Computers and Mathematics with Applications 57 (2009) 1037–1047

Contents lists available at ScienceDirect

Computers and Mathematics with Applications

journal homepage: www.elsevier.com/locate/camwa

A new approach to building histogram for selectivity estimation in query processing optimization[☆]

Xin Lu[a], Jihong Guan[b,*]

[a] *Department of Computer Sci. and Eng., Fudan University, Shanghai 200433, China*

[b] *Department of Computer Sci. and Techl., Tongji University, Shanghai 200092, China*

ARTICLE INFO

Keywords:
Selectivity estimation
Histogram
Query optimization

ABSTRACT

Recently, histograms have been considered as an effective way to produce quick approximate answers to decision support queries. They are also taken as a basic tool for data visualization and analysis. In this paper, we propose a new approach to constructing histograms for selectivity estimation in query processing optimization. Our approach uses a new criterion, i.e., aggregate error minimization, to direct the construction of the target histogram. We develop the algorithm of aggregate error minimization based histogram construction, and demonstrate the effectiveness and efficiency of the proposed approach by experiments over both real-world and synthetic datasets.

1. Introduction

In database systems, the query processing optimizer is an essential module that generates the plan of how a query will be executed. Selectivity estimation plays a key role in the query optimization process. Inaccurate selectivity estimation may lead to the choice of a suboptimal execution plan, and the execution times of an optimal plan and a suboptimal plan may be substantially different.

Consider a relation R, an attribute A of R, and the domain D of A. Suppose there are n tuples in R, and these tuples' values of attribute A are $x_1, x_2, \ldots, x_n$. For an arbitrary value a or interval r in D, the *selectivity estimation* task is to evaluate how many tuples whose values of attribute A are equal to a or fall in r. To this end, we should have the exact or an approximate distribution of attribute A's values over domain D.

There exist a number of statistical methods used in commercial databases for the purpose of selectivity estimation. In the database academia, a lot of effort has also been put on this problem over the past two decades, and a number of approaches have been proposed, including histogram-based approaches [1–14], sampling-based approaches [15,16], and wavelet-based synopses [17] etc.

Histogram is one of the most natural and useful forms of data representation for the purpose of data summary and analysis; it is also a very popular and flexible way to track data distribution in databases. As a research topic that has been studied extensively, quite a number of algorithms have been proposed for efficiently constructing histograms over a single attribute [3–8] or multiple attributes [9–14]. Given a dataset and a fixed amount of storage space, a histogram is constructed to optimize a certain goal function. Usually, it is used to minimize the selectivity estimation error caused by the discrepancy between the built histogram and the real data distribution. In other words, the goal is to make the histogram be as close as possible to the real data distribution.

☆ This work is supported by the National Natural Science Foundation of China under grants — no. 60496327, no. 60573183 and no. 60773123; Jihong Guan is also supported by the Program for New Century Excellent Talents in University of China (NCET-06-0376).

* Corresponding address: Department of Computer Science & Technology, School of Electronics and Information, Tongji University, Shanghai 200092, China. Tel.: +86 21 65109899; fax: +86 21 55664298.

E-mail addresses: luxin@fudan.edu.cn (X. Lu), jhguan@mail.tongji.edu.cn (J. Guan).

0898-1221/$ – see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.camwa.2008.10.056

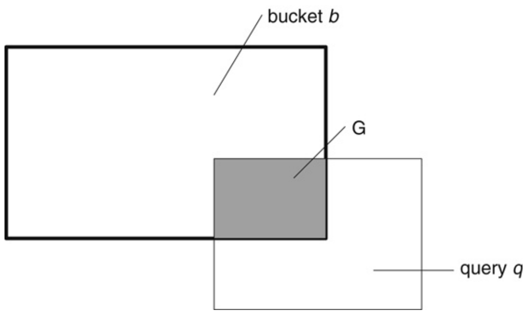


Fig. 1. Illustration of estimating the selectivity inside a bucket.

In this paper, we propose a new concept of error, *aggregate error*, for histogram construction. The *relative aggregate error* is used in histogram construction. Algorithm is developed to build histograms by minimizing the error value. Experiments over both real-world and synthetic datasets are carried out to evaluate the performance of the proposed method.

The remaining of this paper is organized as follows. Section 2 surveys related work of selectivity estimation. Section 3 states the problem of histogram construction. Section 4 introduces the approach to building histograms based on aggregate error minimization. Section 5 presents performance evaluation. And finally Section 6 conclude the paper and highlights future work.

2. Related work

In the database community, selectivity estimation is an important research issue of query processing optimization. Up to now, a number of techniques have been proposed for selectivity estimation, including histogram-based approaches, sampling-based approaches, and wavelet-based technique etc. Considering that this paper is about histogram construction, we survey mainly the approaches of histogram-based selectivity estimation.

Wavelet-based technique is a parametric approach, which was proposed as the replacement to histograms [17]. It uses wavelet decomposition for approximation. Wavelet decomposition is, roughly, the decomposition of a function into wavelets with hierarchical levels of detail. Wavelets are functions that look like decaying oscillating waves. There exist many kinds of wavelets, the most popular are Haar and linear ones. One wavelet represents the rough picture of data distribution over the entire domain. Wavelet-based approaches are inefficient because of the large amount of computation of matrix transformation.

Sampling-based methods have some advantages as follows: (1) They need not collect, store and maintain any statistical information; (2) They do not require imposing any a priori assumption on data distribution; (3) They are suitable for high-dimensional data. For these reasons, sampling-based methods for selectivity estimation have attracted considerable attention of database researchers. An exhaustive overview of classic sampling methods was presented as a survey by Olken and Rotem [15], which covers the research conducted in this area up to the year of 1990. And a more in-depth study on random sampling for use in database systems was given in Olken's Ph.D. dissertation [16].

