Error Minimization for Approximate Computation of Range Aggregates

Xuemin Lin and Qing Zhang
School of Computer Science and Engineering
University of New South Wales
Sydney, NSW 2052, Australia. {lxue,qzhang}@cse.unsw.edu.au

Abstract

Histogram techniques are widely used in commercial database management systems for an estimation of query results. Recently, they have been also used in approximately processing database queries, especially aggregation queries. Existing research results in this area have been mainly focused on constructing a histogram to approximately represent, as accurate as possible on an intuitive base, the original data frequencies. In this paper, we will propose a novel histogram construction method aiming to minimize the average approximate aggregation errors; and we have developed an efficient algorithm to construct near optimal histograms to achieve this goal. Our experiment results showed that the new histogram construction techniques lead to more accurate results than those by existing histogram techniques, and also out-perform the existing wavelet techniques.

1 Introduction

Traditional developments in query processing optimization techniques have focused on generating exact answers in a way that seeks to minimize response time and maximize throughput. However, in many recent database applications it may be too expensive to produce exact answers. For example, getting the exact answer for a complex query issued to a data warehousing may take hours or even days. Sometimes a network or disk storage failure may cause a part of data not accessible; this makes exact answers impossible. Further, in a decision support system an early response by approximate answers is quite helpful because the user can quickly determine a direction to drill down the data. Clearly, approximation is a good alternative in those applications. The quality of an approximate processing is measured by two conflicting parameters: efficiency and accuracy. A "good" approximate query processing usually means a good trade-off between efficiency and accuracy.

Database queries have two forms - *aggregation* and *non-aggregation*. An aggregate returns a numeric value; for instance, COUNT, SUM, AVG, etc. A non-aggregation query returns a set of tuples from a database tables; for example, JOIN.

Aggregates are not only an important class of queries in conventional database system applications but also serve as an important base for *quantitative* data mining problems [10] in modern applications, such as financial market analysis and telecommunications. Approximate processing of aggregates, thus, has recently attracted a great deal of attention. Most research results are based on a *data size reduction* paradigm. Four techniques [6] have been developed, such as *sampling*, *sketches*, *histogram*, and *wavelet*. Sampling [1, 7] is a popular data size reduction technique which takes a small portion of data as representative. To reduce approximation errors caused by applying sampling techniques to data with a big distribution skew, wavelet [9] techniques were firstly adopted by the authors in [17] to approximate query processing. The basic idea is to compress data by *important* wavelet *coefficients* [9]. The sketch techniques [2, 5] have been developed for an estimation of join processing results. The fourth technique is based on histograms, which were originally used in commercial database management systems to capture the distribution statistics of attribute values in query optimizers.

Among these four techniques, histogram is the most popular data reduction technique for approximately processing range aggregates; this is partially because that the histogram technique naturally suits for estimating aggregates. The basic idea of a histogram technique is to partition original data into certain number of "intervals" ("buckets"). The key issues in histogram techniques are: 1) how to partition the original data, and 2) how to approximate the original data in each bucket. The goal is to approximately process aggregates as accurate as possible. Many histogram techniques have been recently developed [8, 13, 15, 14, 18]. In this paper, we will focus on an investigation of more accurate histogram techniques.

The current research in the area may be classified into two categories: 1) new ways to represent each bucket [8, 15, 22] and 2) new ways (optimization goals) [8, 4] to partition data into buckets. In [8, 15, 22], "linear-spline" representation techniques have been developed to represent each bucket in contrast to "conventional" representation techniques [13, 14, 18]. In [4], an "encoding" scheme has been developed as a post-process to further partition data in each bucket following a data partitioning. Most existing techniques in partitioning data have been focused on minimizing approximation errors based on various different intuitive optimal models [13, 14, 18]. The paper [8] presents the first work on a formal investigation of the problem of error minimization in range aggregates with respect to a "uniform" query pattern. With the applications to time series data, the techniques in [8] take the assumption that all the "values" are evenly distributed over a data set while the "frequencies" have an arbitrary distribution. The techniques in [8] are specially developed for such an environment; and they are not quite applicable to the applications where values are not evenly distributed.

