REHIST: Relative Error Histogram Construction Algorithms

Sudipto Guha
University of Pennsylvania
sudipto@cis.upenn.edu

Kyuseok Shim
Seoul National University
shim@ee.snu.ac.kr

Jungchul Woo
Seoul National University
jcwoo@kdd.snu.ac.kr

Abstract

Histograms and Wavelet synopses provide useful tools in query optimization and approximate query answering. Traditional histogram construction algorithms, such as V-Optimal, optimize absolute error measures for which the error in estimating a true value of 10 by 20 has the same effect of estimating a true value of 1000 by 1010. However, several researchers have recently pointed out the drawbacks of such schemes and proposed wavelet based schemes to minimize relative error measures. None of these schemes provide satisfactory guarantees – and we provide evidence that the difficulty may lie in the choice of wavelets as the representation scheme.

In this paper, we consider histogram construction for the known relative error measures. We develop optimal as well as fast approximation algorithms. We provide a comprehensive theoretical analysis and demonstrate the effectiveness of these algorithms in providing significantly more accurate answers through synthetic and real life data sets.

1 Introduction

Motivation and Background: Histograms and Wavelet synopsis provide useful tools in query optimization [13] and approximate query answering [1]. Recently these techniques have also been used in constructing short signatures of time series data for various mining tasks [2]. In all these problems, given a sequence of data values $x_1, \ldots, x_n$, the task is to construct a suitable summary of the data which can be stored in small space (e.g. a small fixed number, say B, of the n coefficients in the Wavelet transform, or a histogram involving B buckets). In the query optimization context, the value x_i is the frequency of i and is non-negative. In the context of time-series data, [2], x_i is the value seen in the stream at time i and may be arbitrary. We make no assumptions about x_i.

Given a query that asks the value at i, a suitable "estimate" of x_i, say $\hat{x}_i$, is constructed and returned as an answer. This is known as a "point query". The error, defined as the *absolute error*, incurred in the process for the point i is $|x_i - \hat{x}_i|$. The objective of good synopsis construction algorithms is to build the summary structure restricted by B that minimizes a suitable function of these errors. Histograms are typically defined to be a piecewise constant representation*, and are constrained to have at most B pieces or "buckets". The popular V-Optimal histogram minimizes the sum of squares of absolute errors (i.e. $\sum_i (x_i - \hat{x}_i)^2$) and was introduced in [12]. Approximation algorithms for the V-Optimal histogram were given in [14, 9, 8, 5, 7].

Authors of [16, 4] rightly point out that, the measures (e.g. sum, sum of squares, etc.) which minimize some function of the absolute errors at the data points are not the most desirable measures†. The drawback of these measures involving the absolute errors is that the error in approximating $x_i = 1000$ by $\hat{x}_i = 1010$ has the same effect of approximating $x_i = 10$ by $\hat{x}_i = 20$. In percentages, the first is a 0.1% error and the latter is a 100% error. Notice, if we multiply all the involved numbers by 100, the disparity in the error remains — the disparity is scale independent and arises because 10 is *relatively small* compared to 1000.

The relative error measures seek to minimize a suit-

Proceedings of the 30th VLDB Conference, Toronto, Canada, 2004

*Quantile summaries are sometimes referred as histograms as well, but we will use histograms to denote piecewise constant representations.

†The discussion here is limited to point queries. For "range queries", which will be out of scope for this paper, the reader is asked to follow the pointers in [15, 6, 10].

able function of $|x_i - \hat{x}_i| / \max\{|x_i|, c\}$, where c is a sanity constant which is used to reduce excessive domination of relative error by small data values. The authors of [16] propose the use of deterministic thresholding to select the wavelet coefficients of the data value. The work in [4] introduces probabilistic thresholding and provides the first theoretical guarantees on the quality of the approximation achieved by the constructed wavelet synopsis. However their proposed solution using wavelets suffers the following drawbacks:

- **Difficulty of optimizing relative error measures.** It is observed in [4] that wavelets are not easily amenable for minimizing functions other than L_2 norms. The fundamental problem is that a change of any wavelet coefficient affects more than a single data value. In fact, in [4], convex programming (not known to be solvable in polynomial time) was used. To simplify, the authors of [4] restricted the problem to select coefficients from the wavelet representation of the data. However if the coefficients are selected from the wavelet representation, the relative error measures do not monotonically decrease as the number of selected coefficients increase.

  Consider a simple example $X = [4, 3, 2, 1]$ whose Haar wavelet transform with normalization is $[2.5, 1, 0.35, 0.35]$. If we do not choose any coefficient to store, the zero vector has 100% error for the maximum relative error (considered in [4]). However, choosing any single coefficient to store gives an error larger than 100% relative error. Interestingly, if we replace 2.5 by 1.6 and use the corresponding wavelet vector, the maximum relative error becomes 60%.

- **Expectation guarantees.** In [16], the authors do not provide any error guarantee for relative error measures with their scheme. In [4], they give expected guarantees on *both* the error and the space used. For example, if we have a solution with error 9 and space 1 (in some unit) and a second solution with error 1 and space 9; then choosing a solution with the same probability 0.5 for each solution will give a solution with expected space and expected error 5. Thus although in expectation we have a space bound of 5 and error bound 5, if we choose any one of the two solutions we will have to settle for twice the expected space or twice the expected error *irrespective* of the number of times we repeat the experiment. Expectation guarantees would have been useful if *any one* of the parameters were obeyed strictly. In fact preliminary calculations show that the variance of the space requirement of algorithms in [4] can be as large as $\Theta(B)$ implying that the space bound is likely to be overshot significantly.

The above issues clearly demonstrate the need for algorithms that minimize relative error measures optimally. Motivated by the above we consider histogram representations instead of wavelet synopses. Many researchers have the following in support of histograms: (1) Any (reasonable) error measure does not increase as the number of buckets increase (2) Histograms consider arbitrary intervals, compared to fixed wavelet boundaries. Thus, histograms offer a richer class of representations and often allow a better representation of the data[‡].

In this paper we propose histogram construction algorithms under relative error measures. We seek to design deterministic optimal and approximation algorithms that obey the space bound strictly. We can summarize our contributions as follows:

- **Optimal algorithms for relative error measures.** We provide the first optimal histogram construction algorithms for minimizing the relative error measures. For minimizing the maximum relative relative error, we give a deterministic algorithm in time required by the complicated approximation strategy in [4]. For the other measures, the running time is similar to the time required for computing the V-Optimal histogram.

- **Histogram construction for data streams.** We also provide truly linear ($O(n)$, with no hidden constants like $B, \frac{1}{\epsilon}$ in the O) approximation algorithms for most of the error measures. The linear algorithms have the special property that they examine the data only once in a left-to-right order and require small memory. Thus, they are suitable for constructing histograms for *streaming time series data* as well. For the rest, the approximation runs in time $O(n \log n)$, with no hidden terms.

- **Extensive experimentation validating our algorithms.** We demonstrate the effectiveness of our histogram construction algorithms in providing highly accurate answers using synthetic and real-life data sets. *We use the real-life data sets used in the previous work.* We compare our algorithms with those proposed in [16, 4] in both quality of answers and execution time.

The paper is organized as follows. In the next section, we present preliminary definitions and formally introduce the problem of constructing histograms with relative error measures. In Section 3 and Section 4, we introduce optimal and approximation algorithms respectively. Section 5 presents experimental results, and finally, we summarize in Section 6. Due to the lack

[‡]It can be shown that a B bucket wavelet synopsis can be represented *exactly* by $3B+1$ bucket histograms, but a B bucket histogram may require $O(B \log n)$ wavelet coefficients.

of space, we omit the proofs of lemmas and theorems presented in the paper. They can be found in [11].

2 Preliminaries & Related Work

2.1 Problem Statement

Let $X = x_1, \ldots, x_n$ be a finite data sequence. The general problem of histogram construction is as follows: given some space constraint B, create and store a compact representation H_B of the data sequence. H_B uses at most B storage and is optimal under some notion of error.