Histograms are the most commonly used form of statistics in practice because they incur almost no run-time overhead and are effective even with a very small amount of storage space. They have been used in DB2, Oracle, and Microsoft SQL Server. Several types of histograms have been proposed and studied, including MinSkew [18], EquiHeight [21], GENHIST [20], STGrid [22], STHoles [9].

Minskew is a histogram construction algorithm originally proposed for selectivity estimation of range queries on non-uniform datasets [18]. At first, Minskew treats the whole data space as a bucket, then it partitions the bucket into a set of small buckets, each of which does not intersect with any other bucket, and the union of all buckets covers the entire data space. Each bucket b_i contains the number $b_i.num$ of objects that fall inside bucket b_i. Fig. 1 shows a query q that intersects with bucket b in two-dimensional space. The gray area corresponds to the intersection between bucket b and the query q. The expected number of objects in the intersection region G of b and q is estimated as $b.num * area(G)/area(b)$, where $area(G)$ and $area(b)$ are the areas of the intersection region G and bucket b, respectively. Generally, the estimated selectivity is obtained by summing the results of all intersecting buckets.

Since the estimation algorithm above applies uniform assumption within a bucket, the more uniform the intra-bucket distribution is, the more accurate the estimation will be. Minskew defines the spatial-skew (denoted as b.skew) for a bucket b as the variance of the spatial densities of all points inside it. It is obvious that a partitioning with small spatial-skewness is likely to be highly accurate in approximating the given data. So Minskew aims at minimizing the weighted sum of spatial-skews of all buckets. Unfortunately, building the optimal partitioning is NP-Hard [19]. To reduce the complexity of constructing good partitionings, Minskew partitions the original space to a grid with $H * H$ regular cells (where H is a parameter), and records the number of objects that fall in cell c as $c.num$. Thus, $b.num$ is equal to the sum of objects falling in all cells covered by bucket b. It then uses a greedy approach to seek a local optimal result for reducing computation cost.

EquiHeight [21] is an approach to constructing equi-height histograms. In an equi-height histogram, each bucket contains approximately the same number of data objects. Considering that data is usually unevenly distributed, a dense area will be divided into several contiguous buckets, and a dense bucket covers a smaller range. It has been shown that equi-height histograms perform well for range queries with a low-skew data distribution [21].

GenHIST, an approach to building multi-dimensional histograms, was presented in [20]. Unlike the histogram techniques discussed above, GenHIST builds overlapping buckets, and maintains average densities inside them. It was observed that b overlapping buckets divide the domain into more than b regions, which thus provides more information about intra-bucket data distribution.

STGrid [22] provides a less accurate, yet much expensive alternative to traditional multi-dimensional histograms for slightly to moderately skewed data. The main distinct feature of STGrid is that buckets are built without any scanning or even sampling of the data. First, the initial multi-dimensional histogram is built from whatever information is available (e.g., using available single attribute histograms to construct high-dimensional histogram by using the independence assumption). Then, the histogram is refined online or off-line by using the actual selectivities of query feedbacks. The more the STGrid method is used, the more accurate an estimation it provides.

Another query-feedback-based histogram technique, STHoles [9], borrows the basic idea from [22] and aims to build workload-aware histograms. In STHoles, a bucket has a form of box and stores the number of tuples enclosed in that box. The buckets are nested, i.e., each bucket, but the first(top) one, is fully enclosed in some other buckets. So, this is another example of histogram technique with overlapping buckets. However, unlike GenHist, STHoles does not allow the buckets to intersect partially: either one is fully enclosed in another, or they do not intersect at all.

In this paper, we propose a new optimization criterion for histogram construction. Our goal is to make intra-bucket distribution as uniform as possible. To this end, we introduce the aggregate error notion and take the minimization of total aggregate error as the optimization function. Actually, as a basic building module our approach can be applied to other histogram approaches to improve their performance.

3. Problem formulation

In this section, we define histogram and formulate the problem studied in this paper.

Consider a relation R containing an attribute X. The values set V of X is the set of values of X that are present in R. For each value $v \in V$, the frequency $f(v)$ of v is the number of tuples $t \in R$ with $t.X = v$. We assume that the elements of V have been sorted in ascending or decreasing order, and denote $f_i = f(v_i)$, the *frequency vector* of X is the ordered set of frequencies of various values in V, i.e., $F = \{f_1, f_2, \dots, f_N\}$, and $N = |V|$.

A *histogram* of data distribution X is constructed by partitioning the frequency vector F of X into B (≥ 1) intervals that are called buckets, and approximating the frequencies and values in each bucket. The result is an approximate data distribution that can be used in place of the actual distribution for selectivity estimation. Of course, the accuracy of any operation performed by 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 the buckets.

Several techniques for the approximation within the buckets 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 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. The latter has been experimentally shown to be more accurate than the former in several estimation problems [1].

The main focus of this paper, however, is on the partition technique. Let $X = \{x_1, x_2, \dots, x_n\}$ be a finite data sequence. The problem of *histogram construction* is as follows: given the storage space constraint B, we try to create a *compact representation HB* of the data sequence, which can be stored totally in B and be accurate in approximating X as closely as possible.

