# Optimal Histograms with Quality Guarantees


**H. V. Jagadish**
AT&T Labs
jag@research.att.com

**Nick Koudas**
University of Toronto
koudas@cs.toronto.edu

**S. Muthukrishnan**
Bell Labs
muthu@research.bell-labs.com

**Viswanath Poosala**
Bell Labs
poosala@research.bell-labs.com

**Ken Sevcik**
University of Toronto
kcs@cs.toronto.edu

**Torsten Suel**
Bell Labs
suel@research.bell-labs.com



## Abstract

Histograms are commonly used to capture attribute value distribution statistics for query optimizers. More recently, histograms have also been considered as a way to produce quick approximate answers to decision support queries. This widespread interest in histograms motivates the problem of computing histograms that are *good* under a given error metric. In particular, we are interested in an efficient algorithm for choosing the bucket boundaries in a way that either minimizes the estimation error for a given amount of space (number of buckets) or, conversely, minimizes the space needed for a given upper bound on the error. Under the assumption that finding optimal bucket boundaries is computationally inefficient, previous research has focused on heuristics with no provable bounds on the quality of the solutions.

In this paper, we present algorithms for computing optimal bucket boundaries in time proportional to the square of the number of distinct data values, for a broad class of optimality metrics. This class includes the *V-Optimality* constraint, which has been shown to result in the most accurate histograms for several selectivity estimation problems. Through experiments, we show that optimal histograms can achieve substantially lower estimation errors than histograms produced by popular heuristics. We also present new heuristics with provably good space-accuracy tradeoffs that are significantly faster than the optimal algorithm. Finally, we present an enhancement to traditional histograms that allows us to provide quality guarantees on individual selectivity estimates. In our experiments, these quality guarantees were highly effective in isolating outliers in selectivity estimates.





**Proceedings of the 24th VLDB Conference**
**New York, USA, 1998**


## 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 can be used to answer various 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 error due to the information lost when the data was summarized. This error can be measured according to some appropriate metric such as the maximum, average, or mean squared error of the estimate.

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

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 that manages a large table containing records of telephone calls indicating the length of each call, and that uses this table to answer queries about the statistics of call duration. We can save much time and space by summarizing this information in a histogram of frequencies of occurrence for calls with lengths in different ranges, at the cost of some error in the answers provided.

In this context, the following question arises:

(1) Given the types of queries 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?

A complementary formulation is the following:

(2) Given the types of queries and a constraint on the

expected error that is acceptable, what representation of the data requires the least space?

In the context of histograms, the representation of the data is obtained by partitioning it into subsets called *buckets*, and the problem becomes that of identifying the best placement of the bucket boundaries. (Section 3 presents a more formal description of these issues.)

**Our Contributions.** The bulk of this paper is devoted to addressing the first of the two questions (1) and (2) above, where the form of summarized data representation is a histogram. In Section 4, we present an algorithm for computing optimal histograms based on dynamic programming. Our main result is that it is possible to obtain an optimal solution for a broad range of error metrics in time that is quadratic in the number of distinct values of the attribute being considered, and linear in the number of buckets being used. In addition, we present an improved version of this algorithm that can compute optimal histograms in a few minutes for data distributions over tens of thousands of values. We also present an even faster algorithm that determines a provably close to optimal histogram, and that combines the dynamic programming-based approach with an additional partitioning heuristic.

In Section 5, we present the results of a set of experiments that compare our algorithms to the previously known heuristics in terms of accuracy and running time.

In Section 6, we show how the second question (the *dual* problem) can be addressed using very similar techniques. We also present an alternative approach that takes essentially linear time to compute a histogram that is provably close to optimal.

Furthermore, we address a known limitation of histograms: Current histogramming techniques do not provide any quality guarantees for individual estimates. This is unlike, say, random-sampling techniques, which usually provide probabilistic error bounds on their estimates [OR86]. This problem has not been significant until recently because histograms have mostly been employed within optimizers, where there is no need to report the errors. However, this is no longer the case in applications such as *approximate query answering systems* and *query profilers*, which provide estimates directly to the user.

In such applications, the confidence of a user is critically dependent upon the provision of error bounds (qualities) for the estimates. Individual estimates using histograms may differ widely in their quality, as some regions of the distribution may be much more difficult to represent than others. In that case, queries involving attribute values from these regions may be significantly less accurate than queries accessing other buckets. If each selectivity estimate was accompanied by some *quality guarantee*, then we could successfully identify "outliers" in the estimates, and flag them as unsuitable for further processing.

In Section 7, we propose an enhancement to histograms and show how it can be used to provide quality guarantees on selectivity estimates for equality and range queries. Our experiments indicate that the enhancement gives significantly better quality guarantees for individual queries than the trivial worst-case bound.

Though our algorithms are very general and work for a large class of error metrics, for the sake of concreteness we present most of our results using a specific error metric, the *Sum Squared Error* (SSE). This metric was chosen because it plays an important role in selectivity estimation - it is identical to the *V-Optimality constraint* which has been shown to minimize the average selectivity estimation error for equality-join and selection queries [IP95].

In Section 8, we briefly discuss how our algorithms can be used to generate optimal histograms for other error metrics such as the metric arising in the context of selectivity estimation for range queries, join queries, and metrics that incorporate knowledge about the query workload. Finally, Section 9 offers some concluding remarks. Due to space constraints, many of the proofs and more general forms of our results, as well as some of the experimental results, could not be included in this paper; the details can be found in [JKS98, MPS98].

## 2 Related Work

The problem of approximating a given data distribution has received considerable attention in several scientific communities. In numerical analysis, the problem has been studied in the context of approximating a given function in a piecewise fashion by a class of simple functions such as polynomials of some fixed degree [CdB72]. However, not much attention has been given to the number of parameters or amount of space required for the representation. Finding an optimal set of "breakpoints" for a piecewise polynomial (or even linear) approximation is believed to be hard due to the continuous domain and the non-linearity of the problem space [dB97].

In statistics, the problem has been posed in connection with non-parametric density estimation as that of constructing a histogram of a given data distribution. But again the effort has focused on minimizing the error without taking space constraints into account [GES85].

