. Such a partition always exists, the challenge is to construct it quickly. We can view the *VOPT* algorithm presented in Figure 1 as using n buckets to represent the non-decreasing error function TERR$[i, k - 1]$ exactly. But we need much fewer buckets if we approximate the function.

However there is a caveat—*we cannot simultaneously approximate* TERR$[i, k]$ *and assume that we know* TERR$[j, k-1]$ *exactly for all* $j < i, k > 2$. The solution is to find a sequence of intervals such that the *approximation* APXERR$[i, k]$ (of TERR$[i, k]$) increased by a $1 + \delta$ factor. If we can show that APXERR$[i, k]$ is close to TERR$[i, k]$ for all i, k (inductively)—then we can claim an approximation. This is the algorithm we present next.

3.2 The AHIST-S: An Approximate Algorithm with Small Space

The AHIST-S algorithm presented in Figure 3 incorporates the idea of approximating the error function. We maintain $(B - 1)$ interval lists implemented through arrays of bounded size, since we have a bound on the sizes of the lists. Each element of the k-th list will store the index number y, SUM$[1, y]$, SQSUM$[1, y]$ and APXERR$[y, k]$ values. We maintain APXERR$[y, 1]$ = TERR$[y, 1]$ = SQERROR$(1, y)$, that is, for representation by one bucket we would compute the error exactly. This will be the base case of the inductive proof.

Example 1. Consider the sequence of numbers $\{1, 2, 3, 4, 5, 6, 7, 8,$ $9, 10, 11, 12, 13, 14, 15, 16, 19\}$ and assume $B = 2$. The TERR$[i, 1]$ values form a nice quadratic increasing function of i for $(1 \le i \le 16)$. Assume that $\delta = 0.99$. The possible values of the endpoint of the first bucket is $[1, 16]$. The algorithm breaks the domain into intervals $[1], [2], [3], [4], [5, 6],$ $[7, 8], [9 \ldots 11], [12 \ldots 15]$ and $[16]$. The values stored in the list are the end points $\{1, 2, 3, 4, 6, 8, 11, 15, 16\}$. To reiterate, the value corresponding
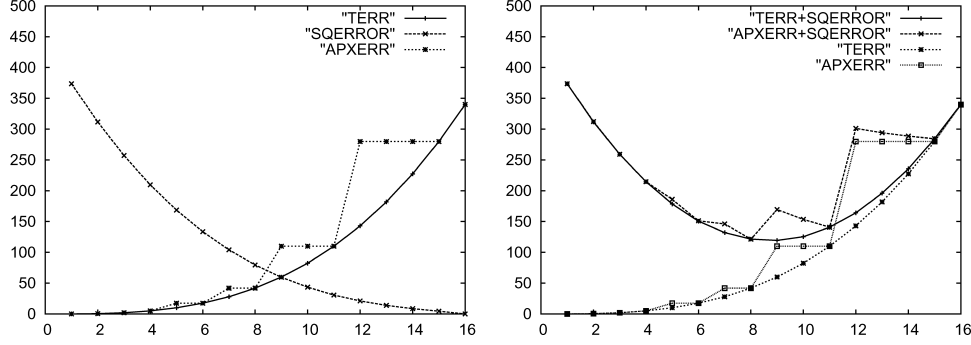
Fig. 4.   The case $B = 2$ and $\delta = 0.99$ for the sequence $\{1, 2, 3, \cdots, 16, 19\}$.

to entry 9 or 10 is not stored because those values are *approximated* by the value corresponding to the entry 11. The comparison of $\text{TERR}[i, 1]$, $\text{APXERR}[i, 1]$ and $\text{SQERROR}(i + 1, 17)$ is presented in Figure 4. The figure also shows the result of the true sum $\text{TERR}[i - 1, 1] + \text{SQERROR}(i, 17)$ and the approximate sum $\text{APXERR}[i - 1, 1] + \text{SQERROR}(i, 17)$. The true minimum is 119.5 at entry $i = 9$ with $[1, 9], [10, 17]$ as the best buckets. We evaluate the sum $\text{APXERR}[i - 1, 1] + \text{SQERROR}[i, 17]$ at $i \in \{1, 2, 3, 4, 6, 8, 11, 15, 16\}$ and get the minimum at $i = 8$ which is 121.556. Thus we get an approximate solution $\{[1, 8], [9, 17]\}$. Notice that the approximate sum is only close to the true sum (for B = 2 it is exactly the same) at the endpoints.

LEMMA 1.   *AHIST-S computes an $(1 + \epsilon)$-approximate B-bucket histogram.*

PROOF.   We will prove by induction that $\text{APXERR}[j, k] \leq (1 + \delta)^{k-1} \text{TERR}[j, k]$. The base case is $k = 1$. In this case we choose the mean of the values in a bucket as the representative, and so does the best histogram with one bucket. Therefore, $\text{APXERR}[j, 1] = \text{TERR}[j, 1]$.

Assume that we have proved the statement for all $k' < k$ and are considering $\text{APXERR}[j, k]$. Suppose that the last bucket chosen in the best approximation of the interval $[1, j]$ with k buckets is $[j', j]$. Let the interval stored in the $(k - 1)$th list which contains j' be $[a_\ell, b_\ell]$ where $a_\ell \leq j' \leq b_\ell < j$. We know that

$$
\begin{aligned}
\text{TERR}[j, k] &= \text{TERR}[j', k - 1] + \text{SQERROR}(j' + 1, j) \\
&\geq \text{TERR}[a_\ell, k - 1] + \text{SQERROR}(j' + 1, j) \; (\text{TERR}[\cdot, k - 1] \text{ is monotone}) \\
&\geq \text{TERR}[a_\ell, k - 1] + \text{SQERROR}(b_\ell + 1, j) \\
&\quad (j' \leq b_\ell < j \text{ and SQERROR is non-increasing over subintervals}) \\
&\geq \left(\frac{1}{(1 + \delta)^{k-2}} \text{APXERR}[a_\ell, k - 1] + \text{SQERROR}(b_\ell + 1, j) \right) \quad (3) \\
&\quad (\text{By Induction Hypothesis.}) \\
&\geq \frac{1}{(1 + \delta)^{k-2}} \left(\frac{1}{1 + \delta} \text{APXERR}[b_\ell, k - 1] + \text{SQERROR}(b_\ell + 1, j) \right) \quad (4)
\end{aligned}
$$

The last equation follows from the fact that both $\textsc{Apxerr}[a_\ell, k-1]$ and $\textsc{Apxerr}[b_\ell, k-1]$ were computed and, by construction, $\textsc{Apxerr}[a_\ell, k-1] \geq \textsc{Apxerr}[b_\ell, k-1]/(1+\delta)$. Therefore, we have

$$
\begin{aligned}
\textsc{Terr}[j, k] &\geq \frac{1}{(1+\delta)^{k-2}} \left(\frac{1}{1+\delta} \textsc{Apxerr}[b_\ell, k-1] + \textsc{Sqerror}(b_\ell+1, j) \right) \\
&\geq \frac{1}{(1+\delta)^{k-1}} (\textsc{Apxerr}[b_\ell, k-1] + \textsc{Sqerror}(b_\ell+1, j)) \\
&\qquad \text{(All quantities are non-negative.)} \\
&= \frac{1}{(1+\delta)^{k-1}} \textsc{Apxerr}[j, k].
\end{aligned}
$$

The last step follows from the fact that we minimized over b_ℓ's to compute the value of $\textsc{Apxerr}[j, k]$. This proves the inductive step. Setting $j = n$ and $k = B$, we get that $\textsc{Apxerr}[n, B]$, which is the cost of our solution, is at most $(1+\delta)^{B-1}$ times the true minimum $\textsc{Terr}[n, B]$. If $\delta = \epsilon/(2B)$, the approximation factor is $(1 + \frac{\epsilon}{2B})^{B-1}$, which is at most $(1+\epsilon)$ for small ϵ (say $\epsilon \leq 1$). This proves the lemma. $\square$

LEMMA 2. *Let $\tau = \min\{B\epsilon^{-1} \log n, n\}$. The size of a list is $O(\tau)$.*

PROOF. Consider a list of size $\ell + 1$ which corresponds to the sequence of intervals $[a_1, b_1], \ldots, [a_\ell, b_\ell], [a_{\ell+1}, b_{\ell+1}]$. We know that $a_1 = 1$ and for all u, $b_u + 1 = a_{u+1}$. From the algorithm we have

$$
\begin{aligned}
\textsc{Apxerr}[b_1+1, k-1] = \textsc{Apxerr}[a_2, k-1] &> (1+\delta)\textsc{Apxerr}[a_1, k-1] \\
\textsc{Apxerr}[b_2+1, k-1] = \textsc{Apxerr}[a_3, k-1] &> (1+\delta)\textsc{Apxerr}[a_2, k-1] \\
\vdots \quad &> \quad \vdots \\
\textsc{Apxerr}[b_\ell+1, k-1] = \textsc{Apxerr}[a_{\ell+1}, k-1] &> (1+\delta)\textsc{Apxerr}[a_\ell, k-1].
\end{aligned}
$$

$$
\textsc{Apxerr}[a_{\ell+1}, k-1] > (1+\delta)^{\ell-1} \textsc{Apxerr}[a_2, k-1]
$$

Notice that $\textsc{Apxerr}[a_1, k-1]$ can be zero. But $\textsc{Apxerr}[a_2, k-1]$ cannot be zero, since the first inequality cannot be satisfied in that case. In that case, $\textsc{Terr}[a_2, k-1] \geq \frac{1}{1+\epsilon} \textsc{Apxerr}[a_2, k-1]$ by Lemma 1, and is therefore non-zero.

Assuming that the input is polynomially bounded integers, the minimum possible nonzero error in a single bucket occurs in the following setting: the values in a bucket are the same except exactly one value which differs by 1 from the rest. Without loss of generality,[4] if the bucket contained s values, we can assume $s - 1$ values are 0 and one value is 1. In this case, the error is $(1 - \frac{1}{s})^2 + (s-1)\frac{1}{s^2}$, which simplifies to $1 - \frac{1}{s}$. Now $s \geq 2$ (otherwise, all values in the bucket are trivially the same!) and therefore the minimum possible nonzero error $\textsc{Terr}[a_2, k-1]$ is $\frac{1}{2}$. Thus, the minimum non-zero $\textsc{Apxerr}[a_2, k-1]$ is also $\frac{1}{2}$ since $\textsc{Apxerr}[a_2, k-1] \geq \textsc{Terr}[a_2, k-1]$.

---

[4]Adding a fixed value C to all the values within a bucket does not change the error.

Procedure FixedWindowHistogram()
begin
1. Compute SUM and SQSUM
2. For k=1 to B-1 {
3.   Initialize k'th queue to empty
4.   CreateQueue[1,n,k]
5. }
6. For i:= end point b_ℓ of $B - 1^{th}$ queue {
7.   APXERR$[n, B]$ = min(APXERR$[n, B]$,
8.   APXERR$[i, B - 1]$ + SQERROR$[i + 1, n]$)
9. }
end

Procedure CreateQueue[a,b,k]
begin
1. If $(a > b)$ return
2. Otherwise If $(a == b)$
3.   insert a at the end of k'th queue
4. Otherwise {
5.   Compute $t =$ APXERR$[a, k]$
6.   /* If $k == 1$ then these are simply
7.   SQERROR$[1, a]$ and SQERROR$[1, c]$
8.   for $k > 1$ minimize over endpoints i
9.   of $k - 1$'th queue */
10.   Perform a binary search to find c such
11.   that APXERR$[c, k] \leq (1 + \delta)t$ and either
12.   APXERR$[c + 1, k] > (1 + delta)t$ or $c == b$
13.   Insert c at the end of k'th queue
14.   CreateQueue[c+1,b,k]
15. }
end

Fig. 5. Algorithm FixedWindowHistogram.

The maximum possible value of APXERR is SQSUM$[1, n]$ which is at most nR^2 where R is the largest value seen. Suppose, for contradiction, that we have $\ell > 1 + 2\delta^{-1} \ln(2nR^2)$. From the above,

$$nR^2 \geq \text{TERR}[a_{\ell+1}, k - 1] > (1 + \delta)^{\ell-1}\text{TERR}[a_2, k - 1] \geq (1 + \delta)^{\ell-1}\frac{1}{2}$$

This means $nR^2 > ((1+\delta)^{\frac{2}{\delta}})^{\ln(2nR^2)}\frac{1}{2}$. For $1 > x > 0$ it is a fact that $(1+x)^{2/x} > e$. Therefore the above equation implies $2nR^2 > e^{\ln(2nR^2)}$, which is a contradiction. Thus $\ell \leq 1 + 2\delta^{-1}(\log n + 2 \log R)$. Assuming R, the maximum value, is polynomially bounded in n we get $\ell = O(\delta^{-1} \log n)$. This proves the lemma. □

THEOREM 1. *The algorithm AHIST-S computes an $(1 + \epsilon)$-approximate B-bucket histogram in $O(nB^2\epsilon^{-1} \log n)$ time and $O(B^2\epsilon^{-1} \log n)$ space.*

PROOF. For each data point j and number of buckets k, we perform the minimization over the endpoint of every interval in the $(k - 1)$th list. This involves comparing $O(B\epsilon^{-1} \log n)$ numbers (using Lemma 2). Since j has n possibilities and k has B different values, the total time complexity is $O(nB^2\epsilon^{-1} \log n)$.

From Lemma 2, we need to maintain $O(B\epsilon^{-1} \log n)$ intervals for each interval list. Thus, the space required to store all interval lists is $O(B^2\epsilon^{-1} \log n)$. □

The time and space complexity expressed as a function of $\tau = \frac{B}{\epsilon} \log n$, are $O(nB\tau)$ and $O(B\tau)$ respectively, as described in Table I.

3.3 Incremental Histograms for Sliding Windows

In this section we summarize the result in Guha and Koudas [2002] where we considered the following problem: "can a data structure be maintained in the context of sliding window streams such that near optimal histogram representations can be computed on-demand efficiently?" As mentioned in the introduction, this yields an approximation algorithm for the original problem of constructing a histogram. The main idea we proposed was a "need based strategy". The algorithm is presented in Figure 5.