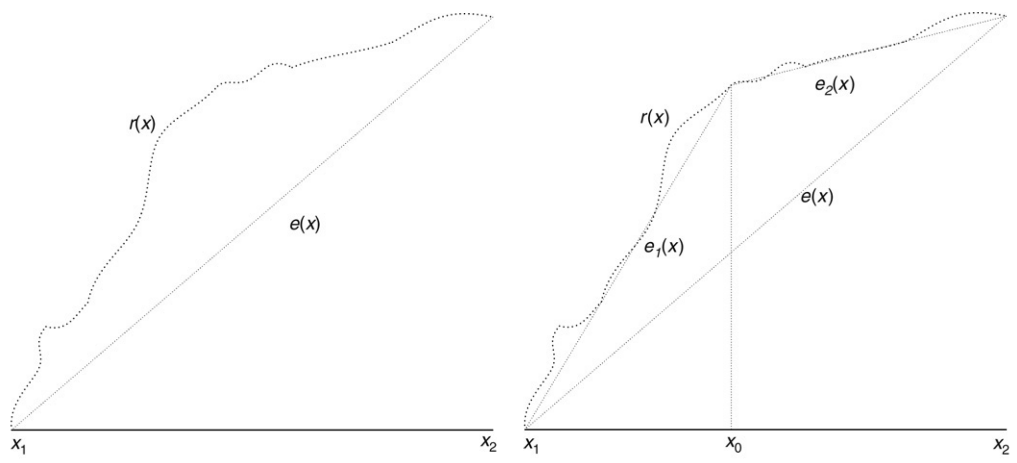
The representation HB collapses the values in a sequence of consecutive points $\{x_i\}$ where $i \in [s_r, e_r]$ into a single value $x(r)$, which forms a bucket b_r, that is, $b_r = (s_r, e_r, x(r))$. The histogram HB is then used to answer queries about the value at point i where $1 \leq i \leq n$. The histogram uses at most B buckets that cover the entire interval $[1, n]$, and saves space by storing only $O(B)$ data items, instead of $O(n)$.

Given a point query x_i, the histogram is used to estimate the value at x_i where $s_r \leq i \leq e_r$, and the result is $x(r)$. Since $x(r)$ is an estimate for the value in bucket b_r, we suffer an error. We give the formal definition of estimation error as follows.

Definition 1. Given a histogram, the *absolute error* and the *relative error* at point x_i where $i \in [s_r, e_r]$ are defined as $|x(r) - x_i|$ and $|x(r) - x_i| / \max\{c, |x_i|\}$ respectively. Here, c is a non-zero constant introduced to guarantee that the denominator of the relative error formula is not too small or even equal to zero.

4. Aggregate error minimization based histogram construction

In this section, we propose a new approach to build histogram for selectivity estimation in query processing optimization. We first introduce the notion of *aggregate error* (simply *AE*), including *absolute aggregate error* (simply *AAE*) and *relative aggregate error* (simply *RAE*). Then we present the histogram construction algorithm in one dimension space based on aggregate error minimization scheme. Finally, we extend the one-dimensional construction algorithm to multi-dimensional situation.



(a) $r(x)$ and $e(x)$. (b) Example of bucket partitioning.

Fig. 2. Estimation function and bucket partitioning.

The core part of a histogram construction algorithm is the partitioning technique employed for grouping the values into buckets. In the MinSkew algorithm mentioned above, a partitioning technique is exploited to partition the data space for minimizing the total skewness of the buckets, i.e, to ensure that data distribution in the buckets is as uniform as possible. Here, we try to minimize the relative aggregate error of the target histogram, which can lead to better query performance.

4.1. *Aggregate error*

Here we introduce the notion of aggregate error. For simplicity of exposition, we first put the problem in one dimension context, and extend it to high-dimensional space later.

Let W be a dataset of one dimension (corresponding to attribute X), we define a function $r(x)$, which records the number of data point $x_i \in W$ such that $x_i < x$. We term function $r(x)$ the *aggregate function* of attribute X. Obviously, $r(x)$ is a monotonic increasing function with regard to x. For example, suppose $W = \{0, 0, 1, 1, 1, 2, 5, 6, 6, 8, 9\}$, we can get $r(0) = 0, r(1) = 2, r(2) = 5, r(3) = 6$, and so on. We then define another function $e(x)$, which is an estimation of $r(x)$, i.e., it estimates the number of data point $x_i \in W$ such that $x_i < x$. We term function $e(x)$ the *estimation aggregate function* of attribute X. The problem is to choose a proper function $e(x)$, which is as simple as possible (for simplicity of computation), and as close to $r(x)$ as possible (as far as performance is concerned).

Usually, while estimating the size of a query by a histogram structure, it is assumed that the data in every bucket is uniform. In this paper, for simplicity of computation, we let $e(x)$ be a liner and incremental function.

Given the aggregate function $r(x)$ over interval $[x_1, x_2]$, we define its estimation aggregate function $e(x)$ as follows:

$$e(x) = \frac{r(x_2) - r(x_1)}{x_2 - x_1} \times (x - x_1) + r(x_1). \tag{1}$$

Fig. 2 illustrates aggregate function, estimation aggregate function and bucket splitting. Fig. 2(a) shows an aggregate function $r(x)$ and its corresponding estimation aggregate function $e(x)$. In Fig. 2(b), to approximate $r(x)$ with $e(x)$ more accurately, we split interval $[x_1, x_2]$ (corresponding to one bucket) into two sub-intervals (corresponding to two small buckets) at x_0. Accordingly, the estimation aggregate function $e(x)$ is replaced by two sub-functions $e_1(x)$ and $e_2(x)$.

To measure the deviation of the estimation aggregate function $e(x)$ from the original aggregate function $r(x)$, we introduce the concept of *aggregate error*, which includes *absolute aggregate error* and *relative aggregate error*. In what follows, we give their definitions respectively.

Definition 2. Given an interval $I = [x_1, x_2]$ of attribute X, its aggregate function $r(x)$ and estimation aggregate function $e(x)$, we define the *absolute aggregate error* between $r(x)$ and $e(x)$ over interval I as follows:

$$AAE(x_1, x_2) = \sum_{x=x_1}^{x_2} (e(x) - r(x))^2. \tag{2}$$

Definition 3. Given an interval $I = [x_1, x_2]$ of attribute X, its aggregate function $r(x)$ and estimation aggregate function $e(x)$, we define the *relative aggregate error* between $r(x)$ and $e(x)$ over interval I as follows:

$$RAE(x_1, x_2) = \sum_{x=x_1}^{x_2} \frac{(e(x) - r(x))^2}{\max(c^2, r(x)^2)}. \tag{3}$$

Note that in database applications, the practical values of any numerical attribute in any relation are discrete and finite, $r(x)$ is actually a monotonic increasing discrete function. This is the reason why we use sum operation rather than integral operation in Eqs. (2) and (3). The estimation aggregate function $e(x)$ is a segment-wise monotonic increasing function. From (2) and (3), we can see that the more closely $e(x)$ approximates $r(x)$ over $I = [x_1, x_2]$, the smaller $AAE(x_1, x_2)$ and $RAE(x_1, x_2)$ are. In next subsection, we will present a new histogram construction approach based on aggregate error minimization.