In the database community, the problem has been studied in the field of query optimization and more specifically in the context of selectivity estimation for relational operators. Several techniques have been proposed [MCS88], including *histograms* [Koo80, SC84, Ioa93, IP95], *sampling* [OR86, LNS90, HS92], and parametric techniques. Histograms are the most commonly used form of statistics in practice (e.g., they are used in DB2, Oracle, and Microsoft SQL Server) because they incur almost no run-time overhead and are effective even with a very small amount of storage space. Several types of histograms have been proposed and evaluated experimentally in terms of their accuracy, including *EquiWidth* and *EquiHeight* [Koo80, SC84], *MaxDiff*, *Compressed*, *End-Biased* (EBV), and *V-Optimal* histograms [IP95, PIHS96]. A formal taxonomy of histograms was proposed in [PIHS96]. The V-Optimal histograms have been shown to minimize the average error for several selectivity estimation problems [IP95], but no efficient algorithms for constructing them have been proposed.

We are not aware of prior work on the dual question of minimizing space given a bound on the acceptable error, or on generating quality guarantees using histograms.

# 3 Definitions and Problem Formulation

In this section we define histograms and formulate the various problems studied in this paper.

Consider a relation $R$ containing an integer valued attribute $X$.[1] The *value set* $V$ of $X$ is the set of values of $X$ that are present in $R$. For each $v \in V$, the *frequency* $f(v)$ is the number of tuples $t \in R$ with $t.X = v$. We assume that the elements of $V$ have been sorted according to some *sort parameter* (following [PIHS96]), most commonly according to the numeric values of the $v_i$, i.e., $V = \{v_i \mid 1 \leq i \leq N\}$ where $i < j$ iff $v_i < v_j$. Given this ordering, and using $f_i = f(v_i)$, the *frequency vector* of $X$ is the ordered set of frequencies $F = \{ f_1, f_2, \ldots, f_N \}$.

A *histogram* of data distribution $X$ is constructed by partitioning the frequency vector $F$ of $X$ into $B$ ($\geq 1$) intervals called *buckets*, and approximating the frequencies and values in each bucket in some succinct fashion, as explained further below. The result is an approximate data distribution that can be used in place of the actual distribution, say, in selectivity estimation. Of course, the accuracy of any operation performed using the histogram depends on the accuracy of the approximation, which is determined by two factors, the partitioning technique employed for grouping values into buckets and the approximation technique employed within each bucket.

Several techniques for the approximation within a bucket have been studied in the literature. The frequencies in a bucket are most commonly approximated by their average. The value domain is approximated either by a continuous distribution in the bucket range [Koo80] or by uniformly placing $m$ values in the bucket range, where $m$ is the total number of distinct values of $V$ grouped into that bucket [PIHS96]. The latter approach has been experimentally shown to be more accurate for several estimation problems [PIHS96].

The main focus of this paper, however, is on the partitioning task. We are interested in computing a histogram of $F$, i.e., a summary vector $H$ of length $B << N$ that approximates $F$. To do so, we partition $F$ into $B$ non-overlapping intervals $I_i$, $0 < i \leq B$, and represent each interval $I_i$ by a single summary element $h_i$ (say, the average). We specify a reconstruction function, $R_H$ that uses $H$ to return for each element $v_i$ in $V$ an estimate of its frequency $f_i$. The simplest, and most widely used, reconstruction function is simply the piecewise constant function: For all values $v_j$ within bucket $I_i$, the estimate $R_H(v_j)$ is set to $h_i$.

In order to evaluate the accuracy of a histogram, we specify an error metric $E(H)$ that defines the total error of the approximation. Typically, $E(H)$ can be represented as $\mathcal{D}(\mathcal{F}, \mathcal{R_H}(\mathcal{V}))$, the distance (using some distance metric, $\mathcal{D}$, such as the mean squared error) between the original vector $F$ and its reconstruction $R_H(V)$.

We now define the main problem that we consider in this paper.

**Definition 1 (Space-bounded histogram problem) :** *Given a vector $F$ of length $N$, a limit $B$ on the length of $H$, and an error metric $E()$, find the histogram $H$ that minimizes $E(H)$.*

The dual problem is as follows.

**Definition 2 (Error-bounded histogram problem) :** *Given a vector $F$ of length $N$, a limit $\epsilon$ on the rror, and an error metric $E()$, find the histogram $H$ of smallest length for which $E(H)$ is at most $\epsilon$.*

On of the most natural choices for the bucket approximation is to choose $h_i = AVG(b_i, e_i), i = 1, \ldots, B$,[2] where $b_i$ and $e_i$ are the end points of the

[1] More general assumptions are possible, and are discussed in [JKS98]. To simplify the presentation, we will assume that $X$ takes only integer values.

[2] There are other possible choices for the $h_i$, such as the geometric mean of the bucket frequencies. It may also sometimes be appropriate to store more than one scalar value per bucket. For instance, one may store the number of cells with non-zero count along with the average. Much of the discussion in the paper can be carried over to such variants in a straightforward manner.

$i$th interval and

$$AVG([b_i, e_i]) = \frac{\sum_{b_i \le k \le e_i} F[k]}{(e_i - b_i + 1)}.$$

Having fixed the choice of the $h_i$, the problem of determining $H$ reduces to that of finding the boundaries of the $B$ buckets.

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

A common metric for measuring the difference between two distributions is the *Sum Squared Error* (SSE), which is defined as follows. For any interval $[a, b]$,

$$SSE([a,b]) = \sum_{k=a}^{k=b} (F[k] - AVG([a,b]))^2$$

The Sum Squared Error is one of the most natural error metrics and the one that we focus on in this paper, though most of our results extend to all decomposable metrics. The space-bounded histogram with SSE as error metric is known in the literature as the *V-Optimal histogram* [IP95]. In the following, we refer to this case as the space-bounded $V$-Optimal histogram, and to its dual (Definition 2) as the error-bounded $V$-Optimal histogram.

## 4 Space-Bounded Histograms

In this section, we provide algorithms for computing space-bounded V-Optimal histograms, i.e., algorithms that attempt to minimize the error for a given number of buckets $B$. We propose three algorithms for the problem, all of which find provably optimal or close to optimal solutions: (1) a basic optimal algorithm based on dynamic programming, (2) an optimized and more sophisticated version of the basic optimal algorithm, and (3) an approximation algorithm with provable performance bounds that is significantly faster than the optimal algorithms.

Before describing the algorithms, we state two important technical lemmas. The proofs are by simple algebraic manipulation, and are omitted due to space constraints.

**Lemma 1** *For any vector $F$ of length $N$ and any $i, j$ with $1 \le i \le j \le N$, we have*

$$SSE([i,j]) = \sum_{i \le k \le j} F[k]^2 - (j - i + 1) \cdot AVG([i,j])^2.$$

Note that if we define arrays $P$ and $PP$ of length $n$ with $P[i] = \sum_{1 \le k \le i} F[k]$ and $PP[i] = \sum_{1 \le k \le i} F[k]^2$, then we have

$$\sum_{i \le k \le j} F[k]^2 = PP[j] - PP[i-1]$$

and

$$AVG[(i,j)] = \frac{P[j] - P[i-1]}{(j - i + 1)}.$$

This means that after spending $O(N)$ time and $O(N)$ space to compute the prefix sum arrays $P$ and $PP$, any $SSE([i,j])$ can be computed in constant time using the above lemma.