Note that the above algorithm does not evaluate all APXERR$[i, k]$ – but *if it did compute them, it would perform almost the same computation as the previous algorithm AHIST-S*. Thus, we can claim that at most $B\epsilon^{-1}\log n$ elements will be inserted in the kth list[5] as before. To analyze the running time of the algorithm observe that at most $B\epsilon^{-1}\log n$ binary searches are performed (one per insertion in the list/queue). Each binary search involves at most $\log n$ computations of APXERR$[c, k]$. Each such computation of APXERR$[c, k]$ will involve a minimization over $B\epsilon^{-1}\log n$ endpoints whose values are already stored in the $(k-1)$th list. Thus the total running time of $CreateQueue[1, n, k]$ is $O(B^2\epsilon^{-2}\log^3 n)$. The total time taken by the algorithm is $O(B^3\epsilon^{-2}\log^3 n)$.

THEOREM 2 (GUHA AND KOUDAS 2002; THEOREM 1). *For a sliding window data stream with window size n, we can output $(1 + \epsilon)$-approximate B-bucket histogram of the last n points seen using $O(B^3\epsilon^{-2}\log^3 n)$ time per new point.*

We omit a discussion of the proof since the result will be improved in the subsequent sections. The above theorem assumed that we were constructing a histogram for every new point—it is easy to see that the time complexity of maintaining the data structures is $O(1)$ per new point. If we construct a histogram only once, after seeing n points, the following corollary is immediate.

COROLLARY 1. *In $O(n + B^3\epsilon^{-2}\log^3 n)$ time and $O(n)$ space we can construct an $(1 + \epsilon)$-approximate B-bucket histogram over a data stream.*

We include the algorithm in Figure 5 in our experimental evaluation.

Implementation Issues. Note that a naive binary search is suboptimal. To see the issue clearly, consider the sequence of following numbers 1, 2, 4, 8, 16, 32, 64, 128, 256 and $\delta = 0.5$. If we are ensuring that the numbers have a "gap" of a factor $(1 + \delta) = 1.5$ then all numbers should be chosen. But a naive implementation will evaluate APXERR$[i, k]$ for $i = 16$, decide to go left, evaluate 4 and subsequently 2 to find the element 1. At this point, it would start repeating the process over the sequence $2, \ldots, 256$, which means evaluating APXERR$[i, k]$ for $i = 16$ again (assuming we take the floors while finding the middle element). We spell out the better alternative along with an improved algorithm.

3.4 The Fastest Algorithm: AHIST-L-Δ

We begin by noting the areas where the algorithm AHIST-S (and the algorithm FixedWindowHistogram) can be improved:

(1) The most amount of time spent by the previous algorithms is in maintaining the interval lists when the APXERR$[i, k]$ values are small. In such a case, the $(1 + \delta)$ approximation reduces to storing many consecutive i's. In fact, in a pathological example we found that all $i = 1 \cdots 47$ were present in the list (the later values stored in the list had larger gaps between consecutive items).

---

[5]The earlier paper [Guha and Koudas 2002] mentions queue; queue and list would mean the same thing for this paper and we would use them interchangeably.

(2) The reverse of the above situation occurs in the bad example noted in the implementation details of the previous section! In this case, suppose $B = 3$ and the APXERR$[i, 2]$ values are 1, 2, 4, 8, 16, 32, 64, 128, 256. Let $\delta = 0.03$. It is clear that all the values should be present in the list. However, the question to ask is: are all of the APXERR$[i, 2]$ values necessary? If we knew that the error was 20 we do need the value 256. But we do not know that the error is 20 till we have solved the problem!

To avoid both issues, we create a bootstrap. We first compute a rough estimate, say instead of 20 we get an upper bound of 80 (factor 4). This is not a good approximation, but allows us to rule out computing any APXERR$[i, 2]$ greater than 80 in the example above. The improvement may appear to be small, but it is not. This is because each item in the kth list uses all the items in the $(k-1)$th list. Reducing the maximum size of the lists by a factor g decreases the running time by a factor g^2. The key idea is a technique in approximation algorithms where we decompose the problem into two parts. In the inner part of the algorithm, we solve the problem assuming a parameter value is within a "good" range. The outer part searches for the appropriate range and sets the parameter. Note that for the idea to be successful, if the parameter is not in the good range, then the inner part must be able to discover that as well.

3.4.1 *The Algorithm* AHIST-L-Δ. As mentioned, we decompose the problem into two parts. The core part assumes that we have an estimate Δ of the error of the optimum histogram. If our estimate Δ is correct, that is, there is a histogram whose error is Δ then the algorithm will return a histogram of error $(1 + \epsilon)\Delta$. If our estimate is incorrect, then the histogram which is returned can have an arbitrary error. This core part of the algorithm will be denoted by SUB-AHIST-L-Δ.

The outer part, AHIST-L-Δ, will try to find Δ. It will first check if there is a solution with zero error. To achieve that, it will invoke SUB-AHIST-L-Δ with $\Delta = 0$. If there is indeed a solution with 0 error, SUB-AHIST-L-Δ will return a histogram of error at most $(1 + \epsilon)0 = 0$ and the problem is optimally solved.

Assuming that SUB-AHIST-L-Δ did not return a histogram with 0 error, the outer algorithm, AHIST-L-Δ, will invoke SUB-AHIST-L-Δ with the minimum possible nonzero error (which is 1/2 for integer input as discussed earlier). In particular if SUB-AHIST-L-Δ returns a histogram of error more than $(1 + \epsilon)\Delta$ for some Δ, then we know that the optimum error is more than Δ. We can then raise Δ to $(1 + \epsilon)\Delta$. Thus, at some point we would have the optimum answer in the range Δ to $(1 + \epsilon)\Delta$, and then invoking SUB-AHIST-L-Δ with the later bound will give a $(1 + \epsilon)^2$ approximation. This is the basic idea but the search for the range of the optimum answer can be improved. We will instead invoke SUB-AHIST-L-Δ with $\epsilon = 1$ (recall that SUB-AHIST-L-Δ is an approximation scheme, which allows us to set ϵ and get an approximation of suitable quality). This leads us faster to the range in which the optimum error lies, but at the end of the search we have more slack, that is, we get a $(1 + 1)^2 = 4$ approximation. From the 4 approximation, we will compute the $(1 + \epsilon)$-approximation in one step.

Procedure AHIST-L-Δ
begin
1. Create SUM, SQSUM to allow computation of SQERROR() in $O(1)$ time
2. $\rho := 1$, /* ρ will remain the same throughout this algorithm */

    /* check if solution is zero */

3. $Cutoff := 0$; $z := 0$
4. E := SUB-AHIST-L-Δ(B, $Cutoff$, ρ, z)
5. **if** $E = 0$ **return** the solution of the above

    /* We now begin the search for upper bounding the error */

6. $\Delta := 1/2$
7. $Cutoff := 4\Delta$; $z := \frac{\Delta}{2B}$ /* $\epsilon = 1$ */
8. E := SUB-AHIST-L-Δ(B, $Cutoff$, ρ, z)
9. **while** $E \geq 4\Delta$ **do** {
10. $\Delta := 2 * \Delta$
11. $Cutoff := 4\Delta$; $z := \frac{\Delta}{2B}$ /* $\epsilon = 1$ */
12.   E := SUB-AHIST-L-Δ(B,$Cutoff$,ρ,z)
13. }

    /* at this point we know that the optimum error is between */
    /* Δ, and 2Δ. Further, we have a solution of error E */

14. $Cutoff := E$; /* we do not need to search for larger error */
                     /* since we already have one of of error E */

15. $z := \epsilon\Delta/(2B)$
16. SUB-AHIST-L-Δ(B, $Cutoff$, ρ, z)
end

Fig. 6.   The AHIST-L-Δ.

The above also would shed some light on what properties we would need for SUB-AHIST-L-Δ, and more importantly, what are the design parameters. The simple guarantee "if there is a histogram with error Δ, then we find a histogram of error $(1 + \epsilon)\Delta$" does not suffice any more. In particular, to get the $(1 + \epsilon)$ approximation from a 4 approximation we need SUB-AHIST-L-Δ to preserve the following stronger condition: *if there is a histogram whose error is at most "Maxestimate", given a value $z > 0$, we will return a histogram whose error is at most the optimum error plus $(B-1)z$*. Now from the 4 approximation we can easily get a $(1+\epsilon)$ approximation by setting $z = \epsilon\Delta/(B-1)$ and *Maxestimate* = 4Δ (the optimum error is between Δ and 2Δ). This would explain the settings of the parameters of AHIST-L-Δ as described in the Figure 6; but before discussing the full algorithm we will add one extra twist to SUB-AHIST-L-Δ. We will add an extra parameter ρ, and SUB-AHIST-L-Δ will achieve the following:

Invariant.   If there is a histogram whose error is at most "Maxestimate", given a value $z > 0$, $\rho \geq 1$, we will return a histogram whose error is at most ρ^{B-1} times the sum of $(B - 1)z$ and the optimum error. *The running time of course will be a function of ρ, z.*

We introduce the parameter ρ because we would require it in the next section to develop a streaming algorithm. Instead of repeating near identical material (corresponding to $\rho = 1$ in AHIST-L-Δ and $\rho > 1$ later), we present one succinct proof/tool that can be used flexibly. This explains the choice of the input parameters in the algorithm SUB-AHIST-L-Δ.

The main complexity of the proof arises from the introduction of the term z. The issue is that in SUB-AHIST-L-Δ the evaluation of APXERR[$i + 5, k$] may rule

out the evaluation of APXERR$[i, k]$—this means that monotonicity of APXERR$[i, k]$ holds only on the set $\{i\}$ for which we have evaluated and stored APXERR$[i, k]$. This forces us to be more careful in the inductive proof. We first describe the algorithm SUB-AHIST-L-Δ in more detail and prove its properties and then subsequently prove the correctness and complexity bounds of the overall algorithm AHIST-L-Δ.

3.4.2 *SUB-AHIST-L-Δ.* Our overall goal is to create the intervals similar to those constructed by AHIST-S. This algorithm evaluated all APXERR$[i, k]$ for $1 \leq i \leq n - 1, 1 \leq k \leq B - 1$ and only retained those APXERR$[i, k]$ which were more than APXERR$[i', k]$ stored by a $1 + \delta$ factor (where i' was the largest element in the kth list less than i).

Reconsider the dynamic programming table constructed by the optimal algorithm; let TERR$[1, 1]$ be the bottom left corner and TERR$[n, B]$ be the top right corner. A metaphoric view of the algorithm AHIST-S could be the following: a "front" which moves from left to right and creates (approximately) the same table as the optimal algorithm, but only chooses to remember a few "highlights". The highlights corresponds to the boundary points which are sufficient to construct an approximate histogram. One way of conceptualizing the algorithm SUB-AHIST-L-Δ is that we want to create a similar table, but *in this case the front is moving from bottom to top*. More formally, all the APXERR$[*, k]$ we want to compute/store are computed before any APXERR$[*, k + 1]$ is computed. Note, we immediately have a problem that APXERR$[i, k + 1]$ may (and in the implementation, actually does) depend on APXERR$[i', k]$ where $i' > i$. This is where the old proof of the algorithm AHIST-S fails and a more subtle argument is required. However note that the "front" of AHIST-S proceeded left to right and therefore was applicable to streams. This property is lost in the bottom to top computation.

The algorithm SUB-AHIST-L-Δ is described in Figure 7. The critical component of the algorithm is the procedure *CreateBestList*, which creates the lists.

CreateBestList. The invocation of CreateBestList(1, n, k, *Cutoff*, ρ, z) (see Figure 7) computes the interval list for the k-th list by proceeding backward from the largest index (i.e., n). The subroutine ensures is that no element i with APXERR$[i, k] > $ *Cutoff* is placed in the list. However after placing an element, *Cutoff* is changed.

Initially when CreateBestList() is invoked, we have *Cutoff* = *Maxestimate* and no APXERR$[i, k]$ with value larger than *Maxestimate* is considered. After *CreateBestList* has found one such i that passes the cutoff, it resets the cutoff to (APXERR$[i, k] - z)/\rho$. The z is the additive error component and ρ is the multiplicative factor mentioned previously. In SUB-AHIST-L-Δ we have $\rho = 1$. *CreateBestList* first checks if APXERR$[start, k]$ passes the cutoff (i.e., is lower). If not, the entire interval can be thrown away without adding an element in the list. If indeed APXERR$[start, k]$ is below the cutoff, then we know that some APXERR$[i, k]$ for $start \leq i \leq end$ needs to be added to the list. The goal is to find the largest such i. We divide the interval $[start, end]$ into two pieces and recurse on the right half. This recursive call may change the cutoff, and when

Procedure SUB-AHIST-L-$\Delta(B, MaxEstimate, \rho, z)$
begin
/* Assume SUM, $sqvsum$ are available and we can compute SQERROR() */

1. **For** $k = 1$ **to** $B - 1$ {
2.   Initialize k-th interval list $Q[k]$ to empty
3. $Cutoff = MaxEstimate$; /* do not consider larger costs */
4.   CreateBestList(1, n, k, $Cutoff$, ρ, z)
5. }
6. APXERR$[n, B] := \infty$
7. **For** $i := 1$ **to** $size(Q[B - 1])$
8.   APXERR$[n, B] := min($APXERR$[n, B],$
        APXERR$[Q[B - 1].b[i], B - 1] +$ SQERROR$(Q[B - 1].b[i] + 1, n))$
9. **return** the solution found in the above step
end

Procedure CreateBestList($start, end, k, Cutoff, \rho, z$)

/* It is recursive and invokes itself with a changed value of $Cutoff$*/

begin
1. Compute APXERR$[start, k]$

    /* For k=1, this is SQERROR$(0, start)$, otherwise we have the following */
    /* APXERR$[start, k] = min_{b \in Q[k-1]}$ APXERR$[b, k - 1] +$ SQERROR$(b + 1, start)$ */
    /* **Important:** the minimization also looks at elements b in $Q[k - 1]$ */
     /* which are larger than/equal to $start$, in that case SQERROR$(b + 1, start) = 0$. */
    /* For these $b \geq start$ we need to only inspect the smallest b larger than/equal to start */

2. **if** (APXERR$[start, k] \geq Cutoff$)
3.   **return** $Cutoff$ /* basically drop the interval */
4. **while** ($start < end$) **do** {
5.   mid := (start+end+1)/2
6. $Cutoff :=$ CreateBestList($mid, end, k, Cutoff, \rho, z$)
     /* **Cutoff changes here** */
7.   **if** (APXERR$[start, k] \geq Cutoff$)
8.   **return** $Cutoff$ /* Drops interval, but list was changed */
9.   end := mid - 1
10. }
11. **if** (APXERR$[start, k] < Cutoff$) {
12.   Insert $start$ at the front of the k-th list $Q[k]$
13. $Cutoff := \frac{\text{APXERR}[start, k] - z}{\rho}$
14. }
15. **return** $Cutoff$
end

Fig. 7. The algorithm SUB-AHIST-L-Δ.

we return we check if APXERR$[start, k]$ is below the current *Cutoff*. We now apply the logic again—if $Cutoff >$ APXERR$[start, k]$ then for some $i' \in [start, mid]$ we need to add i' to the list. We proceed recursively till $mid = start$ or we have decided that the entire subinterval $[start, mid]$ can be discarded. If we are at $mid = start$ and *Cutoff* is still larger than APXERR$[start, k]$ then we need to add $start$ to the list. In all cases we return the updated value of *Cutoff*. *Observe, that no* APXERR$[start, k]$ *is evaluated more than once.*

The Price Paid. Unfortunately, the above comes with a price. The problem arises if we invoke *CreateBestList* with $z > 0$, since for any arbitrary j it is not clear that there is any element b_ℓ in the k-th list such that $b_\ell \geq j$. *The lists can be empty—more so because we will search for an appropriate value of Δ. If Δ is too small, by definition the list will be empty at a j which would have been useful for the histogram.* Anecdotally, this issue of the lists running empty has

been a source of many headaches in the implementations. *At the same time, the empty list is a "proof" that our estimate Δ was too low!*

This naturally leads us to the most interesting lemma in the paper, which at a high level, states that "if the estimations (for *Cutoff*, z) were correct, the list cannot run empty, and would have something useful for us". Before we proceed, we define some terms that will simplify the notation of the following discussion.