4.2. Histogram construction

Still in one-dimensional context, given an attribute X, its value interval $I = [x_s, x_e]$, the set W of values of attribute X over interval I. It is easy to get the aggregate function $r(x)$ of attribute X over interval I. Suppose the specified number of buckets is B, now we are to build a histogram H over attribute X based on aggregate error minimization. Formally, we are to split interval I into B sub-intervals. We denote the ith sub-interval as $I_i = [x_{s_i}, x_{e_i}]$ where $1 \le i \le B$, $x_{s_1} = x_s$, $x_{e_B} = x_e$, and $x_{e_j} = x_{s_{j+1}}$ for $1 \le j \le B - 1$. The splitting is to minimize the aggregate error of $r(x)$ over I. That is,

$$AAE(x_s, x_e) = \min\left(\sum_{i=1}^{B}\sum_{x=x_{s_i}}^{x_{e_i}}(e_i(x) - r(x))^2\right) \tag{4}$$

or

$$RAE(x_s, x_e) = \min\left(\sum_{i=1}^{B}\sum_{x=x_{s_i}}^{x_{e_i}}\frac{(e_i(x) - r(x))^2}{\max(c^2, r(x)^2)}\right). \tag{5}$$

Here, each sub-interval corresponds to a bucket, and $e_i(x)$ means the estimation aggregate function over sub-interval I_i, which is generated by Formula (1). We call the histogram built by using Formula (4) *absolute aggregate error minimization based histogram* (simply *AAEH*), and the one built by using Formula (5) *relative aggregate error minimization based histogram* (simply *RAEH*).

Obviously, to get global optimal solution of Formula (4) and (5) is *NP*-hard. So in this paper, we employ a greedy search scheme to obtain a suboptimal solution. The process of our approach is as follows:

(i) The 1st round: Splitting interval I into two sub-intervals such that the sum of aggregate errors over the two sub-intervals is minimized.
(ii) The 2nd round: Splitting each of the resulting sub-intervals of the first round into two parts, selecting the splitting of a sub-interval that can achieve the largest reduction of aggregate error as the final splitting of this round. Then we get three sub-intervals.
(iii) The ith round: we have gotten i sub-intervals after the previous $(i - 1)$ rounds of splitting. Splitting each of the i sub-intervals, and keep the splitting that can achieve the largest reduction of aggregate error as the final result of this round, then we get $(i + 1)$ sub-intervals.
(iv) The above process continues iteratively till we get B sub-intervals, each of which corresponds to a bucket.

Fig. 2(b) shows how to split an interval (corresponding to one bucket) into two smaller sub-intervals (corresponding to two buckets). We select the splitting point x_0 such that $AAE(x_1, x_0) + AAE(x_0, x_2)$ or $RAE(x_1, x_0) + RAE(x_0, x_2)$ is minimized.

4.3. Extending to multi-dimensional space

Above we have presented an approach to building histogram based on aggregate error minimization scheme in one dimension context, here we extend the approach to multi-dimensional space.

In the multi-dimensional context, we first treat the whole data space as a bucket. Consider any dimension D_i, we seek a candidate partitioning point p_i on dimension D_i which can get the largest aggregate error reduction after partitioning the bucket into two parts over this dimension. We can do that by utilizing the approach proposed in one dimension case. For every dimension, we can find such a candidate partitioning point, which leads to the largest aggregate error reduction over that dimension. Then we choose the dimension whose candidate partitioning point can generate the largest aggregate error reduction as our partitioning dimension, and partition the bucket along the selected dimension's candidate partitioning point. After the first round partitioning, we get two new buckets. In the second round of partitioning, we find a candidate partition point for each bucket by following the above method to the whole space, and choose the bucket to partition that bring the largest aggregate error reduction, thus three buckets are obtained. In such a way, we partition the buckets iteratively till the number of buckets reaches the specified quota B or the storage limit.

We give the aggregate error minimization based histogram construction algorithm in pseudo-code as follows.

In Algorithm 1, we first take the whole data space as a bucket b, and add the bucket b into the bucket list that will contain all buckets produced (Lines 1–2). If the number of buckets in the bucket list is less than the required number, we choose a bucket from the bucket list according to such a rule: the bucket that will lead to the largest aggregate error reduction (Lines 3–14). For every bucket b, we seek a candidate partitioning point that can incur the maximal aggregate error reduction. By doing comparison between buckets, we choose the bucket has the maximal aggregate error reduction (Lines 4–11). We split the bucket selected above, and add the two new generated buckets into bucket list and delete the old one (Line 12–13).

Algorithm 1 Aggregate error minimization based histogram construction

```
1:  whole space as a bucket $b$;
2:  list.add($b$);
3:  while ($list.size < requireNumber$) do
4:      for every bucket $b_i$ in list do
5:          $b_i.error \leftarrow$ the aggregate error of bucket $b_i$
6:          for every candidate partitioning point do
7:              compute the aggregate error of the two new buckets;
8:              $b_i.reduction \leftarrow$ error reduction;
9:          end for
10:         $B \leftarrow$ max error reduction bucket;
11:     end for
12:     buckets $b_1, b_2 \leftarrow partition(B)$;
13:     list.add($b_1, b_2$);
14: end while
```

5. Performance evaluation

In this section, we study the performance of the proposed approach and compare it with three existing methods. Experiments are conducted over both real-world and synthetic datasets. We first describe the datasets and queries used in the experiments, and then introduce briefly the three methods compared with our approach. Finally, we present the results of performance evaluation. All experiments are conducted on a PC with a P4 2.8 GHz CPU and 512 MB memory.

