July 2007

A Note on Linear Time Algorithms for Maximum Error Histograms

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

Kyuseok Shim
Seoul National University, South Korea

A Note on Linear Time Algorithms for Maximum Error Histograms

Abstract
Histograms and Wavelet synopses provide useful tools in query optimization and approximate query answering. Traditional histogram construction algorithms, e.g., V-Optimal, use error measures which are the sums of a suitable function, e.g., square, of the error at each point. Although the best-known algorithms for solving these problems run in quadratic time, a sequence of results have given us a linear time approximation scheme for these algorithms. In recent years, there have been many emerging applications where we are interested in measuring the maximum (absolute or relative) error at a point. We show that this problem is fundamentally different from the other traditional $non l_\infty$ error measures and provide an optimal algorithm that runs in linear time for a small number of buckets. We also present results which work for arbitrary weighted maximum error measures.

Keywords
histograms, algorithms

Concise Papers

A Note on Linear Time Algorithms for Maximum Error Histograms

Sudipto Guha and Kyuseok Shim

Abstract—Histograms and Wavelet synopses provide useful tools in query optimization and approximate query answering. Traditional histogram construction algorithms, e.g., V-Optimal, use error measures which are the sums of a suitable function, e.g., square, of the error at each point. Although the best-known algorithms for solving these problems run in quadratic time, a sequence of results have given us a linear time approximation scheme for these algorithms. In recent years, there have been many emerging applications where we are interested in measuring the maximum (absolute or relative) error at a point. We show that this problem is fundamentally different from the other traditional $\text{non-}\ell_\infty$ error measures and provide an optimal algorithm that runs in linear time for a small number of buckets. We also present results which work for arbitrary weighted maximum error measures.

Index Terms—Histograms, algorithms.

1 Introduction

One of the central problems in database query optimization is obtaining a fast and accurate synopsis of data distributions. Given a query, the optimizer tries to determine the cost of various alternative query plans based on estimates [16], [12], [13]. From the work pioneered in [8], [9], and [14], the focus has been on serial histograms 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. Consider an array $\{x_i\}$ of data values. 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.

More recently, histograms have been used in a broad range of topics, e.g., approximate query answering [1], mining time series data [11], and curve simplification [2], among many others. With this diverse growth in the number of applications, there has been a growth in the number of different error functions, other than the sum of squares, as well. Maximum error metrics arise naturally in the applications where we wish to represent the data with uniform fidelity throughout the domain, instead of an average (sum) measure. In this paper, we focus on maximum error measures and show that these allow significantly faster optimum histogram construction algorithms than the other (sum-based) measures.

In an early paper, Jagadish et al. [10] gave an $O(n^2B)$ algorithm for constructing the best V-Optimal histogram. This algorithm is based on dynamic programming which generalizes to a wide variety of error measures as well. The quadratic running time has been undesirable for large data sets and a large number of approximation algorithms have been introduced which have running time linear in the size of the input at the expense of finding a solution which is $(1+\epsilon)$ times that of the optimal solution (see [5], [6]). However, a natural question has remained regarding the best running time of the optimal algorithm. It is shown in [7] that the optimum histogram under the maximum relative error criterion can be constructed in $O(nB\log^2 n)$ time.

One effect of error measures such as $\sum_i (x_i - \hat{x}_i)^2, \sum_i |x_i - \hat{x}_i|$ is that all the data points are not approximated equally in the optimum solution. While this may not be an issue for many applications, there exists applications where we may be interested in approximating the data at every point with high fidelity. The authors of [3], [4] describe this property of not approximating all points equally as the "bias" of the approximation, and demonstrate that in several situations, this bias is undesirable. The solutions that avoid the bias are pointwise approximations or maximum error metrics, for example, the maximum absolute error and maximum relative error metrics ($\max_i |x_i - \hat{x}_i|$ or $\max_i |x_i - \hat{x}_i| / \max\{c, |x_i|\}$, respectively). The parameter c is a sanity bound that avoids the influence of very small values. In this paper, we show that for these metrics, there exists an $O(n + B^2\log^3 n)$ time algorithm. For general weighted maximum error, the running time increases to $O(n\log n + B^2\log^6 n)$. We note that our techniques extend to "hybrid" measures such as the maximum of the sum of (or sum of squares of) errors in a bucket. However, to keep the discussion concrete and to ease the presentation, we will not focus on these measures.

