Paper Number 218

Choosing Bucket Boundaries for Histograms

H. V. Jagadish
AT&T Laboratories
Florham Park, NJ 07932
jag@research.att.com

Nick Koudas*
Computer Systems Research Group
University of Toronto
koudas@cs.toronto.edu

Kenneth C. Sevcik
Computer Systems Research Group
University of Toronto
kcs@cs.toronto.edu

Abstract

Histograms have long been used to capture attribute value distribution statistics for query optimizers. More recently, there has been a growing interest in the use of histograms to produce quick approximate answers to decision support queries. This motivates finding good strategies for specifying histogram buckets. Under the assumption that finding optimal bucket boundaries is computationally inefficient, previous research has focused on finding heuristics that produce good solutions.

In this paper, we present an algorithm to determine bucket boundaries optimally, in time proportional to the square of the number of distinct data values, for a broad class of optimality metrics. Through experimentation, we show that optimal histograms can have substantially lower reconstruction error than histograms produced according to popular heuristics. We also present a new heuristic, based on our understanding of the optimal solution, which in many cases obtains lower reconstruction error than previously proposed heuristics, with a computation cost that is still quite low.

1 Introduction

It is often the case that a data set cannot be stored or processed in its entirety; only a summarized form is stored. A typical way in which data is summarized is by means of a *histogram*.

The summarized data is used to answer different kinds of queries, in the same way the original data would have been used. The answer obtained is not exact but approximate, and contains an

*Contact Address: Computer Systems Research Group, University of Toronto, 6 King's College Road, Toronto, Ontario, M5S 1A1, CANADA. Phone: (416) 978-1685. FAX: (416) 978-4765

error due to the information lost when the data was summarized. This error can be measured according to some appropriate metric such as mean squared error.

This basic idea has long been used, in the database context, to estimate the result size of relational operators for query optimization. The objective is to approximate the data distribution of the values in a column, and to use the approximation to make quick estimates of the result size of the queries in which this column participates.

The same idea has also been used for data analysis and decision support. It has recently been recognized that histograms can be used to provide fast approximate responses to user queries. Consider an application managing a large table that has one record per telephone call indicating the length of the call, and using this table to answer queries about the statistics of call duration. We can save much space by summarizing this information, in a histogram of frequencies of occurrence, for calls with lengths in different ranges. The price of saving space is the need to tolerate some error in the answers provided to queries.

A question arises regarding the following design issue: Given the type of queries that we wish to support and a constraint on the space we may use, what representation of the data minimizes the expected error in the answers provided to the queries? A complementary formulation is: Given the queries and a constraint on the expected error that is tolerable, what representation of the data requires the least space. Section 3 presents a formal description of the problem being studied.

Our main result is that it is possible to obtain an optimal solution to a broad range of formulations of this problem in time that is proportional to the square of the number of distinct values of the attribute being considered. The bulk of this paper is devoted to addressing the first of the two questions above, where the form of summarized data representation has been chosen to be a histogram. In Section 4.5, we briefly show how the dual problem may also be addressed using the same techniques. In Section 4, we present the *Optimal Histogram Algorithm* (OH) and analyze its complexity.

While it is encouraging to find a quadratic time algorithm to produce an optimum solution[1], there are situations in which still faster techniques are required. We use our understanding of the optimal solution to devise a fast heuristic that can quickly find a heuristic close to optimal. This heuristic is discussed in Section 5.

We present experimental results for some sample data distributions in Section 6, comparing the OH algorithm and our heuristic algorithm with previous approaches. In Section 7, we consider alternative error metrics, indicating their suitability for use with relational operators. In addition, we show how our algorithms can employ these alternative error metrics. We conclude in Section 8 with an assessment of where we stand and directions for future work.

We first review related previous work and cover some background material in Section 2.

[1]The execution time of our algorithm is proportional to the square of the number of candidate bucket boundaries considered.

2 Related Work

The problem of approximating a given data distribution has received much attention in the past in various scientific communities. In numerical analysis, the problem has been formulated in terms of approximating a given function piece-wise with a class of simple functions, usually polynomials of some degree [CdB72]. Much of the work in this area has concentrated on identifying the class of polynomials to use, and computing their parameters in order to minimize the error of the approximation. Little attention has been given to the number of parameters or amount of space required. In fact, finding an optimum set of "breakpoints" for a piece-wise polynomial approximation is believed to be very hard due to the non-linearity of the problem space [dB97].

In statistics, the problem has been posed in connection with non-parametric density estimation as constructing a histogram of a given data distribution. But again the effort focused on minimizing the error without taking space constraints into account [GES85]. In fact, statisticians worry about over-fitting a distribution for a given sample set, so too fine a histogram bucket width is considered undesirable. An "optimum" histogram bucket width, in the statistical sense, is one that neither under-fits nor over-fits the data available [FD81].

In the database community, the problem has received attention in the field of query optimization and more specifically in selectivity estimation for relational operators. Several techniques have been proposed. New uses of parametric techniques [MCS88] have been recently proposed by Chen and Roussopoulos [CR94] and by Sun *et al.* [SLRD93].

Other recent approaches are based on non-parametric techniques for selectivity estimation. The techniques proposed include histograms [Koo80] and sampling [LNS90][HS92]. Several types of histograms have been proposed and evaluated experimentally. These include Equal Depth (Height) and Equal Width histograms [PSC84] [MD98], as well as serial histograms [IC93][Ioa93][IP95][PIHS96]. Serial histograms have been shown to be superior for joins and selections when a list of attribute values in each bucket is maintained. The notion of variance optimality is introduced by Ioannidis and Poosala [IP95], and an exponential time ($O(N^B)$, where B is the number of buckets and N the number of distinct data values) algorithm is presented to achieve this optimality with a serial histogram. In serial histograms, as opposed to value histograms, data items are sorted by frequency rather than by attribute value.

Several heuristic approaches to histogram construction have been proposed. These approaches include:

- Equi-width heuristic (EQW): Equi-width histograms are specified by having each segment include N/B data values (subject to integer constraints).

- Equi-depth heuristic (EQD): Equi-depth histograms are specified by placing the end of the jth bucket at the point where the cumulative distribution function reaches j/B (again subject to integer constraints).

- Maximum Difference (MaxDiff) Heuristic [PIHS96]: The MaxDiff heuristic works as follows. To create B buckets, perform a single pass over the distinct data values, keeping track, using a heap of the $B-1$ maximum differences between successive distinct data values, as well as the index at which each difference occurs. Place the bucket boundaries at exactly these indices. For N distinct data values and B buckets, the complexity of this algorithm is $O(N \log B)$ and it makes use of a heap of B elements.

- End Biased Vopt (EBV) Heuristic [IP95]: The end-biased Vopt (EBV) heuristic was proposed by Ioannidis and Poosala [IP95]. It results in an end-biased histogram suitable for predicting the result size of join queries. The resulting histogram consists of $B-1$ buckets with accurate counts of distinct data values and one bucket that makes the uniformity assumption for all other data value counts not included in the first $B-1$ buckets.

  In order to construct an EBV histogram, we perform a single pass over the data value sequence, and, using a pair of heaps, we keep track of the highest and lowest $B-1$ data value counts. From this collection, we keep the $B-1$ values that differ the most from the remainder. So the construction time of EBV is $O(N \log B)$. Two heaps of B elements each are sufficient.

3 Problem Statement

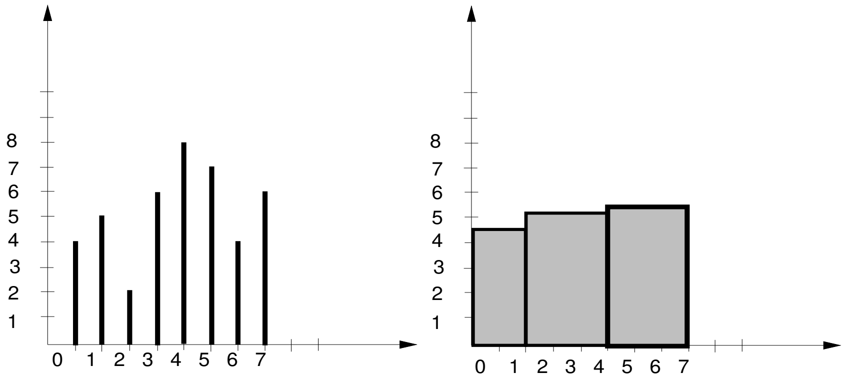
We are given a set of data points $X = \{x_1, \ldots, x_K\}$ where each $x_i, 1 \leq i \leq K$, is an integer drawn from some bounded range Δ.[2] We can enumerate all the values in the range Δ, and for each value A_j in this enumeration, we can determine a count of the number of $x_i \in X$ such that $x_i = A_j$. Call this count v_j, and refer to it as the cell occupancy of cell j. Let N, the number of cells, be the number of distinct values A_j in Δ. We can write an N-long vector V with elements v_j to represent completely (in conjunction with a mapping from index j to value A_j) all the information in the given data set X. We assume that the elements of this vector have been sorted according to some *sort parameter* (following [IP95]), typically the numeric values, A_j.

