Global Optimization of Histograms

H.V. Jagadish[1], Hui Jin[2], Beng Chin Ooi[2], Kian-Lee Tan[2]

[1]Department of Electrical Engineering and Computer Science
University of Michigan
jag@eecs.umich.edu

[2]Department of Computer Science
National University of Singapore
{*jinhui, ooibc, tankl*}*@comp.nus.edu.sg*

ABSTRACT

Histograms are frequently used to represent the distribution of data values in an attribute of a relation. Most previous work has focused on identifying the optimal histogram (given a limited number of buckets) for a single attribute *independent of other attributes/histograms*. In this paper, we propose the idea of *global optimization of histograms*, i.e., single-attribute histograms for a set of attributes are optimized collectively so as to minimize the overall error in using the histograms. The idea is to allocate more buckets to histograms whose attributes are more frequently used and/or distributions are highly skewed. While the accuracy of some histograms is penalized (being assigned fewer buckets), we expect the global error to be low compared to the traditional method (of allocating equal number of buckets to each histogram).

We propose two algorithms to determine the histograms to construct for a collection of attributes. The first is based on dynamic programming, and the second is a greedy algorithm. We compare the overall error of these algorithms against the traditional method. Extensive experiments are conducted and the results confirm the benefits of global optimal histograms in reducing the overall error. The extent of improvement depends on the data and query distributions, ranging from no benefit when there is no significant differences in the data distributions to over a factor of 100 reduction in error in some cases we tried.

The time to compute global optimal histogram using dynamic programming is much longer than the time to compute optimal histograms separately for each attribute, and the difference widens at a faster rate as the number of histograms increases. With the greedy algorithm, the time penalty is small, but the error reduction is somewhat less as well. We propose a third algorithm, called greedy algorithm with remedy, that has running time similar to the greedy algorithm, but produces results close to global optimum. In fact, in every experiment that we tried, this algorithm found the exact global optimum.

ACM SIGMOD 2001 May 21-24, 2001, Santa Barbara, California USA

1. INTRODUCTION

Most DBMSs maintain a variety of statistics of the contents of the database relations in order to estimate various quantities, such as selectivities within cost-based query optimizers [19, 13]. It has been established that the validity of the optimizers' decisions may be critically affected by the quality of these statistics [7, 10]. This is becoming particularly evident in the context of increasingly complex queries, e.g. data analysis queries.

Furthermore, there has been an attempt to produce quick approximate answers to complex decision support queries. One popular way to accomplish this is by means of maintaining sufficient data statistics in the form of "synopsis" data structures [3, 8, 4]. The availability and quality of summarized information becomes all the more important in such scenarios.

Histograms are widely used in DBMSs as a means of summarizing information [13]. A histogram approximates the distribution of data values in attributes of the database relations by grouping the data values into a group of buckets. This grouping into buckets loses information. It is important to choose bucket placement wisely, to minimize this loss of information for any chosen level of data summarization. Poor quality histograms might lead the optimizers to choose suboptimal query execution plans that may degrade the system performance dramatically. They can also produce results, in an approximate query answering system, that are quite far from being correct.

Much work has been reported in the literature on the design of histograms. [16] showed that equi-width histograms have a much higher worse-case and average error for a variety of selection queries than equi-depth histograms. V-Optimal histograms were introduced and shown to minimize the average error for equality selectivity estimation problems [9, 11]. The experimental results in [18] show that V-Optimal(V,A) and V-Optimal(V, F) histograms can generate estimations for range queries with low error, much lower than equi-depth histograms. For multi-attribute selection queries, [15] extended the work of [16] to multi-dimensional histograms. In [17], four techniques were proposed, namely, the singular value decomposition (SVD), Hilbert numbering, PHASED and MHIST, and were experimentally evaluated against multi-dimensional equi-depth histograms.

In real world database management systems, the space allotted to store the histograms is limited. Each histogram is assigned the space of the order of 200 bytes in a catalog [18]. All histograms are given the same amount of storage regardless of the data distributions. However, it takes different numbers of histogram buckets for different data distribution to provide any error guarantees [12]. For example, a uniformly distributed data set needs very few buckets to reach satisfactory estimation, while a very skewed data set might need many more buckets to get the same satisfactory approximation. In other words, given a certain error bound, we require different amount of storage space to construct histograms to achieve this bound for different data distributions.

In this paper, we address the problem of *globally optimizing a set of histograms*: given a certain amount of space to construct a set of histograms, determine the histograms to construct such that the global error is minimized. Such an approach allows us to allocate more buckets to histograms whose attributes are more frequently used and distributions are highly skewed. We present four algorithms to compute this global optimum. The first is an expensive exhaustive enumeration approach; the second, a dynamic programming approach that obtains the global optimum at reasonable cost; the third is a greedy heuristic that is very fast, and comes close to the optimum; the fourth is a modification to the greedy heuristic that is almost as fast as the pure greedy approach, but comes much closer to the global optimum. We conducted extensive experiments that compare the proposed algorithms to the conventional one [12] in terms of accuracy and running time. Our results show that the proposed algorithms can produce better selectivity estimates of range queries compared to traditional method. The errors of the methods are generally much lower than existing techniques. In terms of efficiency, the time to compute global optimal histogram using dynamic programming is much longer than the time to compute optimal histogram separately for each attribute. Moreover, the difference between the two techniques widens at a faster rate as the number of histograms increases. With the greedy algorithm, the time penalty is small, but the error reduction is somewhat less as well. We also show that the greedy algorithm with remedy has running time similar to the greedy algorithm, but produces results close to global optimum. In fact, in every experiment that we tried, this algorithm found the exact global optimum. To our knowledge, till now, there is no reported work that takes the total space limitation for histogram storage into consideration while leveraging the difference of data distributions to achieve global optimization of the histograms.

To free the database designer and administrator from the task of constructing and maintaining every histogram that is related to a query, research has been conducted towards automatically identifying the right set of histograms for a database recently. To identify the critical histograms, [6] proposed techniques to determine the minimal set of histograms for a database based on the workload characteristics. Our proposal is an alternative solution. What is more, our proposal not only takes the workload characteristics into consideration, but also the distribution of the attributes, to identify the critical histograms. Our proposal is much easier to be implemented than the techniques proposed in [6]. As described in the following, when an attribute is assigned only one bucket for its histogram, this histogram of the attribute is not in the minimal set. And its space freed can be assigned to other critical histograms.

The rest of this paper is organized as follows. We define histograms, error metrics and formulate the problems we study in Section 2. In particular, we describe the idea of a *Global Optimal Histogram*. Then, in Section 3, we present the four proposed algorithms to compute the global optimum. In Section 4, we present the results of our experiments, and discuss the effectiveness of the Global Optimal Histogram and the efficiency of the techniques, and finally, we conclude in Section 5.