2 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 $O(n)$ numbers. The histogram is mostly 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. Depending on the situation, the error may be tempered by the importance w_i we attach to each point i.

Definition 1. *Given a weight vector $\{w_1, \ldots, w_i, \ldots\ w_n\}$, s.t., each $w_i \ge 0$, the weighted maximum error for a point $i \in [s_r, e_r]$ with a bucket $b_r = (s_r, e_r, \hat{x}(r))$ is defined as $w_i|\hat{x}(r) - x_i|$.*

Definition 2 (Maximum Error Histograms). *Given a set of weights (which could all be 1), the (serial) histogram problem is to construct a partition of the interval $[1, n]$ in at most B buckets such that we minimize the maximum error.*

Two notable, and well used, examples are 1) the ℓ_∞ or the maximum error, where $w_i = 1$ and 2) the relative maximum error where the weights are $w_i = 1/\max\{c, |x_i|\}$ and, therefore, the relative error at the point i is $|\hat{x}(r) - x_i| / \max\{c, |x_i|\}$, where c is a sanity constant which is used to reduce excessive domination of relative error by small data values. Relative error metrics were

- *S. Guha is with the Department of Computer Information Sciences, University of Pennsylvania, 3451 Walnut Street, Philadelphia, PA 19104. E-mail: sudipto@cis.upenn.edu.*
- *K. Shim is with the Department of Computer Science and Electrical Engineering, Seoul National University, Kwanak PO Box 34, Seoul 151-742, Korea. E-mail: shim@ee.snu.ac.kr.*

Manuscript received 6 May 2006; revised 25 Oct. 2006; accepted 12 Feb. 2007; published online 21 Mar. 2007.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-0236-0506.
Digital Object Identifier no. 10.1109/TKDE.2007.1039.

studied in [3], [7]. In this case, the error of a bucket $b_r = (s_r, e_r, \hat{x})$ is defined as follows (for relative ℓ_∞ error):

$$\mathrm{ERR_M}(s_r, e_r) = \min_{\hat{x}} \max_{i \in [s_r, e_r]} \frac{|x_i - \hat{x}|}{\max\{c, |x_i|\}}.$$

In the above setting, letting c be an absolute constant larger than all numbers in the input converts the error on the previous page to absolute ℓ_∞ error (multiplied by $\frac{1}{c}$) and this is the reason we can discuss **both errors** at the same time. Interestingly, these two cases are truly special, and we showcase their difference with arbitrary weighted maximum error histograms in Section 4.

3 Maximum Error Histograms

In this section, we will focus on the constructing histograms that minimize the maximum absolute error or the maximum relative error. We will first prove a lemma about determining the error of a fixed bucket, and subsequently use that to devise our complete algorithm. The problem of determining maximum error is easy.

Proposition 1. *Given a set of numbers $x_1, \ldots, x_\ell$, the maximum error generated by minimizing maximum errors is defined by the minimum and the maximum over x_i.*

The following lemma focuses on relative error:

Lemma 1 ([7]). *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 below:*

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

Proof. Let $max = \max_i(x_i)$ and $min = \min_i(x_i)$. Suppose the optimum representative value minimizing the maximum relative error is x^*. Notice that setting $x^* = 0$ gives a relative error of at most 1 since $|x_i| \leq \max(|x_i|, c)$; thus, the error with x^* cannot be more than 1.