Consider the example data set shown in figure 1(a). There are 42 data points (that is $K = 42$). For instance, each data point could represent the number of days, in a test period, that one of 42 families in some neighborhood logged on to an Internet service. Clearly, each data point takes an integer value in the range $[0, 7]$. Thus the possible values in the range Δ are $A_1 = 0$, $A_2 = 1$, and so on. There are $N = 8$ distinct values in the range, or 8 cells. Now we can count the number of data points in each cell; for instance, there are 4 points with value 0, so cell A_1 is populated with 4

[2] More general assumptions are possible, and are discussed in Appendix B. These issues are not important for the main arguments in our paper, so we will assume that the x_i take values from a dense integer set for the main presentation in this paper.

4 1 3 5 0 4 1 1 3 7 5 2 6 4 4 6 0 5 2 7 3 7 4 7 5 1 7 4 3 5 4 5 0 1 7 6 4 5 0 3 6 3

(a) Data Set



(b) A data sequence (c) A histogram of the sequence

Figure 1: A cell sequence and one possible histogram representation

data points, and we determine $v_1 = 4$. Proceeding similarly, we obtain all eight elements of vector V as shown in figure 1(b).

We will refer to any sequence of contiguous elements of vector V as a segment, and we will use $s_i = (b_i, e_i)$ to denote the ith segment, which includes the elements $(v_{b_i}, \ldots, v_{e_i})$.

We are interested in creating a summary vector, H, for V. Our objective is to select B non-overlapping segments that together include every element in V, and to represent each segment, s_i, with a single summary element h_i.

We specify a reconstruction function, $R(.)$, that maps from the summary vector H of length B, to a sequence of length N. The simplest, and most widely used, reconstruction function is simply the piecewise constant function: for all values A_j within the range of s_i, the corresponding v_j is set to $f_i(h_i)$, some function of h_i and the range of s_i (typically h_i itself).

We also specify an error metric $E_V(H)$ that defines the total error of the approximation. Typically, $E_V(H)$ is $D(V, R(H))$, the distance (using some distance metric, $D()$, such as mean-squared error) between the original vector V and its reconstruction $R(H)$.

We require the error metric to be point-wise additive. That is,

$$E_V(H) = \sum_{i=1}^{B} F_{V,i}(b_i, e_i, h_i). \quad (1)$$

In other words, the total error can be obtained by adding up the errors of the individual segments (buckets), where $F_{V,i}(b_i, e_i, h_i)$ is a function that computes the error for the ith segment of vector V, where b_i is the beginning point of the segment, e_i is the ending point of the segment, and h_i is the i^{th} element of the summary vector H, and the segments are chosen so that $b_1 = 1$, $e_B = N$, and

$b_{i+1} = e_i + 1,\ 1 < i < B$. Some specific error metrics of interest will be introduced in Sections 4.3 and 7.

Problem 1 (Summary Vector) *Given a vector V of length N, a limit B on the length of H, and an error function $E_V()$, choose vector H to Minimize $E_V(H)$.*

In this paper, we focus on histograms, a special case of summary vectors for which $h_i = \frac{1}{e_i - b_i + 1} \sum_{j=b_i}^{e_i} v_i$, $i = 1, \ldots, B$. [3]

$F_{V,i}(b_i, e_i, h_i)$ now no longer needs h_i as an argument because h_i is determined by b_i, e_i and V. Since we will apply the same error metric to each bucket throughout this paper, the subscript i can be dropped. Finally, V is fixed for any given problem we wish to solve. Thus, we shall write $F(b_i, e_i)$ from now on, in place of $F_{V,i}(b_i, e_i, h_i)$. Furthermore, $b_i = e_{i-1} + 1$.

Therefore, the problem reduces to selecting the integer vector $\mathbf{e}$, defined as follows:

$$\mathbf{e} = (e_1, e_2, \ldots, e_{B-1}, e_B) \tag{2}$$

where, $e_B = N$, $e_1 \geq 1$, and $e_{j-1} < e_j,\ 2 < j \leq B$.

Problem 2 (Histogram Bucket Boundary: Minimum Error) *Given a vector V of length N, and an error function $F()$, choose vector* $\mathbf{e}$ *to Minimize $E_V(H)$.*

Figure 1(c) presents one possible histogram representation of the data set introduced in Figure 1(a). Cells 1 and 2 are grouped together and the average value of the cell occupancy of cells 1 and 2 is used to represent their occupancy. Similarly cells 3 to 5 and cells 6 to 8 are grouped together using the average value of their cell occupancies to represent them in the histogram.

4 The Optimal Histogram (OH) Algorithm

4.1 Description of the Algorithm

The OH algorithm is based on dynamic programming. The basic idea is that a segment can be split in two optimally by considering every possible choice of split point, and for each choice computing the error metric for the two sub-segments. If a sub-segment is potentially further split in two, we compute (for each possible choice of sub-segment based on the first split) its error, by splitting it again optimally and adding up the errors from the two parts. One can continue recursively in

[3] We use the average of frequencies to characterize each bucket of a histogram; equivalently, the sum of frequencies could be used instead. There are other histogram variants, where more than one scalar value is stored per bucket. For instance, in some circumstances, it may be appropriate to store the number of cells with non-zero count along with the average. While much of the discussion in the paper can be carried over in a straightforward way to such variants, the discussion is limited in this paper to a single concrete instantiation for lack of space.

this fashion until B buckets have been created. An exhaustive examination of all possible such recursive divisions produces the desired optimum solution. A naive implementation would take time exponential in the number of splits. The trick is to perform the examination of these cases efficiently, and this is what dynamic programming does.

The OH Algorithm is shown in Appendix A. An example of its use follows, in Figure 2. We begin with an overview of the algorithm.

In a pre-processing phase, for every possible segment $s(j,k)$, $j = 1, \ldots, N$, $k = j, j+1, \ldots, N$, the associated error $F(j,k)$ is calculated according to the chosen error metric.

The core of the algorithm consists of B stages. The first stage is based on the observation that, from any intermediate cell j, the optimal (because it is the only) histogram of one bucket covering cells j through N consists of segment $s(j,N)$ alone. Then, stage b, for b from 2 to B, involves synthesizing optimal b bucket histograms to cover cells j through N based on knowledge of optimal $b-1$ bucket histograms for covering cells $j+1$ through N.

Let $restb(j,b)$ represent the cost of b buckets covering cells j through N under an optimal strategy. That is,

$$restb(j,b) \;=\; min_{all\ bucket\ boundaries\ \mid\ e_{B-b}=j-1} \sum_{i=B-b+1}^{B} F(e_{i-1}+1, e_i)$$

for $1 \leq j < N$, $1 < b \leq B$, where by convention $e_0 = 0$. We also observe $restb(j,1) = F(j,N)$. Using Bellman's equation [Bel54], we can write a recursive expression for $restb(j,b)$.

$$restb(j,b) \;=\; min_{j \leq k \leq N}[F(j,k) \;+\; restb(k+1,b-1)] \tag{3}$$

That is, the cost of an optimal completion strategy using b buckets to cover cells j to N is the sum of the cost of the first segment, $F(j,k)$ and the cost of finishing with $b-1$ buckets optimally chosen to cover cells $k+1$ to N.

Let $nextb(j,b)$ denote the last cell included in the first bucket under the optimal strategy that corresponds to $restb(j,b)$. Over all possible choices of k from j to N, $nextb(j,b)$ is set to the value of k that minimizes $restb(j,b)$. $nextb(j,b)$ is the best value for e_{B-b+1}, given that e_{B-b} has been placed at $j-1$.

The initialization (shown in appendix A) to form the basis for this bootstrapping approach comes from observing that $nextb(j,1) = N$, and recalling that $restb(j,1) = F(j,N)$.