Definition 2.   Let TERR[0, 0] = APXERR[0, 0] = 0 and TERR[i, 0] = ∞; this defines the "zero bucket" case. If we are approximating the empty set with buckets, the error is 0, and if the set is nonempty, the error is ∞. Define $Q[0]$ = {0}. As the reader will notice, this makes the statement of Lemma 3 vacuously true and allows a simpler base case.

Further let SQERROR($i+1$, j) = 0 if $j \leq i$. This is a bucket which is "backward" that is, the right boundary is before the left boundary. This is a fictitious bucket and will be removed from the final solution. These buckets simply mean that the same error can be achieved by a smaller number of buckets. Note that this is used in the pseudocode as well.

LEMMA 3.   *For all k, $j \geq 1$, if MaxEstimate $\geq \rho^k$(TERR[j, k]$+kz$), then there exists an interval $[a_\ell, b_\ell]$ in the kth interval list produced by* CreateBestList *such that $a_\ell \leq j \leq b_\ell$ and* APXERR[b_ℓ, k] $\leq \rho^k$(TERR[j, k]$+kz$).

PROOF.   The statement of the Lemma is true for $k = 0$ vacuously, since TERR[j, k] = ∞ > *MaxEstimate*.

Let us assume the statement is true for $k - 1$. Let the last bucket in the best k-bucket approximation of the interval $[1, j]$ be $[i + 1, j]$ where $i < j$. We have:

$$\text{TERR}[j, k] = \text{TERR}[i, k - 1] + \text{SQERROR}(i + 1, j). \tag{5}$$

Now since *MaxEstimate* > ρ^k(TERR[j, k]$+(k-1)z$) from the above equation we have *MaxEstimate* > ρ^k(TERR[i, $k-1$]$+(k-1)z$) $\geq \rho^{k-1}$(TERR[i, $k-1$]$+(k-1)z$). Since the condition on *MaxEstimate* is satisfied, by the inductive hypothesis, we have an interval $[a', b']$ with b' in the $(k - 1)$th list (denoted by $Q[k - 1]$) satisfying the following conditions:

$$a' \leq i \leq b' \tag{6}$$
$$\text{APXERR}[b', k - 1] \leq \rho^{k-1}(\text{TERR}[i, k - 1] + (k - 1)z). \tag{7}$$

Now, consider the interval $[s', e']$ which decided the status of j in the kth list, that is, $Q[k]$. There are two cases to consider:

Case (A): j was inserted, and it must have been that $j = s'$.
Case (B): The entire interval $[s', e']$ containing j was dropped.

But in either case, we evaluated APXERR[s', k]. In evaluating APXERR[s', k], we minimized over all the elements in $u \in Q[k - 1]$ the sum APXERR[u, $k - 1$] + SQERROR($u + 1$, s'). Now $b' \in Q[k - 1]$ and since we minimized over all $u \in Q[k - 1]$, we have:

$$\text{APXERR}[s', k] \leq \text{APXERR}[b', k - 1] + \text{SQERROR}(b' + 1, s'). \tag{8}$$

because $b' \in Q[k-1]$ and we explicitly minimized APXERR$[s', k]$. Observe that we are using the "generalized" SQERROR$(b'+1, s')$ where SQERROR$(b'+1, s') = 0$ if $s \le b'$.

Now $i \le b'$ (by Eq. (6)) and $s' \le j$ (since $j \in [s', e']$). We claim that SQERROR$(b'+1, s') \le$ SQERROR$(i+1, j)$. There are two possibilities concerning b' and s', i.e., $b' < s'$ or $b' \ge s'$. If $b' < s'$ then we have SQERROR$(b'+1, s') \le$ SQERROR$(i+1, j)$ because $[b'+1, s']$ is a subinterval of $[i+1, j]$. Otherwise, if $b' \ge s'$ then by the definition of generalized SQERROR we have SQERROR$(b'+1, s') = 0$, and again $0 =$ SQERROR$(b'+1, s') \le$ SQERROR$(i+1, j)$ (SQERROR cannot be negative).

From Equations (7), (8) and SQERROR$(b'+1, s') \le$ SQERROR$(i+1, j)$ we get:

$$\text{APXERR}[s', k] \le \rho^{k-1}(\text{TERR}[i, k-1] + (k-1)z) + \text{SQERROR}(i+1, j).$$

The above implies:

$$
\begin{aligned}
\text{APXERR}[s', k] &\le \rho^{k-1}(\text{TERR}[i, k-1] + \text{SQERROR}(i+1, j) + (k-1)z) \\
&= \rho^{k-1}(\text{TERR}[j, k] + (k-1)z) < \textit{MaxEstimate}. \quad (9)
\end{aligned}
$$

Now the above means that we could not have dropped $[s', e']$ if $Q[k]$ was empty. Because when the list is empty, $\textit{Cutoff} = \textit{MaxEstimate}$, and the above equation contradicts the condition for dropping $[s', e']$. That means there is an element greater or equal to j in $Q[k]$. Let b_ℓ be the smallest such element.

If we were in case (A), that is, we inserted j in $Q[k]$, then $b_\ell = j$ and Eq. (9) proves the lemma. Therefore, for the remainder of the proof, we can assume that we are in case (B), and we dropped the entire interval $[s', e']$. That could have only happened if on the last insertion in $Q[k]$ (before dropping $[s', e']$) we must have set $\textit{Cutoff}$ such that APXERR$[s', k] \ge \textit{Cutoff}$. Let u be the element for which we set this $\textit{Cutoff}$. Therefore $\textit{Cutoff} = (\text{APXERR}[u, k] - z)/\rho$. But, in this case, u is the smallest element larger than j in $Q[k]$ and $u = b_\ell$. Thus, we have:

$$\text{APXERR}[s', k] \ge \textit{Cutoff} = \frac{\text{APXERR}[b_\ell, k] - z}{\rho}.$$

Combined with Eq. (9) this implies

$$\frac{\text{APXERR}[b_\ell, k] - z}{\rho} \le \text{APXERR}[s', k] \le \rho^{k-1}(\text{TERR}[j, k] + (k-1)z)$$

which after rearrangement proves the lemma for k. Therefore by induction, the lemma holds for all $k \le B$. □

The above (partially) proves the guarantees on the quality of approximation. We will shortly see how to use the guarantees. But before that we need to bound the running time. This is achieved by the following:

LEMMA 4.   *CreateBestList*$(1, n, k, \textit{Maxestimate}, \rho, z)$ *runs in* $O(\lambda^2 \log n)$ *time and creates a list of size* $O(\lambda)$ *where* $\lambda = \min\{\frac{\textit{Maxestimate}}{z}, \frac{1}{\rho-1} \log n\}$.

PROOF.   Suppose we executed line (4) for a particular invocation of *CreateBestList* with the parameters $(\textit{start}, \textit{end}, k, \textit{Cutoff}, \rho, z)$. Then we are guaranteed that, when we return from this invocation, $\textit{Cutoff}$ would have decreased (which also means the interval list would have increased by one). This

follows from the fact that if we did not change *Cutoff* in the recursive calls in the lines (4)–(10), we cannot return from inside of this loop since we entered the loop under the guarantee that *Cutoff* > APXERR[k, *start*]. In this case, we will proceed to lines (12) and (13). Between successive insertions *Cutoff* decreases by z and a factor of ρ. Immediately, we can see that the list size is at most $O(\lambda)$ where $\lambda = \min\{\frac{Maxestimate}{z}, \frac{1}{\rho-1}\log n\}$. The first part arises from the fact that each insertion into the queue corresponds to a difference of z and the maximum possible value in the queue is *Maxestimate*. The second part follows due to the same reasons as Lemma 2, i.e., geometric increase in factors of ρ and a bound of nR^2 on the maximum value. Observe that the bound is defined as long as we do not simultaneously have $\rho = 1$ and $z = 0$.

Let i and i' be two consecutive items in the list where i is followed by i', i.e., i' was inserted before. Let us focus on calculating the number of invocations of *CreateBestList* between (and not including) the two invocations that inserted i and i'. Each of these invocations did not add any item to the list or change *Cutoff* (since i and i' were consecutive in the list). Thus each of these invocations must have returned from line (3). Otherwise they would have changed the list as we discussed above. Thus, these invocations did not recursively call *CreateBestList*. Each of them took $O(1)$ time excluding an evaluation of APXERR[i, k]. Further these invocations corresponded to disjoint intervals.

Consider the binary tree built on $[1, \ldots, n]$—where the nodes correspond to the intervals and each interval is recursively halved. For each element (interval of length 1), there exists a unique path in the tree between i and i'. The invocations of *CreateBestList* between insertions of i' and i correspond to the intervals whose *parents* are in that unique path between i' and i. Because the path is of length at most $\log n$, the number of such invocations between any i' and i is $O(\log n)$.

Putting everything together, we enter $O(\lambda)$ values in the list, and between each entry we invoke *CreateBestList* at most $O(\log n)$ times. Overall, we invoke *CreateBestList* at most $O(\lambda \log n)$ times. Each call to *CreateBestList* leads to exactly *one* evaluation of APXERR[*start*, k]. Each evaluation of APXERR[*start*, k] is a minimization over $O(\lambda)$ values (size of the $(k-1)$th list). Thus, to construct each entire list we take $O(\lambda^2 \log n)$ time.   □

COROLLARY 2.   CreateBestList$(1, n, k, (2+2\epsilon)\Delta, 1, \frac{\epsilon\Delta}{B-1})$ *takes* $O(B^2\epsilon^{-2}\log n)$ *time and generates a* $O(B\epsilon^{-1})$ *size list.*

LEMMA 5.   *If* $\Delta \leq$ TERR[n, B] $\leq 2\Delta$*, then SUB-AHIST-L-*$\Delta(B, (2 + 2\epsilon)\Delta, 1, \frac{\epsilon\Delta}{B-1})$ *returns a histogram that has error at most* $(1 + \epsilon)$TERR[n, B] *in* $O(B^3\epsilon^{-2}\log n)$ *time.*

PROOF.   Suppose the last bucket of the optimum solution was $[i, n]$ and thus TERR[i, $B - 1$] + SQERROR$(i + 1, n)$ = TERR[n, B] $\leq 2\Delta$. By Lemma 3, since TERR[i, $B - 1$] $\leq 2\Delta$ we get an $i' \geq i$ in the $(B - 1)$th list such that (note, $\rho = 1$ and $z = \frac{\epsilon\Delta}{B-1}$),

$$\text{APXERR}[i', B - 1] \leq \rho^{B-1}\left(\text{TERR}[i, B - 1] + (B - 1)z\right) = \text{TERR}[i, B - 1] + \frac{\epsilon\Delta}{B - 1}$$

Because $i \leq i' \leq n$ we have $\textsc{Sqerror}(i' + 1, n) \leq \textsc{Sqerror}(i + 1, n)$. Adding this to the equation above we get,

$$\textsc{Apxerr}[i', B - 1] + \textsc{Sqerror}(i' + 1, n) \leq \textsc{Terr}[i, B - 1] + \epsilon \Delta + \textsc{Sqerror}(i + 1, n)$$

The right hand side is $\textsc{Terr}[n, B] + \epsilon \Delta$. Now $\textsc{Apxerr}[n, B] \leq \textsc{Apxerr}[i', B - 1] + \textsc{Sqerror}(i' + 1, n)$ because we minimize over the elements in the $(B - 1)$th interval list. Because $\textsc{Terr}[n, B] \geq \Delta$, we have

$$\textsc{Apxerr}[n, B] \leq \textsc{Terr}[n, B] + \epsilon \Delta \leq (1 + \epsilon)\textsc{Terr}[n, B]$$

The running time follows from Corollary 2.   □

3.4.2.1 *The Performance of AHIST-L-Δ.*   We are now ready to analyze the algorithm AHIST-L-Δ given in Figure 6.

THEOREM 3.   *In $O(n + B^3(\log n + \epsilon^{-2})\log n)$ time and $O(n + B^2\epsilon^{-1})$ space, AHIST-L-Δ can compute an $(1+\epsilon)$-approximate B-bucket histogram of n points. Furthermore, for a sliding window model we can compute a histogram of the previous n elements in time $O(B^3(\log n + \epsilon^{-2})\log n)$.*

PROOF.   The maximum possible values of Δ is nR^2 where R is the maximum number seen anywhere. Recall, from the introduction that R is assumed to be polynomially bounded and $\log(nR^2) = O(\log n)$. Unless we have a histogram of zero error, the error is at least $\frac{1}{2}$. So initially we satisfy that Δ is a lower bound on the error. If the optimum solution is between Δ and 2Δ, because $\epsilon = 1$ is passed to AHIST-L-Δ by Lemma 5, we are guaranteed a $(1 + 1)$-approximation. Thus, the solution returned must have cost at most 4Δ. If not, then we are guaranteed that no solution exists below 2Δ, and we increase Δ.

If indeed we see a histogram with error $E \leq 4\Delta$, we are guaranteed that it is a 4 approximation and by Lemma 5 we get a $(1 + \epsilon)$-approximation. Because Δ increases by factors of 2, we try at most $\log(nR^2) = O(\log n)$ values. For each of these invocations we set $\epsilon = 1$ and the running time of each is $O(B^3 \log n)$ which totals to at most $O(B^3 \log^2 n)$ time. In fact, we can replace $\log n$ by log of the optimum error.