The next lemma is needed for the optimized version of the basic algorithm, and states a useful monotonicity property of the SSE metric.

**Lemma 2** *For any vector $F$ and any $i, j, k$ with $0 \le i \le k < j \le N$,*

$$SSE([i,j]) \ge SSE([i,k]) + SSE([k+1,j]).$$

### 4.1 Basic Optimal Algorithm

We now present an optimal algorithm for computing V-Optimal histograms based on dynamic programming. In our description, we focus on computing $SSE^*$, the SSE of the optimal histograms; the corresponding bucket boundaries can be obtained by maintaining an additional array that keeps track of the bucket boundaries of the partial solutions evaluated during the run of the algorithm. We point out that the algorithm is not restricted to the SSE error metric, but can be applied to a wide class of error metrics.

Define $SSE^*(i,k)$ to be the minimum SSE for the prefix vector $F[1,i]$ using at most $k$ buckets. The crucial observation underlying the algorithm is that

$$SSE^*(i,k) = \min_{1 \le j < i} \{SSE^*(j, k-1) + SSE([j+1, i])\}, \tag{1}$$

that is, the solution for $k$ buckets can be reduced to the case of $k-1$ buckets by considering all possible left boundaries of the rightmost ($k$th) bucket.

Thus, in order to calculate $SSE^* = SSE^*(N, B)$, we use dynamic programming and calculate $SSE^*(i,k)$ for all $1 \le i \le N$ and $1 \le k < B$, in increasing order of $k$, and for any fixed $k$, in increasing order of $i$. We store all computed values of the $SSE^*(i,k)$ in a table. Thus, when a new $SSE^*(i,k')$ is calculated using Equation (1), any $SSE^*(j,k)$ that may be needed can be retrieved by a table lookup.

There are a total of $O(N \cdot B)$ calculations of values $SSE^*(i,k)$, and each involves looping over $O(N)$ values of $j$ in Equation (1). For each $j$, we perform a table lookup for $SSE^*(j,k)$, and a call to find $SSE([j+1,i])$ that takes constant time by Lemma 1.

**Theorem 1** *The space-bounded $V$-Optimal histogram with $B$ buckets can be computed in $O(N^2B)$ time.*

### 4.2 Faster Implementation of the Optimal Algorithm

The algorithm described above is already quite efficient and can compute large histograms on thousands of elements and hundreds of buckets in a few minutes. We now present a technique that gives another significant reduction in the running time on most input data. Note that the faster algorithm still guarantees an optimum solution, and that on worst-case input data, the algorithm takes time $O(N^2B)$, as before. However, this case seems unlikely to arise in practice. The algorithm applies to a wide class of error metrics satisfying the monotonicity property of Lemma 2.

Consider the implementation of the basic optimal algorithm from the last subsection. Recall the computation of $SSE^*(i,k)$, and note that we have already computed and stored all entries $SSE^*(i',k-1)$ with $i' < i$. (These are the only entries we need to compute $SSE^*(i,k)$.) Suppose the algorithm now computes

$$SSE^*(i,k) = \min_{1 \le j < i}\{SSE^*(j,k-1) + SSE([j+1,i])\},$$

by iterating $j$ from $i-1$ down to 1. Note that as $j$ decreases, $SSE([j+1,i])$ monotonically increases due to Lemma 2. Thus, as soon as we arrive at a $j_0$ such that $SSE([j_0+1,i]) > S_0$, where $S_0$ is the minimum solution found thus far, we can stop the search, as all other values of $j$ will lead to even larger errors.

This termination condition for the inner loop already results in a performance improvement. However, we can take this process much further. Assume that $S_0$ is some initial ("seed") value that provides an upper bound for $SSE^*(i,k)$. Then we can use binary search to find $j_0$, the minimum $j$ such that $SSE([j_0+1,i]) > S_0$, and as before, we can conclude that the optimum solution is obtained by some $j > j_0$.

Now we observe that $SSE^*(j,k-1)$ monotonically increases as $j$ increases, also due to Lemma 2. Thus, $SSE^*(j_0,k-1)$ is a lower bound for any $SSE^*(j,k-1)$ with $j > j_0$. We can now define $S_1 = S_0 - SSE^*(j_0,k-1)$, and perform another binary search that finds $j_1$, the minimum $j$ such that $SSE([j+1,i]) > S_1$, and we can conclude that the minimum solution is obtained by some $j > j_1$. In general, we define

$$S_m = S_0 - SSE^*(j_{m-1}, k-1)$$

and repeat this process until $j_m = j_{m-1}$. We then use this $j_m$ as the lower limit for $j$ in the innermost loop, and compute the optimum solution.

A good initial value for $S_0$ can be obtained by running the innermost loop of the basic algorithm for about $N/B$ iterations. As stated before, the worst-case running time is still $O(N^2B)$, but we expect the algorithm to be significantly faster than the basic algorithm in most cases.

### 4.3 An Approximation Algorithm

Our third algorithm is a fast approximation scheme with provable performance bounds that leverages the dynamic programming schemes of the previous subsections. The overall idea is quite simple: We first partition the array $A$ into $\ell$ disjoint chunks, for some $\ell$, and then use the algorithm from the previous subsection to compute a histogram within each chunk.

A complication arises from the fact that we have to decide how to allocate buckets to the chunks, such that we use exactly $B$ buckets overall. We solve this problem by implementing an additional dynamic programming scheme over the number of buckets allocated to each chunk, which then repeatedly calls the dynamic programming algorithm inside each chunk with varying numbers of buckets. The details are non-trivial, and omitted for space constraints. The approximation guarantee and running time of the algorithm can be summarized as follows.

**Theorem 2** *For any integers $\ell$ and $B$, our approximation algorithm computes a histogram with $B+\ell$ buckets and total SSE at most $\delta$, where $\delta$ is the SSE of the optimal histogram on $B$ buckets. Moreover, if the $\ell$ chunks are chosen to be of equal width, then the algorithm runs in time $O(\frac{N^2B}{\ell})$.*

We implemented this algorithm based on the fast version of the optimal algorithm from the previous subsection, with an additional pruning technique applied to the top-level dynamic programming scheme. For the initial partitioning into chunks, we restrict ourselves to equal-sized chunks in this paper. As demonstrated in the next section, we obtain significant speedups over the optimal algorithms with only a slight decrease in the precision of the histogram.