- **Case 1** ($c \leq min \leq max$). The relative error function is continuous at x^* and it monotonically increases as the value x_i moves away from x^* as the following formula illustrates:

$$\frac{|x^* - x_i|}{\max\{|x_i|, c\}} = \begin{cases} (x^* - x_i)/x_i & \text{if } x_i \leq x^* \\ (x_i - x^*)/x_i & \text{if } x_i > x^*. \end{cases}$$

  Thus, we can see that the maximum relative error is either at min or max. Let $R_{min} = (x^* - min))/min$ and $R_{max} = (max - x^*)/max$. Then, in order to find the optimal representative value, we need to compute the value of x^* satisfying $R_{min} = R_{max}$. The value of x^* becomes the harmonic mean, $(2\max \cdot \min)/(\max + \min)$ and it results in the error of $(\max - \min)/(\max + \min)$.
- **Case 2** ($min \leq max \leq c$). This case is symmetric to the above case. Thus, with similar argument, we get $\frac{\min - \max}{\max + \min}$.
- **Case 3** ($-c \leq \min \leq c \leq \max$). We split into two cases: 1) $\min \leq x^* \leq c$ or 2) $c \leq x^* \leq \max$. Thus, we have

  1. **When** $\min \leq x^* \leq c$,

$$\frac{|x^* - x_i|}{\max(|x_i|, c)} = \begin{cases} (x^* - x_i)/c & \text{if } x_i \leq x^* \\ (x_i - x^*)/c & \text{if } x^* \leq x_i \leq c \\ (x_i - x^*)/x_i & \text{if } c \leq x_i. \end{cases}$$

  2. **When** $c \leq x^* \leq \max$,

$$\frac{|x^* - x_i|}{\max(|x_i|, c)} = \begin{cases} (x^* - x_i)/c & \text{if } x_i \leq c \\ (x^* - x_i)/x_i & \text{if } c \leq x_i \leq x^* \\ (x_i - x^*)/x_i & \text{if } x^* \leq x_i. \end{cases}$$

  For both above cases, the expression of R_{min} and R_{max} are the same, respectively. Thus, we can calculate x^* by solving the equation of $R_{min} = R_{max}$. We get $x^* = \frac{\max(\min+c)}{\max+c}$ and the optimal maximum relative error becomes $\frac{(\max-\min)}{(\max+c)}$.
- **Case 4** ($\min \leq -c \leq \max \leq c$). This case is symmetric to the above case. Thus, with similar argument, we get the maximum relative error of $\frac{\max-\min}{c-\min}$.
- **Case 5** ($-c \leq \min \leq \max \leq c$). As the formula below illustrates, the relative error function is continuous at x^* and it monotonically increases as the value x_i moves away from x^*:

$$\frac{|x^* - x_i|}{\max(|x_i|, c)} = \begin{cases} (x^* - x_i)/c & \text{if } x_i \leq x^* \\ (x_i - x^*)/c & \text{if } x_i > x^*. \end{cases}$$

  We can calculate x^* by solving the equation of $R_{min} = R_{max}$. We get $x^* = \frac{(\max+\min)}{2}$ and the optimal maximum relative error becomes $\frac{\max-\min}{2c}$.
- **Case 6** ($\min \leq -c < c \leq \max$). We can see that the relative error function becomes larger than one when x^* is nonzero, while it is one when x^* is zero. Thus, we get $x^* = 0$ and the optimal maximum relative error becomes 1. □

Computing Maximum and Minimum of Intervals Efficiently. In our algorithm, we would evaluate $\mathrm{ERR_M}(i, j)$ for many different intervals $[i, j]$. However, it is clear that these intervals are all related, and we should be able to create a data structure that allows us to compute $\mathrm{ERR_M}(i, j)$ efficiently for all i, j. 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 correspond 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 error for the bucket. It reduces the time complexity of computing the minimum (or maximum) to $O(\log n)$.

3.1 The Algorithm

In [7], we have shown that the algorithm for computing maximum relative error can be found in $O(Bn \log^2 n)$ time and $O(Bn)$ space. In this section, we provide a better algorithm. Assume that the B bucket optimal histogram with the maximum error measure for the interval $[1, n]$ has the error of $\Delta^\star$. For the bucket of the interval $[1, s]$ for an s with $1 \leq s \leq n$, if s is smaller than the right boundary of the first bucket in the optimal histogram, the error of the bucket for $[1, s]$ is at most $\Delta^\star$. However, if s is larger than the right boundary of the first bucket in the optimal histogram, the error of

```
                                    Procedure OptHist(i,n,k)
procedure TryThreshold(Δ,i,n,k)     begin
begin                                 if (k = 1) return ERR_M(i, n);
  if (i > n) return true;             low = i;
  if (k = 1)                          high = n;
    return (ERR_M(i, n) ≤ Δ);         while (high > low) {
  low = i;                              mid = (high + low)/2;
  high = n;                             Δ = ERR_M(i, mid);
  while (high > low) {                  if TryThreshold(Δ,mid+1,n,k-1)
    mid = (high + low + 1)/2;             high = mid;
    if ERR_M(i, mid) > Δ                else low = mid + 1;
      high = mid − 1;                   }
    else low = mid;                     Δ = ERR_M(i, low);
  }                                     if (low > i)
  return                                  and (OptHist(low+1,n,k-1)< Δ)
    TryThreshold(Δ,low+1,n,k-1)             return OptHist(low+1,n,k-1);
end                                     else return Δ;
                                    end
```