5.1. The datasets

We use both real-world and synthetic datasets in the experiments. Following are some details of the datasets.

- *Real-world dataset*: We use the NJ Road dataset from the TIGER database,[1] which is developed at the U.S. Census Bureau to support its mapping needs for the Decennial Census and other Bureau programs. TIGER database has been widely used for testing algorithms of GIS and spatial information processing. The NJ Road dataset contains the road information of the New Jersey State.
- *Synthetic datasets*: We generate one/multi-dimensional datasets of uniform, normal and Zipfian distribution. The Zipfian datasets have various levels of skewness, which is controlled by the parameter z of Zipfian distribution. We vary the value of z between 0.3 (low skewness) to 2.0 (high skewness).

5.2. The query sets

The query sets consist of a large number of range queries lying within the input data space. The centers of the range queries are selected randomly. On each dimension of the test data, the average width of the queries (referred to as parameter *QSIZE* in the experiments) is changed from 2% to 25% of the range covered by the test data in that dimension. For example, for two-dimensional data, the query area varies from 0.04% to 6.25% of the coverage of the test dataset. So for high-dimensional datasets, we should choose relatively larger query sizes (i.e., larger *QSIZE* values) to make the query space be large enough.

5.3. The evaluated methods

In this paper, we implemented two versions of the proposed approach: *absolute aggregate error minimization* based and *relative aggregate error minimization* based, which are referred to as AAEH and RAEH respectively. Considering that RAEH performs better than AAEH, we present only results of RAEH. We compare REAH with three existing methods: two of which are histogram-based, MinSkew and GenHist, and the other one is a sampling-based approach. MinSkew is a typical histogram approach, and GenHist represents the state-of-the-art technique. For all histogram-based approaches tested in this paper, we assume that data distributes uniformly within each bucket.

The GenHist algorithm starts from the uniform grid shaped buckets, and then iteratively expands the dense buckets. The buckets are built in such a way that the average density in a bucket is close to the average densities of its neighbor buckets. The most dense regions are partitioned into new buckets.

[1] Available at http://www.census.gov/geo/www/tiger/.

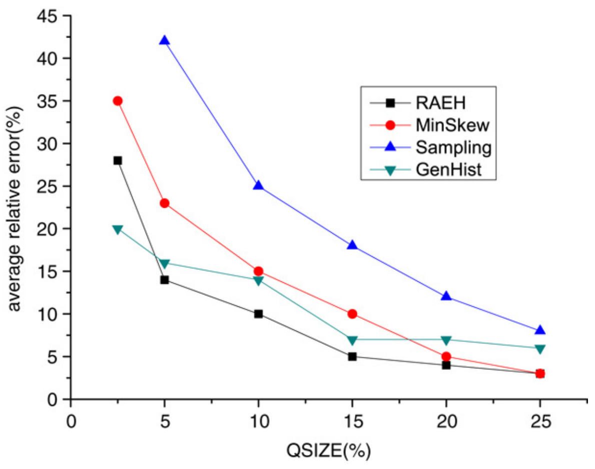


Fig. 3. Performance vs. query size.

The sampling approach works as follows. First, collecting a sample data from the input data. Given a query, the selectivity of the query is evaluated on the sample, and the result is then scaled appropriately to obtain the estimation of the query's selectivity. Suppose that the size of the sample is n, the original dataset size is N, and the number of sampled data objects that conform to the given query is m, then the estimated selectivity is $m * N/n$.

To compare the four approaches, we use performance metrics as follows: *average relative error*, *histogram construction time* and *selectivity estimation time*. The histogram construction time consist of the time taken by a construction algorithm to preprocess the data, build appropriate data structures, and generate the final buckets. Average relative error indicates the averaged relative error over a set of queries. Let e_i be the estimated answer and r_i is the real answer for a query q_i from the testing queries set Q, the average relative error is evaluated as follows:

$$\frac{\sum_{q_i \in Q} |e_i - r_i|}{\sum_{q_i \in Q} r_i}. \tag{6}$$

In experiments, we set the query-set size to 2500. For each query, we first measure its relative error of selectivity estimation, and then average the relative errors of all 2500 queries to get the averaged relative error.

5.4. Experimental results

In this subsection, we present the performance results of the four approaches in approximating query result size. We study how performance is impacted by query size, the numbers of samples/buckets, dimensionality, and dataset size. In each experiment, we vary the value of parameter under investigation and keep the remaining parameters fixed at their default values. Note that each bucket keeps two data points, if the histogram approaches use n buckets, the sampling method will use $2n$ samples correspondingly. Without specific indication, QSIZE is 20%; the number of samples/buckets is 1500/750, the dimensionality is 4, and Zipfian data is used. We also check the construction and computation efficiency of the evaluated methods.

5.4.1. Impact of query size

This experiment studies the impact of query size on performance over the NJ Road dataset. Fig. 3 shows how the relative error changes as query size increases from 2.5% to 25%. Here we use 100 buckets to approximate the NJ Road dataset.

From Fig. 3, we can see that the relative error value decreases with the increasing query size. This is because the error of estimation arises only from those buckets that are partially covered by the query. Since each bucket stores the exact number of data objects contained by itself, those buckets that fall entirely in the query range contribute no error. The larger the query is, the more buckets are fully covered the query, and relatively the fewer buckets contribute to the estimation error.

Though all methods show better accuracy as the size of query grows, RAEH outperforms the others in most of the QSIZE range.

5.4.2. Impact of dimensionality

Here we investigate the impact of dimensionality on performance. We use synthetic Zipfian datasets of size 20,000. The results are shown in Fig. 4.