In this paper, we present a framework for partitioning data, with an arbitrary value distribution, into buckets so that the average approximate errors in processing aggregates may be minimized with respect to a uniform query pattern. Note that the techniques in [8] is not applicable to our framework, and the corresponding optimization problem tends to be computational intractable. In this paper, we will first concentrate on a mathematic analysis of the dominant parts in our optimization model to obtain a simplified model, which is quite close to the original optimization model. Then we present an efficient algorithm to deliver the optimal solution to the simplified model to serve as a near optimal solution to our original optimization model. These are the main contributions of the paper. Our experiment results suggest that the new histogram construction technique in this paper may greatly improve the accuracies of the existing histogram techniques, and also out-perform the existing wavelet techniques. Our experiment shows that the minimum improvement of accuracy by our new techniques over the existing ones is about 50%.

The rest of this paper is organized as follows. Section 2 presents the relevant definitions, an overview on the existing histogram techniques, and a motivation of our research in the paper. Section 3 presents the details of our new data partitioning model, as well as the data partitioning algorithm. Section 4 reports our experiment results. This is followed by a conclusion and remarks.

2 Background

Given a relation R and an attribute X of R, the *domain* D of X is the set of all possible values of X, and a finite set V $(\subseteq D)$ denotes the distinct values of X in an *instance* r of R. Let V be ordered; that is $V = \{v_i : 1 \leq i \leq n\}$ where $v_i < v_j$ if $i < j$. The instance r of R restricted to X is denoted by T, and can be represented as follows.

$$T = \{(v_1, f_1), (v_2, f_2), \ldots, (v_n, f_n)\}.$$

In T, each v_i is distinct and is called a *value* of T; and f_i is the occurrence of v_i in T and is called the *frequency* of v_i. Note that in this paper T may be called *data distribution* or *data set* alternately; that is, "data set" and "data distribution" will be used as synonym in this paper.

A *histogram* on data set T is constructed by the following two steps.

Step 1: partitioning the values of T into β ($\geq$1) disjoint intervals (called *buckets*) - $\{B_i : 1 \leq i \leq \beta\}$, such that each value in B_i is smaller than that in B_j if $i < j$.

Step 2: approximately representing the frequencies and values in each bucket.

The *width* of a bucket is $v_j - v_i$ where v_j and v_i are respectively the maximal value and minimum value in the bucket. The *spread* s_i of v_i (for $1 \leq i \leq n-1$) is defined as $v_{i+1} - v_i$. The *area* a_i of v_i is defined as $f_i * s_i$ for $1 \leq i \leq n$ where we make $s_n = 1$.

2.1 Existing Histogram Techniques

In a histogram, each bucket stores an approximate representation of the original data distribution in the bucket; this involves two approximate representations in a bucket: an approximation of the value distribution and an approximation of the frequency distribution. To minimize the storage space of the information to be stored in a bucket, most existing data partitioning algorithms aim to partition the data such that the data distribution in each bucket is as close to a uniform distribution as possible. Consequently, the value distribution in a bucket may be approximately represented by the *uniform spread assumption* (that is, the values are assumed to evenly span the bucket [18]), while the frequency distribution may be approximately represented by a constant [6] - the average frequency in the bucket.

With the uniform spread assumption, to represent a value distribution in a bucket we only need to store the minimum and maximum values in the bucket together with the number of distinct values in this bucket; the other values can be approximately derived according to this assumption. Figure 1(a) shows the information to be stored for each bucket.

The existing histogram techniques for generating buckets, based on the above histogram representation, may be summarized below according to different data partitioning goals:

- *Equi-width* [14]: bucket widths equal each other.
- *Equi-sum* [19, 14]: the sum of the frequencies in each bucket is the same.
- *Maxdiff* [18]: the data distribution is partitioned such that the differences of the frequencies between adjacent boundaries are maximized.
- *V-optimal* [12, 15, 22]: Partition data such that $\sum_{j=1}^{\beta} \sum_{k=1}^{n_j} (F_{j,k'} - F_{j,k})^2$ is minimized, where β is

the number of buckets, n_j is the number of distinct values in the j'th bucket. $F_{j,k}$ is the original frequency (or area)of k'th distinct value in the j'th bucket, and $F_{j,k'}$ is the corresponding approximate frequency (or area).

Experiment results [22] suggest that in most applications, Maxdiff and V-optimal out-perform Equi-width and Equi-sum [18]. V-optimal usually leads to more accurate approximate aggregation results than Maxdiff [12] does.