The last invocation of SUB-AHIST-L-Δ requires $O(B^3\epsilon^{-2}\log n)$ time and the total time taken is $O(B^3(\epsilon^{-2} + \log n)\log n)$.   □

In retrospect, setting $z = 0$ and $\rho = 1 + \delta = 1 + \frac{\epsilon}{2B}$ gives us the algorithm FixedWindowHistogram.

Implementation Details.   The computation of $\textsc{Apxerr}[start, k]$ is the bottleneck in the above algorithm. For $\epsilon \ll 1$, the list sizes are large. We first find a quick estimate (which is a 2 approximation, but we do not use this fact) and it allows us not to consider all b_ℓ in the $(k - 1)$th list which have $\textsc{Apxerr}[b_\ell, k - 1]$ larger than our quick estimate. Furthermore we can stop considering all b_ℓ such that $\textsc{Sqerror}(b_\ell + 1, start)$ is greater than our estimate. Both of these yield significant benefits; however, we cannot prove that the pruning strategy yields time complexity.
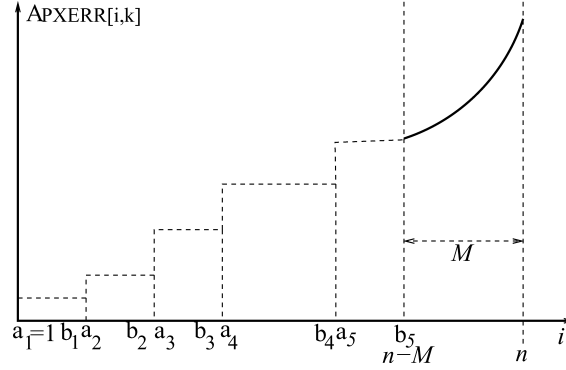
Fig. 8.   Approximating $\text{TERR}[i, k]$.

3.5 A $O(n)$ Time Streaming Algorithm: AHIST-B

The algorithm AHIST-S reads one element at a time and uses $O(B^2\epsilon^{-1}\log n)$ space and operates on a data stream. However, the running time of the algorithm is $O(nB^2\epsilon^{-1}\log n)$. The algorithm AHIST-L-$\triangle$ shows that if we were allowed to store $O(n)$ information, the running time is $O(n)$ (for large n). A natural question in this regard is: "can we get the best of both worlds?" Is there a streaming algorithm that uses small space (a small polynomial in B, $\log n$ and ϵ^{-1}) and takes $O(n)$ time (for large n)? We show that we can achieve such a streaming algorithm. This direction was first studied in Guha et al. [2004] in the context of relative error. The basic framework is the same as the AHIST-S algorithm except that we read a block of M, $M \ll n$ elements at a time. The central idea is a function *ExtendList* which would "extend" the lists after reading each block of M elements.

The idea of the function *ExtendList* is illustrated in Figure 8. Guha et al. [2004] describe the idea in detail. We summarize the main points for the sake of completeness, as well as for comparison, since we improve the algorithm. We maintain the increasing "staircase", which is the approximation. Assume that we processed r blocks of data values whose interval is $[1, n - M]$ and we are about to process the next block of M numbers. This new block, which we are reading, defines the solid section of the figure and we need to approximate that section into a staircase. There are two issues involved. First, while trying to compute $\text{APXERR}[i, k]$, for an element in the current block the elements in the $(k-1)$th list of the older blocks will take part in minimization. Second, instead of starting from the first item of the new block, a is set to the start point of the last interval of the kth list constructed for $[1, n - M]$. This means that the last entry of the kth list constructed for $[1, n - M]$ may be dropped. It also means that the intervals need to keep track of the start points as well as endpoints. We have $\lceil \frac{n}{M} \rceil$ blocks and for each block we spend time $O(B^3\epsilon^{-2}(\log M)\log^2 n))$. We quote the next theorem (restated in terms of V-Optimal error),

THEOREM 4 [GUHA ET AL. 2004: THEOREM 4.5]. *We can construct a $(1 + \epsilon)$-approximate B-bucket histogram in $O(n)$ time and $O(B^3\epsilon^{-2}(\log^2 n)(\log\log n + \log\frac{B}{\epsilon}))$ space.*

Naturally, $n > M = B^3\epsilon^{-2}(\log^2 n)(\log\log n + \log\frac{B}{\epsilon})$ in context of the above theorem. For more details and proof, see Guha et al. [2004]. In what follows, we show how to improve the above.

3.6 An Improved Algorithm: AHIST-B

Let us first investigate the algorithm AHIST-L-$\triangle$and the places from where the improvement arises, and if the strategy can be applied to streams. In AHIST-L-$\triangle$, the most important factor that affected the running time was the pruning achieved by the "Maxestimate" and *Cutoff*. Their effect was twofold. First, we did not compute the values which were large (because of *Cutoff*). Second, and somewhat in a less obvious way, we did not compute the values which were too small because of z. For a small APXERR$[i, k]$ setting *Cutoff* = APXERR$[i, k] - z$ ensures that *Cutoff* is negative and no element is added to the list subsequently.

The first idea cannot be implemented in streaming because these values, which appear to be large, currently may be useful as more data arrives. We have to compute the entire staircase for all k and pruning based on this idea cannot apply.

The second idea is also problematic. We cannot define "small" since we do not know the total error. A small value of APXERR$[i, k]$ which we have computed may be useful later if the subsequent blocks have all elements set to 0. The same value would be useless if the later blocks have a very large variations in numbers (and thus the total error is large).

However, the second idea can be applied *partially*. The kernel of the idea is that "we may have to compute a small value, but we need not reuse the value as we gather evidence that the value is less relevant". The idea is natural, but the question remains: how do we determine if a value APXERR$[i, k - 1]$ is relevant?

The answer is that APXERR$[i, k - 1]$ is used to compute APXERR$[j, k]$. Assume that we discover that APXERR$[j, k]$ is between 1000 and 2000 and the values of APXERR$[i, k - 1]$ and APXERR$[i', k - 1]$ differ by 1 (say $i < i'$). While computing APXERR$[j, k]$ more precisely we need not compute both APXERR$[i, k - 1] +$ SQERROR$(i + 1, j)$ and APXERR$[i', k - 1] +$ SQERROR$(i' + 1, j)$—the latter cannot be larger than the former by more than the difference APXERR$[i', k - 1] -$ APXERR$[i, k - 1]$ since SQERROR$(i + 1, j)$ is monotone nonincreasing in i.

> *The Main Idea.* For every APXERR$[j, k]$, instead of minimizing over the entire $(k - 1)$th list $Q[k - 1]$, we will create a sublist SUBQ$[k - 1]$ and only use these elements. This list SUBQ$[k - 1]$ will be created on the fly from $Q[k - 1]$, based on a 4 approximation of APXERR$[j, k]$, which we will derive first. Note that we will keep the $Q[k - 1]$ unchanged since it may be needed later.

A less intriguing observation is that if we keep track of the estimate APXERR$[n, B]$ of the optimum, where n is the last element of the last block read, we can discard all items in $Q[k]$ smaller than some small constant times APXERR$[n, B]/B$. This simply means that as we gather evidence that the optimum is simply large, we do not care about the small values. In contrast the
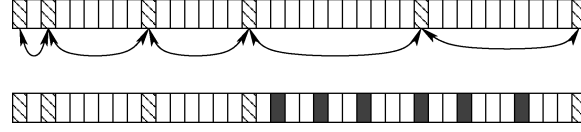
Fig. 9.   The lists $Q[k-1]$, $G[k-1]$, SuBQ$[k-1]$.

idea of SuBQ means that even though we do not know if TERR$[j, k]$ will be useful or not, in approximating TERR$[j, k]$ we can have APXERR$[j, k]$ − TERR$[j, k]$ proportional to TERR$[j, k]$ and still maintain an approximation factor. Observe that this also addresses the first improvement factor of AHIST-L-Δ partially as well. We cannot avoid computing the large values; but while computing the large values we a greater latitude in approximation. This allows us to perform less work. We reiterate that this does not mean that we relax the approximation guarantee, but the difference between the approximate and the optimal solution can be more if the optimal solution was already large. The improved algorithm is given in Figure 10. The new part is the *ExtendBestList* function.

ExtendBestList.   The pseudocode and the main idea is expressed in Figure 9. The array $Q[k-1]$ is shown on the upper part in Figure 9. We ensure that for two consecutive elements belonging to the array (endpoints of intervals ending at i and i') in the array satisfies APXERR$[i, k-1] \leq Cutoff =$ APXERR$[i', k-1]/\rho$. The last element in $Q[k-1]$ corresponds the last element of the last block (say y) read so far. This setup is similar to AHIST-S. We ensure that the first element u in the array $Q[k-1]$ also satisfies APXERR$[u, k-1] \geq Optestimate/(4B)$ where $Optestimate =$ APXERR$[y, B]$, since any value smaller than such can be ignored (follows from proof of Lemma 5). This is the more obvious idea mentioned earlier.

Along with $Q[k-1]$ we maintain a chain of subelements (indicated by the chain of pointers) any two consecutive elements a and a' satisfy APXERR$[a, k-1] \leq$ APXERR$[a', k-1]/2$ and a' is the largest sub-element in $Q[k-1]$ for which the condition holds.[6] Define this list of endpoints to be $G[K-1]$. This can be maintained easily as new elements are added to $Q[k-1]$ (at the right end).

If we were to use the elements a of $G[K-1]$ to minimize APXERR$[a, k-1] +$ SQERROR$(a+1, j)$ to get APXERR$[j, k]$, then we would get a 2 approximation for APXERR$[j, k]$. This follows because we will recursively maintain APXERR$[a, k-1]$ to be a close, that is, $(1 + \frac{(k-1)\epsilon}{2B})$ approximation of TERR$[a, k-1]$. Now, repeating the arguments in the proof of Lemma 5 (setting $n = j$ and $B = k$ in that proof), we can show that

$$\min_{a \in G[k-1]} \text{APXERR}[a, k-1] + \text{SQERROR}(a+1, j)$$

is a $(1 + \frac{(k-1)\epsilon}{2B}) * 2$ approximation. This explains the choice of 2, since we want the product to be at most 4.

Now we set *Cutoff* to be this 4 approximation (say $= C_{jk}$) of TERR$[j, k]$ and find the largest index u in $Q[k-1]$ such that APXERR$[u, k-1 \leq Cutoff$. We now proceed backward in $Q[k-1]$ to find the sequence of elements such that two

---

[6]The 2 can be changed to any constant $\sigma > 1$, but then we would first construct a 2σ approximation to APXERR$[j, k]$.

Procedure AHIST-B()
begin
1. Initialize every list $Q[k]$ to empty. Set z=0.
2. **For** $r = 1$ **to** n/M {
3.   Read the next block of M elements
4.   Compute SUM[i] and SQSUM[i] for $1 \leq i \leq M$ in the current block
5.   using SUM[M] and SQSUM[M] in the last block
6.   **For** $k = 1$ **to** $B - 1$ {
7.   Initialize k-th interval list $Q[k]$ to empty
8.   ExtendBestList(Q,k,Optestimate) /* explained in text */
9.   }
10.   APXERR$[n, B] := \infty$
11.   **For** $i := 1$ **to** $size(Q[B - 1])$
12.   APXERR$[n, B] := min($APXERR$[n, B],$
                APXERR$[Q[B - 1].b[i], B - 1] +$ SQERROR$(Q[B - 1].b[i] + 1, n))$
13. $Optestimate := \frac{\text{APXERR}[n,B]}{4B}$
14. }
end

Fig. 10.   The algorithm AHIST-B.

consecutive p' and p in that sequence (p' is chosen first and $p' > p$) satisfies APXERR$[p, k - 1] \leq$ APXERR$[p', k - 1] - z_{jk}$ where $z_{jk} = C_{jk}/(16B)$. Thus we arrive at a set of elements shown as shaded in the lower part of Figure 9. This is the list SUBQ$[k - 1]$. This list may or may not have overlap with $G[k - 1]$. *The reader must have noticed the similarity with the CreateBestList by now—in fact this is the idea, that we run a similar algorithm but adjust z depending on the j, k we are considering currently.* Since C_{jk} was at most 4TERR$[j, k]$ setting $z_{jk} = \epsilon C_{jk}/(16B)$ ensures that $z_{jk} \leq \epsilon$TERR$[j, k]/(4B)$. We can now repeat the proof of Lemma 3 and convince ourselves that we approximate APXERR$[j, k]$ recursively up to a factor $(1 + \frac{k\epsilon}{2B})$.

Analysis.   Observe that the size of $G[k - 1]$ is $O(\log n)$, because APXERR$[*, k - 1]$ of any two alternate elements in $G[k - 1]$ increase by factor 2 and the maximum value is nR^2. Thus, we compute a 4 approximation to APXERR$[j, k]$ in $O(\log n)$ time. Now, proceeding backward, we will only choose at most $O(B/\epsilon)$ elements in SUBQ$[k - 1]$. To identify each item in SUBQ$[k - 1]$, we will need to perform a binary search (exactly the same as CreateBestList), but the size of $Q[k - 1]$ is $O(\tau)$ (as we saw in AHIST-S). We can therefor summarize the discussion as:

LEMMA 6.   *We evaluate each* APXERR$[j, k]$ *in* $O(\frac{B}{\epsilon} \log \tau + \log n)$ *time. Note that in each such evaluation* SQERROR *is evaluated* $O(B\epsilon^{-1} + \log n)$ *times which corresponds to the sum of the sizes of* SUBQ$[k - 1]$ *and* $G[k - 1]$.

THEOREM 5.   *The algorithm AHIST-B (Figure* 10*) takes* $O(n + M\tau)$ *time and* $O(B\tau + M)$ *space where* $\tau = \min\{B\epsilon^{-1} \log n, n\}$ *and* $M = B(B\epsilon^{-1} \log \tau + \log n) \log \tau$.