The remainder of the algorithm simply uses equation (3) to construct solutions for successively larger numbers of buckets b, from 2 to B. After $restb(j,b)$ and $nextb(j,b)$ are calculated for all j such that $1 \leq j \leq N$ and b such that $1 \leq b \leq B$, the final solution can be extracted bucket by bucket. The first bucket starts at 1 and ends at $nextb(1,B)$. If k is the last cell of the previous bucket, then the next bucket starts at $k+1$ and ends at $nextb(k+1,b^*)$ where b^* is the number of buckets available to cover cells $k+1$ to N. Pseudo-code for all phases of the algorithm are shown in appendix A.

4.2 An Example

Figure 2(a) shows the values of the error function Mean Squared Error for an execution of the algorithm on the sequence of size $N = 8$, introduced earlier in figure 1. There are 36 ($\frac{N(N+1)}{2} = 36$, for $N = 8$) computations of the Mean Square Error function, one for each possible segment.

Figure 2(b) shows the evolution of *nextb* and *restb* computation during the execution for number of buckets, $B = 4$. The lines connecting the boxes link the segments that form the optimal solution for four buckets. Finally figure 2(c) shows the optimal grouping of the data sequence for $B = 4$.

For example, for $B = 4$, the four linked boxes indicate that the four segments that form buckets of the histogram are respectively $s_1(1, 2)$, $s_2(3, 3)$, $s_3(4, 6)$ and $s_4(7, 8)$.

4.3 Choosing an Error Function

The choice of the error function is important, since it influences the buckets formed and determines which properties of the distribution under consideration are preserved in the summary vector. The error metric should be selected based on the intended usage of the histogram.

The *Mean Square Error(MSE)* is defined as follows:

$$F_{MSE}(b_i, e_i) = \sum_{j=b_i}^{e_i} (v_j - h_i)^2 \tag{4}$$

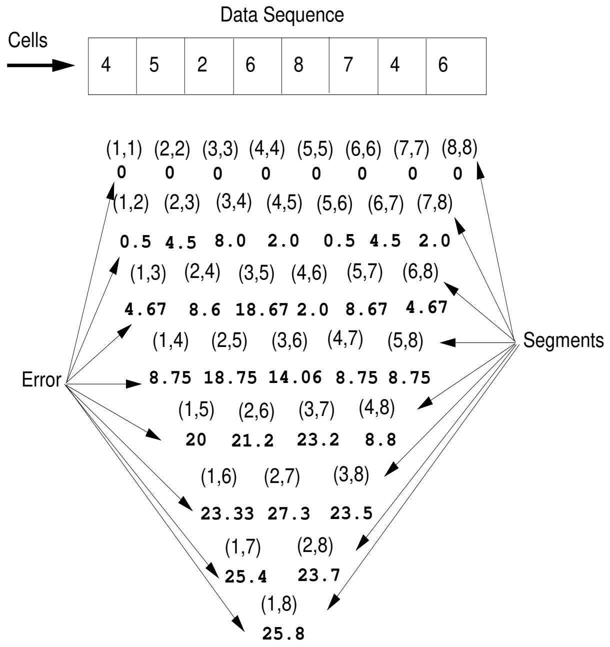
where h_i is the mean of all cell frequencies in bucket i. The mean square error expresses the distance (in the L_2 norm sense) between the two sequences. This is the most natural error metric, and the one that we focus on first. For many applications, this is likely to be the most appropriate metric. For the bulk of our presentation in this paper, we will focus on this as the standard metric. See Sec. 7 for a discussion of other metrics.

Recall that $F(j, k)$ is a function that calculates the estimation error incurred if segment (j, k) is represented as a single bucket of a histogram. This function represents the error cost for the segment, and must be computed for all values of j, k, where $1 \leq j \leq k \leq N$ in the preprocessing phase of the OH algorithm.

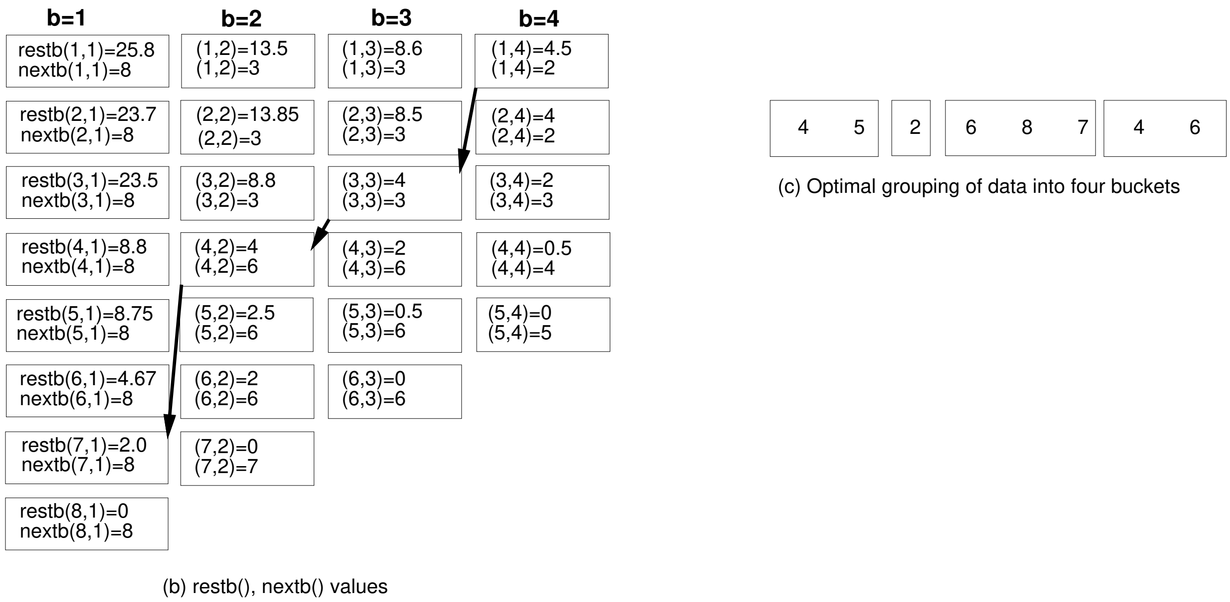
For OH, we need to determine the error $F(j, k), 1 \leq j < k \leq N$ for all $\frac{N(N+1)}{2}$ segments in a preprocessing phase. This error computation can be done efficiently, based on the observation that:

$$F(j, k) = \sum_{i=j}^{k} (v_i - \frac{\sum_{l=j}^{k} v_l}{k - j + 1})^2 = \sum_{i=j}^{k} v_i^2 - \frac{1}{(k - j + 1)} (\sum_{i=j}^{k} v_i)^2 \tag{5}$$

By retaining the partial sums used to compute $F(j, k-1)$, the computation of $F(j, k)$ requires only a few arithmetic operations. Thus, it is possible to compute $F(j, k)$ for all values of j, k with

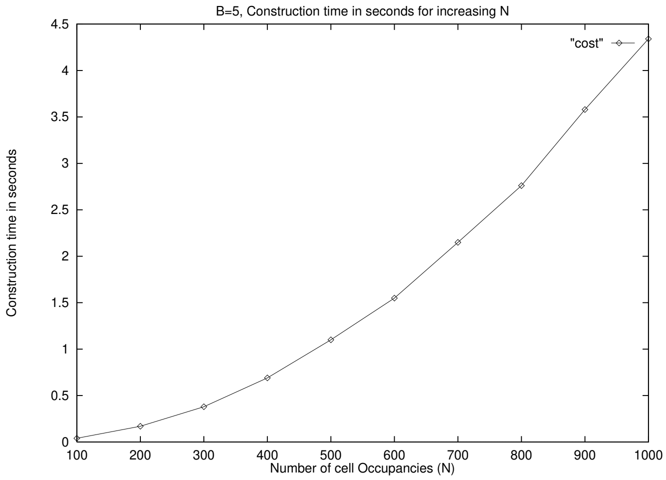


(a) Segments and F values using Mean Squared Error metric for $N = 8$

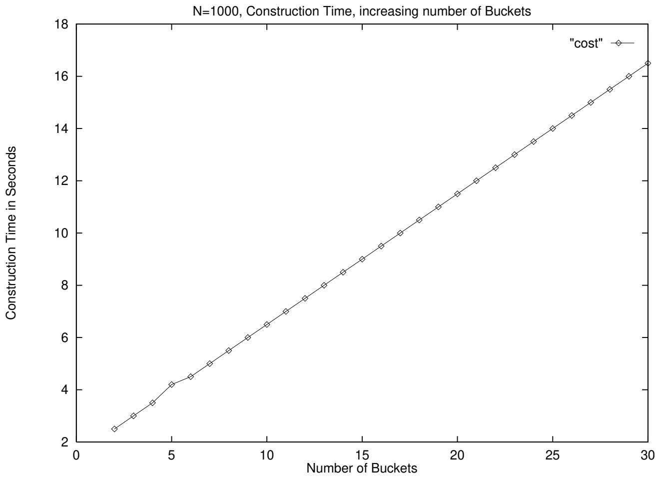


(b) *nextb* and *restb* values (c) Optimal Buckets formed

Figure 2: Example Operation of OH Algorithm on a sequence of size 8, for $B = 4$



(a) Increasing N for B=5



(b) Increasing B for N=1000

Figure 3: Construction times for Optimal Histograms

$O(N^2)$ computation cost.