The representation collapses the values in a sequence of consecutive points x_i where $i \in [s_r, e_r]$ (say $s_r \le i \le e_r$) into a single value $\hat{x}(r)$, thus forming a bucket b_r, that is, $b_r = (s_r, e_r, \hat{x}(r))$. The histogram H_B is used to answer queries about the value at point i where $1 \le i \le n$. 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.

For a point query, the histogram is used to estimate the x_i, and for $s_r \le i \le e_r$, the estimate is $\hat{x}(r)$. Since $\hat{x}(r)$ is an estimate for the values in bucket b_r, we suffer an error.

Definition 2.1 The absolute and the relative errors for a point $i \in [s_r, e_r]$ are respectively defined as

$$|\hat{x}(r) - x_i| \quad \text{and} \quad |\hat{x}(r) - x_i| / \max\{c, |x_i|\}.$$

The error of the histogram H_B can be defined as a function of these point errors. The most popular ways are: (1) sum of errors at every point i, (2) sum of squared errors at every point i, or (3) maximum error considering every point i. Each of the choices induces a natural notion of error for a bucket. We introduce the following definitions:

Definition 2.2 Given an interval $[s_r, e_r]$, we define

- $\textsc{SSQError}(s_r, e_r) = \min_{\hat{x}(r)} \sum_{i=s_r}^{e_r} (x_i - \hat{x}(r))^2$
- $\textsc{Err}_{\mathrm{M}}(s_r, e_r) = \min_{\hat{x}(r)} \max_{i \in [s_r, e_r]} \frac{|x_i - \hat{x}(r)|}{\max\{c, |x_i|\}}$
- $\textsc{Err}_{\mathrm{sq}}(s_r, e_r) = \min_{\hat{x}(r)} \sum_{i=s_r}^{e_r} \frac{(x_i - \hat{x}(r))^2}{\max\{c^2, x_i^2\}}$
- $\textsc{Err}_{\mathrm{S}}(s_r, e_r) = \min_{\hat{x}(r)} \sum_{i=s_r}^{e_r} \frac{|x_i - \hat{x}(r)|}{\max\{c, |x_i|\}}$

Definition 2.3 Let $\textsc{Terr}[j, k]$ be the error of the best k bucket histogram representation for $x_1, \ldots, x_j$ under the sum of squared absolute error measure. Thus the optimum histogram construction problem under this measure is to find the histogram for $\textsc{Terr}[n, B]$.

Similarly, we define $\textsc{Trerr}_{\mathrm{M}}[j, k]$, $\textsc{Trerr}_{\mathrm{sq}}[j, k]$ and $\textsc{Trerr}_{\mathrm{S}}[j, k]$ as the errors of the best k bucket histogram representation for $x_1, \ldots, x_j$ under the maximum relative error, sum of squared relative error, and sum of relative error respectively.

There exists an efficient algorithm to determine the V-Optimal histogram given by Jagadish *et. al.*, [14], which requires time $O(n^2 B)$ and $O(Bn)$ space. We will review their algorithm in next.

2.2 V-Optimal Histogram Construction

Two important contributions are made in [14]. First, the value of $\hat{x}(r)$ for a bucket $[s_r, e_r]$ which achieves the minimum error, $\textsc{SSQError}(s_r, e_r)$, is the mean of the values $x_{s_r}, \ldots, x_{e_r}$. Thus, we have

$$\textsc{SSQError}(i, j) = \sum_{\ell=i}^{j} x_\ell^2 - \frac{1}{j - i + 1} \left(\sum_{\ell=i}^{j} x_\ell \right)^2 \quad (1)$$

Second, if the last bucket in the optimal histogram contains the data points denoted by the interval $[i + 1, n]$, then the rest of the buckets must form an optimal histogram with $(B - 1)$ buckets for the interval $[1, i]$[§]. The best i will minimize the sum of the two parts.

Generalizing the discussion (from B to an arbitrary number of buckets $k > 2$, and from n to an arbitrary point $j > 1$), the following dynamic programming algorithm arises immediately:

$$\textsc{Terr}[j, k] = \min_{1 \le i < j} \{\textsc{Terr}[i, k-1] + \textsc{SSQError}(i+1, j)\} \quad (2)$$

To compute $\textsc{SSQError}(i + 1, n)$ in $O(1)$ time, two arrays $\textsc{Sum}$ and $\textsc{Sqsum}$ are maintained,

$$\textsc{Sum}[1, i] = \sum_{\ell=1}^{i} x_\ell \qquad \textsc{Sqsum}[1, i] = \sum_{\ell=1}^{i} x_\ell^2 \quad (3)$$

The sums in Equation (1) can be replaced by $\textsc{Sum}[1, j] - \textsc{Sum}[1, i-1]$ and $\textsc{Sqsum}[1, j] - \textsc{Sqsum}[1, i-1]$ respectively. In Equation (2), the value of j can have at most n values. Since each entry of Terr[j,k] requires $O(n)$ time, with $O(nB)$ entries, we can establish that the complexity of the optimal histogram construction is $O(n^2 B)$.

3 Optimal Histograms under Relative Error

In this section, we focus on developing optimum algorithms for the relative error measures. We first show that the optimum histogram under the maximum relative error criterion can be constructed in $O(nB \log^2 n)$ time. Note that the algorithm in [4] which returns an approximate wavelet decomposition has a similar running time. We subsequently investigate the sum of relative error measure and finally the sum of squares of relative error measure.

[§]Otherwise, it is easy to observe that the error of the optimal solution can be decreased by taking the optimal histogram with $(B - 1)$ buckets and the last bucket defined on the points belonging to $[i + 1, n]$.

Case	Representative	Error
$\max \geq \min \geq c$	$\frac{(2*\max*\min)}{(\max+\min)}$	$\frac{\max-\min}{\max+\min}$
$\min \leq \max \leq -c$	$\frac{(2*\max*\min)}{(\max+\min)}$	$\frac{\min-\max}{\max+\min}$
$-c < \min < c \leq \max$	$\frac{\max(\min+c)}{\max+c}$	$\frac{\max-\min}{\max+c}$
$\min \leq -c \leq \max \leq c$	$\frac{\min(c-\max)}{c-\min}$	$\frac{\max-\min}{c-\min}$
$-c \leq \min \leq \max \leq c$	$\frac{(\max+\min)}{2}$	$\frac{\max-\min}{2c}$
$\min \leq -c < c \leq \max$	0	1.0

Table 1: Optimal Maximum Relative Error

Procedure NaiveHistErr_M()
begin
1. **for** j := 1 **to** n **do** {
2. $\text{Trerr}_\text{M}[j,1] := \text{Err}_\text{M}(1,j)$
3. **for** k := 2 **to** B **do** {
4. max := $-\infty$; min := ∞; $\text{Trerr}_\text{M}[i,k] := \infty$
5. **for** i := j - 1 **down to** 1 **do** {
6. If (max $< x_{i+1}$) then max := x_{i+1}
7. If (min $> x_{i+1}$) then min := x_{i+1}
8. $\text{Trerr}_\text{M}[j,k] := \min(\text{Trerr}_\text{M}[j,k], \max($
$\text{Trerr}_\text{M}[i,k-1], \text{Err}_\text{M}(i+1,j))$
9. }
10. }
11. }
end

Figure 1: The NaiveHistErr_M

3.1 Maximum Relative Error

Recall that the maximum relative error of a bucket $b_r = (s_r, e_r, \hat{x})$ is defined as follows:

$$\text{Err}_\text{M}(s_r, e_r) = \min_{\hat{x}} \max_{i \in [s_r, e_r]} \frac{|x_i - \hat{x}|}{\max\{c, |x_i|\}}$$

We can prove the following:

Lemma 3.1 *Given a set of numbers $x_1, \ldots, x_\ell$, the maximum relative error generated by minimizing maximum relative errors is defined by the minimum and the maximum over these x_i as described in Table 1.*

A Simple Algorithm: Lemma 3.1 allows us to consider a naive optimal algorithm, NaiveHistErr_M, with $O(n^2B)$ running time. The NaiveHistErr_M is illustrated in Figure 1. The computation of Err_M can be performed in time $O(1)$ if we are maintaining the running minimum and the maximum incrementally over the interval $[i+1, j]$ as i changes (decreases).