Observing the limitation of the frequency approximation in a bucket, [15, 22] propose to use a straight line $q * v + c$ to represent the frequency distribution in a bucket; this is a generalization of the conventional representation (the representation above) where $q = 0$ and c is set to the average frequency. As pointed in [22], the technique in [15] can guarantee the exact frequency summation and the exact summation of values multiplied by the corresponding frequencies in each bucket when values evenly span the bucket. However, such a property cannot be guaranteed by the technique in [15] if values are not evenly distributed in a bucket, although it is very important in processing an aggregate over a large range. To resolve this, we proposed [22] to fix q and c in a bucket by imposing the two restrictions - the fixed frequency summation and the fixed summation of values multiplied by the corresponding frequencies in the bucket. Both papers [15, 22] proposed to use the V-optimal model afterwards to obtain a data partitioning. Note that in [15, 22], the information required for a bucket is illustrated in Figure 1(b). Gilbert *et al.* [8] proposed to investigate the average error minimization for a *uniform* distributed query pattern; that is each range takes the same probability. In their investigation, they assume the whole value set is evenly distributed; and thus the original value set can be exactly retrieved from a histogram. The paper [4] goes a step further by proposing a post-processing technique once data are partitioned; and thus it may also be applicable, as a post-process, to our techniques.

Starting Value	End Value	Number of Values	Average Frequency

(a): conventional representation

Starting Value	End Value	Number of Values	q	c

(b): linear spline representation

Figure 1. Two representations

2.2 Motivation

The current research focus in this area has mainly addressed the issue of an "optimal" generation of a data partition, such that the frequency distribution in each bucket may be approximately represented by a line as accurate as possible. It has been focused on the minimization of *variance* [16] between the actual frequencies and approximate frequencies.

Note that the value distribution may also have a great impact on an approximation query processing result. This has been addressed in [18], where an *area* variance was proposed to replace a pure frequency variance. The area variance has been extended to cover linear-spline based histograms in [22]. It has been shown [18, 22] that the area variance may greatly improve the accuracy comparing with a pure frequency variance. However, the minimization of variances does not necessarily guarantee the minimization of approximation errors.

The average error minimization techniques in [8] focused on the situation where the values involved is evenly distributed. However, it is not quite applicable to a more general situation when values involved takes an arbitrary distribution unless we add all possible values such that many (probably most) values take the zero frequency. Clearly, adding many uninvolved values will greatly increased data partitioning costs. Further, the proposed bucket presentations in [8] are not as accurate as that in [15, 22]; this will be confirmed by our experiments.

In the next section, we will present our results on minimization of average approximation errors by using the bucket representation technique in [22]. Our algorithms are based only on the values involved, which take an arbitrary distribution.

3 Minimization of Average Aggregate Errors

In this section, we will present a new histogram technique. The technique is based on the minimization of average errors.

Suppose that a data set (data distribution) $T = \{(v_1, f_1), (v_2, f_2), \dots, (v_n, f_n)\}$ is given, which is partitioned into β disjoint buckets $\{B_i : 1 \leq i \leq \beta\}$, where the partition is denoted by B. Against a histogram built on the data partition B, an aggregate over a range $[x, y]$ will be processed by the following two steps. First is allocating x and y in the partition B. Second is using the corresponding buckets to answer the query.

Note that while processing aggregates over a range $[x, y]$ against a data set T, we need to find all the distinct values in $[x, y]$ to answer the queries. However, it is impossible to find exact distinct values in $[x, y]$ in a histogram of T due to an approximate representation of the value distribution in each bucket unless values are evenly distributed. As mentioned earlier, in a bucket of a histogram the i'th distinct value may be approximately obtained by a uniform-spread model [11]. In this paper, we will use this uniform-spread model.

Given a partition B, a histogram H on B, and a range $[x, y]$, the error generated by an approximate query over the range against H is denoted by $e_B^H(x, y)$. Note that in this paper, we will always choose a specific linear-spline form from [22] (to be described in the following subsection); this implies that once a data partition is given, H is unique. Consequently, $e_{T,B}^H(x, y)$ may be abbreviated to $e_{T,B}(x, y)$. There are several ways to specify an error [11].

Without loss of generality, in this paper we use an absolute error function; that is,

$$e_{T,B}(x,y) = |A_T(x,y) - A'_{T,B}(x,y)|. \quad (1)$$

Here, $A_T(x,y)$ represent the exact answer to a range aggregate over $[x,y]$, and $A'_{T,B}(x,y)$ is an approximate answer. Note that there are many types of aggregates. In this paper, we will use COUNT aggregate in our study for generating a data partition.