PROOF.   Over the lifetime of the algorithm, once again we insert at most $O(\tau + \lceil \frac{n}{M} \rceil)$ elements in every list, since the inserted elements (except the last items in blocks) still grow in their APXERR values by a factor of $1 + O(\frac{\epsilon}{B})$. Furthermore, for every insertion of an element in the list, we have $O(\log M)$ elements

evaluated (due to a reason similar to *CreateBestList* but the size of a block is M). But now, when we evaluate APXERR$[i', k]$ at any point, we can use the bound from Lemma 6. The total time of insertions in the lists is (considering all B lists)

$$B(\log M)(B\epsilon^{-1}\log\tau + \log n)\left(\tau + \left\lceil \frac{n}{M} \right\rceil\right). \qquad (10)$$

We need to add $O(M\lceil\frac{n}{M}\rceil) = O(n)$ to the above since that is the time to read the blocks, create SUM and SQSUM etc. Observe that the overall space requirement is $O(M + B\tau)$. To get the coefficient of n to be a constant, we would like

$$B\epsilon^{-1}\log\tau + \log n = O\left(\frac{M}{B\log M}\right).$$

If $M = B(B\epsilon^{-1}\log\tau + \log n)\log\tau$, we can observe that $\log M = O(\log\tau)$ and the above condition necessary to set the coefficient of n to a constant is achieved. The running time is $O(n + M\tau)$ and the space required is $O(M + B\tau)$. $\square$

4. GENERALIZATIONS OF THE APPROXIMATION TECHNIQUES

In this section, we will revisit the results in the previous section to generalize our approximation schemes to cover a broad range of histograms and error measures. Observe that the following properties are used in our approximation schemes (the first three are required by the optimal algorithm):

(P1) The error of a bucket SQERROR(i, j) only depends on the i, j and $x_i, x_{i+1}, \ldots, x_j$.

(P2) The overall error, TERR$[n, B]$, is the sum of the errors of the B buckets.

(P3) We can maintain $O(1)$ information for each element such that given any i, j the value of SQERROR$(i + 1, j)$ can be computed efficiently. In the algorithms we maintained SUM, SQSUM values to compute SQERROR$(i + 1, j)$ in $O(1)$ time.

(P4) The error is *interval monotone*, that is, for any interval $[i, j]$ we have SQERROR$(i, j) \le$ SQERROR$(i, j + 1)$ and SQERROR$(i, j) \le$ SQERROR$(i - 1, j)$.

(P5) The value of the largest number R (therefore the maximum error) and the minimum nonzero error is polynomially bounded in n.

The fact that the above properties suffice for the correctness of lemmas and theorems can be proved by inspection. We now show the most general theorem that can be achieved on the basis of the algorithms we have seen.

THEOREM 6.   *Suppose we are given a histogram construction problem where the error E_T satisfies the conditions $(P1)$–$(P5)$ Suppose the error of a bucket $E_B(i + 1, j)$ can be computed in time $O(Q)$ from the records INFO$[i]$ and INFO$[j]$ each requiring $O(P)$ space. Assume that the time to create the $O(P)$ structure is $O(T)$ then by changing the function that computes the error given the endpoints we achieve the following (recall $\tau = B\epsilon^{-1}\log n$):*

(i)   *We can find the optimum histogram in $O(nT + n^2(B + Q))$ time and $O(n(P + B))$ space based on the* VOPT *algorithm.*

(ii) *In $O(nT + nQB\tau)$ time and $O(PB\tau)$ space, we can find a $(1+\epsilon)$ approximation to the optimum histogram based on AHIST-S.*

(iii) *In $O(nT + QB^3(\log n + \epsilon^{-2})\log n)$ time and $O(nP)$ space, we can find a $(1+\epsilon)$ approximation to the optimum histogram based on AHIST-L-$\triangle$.*

(iv) *In $O(nT + M_Q\tau)$ time and $O(PB\tau + M_Q)$ space we can find a $(1+\epsilon)$ approximation to the optimum histogram based on AHIST-B, where $M_Q = B(\frac{QB}{\epsilon} + Q\log n + \frac{B}{\epsilon}\log\tau)\log(Q\tau)$.*

(v) *The algorithms* (ii) *and* (iv) *extend to data streams where the input $x_1, \ldots, x_i, \ldots$ arrive in increasing order of i.*

In the above, only part (iv) has a different form than previously seen. This is because the running time, based on Lemma 6, changes to:

$$B(\log M_Q)((B\epsilon^{-1} + \log n)Q + B\epsilon^{-1}\log\tau + \log n)\left(\tau + \left\lceil \frac{n}{M_Q} \right\rceil\right).$$

The extra term accounts for the fact that SQERROR() needs $O(Q)$ time instead of $O(1)$. The above theorem assumes that input items arrive one by one and we preprocess them. The next theorem applies to the strategy that instead of looking at items one by one, we can preprocess the entire data (before we embark on histogram construction) in one shot so that we can support some efficient querying during the process of histogram construction. The theorem is applied in Corollary 3.

THEOREM 7. *Suppose we are given a histogram construction problem where the error E_T satisfies the conditions $(P1)$–$(P5)$. Suppose on we can preprocess the input in $O(nT)$ time and $O(nP)$ space such that the bucket error $E_B(i + 1, j)$ can be computed in time $O(Q)$ using the preprocessed data structure, in $O(nT + QB^3(\log n + \epsilon^{-2})\log n)$ time and $O(nP)$ space, we can find a $(1 + \epsilon)$ approximation to the optimum histogram based on AHIST-L-$\triangle$.*

We consider the following examples (i) approximation by degree d polynomials (ii) χ^2-test error, defined in Donjerkovic et al. [1999] in defining compressed histogram (iii) sum of absolute error. As indicated in the introduction, the algorithms carry over to the relative error setting, see Guha et al. [2004].

4.1 Approximation by Piecewise Degree-*d* Polynomials

Suppose instead of using a piecewise constant representation we use a piecewise linear representation. The error is still the sum of squares of the error seen at each point i. The bucket $[a+1, b]$ is represented by the polynomial $p_1(i-a)+p_0$ where the coefficients are p_1 and p_0. The error seen by the bucket is

$$E_B(a + 1, b) = \sum_{i=a+1}^{b} (p_1(i - a) + p_0 - x_i)^2.$$

It is easy to see that the error is interval-monotone. The standard method of finding the best values of p_0 and p_1 is to set both partial derivatives, with

respect to p_0 and p_1, to 0 and solve the resulting equations. The equations are:

$$\sum_{i=a+1}^{b} p_1(i-a) + (b-a)p_0 = \sum_{i=a+1}^{b} x_i$$

$$\sum_{i=a+1}^{b} p_1(i-a)^2 + \sum_{i=a+1}^{b} (i-a)p_0 = \sum_{i=a+1}^{b} (i-a)x_i.$$

If we set $b - a = r$, the above simplifies to

$$p_1 \frac{r(r+1)}{2} + rp_0 = \sum_{i=a+1}^{b} x_i$$

$$p_1 \frac{r(r+1)(2r+1)}{6} + \frac{r(r+1)}{2} p_0 = \sum_{i=a+1}^{b} i \cdot x_i - a \sum_{i=a+1}^{b} x_i.$$

If we store $\sum_{i=1}^{b} i \cdot x_i$, $\sum_{i=1}^{b} x_i$ and $\sum_{i=1}^{b} x_i^2$, we can find p_0, p_1 and $\mathrm{E}_B(a+1, b)$. Thus, we can apply Theorem 6 with $P = Q = T = O(1)$. In case of representation by degree-d polynomials, the error is given by

$$\mathrm{E}_B(a+1, b) = \sum_{i=a+1}^{b} \left[\left(\sum_{j=0}^{d} p_j(i-a)^j \right) - x_i \right]^2.$$

This sets up $(d+1)$ linear equations which give us the best $p_0, \ldots, p_d$, namely (once again, if $r = b - a$)

$$\begin{bmatrix} r & \sum_{i=1}^{r} i & \cdots & \sum_{i=1}^{r} i^d \\ \sum_{i=1}^{r} i & \sum_{i=1}^{r} i^2 & \cdots & \sum_{i=1}^{r} i^{d+1} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{i=1}^{r} i^d & \sum_{i=1}^{r} i^2 & \cdots & \sum_{i=1}^{r} i^{2d} \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \\ \vdots \\ p_d \end{bmatrix} = \begin{bmatrix} \sum_{i=a+1}^{b} x_i \\ \sum_{i=a+1}^{b} (i-a)x_i \\ \vdots \\ \sum_{i=a+1}^{b} (i-a)^d x_i \end{bmatrix}$$

Thus, we have $(d+1) \times (d+1)$ system of linear equations set up, which we can solve in $O(d^3)$ time using standard techniques. Observes the number of different coefficients in the matrix on the left is $2d + 1$. The $\sum_{i=1}^{r} i^m$ can be computed if the sum has been computed in time $O(m)$ for the previous $m - 1$ powers. Thus, the coefficients in the matrix can be computed in time $O(d^2)$ overall. The simplest way to compute the right hand side is to express each item into at most $O(d)$ terms like $\sum_{i=1}^{b} i^m x_i$. Thus, the right hand side can also be computed in time $O(d^2)$. In overall $O(d^3)$ time, we can answer $\mathrm{E}_B(a+1, b)$ if we store $\sum_{i=1}^{b} i^m x_i$ for $m = 0, 1 \ldots, d$ and $\sum_{i=1}^{b} x_i^2$. In this case, we have $T = P = O(d)$ and $Q = O(d^3)$ and thus we can summarize as follows:

THEOREM 8. *For constructing the best representation using piecewise degree-d polynomials under the ℓ_2^2 norm (same as V-Optimal measure) Theorem 6 can be applied setting $T = P = O(d)$ and $Q = O(d^3)$.*

4.2 The χ^2 Error

Under the celebrated χ^2 goodness of fit, the error of the hypothesis h_i "fitting" the data x_j is given by $\frac{(x_j - h_i)^2}{h_i}$. This was suggested by [Donjerkovic et al. 1999]. Thus, if we assume that the data *can be approximated by B buckets in which the distribution is uniform* (*constant*), we can try to use the χ^2 goodness of fit to find the best possible buckets that fit the data. The error of the bucket $[a+1, b]$ represented by p, is $\mathrm{E}_B(a+1, b) = \sum_{i=a+1}^{b} (x_j - p)^2 / p$. The total error is

$$\sum_{i=1}^{B} \sum_{j \in [s_i, e_i]} \frac{(x_j - h_i)^2}{h_i}$$

The error $\mathrm{E}_B(a+1, b)$ is minimized when we have $p^2 = (\sum_{i=a+1}^{b} x_i^2)/(b-a)$, and further we can prove that $\mathrm{E}_B()$ is interval monotone. Thus, the overall error of a bucket is

$$\mathrm{E}_B(a+1, b) = 2 \left(\sqrt{(b-a) \sum_{i=a+1}^{b} x_i^2 - \sum_{i=a+1}^{b} x_i} \right)$$

Thus, $\mathrm{E}_B(a+1, b)$ can be computed from $\textsc{Sum}[i]$ and $\textsc{Sqsum}[i]$ in $O(1)$ time. We thus get the following:

THEOREM 9. *For the χ^2-error Theorem 6 is applicable with $T = P = Q = O(1)$.*

4.3 The Sum of Absolute Errors

Several researchers (e.g., [Poosala et al. 1996] and [Matias et al. 1998]), have proposed that the sum of the errors $|x_j - h_i|$ at each point j is a desirable error function in several scenarios. The overall error with a B bucket histogram in this case is:

$$\sum_{i=1}^{B} \sum_{j \in [s_i, e_i]} |x_j - h_i|$$

The representative of a bucket $\mathrm{E}_B(i+1, j)$ in this case is given by the *median* of the values $x_{i+1}, \ldots, x_j$. The computation of the error is straightforward if along with finding the median, we also compute the sum of the values above and below the median. It is quite easy to see that the function $\mathrm{E}_B(i, j)$ is interval-monotone. The following is straightforward, see Guha et al. [2004].

PROPOSITION 1. *Given n numbers $x_1, \ldots, x_n$, we can preprocess the numbers using $O(n \log n)$ time and space such that given any interval $[i, j]$ we can compute the median of the numbers $x_i, \ldots, x_j$ (as well as the sum of distances from the median) in $O(\log^2 n)$ time.*

The algorithm that achieves the above basically follows the merge-sort routine, but maintains the sorted sublists created corresponding to the different intervals. Given an interval $[i, j]$ we can decompose the interval into at most $2 \log n$ intervals which cover the interval $[i, j]$ exactly and corresponds to the

intervals defined by the merge-sort tree. Given k sorted lists, we can easily find an element of rank s in time $O(k \log n)$ using a carefully modified binary search. Given the above and Theorem 7, we get the following corollary.

COROLLARY 3. *For the sum of the absolute errors (ℓ_1-error), in $O(n \log n + B^3(\log n + \epsilon^{-2}) \log^2 n)$ time and $O(n \log n)$ space, we can find a $(1 + \epsilon)$ approximation to the optimum histogram based on AHIST-L-Δ.*

5. EXPERIMENTAL RESULTS

We conducted experiments on real-life as well as synthetic data sets to evaluate the performance gains achieved by the approximation algorithms. Our focus was to demonstrate the effectiveness of the approximation techniques. Therefore we present the comparisons between the approximate and optimum algorithms for the V-Optimal error only. We used our implementations of the faster V-Optimal histogram construction algorithm mentioned in Jagadish et al. [1998].

5.1 Algorithms

We evaluated the various schemes and show the performance figures of the following:

—*VOPT* represents the V-Optimal histogram construction algorithm [Jagadish et al. 1998] presented in Section 2.3.

—*GK02* represents the algorithm described in Guha and Koudas [2002]. As mentioned in the introduction, this algorithm was developed for constructing histograms for sliding window streams and was not designed to be the best in class histogram construction as is the goal of this article. This algorithm in particular serves as a foil for demonstration of the benefit of the algorithm AHIST-L-Δ.

—*AHIST-L-Δ* represents the approximate histogram construction algorithm described in Section 3.4. This is the best offline approximation algorithm.

—*AHIST-B* represents the improved hybrid algorithm in Section 3.5 based on the ExtendBestList algorithm. This is the best streaming approximation algorithm. We tried block sizes of 256 to 4096 in powers of 2. We made sure that block size was no more than half the number distinct values of data, that is, there were always two or more blocks. The algorithms are labeled as *AHIST-B-256*, etc.

All experiments reported in this section were performed on 2.0 GHz Pentium-4 machine with 1 GB of main memory, running the Linux operating system. All the methods were compiled using version 3.2.2 of the gcc compiler.