An Improved $\mathbf{O}(nB\log^2 n)$ Construction: Consider line (8) of Figure 1, where $\text{Trerr}_\text{M}[j,k]$ is

$$\min_i\{\max(\text{Trerr}_\text{M}[i,k-1], \text{Err}_\text{M}(i+1,j))\}$$

Observe that $\text{Trerr}_\text{M}[i,k-1]$ is an increasing function and Err_M is a decreasing function of i. This problem can be solved faster.

Lemma 3.2 *To compute $\min_x\{\max(F(x), G(x))\}$ where $F(x)$ and $G(x)$ are non-decreasing and non-increasing functions respectively, we can perform binary search for the value of x such that $F(x) > G(x)$ and $F(x-1) < G(x-1)$, and we can take minimum of $G(x-1)$ and $F(x)$.*

By the above Lemma, we find the smallest i such that $\text{Trerr}_\text{M}[i,k-1] \geq \text{Err}_\text{M}(i+1,j)$. Let the value of i found by the binary search be i^*. The minimum is achieved at either $i = i^* - 1$ or $i = i^*$. This algorithm, OptHistErr_M, is presented in Figure 2. We can verify that the running time of the algorithm is $O(Bn\log n)$ times of the time to compute $\text{Err}_\text{M}(i,j)$.

Procedure OptHistErr_M()
begin
1. **for** j := 1 **to** n **do** {
2. $\text{Trerr}_\text{M}[j,1] := \text{Err}_\text{M}(1,j)$
3. **for** k := 2 **to** B **do** {
4. $low := 1$; $high := j-1$; $\text{Trerr}_\text{M}[j,k] := \infty$
5. **while** ($low < high$) **do** {
6. $mid := (high + low + 1)/2$
7. **if** $\text{Trerr}_\text{M}[mid, k-1] \geq \text{Err}_\text{M}(mid+1, j)$
8. $low := mid$
9. **else** $high := mid - 1$
10. }
11. **if** ($\text{Trerr}_\text{M}[low, k-1] < \text{Err}_\text{M}(low, j)$)
12. $\text{Trerr}_\text{M}[j,k] :=$
$\min(\text{Trerr}_\text{M}[j,k], \text{Err}_\text{M}(low,j))$
13. **else** $\text{Trerr}_\text{M}[j,k] :=$
$\min(\text{Trerr}_\text{M}[j,k], \text{Trerr}_\text{M}[low, k-1])$
14. }
15. }
end

Figure 2: The OptHistErr_M

To compute $\text{Err}_\text{M}(i,j)$ efficiently, we use an interval tree defined below:

Definition 3.3 Given an interval on $[1,n]$ we construct an interval tree which is a binary tree over subintervals of $[1,n]$. The root of the tree corresponds to the entire interval $[1,n]$ and the leaf nodes corresponds to the intervals of length one, e.g. $[i,i]$. For the interval $[i,j]$ of a node in the interval tree, we store the minimum and the maximum of $x_i, \ldots, x_j$. The children of a node with the interval $[i,j]$ correspond to the two (near) half-size intervals $[i, r-1]$, $[r,j]$ where $r = \lfloor \frac{i+j+1}{2} \rfloor$.

It is easy to observe that an interval tree can be constructed in $O(n)$ time and will require $O(n)$ storage. Given an arbitrary interval $[i,j]$, we partition $[i,j]$ into $O(\log n)$ intervals such that each of the resulting subintervals belong to the interval tree. Using the decomposed subintervals, we find the optimal maximum relative for the bucket. It reduces the time complexity of computing the minimum (or maximum) to $O(\log n)$.

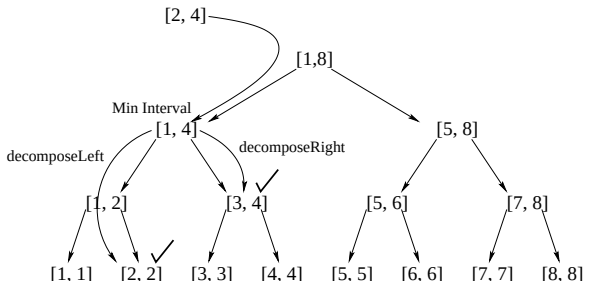


Figure 3: The Steps of Decomposing [2, 4] with an Interval Tree

Decomposing $[i, j]$ into subintervals For an interval $[i, j]$, we traverse the interval tree starting from the root node to find the smallest interval $[i_m, j_m]$ that contains $[i, j]$ as a subinterval. If $[i_m, j_m]$ is exactly $[i, j]$, we are done. Otherwise, let $[i_m, m_m]$ and $[m_m + 1, j_m]$ be the left and right children of $[i_m, j_m]$.

It must be that $i \leq m_m < j$, otherwise there exists an interval smaller than $[i_m, j_m]$ that contains $[i, j]$ entirely. We will recursively find a set of intervals partitioning $[i, m_m]$ and similarly $[m_m + 1, j]$.

Let us focus on $[i, m_m]$, since the other case is symmetrical. We term this interval as "active", in the sense that we seek to partition it. We start from the root of a subtree corresponding to $[i_m, m_m]$. If $[i_m, m_m]$ is the same as $[i, m_m]$ we are done. If $i \neq i_m$ then the interval $[i_m, m_m]$ has length at least 2 and let $[i_m, i'], [i' + 1, m_m]$ be its children. If $i \leq i'$, we add the interval $[i' + 1, m_m]$ to our set and set $[i, i']$ as an active interval and move to the left subtree. Otherwise if $i > i'$ the same interval remains active but we move to the right subtree. We traverse down the tree adding at most one interval in each downward move. Thus, the subintervals found for both $[i, m_m]$ and $[m_m + 1, j]$ can be at most $2 \log n$ and this decomposition takes $O(\log n)$ time.

Example 3.4 We illustrate the decomposition of $[2, 4]$ for a given interval tree in Figure 3. The interval $[2, 4]$ is decomposed into $[2, 2]$ and $[3, 4]$. ∎

Lemma 3.5 *The height of an interval tree corresponding to $[1, n]$ is at most $O(\log n)$. For any interval $[i, j]$, we can express the interval as a set of at most $O(\log n)$ non-overlapping intervals belonging to the nodes in the interval tree. This decomposition can be performed in $O(\log n)$ time. Furthermore, we can compute the maximum and the minimum over $x_i, \ldots, x_j$ in $O(\log n)$ time.*

Therefore, we conclude with the following:

Theorem 3.6 *The optimum maximum relative error for n values and B buckets can be found in time $O(Bn \log^2 n)$ and space $O(Bn)$. The $O(Bn)$ space is required to output the representative values and bucket boundaries after the optimum error has been found, the error itself can be computed in $O(n)$ space.*

3.2 Sum of Squared Relative Errors

Recall that the sum of squared relative error, ERR_{sq}, for the bucket $b_r = (s_r, e_r, \hat{x}(r))$ is defined as follows:

$$\text{ERR}_{\text{sq}}(s_r, e_r) = \min_{\hat{x}(r)} \left(\sum_{j=s_r}^{e_r} \frac{(\hat{x}(r) - x_j)^2}{\max\{c^2, x_j^2\}} \right)$$

The right hand side of the above equation can be rewritten as $\min_{\hat{x}(r)} \left(A\hat{x}(r)^2 - 2B\hat{x}(r) + C \right)$ where if we denote by $w_j = \max\{c^2, x_j^2\}$,

$$A = \sum_{j=s_r}^{e_r} \frac{1}{w_j} \quad \& \quad B = \sum_{j=s_r}^{e_r} \frac{x_j}{w_j} \quad \& \quad C = \sum_{j=s_r}^{e_r} \frac{x_j^2}{w_j}$$