Suppose that the probability distribution for ranges $[x,y]$ is a continuous distribution over a data set T, which can be represented by a *probability density function* $P(x,y)$ [16] where $\int_{v_1}^{v_n}\int_{x}^{v_n} P(x,y)dxdy = 1$. The average error for range aggregations (COUNT) over T against a data partition B is:

$$E_{T,B} = \int_{v_1}^{v_n}(\int_{x}^{v_n} P(x,y) * e_{T,B}(x,y)dy)dx \quad (2)$$

Our goal is to find a data partition B such that (2) will be minimized. As mentioned earlier, in this paper we will investigate the minimization of (2) with respect to a uniform query distribution; that is $P(x,y) = \frac{2}{(v_n - v_1)^2}$. Therefore, (2) can be represented as follows.

$$E_{T,B} = \frac{2}{(v_n - v_1)^2}\int_{v_1}^{v_n}(\int_{x}^{v_n} e_{T,B}(x,y)dy)dx \quad (3)$$

The problem of finding a data partition B to minimize the goal function tends to be computation intractable. Below we first present a reasonably tight upper-bound on the goal function in (3); this will be done based on a mathematic analysis of the goal function. Then we will adopt the dynamic programming technique developed in [3, 12] to generate an optimal data partition against the upper-bound.

The rest of the section will be organized as follows. We first describe the linear-spline model used in our study. Then we will provide an upper-bound for (3). This will be followed by an efficient optimal algorithm against the upper-bound.

3.1 A Linear-spline model

In [22], we proposed to use a linear-spline $qv + c$ to represent the frequency in a bucket where q and c are fixed by imposing the following constraint:

> q and c should be chosen to make the COUNT and SUM over the bucket by $qv + c$ are, respectively, equal to those over the bucket by using the original data set.

Note that in a bucket $[a_i, b_i]$ with k distinct values, the k distinct values in a histogram restricted to the the bucket $[a_i, b_i]$ may be approximately retrieved by the following formula (4) in the uniform-spread assumption.

$$v'_l = a_i + l * \frac{b_i - a_i}{k-1} \text{ (for } 0 \le i \le k-1). \quad (4)$$

It has been shown [22] that such a linear-spline model out-performs the linear-spline model [15] when the original spreads in a given data set are not the same. However, these two linear-spline models are equivalent when the original spreads are the same (uniform).

In this paper, we will use the linear spline model in [22] to model a histogram.

3.2 Upper-bounds

In this subsection, we work out good up-bounds for (3). The deduction is quite mathematics involved.

Suppose that $B = \{B_i = [a_i, b_i] : 1 \le i \le \beta\}$ is a disjoint partition, with β buckets, of the distinct values of T, where $a_1 = v_1$ and $b_\beta = v_n$. Suppose that C_{B_i} denotes the COUNT value over B_i against T. Clearly, the COUNT value over B_i against the histogram using the linear-spline model in [22] is also C_{B_i}. By this property and elementary calculus, we can derive the following upper-bound for (3). Due to the space limitation, the interested readers may refer to our full paper on the authors web page for a detailed proof.

Theorem 1

$$\begin{aligned} E_{T,B} \quad \le \quad & \frac{2}{(v_n - v_1)^2}\sum_{i=1}^{\beta}(\int_{a_i}^{b_i}\int_{x}^{b_i} e_{T,B}(x,y)dydx \quad (5) \\ & +(v_n - b_i)\int_{a_i}^{b_i} e_{T,B}(x,b_i)dx \\ & +(a_i - v_1)\int_{a_i}^{b_i} e_{T,B}(a_i,x)dx) \end{aligned}$$

(Note that $a_1 = v_1$ and $b_\beta = v_n$.)

In the upper-bound given by the right side of (5), there are three parts. The theorem below shows that the second and third parts may be represented by a similar form; see the full paper for a detailed proof.

Theorem 2 $\forall B_i \in B$, $\int_{a_i}^{b_i} e_{T,B}(x,b_i)dx = \int_{a_i}^{b_i} e_{T,B}(a_i,x)dx$.

From Theorem 2, it follows that the right side of (5) may be re-written as:

$$\begin{aligned} & \frac{2}{(v_n - v_1)^2}\sum_{i=1}^{\beta}(\int_{a_i}^{b_i}\int_{x}^{b_i} e_{T,B}(x,y)dydx + \quad (6) \\ & (v_n - v_1 - (b_i - a_i))\int_{a_i}^{b_i} e_{T,B}(a_i,x)dx) \end{aligned}$$

The calculation of the first part in (6) may have to be carried out in a quadratic time. By elementary calculus, we can show that the first part is usually not a dominant factor; it is stated as the following theorem.