From Fig. 4, it is clear that for all approaches evaluated the relative error grows with the increasing of dimensionality. For low-dimensional data ($d \leq 4$), histogram-based techniques outperform the sampling approach; while for high-dimensional

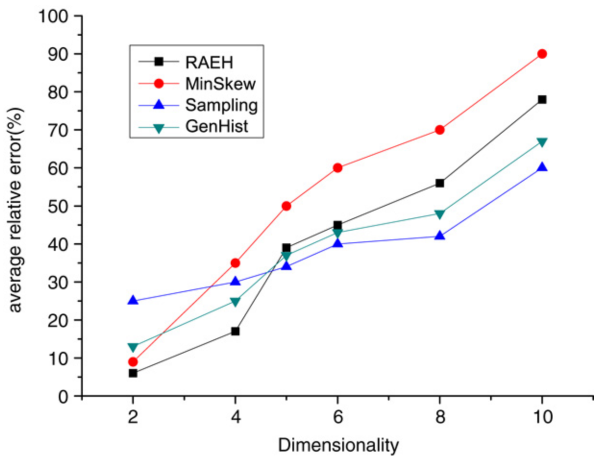


Fig. 4. Performance vs. dimensionality.

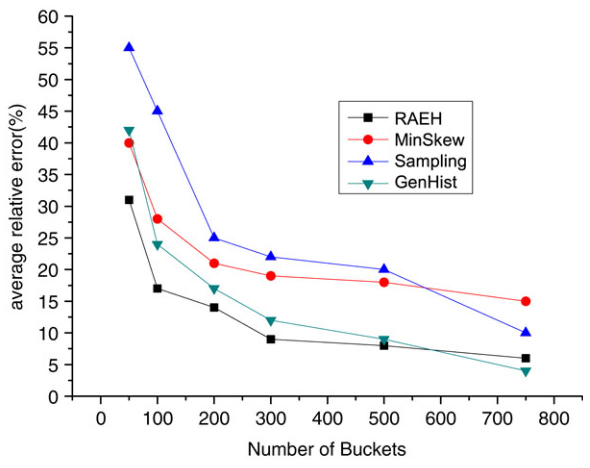


Fig. 5. Performance vs. number of buckets (for small queries with QSIZE = 10%).

data, the sampling approach surpasses the others. As the dimensionality increases, performances of histogram approaches degrade sharply, while the sampling approach's performance degrades moderately. The reason lies in that in the high dimension cases, most of the buckets intersect with the query window, and in the intersection parts, the assumption of data uniform distribution is invalid, so the error increase rapidly. It is shown that for a multi-dimensional histogram with non-overlapping buckets, a d-dimensional range query can intersect $O(b^{\frac{d-1}{d}})$ buckets (where b is the number of buckets) [20]. Although the sampling algorithm is not evidently influenced by dimensionality, data in the high-dimensional space is very sparse, the error still increases.

Among the histogram approaches, RAEH outperforms the other techniques for low-dimensional data ($d \leq 4$). For high-dimensional data, GenHist is the best one, and RAEH is the second.

5.4.3. Impact of the number of buckets

In the part, we examine the impact of the number of buckets on the performance of the evaluated approaches. We conduct two groups of experiments: one with different QSize values, the other with different data distributions.

Figs. 5 and 6 show the experimental results, which correspond to the results for small queries (QSIZE = 10%) and large queries (QSIZE = 25%) respectively. As expected, assigning more space for approximation can get better estimation. With more storage space, more detail of the input data can be approximated, thus the error can be reduced. Still, RAEH beats the others in almost all cases.

We can also see that when the number of buckets reaches 300, the error reduces very slowly as the number of buckets continues to grow. This means that there exists a threshold of storage for buckets. After exceeding the threshold, the effectiveness of error reduction is trivial, more storage means more waste.

Fig. 7 presents the results for three different data distributions: Zipfian, uniform, normal. For uniform distribution, the three histogram approaches have almost the same performance. This is natural and reasonable, because all the three

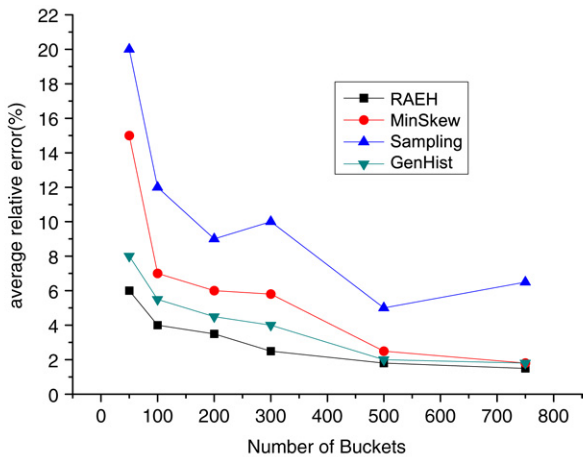
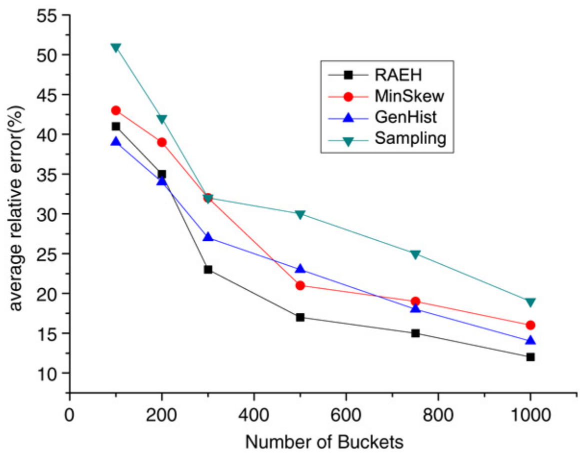
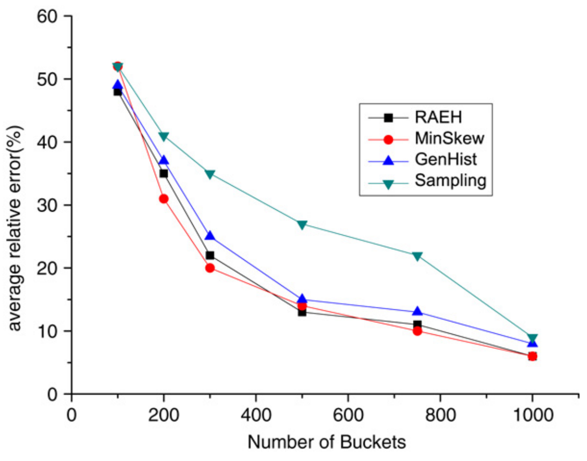