4.4 Computational Cost of the OH algorithm

The number of operations (mostly additions and assignments) to carry out the calculations in the preprocessing phase of the algorithm is approximately $5N^2$, for MSE and several other error metrics. The inner loop is executed about $N^2/2$ times and involves about 10 operations per iteration.

The computational cost of the induction phase is approximately $BN^2/2$ executions of the innermost loop (which involves one to six additions, comparisons and assignments each time it is executed). The extraction phase requires a negligible amount of computation since it is independent of N and is linear in B. For B and N values of interest, the time spent in the induction phase dominates the computation time. For up to 1000 cells (N=1000) and 100 buckets (B=100), the optimal histogram can be computed in well under a minute on a current workstation (SUN SPARC 20).

The OH algorithm, based on a dynamic programming approach, permits solution of much larger problems than were previously thought to be amenable to exact solution because it is the first algorithm proposed for this problem to have polynomial time complexity in the number of cells. (We emphasize that the time is proportional to the square of the number of cells, or distinct data values, and not the square of the size of the data set, which could be substantially larger).

Figures 3(a) and 3(b) present the total cost of the OH algorithm as a function of the number of cells and as a function of the number of buckets. In the graphs, we include the total cost of of all four phases of the OH algorithm in figure A. Figure 3(a) presents the construction time for $B = 5$ as a function of the number of cells. The construction time increases quadratically, as discussed. As shown in figure 3(b), the construction time increases linearly with the number of buckets, which is consistent with our analytical observations.

The overall memory requirement of OH is $\frac{3}{2}N^2$. This memory requirement can be traded off for computational cost if one does not pre-compute the $F(j,k)$ function values, but instead computes each individual value on demand during the induction phase.

4.5 Satisfying Error Constraints

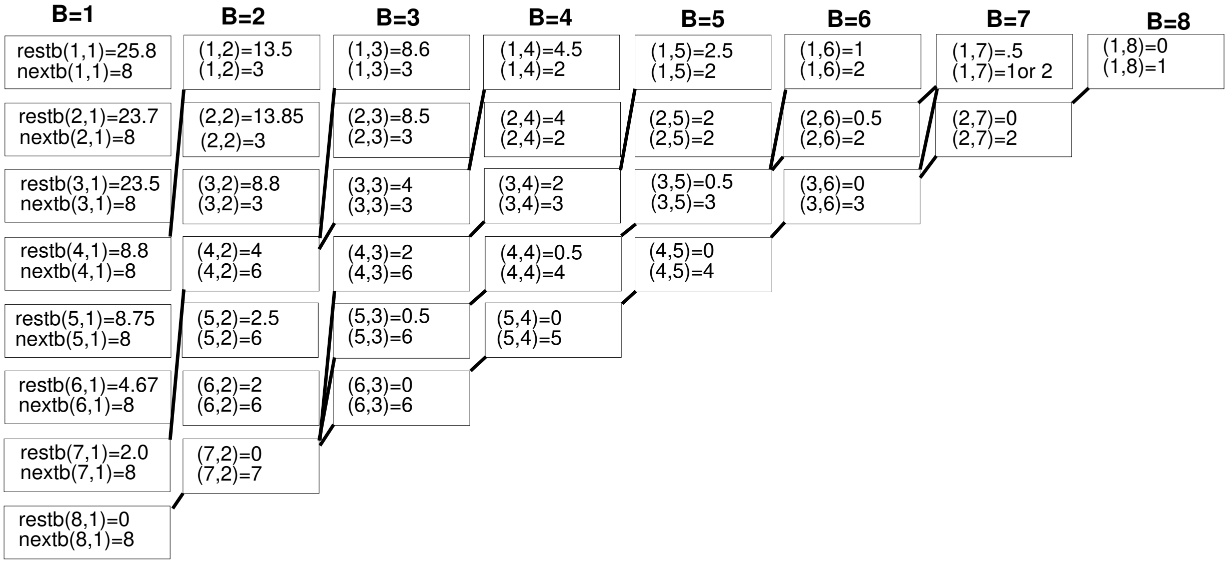
In this section, we elaborate on an alternative formulation of the problem, that is: Given a constraint on the expected error that is tolerable, what representation of the data requires the least space? Here we show how the OH algorithm could be used to solve this problem as well. Formally, the problem is stated as:

Problem 3 (Histogram Bucket Boundary: Minimum Storage) *Given a vector V of length N, and an error function $F()$, choose vector* $\mathbf{e}$ *of smallest length for which the value of $E_V(H)$, computed with equation 1, is no greater than a specified maximum error threshold, $\mathcal{E}$.*

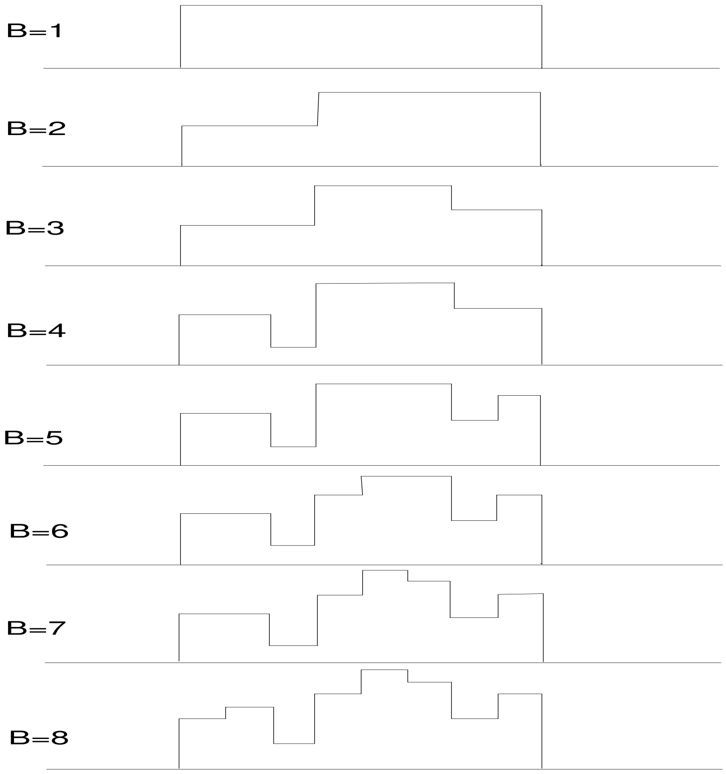
During the induction phase of OH, computation of $restb()$ and $nextb()$ takes place up to a pre-specified number of buckets B. During the iteration in the induction phase, $restb(1,b)$ represents the cost of b buckets covering cells 1 through N under the optimal strategy, for $b = 1, \ldots, B$. This suggests a strategy for satisfying error constraints. Given an error, $\mathcal{E}$, in the data representation that is tolerable, we let the number of buckets, B, vary from 2 to N and with each increment of the value of B, we check if $restb(1,B) \leq \mathcal{E}$. When this becomes true, we can stop the computation, because the optimal data representation satisfying the error constraint consists of B buckets. The solution can be extracted bucket by bucket starting at 1, in exactly the same way the solution of OH is extracted. This leads to an $O(N^3)$ algorithm, in the worst case, for determining the smallest data representation, that satisfies an error constraint.

Figure 4(a) presents the values of $nextb()$ and $restb()$ for the data set of figure 2(a), for B values ranging from 1 to N. Assuming that the error constraint $\mathcal{E} = 2$ is specified, by iteratively computing the values of $restb(1,B)$, $B = 1, \ldots, N$, we can stop the computation at $B = 6$ because $restb(1,6) \leq 2$. By following the arrows connecting the boxes (starting at box with $B = 6$, in this case), the optimal data representation can be extracted. Figure 4(b) presents the optimal buckets formed for every value of $B = 1, \ldots, N$.