5.2 Data Sets

5.2.1 *Synthetic Data Sets.* The synthetic data sets allowed us to vary the parameters in a controlled fashion. We considered one-dimensional synthetic data distribution. The data sets are generated with Zipfian frequencies for various levels of skew. We varied the skew parameter values between 0.3 (low skew)
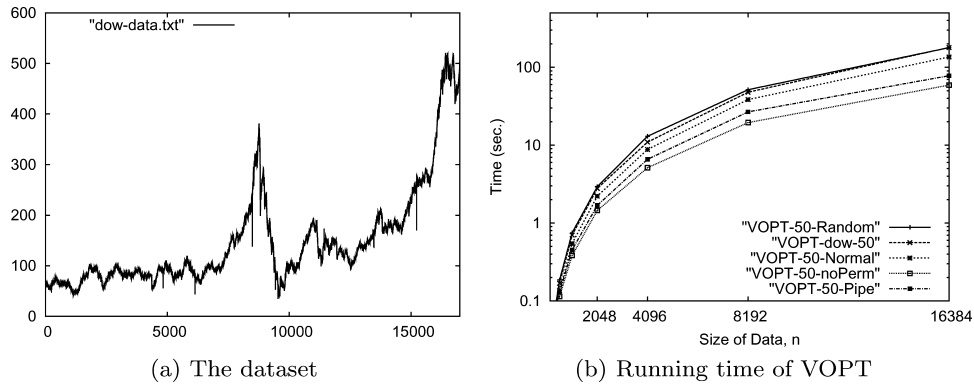
(a) The dataset   (b) Running time of VOPT

Fig. 11. The real life dataset, (b) compares the running time of VOPT on prefixes of this data compared to the synthetic data.

and 2.0 (high skew), the distinct values between $256 (= 2^8)$ and $65536 (= 2^{18})$, and the tuple count was set to 1,000,000. Note that we did not vary the number of tuples as the time and space complexities are independent of this.

A permutation step was also applied on the produced Zipfian frequencies to decide the order of frequencies over the data domain. We experimented with four different permutation techniques that were used in Garofalakis and Gibbons [2004, 2002]: *NoPerm*, *Normal*, *PipeOrgan* and *Random*. The detailed description of these permutations are presented below:

—*NoPerm* does not change the order of frequencies produced by the Zipfian data generator, that is, smaller values have higher frequencies.
—*Normal* permutes the frequencies to resemble a bell-shaped normal distribution, with higher frequencies at the center of the domain.
—*PipeOrgan* permutes the frequencies in a "pipe-organ"-like arrangement, with higher frequencies at the two ends of the data domain.
—*Random* permutes the frequencies in a completely random manner over the data domain.

5.2.2 *Real Life Data Set.* To see how effectively AHIST performs, we measured its behavior over real-life data sets. Because the key aspect of the approximate histogram constructions is improved asymptotic performance with guaranteed near optimal quality, we needed large datasets. We show the results for the Dow-Jones Industrial Average (DJIA) data set available at StatLib[7] that contains Dow-Jones Industrial Average (DJIA) closing values from 1900 to 1993. There were a few negative closing values and some obvious errors (like 100.**,10,100.** for consecutive closings)—we removed these errors, and focused on the first 16384 values so that we can compare the running time of the VOPT for this dataset and synthetic data. The dataset is plotted in Figure 11(a). Figure 11(b) shows the running time of the VOPT algorithm on the datasets at $B = 50$ when skew was set to 1. For the DJIA data sets, we used the prefixes
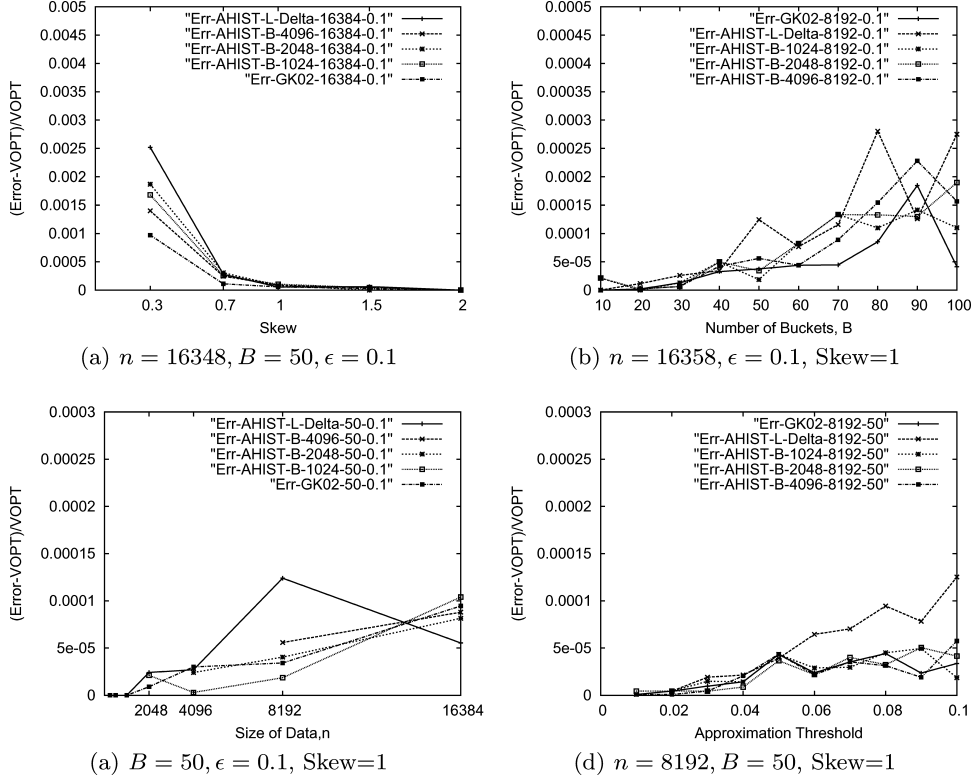
[7]http://lib.stat.cmu.edu/datasets/djdc0093.

Fig. 12. Quality of the histograms obtained.

to create datasets of different sizes. The Figure establishes that the VOPT algorithm takes similar running times on the same size of data.

5.3 Experimental Results—Synthetic Data Sets

We present some of our experimental results with synthetic data sets for frequency permutation and settings of Zipfian skew.

5.3.1 *The Quality of Histograms Constructed.* The most important issue is obviously the quality of the algorithms with respect to the optimum solution. As expected, the histograms were within $(1+\epsilon)$ factor of the true error (computed by VOPT). However, the actual error was significantly less. We show the results in Figure 12. Subfigure (a) represents the error as the skew was varied, (b) represents the errors as the number of buckets was varied in the range 10–100. Figure 12(d) shows, that setting $\epsilon = 0.1$ gave a fairly accurate histogram already, the quality improved for smaller values. For the rest of the article, we report the results for $\epsilon = 0.1$ mostly, except to show explicit dependence on ϵ.

5.3.2 *Running Times: Skew of Data.* Figure 13 reports the performance of the algorithms as the skew parameter was varied. All the algorithms improved as the skew value increased. However, in case of VOPT, the improvement (drop

(a) $n = 16358$, $B = 50$, Random

(b) $n = 16384$, $B = 50$, Normal

(c) $n = 16358$, $B = 50$, Pipe
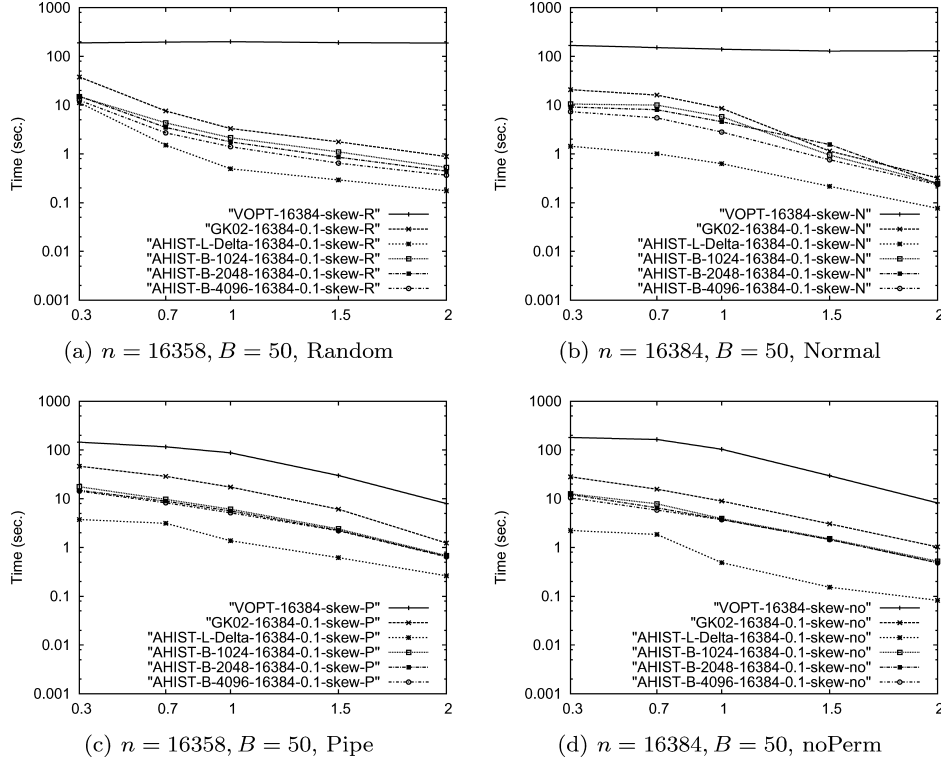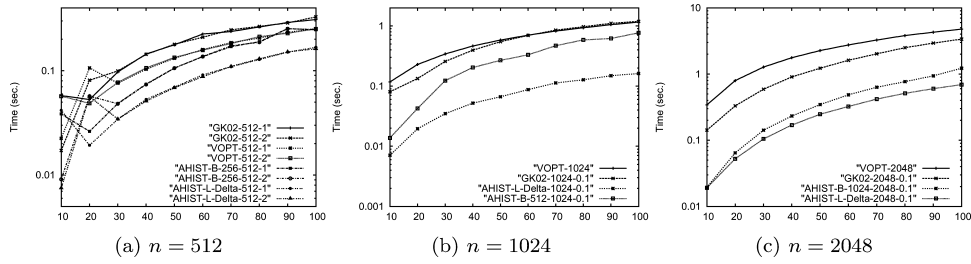
(d) $n = 16384$, $B = 50$, noPerm

Fig. 13. Running times on varying the skew parameter.

in running time) was significantly smaller and much less dramatic compared to the approximation algorithms. This is expected since for sharply concentrated distribution there are clear notions of "right" bucket boundaries. The approximation algorithms found these boundaries quickly, and these boundaries were stored in the queue. The algorithm VOPT, it also stops searching after finding the "right" boundary—but had to run over at least one bucket entirely to hit the boundary. The approximation algorithms "jumped" from boundary to boundary and were faster than the optimal algorithm in running time.

This (drop) was particularly striking in case of the Random permutation. The distributions are shown in Figure 14, which illustrates the characteristics of the data, and the intuition of "right" boundaries. Notice that in Figure 13(a) after the dramatic drop, the approximation algorithms flattened out much more compared to Figures 13(b), (c) (d). This is easy to see—for the Normal, noPerm and Pipe permutations, for very high skew value two to three buckets capture the distribution. The running time of noPerm was the fastest for all algorithms, including VOPT, since there was heavy pruning due to the monotonic distribution. Whereas for Random, optimum stays the same—because of the "random" nature two to three buckets are never sufficient to describe the data. The approximation algorithms for the random permutation dropped very quickly and found the important buckets—but did not improve dramatically with large skew since the number of boundaries did not drop sufficiently faster.

(a) Random, Skew=0.3   (b) Random, Skew=1.0   (c) Random, Skew=2.0
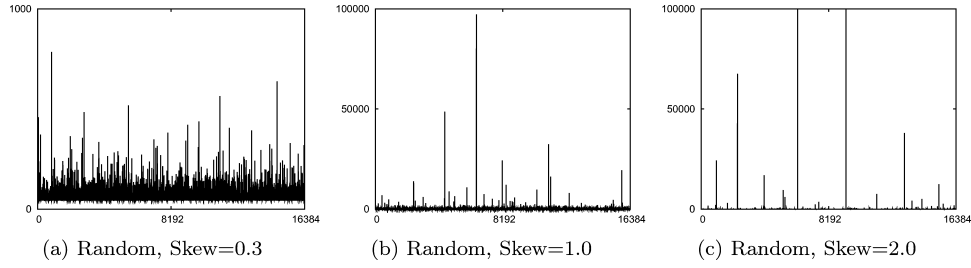
Fig. 14.   Zipfian distribution under random permutation $n = 16384$.

(a) $n = 512$ (b) $n = 1024$ (c) $n = 2048$

Fig. 15.   Running time for small n as B is varied $\epsilon = 0.1, skew = 1$.

In the rest of the article, we report only the experiments with skew $= 1$, since that appears to be the median value.

5.3.3 *Running Times: Dependence on B.*   The running time of the algorithms are compared as a function of the parameter B. The number of distinct elements is varied from small value to a large value. We could not run the VOPT algorithm for $n > 16384$.

5.3.3.1 SMALL n.   For very small n, $n = 256$, all the algorithms took a small amount of time—their running time varied across runs since they were so small. For $n = 512$, the running times stabilized for larger B across runs. This is shown in Figure 15(a). Already AHIST-L-Δ was significantly faster than VOPT. AHIST-B was faster for small values of B. At larger n, Figures 15(b) and (c), the approximation algorithms began to dominate VOPT. The parameter M was set to $n/2$ in these cases—it is smaller than what is suggested by the function of $B, \epsilon, \log n$. Note that all the approximation algorithms assume that $B\epsilon^{-1} \log n$ (natural logarithm ln) is smaller than n, otherwise, the optimization is no better than VOPT, for example, $B = 30, \epsilon = 0.1$ means that $n/\log n > 300$. This means $n > 2325$ and naturally we would see approximation algorithms performing significantly better at or close to this value of n.

5.3.3.2 LARGER n.   With larger values of n, the approximation algorithms performed orders of magnitude better. This is shown in Figure 16.