Fig. 1. The $\mathrm{OptHistERR_M}$ algorithm.

the bucket for $[1, s]$ is at least $\Delta^\star$. Assuming the errors of the buckets $[1, s]$ and $[1, s+1]$ are Δ_s and Δ_{s+1}, respectively, we are interested in the largest s such that there does not exist a $(B-1)$ bucket histogram whose error is at most Δ_s for the interval $[s+1, n]$, but exist a $(B-1)$ bucket histogram whose error is at most Δ_{s+1} for the interval $[s+2, n]$. In this case, the error of the optimal histogram $\Delta^\star$ is minimum of Δ_{s+1} and the error of the best $(B-1)$ bucket histogram for the interval $[s+2, n]$. Since the max error of the bucket for $[1, s]$ is monotonically increasing with s, we can perform binary search to find the largest s satisfying the condition. As we find the largest s, we perform the same procedure recursively for the interval $[s+2, n]$ with $(B-1)$ buckets.

The linear time algorithm *OptHist* for constructing an optimal histogram with the max error measures is given in Fig. 1. *OptHist* invokes $TryThreshold(\Delta, i, n, k)$ to check whether there exist a k bucket histogram for the interval $[i, n]$, where the max error is at most Δ. $TryThreshold(\Delta, i, n, k)$ finds the largest value low using a binary search such that the error of the bucket $[i, low]$ is at most Δ and the error of the histogram of the interval $[low+1, n]$ using $(k-1)$ buckets is larger than Δ. After we find the largest low, we call $TryThreshold(\Delta, low+1, n, k-1)$ recursively and return its result.

Lemma 2. *If there is a way of partitioning the interval $[i, n]$ into k intervals such that the maximum error is no more than Δ, $TryThreshold(\Delta, i, n, k)$ returns true.*

Proof. The procedure finds the largest value low such that the error of the bucket $[i, low]$ is at most Δ. Thus, if there is a way of partitioning $[i, n]$ into k buckets such that the maximum error is no more than Δ, then if the first bucket of this (unknown) solution is $[i, z]$ then $z \leq low$. Therefore, there exists a way of partitioning $[low+1, n]$ into $(k-1)$ buckets such that the maximum error is at most Δ. This partitioning can be derived by erasing all the buckets that end before low in the k-bucket solution for $[i, n]$. Now, we have a recursive condition set up, which is checked when $k = 1$. □

Lemma 3. *Procedure $\mathrm{OptHist}(i, n, k)$ returns the best possible error from partitioning $[i, n]$ into k buckets.*

Proof. We will prove the lemma by induction. The statement is clearly true for $k = 1$.

If $k > 1$, the procedure computes low to be the smallest j such that $\mathrm{ERR_M}(i, j) = \Delta$ and there is a solution of error Δ for $[j+1, n]$ using $(k-1)$ buckets. This already means that there is a solution of error Δ for $\mathrm{OptHist}(i, n, k)$. If $low = i$, we actually have $\Delta = 0$ and that is the best possible answer.

If $low > i$, we also know that there does not exist a solution of covering $[i, n]$ using k buckets with error $\mathrm{ERR_M}(i, low-1)$ (otherwise, we would have chosen a lower value of low). Thus, if the optimum error for covering $[i, n]$ with k buckets is $z^\star$, then

$$\mathrm{ERR_M}(i, low-1) < z^\star \leq \Delta.$$

Notice that under no condition will we return a solution greater than Δ. Thus, if $z^\star = \Delta$, we have nothing to prove.