2. DEFINITIONS AND PROBLEM FORMULATION

Consider a relation R containing an attribute X. 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 $r \in R$ with $r.X = v$. We assume that the elements of V have been sorted according to some sort parameter(following [18]), commonly according to the values of V, e.g. $V = \{v_i | 1 \leq i \leq N\}$ where $i < j$ iff $v_i < v_j$ and N is the number of distinct values. We denote $f(v_i)$ as f_i, and define the data distribution of X as the set of pairs $T = \{(v_1, f_1), (v_2, f_2), \ldots, (v_N, f_N)\}$.

A histogram of an attribute X is constructed by partitioning the data distribution T into $b(\geq 1)$ buckets, and approximating the frequencies and values in each bucket. The accuracy of any operation performed using a histogram depends on the accuracy of the approximation.

Let M denote the number of histograms in the system catalog, and let the set of histograms be $H = \{h_i | 1 \leq i \leq M\}$. As noted in the last section, for real world DBMS, the space in a system catalog for storing the histograms is limited. Let B be the total number of buckets used to represent all the histograms. Let us denote by the pair (h_i, b_i) the allocation of b_i buckets to histogram h_i, $(1 \leq i \leq M)$.

Definition 1 A set of histograms $H = \{h_i | 1 \leq i \leq M\}$ is called the set of **Global Optimal Histograms (GOH)**, $\{(h_i, b_i) | 1 \leq i \leq M\}$, if $\sum_{i=1}^{M} b_i \leq B$ and $min(\sum_{i=1}^{M} p_i \cdot e_i)$, where e_i is the error of histogram h_i and p_i is the probability of attribute X_i being queried.

2.1 Error Metrics

From an application perspective, the error metric we most care about are *absolute error* and *relative error*[14] in the estimate returned by a histogram. Let S_i be the actual size of a query q_i and let S_i' be the estimated size of the query by a histogram. The *absolute error* of the query is defined as:

$$e_i^{abs} = |S_i - S_i'|$$

And the *relative error* of the query is defined as:

$$e_i^{rel} = \frac{e_i^{abs}}{S_i} = \frac{|S_i - S_i'|}{S_i}$$

Absolute error and relative error are calculated based on a specific query or a set of queries. As such, it is very hard to apply absolute error or relative error directly to assess histogram quality universally.

We address this difficulty by introducing an error metric **Average Standard Deviation**, which can be calculated directly from a histogram and its corresponding data distribution. Our proposal is an enhancement of intrinsic histogram quality metrics proposed by a long line of researchers. (See next paragraphs). Our extensive experiments show that minimization of this error metric leads to reduction in both absolute error and relative error over a wide range of queries.

To evaluate the accuracy of a histogram, [12] proposed an error metric, *Sum Squared Error*, which is defined as follows:

for any interval $[i, j], i \leq j$,

$$SSE(i,j) = \sum_{k=i}^{j} (f_k - avg(i,j))^2$$

where

$$avg(i,j) = \frac{\sum_{k=i}^{j} f_k}{j - i + 1}$$

A histogram constructed with the minimal SSE under a limited number of buckets is known as a V-Optimal histogram [11]. V-optimal(V,F), V-optimal(V,A) histograms were reported [18] among the most accurate histograms for equality and range selection. If storage has already been allocated at the level of individual histograms, then the best we can do is to construct a V-Optimal(V,F) histogram for each.

We note that $\sqrt{SSE}$ is actually the standard deviation, σ, of a data distribution. Clearly, minimizing SSE minimizes σ. Different attributes may have different number of values, distinct values and their corresponding histograms may have different number of buckets. To make the error metrics of different histograms comparable, they have to be scaled. This scaling is much easier done for standard deviation than for SSE, since the former is measured in the same units as the attribute in question, whereas SSE is measured in square units. As such, we prefer stating the V-Optimal problem in terms of minimizing standard deviation rather than of minimizing SSE.

Because the estimation error for a range query is caused by the bucket in which a query 'straddles', if the average of standard deviation of each bucket is low, we argue and our comprehensive experiments show, the average error of the estimation would also be low. In other words, we should use the standard deviation normalized by the number of buckets as the error metric. We denote it as *average standard deviation*:

$$\sigma^{avg} = \frac{\sqrt{SSE}}{number\ of\ buckets}$$

We customize the definition of GOH as

> **Definition** 2 A set of histograms $H = \{h_i | 1 \leq i \leq M\}$ is called the set of **Global Optimal Histograms (GOH)**, $\{(h_i, b_i) | 1 \leq i \leq M\}$, if $\sum_{i=1}^{M} b_i \leq B$ and $min(\sum_{i=1}^{M} p_i \cdot \sigma_i^{avg})$, where σ_i^{avg} is the standard deviation of histogram h_i normalized by b_i, and p_i is the probability of attribute X_i being queried.

p_i can be obtained from workloads by monitoring and logging queries on the database systems. For simplicity, we will set it as 1 in this paper as this does not affect our results.

We note here that GOH proposed in this paper can be applied to other kinds of single attribute histograms, such as Compressed histograms, Maxdiff histograms, and multi-dimensional histograms. [17] reported that the techniques proposed require less space for low and high dependency multi-dimensional histograms than that for intermediate dependency histograms given a certain error criterion. If intermediate dependency histograms are assigned more space than low and high dependency multi-dimensional histograms, we can expect improvement in the overall quality of a group of multi-dimensional histograms.

3. GLOBAL OPTIMAL HISTOGRAMS

While we consider different data distributions of attributes, we observe that evenly distributed data needs fewer buckets to get a satisfactory approximation while very skewed or irregularly distributed data needs a lot more buckets to reach certain error criteria. In fact, for uniformly distributed data, 5 buckets or 50 buckets do not make much difference in selectivity estimation. But, for skewed data, even if we increase the number of buckets only by 1 to approximate the data distribution, the quality of the histogram might increase significantly. These are illustrated in Example 3.1. We denote $\sigma_i^{avg}(j, N_i)$ as the average standard deviation of histogram h_i with the number of buckets j, and its data distribution T_i has N_i distinct values.

Example 3.1 Consider two data distributions: $T_1 = \{(1,50), (2,50), (3,50), (4,51)\}$ and $T_2 = \{(10,70), (11,2), (12,80), (13,3)\}$. For T_1, we have $\sigma_1^{avg}(1,4) = 0.87$ and $\sigma_1^{avg}(2,4) = 0$. For T_2, we have $\sigma_2^{avg}(1,4) = 72.85$, $\sigma_2^{avg}(2,4) = 30.01$, $\sigma_2^{avg}(3,4) = 16.03$ and $\sigma_2^{avg}(4,4) = 0$. While 2 buckets and 4 buckets make no difference to T_1, 2 buckets and 4 buckets make much difference to T_2 in error reduction. ♣

To figure out where our buckets may be best spent, we introduce the notion of *marginal gain*. By assigning more buckets to histograms with greatest marginal gain, we hope to increase the total quality of the histograms in a catalog.