The pseudo-code for the OH algorithm, given in appendix A, shows that the outermost loop goes from 1 to B, which could become as large as N in the formulation of the problem being considered here. However, once we have attained the desired error threshold $\mathcal{E}$, there is no point in continuing execution of the outer loop, so one may as well stop, for a total time complexity of $O(N^2B)$, with the only catch being that B is not known at the start of the execution and could be as large as N for sufficiently small $\mathcal{E}$.



(a) $restb()$ and $nextb()$ values for $B = 1, \ldots, N$.



(b) Optimal Buckets Formed for $B = 1, \ldots, N$

Figure 4: Computing optimal data representations satisfying specific error constraints using OH

5 A Heuristic Algorithm for Histogram Construction

While the OH algorithm is quite efficient, its running time may be too long for some applications. In such cases, one desires heuristics that can offer solutions nearly as good as the one offered by OH, but with computational cost close to linear in N, the number of cells.

Greedy heuristics are common in many optimization problems. Given a bucket to be split in two, one can determine the optimal split point for the bucket in time linear in the number of cells in the bucket (assuming that the $F(j,k)$ function values have all been pre-computed). A greedy algorithm could try each candidate split point in turn, and evaluates the sum of the error metrics from the two buckets that result. The split is then placed at the point that minimizes this error. This idea is the basis for the Greedy Bucket Split (GBS) algorithm.

GBS works as follows: We perform a single pass over the cell sequence and identify the $S = \lfloor f * B \rfloor$ maximum differences between successive non-zero cell occupancies (and their cell indices), where f is a parameter described below. The remaining $B - S - 1$ buckets are formed by greedily splitting the buckets with the largest error metric, each time inserting a bucket boundary by performing a pass over the cells in the bucket to be split and placing the boundary in the position that minimizes the total error in the two newly formed buckets. The worst case complexity of this approach is $O((N-S)^2(B-S))$. However, if the function is uniform over the range of cells, then the time required is only $O((N-S)\log(B-S))$, since each portion of the range will be scanned approximately $\log(B-S)$ times to make $B-S-1$ additional buckets. The parameter f takes a value between 0 and 1, and is used to make a tradeoff between speed and accuracy. For $f = 1$, GBS becomes the MaxDiff heuristic [PIHS96] with low time and space complexity, but significant error. For $f = 0$, GBS has a longer running time, but lower error. In the next Section we will experimentally determine reasonable values for this tuning parameter.

6 Experimental Results

To assess the performance of OH relative to heuristics proposed in the literature, we conducted a series of experiments, which are described in this section. We begin by specifying the data sets used.

6.1 Description of Data Sets

We describe experiments using two real data sets, extracted from the census-year SGI adult data set[4]. The size of the SGI data set is 47MB and it contains records with 14 attributes in total, 6 continuous and 8 nominal. By taking appropriate projections, we constructed the following sets:

[4]This data set is available from www.kdnuggets.com

- D1: A density function on the first attribute, representing the age of the persons in the census database. The resulting data set has 74 cells ($N = 74$).

- D2: A density function on the third attribute (fnlwgt). The resulting data set has 732 cells ($N = 732$).

While we only present performance curves for the two data sets indicated above, these experiments are representative of the relative performance of the histograms for the Mean Squared Error metric on a variety of data sets. Substantially similar results were obtained on synthetic data sets having distributions of Zipfian and Gaussian forms, as well as on some very large proprietary AT&T data sets.

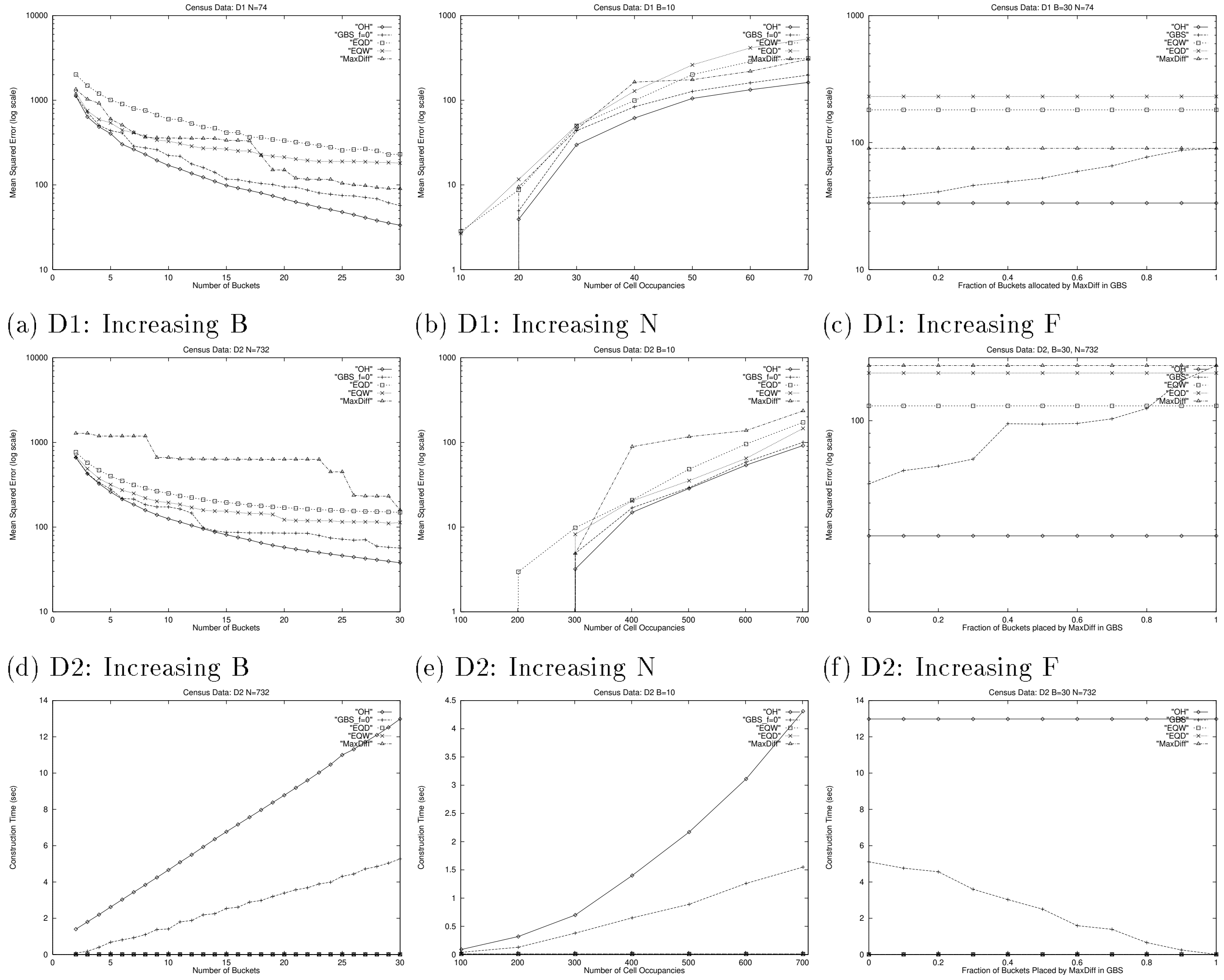
6.2 Performance Comparison

For each of the two data sets, the accuracy with respect to mean squared error was compared for histograms produced by the Optimal Histogram algorithm (OH), and our Greedy Bucket Split heuristic (GBS), and several heuristics previously proposed in the literature. We also examine the tradeoffs involved in the choice of the f parameter in GBS. Previous proposals used for comparison include the Maximum Difference heuristic (MaxDiff) [PIHS96], the End-Biased VOpt heuristic (EBV) [IP95], the equi-width heuristic (EQW), and the equi-depth heuristic (EQD).

We found EBV to be extremely inaccurate, since it only records the B highest frequencies, and makes the uniformity assumption for the rest. For our data sets, with many cells of non-zero occupancy, this is not a good strategy, and performs scarcely better than just making a uniformity assumption across the entire data set, keeping no histogram at all. We have not included EBV in most of our figures so that the scale could be kept at a value at which one can better see the difference between the other heuristics.