Suppose the optimum solution is strictly less than Δ. Then, the first bucket in the optimum solution must be some $[i, i']$, where $i' < low$. But, if we (possibly) increase the first bucket to $[i, low-1]$, then the error of the first bucket is still less than $z^\star$, and this cannot increase the error of the remaining buckets of the optimal solution. Thus, there must be a solution of error $z^\star$ for covering $[low, n]$ by $(k-1)$ buckets. By inductive hypothesis, we would compute the correct answer in $\mathrm{OptHist}(low, n, k-1)$ and since $z^\star < \Delta$, we have computed the correct answer. □

The running time of the procedure TryThreshold can be expressed by the simple recurrence

$$g(k) = \log^2 n + g(k-1).$$

The first $\log$ term comes from binary search and the second $\log$ term comes from the time taken to evaluate $\mathrm{ERR_M}()$ using an interval tree. Obviously, $g(0) = 0$. Thus, $g(k) = ck \log^2 n$ for some constant c.

The running time of OptHist is therefore given by the following recurrence:

$$f(k) = g(k) \log n + f(k-1).$$

The $\log$ term appears from the binary search. Thus, $f(k) = ck^2 \log^3 n$. To this, we must add the preprocessing time to create the interval tree, which is $O(n)$. Therefore, we can summarize the following:

Theorem 1. *We can compute the optimum histogram under maximum or maximum relative error in $O(n + B^2 \log^3 n)$ time and $O(n)$ space.*

4 Extensions: Weighted Maximum Errors

Let us revisit the general problem of minimizing arbitrary weighted errors $\{w_i\}$. The most basic problem is already interesting: Given numbers $x_i, \ldots, x_j$, and corresponding nonnegative weights $w_i, \ldots, w_j$, compute the x^* that minimizes $\min_x \max_{i \leq r \leq j} w_r |x - x_r|$. This corresponds to the *representation problem of a single bucket*. The best way to view the solution is to focus on Fig. 2a where the three points define *cones* where the slope of the point corresponding to x_r is the corresponding w_r. This cone depicts how the function $w_r|x - x_r|$ behaves as x is varied.

The x^* corresponds to the lowest point in the intersection of all these cones. To compute the x^*, observe that the intersection of cones is a convex region (because each cone is a convex region).

Definition 3. *Define the boundary of the intersection of the cones to be the "profile" for the set of numbers $x_i, \ldots, x_j$. The profile is a convex chain of line segments (stored in sorted order); the number of segments is at most $2|j - i| + 2$. The minimum error and x^* can be computed from the profile using binary search.*

Now, we can divide the point set into two (arbitrary) halves and compute the boundary of each of the convex regions and compute the intersection of these two convex regions, similar to the MergeHull algorithm [15]. Fig. 2b illustrates the process, It is straightforward to see that if we maintain each of the boundaries as convex chains, we can perform a "walk" from left to right and compute the boundary of the intersection. However, that would mean that each merge step (over all recursive divisions) takes as much time as there are lines, and the number of lines is as most twice the number of original points. This gives a divide and

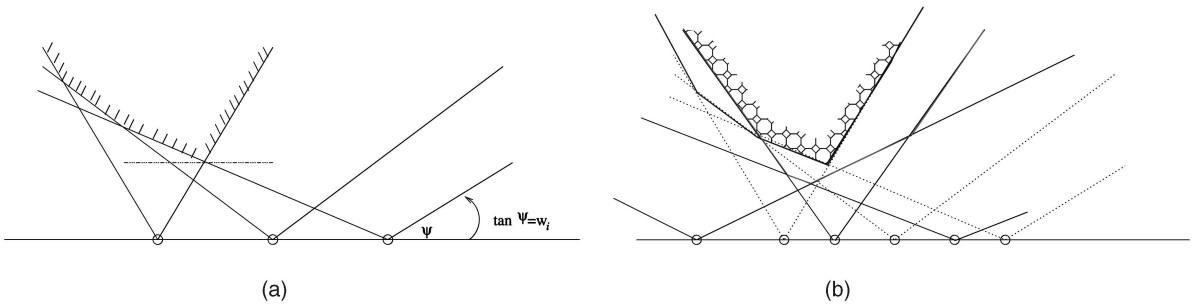

Fig. 2. (a) The shaded region indicates the convex region and the lowest point is the desired x. (b) Shows how to compute the intersection of these convex regions, provided they are maintained in a sorted order, in a manner similar to mergesort.