Definition 3 Marginal Gain (m) The marginal gain of a data distribution is the improvement in the quality or accuracy (i.e., reduction in errors) of its representative histogram as a result of additional buckets being allocated.

$$m(i,j) = \sigma^{avg}(i,N) - \sigma^{avg}(j,N)$$

where i and j are the number of buckets assigned and $1 \leq i \leq j$, N is the number of distinct values of the data distribution.

Example 3.2 Continuing example 3.1, we find that $m_1(1,2) = 0.87$ and $m_1(2,3) = 0$. Similarly, we get $m_2(1,2) = 42.84$, $m_2(2,3) = 13.98$, $m_2(3,4) = 16.03$ and $m_2(4,5) = 0$. In other words, we should not even devote a second bucket to the first distribution until we have been able to devote four buckets to the second distribution. ♣

Using the above intuitions, in this section, we first present an exhaustive algorithm to find the set of *Global Optimal Histograms (GOH)* based on the average standard deviation criteria and then provide a dynamic programming algorithm to build *GOH*. After that, we propose a greedy algorithm to construct approximate *GOH* which is significantly faster than the exhaustive or dynamic programming algorithms. Because the greedy algorithm misses optimal solutions in some situations, we present a remedy for the algorithm, that, while not provably optimal, comes very close in practice.

An important contribution of [12] is that it provided a dynamic programming algorithm *(DP)* that constructs a V-Optimal histogram in $O(N^2b)$ time. In constructing *GOH*, we apply *DP* to construct each individual V-Optimal histogram. We do so, irrespective of which of these four algorithms we use to build the *GOH*.

3.1 Exhaustive Algorithm

In this section, we provide an exhaustive algorithm that achieves the *GOH*. The exhaustive algorithm essentially tries all possible allocations to obtain the optimal solution.

For every histogram that we want to construct, we apply the *DP*[12] to calculate

$$\sigma_i^{avg}(b, N_i)$$

for all $1 \leq b \leq B - M + 1, i \in [1, M]$, where N_i is the number of distinct values of the ith attribute. We then store the computed values in an array. Next, all the different combinations of different number of buckets assigned to each histogram are enumerated and for each combination, we compute $\sum_{i=1}^{M} \sigma_i^{avg}(b_i, N_i)$, where $\sum_{i=1}^{M} b_i = B$. The set $\{(h_i, b_i) | 1 \leq i \leq M\}$ that has the $min(\sum_{i=1}^{M} \sigma_i^{avg}(b_i, N_i))$ is the *GOH*.

To calculate every $\sigma_i^{avg}(b, N_i)$ requires $O(\sum N_i^2 B)$. And to enumerate every combination to calculate $\sum \sigma_i^{avg}(b_i, N_i)$, the time complexity is $O(C_{B-1}^{M-1})$, which is a large exponential in the number of histograms. While this algorithm generates the optimal solution, it is clear that it is not practical for large number of buckets and histograms. We have included it merely for the purpose of comparison to determine the quality of the proposed algorithms.

3.2 Optimal Algorithm based on Dynamic Programming

We observe that

$$min(\textstyle\sum_{i=1}^{k} \sigma_i^{avg}(b_i, N_i)) = min(min(\sum_{i=1}^{k-1} \sigma_i^{avg}(b_i, N_i)) + \sigma_k^{avg}(b_k, N_k))$$

where b_k ranges from 1 to $B - M + 1$, $\sum_{i=1}^{k-1} b_i$ is from $(B - M + k - 1)$ to $(k - 1)$ correspondingly. The solution for $min(\sum_{i=1}^{k} \sigma_i^{avg}(b_i, N_i))$ can then be reduced to

$$min(\sum_{i=1}^{k-1} \sigma_i^{avg}(b_i, N_i))$$

Thus, we can apply dynamic programming to calculate

$$min(\sum_{i=1}^{M} \sigma_i^{avg}(b_i, N_i))$$

for all $1 \leq i \leq M$ and $k \leq \sum_{i=1}^{k} b_i \leq (B - M + k)$.

Dynamic Programming Algorithm (GOHDP)

1. for $i = 1$ to M
   for $b = 1$ to $(B - M + 1)$
   Calculate $\sigma[i][b]$;
2. for $i = 1$ to $(B - M + 1)$
   $min\_sum\_error[1][i] = \sigma[1][i]$;
3. for $j = 2$ to M
   for $k = 1$ to $(B - M + 1)\{$
   $min\_sum\_error[j][k] =$
   $min(min\_sum\_error[j-1][p] + \sigma[j][q])$
   where $p, q \geq 1$ and $p + q = B - M$;
   record the p and q that
   achieve min_sum_error;
   $\}$
4. $min\_sum\_error[M][B-M+1]$ is the result: minimal sum of average standard deviation;

When we get the $min\_sum\_error[M][B - M + 1]$, we can trace back the array to retrieve b_i assigned to each h_i.

This algorithm also achieves perfect *GOH* but it is much more time efficient than the exhaustive algorithm. To calculate all the $\sigma_i^{avg}(b, N_i)$, the time complexity is $O(\sum N_i^2 B)$. And to execute the dynamic programming part, the time complexity is $O(BM)$. So, the time complexity of this dynamic programming algorithm is polynomial. Assume every N_i being equal, $N_i = N$, the time complexity of this optimal algorithm is of $O(N^2 BM)$.

3.3 Greedy Algorithm

Even though the dynamic programming algorithm is considerably more efficient than the exhaustive algorithm, it is still computationally expensive. In this section, we propose a greedy algorithm to get an approximate *GOH* quickly. Before describing the algorithm, we present a lemma and a theorem.

According to the definition of the average standard deviation and V-Optimal(V,F) histogram, we have the following two straightforward results:

Lemma 1, *For any data distribution with the number of distinct values N, and any i, j with* $1 \leq i < j$, *we have*

$$\sigma^{avg}(i, N) \geq \sigma^{avg}(j, N)$$

Based on the definition of marginal gain and lemma 1, we have the following theorem.

Theorem 1, *For* $1 \leq i \leq j$,

$$m(i, j) \geq 0; \quad When\ i = j,\ m(i, j) = 0.$$

As $\sum \sigma_k^{avg}(1, N_k)$ is a constant for a group of attributes, and

$$\sigma_k^{avg}(1, N_k) = \sigma_k^{avg}(j, N_k) + m_k(1, j), \quad where\ 1 \leq j$$

given a total number of buckets, the larger the value of $\sum m_k(1, j)$, the smaller $\sum \sigma_k^{avg}(j, N_k)$ would be. For DP, to construct a V-Optimal(V,F) histogram of an attribute, SSE(i-1, N) must be calculated before SSE(i, N) can be obtained. This is computationally expensive and storage inefficient. However, from theorem 1, we know that $m(i, i+1) \geq 0$ for all $i \geq 1$. The proposed greedy algorithm exploits this result by assigning every histogram one bucket initially, and repeatedly allocates an additional bucket to the the histogram with the highest marginal gain (until all buckets are allocated).