Since $A > 0$, the value of $\text{ERR}_{\text{sq}}(\hat{x})$ is minimized when $\hat{x}$ is B/A, and its minimum value becomes $C - B^2/A$. The aggregated sum values of A, B and C as are stored in the arrays ASUM, BSUM and CSUM respectively to allow computing $\text{ERR}_{\text{sq}}(i+1, j)$ in $O(1)$ time. Thus, using the following recursive definition for $k > 1$,

$$\text{TRERR}_{\text{sq}}[j, k] = \min_i (\text{TRERR}_{\text{sq}}[i, k-1] + \text{ERR}_{\text{sq}}(i+1, j))$$

we can prove the following:

Theorem 3.7 *In $O(Bn^2)$ time, we can compute the optimal histogram under sum of squared relative error measure.*

3.3 Sum of Relative Errors

The sum of relative errors ERR_{S} for a bucket $b_r = (s_r, e_r, \hat{x}(r))$ is defined as follows:

$$\text{ERR}_{\text{S}}(s_r, e_r) = \min_{\hat{x}(r)} \sum_{i=s_r}^{e_r} \frac{|x_i - \hat{x}(r)|}{\max\{c, |x_i|\}} = \min_{\hat{x}(r)} g(\hat{x}(r))$$

Definition 3.8 Given $V = \{x_{s_r}, x_{s_r+1}, ..., x_{e_r}\}$, and $m = e_r - s_r + 1$. Let $V_s = \{v_1, v_2, ...v_m\}$ denote the elements of V in the sorted order. If $v_i = v_j$ then if $i < j$ we say that v_i is before v_j in the order.

To simplify the notation, since we are in the context of a particular bucket b_r we will use $\hat{x}$ to represent $\hat{x}(r)$. Using the above equation, we rewrite $g(\hat{x}) = P(\hat{x}) \cdot \hat{x} + Q(\hat{x})$.

$$P(\hat{x}) = 2 \sum_{j: v_j \leq \hat{x}} \frac{1}{max\{c, |v_j|\}} - \sum_{j=1}^{m} \frac{1}{max\{c, |v_j|\}} \quad (4)$$

$$Q(\hat{x}) = \sum_{j=1}^{m} \frac{v_j}{max\{c, |v_j|\}} - 2 \sum_{j: v_j \leq \hat{x}} \frac{v_j}{max\{c, |v_j|\}} \quad (5)$$

We can show that the minimum of ERR_S is obtained at $v_k \in V_s$ such that k is the least index satisfying $P(v_k) \geq 0$, which is Theorem 3.15. The proof is involved and we provide a road map omitting the proofs of the lemmas.

Lemma 3.9 *The values of $P(\hat{x})$ and $Q(\hat{x})$ do not change with $\hat{x}$ for $v_k \leq \hat{x} < v_{k+1}$.*

Lemma 3.10 *The function $g(\hat{x})$ is continuous at every $v_j \in V_s$.*

Lemma 3.11 *When $P(v_k) < 0$ for $v_k \in V_s$, $g(\hat{x})$ is a decreasing function and we have $g(v_{k+1}) \leq g(\hat{x}) \leq g(v_k)$ for $\hat{x}$ such that $v_k \leq \hat{x} \leq v_{k+1}$.*

If $P(v_k) > 0$, $g(\hat{x})$ is an increasing function and we have $g(v_k) \leq g(\hat{x}) \leq g(v_{k+1})$ for $\hat{x}$ with $v_k \leq \hat{x} \leq v_{k+1}$.

Lemma 3.12 *For $\hat{x} \in (-\infty, v_1]$, we have $P(\hat{x}) < 0$ and $g(v_1) \leq g(\hat{x})$. Furthermore, for $\hat{x} \in [v_m, \infty)$, $P(\hat{x}) > 0$ and $g(v_m) \leq g(\hat{x})$ holds.*

Lemma 3.13 *$P(v_i)$ is an increasing function of i and $P(v_m) > 0$ holds.*

Lemma 3.14 *If $P(v_k) < 0$ for $v_k \in V_s$, the function $g(\hat{x})$ is minimum at $\hat{x} = v_{k+1}$ for $\hat{x} \in (-\infty, v_{k+1}]$. When $P(v_k) > 0$ for $v_k \in V_s$, the function $g(\hat{x})$ is minimum when $\hat{x} = v_k$ for $\hat{x} \in [v_k, \infty)$.*

Theorem 3.15 *The minimum of $g(\hat{x})$ is achieved at $v_k \in V_s$ such that k is the least index satisfying $P(v_k) \geq 0$.*

Optimal Histogram Algorithm: The algorithm presented in Figure 4 gives an optimal histogram for ERR_S in $O(n^2(B + \log n))$ time. The binary tree T_B in the algorithm help us to maintain the sorted order in $O(\log n)$ time ¶.

One strategy would be to obtain the sorted ordered set V_s of $V = \{x_{i+1}, \ldots, x_j\}$, and find v_k satisfying Theorem 3.15. This would result in an $O(n^3)$ algorithm. We can improve the complexity by storing additional information in the binary tree to find v_k faster and computing $\text{ERR}_S(i, j)$ in $O(\log n)$ time. With every node t with key x_t in the tree, we keep a multiplicity field, d_t, which denotes the number of x-values seen which are equal to x_t. Let the subtree rooted at t is T. Each node is also augmented with two other fields m and vm to keep the following values:

$$m = \sum_{x \in T} \frac{1}{\max\{c, |x|\}} \quad \text{and} \quad vm = \sum_{x \in T} \frac{x}{\max\{c, |x|\}}$$

As we insert a new key x to the binary tree, the values in m and vm of the nodes in the binary search tree must be updated appropriately.

¶Under balanced tree implementations, we use AVL-trees.

Procedure OptHistErr$_S$()
begin
1. **for** i := 1 **to** n **do** {
2. $\quad$ TrErr$_S[i, 1]$:= Err$_S(1, i)$
3. $\quad$ **for** k := 2 **to** B **do**
4. $\quad\quad$ TrErr$_S[i, k]$:= ∞
5. $\quad T_B := \emptyset$
6. $\quad$ **for** j := i-1 **to** 1 **do** {
7. $\quad\quad$ // T_B contains values in $[j + 2, i]$
8. $\quad\quad$ insert(T_B, x_{j+1})
9. $\quad\quad$ // T_B now contains values in $[j + 1, i]$
10. $\quad\quad$ Compute Err$_S(j + 1, i)$ using T_B
11. $\quad\quad$ **for** k := 2 **to** B **do** {
12. $\quad\quad\quad$ TrErr$_S[i, k]$:=min{TrErr$_S[i, k]$, TrErr$_S[j, k - 1]$ + Err$_S(j + 1, i)$}
13. $\quad\quad$ }
14. $\quad$ }
15. }
end

Figure 4: The OptHistErr$_S$

Computation of $\text{ERR}_S(i, j)$**:** We will traverse down a path leading to the appropriate v_k giving the solution to ERR_S. Suppose we are at a node t with value x_t, and its nearest ancestor node t_n having $P(x_{t_n}) > 0$. Let the subtree rooted at t is T. We maintain the following quantities while traversing T_B:

$$\text{Ex}.m = \sum_{x \notin T, x < x_t} \frac{1}{\max\{c, |x|\}} \quad \& \quad \text{Ex}.vm = \sum_{x \notin T, x < x_t} \frac{x}{\max\{c, |x|\}}$$

Ex has information regarding the keys in the nodes that are less than x_t and do not belong to the subtree T rooted at t. Let $P_0 = T_B.m$ and $Q_0 = T_B.vm$. We first compute

$$P^*(x_t) = 2\text{Ex}.m + 2\text{Left}(T).m - P_0$$

where Left(T) is the left subtree of t in T. Note that Left$(T).m = 0$ if the left subtree is empty. Since the formula of $P^*(x_t)$ is the same as that of $P(x_t)$ except the contribution from $v_j = x_t$, we always have $P^*(x_t) < P(x_t)$. In each node t, we perform the following steps according to the condition below:

When $P^*(x_t) + d_t / \max\{c, |x_t|\} < 0$, we know that v_k cannot be x_t (or any of its duplicates) and we need to investigate the right subtree of T. In the recursive call to the right subtree, we pass the current t_n if we have t_n in this traversal. We update Ex.m and Ex.vm by adding $t.m$ and $t.vm$ respectively.