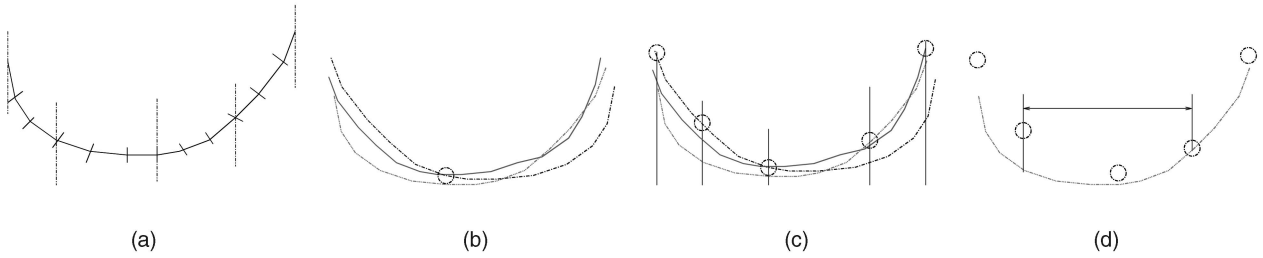

Fig. 3. The algorithm for computing the error of $[i, j]$ in pictures. (a) Shows the division of the first profile into nearly four equal size sets, and define a_1, a_2, a_3, a_4, and a_5. (b) Shows the overall minimum in the intersection of convex regions, the circled point is the optimum. (c) Shows circles at the result of evaluating $\max_i w_i|x - x_i|$ at these values. (d) Shows the information available to the algorithm and how the recursion proceeds.

conquer algorithm to compute the x^*, in time $O(m \log m)$, where $m = |j - i|$. This is clearly not desirable, because then the time to evaluate the error of a bucket may be $O(n \log n)$ and, thus, we would have a $O(n + B^2 n \log^3 n) = O(B^2 n \log^3 n)$ algorithm along the lines of Theorem 1. However, we now use the same principle as in Section 3 to speed up the computation. We prove a basic fact first.

Claim 1. *Suppose we seek to minimize a convex function $f(x)$. If we observe $f(x)$ at the set of distinct values $a_1 < a_2 < \ldots < a_k$, and $f(a_i)$ achieves the minimum, then* $\arg\min_x f(x) \in [a_{i-1}, a_{i+1}]$.

Proof. Suppose otherwise; let x^* be the value that achieves the minimum and this value is less than that of $f(a_i)$. If $x^* < a_{i-1}$, then we have $x^* < a_{i-1} < a_i$ and $f(x^*) < f(a_i) \leq f(a_{i-1})$; which implies that the function is not convex (it increases and then stays the same or decreases which is not possible for a convex function). Thus, $x^* < a_{i-1}$ implies that $f(x^*) = f(a_i)$. If $x^* > a_{i+1}$, then we have $a_i < a_{i+1} < x^*$ and $f(x^*) < f(a_i) \leq f(a_{i+1})$; this also implies that the function remains the same (or increases) and then decreases which is not allowed for convex functions.□

The next lemma captures the fact that we can share the computation of the maximum error across different intervals.

Lemma 4. *For all weighted maximum errors, we can precompute a data structure in $O(n \log n)$ space and time, such that subsequently on any interval $[i, j]$ of interest we can compute the minimum error (and the x) achieved in representing $x_i, \ldots, x_j$ using a single value x, in time $O(\log^4 n)$.*

Proof. Once again, we construct an interval tree over $[1, n]$ by recursive halving. For each half, we compute and store the profile. The size of the profile is at most twice the number of points—therefore, over all the $O(\log n)$ recursive levels, the space used is $O(n \log n)$.