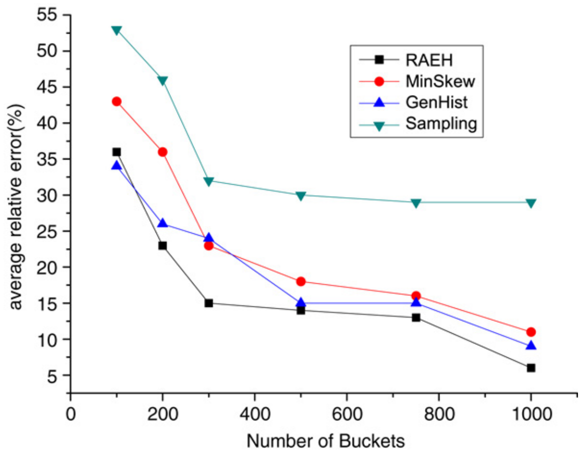
Fig. 6. Performance vs. number of buckets (for large queries with QSIZE = 25%).



(a) Zipf distribution.



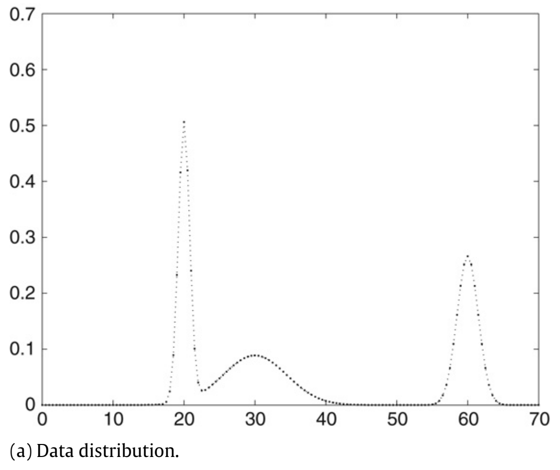
(b) Uniform distribution.



(c) Normal distribution.

Fig. 7. Performance vs. Number of buckets (for different distributions).

histogram approaches try to make intra-bucket as uniform as possible, and when the data is uniformly distributed, they will obtain nearly similar histograms. For Zipfian and normal distributions, our RAEH approach performs best, which indicates that our approach is effective in producing accurate histograms over non-uniform distribution datasets.



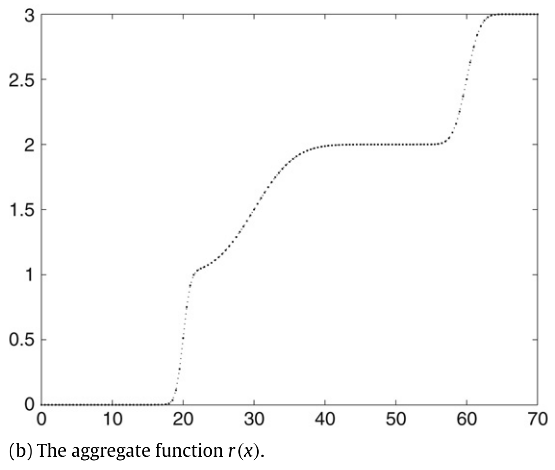


Fig. 8. One-dimensional dataset with mixed normal distributions and its aggregate function.

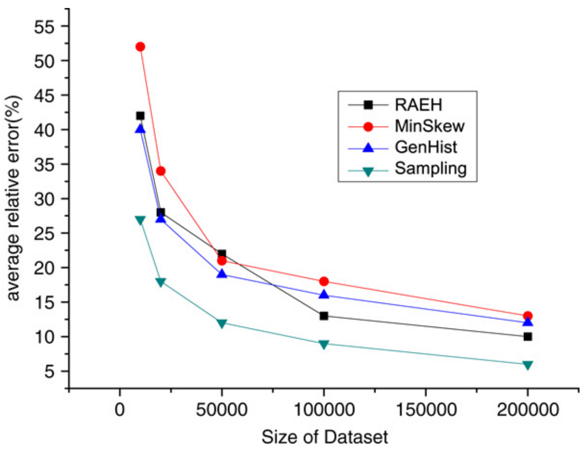


Fig. 9. Performance vs. size of dataset.

To further illustrate the advantage of our RAEH method, let us check a simple example in Fig. 8. Fig. 8(a) shows the distribution of a one-dimensional dataset, which is actually the mix of three normal distributions whose centers locate at 20, 30 and 60 respectively; Fig. 8(b) is the aggregate function of the dataset in Fig. 8(a). Essentially, our approach tries to approximate the aggregate function, while MinSkew and GenHist as well as most other histogram approaches aim at approximating the original dataset distribution. Comparing Fig. 8(a) and (b), obviously, the aggregate function is simpler and easier to approximate than the original data distribution. So it is easier and more accurate to approximate the aggregate function by our approach than to approximate the original data distribution with the traditional histogram approaches. This is the reason why our approach outperforms the other histogram methods.

5.4.4. Impact of dataset size

In this experiment, we examine the impact of dataset size on performances of the four approaches. We use Zipf distribution data with parameter $z = 0.8$, and dimensionality $= 6$. The size of dataset changes from 10,000 to 200,000. As the dataset size increases, we increase the number of samples/buckets proportionally. Here, we set 5% as the ratio of the number of samples/buckets to dataset size. Thus, if the size of dataset is 100,000, we will assign 5000 samples space for the sampling method, and 2500 buckets for the three histogram approaches. The experimental results are illustrated in Fig. 9. From Fig. 9, we can see that as dataset size increases, the performances of all approaches get improved, which is due to the absolute increasing of the number of samples/buckets and the improvement of sampling/histogram accuracy. For six-dimensional data, the sampling method has the best performance.