Figure 5(a) presents the accuracy, using the MSE error metric, of the algorithms, for data set D1 as a function of the number of buckets. As expected, the accuracy of all the techniques improves as the number of buckets increases. However, this improvement rapidly flattens out for EQD and EQW, whereas it does not for the others. In consequence, both EQW and EQD histograms start by being close to OH with respect to MSE for a small number of buckets, but end up being five and six times worse as the number of buckets increases. The accuracy of MaxDiff histograms is between that of EQW and EQD for a small number of buckets, but is better than both EQW and EQD as the number of buckets increases. However, GBS and OH consistently outperform MaxDiff by a wide margin.

Figure 5(d) presents result of the same experiment with data set D2. The results are similar, except that MaxDiff appears to be even more inaccurate. GBS continues to track the performance of OH closely. The gulf between OH and GBS on the one hand, and the other heuristics on the other hand, widens as the number of buckets is increased.



(a) D1: Increasing B (b) D1: Increasing N (c) D1: Increasing F

(d) D2: Increasing B (e) D2: Increasing N (f) D2: Increasing F

(g) D2: CT Increasing B (h) D2: CT Increasing N (i) D2: CT Increasing F

Figure 5: OH, GBS, EQW, EQD, MaxDiff, for D1 and D2, using the Mean Squared Error metric. CT: Construction Time.

Figure 5(b) presents the estimation accuracy of the algorithms as the number of distinct data values is increased, using 10 buckets for all histograms for data set D1. The number of data values is adjusted by selecting an appropriate fraction of the dataset. Again GBS (with f=0) performs nearly as well as OH. Notice that as N increases, there is a sharp increase in inaccuracy for EQW, EQD and MaxDiff, histograms. Essentially the same results are observed with data set D2 for which results of the same experiment are plotted in figure 5(e). We used B =50 buckets for this data set.

Figure 5(c) presents the mean squared error for the algorithms as a function of the fraction of buckets allocated by MaxDiff in GBS (f). We choose $B = 30$ buckets in total to be allocated in this experiment, for data set D1 in order to show the trends between GBS and the other algorithms for a wide range of values. The value of f determines the fraction of those allocated using the MaxDiff heuristic. OH, MaxDiff, as well as, EQW and EQD are independent of f, so their performance curves are simply constant. As the value of f increases (more buckets allocated with MaxDiff), the estimation accuracy of GBS deteriorates, since GBS becomes less effective in accurately placing bucket boundaries. GBS's estimation accuracy matches that of MaxDiff for f=1. Figure 5(f) presents the same tradeoff between estimation accuracy and the value of f for data set D2.

The construction times of all histograms for data set D1 is very small, since D1 has a small number of cells ($N = 74$). Figures 5(g)(h)(i) present the construction times of all histograms, for data set D2 ($N = 732$), for increasing values of B, N and f respectively. EQW, EQD, and MaxDiff histograms have low construction times, and their corresponding curves overlap in the figures. The construction time of GBS histograms is a little more, but still substantially lower than that of OH histograms, yet GBS histograms can approximate the accuracy of OH histograms very well.

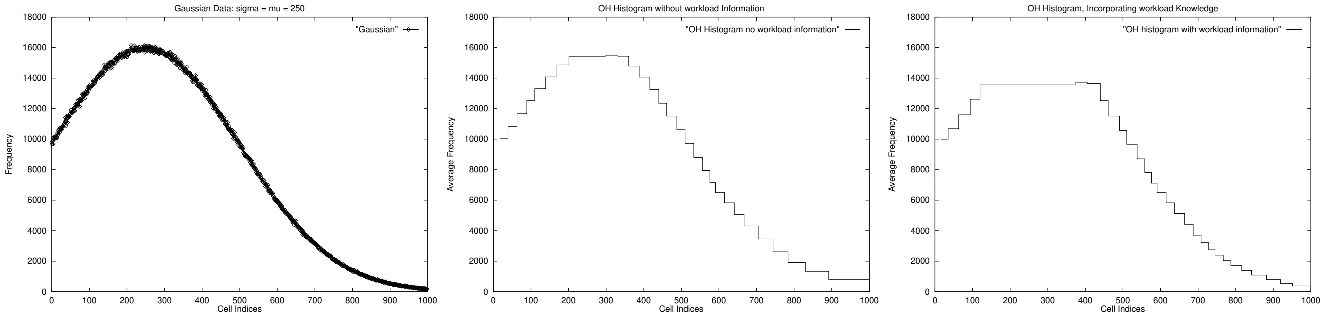
The main lesson from our experiments in varying the fraction of buckets allocated with MaxDiff in GBS is that there is no good "knee" or natural choice of f value at which one can attain a great deal of error reduction for only a small computational penalty. Instead, one has a continuum of choices, corresponding to different tradeoffs.

7 Other Error Metrics

7.1 Join Queries

The quality of a reduced data representation must be measured based on known (or expected) query loads. Sometimes, there is no information available regarding the expected query load, and in this case the best one can hope to do is to minimize the worst-case error, or the average error assuming a uniform distribution of queries (as our MSE metric has done so far).

However, if information is available regarding expected query loads, this must be folded into the error metric since it can affect the resulting optimal histogram quite significantly. To illustrate this



(a) Gaussian Data (b) OH histogram (c) OH histogram with workload info

Figure 6: Incorporating Workload Knowledge

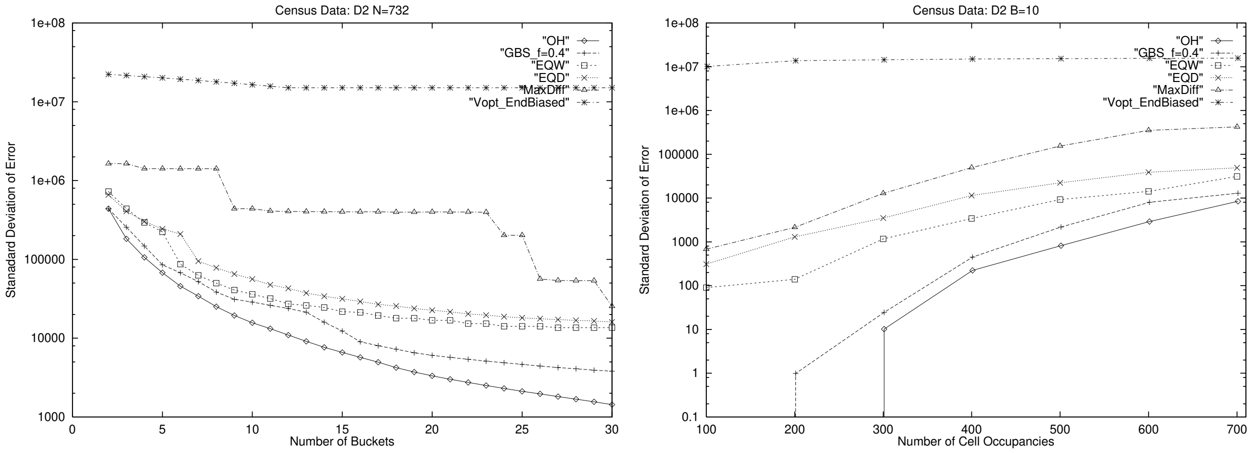
point, consider a synthetic data set drawn from a (truncated) Gaussian distribution and shown in Figure 6(a) along with a histogram with 30 buckets, created optimally using the MSE metric (Figure 6(b)). Suppose we know that outliers are of greatest interest in this data set, and are likely to be queried most often, we could optimize the histogram for a situation where we expect attribute values to be selected by a query inversely proportional to the frequency of occurrence of the attribute value. For this query mix, a histogram created optimally to minimize the weighted mean square error looks like Figure 6(c). Observe how the buckets at the left have been expanded greatly, to allow more buckets to be used to represent data in the tail of the distribution, to the right.

In fact, we know that many data distributions in real-life are extremely skewed. We also know that access patterns to these data are extremely skewed. Therefore, it is important to take this skew into account when choosing histogram boundaries. OH can easily accommodate such information by weighting the optimization metric. All one has to do is to factor in the weights during the pre-processing phase when the $F(i, j)$ function values are calculated. The rest of the algorithm remains unchanged. None of the other techniques can, and in fact no one has looked at this issue at all in the existing literature. This is not surprising, since no single heuristic can possibly generate histograms similar to both Figure 6(b) and (c), for example.

7.1.1 Estimating Join Query Size

Even in the absence of knowledge of the query mix, it is sometimes possible to take into account knowledge of query type to develop a superior error metric. For instance, join query result sizes depend upon the product of the counts for the corresponding attribute values. It has been shown in [IP95] that the error in the estimate of the query result size is minimized when the variance of the counts of cell occupancies in each bucket is minimized. This observation leads to the following metric.