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. Now, we have $O(\log n)$ profiles and we have to compute the minimum point in their intersection. Computing the intersection explicitly requires too much time—we will use the prune and search technique.

Specifically, we will proceed in a round robin fashion over the profiles. Suppose we have picked the first profile: If this profile has over eight line segments, we will divide *this profile* into four partitions such that each partition has almost the same number of line segments. This can be done easily because we store the profiles as sorted arrays. The boundaries of these four pieces would define five points a_1, a_2, a_3, a_4, and a_5. We will evaluate $\max_i w_i|x - x_i|$ for these five values of x using all the profiles; note that for a particular profile and particular a_j, this involves a $O(\log n)$ binary search, because we have to determine the intersection of the $x = a_j$ vertical line with the profile. This means we would use $O(5 \log n) = O(\log n)$ time per profile to estimate the intersection and, therefore, $O(\log^2 n)$ time over the $O(\log n)$ profiles. At this point, we can use Claim 1, and at most $2/3$ of the segments are of interest.[1] The result of this computation is declared as a phase—Fig. 3 shows the computation over a phase.

This means, after $O(\log n)$ such phases (and $O(\log^3 n)$ time), we would have reduced the first profile to less than eight segments. We would now proceed to the second profile, and so on. Note that we always maintain a region containing x^*.

When we finish the above process after $O(\log^4 n)$ time, each profile would have eight line segments each and we can compute the solution over these $O(8 \log n)$ remaining segments in $O(\log^2 n)$ time easily. □

Note that the algorithm can be analyzed better using amortization and/or randomization. As we reduced the first profile, we could also be shrinking the other profiles—we did not consider that. Using randomization, the time can be made $O(\log^3 n)$, we need to repeatedly pick a profile with a probability proportional to the number of remaining segments of interest. After the division, this would reduce the total number of segments of interest across all profiles by a factor of $2/3$. At this point, we again probabilistically choose the profile to be reduced. However, the proof would require verifying that this event happened with high probability—and we omit the discussion in the interest of space and simplicity. Note that even in the weighted case, the error of a

1. Due to odd/even issues in the partitioning, we may have two extra lines which increase the fraction from $1/2$.

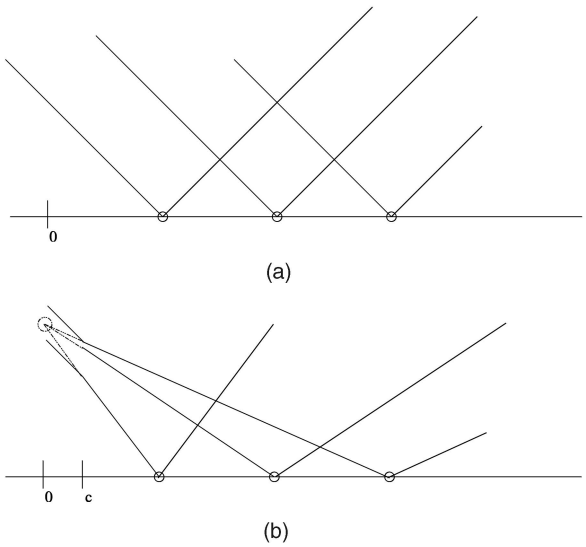

Fig. 4. The cone in the middle is dominated by the adjacent cones. (a) Cones for ℓ_∞. (b) Cones for relative maximum error.

bucket $[i, j]$ does not decrease as j increases. This was the key property used in the proof of Theorem 1. Combining that proof with Lemma 4, we get the following:

Theorem 2. *We can compute the optimum histogram under arbitrarily weighted maximum error in $O(n \log n + B^2 \log^6 n)$ time and $O(n \log n)$ space.*

It is interesting to observe why the maximum and maximum relative error measures are special—if for these weights we draw the cones, then the cones all merge at the same point. For maximum error, the point is at ∞ because the sides of the cones are parallel, this is shown in Fig. 4a. For the maximum relative error, the cones (in the absence of the sanity constant c) intersect at the point $(0, 1)$, which implies that the relative error is 1 if we approximate every (large) value by 0. The constant c makes the situation a bit more complicated, see Fig. 4b, the region $[-c, c]$ distorts the cones into possibly nonconvex shapes. This is why we had to explicitly analyze these regions separately in Lemma 1. But, in both of these examples, the cone in the middle is again dominated by the two adjacent cones. This shows that only the maximum and the minimum values matter for these error measures and why these measures are similar.