Theorem 3 $\forall B_i \in B$, $\int_{a_i}^{b_i}\int_{x}^{b_i} e_{T,B}(x,y)dydx \le (b_i - a_i)\int_{a_i}^{b_i} e_{T,B}(a_i,x)dx$.

From Theorem 1, (6), Theorem 2, and Theorem 3, the following is immediate:

$$E_{T,B} \leq \frac{2}{v_n - v_1} \sum_{i=1}^{\beta} (\int_{a_i}^{b_i} e_{T,B}(a_i, x)dx) \quad (7)$$

It should be clear, $v_n - v_1 - (b_i - a_i)$ is usually larger than $b_i - a_i$ (for $1 \leq i \leq n$) unless there is a huge bucket. In fact, in our experiment we are always able to obtain the same data partitioning result for the goal functions in (7) and (6) respectively. However, the upper-bound in (7) leads to a much faster algorithm than the upper bound in (6) does. Below we present our algorithm to produce a data partition B with β buckets to minimize (7).

3.3 Data Partition Algorithm

In this subsection, we will present an efficient algorithm to partition a data set $T = \{(v_1, f_1), (v_2, f_2), \ldots, (v_n, f_n)\}$ such that the upper-bound in (7) is minimized. Since v_1 and v_n are two constants for a given T, we need only to find a data partition B with β buckets such that the following goal function is minimized.

$$\sum_{i=1}^{\beta} (\int_{a_i}^{b_i} e_{T,B}(a_i, x)dx) \quad (8)$$

We first show a linear algorithm to calculate an integration $\int_{a_i}^{b_i} e_{T,B}(a_i, x)dx$.

3.3.1 Integration Calculation

Our algorithm for calculating the integration $\int_{a_i}^{b_i} e_{T,B}(a_i, x)dx$ for a bucket B_i is based on a *merge* paradigm in external merge sort algorithm [20]. It runs in linear time with respect to the number of distinct values in $[a_i, b_i]$.

Suppose that $B_i = [a_i, b_i]$ contains k distinct values in T, that is, B_i contains $\{(v_{j_i}, f_{j_i}), (v_{j_i+1}, f_{j_i+1}), \ldots, (v_{j_i+k-1}, f_{j_i+k-1})\}$, and $v_{j_i} = a_i$ and $v_{j_i+k-1} = b_i$. In a histogram representation of B, the k distinct values are approximately represented, by a uniform-spread assumption, as the following:

$$v'_{j_i+l} = a_i + l * \frac{b_i - a_i}{k-1} \quad (0 \leq l \leq k-1).$$

The k corresponding frequencies are represented as:

$$f'_{j_i+l} = q * v'_{j_i+l} + c.$$

In general, the relationship between $V_i = \{v_{j_i+l} : 0 \leq l \leq k-1\}$ and $V'_i = \{v'_{j_i+l} : 0 \leq l \leq k-1\}$ may be arbitrary. This implies that the combination of V_i and V'_i may divide $[a_i, b_i]$ into at most $2k-3$ subintervals (noting $a_i = v_{j_i} = v'_{j_i}$ and $b_i = v_{j_i+k-1} = v'_{j_i+k-1}$). However, in each subinterval, $e_{T,B}(a_i, x)$ is a constant, and can be determined by the values in V_i and the values in V'_i before the subinterval. More specifically, given an x ($v_{j_i+p} \leq x < v_{i_{j_i+p+1}}, v'_{j_i+m} \leq x < v'_{j_i+m+1}$), $e_{T,B}(a_i, x)$ is equal to:

$$e_{T,B}(a_i, x) = |\sum_{l=0}^{p} f_{j_i+l} - \sum_{l=0}^{m} f'_{j_i+l}| \quad (9)$$

Clearly, once V_i and V'_i are merged together to form a partition on $[a_i, b_i]$, we need only to run a linear scan over the partition to calculate the integration. For instance, we have a bucket which holds three value-frequency points $\{(0, 4), (1, 5), (4, 9)\}$. In the histogram, we approximately reconstruct the bucket by $\{(0, 4.75), (2, 6), (4, 7.25)\}$. The combination of them form 3 subintervals: $[0, 1)$, $[1, 2)$, and $[2, 4)$. Then:

$$\begin{aligned}&\int_0^4 e(0, x)dx \\ &= |4.75 - 4| * (1 - 0) + |4.75 - 4 - 5| * (2 - 1) \\ &+ |6 + 4.75 - 4 - 5| * (4 - 2) \\ &= 8.5.\end{aligned}$$

To calculate the integration, the critical part is to merge V_i and V'_i together according to their ordering. However, as V_i and V'_i is already ordered, this can be easily done by the merge paradigm in the external merge sort [20], and runs in a linear time with respect to k. The algorithm for calculating the integration consists of the following two steps.