5.3.4 *Running Times: Fixed B.*   We varied the parameter B for all the algorithms. The results are shown in Figure 17. Recall that at $B = 30$ we calculated
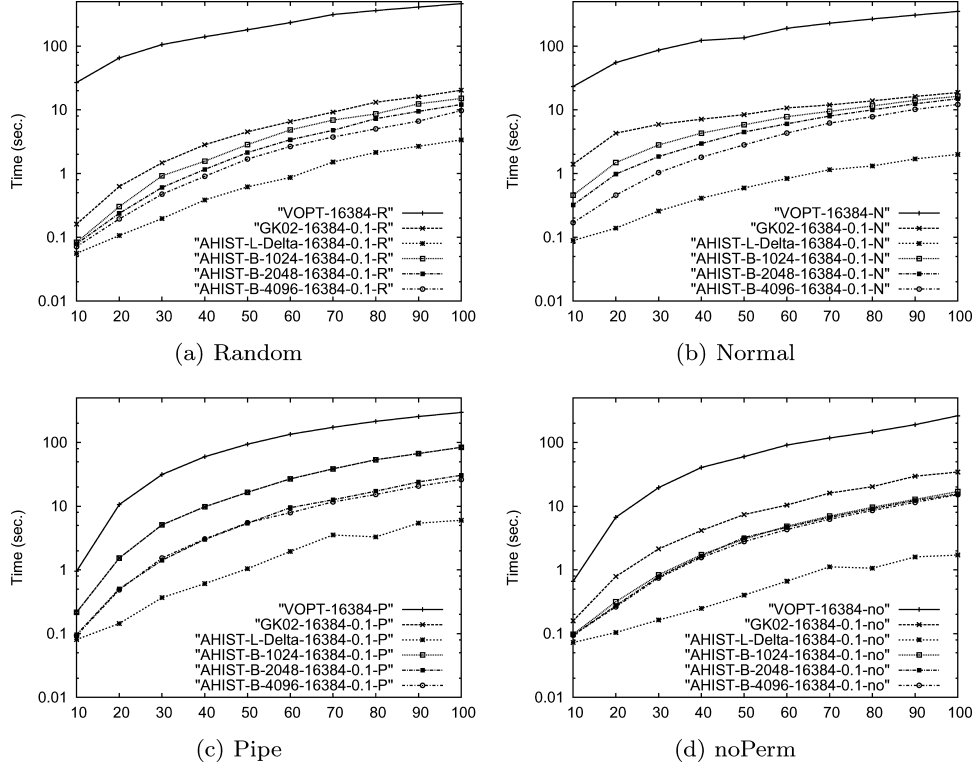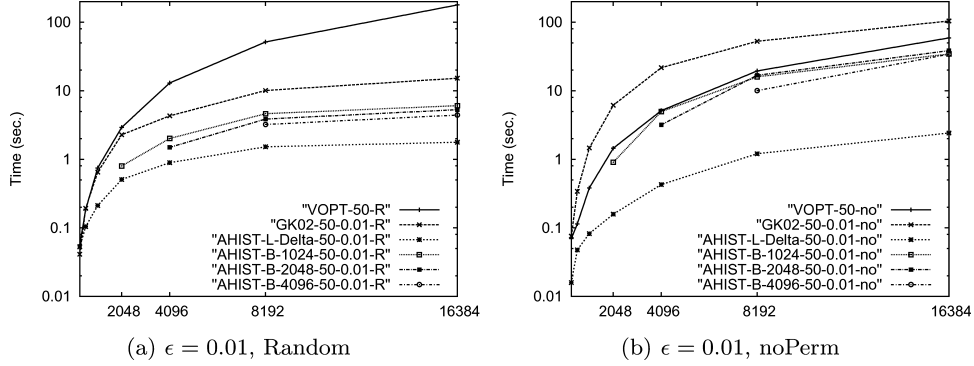
Fig. 16. Running time for large n as B is varied, $n = 16384$, $\epsilon = 0.1$, $Skew = 1$.
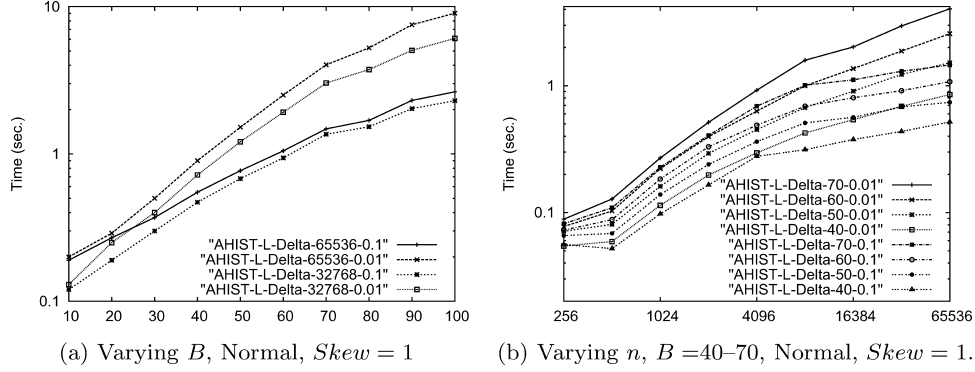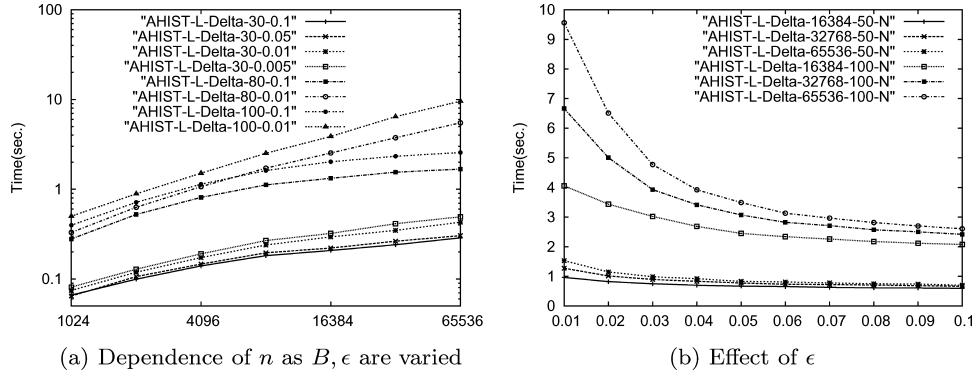
the "crossover" point to be 2325; and we see that for $n \geq 4096$ the approximation algorithms are constantly winning.

5.3.5 *Effect of ϵ.* The effect of small epsilon, $\epsilon = 0.01$ is shown in Figure 18, B is fixed at 50. Further results are shown in Figures 20 and 22, which show how the algorithms AHIST-L-Δ and AHIST-B scaled with ϵ. We show the graphs for Random and noPerm only, which are the extremes for VOPT. Characteristics for Pipe is similar to noPerm, and characteristics for Normal is similar to Random. Note that AHIST-L-Δ is significantly faster than VOPT in any of these cases. GK02 performed well compared to VOPT on Random (and Normal) but did worse on noPerm (and Pipe). The AHIST-B algorithms performed well or just slightly better. Note that the theoretical crossover point for the approximation algorithms to perform well in this case for $B = 30$ is 23250 (since ϵ is smaller), which explains the issue with noPerm and Pipe. We will show that in case of real life data, the approximation algorithms performed significantly better at much smaller values of n compared to the crossover.

5.3.6 *Scaling up: AHIST-L-Δ.* The scale-up experiments for the algorithm AHIST-L-Δ are summarized in Figure 19. Due to lack of space we show the results for the parameter $Skew = 1$, and Normal permutations only, the behavior is similar for alternate settings of the skew value and the permutation. The

(a) Random

(b) Normal

(c) Pipe

(d) noPerm
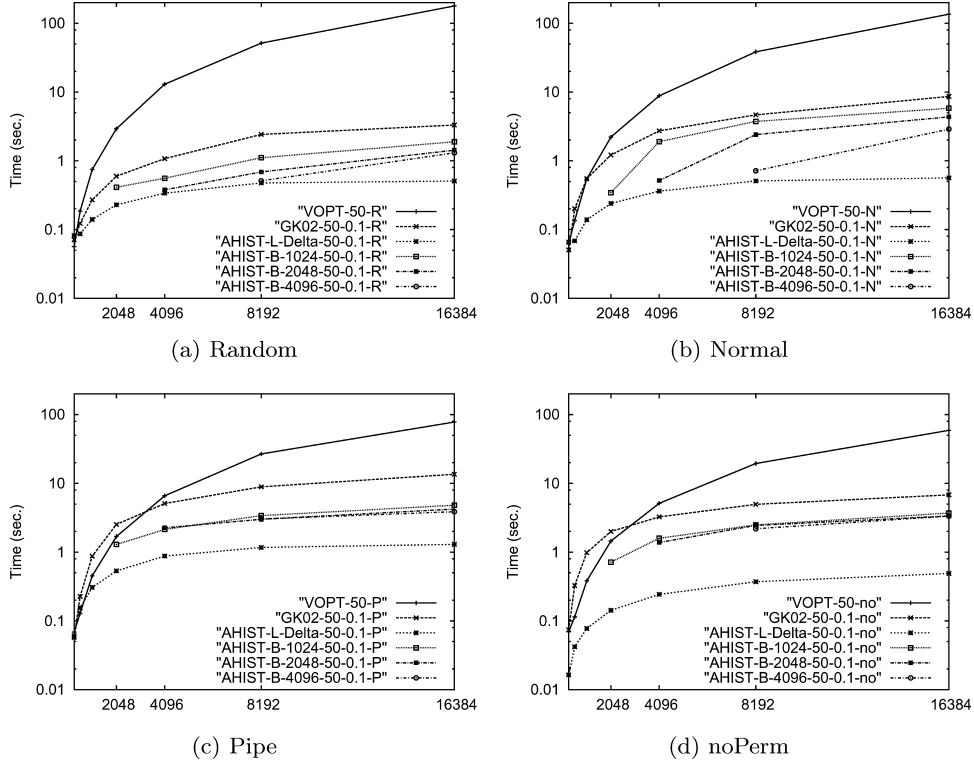
Fig. 17. Performance as n is varied, $skew = 1$, $\epsilon = 0.1$, $B = 50$.

(a) $\epsilon = 0.01$, Random

(b) $\epsilon = 0.01$, noPerm

Fig. 18. Effect of ϵ as B is varied, $n = 16384$, $skew = 1$.
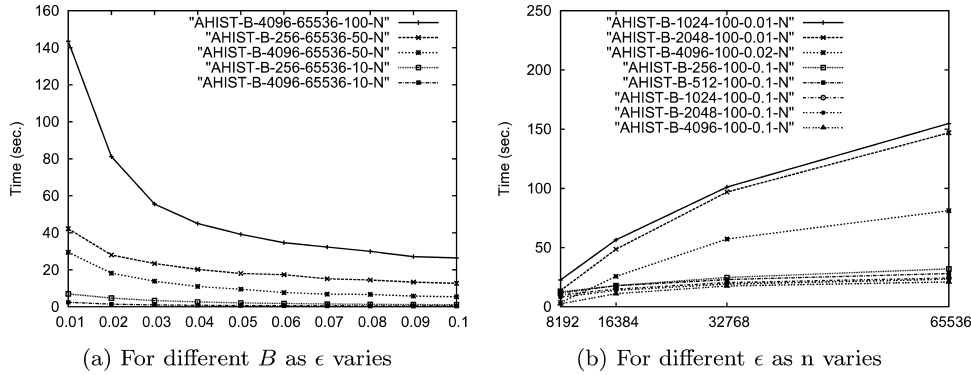
subfigure (a) shows how the program behaved as B increased. Interestingly, initially (for small B) the running time of AHIST-L-Δ depended on n rather than ϵ. As the parameter B increased the dependence shifted to ϵ. This is also shown in Figure 20(a) where for $B = 30$ the algorithm performed similarly for $\epsilon \in [0.005, 0.1]$, but for larger B the $\epsilon = 0.01$ and $\epsilon = 0.1$ cases got clearly separated. How this separation started is shown in more detail in Figure 19(b). Figure 20(b) shows that for smaller B the algorithm behaved comparably as

(a) Varying B, Normal, $Skew = 1$ (b) Varying n, $B =$40–70, Normal, $Skew = 1$.

Fig. 19.   Running time of AHIST-L-Δ.

(a) Dependence of n as B, ϵ are varied   (b) Effect of ϵ

Fig. 20.   Effects of the parameters on running time of AHIST-L-Δ, Normal perm., $skew = 1$.

ϵ, n was varied, whereas for larger B the impact of larger n was felt more at smaller ϵ. These behaviors are consistent with the $O(n + B^3(\epsilon^{-2} + \log n) \log n)$ running time. For a small B, the linear term dominates and the performance of the algorithm for various ϵ is similar. The second term is becomes more influential when B is large and smaller ϵ affects the running time. For a sufficiently large B, the latter term becomes important when ϵ gets smaller, as is shown in Figure 20(a) and (b).

However, the algorithm easily remained feasible for large n and reasonable B. This performance definitely sets this algorithm aside as the best in class histogram construction algorithm. As we will see later, the error of the algorithm, even on real life data sets, was significantly below the threshold set. Coupled with the fast running time AHIST-L-Δ gives us a truly attractive algorithm for histogram construction problems. The required care in design and implementation of the algorithm definitely pays off in terms of the improved performance.