Let p_i denote the number of non zero cell occupancies in bucket i. Thus, $p_i = \sum_{j=e_{i-1}+1}^{e_i} \delta(v_j)$,



(a) D2: Increasing B

(b) D2: Increasing N

Figure 7: OH, GBS, EBV, EQW, EQD, MaxDiff, for D2, using the Join Query Metric

where $\delta(v_j)$ is one if $v_j > 0$ and zero otherwise.

The **Join Query Metric** (JQM) is defined as follows:

$$F_{JQM}(b_i, e_i) = \frac{p_i}{P_i - 1} \sum_{l=b_i}^{e_i} \delta(v_l)(v_l - h_i)^2 \tag{6}$$

In the following section we present experimental results, using this metric to construct histograms for estimating the result size of (self-)join queries. Following [IP95] we use the standard deviation of error, to compare the histograms. The standard deviation of error is defined as, $\sqrt{(}E[(S - S')])$, where S is the actual join result size and S' is the join result size predicted by the histograms.

7.1.2 Experimental Results

The algorithms compared are OH, GBS, MaxDiff, EQW, EQD, and EBV.

Figure 7(a) and 7(b) present the accuracy of the algorithms for D2, with the number of buckets increasing (figure 7(a)) and the number of cells increasing (figure 7(b)). It is interesting to notice in figure 7(a), that the accuracy of both OH and GBS improves sharply as the number of buckets increases, but for other histograms, accuracy levels off as the number of buckets increases. For this data set and for a small number of cells (figure 7(b)), OH and GBS are able to predict the join result size almost exactly. EQW and EQD histograms perform much better than MaxDiff and EBV for this data set. We present in figure 7(c) the construction time of all histograms as the number of buckets increases. It is evident from the figure that GBS (f=0.4) has much lower construction time than the optimal algorithm, and yet GBS is able to come close to OH's accuracy in prediction.

Results on D1 and the other data sets were similar but are not presented here for lack of space.

7.2 Range Queries

7.2.1 Estimating Range Query Size

The metrics discussed thus far have focused on the error in reconstruction of a single point (frequency count for a single attribute value cell). However, queries are frequently specified with range selections. Buckets that are completely included in a range introduce no error at all. At most one bucket at each end of the range is partially included, and we could have an error in estimating the total count for the values in the partial bucket included. In consequence, a pair of adjacent cells with counts much higher and much lower than the average for the bucket, may not introduce much error since the pair of cells together will be included in most range queries – an error in induced only when one of the cells is included in some range and the neighboring cell is not. In comparison, a gradual "drift" of frequencies in a bucket could be significant since ranges that include the right end of the bucket would consistently be over- (under-) estimated whereas ranges that include the left end of the bucket would be under- (over-) estimated.

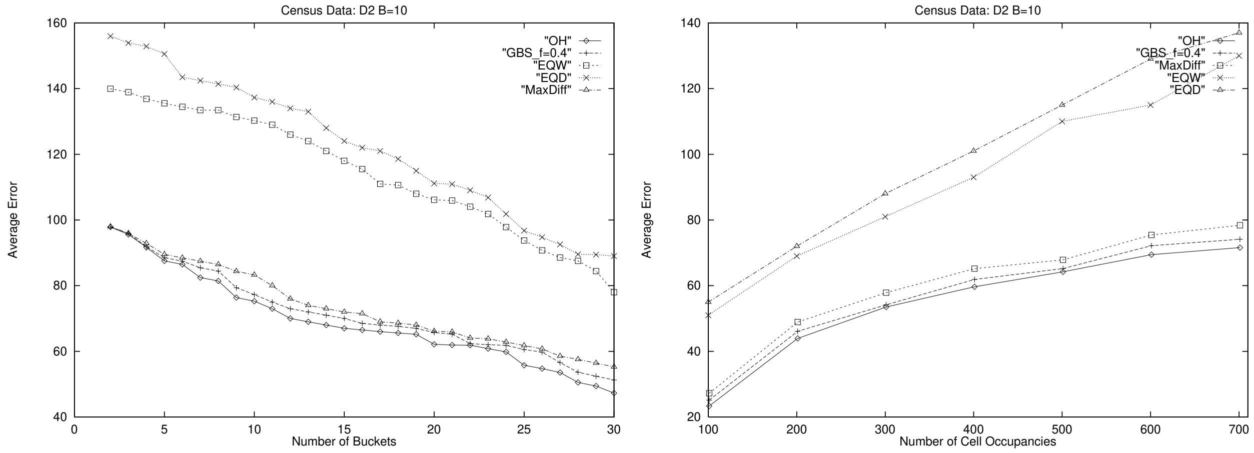
If a range query mix is specified, then this mix can be taken into account in computing the cost metric. For each $F(i,j)$ to be evaluated, determine the boundaries of the range queries that lie between i and j. For each of these queries, there is a corresponding error that can be determined (depending on whether the left, right, or both boundaries of the query range are included) and weighted by the query probability. Once this has been done in the pre-processing step, the OH algorithm can run normally. The results can be stunningly different than for the MSE metric. For example, consider a distribution of an age attribute, as in data set D1. Suppose we know that typical range queries ask about "even" ranges: *i.e.* about 30-40 but not about 32-44. In other words, with high probability range queries have boundaries at values that are multiples of 10. Then an optimal histogram on this data set will also have bucket boundaries only at multiple of 10. In the absence of query probability information, one can still observe in general for range queries that in the estimation of result sizes for range queries, the error in the frequency estimate for cells in any bucket gets weighted by the range extent of the bucket. This error is limited by minimizing the variance of the "area" inside buckets, where the area of cell i is defined by $a_i = (j-i)v_i$, $\forall i,\ 1 \le i \le N,\ \ \delta(v_i) > 0$, and j denotes the first index, $i < j$ such that $\delta(v_j) > 0$.

The **Range Query Metric** (RQM)[PIHS96] is defined as follows:

$$F_{RQM}(b_i, e_i) = \frac{p_i}{p_i - 1} \sum_{l=b_i}^{e_i} \delta(v_l)(a_l - a_i)^2 \tag{7}$$

where a_i represents the average area associated with values in bucket i.

In the following section we use the RQM to construct histograms for predicting the result of range queries. Following [PIHS96] we measure the accuracy of range query prediction for a set Q



(a) D2: Increasing B (b) D2: Increasing N

Figure 8: OH, GBS, MaxDiff, EQW, EQD, for D2 using the Range Query Metric

of L queries as:

$$E = \frac{100}{L} \sum_{q \in Q} \frac{|S_q - S_q^{'}|}{S_q} \tag{8}$$

where L total number of queries (1000 in our experiments), S_q and $S_q^{'}$ the actual and estimated size of the query result, respectively.

7.2.2 Experimental Results

We compared the estimation accuracy of OH, GBS, EQD, EQW and MaxDiff, using the Range Query Metric. EBV is not included in these experiments since it is highly inaccurate for range query predictions, due to its uniformity assumption.

The starting point of each query range is uniformly distributed over the attribute domain, and the ending point is uniformly distributed between the starting point and the end of the attribute domain.

Figure 8(a) and 8(b) present the trends in accuracy of the three algorithms for data set D2, as the number of buckets increases (Figure 8(a)) and as the number of cells increases (Figure 8(b)). GBS (with f=0.4) follows OH in accuracy very closely. Other heuristics are not too far behind – the averaging effect kicks in with large ranges where underestimated counts for one cell are compensated for by overestimated counts for another.

8 Conclusions

In this paper, we introduced the OH algorithm, which can be used to find an optimal placement of bucket boundaries for a histogram in time proportional to the square of the number of distinct data values being considered, which is typically much smaller than the data set size. Our algorithm is applicable to a variety of error metrics to be minimized, and can effectively take into account any knowledge available regarding the expected query mix. This algorithm also provides a solution to the dual problem of identifying the histogram requiring the least storage space to achieve an estimation error that is within a specified bound.

For ease of exposition, we have focused in this paper on value histograms, in the example and explanation. However, the OH algorithm is equally applicable if frequency is used as the sort parameter, leading to a serial histogram, or even if some other sort parameter is used. Thus, in particular, the OH algorithm can be used to find the optimal histogram (for variance optimality) sought by Ioannidis and Poosala [IP95] in $O(N^2)$ time.