Step 1: Sort merge V_i and V'_i together to form a disjoint partition on B_i.

Step 2: Scan the partition once to calculate the integration by taking the advantage that $e_{T,B}(a_i, x)$ is a constant in each partitioned subinterval.

As described above, both steps run in $O(k)$ time.

3.3.2 Dynamic Programming based Data Partition Algorithm

We now present an efficient algorithm to partition a give data set $T = \{(v_1, f_1), (v_2, f_2), ..., (v_n, f_n)\}$ with at most β buckets such that (8) is minimized. The algorithm follows the dynamic programming technique developed in [3, 12]. We briefly described it below.

Let $M^*(X, Y)$ represent the optimal result for using at most Y buckets to partition the first X values of T. Let $M[a, b]$ denote the bucket containing the consecutive $\{(v_a, f_a), ..., (v_b, f_b)\}$ in T. Below is the crucial formula.

$$M^*(n, \beta) = \min_{1 \leq j \leq n-1} \{M^*(j, \beta - 1) + M[j + 1, n]\}$$

Thus, in order to calculate $M^*(n, \beta)$, we must calculate $M^*(i, k)$ for $1 \leq i \leq n$ and $1 \leq k < \beta$. Applying the dynamic programming based algorithm from [12] to our problem, there will be $O(\beta n^2)$ iterations to do the computation of $M^*(i, k) + M[i + 1, l]$ where $l \leq n$; and a computation of $M^*(i, k) + M[i + 1, l]$ may be done in constant time (i.e., the computation of $M[i+1, l]$) in [12] if a linear pre-process is performed.

Note that in our problem, to make the computation of each $M^*(i, k) + M[i + 1, l]$ be constant time we may have to pre-compute $M[a, b]$ for every possible sub-interval in T.

Further, as shown in the last sub-sub-section, the computation of $M[a,b]$ takes linear time. Therefore, the pre-process runs in $O(n^3)$ time. This implies that the data partitioning algorithm will run in $O(n^3 + \beta n^2) = O(n^3)$ time. It occupies $O(n^2)$ space mainly due to the pre-process. On the other hand, if each $M[a,b]$ is not pre-computed and stored, then the algorithm may be implemented in $O(\beta n^3)$ with $O(\beta n)$ space requirement. So there is always a trade-off between space and efficiency.

It is worth to note that if we use the goal function in (6) to replace the goal function in (8), then the dynamic programming paradigm above also works. However, the complexities will be increased to $O(n^4)$ time with $O(n^2)$ space and $O(\beta n^4)$ with $O(\beta n)$ space, respectively. This increment is due to the calculation of each $\int_{a_i}^{b_i} \int_x^{b_i} e_{T,B}(x,y)dydx$, which can be done in a similar way to the paradigm in the last subsection but runs a quadratic time.

3.4 Different Error Functions

We may also replace the absolute value based error function by other error functions (for instance, variance based error function) without need of a change of techniques developed in this paper. We do not list all of them for the space reason.

4 Experiment Results

The data sets used in our experiments are synthesized *zipf* [21] data, which is the most popular benchmark to evaluate the histogram techniques. Two databases have been generated; each database consists of 10 data sets.

- The database 1 is used to evaluate the performance of histogram techniques in the environment where all distinct values are evenly distributed over a data set with $100,000$ records. Each data set in the database 1 uses the integers in $[1, 1000]$ as the value set.
- A data set in the database 2 contains $100,000$ records with 1001 different values from the domain $[0, 100000]$.

Note a data set from the database 2 has its values not evenly distributed but zip-distributed. The generation of each data set follows the three steps below.

- *Generating Frequencies:* Different frequencies are generated according to *zipf law* and the *zipf* parameter $z = 0.5$. This means a medium skew frequency distribution.
- *Generating Values:* The spreads of values follow one distribution of *zipf_inc*, *zipf_dec*, *cusp_min*, *cusp_max* and *zipf_ran* [18]. The Zipf parameter $z = 1.0$. (This is not used in database 1.)
- *Generating Data Distribution:* Frequencies are randomly assigned to different values.