If $P^*(x_t) + d_t / \max\{c, |x_t|\} \geq 0$ and $P^*(x_t) < 0$, we have $v_k = x_t$(the condition $P() > 0$ is satisfied by some duplicate of x_t). In this case we are done.

Otherwise, we have $P^*(x_t) \geq 0$ and the value of x_t may be the solution, but we cannot be assured of it without inspecting the rest of the subtree. We replace t_n as t and investigate the left subtree Left(T).

By this process we recursively maintain the invariant that: *the solution v_k is contained in the current subtree or is the stored value in t_n.* If we reach a leaf

node and decide to take the (empty) right path we know that the stored value in t_n must be v_k. If we attempt to take the empty left path then the stored value of the leaf node must be v_k.

To get ERR_S, we simply calculate

$$Q^*(x_t) = Q_0 - 2\text{EX}.vm - 2\text{LEFT}(T).vm$$

and compute $P^*(x_t)x_t + Q^*(x_t)$ which is actually $P(x_t)x_t + Q(x_t)$. Thus, we can claim the following:

Lemma 3.16 *We can compute v_k, equivalently $\text{ERR}_\text{S}(i,j)$ given an AVL-tree, in time $O(\log n)$.*

Thus, we conclude with the following Theorem.

Theorem 3.17 *We can compute the optimal histogram under sum of relative error measure in time $O(n^2(B + \log n))$.*

4 Approximate Histograms for Relative Error

Due to the lack of space we relegate the discussion on previous work on approximate V-Optimal histograms to [11]. We point out why newer approximation algorithms are needed for relative error measures.

The approximation algorithm in [14] gives a factor 3 approximation (ignoring other issues) and we will be interested in $(1+\epsilon)$ approximations. The approximate algorithms in [9, 8] depend on the fact that the error for a bucket in the V-Optimal histogram depends on the mean and the sum of the values in the bucket. However, the sum of relative error ERR_S in a bucket depends on *the sorted order* of the values in the bucket and not on simple aggregate values. Thus, the algorithms in the style of [8, 9] cannot work for ERR_S without substantial modification. The algorithms in [5, 7] use properties of the L_2 norm and are too dependent on the definition of V-Optimal error to be useful in context of relative error.

For maximum relative error ERR_M, we need to maintain the maximum and minimum values in a bucket to compute the error. An algorithm similar to [9] can give a $(1+\epsilon)$ approximation — but unfortunately the running time of such an algorithm will be $O(nB^3\epsilon^{-2}\log n)$ which is greater than the time required by the optimal algorithm! An algorithm like [8] cannot maintain the minimum or maximum for an arbitrary interval since the lazy evaluation strategy does not inspect all the elements (which is necessary to find the maximum and minimum).

Based on the above facts, we need to design new approximation algorithms. We give a $(1+\epsilon)$-approximation algorithm for the maximum relative error requiring $O(n)$ time and $O(B^2\epsilon^{-1}\log n\,(\log\log n + \log\frac{B}{\epsilon})^3)$ space in Section 4.1. Subsequently we provide the a $(1+\epsilon)$ approximation for sum of squared relative error in $O(n)$ time but $O(B^3\epsilon^{-2}\log^2 n(\log\log n + \log\frac{B}{\epsilon}))$ space. These results extend naturally to the streaming context, and are directly applicable for time series data.

For sum of relative error we give a $O(n\log n)$ time and space algorithm in Section 4.3.

4.1 ERR_M: Maximum Relative Error

The main idea behind approximate histograms in [8, 9] is the observation that an increasing function can be approximated by a step function. We will read the input in blocks of which each contains M data values.

Let δ be $\epsilon/(2B)$. We approximate the function $\text{TRERR}_\text{M}[i,k+1]$ at a subset of the points by the function $\text{APXERR}_\text{M}[i,k+1]$. We let $\text{APXERR}_\text{M}[i,1] = \text{ERR}_\text{M}(1,i)$ and $\text{APXERR}_\text{M}[i,k]$ is defined for $k>1$ as:

$$\max_l\{\text{APXERR}_\text{M}[e_l^k, k-1], \text{ERR}_\text{M}(e_l^k+1, i)\}$$

We will not evaluate $\text{APXERR}_\text{M}[i,k]$ for all i, and retain the values of $\text{APXERR}_\text{M}[i,k]$ at the endpoints of several intervals $[s_l^k, e_l^k]$ only such that:

$$(1+\delta)\text{APXERR}_\text{M}[s_l^k, k] \;\geq\; \text{APXERR}_\text{M}[e_l^k, k] \qquad (6)$$

with the property that the intervals $[s_l^k, e_l^k]$ are mutually disjoint for the same k and $\bigcup_l [s_l^k, e_l^k]$ is all the input we have seen so far.

Suppose we have stored a set of values $\text{APXERR}_\text{M}[i,k]$ with $1 \leq i \leq n-M$, as shown in figure 5. The dotted line corresponds to the approximate intervals stored for the interval $[1, n-M]$, and corresponds to $\text{APXERR}_\text{M}[i,k]$ (estimated or approximated $\text{TRERR}_\text{M}[i,k]$). The solid line corresponds to $\text{APXERR}_\text{M}[i,k]$ for the last M values — note that we will not even compute $\text{APXERR}_\text{M}[i,k]$ for all these points. But if we did, we would get the solid line.

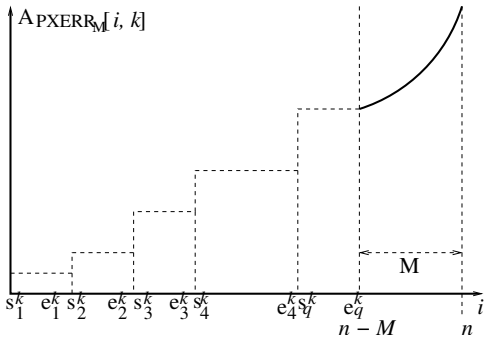


Figure 5: Extending $\text{APXERR}_\text{M}[i,k]$

We have B lists of intervals in which the k-th list stores the set of intervals corresponding to Equation (6). Intuitively, we approximate $\text{TRERR}_\text{M}[i,k]$ with $1 \leq i \leq n-M$. We will extend the B interval lists to include the approximation of $\text{TRERR}_\text{M}[i,k]$ for the last M data values. Assume that $[s_q^k, e_q^k]$ is the last interval of $[1, n-M]$ for $\text{APXERR}_\text{M}[i,k]$.

We find the *maximum* for $n - M + 1 \leq e \leq n$ such that $\text{APXERR}_\text{M}[e,k] \leq (1+\delta)\text{APXERR}_\text{M}[s_q^k, k]$. This e defines an interval $[s_q^k, e]$ that replaces the interval $[s_q^k, e_q^k]$ in k-th interval list. We start a new interval from $e+1$ and repeat this process to find next intervals until we run out of the last M data values. Notice we can use binary search to find e, and subsequently next e etc. To extend the k-th list by one element of APXERR_M, we require binary search with $O(\log M)$ time. Suppose n_q is the maximum number of elements in any list. The number of evaluating $\text{APXERR}_\text{M}[i,k]$ to extend k-th list is $O(n_q * \log M)$. Thus, the time complexity of extending k-th list becomes $O(n_q * \log M)$ times that of evaluating $\text{APXERR}_\text{M}[i,k]$.

Extending k-th List: Assume that we processed r blocks of data values whose interval is $[1, n-M]$ and we are about to process the $(r+1)$-th block whose size is M. With each end-point e_l^k of intervals in $[1, n-M]$, we maintain the minimum and the maximum values seen in $[e_l^k,\ e_q^k]$ where e_q^k is the end of the last (i.e. r-th) block. We will update these values and they will help us calculate $\text{ERR}_\text{M}(e+1, n)$. We also build an interval tree with $O(M)$ time which is introduced in Section 3.1. We describe the algorithm at a high level only, the detailed description of the algorithm can be found in [11]. We set

$$\text{APXERR}_\text{M}[i,1] = \text{TRERR}_\text{M}[i,1] = \text{ERR}_\text{M}(1,i)$$

Now, $\text{APXERR}_\text{M}[i,k]$ for $k > 1$ is:

$$\min_l\{\max(\text{APXERR}_\text{M}[e_l^{k-1}, k-1], \text{ERR}_\text{M}(e_l^{k-1}+1, i))\}$$

where e_l^{k-1} are end points of the $(k-1)$-th list. By Lemma 3.2, computation of $\text{APXERR}_\text{M}[i,k]$ can be done with binary search in $O(\log n_q)$ time.