## 5 Experimental Results

To assess the performance of various partitioning techniques, we conducted a series of experiments which are described in this section. We begin by specifying the data sets used. Due to space constraints we only give a sample of the most interesting results; more can be found in [JKS98, MPS98].

### 5.1 Experimental Testbed

We describe experiments using the following two real data sets, extracted from census statistics.

- D1: A density function on the third attribute of the SGI adult data set.[3] This data set has 732 unique values ($N = 732$).

- D2: The hourly *wages* of people from a census data set[4], with $N = 30200$.

In addition, in the comparison of the running times, we also generated data according to a randomly permuted Zipf distribution [Zip49]. The frequency vectors of the two real data sets are plotted in Figures 1 and 2. Observe that the first set is relatively smooth, whereas the second set has a large number of spikes. (The second set has in fact similar properties as a randomly permuted Zipf distribution.) As we show in our experiments, this difference has a significant effect on the relative performance of the different techniques.

We studied the accuracy, plotted as the *Mean Squared Error* $\frac{SSE}{N}$, and the running time, for the following partitioning techniques: (1) The basic optimal algorithm of Subsection 4.1 (NAIVE_DP), (2) the faster optimal algorithm of Subsection 4.2) (DP), and (3) the approximation algorithm of Subsection 4.3 (CHUNK). We compared these new algorithms with the following known techniques:

- *MHIST* - a greedy heuristic that repeatedly selects and splits the bucket with the highest SSE. This is the one-dimensional variant of the multi-dimensional MHIST algorithm proposed in [PI97].

- *MaxDiff* - a heuristic that places the bucket boundaries between those $B$ pairs of adjacent values that differ the most in their frequencies [PIHS96].

- *EquiDepth* - a heuristic that partitions the distribution such that the sum of the frequencies in each bucket is approximately equal [SC84].

- *EquiWidth* - a trivial heuristic that partitions the distribution into buckets of equal width [Koo80].

In the next two subsections we present the running times and accuracies of the various techniques.

### 5.2 Running Times

We first compare the running times of the three new algorithms based on dynamic programming (NAIVE_DP, DP, and CHUNK). For this purpose, we used a randomly permuted Zipf distribution with skew parameter $z = 0.85$ and varied the number of distinct values $N$. We set the space to 100 buckets and chose the number of partitions in CHUNK as 20; this means that the algorithm is guaranteed to do as least as good as the optimal algorithms with 80 buckets.

The results are shown in Figure 3. The limited range of input sizes presented already shows a very clear difference in performance between the three algorithms. In particular, the fastest algorithm (CHUNK) outperforms the slowest one (NAIVE_DP) by about two orders of magnitude. If we increase the input size to tens and hundreds of thousands, the running time of NAIVE_DP quickly rises to several hours, while CHUNK still runs in seconds or a few minutes. The running time for CHUNK can of course be further reduced by increasing the number of partitions beyond 20. In fact we found that increasing the number to 50 with 100 buckets results in little decrease in the accuracy.

Next, we compare the running times of the new and old techniques. For this purpose, we fixed the number of buckets $B$ at 100 and varied the number of unique values $N$ from 500 to 20000. The results are shown in Table 1. Note that the times for EquiWidth, EquiDepth, MHIST, and MaxDiff are negligibly small. CHUNK, on the other hand, while fast for small values of $N$, ultimately scales quadratically. As a result, the algorithm is significantly slower than the heuristics, but is still much faster than the two other dynamic programming algorithms.

| | Input Size ($N$) | | | |
|---|---|---|---|---|
| Algorithm | 1000 | 5000 | 10000 | 20000 |
| DP | 0.56 | 12.58 | 51.24 | 253.6 |
| CHUNK | 0.21 | 2.8 | 9.93 | 48.1 |
| MHIST | * | 0.02 | 0.05 | 0.06 |
| MaxDiff | * | * | 0.02 | 0.03 |
| EquiDepth | * | * | * | * |
| EquiWidth | * | * | * | * |

Table 1: Running Times in Seconds. An asterisk is used to denote times less than 0.01 seconds.

### 5.3 Accuracy

The Mean Squared Errors ($SSE/N$) of the different techniques, as a function of the number of buckets, are presented in Figures 4 and 5. In both cases, CHUNK performs basically as well as DP (which is optimal). The performance of other techniques varies significantly between the two data sets.

Not surprisingly, MHIST performs very well for smooth data (D1), where it essentially matches CHUNK and DP, but fails completely on spiked data (D2), where it is as bad as the trivial EquiDepth and EquiWidth heuristics. MaxDiff, on the other hand, essentially matches CHUNK and DP on the spiked data (D2), but performs even worse than EquiDepth and EquiWidth on the smooth data. We also observe that

[3]available at www.kdnuggets.com
[4]available at www.census.gov/DES/www/welcome.html

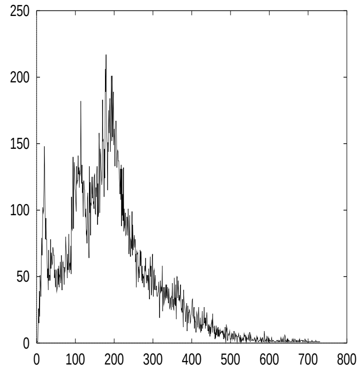


Figure 1: Data Set D1

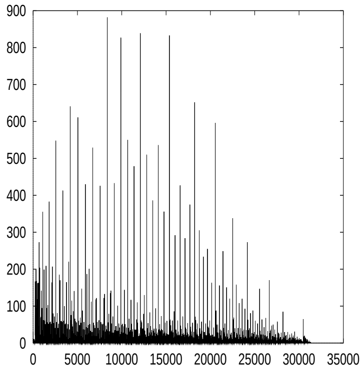


Figure 2: Data Set D2

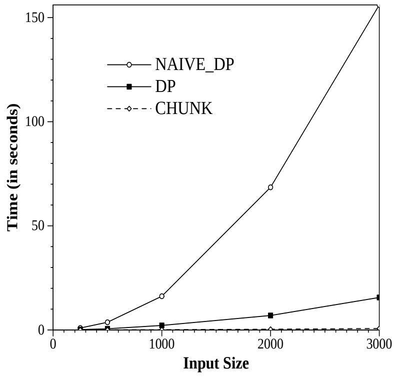



Figure 3: Running Times of Various Dynamic Programming-Based Algorithms

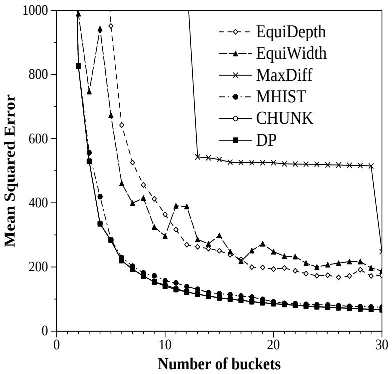



Figure 4: Effect of Bucket Space on Error (D1 data set)