Note that the number of buckets to be used in a histogram controls a data reduction "degree". If the number of buckets equals the number of distinct values, then there is no data reduction, and consequently all histograms will produce the same result - the exact result.

In our experiment, we compared the performance of our data partition algorithm - *MINHERR*, as presented in Section 3 for minimizing (8), with the following 6 algorithms.

- V-optimal(area) [18, 12]: We use average area instead of average frequency as a parameter for optimization, because this gives the best performance [22] for V-optimal model.
- BSW: the linear wavelet technique [17] based on Biorthogonal Spline Wavelet. We use *MATLAB's bior2.4* to construct BSW, which has a better performance than Haar wavelets.
- LSLS (area) [15]: a Linear Spline of representation of a bucket, in combining with the Least Square method to fix the two parameters in a line. Note that the data partitioning algorithm for LSLS follows a similar objective to that in V-optimal method; the difference is that in LSLS, line is used instead of average frequency. Further, in our experiments we also use the area as parameter for LSLS; this is because it gives a better performance [22].
- LSCSG(area) [22]: a linear spline histogram technique. It is the same as LSLS (area) except that the two parameters are not fixed by the least square methods.
- SAP0: the error minimization algorithm from [8].
- SAP1: another error minimization algorithm from [8]. The difference between SAP0 and SAP1 is that SAP0 uses average frequency summation in a bucket, while SAP1 uses two lines in a bucket.

In our experiments, we recorded only the accuracies of these algorithms for constructing histograms for approximate aggregation. Note that these algorithm all run in a similar lower order polynomial time. In the applications where updates are not frequent, a histogram construction algorithm does not have to run very often; and thus, the accuracy of these algorithm will be the key for approximate range queries.

Note that in our experiments, we have enforced that every data reduction technique occupies same storage space, to be fair. For the database 1, we do not need to store information about the number of distinct values for the histograms and need only to store the minimum value for each bucket since we know that the values are evenly distributed. Therefore, we need only record 2 fields for each bucket in V-optimal, while respectively in LSLS (area), LSCSG (area), and MINHERR, 3 fields need to be recorded (see Figures 1(a) and 1(b) for reference). In BSW, for each remaining coefficient we need to record their position as well; and thus two fields are needed for storing one coefficient. Consequently, the number of wavelet coefficients stored in BSW is equal to the number of buckets in V-Optimal (area), while the numbers of buckets in LSCSG(area), LSLS(area) and MINHERR, respectively, are about 67% of that in V-optimal(area). Note that the numbers of buckets in SAP1 and SAP0, respectively, have been taken the same as that in V-optimal according to the suggestions from [8].

Note that for the database 2, it may not necessary to store the number of distinct values in a bucket if we calculate the number of records using a continuous model [4, 6]. However, this information is necessary for some other aggregations; for instance, the total summation of the record values in a range. Therefore, we keep this information in our histograms; see Figures 1(a) and 1(b) for storage requirements. For similar reasons to those in the last paragraph, the number of wavelet coefficients, now, should be twice of the number of buckets in V-optimal(area), while the numbers of buckets in LSCSG(area), LSLS(area) and MINHERR, respectively, are 80% of that in V-optimal(area). Again, the numbers of buckets in SAP1 and SAP0, respectively, are the same as that in V-optimal.

In our experiments, we use the bucket numbers $20, 25, 30, 35, 40, 45, 50$ in V-optimal(area) as the reference values, while the numbers in the other algorithms are adjusted according to the above ratios.

In our experiments, we targeted the 3 most popular aggregates - COUNT, SUM, and AVG. Since AVG is derived from a division between SUM and COUNT, we focused only on two types of range queries, COUNT and SUM. For each data set and each type of range aggregates, 1000 queries are randomly generated with the form: $\{x \leq values \leq y \mid x < y\}$ Here x and y are randomly selected from the value domains.

Let A_i denotes the actual result of a query q_i and A_i' denotes the approximately calculated result. The error metrics used to evaluate our histograms are:

- **absolute error:** $e_i^{abs} = |A_i - A_i'|$
- **average absolute error:** $\overline{e_N^{abs}} = \frac{\sum_{i=1}^{N} e_i^{abs}}{N}$ **,** where N represents the number of queries.
- **relative error:** $e_i^{rel} = \frac{|A_i - A_i'|}{A_i}$
- **average relative error:** $\overline{e_N^{rel}} = \frac{\sum_{i=1}^{N} e_i^{rel}}{N}$ **,** where N represents the number of queries.