In each step of this binary search, $\text{APXERR}_\text{M}[e_l^{k-1}, k-1]$ is already memorized, but we need to compute $\text{ERR}_\text{M}(e_l^{k-1}+1, i)$. We find the maximum and minimum values using the interval $[e_l^{k-1}+1,\ i]$. If both $e_l^{k-1}+1$ and i are in the current block (i.e. $(r+1)$-th block), we use the interval tree and it takes $O(\log M)$ time. If $e_l^{k-1}+1$ (therefore e_l^{k-1}) belonged to a previous block (r-th block), we already know the maximum and minimum values from $e_l^{k-1}+1$ to end of the last (i.e. r-th) block. We can also get the minimum and maximum of the values for the current block using the interval tree in $O(\log M)$ time.

The time to compute $\text{APXERR}_\text{M}[i,k]$ for $k > 1$ with binary search is $O(\log n_q \log M)$ because binary search and evaluating $\text{ERR}_\text{M}()$ takes $O(\log n_q)$ and$O(\log M)$ respectively. Since the number of evaluations of $\text{APXERR}_\text{M}[i,k]$ needed to extend k-th list is $O(n_q * \log M)$, the total time to extend k-th list is $O(M + n_q(\log M)^2(\log n_q))$ where $O(M)$ is the time to read and process a single block of data.

Procedure ApproxHist$\text{ERR}_\text{M}()$
begin
1. **For** $r = 1$ **to** n/M {
2. Read the next block of M elements
3. Create an interval tree
   with M elements for min/max queries
4. **For** $k = 1$ **to** $B-1$
5. Extend k-th List
6. **For** $k = 1$ **to** $B-1$
7. Update min/max values of end-points
8. }
9. //S is a set of end-points of $(B-1)$-th queue
10. $\text{APXERR}_\text{M}(n, B) = min_{i \in S}$
   $(\text{APXERR}_\text{M}(i, B-1) + \text{ERR}_\text{M}(i+1, n))$

end

Figure 6: Approx. Max Relative Error

Lemma 4.1 *The maximum number of elements in the interval lists, n_q is $O(B\epsilon^{-1}\log n)$.*

Lemma 4.2 *The time to read the next M data values and extend all of B lists is given by:*

$$O\left(M + (B^2\epsilon^{-1}\log n)(\log^2 M)(\log\log n + \log(B/\epsilon))\right)$$

Thus, the total time of *ExtendList* is n/M times the above time to process a single block, it becomes

$$O\left(n + \frac{n}{M}(B^2\epsilon^{-1}\log n)(\log^2 M)(\log\log n + \log\frac{B}{\epsilon})\right)$$

Observe that, for our choice of M, this is $O(n)$.
The following can be proved by induction on k'.

Lemma 4.3 (Proof of Correctness) *For any i for which $\text{APXERR}_\text{M}[i,k']$ is evaluated, we have*

$$\text{APXERR}_\text{M}[i,k'] \leq (1+\delta)^{k'}\text{TRERR}_\text{M}[i,k']$$

When $\delta = \epsilon/(2B)$, by the inductive claim, we have

$$(1+\epsilon/(2B))^B\text{TRERR}_\text{M}[n,B] \leq (1+\epsilon)\text{TRERR}_\text{M}[n,B]$$

for $0 \leq \epsilon < 1$, where $\text{TRERR}_\text{M}[n,B]$ is the optimal error. Thus, the approximate error $\text{APXERR}_\text{M}[n,B]$ is at most $(1+\epsilon)\text{TRERR}_\text{M}[n,B]$ and we have a $(1+\epsilon)$-approximation of the optimal error.

The approximation algorithm, ApproxHist$\text{ERR}_\text{M}()$, is presented in Figure 6. Observe that if we set $M = (B^2\epsilon^{-1}\log n)(\log^2 M)(\log\log n + \log\frac{B}{\epsilon})$, then the running time of ApproxHist$\text{ERR}_\text{M}()$ is $O(n)$, with relatively small space. Notice that since we would be reading block by block, we would get a streaming algorithm. Observe for this value of M, we have $\log M = O(\log\log n + \log\frac{B}{\epsilon})$. Putting everything together, we have

Theorem 4.4 *We can find a $(1+\epsilon)$ approximation to maximum relative error histogram in time $O(n)$ and space $M = \Omega(B^2\epsilon^{-1}\log n(\log\log n + \log\frac{B}{\epsilon})^3)$. This algorithm considers the input sequence left to right and looks at every input value (block) at most once. Thus this algorithm applies to streaming data as well.*

4.2 Sum of Squared Relative Errors

In the case of ERR_{sq}, we would need to store ASUM, BSUM and CSUM for each i and k such that $\text{ERR}_{\text{sq}}[i,k]$ increases by a factor of $(1+\epsilon/(2B))$. We apply a similar algorithm as in Figure 6 to incrementally read blocks of M points each and construct the requisite approximations. The block size determines the space requirement and will be *larger* in this case than that for ERR_{M} as the reason will be seen shortly. We can prove that Lemma 4.1 holds in this case as well. Notice that in this case, since we have to compute

$$\text{TRERR}_{\text{sq}}[i,k] = \min_l \left\{\text{TRERR}_{\text{sq}}[e_l^{k-1}, k\text{-}1] + \text{ERR}_{\text{sq}}(e_l^{k-1}+1, i)\right\}$$

which means we cannot use binary search and have to spend $O(n_q)$ time to evaluate the minimum. However, evaluation of $\text{ERR}_{\text{sq}}(e_l^{k-1}+1, i)$ can be performed in time $O(1)$ now using the stored ASUM, BSUM and CSUM. Thus, k-th interval list can be extended in time $O(M + (n_q)^2\log M)$. Thus the overall time will be $O(n + \frac{n}{M}(n_q)^2\log M)$ – *therefore to achieve $o(n)$ running time, M needs to be larger in this case.*

Theorem 4.5 *We can construct a $(1+\epsilon)$-approximation for the sum of squared relative error histograms in time $O(n)$ and space $O(B^3\epsilon^{-2}(\log^2 n)(\log\log n + \log\frac{B}{\epsilon})$.*

4.3 Sum of Relative Errors

We propose a similar algorithm as in previously subsections. However, for sum of relative errors, computation of $\text{ERR}_{\text{S}}(i,j)$ does depend *not only* on some aggregate statistics at the endpoints i and j, *but also* on the entire set of values in the interval. Thus, to answer ERR_{S} for arbitrary intervals quickly, preprocessing step of building an interval tree is required. We can prove the following (along the same lines as the proofs of Lemma 4.1, and Lemma 4.3):

Lemma 4.6 *If the evaluation of $\text{ERR}_{\text{S}}(i,j)$ for any $[i,j]$ can be supported in $O(T)$ time with preprocessing $O(P)$, we can construct a $(1+\epsilon)$ factor approximation of the optimal histogram in time $O(P + nB^3\epsilon^{-2}T\log^3 n)$.*

Lemma 4.7 *We can evaluate ERR_{S} for an arbitrary interval with $P = O(n\log n)$ and $T = O(\log^3 n)$.*

From Lemma 4.6, and Lemma 4.7, we conclude:

Theorem 4.8 *In $O(n\log n + B^3\epsilon^2\log^6 n)$ time and $O(n\log n)$ space, we can compute a $(1+\epsilon)$-approximation for the sum of relative errors histograms.*

5 Experimental Result

To investigate the performance gains of REHIST over existing techniques, we conducted experiments using real-life as well as synthetic data sets. The sanitary bound c is set to the 10-percent value in the data as in [3, 4]. We used our implementations of the probabilistic thresholding scheme [3, 4] and the deterministic thresholding scheme [17] as representatives of traditional summarization techniques that considers relative errors as objective function.