Greedy Algorithm (GOHGA)

1. For i = 1 to M
   Compute $\sigma_i^{avg}(1, N_i)$, $\sigma_i^{avg}(2, N_i)$ and $m_i(1, 2) = \sigma_i^{avg}(1, N_i) - \sigma_i^{avg}(2, N_i)$.
2. For i = 1 to M, find the maximum m_i. Assume this is the kth histogram. Increase this histogram's number of bucket by 1, $b_k = b_k + 1$. Calculate this data distribution's new bucket marginal gain $m_k(b_k, b_k + 1) = \sigma_k^{avg}(b_k, N_k) - \sigma_k^{avg}(b_k + 1, N_k)$
3. Goto step 2, until the total number of buckets is B.

The greedy algorithm is very efficient and manages to approximate *GOH* very well in most cases as our extensive experiments show.

The time complexity of this greedy algorithm is $O(\sum(N_i^2 b_i))$. If we assume all $N_i = N$, the time complexity is $O(N^2 B)$. So, it is much more efficient than the optimal algorithm based on dynamic programming.

3.4 Greedy Algorithm with Remedy

Upon examination of the cases where the greedy algorithm fails to achieve the optimum, we notice that in all these cases there is an "inversion" in marginal gain. That is, $m(i, i+1)$ might be less than $m(i+1, i+2)$, for some value of i for some distribution.

To assign each bucket, the greedy algorithm only considers the marginal gain of one additional bucket granted to each data distribution and selects the one with the largest gain. This process would be optimal if marginal gains decrease monotonously with more buckets – that is, if the "law of diminishing returns" is followed exactly. However, in practice, it appears that marginal gain has only a generally decreasing trend as number of buckets is increased, with occasional exceptions. This might lead the greedy algorithm to reach a sub-optimal result, as illustrated in the following example.

Example 3.3 Suppose histogram H_1 has been allocated 3 buckets and has $m_1(3, 4) = 30$, $m_1(4, 5) = 130$. We also have histogram H_2 with 4 buckets and has $m_2(4, 5) = 40$, $m_2(5, 6) = 40$. Assume there are 2 more buckets to assign. *GOHGA* will choose to raise the number of buckets of H_2 by 2 instead of assigning the two buckets to H_1 to get the optimal result.♣

This problem can be avoided if we could get the greedy algorithm to examine not just the single step marginal gain, but also the marginal gain after more steps. Doing this indefinitely is not cost effective. However, checking just one additional bucket allocation is possible to do, at limited additional computational cost. Based on this idea, we propose a modification to the greedy algorithm.

From the results of step 3 of the greedy algorithm, we perform the following steps repeatedly until there is no further gain:

1. We check every histogram whether there is an inversion if two extra buckets are assigned. We then determine the largest marginal gain of step-width-2 and its corresponding histogram. Then, we find the two smallest marginal gain of step-width-1 that the buckets have already been assigned. If their sum is less than the largest marginal gain of step-width-2 found, we perform the following remedy: we decrease the bucket allocation of the first two histograms by 1 each and increase the latter histogram's bucket allocation by 2.
2. We transfer one bucket from a histogram to another. The transfer is considered beneficial if $\sum \sigma^{avg}$ is reduced. If the transfer is not beneficial, we undo the allocation.

4. PERFORMANCE STUDY

In this section, we experimentally study the benefits of *GOH* compared to the histograms constructed by assigning each one the same number of buckets. Then we discuss the efficiency of the algorithms. We denote the algorithm that constructs each V-Optimal histogram with the same number of buckets using dynamic programming as *DP* [12], the exhaustive algorithm as *GOHEA*, the dynamic programming algorithm as *GOHDP*, the greedy algorithm without remedy as *GOHGA* and the greedy algorithm with remedy as *GOHGR*.

We investigate the gain of *GOH* for estimating range query result size. Absolute error and relative error are applied as the ultimate error metric. The average error due to a histogram was computed over a set of queries. That is, for a set of L queries. The average absolute error E^{abs} was computed as:

$$E^{abs} = \frac{1}{L}\sum_{i=1}^{L} e_i^{abs}$$

and the average relative error E^{rel} was computed as:

$$E^{rel} = \frac{1}{L}\sum_{i=1}^{L} e_i^{rel}$$

We can now evaluate the quality of a collection of histograms. These histograms should be regarded as one entity because they are optimized as a whole. With a weighting corresponding to the frequency of access p_i for an attribute, and each absolute error of a query as $e_i^{abs} = |S_i - S_i'|$, we define the absolute error upon a group of histograms as:

$$\alpha^{abs} = \sum_{i=1}^{M} p_i|S_i - S_i'| = \sum_{i=1}^{M} p_i e_i^{abs}$$

and the relative error as:

$$\alpha^{rel} = \frac{\alpha^{abs}}{\sum_{i=1}^{M} p_i S_i}$$

where M is the number of histograms, S_i is the actual size of a query and S_i' is the estimated size of the query by histogram h_i. As p_i is set as 1 in section 2, every attribute in the group has the same probability to be accessed. we have the absolute error as:

$$\alpha^{abs} = \sum_{i=1}^{M} e_i^{abs}$$

and the relative error as:

$$\alpha^{rel} = \frac{\alpha^{abs}}{\sum_{i=1}^{M} S_i}$$

The average error due to a collection of histograms should be computed the same as that for a histogram:

$$E_{group}^{abs} = \frac{1}{L}\sum_{i=1}^{L} \alpha_i^{abs}$$

$$E_{group}^{rel} = \frac{1}{L}\sum_{i=1}^{L} \alpha_i^{rel}$$

4.1 A Comparative Study on Effectiveness

The data sets we used in our experiments included synthesized zipf data [20], TPC-D data [1] and UCI KDD Database Repository [2]. The results of our experiments did not vary significantly for different data sets or different range query sets. To illustrate the benefits of *GOH* and accuracy of the algorithms proposed, we present test results on two groups of data sets.

For both data sets, the range queries executed were of the form $b \leq X \leq a$, $b \leq X$ and $X \leq a$, where a and b were generated randomly and was in the domain of each data set. On each data set, $10,000$ such queries were executed and average absolute and relative error were calculated. Here we only present the results of $X \leq a$ as the results of the other query types are similar.

It was reported in [18], each histogram was stored in a catalog in the order of 200 bytes. As such, the total number of buckets used in our study are in increments of 25.

4.1.1 Test Group 1

The first test group contained five data sets. All the five data sets were synthesized zipf data with $z = 0, z = 0.01, z = 0.1, z = 1.0$ and $z = 2$ individually. Each data set was generated with the size of $500,000$ and $N = 500$. The frequencies of the synthesized zipf data were mapped to the values randomly.