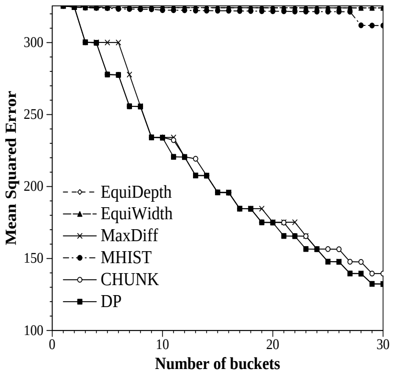



Figure 5: Effect of Bucket Space on Error (D2 data set)

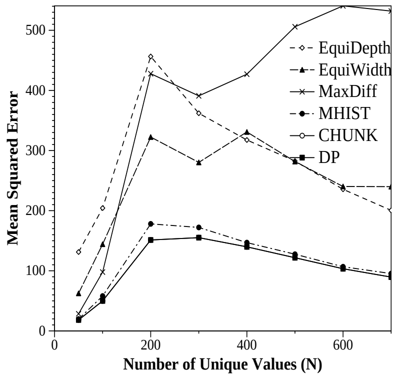



Figure 6: Effect of Data Size ($N$) on Error (D1 data set)

EquiDepth and EquiWidth benefit from more buckets for smooth data (though they are still very inaccurate), but fail to improve for spiked data. Thus, only DP and CHUNK achieve good accuracy on both smooth and spiked data.

Figure 6 presents the accuracy of the algorithms on data set D1, as a function of the number of values $N$, for a fixed number of 30 buckets. The relative performance of various techniques remains similar to the conclusions drawn above on D1. Interestingly, the mean squared errors first increase and then decrease as $N$ grows. This is due to the nature of D1: the data is more irregular in the beginning and then tapers off to a more uniform tail at the end.

## 6 Error Bounded Histogram

In this section, we describe our algorithms for computing error-bounded $V$-Optimal histograms, as defined in Section 3. We present the following results: (1) An algorithm for finding the optimal histogram that follows directly from the results in Section 4, (2) a new dynamic programming-based approximation algorithm with proven guarantees that is suitable when $\epsilon$ is small, and (3) an approximation algorithm with provable accuracy bounds that runs in essentially linear time, and that is the main technical result in this section. As before, our results hold for a wide class of error metrics, though we focus on the SSE metric.

**Definition 3** *We say an algorithm is an $(\alpha, \beta)$-approximation to the error-bounded $V$-Optimal histogram problem with error limit $\epsilon$ if it returns a partition with total $SSE$ at most $\alpha\epsilon$ using at most $\beta B^*$ buckets, where $B^*$ is the optimum solution.*

**The Primal Approach.** The algorithm is immediate given the results in Section 4. We run the algorithm for the space-bounded $V$-Optimal histogram problem, and terminate once we compute an $SSE^*(N, k)$ that is at most $\epsilon$. Thus,

**Theorem 3** *There exists an $O(N^2B^*)$ time algorithm to find a space-bounded $V$-Optimal histogram with error at most $\epsilon$, where $B^*$ is the optimum solution.*

**The Dual Approach.** For simplicity, assume that the error metric is integral. (This is not the case for

SSE, and we will also discuss the case of non-integral metrics.) Our solution is again based on dynamic programming. We focus only on computing the optimum number of buckets $B^*$; it is easy to extend this to computing the corresponding placement of the bucket boundaries. Define $B(\Delta, i)$ to be the optimum solution to the error-bounded $V$-Optimal histogram problem on the prefix $A[1, i]$ with error bound $\Delta$. We have,

$$B(\Delta, i) = \min_{1 \leq j < i} \{B(\Delta - SSE([j+1, i], j) + 1\}.$$

We need to calculate $B(\Delta, i)$ for each $1 \leq i \leq N$ and $1 \leq \Delta \leq \epsilon$, and each such term requires $O(N)$ time using the recursion above and employing dynamic programming. When the error metric is possibly non-integral (as with SSE), the range of values that $\Delta$ can take is large. Thus, we "discretize" the error in steps of $\kappa$ for some suitable choice of $\kappa$, and apply the dynamic programming above for integral error metrics with appropriate rounding to the next multiple of $\kappa$; the details are omitted. We can show:

**Theorem 4** *There exists a $(1+\frac{B^*\kappa}{\epsilon}, 1)$-approximation algorithm for the error-bounded $V$-Optimal histogram with parameter $\epsilon$ that runs in time $O(\frac{N^2\epsilon}{\kappa})$, where $B^*$ is the optimal number of buckets.*

This algorithm has interesting trade-offs for suitable choices of $\kappa$. For example, when $\kappa = \epsilon/\sqrt{B^*}$ for $\epsilon < 1$, this algorithm takes time $O(N^2\sqrt{B^*})$ and achieves an error of at most $O(\sqrt{B^*}\epsilon)$ with an optimum number $B^*$ of buckets. Thus, the algorithm is faster than the one in Theorem 3, but only gives an approximate solution.

**An Approximation Approach.** Our main technical result in this section is a fast approximation algorithm for the error-bounded $V$-Optimal histogram problem, with guaranteed accuracy. The algorithm is based on an interesting technical idea in that we use the solution to the similar problem of minimizing the maximum SSE in any bucket in order to get an approximate solution for the $V$-Optimal problem of minimizing the total SSE. This approach works because of two observations. Firstly, a solution that minimizes the maximum SSE in any bucket can be found more efficiently than a solution to the $V$-Optimal problem. Secondly, we can prove that the solution for this simpler problem can be used to get an approximate solution to our $V$-Optimal problem. We will first formalize these observations before presenting our algorithm.