5.3.7   *Scaling Up AHIST-B: How Important Is M?*.   Figures 21 and 22 show the scale-up behavior for the algorithm AHIST-B. From Figure 21, it is clear that M was less important compared to ϵ at large B. This is shown more effectively
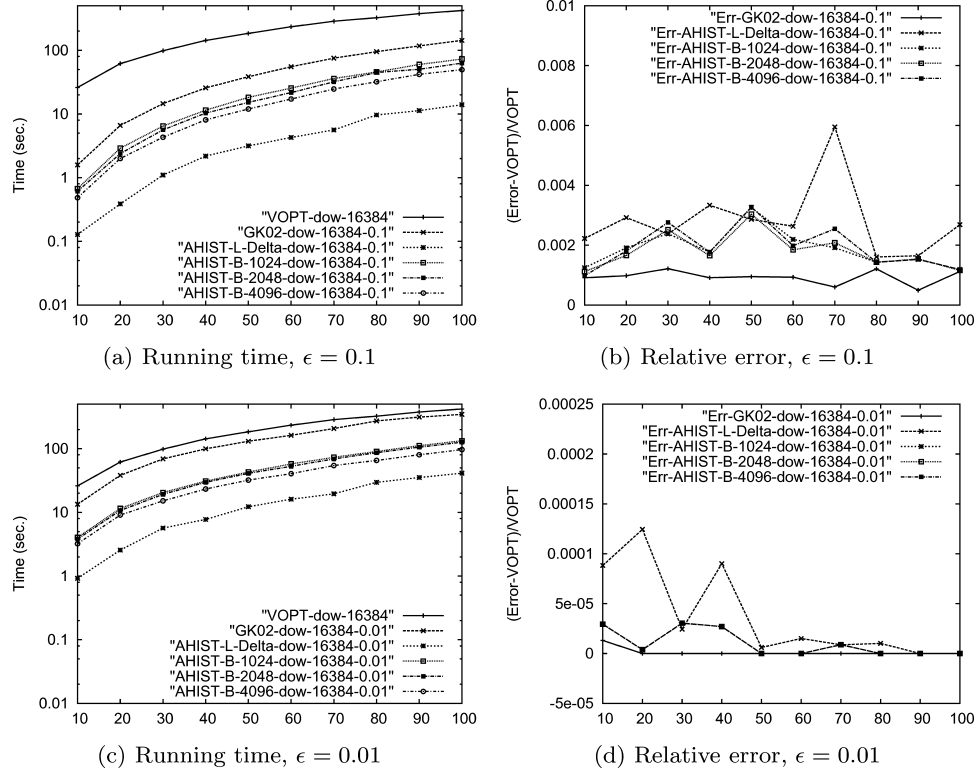
(a) n=32768   (b) n=65536
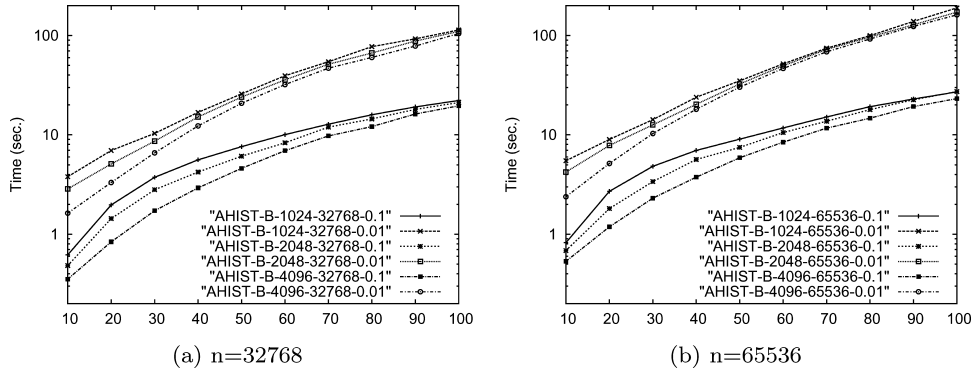
Fig. 21.   Running time of AHIST-B as *B* varies.

(a) For different *B* as ϵ varies   (b) For different ϵ as n varies

Fig. 22.   Running time of AHIST-B as *M* is changed from 256 to 4096.

in Figure 22(a) where for *M* between 256 and 4096 the algorithm performed in a "band" that was determined by *B*, but was less sensitive to ϵ when *B* was small. This is again consistent with the analysis of the time complexity. Figure 22(b) shows that for a fixed *B* and $\epsilon = 0.1$ the parameter *M* did not influence the running time. However, the same figure shows that when ϵ was changed (keeping *B* the same) there was a shift in the entire "band". Note that AHIST-B-4096 performed worse at $\epsilon = 0.2$ compared to AHIST-B-256 at $\epsilon = 0.1$ for sufficiently large *n*.

Once again, the error of these histograms were significantly below threshold (also in real life data sets). The running time of these algorithms, their space bound and streaming nature make them a uniquely attractive candidate for histogram construction algorithms.

5.4 Experimental Results—Real-life Data Sets

5.4.1 *Performance Over the Entire Data as B Varies.*   In Figures 23(a) and (c), we show the running times as *B* varies—the trends are nearly identical to the trends in the synthetic data, specially the Normal datasets.

(a) Running time, $\epsilon = 0.1$

(b) Relative error, $\epsilon = 0.1$

(c) Running time, $\epsilon = 0.01$

(d) Relative error, $\epsilon = 0.01$

Fig. 23.   Characteristics as B varies, $n = 16384$.

5.4.2 *The Error in Approximation.*   Figures 23(b) and (d) shows the error (in terms of a fraction of the error obtained by the VOPT algorithm) for different B. The approximation algorithms are much faster than VOPT and have very small error, specially for $\epsilon = 0.01$. Observe that all approximation algorithms returned answers which are far below the error threshold. Thus, we have a strong case for using the approximation algorithms.

5.4.3 *Running Times as a Function of n for Fixed B.*   Figure 24 shows the running times of the various algorithms for setting of $\epsilon = 0.1, 0.01$. We used the prefixes of the same dataset to get different values of n (which were powers of 2).

5.5 Summary of Trends

From the figures it is immediate:

(1) The approximation algorithms performed well in most of the datasets, including the real life dataset. AHIST-L-Δ was significantly faster than VOPT over almost all scenarios covered. The algorithm AHIST-B also dominated VOPT in most of the scenarios. The algorithm GK02 also performed well, compared to VOPT specially in the real life dataset.
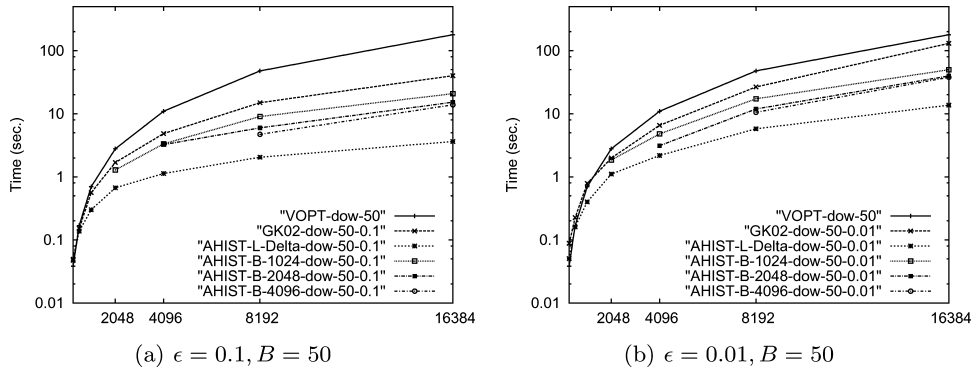
Fig. 24. Running time as n (the prefix size) varies.

(2) AHIST-L-Δ usually had the largest error amongst the approximation algorithms, but the error was usually below the ϵ by at least a factor of 15.

(3) The algorithm AHIST-L-Δ scaled extremely well and AHIST-B scaled well. Both these algorithms performed significantly better than the worst case guarantees (in running time and error).

Based on the trends we can easily conclude that AHIST-L-Δ and AHIST-B are attractive options for histogram construction algorithms. Note that AHIST-B is also a bounded space streaming algorithm, which makes it significantly appealing.

6. CONCLUSIONS

Histogram construction is a problem of central interest to many database applications. A variety of database applications including, approximate querying, similarity searching and data mining, rely on accurate histograms.

Previous histogram construction algorithms that applied to a broad class of error measures required $O(n^2 B)$ time and $O(nB)$ space for finding the optimum histogram. In this article, we gave the first $(1 + \epsilon)$ approximation algorithm for any $\epsilon > 0$ that runs in linear time. We also showed that our technique generalizes to several interesting histogram construction problems, notably using piecewise degree-d polynomials. Our algorithms work in the model where the data items x_i are presented one at a time in an increasing order of i. Thus, for time series applications our algorithms are one-pass stream algorithms.

Finally, we demonstrated the effectiveness of the approximation schemes using synthetic and real life data sets. Since the overall algorithmic technique is the same for different error measures, we reported the performance of approximating V-Optimal histograms. The results for other measures were similar and confirmed that our approximation technique is an important tool for constructing accurate histograms faster.

REFERENCES

ACHARYA, S., GIBBONS, P., POOSALA, V., AND RAMASWAMY, S. 1999. The aqua approximate query answering system. In *Proceedings of the ACM SIGMOD Conference*. ACM, New York, 574–576.

ALON, N., MATIAS, Y., AND SZEGEDY, M. 1999. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci. 58*, 1, 137–147.

BERTOLOTTO, M. AND EGENHOFER, M. J. 1999. Progressive vector transmission. In *Proceedings of the 7th ACM Symposium on Advances in Geographical Information Systems*. ACM, New York, 152–157.

BORODIN, A. AND EL-YANIV, R. 1998. *Online Computation and Competitive Analysis*. Cambridge University Press, Cambridge, MA.

DATAR, M., GIONIS, A., INDYK, P., AND MOTWANI, R. 2002. Maintaining stream statistics over sliding windows. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA 2002)*. ACM, New York, 635–644.

DONJERKOVIC, D., IOANNIDIS, Y. E., AND RAMAKRISHNAN, R. 1999. Dynamic histograms: Capturing evolving data sets. CS-TR 99-1396, University of Wisconsin, Madison, WI. Mar.

FEIGENBAUM, J., KANNAN, S., STRAUSS, M., AND VISWANATHAN, M. 2002. An approximate l1-difference algorithm for massive data streams. *SIAM J. Comput. 32*, 1, 131–151.

GAROFALAKIS, M. N. AND GIBBONS, P. B. 2002. Wavelet synopses with error guarantees. In *Proceedings of the ACM SIGMOD Conference*. ACM, New York, 476–487.

GAROFALAKIS, M. N. AND GIBBONS, P. B. 2004. Probabilistic wavelet synopses. *ACM Trans. Datab. Syst. 29*, 43–90.

GIBBONS, P. AND MATIAS, Y. 1999. Synopsis data structures for massive data sets. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA 1999)*. ACM, New York, 909–910.

GILBERT, A. C., GUHA, S., INDYK, P., KOTIDIS, Y., MUTHUKRISHNAN, S., AND STRAUSS, M. 2002. Fast, small-space algorithms for approximate histogram maintenance. In *Proceedings of the ACM Symposium on Theory of Computing*. ACM, New York, 389–398.

GILBERT, A. C., KOTIDIS, Y., MUTHUKRISHNAN, S., AND STRAUSS, M. 2001. Optimal and approximate computation of summary statistics for range aggregates. In *Proceedings of the ACM Symposium on Principles of Database Systems*. ACM, New York.

GREENWALD, M. AND KHANNA, S. 2001. Space-efficient online computation of quantile summaries. In *Proceedings of the ACM SIGMOD Conference*. ACM, New York.

GUHA, S. 2005. Space efficiency in synopsis construction problems. In *Proceedings of the VLDB Conference*. 409–420.

GUHA, S. AND HARB, B. 2006. Approximation algorithms for wavelet transform coding of data streams. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA 2006)*. ACM, New York.

GUHA, S., INDYK, P., MUTHUKRISHNAN, S., AND STRAUSS, M. 2002a. Histogramming data streams with fast per-item processing. In *Proceedings of the ICALP*. 681–692.

GUHA, S., KOUDAS, N., AND SRIVASTAVA, D. 2002b. Fast algorithms for hierarchical range histogram construction. In *Proceedings of the ACM Symposium on Principles of Database Systems*. ACM, New York, 180–187.

GUHA, S. AND KOUDAS, N. 2002. Approximating a data stream for querying and estimation: Algorithms and performance evaluation. In *Proceedings of the IEEE 18th International Conference on Data Engineering (ICDE'02)*. IEEE Computer. Society Press, Los Alamitos, CA, 567–576.

GUHA, S., KOUDAS, N., AND SHIM, K. 2001. Data streams and histograms. In *Proceedings of the ACM Symposium on Theory of Computing*. ACM, New York, 471–475.

GUHA, S., SHIM, K., AND WOO, J. 2004. REHIST: Relative error histogram construction algorithms. In *Proceedings of the VLDB Conference*, 300–311.

HOCHBAUM, D. 1996. *Approximation Algorithms for NP Hard Problems*. Brooks/Cole Pubishing Company.

INDYK, P. 2000. Stable distributions, pseudorandom generators, embeddings and data stream computation. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*. IEEE Computer Society Press, Los Alamitos, CA, 189–197.

IOANNIDIS, Y. AND POOSALA, V. 1995. Balancing histogram optimality and practicality for query result size estimation. In *Proceedings of the ACM SIGMOD Conference*. ACM, New York, 233–244.

IOANNIDIS, Y. E. 1993. Universality of serial histograms. In *Proceedings of the VLDB Conference*, 256–267.

IOANNIDIS, Y. E.   2003.   The history of histograms (abridged). In *Proceedings of the VLDB Confer-ence*, 19–30.

JAGADISH, H. V., KOUDAS, N., MUTHUKRISHNAN, S., POOSALA, V., SEVCIK, K. C., AND SUEL, T.   1998.   Optimal histograms with quality guarantees. In *Proceedings of the VLDB Conference*, 275–286.

KEOGH, E., CHAKRABATI, K., MEHROTRA, S., AND PAZZANI, M.   2002.   Locally adaptive dimensionality reduction for indexing large time series databases. *ACM Trans. Datab. Syst. 27*, 2, 188–228.

KOOI, R.   1980.   The optimization of queries in relational databases. Case Western Reserve University.

KOUDAS, N., MUTHUKRISHNAN, S., AND SRIVASTAVA, D.   2000.   Optimal histograms for hierarchical range queries. In *Proceedings of the ACM Principles of Database Systems*. ACM, New York, 196–204.

MANKU, G. S., RAJAGOPALAN, S., AND LINDSAY, B.   1998.   Approximate medians and other quantiles in one pass and with limited memory. In *Proceedings of the ACM SIGMOD Conference*. ACM, New York, 426–435.

MATIAS, Y., VITTER, J. S., AND WANG, M.   1998.   Wavelet-based histograms for selectivity estimation. In *Proceedings of the ACM SIGMOD Conference*. ACM, New York, 448–459.

MURALIKRISHNA, M. AND DEWITT, D. J.   1988.   Equi-depth histograms for estimating selectivity factors for multidimensional queries. In *Proceedings of the ACM SIGMOD, Conference*. ACM, New York, 28–36.

MUTHUKRISHNAN, S. AND STRAUSS, M.   2004.   Approximate histogram and wavelet summaries of streaming data. DIMACS TR 52.

POOSALA, V., IOANNIDIS, Y., HAAS, P., AND SHEKITA, E.   1996.   Improved histograms for selectivity estimation of range predicates. In *Proceedings of the ACM SIGMOD Conference*. ACM, New York, 294–305.

SELINGER, P. G., ASTRAHAN, M. M., CHAMBERLIN, D. D., LORIE, R. A., AND PRICE, T. G.   1979.   Access path selection in a relational database management system. In *Proceedings of the ACM SIGMOD Conference*. ACM, New York, 23–34.

THAPER, N., GUHA, S., INDYK, P., AND KOUDAS, N.   2002.   Dynamic multidimensional histograms. In *Proceedings of the SIGMOD Conference*. ACM, New York, 428–439.

VAZIRANI, V.   2001.   *Approximation Algorithms*. Springer-Verlag, New York.