Our first experiment is to study the effectiveness of the various schemes given that the total number of buckets available is 75. For DP, each histogram was assigned 15 (= 75/5) buckets. Figure 1 shows the results on the average absolute error for each histogram (data sets 1 to 5) as well as the average absolute error for the collection of 5 data sets. From the figure, we note that all the three schemes GOHEA, GOHDP and GOHGR have the same average relative error. This shows clearly that GOHGR's bucket allocation is optimal in terms of the optimization criterion. We note that all the schemes do not have any error for data set 1. This is because data set 1's distribution is uniform. It turns out that all GOH-based schemes allocate only 1 bucket to this data set. Moreover, the answers for the queries can be determined accurately. Next, we also note that the GOH-based schemes have a larger error for data sets 2 and 3 than the DP scheme, but the errors for data sets 4 and 5 for the GOH-based schemes are significantly lower than the DP scheme. These results are expected as the GOH-based schemes favor highly skewed distributions by allocating more buckets to them at the expense of lowly skewed distributions.

Finally, we observe that the overall performance for GOH-based schemes are superior compared to the DP scheme (see the group result in figure) – we see that the average absolute error decreased by nearly a half, from 2997 to 1545.

Figure 2 shows the result of the average relative error. As shown, for lowly skewed data sets (sets 2-3), the relative error for GOH-based schemes are slightly worse than the DP scheme. However, the difference is negligible. On the other hand, for highly skewed data sets (set 4 and 5), the GOH-based schemes continue to perform well, in the sense that the relative error remains low. For the DP scheme, the relative error increases significantly as the data distributions become more skewed. Again, on the whole, GOH-based scheme is shown to be more effective than the DP scheme with the average relative error decreased over 4 times, from 0.95% to 0.22%. The explanations for these results are similar to that for the absolute error results, i.e., GOH-based schemes are able to allocate more buckets to highly skewed data sets and thus allow the histograms to reflect more accurate summary data.

Figure 3 shows the average standard deviation results. While the average standard deviation of GOH-based schemes for data sets 2 and 3 is slightly larger than the DP scheme, that of data sets 4 and 5 was much lower. The resultant effect is that the GOH-based schemes' total average standard deviation is less than half that of the DP scheme.

Looking at the three figures (Figures 1-3), we observe that there is a correlation in the average standard deviation results with the average absolute and relative errors, i.e., the relative performance among the various schemes are the same in all these figures. This is expected. Since a low average standard deviation implies a more accurate histogram representation, the relative and absolute errors would also be low. This result is confirmed in the three figures. When a histogram is assigned more buckets, it captures the original data distribution more accurately. For data sets 2 and 3, GOH-based schemes allocate fewer buckets to represent the histogram than the DP scheme. As a result, the average standard deviation for the GOH-based schemes are larger than the DP scheme, and so are the average absolute and relative errors. On the other hand, for data sets 4 and 5, GOH-based schemes assign these histograms more buckets, so the average standard deviation, average absolute and average errors are lower than those of DP. It turns out that the effect on the overall allocation shows similar correlation.

We also evaluated the effectiveness of the proposed schemes as the total number of buckets varies from 25 to 100. The corresponding bucket allocation for each histogram under DP is given by (total number of buckets/5). We shall only present the results for the collection of histograms, rather than individual histogram.

Figure 4 and Figure 5 show that *GOH*-based schemes scale very well with different assignment of space. First, we observe that as the number of buckets increases, all scheme's errors also reduces. Second, the average absolute errors for GOH-based schemes are about half that of the DP technique, and their average relative errors are also much lower than the DP scheme. Third, as shown in Figure 5, we see that the solution obtained by GOHGA is a little worse than that by the other GOH-based schemes in the test of total number of buckets 50. This is because GOHGA is greedy in nature, and may miss the optimal solution. Finally, Figure 6 shows the average standard deviation results as the number of buckets varies. The result is consistent with earlier observations that the GOH-based schemes can outperform the DP scheme by a wide margin by having smaller average standard deviation.

4.1.2 Test Group 2

The second test group contained five data sets. They were extracted from TPC-D data sets with skewed data [5]. We took the 4th attribute of customer.tbl, the 6th attribute of part.tbl, the 4th attribute of supplier.tbl, and the 3rd and 5th attributes of lineitem.tbl as test data sets. The TPC-D data was populated with the scale 0.2 and $z = 1.0$. In total, the total cardinality of the datasets contain about 2.36M tuples.

As before, we shall report the results when we use a total of 75 buckets first. Figure 7 shows the absolute error result. As in the result for the test group 1, the absolute errors for the GOH-based schemes for data sets 1, 2 and 3 is larger than the DP technique, while that for data set 4 is slightly lower and that for data set 5 is very much smaller. For the GOH-based schemes, the group's average absolute error is nearly a half that of the DP scheme.

Figure 8 shows the relative error result. While the relative performance is largely the same as that for test group 1 for individual histograms, the difference between GOH-based schemes and the DP schemes for some individual histograms is greater. For example, for data set 3, GOH-based schemes is 8 times worse than the DP schemes. However, we see that for the overall result, the GOH-schemes remain superior - while the relative error for DP is 1.07%, the relative error for GOH-based schemes is about 0.46%. Upon investigation, we note that this is because data sets 1, 2 and 3 have relatively fewer number of tuples and fewer number of distinct values compared to data sets 4 and 5. The penalty of the quality of histograms of data sets 1, 2 and 3 is not comparable to the improvement of the quality of histograms of data sets 4 and 5.

Finally, Figure 9 shows that the relative performance of the various schemes remain largely the same for the standard deviation result. The total of average standard deviation of GOH-based schemes is less than half that of the DP scheme.

We also repeated the experiments when the total number of buckets are 25, 50, 75 and 100. The results of the experiments are shown in Figure 10, Figure 11 and Figure 12. As shown, the relative performance between the GOH-based schemes and the DP algorithm is largely similar to test group 1's results, i.e., GOH-based schemes outperform DP scheme.

To summarize, the results of the experiments indicate that for all the GOH-based schemes, the average absolute and relative errors of the histograms was much lower than the DP scheme. Among the GOH-based schemes, the greedy algorithms are comparable to the optimal schemes. Moreover, we note that GOHGR can outperform GOHGA with-

out excessive overhead (as we shall see in the subsequent experiments).

4.2 On the Effectiveness of GOH

In the last subsection, we observe that the benefits of *GOH* differs considerably in different test groups — different combination of data sets. In this section, we discuss about when *GOH* gain the most. More experimental results are presented to support our discussion.

Intuitively, compared to assigning the same number of buckets to different histograms, when a histogram of a data distribution with high marginal gain and a histogram of a data distribution with low marginal gain are both constructed in a catalog, *GOH* can gain much by allotting the one with high marginal gain many more buckets than the one with low marginal gain. For example, if one histogram is built for a very irregular distributed data and another is constructed for a uniformly distributed data, *GOH* will benefit significantly. On the contrary, building histograms for data distributions with little difference of marginal gain, the gain of *GOH* will be low.