**Lemma 3** *Given a partitioning with $B$ buckets and a total SSE of at most $\epsilon$, there exists a partitioning in which the maximum SSE in any bucket is at most $\epsilon'$ with $B + 2\frac{\epsilon}{\epsilon'}$ buckets.*

**Proof.** We provide an algorithm that converts a partition with SSE at most $\epsilon$ and $B$ buckets into one with maximum SSE at most $\epsilon'$. Any bucket with SSE at most $\epsilon'$ is left unchanged. Using the property of the SSE metric in Lemma 2, it follows that there exist at most $\frac{\epsilon}{\epsilon'}$ buckets with an SSE greater than $\epsilon'$. We split all such buckets in two steps as follows. In the first step, we traverse them from left to right and lay down dividers as soon as the SSE of the interval seen thus far exceeds $\epsilon'$. Clearly, this step introduces at most $\frac{\epsilon}{\epsilon'}$ additional buckets [5]. In the second step, we consider any bucket that has an SSE greater than $\epsilon'$, and partition it into two buckets, one containing only the rightmost element in the interval (the SSE of this interval is zero), and the other containing the rest. The second step introduces at most $\frac{\epsilon}{\epsilon'}$ additional buckets. ∎

**Lemma 4** *The error-bounded histogram problem with a maximum SSE in any bucket of at most $\epsilon$ can be solved in $O(\min\{B^* \log N, N\})$ time after $O(N)$ time preprocessing, where $B^*$ is the optimal solution.*

**Proof.** The details are in [MPS98]. ∎

We now describe our approximation algorithm, which, at the high level, is somewhat non-intuitive. It simply consists of determining the smallest $B$ (say $B'$) such that the optimum solution to the error-bounded histogram problem with a maximum SSE of at most $\frac{\epsilon}{B}$ in any bucket is at most $3B$. In order to do this, we repeatedly use the algorithm in Lemma 4, while performing a binary search for the value $B'$. Our algorithm returns the $B'$ thus found, and the corresponding partition, as the solution to the error-bounded $V$-Optimal histogram problem. The following can be shown:

**Theorem 5** *The algorithm above is an $(3,3)$-approximation algorithm for the error-bounded $V$-Optimal histogram problem that takes time $O(N + B^* \log B^* \log N)$, where $B^*$ is the optimum solution.*

**Proof.** The running time follows from Lemma 4 in a straightforward manner. We claim that the histogram computed by our algorithm is an $(3,3)$-approximation. Consider the optimal partitioning for the error-bounded $V$-Optimal histogram problem with SSE at most $\epsilon$. By Lemma 3 with $\epsilon' = \epsilon/B^*$, there exists a partition with SSE at most $\epsilon/B^*$ in any bucket using at most $3B^*$ buckets. It follows that the $B'$ our algorithm finds is at most $B^*$, and the solution returned has at most $3B^*$ buckets. Any such solution with a maximum SSE of at most $\epsilon/B'$ in each bucket has a total SSE of at most $3\epsilon$. This establishes the theorem. ∎

[5] The straightforward strategy would repeatedly place dividers just *before* the SSE of the interval seen thus far exceeds $\epsilon'$. It is easy to convince oneself that this strategy will only provide guarantees that are much worse than the one we prove here.

## 7 Quality Guarantees

In general, the accuracy of a selectivity estimate can vary widely from one query to the next, as the accuracy of the histogram may be different for different parts of the frequency distribution. While previous work seems to have largely ignored this problem, we believe that it is often highly desirable to have the histogram return some measure of the accuracy of the estimate. This would be particularly useful for applications that require a high degree of accuracy, e.g., approximate query processing.

For this purpose, we propose to augment histograms with additional information that gives guaranteed or statistical bounds ("quality guarantees") on the accuracy of a selectivity estimate. Of course, we could always return the worst-case error over all queries as our quality guarantee, but our goal is to return a good bound for each individual query. We present possible solutions for equality and range selection queries, and give experimental results that show the improvements they achieve over the worst-case guarantees.

### 7.1 Quality Guarantees for Equality and Range Queries

We define the *quality* of a selectivity estimate as an upper bound on its absolute error. Let $R$ be a relation and let $f_a[k]$ be the frequency of value $k$ in attribute $a$ of $R$. Let the buckets in the histogram be $b_1, b_2, \ldots$, in increasing order of the attribute values contained in them.

**Equality Selections:** The result size of an equality predicate $a == k$ is approximated by the average frequency $f_i$ of the bucket $b_i$ containing $k$. Then the absolute error of this estimate is $e_k = \mid f_a[k] - f_i \mid$. Let $E_i$ be the maximum error in bucket $b_i$, i.e., $E_i = MAX\{e_k \mid k \in b_i\}$. A natural choice is to return $E_i$ as the quality guarantee for an estimate of an equality query. Note that this requires storing an additional value ($E_i$) with each bucket.

**Range Selections:** For simplicity, we consider one-sided range predicates of the form $a \leq k$, though the scheme can be easily generalized. Let $k$ fall into bucket $b_i$, and let $m, M$ be the smallest and largest attribute values in $b_i$. Then the estimate of the result size of the query $a \leq k$ is given by

$$\sum_{j=1}^{i-1} S_j + (k - m + 1) \cdot f_i,$$

where $S_j = |b_j| \cdot f_j$ is the sum of the frequencies in bucket $b_j$. We derive an upper bound on the error by observing that no error at all is incurred for the first $i-1$ buckets, since they are completely within the range and hence accurately captured by the $S_j$. An upper bound for bucket $b_i$ can be computed by using the equality selection bound $E_i$ for each value in $b_i$, giving a quality guarantee of $MIN(k-m+1, M-k+1) \cdot E_i$.

Note that this does not require any additional storage beyond the $E_i$ already used to bound the equality selection errors. However, we could decide to also maintain in each bucket the average absolute error. This average error could be returned as another quality measure, or it could be used in conjunction with the $E_i$ values to derive even tighter bounds on the (maximum, not average) estimation error of range queries, by using Markov's Inequality.

In general, by maintaining appropriate types of statistics in each bucket, we could derive improved estimates as well as quality guarantees for various types of queries. Of course, such extra statistics increase the space used by the histogram, but this may sometimes be a worthwhile expense. The best selection of statistics depends on the particular space and accuracy requirements of the application. If space is very tight, one might just provide flags that distinguish "good" from "bad" buckets.

### 7.2 Experimental Results

We briefly discuss the results of our experiments on the quality guarantees for equality and range selection queries derived above. We compare these bounds with the actual errors in estimating the result sizes of equality and range selections, and with the naive worst-case bound given by using the maximum error over all buckets.

Figures 7 and 8 compare the three bounds as functions of the bucket counts, averaged over all possible predicates of the form $X == a$ and $X \leq a$, with $1 \leq a \leq N$. The input is a randomly permuted Zipf distribution with $z = 0.8$, and the buckets are formed using a V-Optimal histogram, We observe that the naive worst-case bound for the error (the upper curve) is significantly higher than the average actual error (the lower curve), and that the bounds derived above lead to error bounds (the middle curve) that are significantly closer to the actual error.