5 Summary

Histograms and Wavelet synopsis provide useful tools in query optimization and approximate query answering. The previous algorithm for constructing an optimal histogram with the maximum error criterion takes $O(Bn \log^2 n)$ time and $O(Bn)$ space. In this paper, we presented a linear time optimal algorithm for the maximum error and maximum relative error measures (when B is small, i.e., $B = o(\sqrt{n}/log^2 n)$). We extended the algorithm to arbitrary weights increasing the space and time bounds by small $(\log^2 n)$ factors.

Acknowledgments

S. Guha's research is supported in part by an Alfred P. Sloan Research Fellowship and by a US National Science Foundation Award CCF-0430376. K. Shim's research is supported by the Ministry of Information and Communication, Korea, under the College Information Technology Research Center Support Program, grant number IITA-2006-C1090-0603-0031.

References

[1] S. Acharya, P. Gibbons, V. Poosala, and S. Ramaswamy, "The Aqua Approximate Query Answering System," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* pp. 574-576, 1999.
[2] M. Bertolotto and M.J. Egenhofer, "Progressive Vector Transmission," *Proc. Seventh ACM Symp. Advances in Geographical Information Systems,* pp. 152-157, 1999.
[3] M.N. Garofalakis and P.B. Gibbons, "Wavelet Synopses with Error Guarantees," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* pp. 476-487, 2002.
[4] M.N. Garofalakis and A. Kumar, "Deterministic Wavelet Thresholding for Maximum-Error Metrics," *Proc. 23rd ACM SIGMOD-SIGACT-SIGART Symp. Principles of Database Systems,* pp. 166-176, 2004.
[5] S. Guha, N. Koudas, and K. Shim, "Data Streams and Histograms," *Proc. 33rd Ann. ACM Symp. Theory of Computing,* pp. 471-475, 2001.
[6] S. Guha, N. Koudas, and K. Shim, "Approximation and Streaming Algorithms for Histogram Construction Problems," *ACM Trans. Database Systems,* vol. 31, no. 1, 2006.
[7] S. Guha, K. Shim, and J. Woo, "REHIST: Relative Error Histogram Construction Algorithms," *Proc. Very Large Data Bases Conf.,* pp. 300-311, 2004.
[8] Y.E. Ioannidis, "Universality of Serial Histograms," *Proc. Very Large Data Bases Conf.,* pp. 256-267, 1993.
[9] Y. Ioannidis and V. Poosala, "Balancing Histogram Optimality and Practicality for Query Result Size Estimation," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* pp. 233-244, 1995.
[10] H.V Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K.C. Sevcik, and T. Suel, "Optimal Histograms with Quality Guarantees," *Proc. Very Large Data Bases Conf.,* pp. 275-286, 1998.
[11] E. Keogh, K. Chakrabati, S. Mehrotra, and M. Pazzani, "Locally Adaptive Dimensionality Reduction for Indexing Large Time Series Databases," *ACM Trans. Database Systems,* vol. 27, no. 2, pp. 188-228, 2002.
[12] R. Kooi, "The Optimization of Queries in Relational Databases," PhD thesis, Case Western Reserve Univ., 1980.
[13] M. Muralikrishna and D.J. DeWitt, "Equi-Depth Histograms for Estimating Selectivity Factors for Multidimensional Queries," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* pp. 28-36, 1988.
[14] V. Poosala, Y. Ioannidis, P. Haas, and E. Shekita, "Improved Histograms for Selectivity Estimation of Range Predicates," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* pp. 294-305, 1996.
[15] F.P. Preparata and M.I. Shamos, *Computational Geometry: An Introduction.* Springer-Verlag, 1985.
[16] P.G. Selinger, M.M. Astrahan, D.D. Chamberlin, R.A. Lorie, and T.G. Price, "Access Path Selection in a Relational Database Management System," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* pp. 23-34, 1979.