In the following experiment, 6 sets of zipf data were synthesized. Each had the size of $500,000$ and $N = 698$. z was varied from 0 to 2.0. The frequencies of the synthesized zipf data were mapped with the values in decreasing order. The data sets with $z = 0$ and $z = 2.0$ were assigned to test group 3. The data sets with $z = 0.02$ and $z = 1.80$ were assigned to test group 4. And the data sets with $z = 1.8$ and $z = 2.0$ were assigned to test group 5. The data distributions are depicted in Figure 13. We applied GOHDP to construct *GOH* for each of these three test groups. Each *GOH* has 30 buckets. For comparison, a histogram with 15 buckets for each data set were constructed by DP schemes. The average absolute and relative error for each test group are presented in Figure 14 and Figure 15.

It is clear from the figures that in test groups 3 and 4, the *average error* were reduced greatly by *GOH* compared to DP, nearly 6 times for test group 3 and nearly 3 times for test group 4, while the average error of test group 5 decreased little, about 3%. To understand this phenomenon, let us examine the data distribution in Figure 13. We see that when the z value is high, $z = 1.8$ and $z = 2.0$, the data distribution is highly skewed, but when z value is small, $z = 0$ and $z = 0.02$, all values have approximately the same number of occurrences. For test groups 3 and 4, we combined highly skewed data (with $z = 2.0$ and $z = 1.8$) with uniformly distributed data (with $z = 0$ and $z = 0.02$). As a result, more buckets are being allocated to the highly skewed data (which has higher marginal gain) by GOH-based schemes. Thus, the *GOH* obtained reduced the error significantly compared to the DP scheme. In test group 5, the two data distributions combined were with little difference of skew, that is to say, with almost the same marginal gains. Thus, the benefit of *GOH* diminishes.

4.3 A Comparative Study on Efficiency

In this section, we show the time efficiency of the algorithms: DP, GOHGA, GOHGR, GOHDP and GOHEA.

In this set of experiments, 8 data sets are generated with

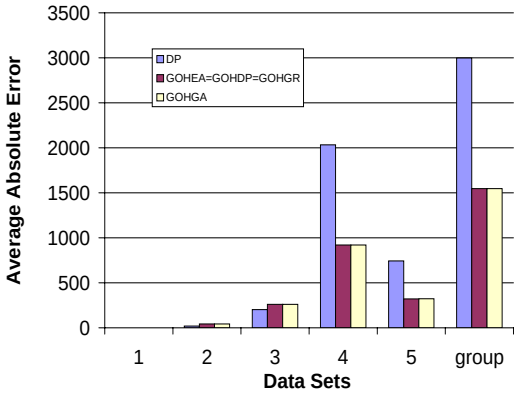

Figure 1: Absolute Error of Test Group 1(Total number of buckets: 75)

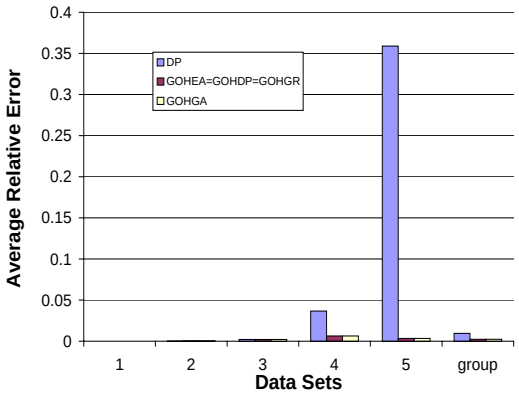

Figure 2: Relative Error of Test Group 1(Total number of buckets: 75)

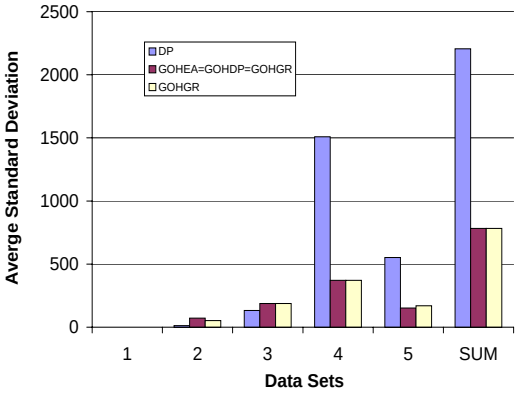

Figure 3: Average Standard Deviation of Test Group 1(Total number of buckets: 75)

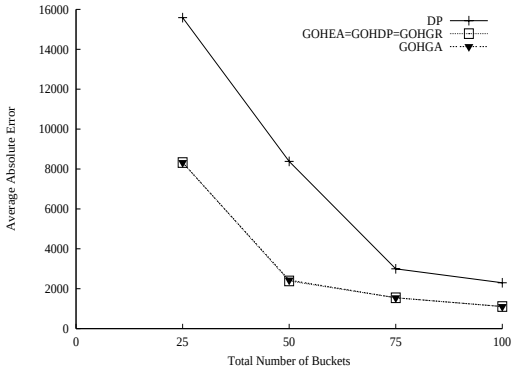

Figure 4: Effect of Number of Buckets on Average Absolute Error(Test Group 1)

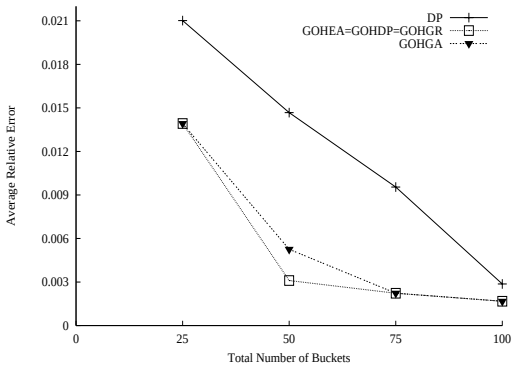

Figure 5: Effect of Number of Buckets on Average Relative Error(Test Group 1)

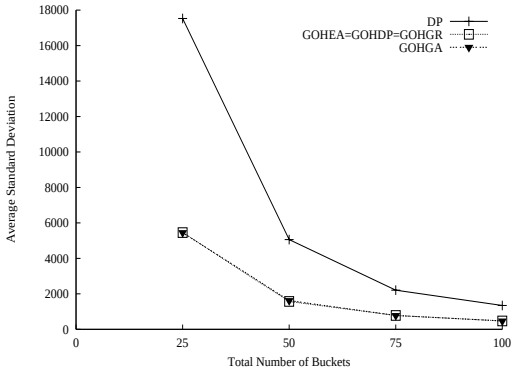

Figure 6: Effect of Number of Buckets on Average Standard Deviation(Test Group 1)

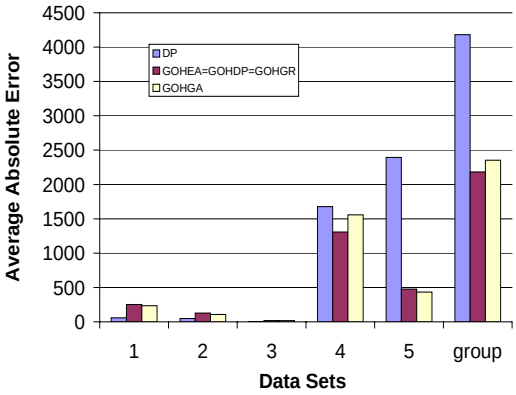

Figure 7: Absolute Error of Test Group 2(Total number of buckets: 75)