## 8 Other Applications

### 8.1 Work Load Information

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 SSE metric has done so far).

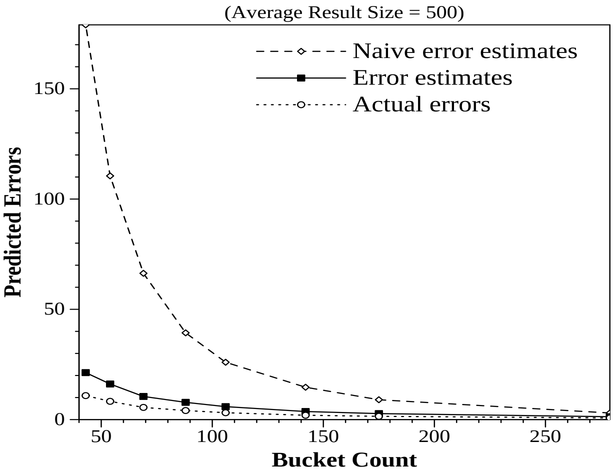



Figure 7: Quality Guarantees: Equality Predicates

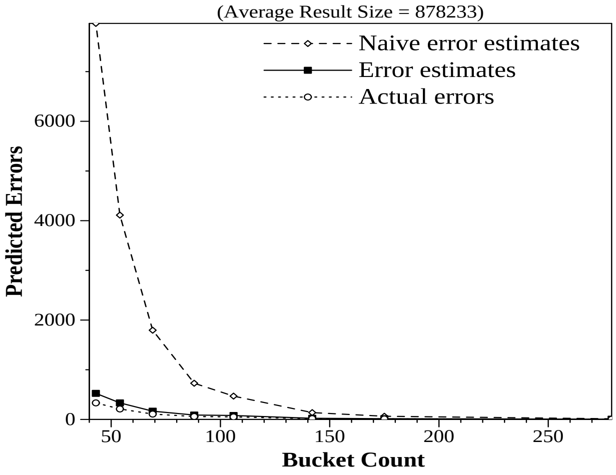



Figure 8: Quality Guarantees: Range Predicates

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 point, consider a synthetic data set drawn from a (truncated) Gaussian distribution and shown in Figure 9(a) along with a histogram with 30 buckets, created optimally using the SSE metric (Figure 9(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 9(c). Observe the difference in the two histograms.

Optimal histograms 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 $SSE(i, j)$ (or any other metric) function values are calculated. The rest of the algorithm remains unchanged (details are available elsewhere [JKS98]). None of the other techniques accommodate this and, in fact, no one has addressed this issue in the existing literature. It is not surprising that no single heuristic can possibly generate histograms even remotely similar to *both* Figures 9(b) and (c), for example.

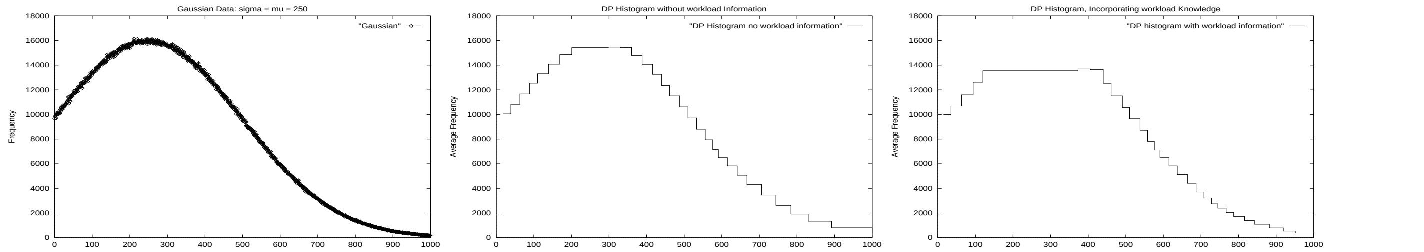
In fact, we know that many data distributions in real-life are extremely skewed. We also know that access patterns to these data are extremely skewed. Therefore, it is important to take this skew into account when choosing histogram boundaries.

### 8.2 Range Queries

The metrics discussed thus far have focused on the error in reconstruction of a single point. 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 attribute values with counts much higher and much lower than the average for the bucket, may not introduce much error since the pair of attribute values together will be included in most range queries – an error is induced only when one of the attribute values is included in some range and the neighboring cell is excluded from that range. 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 error metric. For each $SSE(i, j)$ (or any other metric) 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 DP algorithm can run normally. The results can be stunningly different than for the SSE metric. For example, consider a distribution of an age attribute. 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




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

Figure 9: Incorporating Workload Knowledge

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. Following [PIHS96] an error metric can be constructed which minimizes the variance of the "area" inside buckets. Details are available elsewhere [JKS98].

**Experimental Results.** We compared the estimation accuracy of DP, EquiWidth, EquiDepth, and MaxDiff.

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.

Figures 10(a) and 10(b) present the trends in accuracy of the three algorithms for data set D1 as the number of buckets increases (Figure 10(a)) and, as the number of cells increases (Figure 10(b)).

### 8.3 Join Queries

Histograms are also useful for estimating the result size of join queries. It has been argued in [IP95] that the error in the estimate of the query result size is minimized when the variance of the counts of frequencies in each bucket is minimized. This turns out to be the same as our SSE metric. We next present sample experimental results investigating the accuracy of join result size estimation of various types of histograms. More discussion and experimental results are available elsewhere [JKS98].

**Experimental Results.** The algorithms compared are DP, EquiWidth, EquiDepth and End-Biased (EBV). For all the graphs in this subsection dealing with accuracy of estimation, the standard deviation of the prediction error is used as previously suggested [IP95].

Figures 11(a) and 11(b) present the accuracy of the algorithms for D2 with increasing number of buckets (Figure 11(a)) and, with increasing number of cells (Figure 11(b)). For this data set and for a small number of frequencies, DP is able to predict the join result size almost exactly.

## 9 Concluding Remarks

In this paper, we have studied the problem of computing optimal histograms, which minimize the error for a given amount of space. We have provided several algorithms based on dynamic programming that are the first to efficiently and precisely compute optimal histograms under a large class of error metrics, including the well-known V-Optimal histograms, and have shown that the performance of our algorithms can be improved by several orders of magnitude through the use of several nontrivial optimizations. Our experiments show that the algorithms obtain significantly better accuracy than the known heuristics.