5.1 Synthetic Data Sets

We considered one-dimensional synthetic data distribution. The data sets were generated with Zipfian frequencies for various levels of skew controlled by the z parameter of the Zipfian. The tuple count was set to 10^6. We varied the z parameter values between 0.3 (low skew) and 2.0 (high skew), the distinct values between 128 ($= 2^7$) and 16384 ($= 2^{14}$). Note that the time and space complexities are not dependent on number of tuples and thus we did not vary this parameter.

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 [3, 4]: *NoPerm*, *Normal*, *PipeOrgan* and *Random*. `Normal` permutes the frequencies to resemble a bell-shaped normal distribution, with higher frequencies at the center of the domain. Due to the lack of space, we present the experimental results with Normal permutation only. The detailed description of these permutations are presented in [3, 4].

5.2 Algorithms

Since the probabilistic [4] and deterministic thresholding schemes [17] did not consider all of three relative error measures, we modified their algorithm to report these errors as well. We conducted a comprehensive performance evaluation of the various schemes. Specifically, we show the performance figures of the following schemes:

- **V-OPT:** It represents the V-Optimal histogram construction algorithm [14] presented in Section 2.2. Even though the error metric used for V-optimal histograms is different, it is interesting to see how other state-of-the-art histogram techniques compare against REHIST which minimizes relative error.

- **P-Wavelet:** This is our implementation of the probabilistic thresholding scheme of [3]. [‖] We used the same default parameter values that were used in [3]. For instance, we set the value of q to 10 and the perturbation parameter δ to $\min\{0.01, c/100\}$. As the authors in [3, 4] suggested, we performed five trials using different random seeds, and the synopsis with the least value was chosen.

- **D-Wavelet:** It is the deterministic thresholding method that was introduced in [17].

- **REHIST-OPT:** These represent optimal histogram construction algorithms presented in Section 3. Depending on the error measure used, REHIST-OPT-M, REHIST-OPT-SQ and REHIST-OPT-S represent OptHistERR_{M}, OptHistERR_{sq}, and OptHistERR_{S} respectively .

- **REHIST-APPROX:** These represent approximate histogram construction algorithms described in Section 4. REHIST-APP-M, REHIST-APP-SQ and REHIST-APP-S represent approximation algorithms using ERR_{M}, ERR_{sq}, and ERR_{S} respectively .

  Since none of the competitive suggestions consider streaming data, the comparisons are given for offline algorithms only, with blocksize as the number of distinct values.

Note: In case of all the histogram algorithms, we found that first obtaining a coarse approximation (say factor 2) and using that solution to prune the lists (not to use $\text{APXERR}_{\text{M}}[i,k]$ larger than this coarse value) has significant performance benefits.

All experiments reported in this section were performed on Pentium-4 2.8 GHz machine with 512 MB of main memory, running Linux. All the methods were implemented using GCC compiler of Version 2.95.3.

5.3 Experimental Results - Synthetic Data

We present some of our experimental results with synthetic data sets. For wavelet methods, each coefficient requires two numbers: the coefficient value and the coefficient index. Similarly, in histograms, starting boundary value of the next bucket is the current bucket's ending value plus one, each bucket needs two numbers: its representative value and ending boundary value. Thus, the number of buckets in histogram and the number of coefficients in wavelets methods were set to be equal and represented by B. The default value of B was set to 50. The default skew was 1.0 and the default number of distinct values was set to 2048.

Number of Buckets: Figure 7-(a), (b) and (c) represent the graphs as the number of buckets B is varied. We plot the maximum relative error for ERR_{M} in the graph. However, for ERR_{S} and ERR_{sq}, we plot the average per distinct value. We notice that the accuracy of REHIST is much better than P-Wavelet, D-Wavelet and V-OPT. Typically, as we expected, V-OPT is the worst performer. For small B, the wavelet methods are bad. As expected, the relative error measures decrease with increasing the number of buckets B. The quality of histograms produced by REHIST-APP was almost the same as REHIST-OPT.

Skew Parameter: We also varied the skew parameter. The relative error measures increase with higher skew parameter value. The graphs show that REHIST results in significant improvement in quality as compared to both wavelet methods and V-OPT. The quality of REHIST-APP was very close to that of REHIST-OPT. Due to the lack of space, we can not present the graphs in this paper.

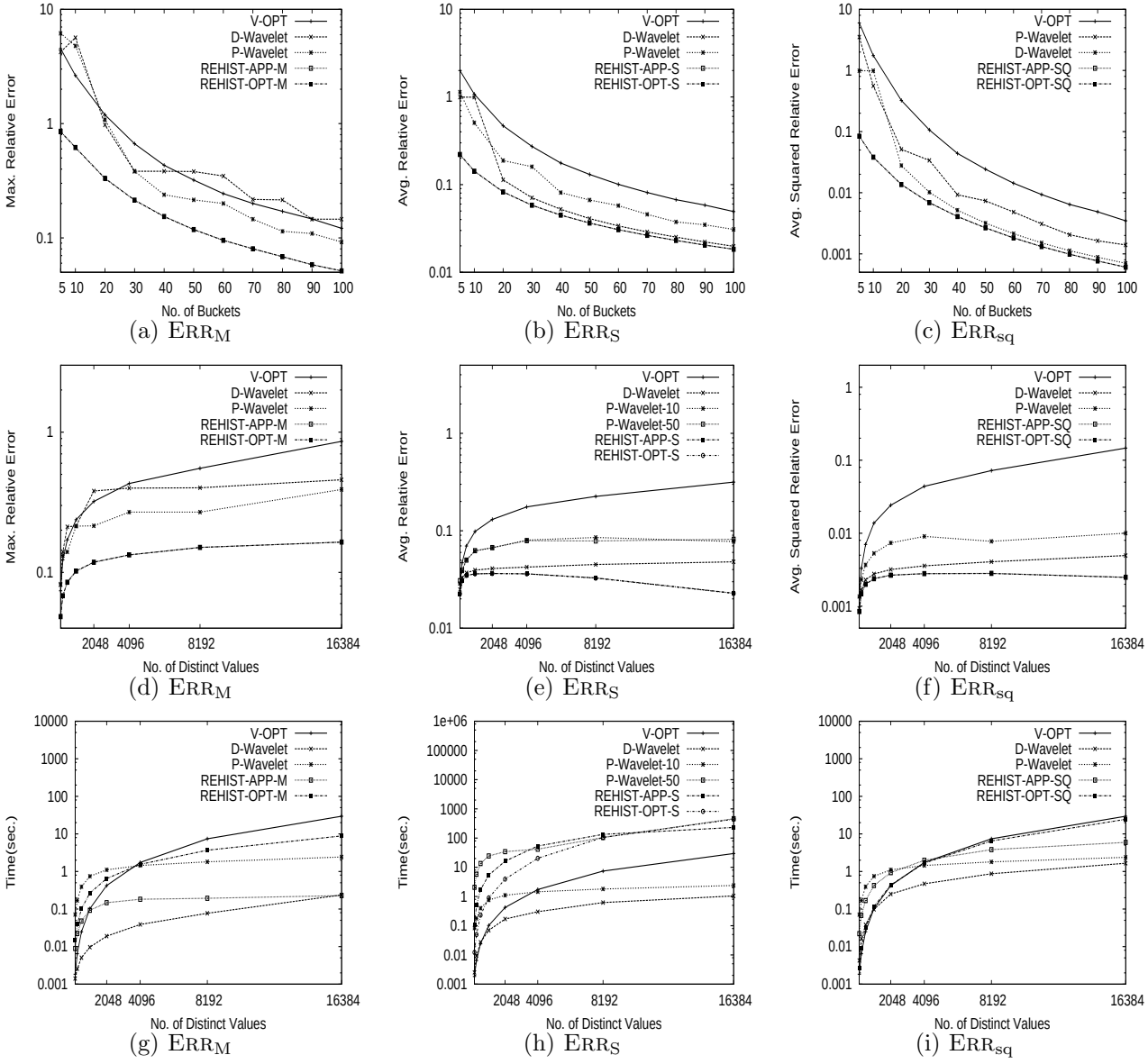
Number of Distinct Values: Figure 7-(d), (e) and (f)) were obtained by varying the number of distinct values. These graphs again show that the solutions obtained by REHIST are much more accurate than D-Wavelet, P-Wavelet and V-OPT. V-OPT is again the worst performer. We also compare execution times in Figure 7-(g), (h) and (i). D-Wavelet is the fastest. For ERR_{M} and ERR_{sq}, REHIST-APP was at least as fast as P-Wavelet. However, P-Wavelet is faster than REHIST-APP for ERR_{S}. Since increasing q value for P-Wavelet improves the quality of wavelet synopsis, we increased the value of q from 10 to 50. However, while the execution time with $q = 50$ is typically 10 times slower than that with $q = 10$, the quality with $q = 50$ was not significantly better. Thus, spending more time and increasing q does not help for P-Wavelet. In this regard, the running time of REHIST appears justifiable considering the significantly better quality of representations constructed by REHIST. Although REHIST-OPT is the slowest, for small number of distinct values, REHIST-OPT is faster than REHIST-APP. As expected, this is reversed as we increased the number of distinct values.