5.4.5. Construction time and selectivity estimation time

Here we check the statistics construction time and the selectivity estimation time taken by different methods. A four-dimensional Zipfian dataset of size 500,000 is tested, the number of buckets is 1000, and 20,000 queries with QSIZE = 20%

Table 1
Time cost comparison (seconds).

Methods	Construction time	Estimation time
Sampling	30	6
MinSkew	480	30
GenHist	530	32
RAEH	510	26

are processed. For histogram-based approaches, the statistics construction time means the time spent on histogram construction; for the sampling approach, it is the time cost in the sampling process. Table 1 shows the results. The selectivity estimation time is the sum of time costs over 20,000 queries.

The sampling approach requires only one pass to scan the dataset for establishing the statistics, while the histogram approaches require several scans of the dataset. Therefore, a drawback of the histogram approaches compared to the sampling approach is that they need more construction time. As for the histogram approaches, GenHist cost more time than the others on constructing histogram.

In general, as far as time consuming is concerned, the sampling approach is the most efficient method; our approach is more efficient than GenHist.

6. Conclusion

In this paper, we investigate histogram construction for selectivity estimation based on a new criterion: aggregate error. We develop the histogram construction algorithms based on relative aggregate error minimization. We demonstrate the effectiveness and efficiency of the algorithm by extensive experiments over synthetic and real-world datasets. Compared with the sampling approach and two histogram-based approaches, our relative aggregate error based histogram construction algorithm (RAEH) performs best for low-dimensional data ($d \leq 4$); and for high-dimensional data, our RAEH algorithm is comparable to the state-of-the-art histogram-based approach.

References

[1] V. Markl, N. Megiddo, M. Kutsch, T. Minh Tran, P.J. Haas, U. Srivastava, Consistently estimating the selectivity of conjuncts of predicates, in: Proc. of VLDB'05, 2005, pp. 373–384.
[2] S. Guha, K. Shim, J. Woo, REHIST: Relative Error Histogram Construction Algorithms, in: Proc. of VLDB'04, 2004, pp. 300–311.
[3] P. Gibbons, Y. Mattias, V. Poosala, Fast incremental maintenance of approximate histograms, in: Proc. of VLDB'97, 1997, pp. 466–475.
[4] S. Guha, N. Koudas, Approximating a data stream for querying and estimation: Algorithms and performance evaluation, in: Proc. of ICDE'02, 2002, pp. 567–579.
[5] S. Guha, N. Koudas, K. Shim, Data streams and histograms, in: Proc. of STOC'01, 2001, pp. 104–115.
[6] H.V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K.C. Sevcik, T. Suel, Optimal histograms with quality guarantees, in: Proc. of VLDB'98, 1998, pp. 275–286.
[7] Y. Mattias, J.S. Vitter, M. Wang, Dynamic maintenance of wavelet-based histograms, in: Proc. of VLDB'00, 2000, pp. 101–111.
[8] V. Poosala, Y. Ioannidis, P. Haas, E. Shekita, Improved histograms for selectivity estimation of range predicates, in: Proc. of SIGMOD'96, 1996, pp. 294–305.
[9] N. Bruno, L. Gravano, S. Chandhuri, STHoles: A workload aware multidimensional histogram, in: Proc. of SIGMOD'01, 2001, pp. 211–222.
[10] D. Gunopulos, G. Kollios, V. Tsotras, C. Domeniconi, Approximating multi-dimensional aggregate range queries over real attributes, in: Proc. of SIGMOD'00, 2000, pp. 463–474.
[11] J. Lee, D. Kim, C. Chung, Multidimensional selectivity estimation using compressed histogram information, in: Proc. of SIGMOD'99, 1999, pp. 205–214.
[12] V. Poosala, Y. Ioannidis, Selectivity estimation without the attribute value independence assumption, in: Proc. of VLDB'97, 1997, pp. 486–495.
[13] J. Vitter, M. Wang, Approximate computation of multidimensional aggregates on sparse data using wavelets, in: Proc. of SIGMOD'99, 1999, pp. 193–204.
[14] Y. Wu, D. Agrawal, A.E. Abbadi, Applying the golden rule of sampling for selectivity estimation, in: Proc. of SIGMOD'01, 2001, pp. 449–460.
[15] F. Olken, D. Rotem, Random sampling from database files: A survey, in: Proc. of SSDBM'90, 1990, pp. 92–111.
[16] F. Olken, Random sampling from databases, Ph.D. Thesis, University of California, April 1993.
[17] Y. Matias, J. Scott Vitter, M. Wang, Wavelet-based histograms for selectivity estimation, in: Proc. of SIGMOD'98, 1998, pp. 448–459.
[18] S. Acharya, V. Poosala, S. Ramaswamy, Selectivity estimation in spatial databases, in: Proc. of SIGMOD'99, 1999, pp. 13–24.
[19] S. Muthukrishnan, V. Poosala, T. Suel, On rectangular partitioning in two dimensions: Algorithms, complexity, and applications, in: Proc. of ICDT'99, 1999, pp. 236–256.
[20] D. Gunopoulos, G. Kollios, Vassilis Tsotras, Carlotta Domeniconi, Selectivity estimators for multi-dimensional range queries over real attributes, VLDB Journal 14 (2) (2005) 137–154.
[21] G. Piatetsky-Shapiro, C. Connell, Accurate estimation of the number of tuples satisfying a condition, in: Proc. of SIGMOD'84, 1984, pp. 256–276.
[22] A. Aboulnaga, S. Chaudhuri, Self-tuning histograms: Building histograms without looking at data, in: Proc. of SIGMOD'99, 1999, pp. 181–192.