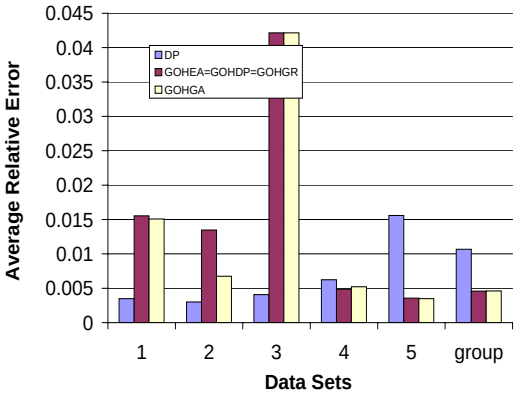

Figure 8: Relative Error of Test Group 2(Total number of buckets: 75)

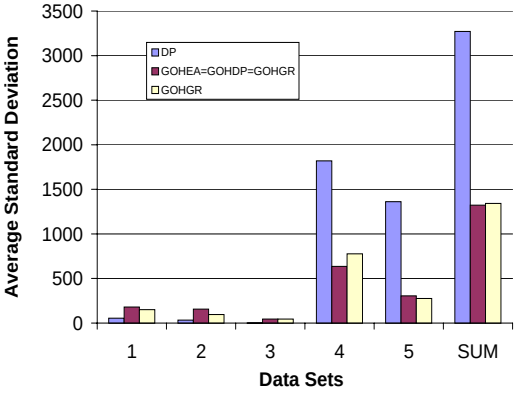

Figure 9: Average Standard Deviation of Test Group 2(Total number of buckets: 75)

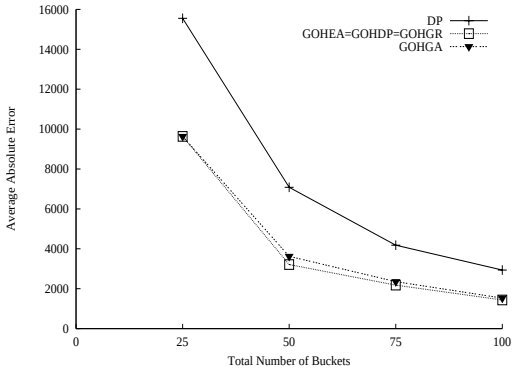

Figure 10: Effect of Number of Buckets on Average Absolute Error(Test Group 2)

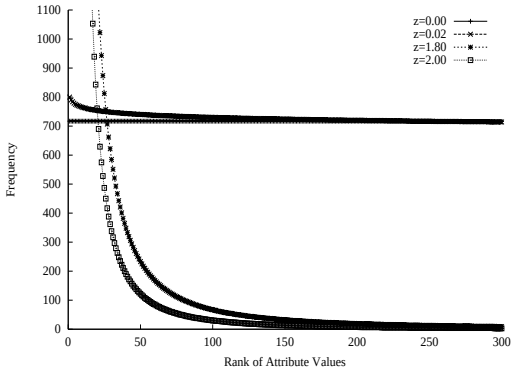

Figure 13: Zipf data distribution

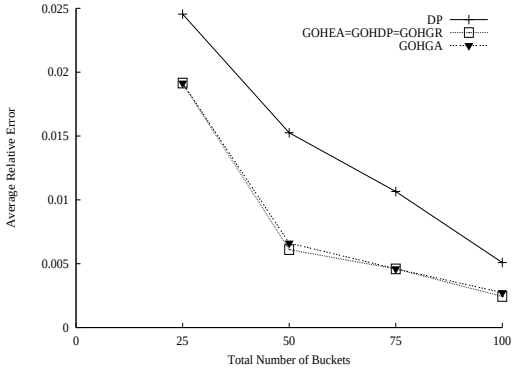

Figure 11: Effect of Number of Buckets on Average Relative Error(Test Group 2)

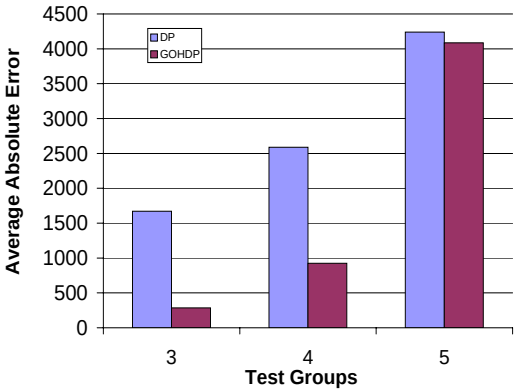

Figure 14: Average Absolute Error

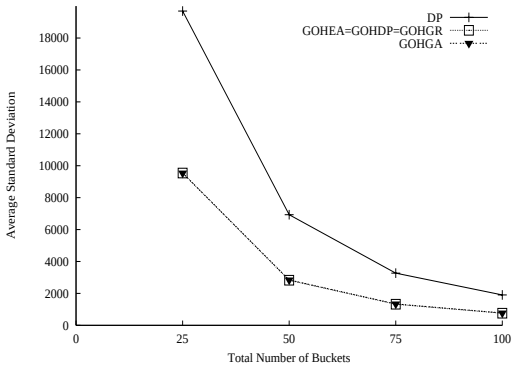

Figure 12: Effect of Number of Buckets on Average Standard Deviation(Test Group 2)

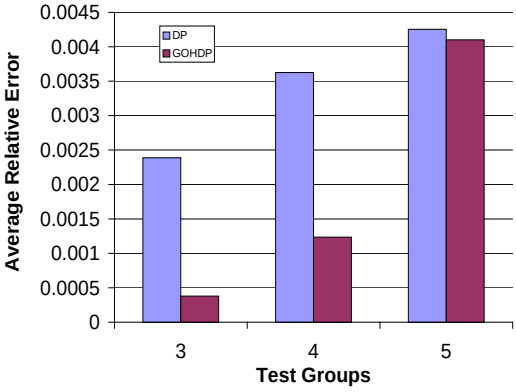

Figure 15: Average Relative Error

$z = 0$, $z = 0.02$, $z = 0.04$, $z = 0.06$, $z = 0.08$, $z = 0.10$, $z = 1.80$ and $z = 1.00$ respectively. We shall denote these data sets as DS_i, $1 \leq i \leq 8$. Each data set has the size of 500,000 and $N = 698$. The frequencies of the zipf data are mapped with the values in decreasing order. A total of 7 experiments are conducted. Experiment j contains data sets DS_1 to DS_{j+1}. The total number of buckets allocated to each experiment is proportional to the number of data sets, with each data set having an average of 15 buckets.

The time spent to construct each group is shown in Figure 16. As shown, GOHEA is the most inefficient. As the number of histograms and buckets increase, the processing time of GOHEA increases exponentially. Thus, GOHEA is not a practical strategy, and has been included in our study as a basis to measure the effectiveness of the other schemes.