5.4 Experimental Results - Real-life Data Sets

We also experimented with real-life data sets. We used the *Cover Type* data set from the National Forest Service, which was downloaded from UC Irvine[**]. There are 581,012 tuples in the data set. Among 54 attributes, we report "hillshade3pm" (`CovType-HS3`) and "aspect" (`CovType-A`). Because these attributes have widely different distributions, they were used for performance study in [3, 4]. `CovType-HS3` measures a

[‖] It is an improved version of [4] using binary search.

[**] Available at ftp://ftp.ics.uci.edu/pub/machine-learning-databases.

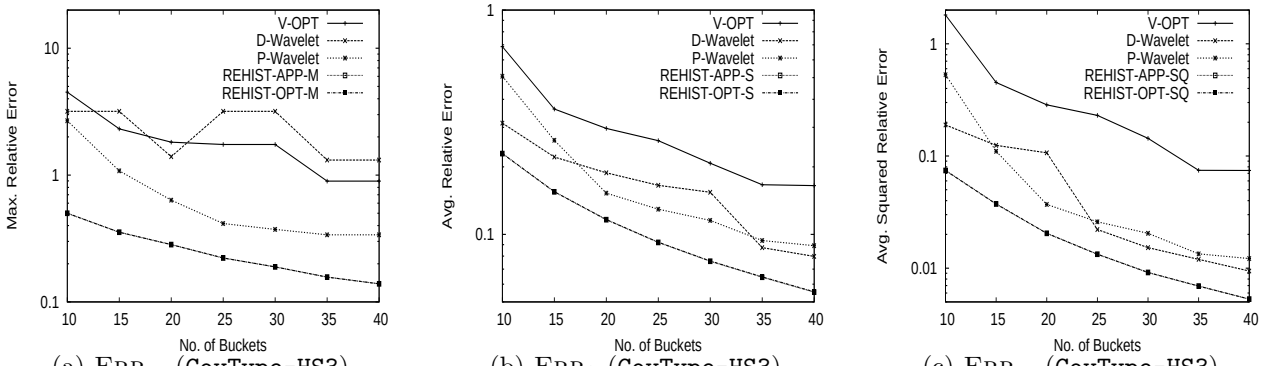


(a) ERR_{M} (b) ERR_{S} (c) ERR_{sq}

(d) ERR_{M} (e) ERR_{S} (f) ERR_{sq}

(g) ERR_{M} (h) ERR_{S} (i) ERR_{sq}

Figure 7: Approximation Error for Normal Zipfian permutation with Synthetic Data Sets



(a) ERR_{M} (`CovType-HS3`) (b) ERR_{S} (`CovType-HS3`) (c) ERR_{sq} (`CovType-HS3`)

Figure 8: Approximation Error for Real-Life Data Set

hillshade index (from 0 to 255) at 3 pm on the summer solstice. Its histogram is bell-shaped and relatively smooth. `CovType-A` has uniformly spread distribution with a pipe-organ-style fluctuation and considerable peaks of noise. Due to lack of space, the quality of histograms were plotted for `CovType-HS3` only in Figure 8. Our results show that REHIST provides significantly better accuracy than D-Wavelet, P-Wavelet and V-OPT.

6 Summary

Histograms and Wavelet synopsis provide useful tools in query optimization and approximate query answering. Previous algorithms for relative error use wavelet approximation schemes with deterministic or probabilistic thresholding. The deterministic scheme suggests heuristics which are not guaranteed to minimize relative error. The probabilistic scheme proposes a complicated optimization, and proceeds to provide an approximation, which holds in expectation only. Expected guarantees are not sufficient for minimizing maximum error objective. We presented optimal as well as faster approximation algorithms with several relative error measures. We did comprehensive analysis of time and space complexities of these algorithms and, with synthetic and real-life data sets, demonstrated the effectiveness of our algorithms in providing significantly more accurate answers compared to the wavelet based methods and V-Optimal algorithm.

Acknowledgments

The authors wish to thank the referees for many useful comments which have helped in improving the presentation of the manuscript. The work was supported by the Ministry of Information and Communication in Korea through the University Information Technology Research Center (ITRC) Support Program.

References

[1] S. Acharya, P. Gibbons, V. Poosala, and S. Ramaswamy. The Aqua Approximate Query Answering System. *Proc. of ACM SIGMOD*, 1999.

[2] K. Chakrabarti, E. J. Keogh, S. Mehrotra, and M. J. Pazzani. Locally adaptive dimensionality reduction for indexing large time series databases. *ACM TODS*, 27(2):188–228, 2002.

[3] M. N. Garofalakis and P. B. Gibbons. Probabilistic wavelet synopses. *To appear in ACM TODS*.

[4] M. N. Garofalakis and P. B. Gibbons. Wavelet synopses with error guarantees. In *Proc. of ACM SIGMOD*, 2002.

[5] A. C. Gilbert, S. Guha, P. Indyk, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Fast, small-space algorithms for approximate histogram maintenance. In *Proc. of ACM STOC*, 2002.

[6] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Optimal and approximate computation of summary statistics for range aggregates. In *Proc. of ACM PODS*, 2001.

[7] S. Guha, P. Indyk, S. Muthukrishnan, and M. Strauss. Histogramming data streams with fast per-item processing. In *Proc. of ICALP*, 2002.

[8] S. Guha and N. Koudas. Approximating a Data Stream for Querying and Estimation: Algorithms and Performance Evaluation. In *Proc. of ICDE*, 2002.

[9] S. Guha, N. Koudas, and K. Shim. Data Streams and Histograms. In *Proc. of STOC*, 2001.

[10] S. Guha, N. Koudas, and D. Srivastava. Fast algorithms for hierarchical range histogram construction. In *Proc. of ACM PODS*, 2002.

[11] S. Guha, K. Shim, and J. Woo. REHIST: Relative error histogram construction algorithms. Techical Report, Seoul National University, Seoul, Korea, July 2004.

[12] Y. Ioannidis and V. Poosala. Balancing Histogram Optimality and Practicality for Query Result Size Estimation. *Proc. of ACM SIGMOD*, 1995.

[13] Y. E. Ioannidis. Universality of serial histograms. In *Proc. of the VLDB Conference*, 1993.

[14] H. V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. C. Sevcik, and T. Suel. Optimal Histograms with Quality Guarantees. In *Proc. of the VLDB Conference*, 1998.

[15] N. Koudas, S. Muthukrishnan, and D. Srivastava. Optimal histograms for hierarchical range queries. In *Proc. of ACM PODS*, 2000.

[16] Y. Matias, J. S. Vitter, and M. Wang. Wavelet-Based Histograms for Selectivity Estimation. *Proc. of ACM SIGMOD*, 1998.

[17] J. Vitter and M. Wang. Approximate computation of multidimensional aggregates on sparse data using wavelets. *Proc. of SIGMOD*, 1999.