We also presented an inexpensive heuristic, GBS, that can approach the accuracy of the OH algorithm and yet has lower construction costs. We presented experimental results based on real data sets, showing that our algorithms outperform previous approaches in many situations of interest.

In our future work we will explore the efficient construction of multidimensional histograms.

Acknowledgments

We are grateful to Christina Christara, Carl de Boor, and Andrew Odlyzko for generously giving us their time to educate us regarding the related numerical analysis literature. We are grateful to Nikolaos Sidiropoulos for helping us with his expertise in dynamic programming. We also thank Stefan Berchtold and Flip Korn for giving us useful comments on earlier drafts of this manuscript.

References

[Bel54] R. E. Bellman. The Theory of Dynamic Programming. *Bull. Amer Math Soc. Vol 60*, pages 503–516, 1954.

[CdB72] S. D. Conte and Carl de Boor. *Elementary Numerical Analysis: An algorithmic approach.* McGraw Hill Publishing Company, 1972.

[CR94] C. M. Chen and N. Roussopoulos. Adaptive Selectivity Estimation Using Query Feedback. *Proceedings of ACM SIGMOD 1994, Minneapolis, Minnesota*, pages 161–172, May 1994.

[dB97] Carl de Boor. personal communication. 1997.

[FD81] David Freedman and Persi Diaconis. On the Historgram as a density Estimator: L_2 Theory. *Zeitschrift fur Wahrscheinlichkeitstheorie*, 57:453–476, 1981.

[GES85] T. Gasser, J. Engel, and B. Seifert. Non parametric density estimation. *Ann. Stat.*, September 1985.

[HS92] P. Haas and A. Swami. Sequantial Sampling Procedures for Query Size Estimation. *Proceedings of ACM SIGMOD, San Diego, CA*, pages 341–350, June 1992.

[IC93] Y. Ioannidis and S. Christodoulakis. Optimal Histograms for Limiting Worst-Case Error Propagation in the Size of Join Results. *ACM Transactions on Database Systems, Vol. 18, No. 4*, pages 709–748, December 1993.

[Ioa93] Y. Ioannidis. Universality of Serial Histograms. *Proceedings of VLDB, Dublin Ireland*, pages 256–277, August 1993.

[IP95] Y. Ioannidis and Viswanath Poosala. Balancing Histogram Optimality and Practicality for Query Result Size Estimation. *Proceedings of ACM SIGMOD, San Hose, CA*, pages 233–244, June 1995.

[Koo80] R. P. Kooi. The Optimization of Queries in Relational Databases. *PhD Thesis, Case Western Reserve University*, September 1980.

[LNS90] R. J. Lipton, J. F. Naughton, and D. A. Schneider. Practical Selectivity Estimation Through Adaptive Sampling. *Proceedings of ACM SIGMOD, Atlantic City, NJ*, pages 1–11, May 1990.

[MCS88] M. V. Mannino, Paicheng Chu, and Thomas Sager. Statistical Profile Estimation in Database Sysems. *ACM Computing Surveys, Vol. 20, No. 3*, pages 191–221, September 1988.

[MD98] M. Muralikrishna and D. J. DeWitt. Equi-depth histograms for estimating selectivity factors for multidimensional queries. *Proceedings of ACM SIGMOD, Chicago, IL*, pages 28–36, June 1998.

[PIHS96] V. Poosala, Y. Ioannidis, P. Haas, and E. Shekita. Improved Histograms for Selectivity Estimation of Range Predicates. *Proceedings of ACM SIGMOD, Montreal Canada*, pages 294–305, June 1996.

[PSC84] G. Piatetsky-Shapiro and C. Connell. Accurate Estimation of the Number of Tuples Satisfying a Condition. *Proceedings of ACM SIGMOD, Boston MA*, pages 256–276, June 1984.

[SLRD93] Wei Sun, Yibei Ling, Naphtali Rishe, and Yi Deng. An Instant and Accurate Size Estimation Method for Joins and Selections in a Retrieval Intensive Environment. *Proceedings of ACM SIGMOD, Washington, DC*, pages 79–88, June 1993.

[Weg83] E. J. Wegman. Density estimation. In S. Kotz and N. Johnson, editors, *Encyclopedia of Statistical Sciences*, volume 2. Wiley, 1983.

A The OH Algorithm

Objective: Find an Optimal histogram with respect to the estimation error specified for the given sequence of values and number of buckets.

Inputs: N - Number of cells

$v_1, \ldots, v_N$ - cell occupancies

B - Number of Buckets permitted.

Preprocessing Phase:

for j from 1 to N

  for k from j to N

    compute $F(j,k)$ {Compute the error of segment *s(j,k)*}

  endfor

endfor

Initialization:

for j from 1 to N

  $restb(j,1) = F(j,N)$

  $nextb(j,1) = N$

endfor

Induction Phase:

for b from 2 to B

  for j from 1 to N

    for k from j to N

      $kcost = F(j,k) + restb(k+1, b-1)$

      if $k = j$ or $kcost < restb(j,b)$ then

        $restb(j,b) = kcost$

        $nextb(j,b) =$ k

      endif

    endfor

  endfor

endfor

Extraction Phase:

$e_0 = 0$

for b from 1 to B

  $e_b = nextb(e_{b-1}+1, B+1-b)$

endfor

Result: return $(e_1, e_2, \ldots, e_{B-1}, e_B)$ where the jth buckets covers cells e_{j-1}+1 to e_j (and $e_0 = 0$).

B Non-integer DataSets

While the presentation in the paper was restricted to data sets whose elements took integer values over a (dense) bounded range, histogramming techniques, and our algorithms, are applicable to many other data sets as well.

B.1 Categorical Attributes

If we are dealing with some categorical attribute, then we develop a mapping from the category label to a unique integer. One very simple such label is simply to list, in a mapping table, all the category labels that appear in the data set, and associate an index integer with each row in the table.

B.2 Attributes Taking Fractional Values

Suppose each x_i, $1 \leq i \leq K$ is drawn from some range D of rational numbers. Since D includes only rational numbers, the number of possible different values in D is countable (even if infinite) and it is possible to enumerate D. For each value A_j in this enumeration, we can determine a count of the number of $x_i \in X$ such that $x_i = A_j$. Call this count w_j, and refer to it as the cell occupancy of cell j.

Since there could be infinitely many values A_i in the range D, the vector W of all w_i could be infinitely long. Let the length of this vector be $\mathcal{N}$. However, we know that there are exactly K elements in X, which could take at most K distinct values. Therefore, at most K of the w_i are non-zero, and all the rest are zero. Capturing only these non-zeroes in a vector, we write $V = v_l, 1 \leq l \leq N$, for some $1 \leq N \leq K$ as the *compact frequency vector* of the data set X, where v_l is shorthand for a non-zero w_{i_l}. That is $V = \mathit{compact}(W)$, where the function $\mathit{copact}()$ has the obvious meaning.

Now, our problem becomes one of solving a Semi-Markov Decision Process, rather than a Markov Decision Process in choosing where to place bucket boundaries. Essentially the same dynamic programming solution applies.

A more practical question is whether one can expect any sort of reasonable reconstruction over such a sparse value set, and the answer essentially is that one cannot. For example, suppose we recorded at what time a telephone call started down to a microsecond granularity, created a histogram of telephone call start times, and then asked how many telephone calls started at some time specified to the microsecond. Even with millions of telephone calls recorded per day, the answer to our query would be (close to) zero since no telephone calls start in most microseconds. But if we picked a microsecond for our query in which a telephone call did start, then the zero answer obtained would be incorrect. The point is that such precise queries are not usual in a sparse value range. Instead, one would expect most queries to specify a range themselves, and to ask about how many telephone calls started within some specified range of times. In this case, one is best off thinking about the problem in continuous terms, as in the next subsection.

B.3 Attributes Taking Real Values

The set of data values X, with each element being real number, can be viewed as a set of realizations from some probability distribution that we cannot observe directly. We can use any desired statistical technique to obtain the best possible guess at this probability distribution. For instance, one could use the kernel technique [Weg83].

Given a compressed representation, its reconstruction error is then computed in terms of the difference between a reconstructed probability distribution and the original probability distribution. Instead of our error metric being point-wise additive, we now have it be integrable, with an analytic representation over any range. A dynamic programming algorithm in the style of OH can then be applied.

Alternatively, one could quantize the value range into discrete "cells", each small enough that we guarantee query ranges can always be made to coincide with cell boundaries. These discrete cells can now be numbered with an integer index and the techniques presented in the paper applied directly.