We have also studied the dual problem of minimizing the space required to meet a given error bound, and have proposed a way of augmenting histograms to return *quality guarantees* for selectivity estimation queries. Finally, we have extended our techniques to incorporate knowledge of the query work load and to identify optimal histograms for range and join queries.

## Acknowledgements

Thanks to Stefan Berchtold, Christina Christara, Carl de Boor, Yannis Ioannidis, Flip Korn, Andrew Odlyzko, Nikolaos Sidiropoulos and Sridhar Ramaswamy for valuable discussions and comments on the writeups. We also thank the reviewers for their comments, and Betty Salzberg for overseeing the preparation of the final paper.

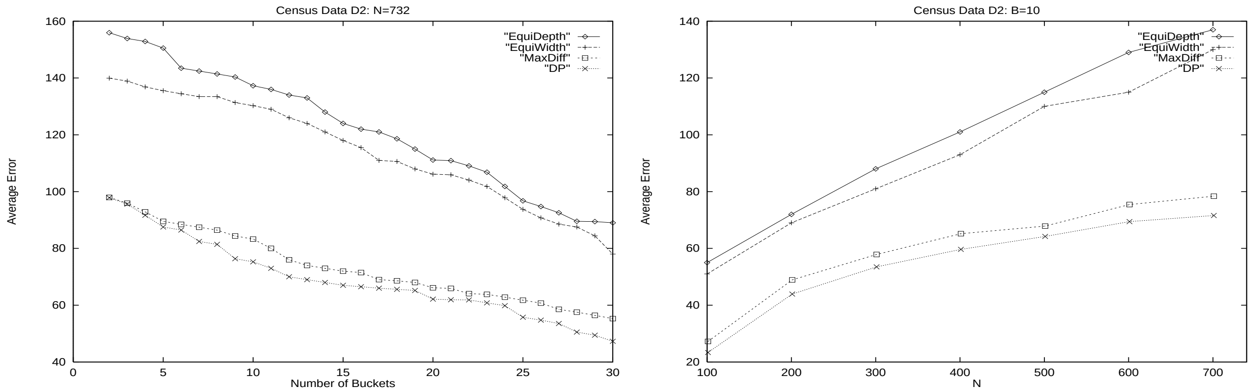
## References

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

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

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

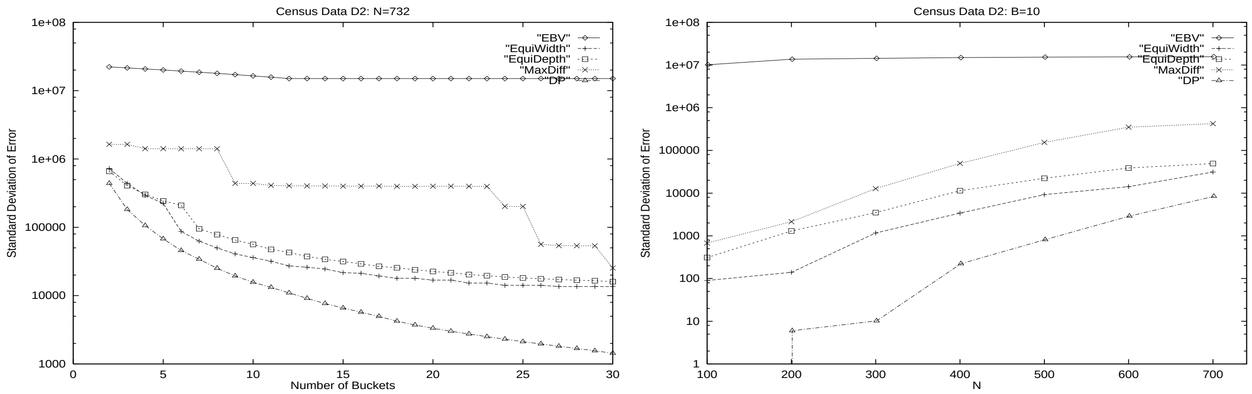
[HS92] P. Haas and A. Swami. Sequential Sampling Procedures for Query Size Estimation. *Proceedings of ACM SIGMOD, San Diego, CA*, pages 341–350, June 1992.




(a) D1: Increasing B

(b) D1: Increasing N

Figure 10: DP, EquiWidth, EquiDepth, MaxDiff for D1




(a) D1: Increasing B

(b) D1: Increasing N

Figure 11: DP, EquiWidth, EquiDepth, EBV, MaxDiff for D1


[Ioa93] Y. Ioannidis. Universality of serial histograms. *Proc. of the 19th Int. Conf. on Very Large Databases*, pages 256–267, December 1993.

[IP95] Y. Ioannidis and V. Poosala. Balancing histogram optimality and practicality for query result size estimation. *Proc. of ACM SIGMOD*, pages 233–244, May 1995.

[JKS98] H. V. Jagadish, Nick Koudas, and K. C. Sevcik. Choosing Bucket Boundaries for a Histogram. *University of Toronto Technical Report, TR-375*, May 1998.

[Koo80] R. P. Kooi. *The optimization of queries in relational databases.* PhD thesis, Case Western Reserver University, Sept 1980.

[LNS90] R. J. Lipton, J. F. Naughton, and D. A. Schneider. Practical selectivity estimation through adaptive sampling. *Proc. of ACM SIGMOD*, pages 1–11, May 1990.

[MCS88] M. V. Mannino, P. Chu, and T. Sager. Statistical profile estimation in database systems. *ACM Computing Surveys*, 20(3):192–221, Sept 1988.

[MPS98] S. Muthukrishnan, V. Poosala, and T. Suel. Optimal Histograms with Quality Guarantees. *Bell Labs Technical Report*, May 1998.

[OR86] F. Olken and D. Rotem. Simple random sampling from relational databases. *Proc. of the 12th Int. Conf. on Very Large Databases*, August 1986.

[PI97] V. Poosala and Y. Ioannidis. Selectivity estimation without the attribute value independence assumption. *Proc. of the 23rd Int. Conf. on Very Large Databases*, August 1997.

[PIHS96] V. Poosala, Y. Ioannidis, P. Haas, and E. Shekita. Improved histograms for selectivity estimation of range predicates. *Proc. of ACM SIGMOD*, pages 294–305, June 1996.

[SC84] G. P. Shapiro and C. Connell. Accurate estimation of the number of tuples satisfying a condition. *Proc. of ACM SIGMOD*, pages 256–276, 1984.

[Zip49] G. K. Zipf. *Human behaviour and the principle of least effort.* Addison-Wesley, Reading, MA, 1949.