For GOHDP, we also observe that its processing time increases dramatically as the number of histograms increases. We also note that the time spent by GOHGA and GOHGR are comparable and are both very close to that by DP. Since both algorithm generate better GOH than DP, this makes the greedy algorithms promising strategies to adopt. These results confirm our analysis in Section 3.

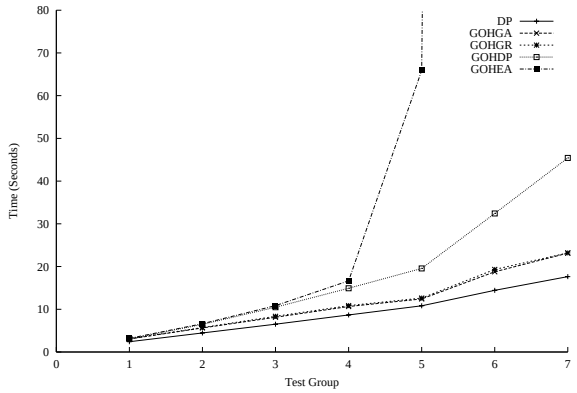

Figure 16: Running time of various algorithms

5. CONCLUSIONS

In this paper, we have proposed a novel idea of *Global Optimal Histograms*. Instead of generating an optimal histogram for each attribute (independent of other attributes/histograms), we generate a collection of histograms. By having a global view of the collection of histograms, it allows us to allocate more buckets (space) to histograms that represent highly skewed data and/or more frequently accessed data, and fewer buckets to histograms that captures uniformly distributed data. As a result, the overall quality of the histograms improves, which means that the system can provide overall better statistics.

We developed a dynamic programming algorithm for computing *GOH* in time proportional to the square of the number of distinct data values. The polynomial running time of this algorithm when compared with the polynomial running time of dynamic programming techniques to compute the V-optimal histogram for individual attributes in isolation, such as [12], is worse only by a factor equal to the number of histograms.

We also developed a greedy algorithm, and a remedy for common errors in the greedy algorithm, which are much more time efficient than dynamic programming.

Our extensive experiments and results show the benefits of *Global Optimal Histograms* using V-Optimal histograms and the *absolute error* and *relative error* metric. *GOH* was shown able cut error by a factor greater than 5 compared to the traditional approach. The experiments showed that the extent of quality improvement varied in different data sets. When highly skewed data sets joined with uniform distributed data sets, the gain of *GOH* was the largest. Our results also show that the greedy algorithm with remedy approximated *GOH* very well (achieved *GOH* in all of our experiments) without sacrificing computational efficiency.

In future research, we plan to apply *GOH* in constructing a set of Maxdiff histograms, because Maxdiff histogram gives best overall performance [18] for highly skewed data, and also to extend *GOH* to multi-dimensional histograms.

6. ACKNOWLEDGEMENTS

The work of H.V. Jagadish was supported in part by NSF under grant IIS-0002356.

7. REFERENCES

[1] *TPC BENCHMARK D(Decision Support).* Transaction Processing Performance Council http://www.tpc.org.

[2] UCI KDD Database Repository – the most popular datasets used for research in machine learning and knowledge discovery . http://kdd.ics.uci.edu.

[3] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. The Aqua Approximate Query Answering System. In *Proceedings of the ACM SIGMOD Conference*, pages 574–576, 1999.

[4] D. Barbará, W. DuMouchel, C. Faloutsos, P. J. Hass, J. M. Hellerstein, Y. E. Ioannidis, H. V. Jagadish, T. Johnson, R. Ng, V. Poosala, K. A. Ross, and K. C. Sevcik. The New Jersey Data Reduction Report. *IEEE Data Engineering Bulletin*, 20(4), 1997.

[5] S. Chaudhuri and V. R. Narasayya. TPC-D Data Generation with Skew. Available via anonymous ftp from ftp.research.microsoft.com/users/viveknar/tpcdskew.

[6] S. Chaudhuri and V. R. Narasayya. Automating Statistics Management for Query Optimizers. In *Proceedings of the International Conference on Data Engineering*, pages 339–348, 2000.

[7] S. Christodoulakis. Implications of Certain Assumptions in Database Performance Evaluation. *ACM Transactions on Database Systems*, 9(2):163–186, 1984.

[8] P. B. Gibbons and Y. Matias. New Sampling-Based Summary Statistics for Improving Approximate Query

Answers. In *Proceedings of the ACM SIGMOD Conference*, pages 331–342, 1998.

[9] Y. E. Ioannidis. Universality of Serial Histograms. In *Proceedings of the 19th International Conference on Very Large Databases*, pages 256–267, 1993.

[10] Y. E. Ioannidis and S. Christodoulakis. On the Propagation of Errors in the Size of Join Results. In *Proceedings of the ACM SIGMOD Conference*, pages 268–277, 1991.

[11] Y. E. Ioannidis and V. Poosala. Balancing Histogram Optimality and Practicality for Query Result Size Estimation. In *Proceedings of the ACM SIGMOD conference*, pages 233–244, 1995.

[12] H. V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. C. Sevcik, and T. Suel. Optimal Histograms with Quality Guarantees. In *Proceedings of the 24th International Conference on Very Large Databases*, pages 275–286, 1998.

[13] M. V. Mannino, P. Chu, and T. Sager. Statistical Profile Estimation in Database Systems. *ACM Computing Surveys*, 20(3), 1988.

[14] Y. Matias, J. S. Vitter, and M. Wang. Wavelet-Based Histograms for Selectivity Estimation. In *Proceedings of the ACM SIGMOD conference*, pages 448–459, 1998.

[15] M. Muralikrishna and D. J. DeWitt. Equi-Depth Histograms for Estimating Selectivity Factors for Multi-Dimensional Queries. In *Proceedings of the ACM SIGMOD Conference*, pages 28–36, 1988.

[16] G. Piatetsky-Shapiro and C. Connel. Accurate Estimation of the Number of Tuples Satisfying a Condition. In *Proceedings of the ACM SIGMOD conference*, pages 256–276, 1984.

[17] V. Poosala and Y. E. Ioannidis. Selectivity Estimation Without the Attribute Value Independence Assumption. In *Proceedings of the 23rd International conference on Very Large Databases*, pages 486–495, 1997.

[18] V. Poosala, Y. E. Ioannidis, P. J. Hass, and E. J. Shekita. Improved Histograms for Selectivity Estimation of Rage Predicates. In *Proceedings of the ACM SIGMOD Conference*, pages 294–305, 1996.

[19] P. G. Selinger, D. D. Astrahan, R. A. Chamberlain, R. A. Lorie, and T. G. Price. Access Path Selection in a Relational Database Management System. *Proceedings of the ACM SIGMOD conference*, pages 23–34, 1979.

[20] G. Zipf. *Human Behavior and the Principle of Least Effort.* Addison Wesley, 1949.