All the generated data sets and histograms are stored in an *Oracle* DBMS and our experiments are done on a Pentium III 700 MHz CPU, 256MB memory computer with Linux 2.4.7.

Since the results do not vary significantly on different data sets within a database, we only show some typical results here; that is, one data set per database.

Bucket Number	Database 1: Absolute Errors of COUNT Query					
	V−opt	LSLS (LSCSG)	SAP0	SAP1	BSW	MINHERR
20	825.74	647.24	370.25	354.76	**34.76**	60.65
25	460.32	392.5	156.31	132.47	**33.25**	40.28
30	380.16	159.26	51.4	43.24	**32.59**	37.25
35	178.2	87.46	42.68	41.28	32.07	**31.16**
40	170.47	54.9	40.32	40.25	31.82	**20.7**
45	48.92	48.25	37.18	34	31.06	**15.42**
50	39.26	31.66	28.95	25.3	30.75	**14.18**

Figure 2. Approximate COUNT queries

Bucket Number	Database 1: Relative Errors of COUNT Query (%)					
	V−opt	LSLS (LSCSG)	SAP0	SAP1	BSW	MINHERR
20	67.47	30.93	15.62	13.78	13.74	**10.32**
25	42.58	18.27	11.04	11.06	13.01	**7.27**
30	17.47	11.5	8.36	7.45	12.76	**5.86**
35	12.58	10.72	7.15	6.18	11.54	**4.17**
40	11.59	8.36	6.54	6.03	11.25	**3.02**
45	7.8	7	5.28	4.21	11.21	**2.18**
50	5.16	4.25	4.16	3.17	11.16	**2.1**

Figure 3. Approximate COUNT queries

Bucket Number	Database 1: Relative Errors of SUM Query (%)					
	V−opt	LSLS (LSCSG)	SAP0	SAP1	BSW	MINHERR
20	68.21	31.45	16.03	14.22	13.82	**10.84**
25	43.15	18.76	11.46	11.27	13.25	**7.62**
30	18.32	11.94	8.58	7.96	13.01	**6.14**
35	13.14	10.83	7.42	6.84	12.71	**4.25**
40	12.58	9.05	7.18	6.12	12.07	**3.07**
45	8.25	7.47	5.7	4.58	12.01	**2.36**
50	6.13	4.82	4.54	3.39	11.94	**2.23**

Figure 4. Approximate SUM queries

Figures 2, 3, and 4 show our experiment results for the database 1. In our experiments, we focused on the the average relative errors since this performance index is more explanatorily and the space is limited. However, for the database 1 we also recorded the average absolute errors for COUNT since this index was evaluated in [8]. It is interested to notice that unlike the other methods, adding more coefficients in BSW does not improve very much the accuracy. Further, MINHERR outperforms SAP0 and SAP1 although SAP0 and SAP1 are the two optimal algorithms to achieve the minimum average errors. This is because SAP0 and SAP1 employed different bucket representations than that in MINHERR; and the algorithmic paradigms cannot be applied to our representation to provide the optimal solutions. Moreover, for the average absolute errors, BSW outperforms our MINHERR when the number of buckets is small. The MINHERR outperforms all the other techniques significantly for the average relative errors; the minimum improvement is about 50%. Figures 5 - 6 show the experiment results for the database 2 where Figure 5 records the average relative error for approximate COUNT and Figure 6 records the average relative error for approximate SUM. Clearly, the experiments results for the database 2 suggested that the performances of the 7 algorithms (including ours) follow a similar pattern to that for the database 1, except

Bucket Number	Database 2: Relative Errors of COUNT Query (%)						
	V−opt	LSLS	LSCSG	SAP0	SAP1	BSW	MINHERR
20	22.68	5.19	4.65	65.48	66.86	**2.54**	4.43
25	11.54	4.89	2.85	64.8	68.41	**2.47**	2.6
30	8.84	4.76	2.47	54.63	51.01	**2.33**	2.39
35	7.94	4.54	2.09	52.6	54.46	2.22	**1.83**
40	7.18	4.21	1.68	52.14	51.7	2.16	**1.66**
45	4.69	2.7	1.01	47.65	48.05	2.02	**0.97**
50	3.89	2.19	0.84	49.78	50.76	1.85	**0.81**

Figure 5. Approximate COUNT queries

Bucket Number	Database 2: Relative Errors of SUM Query (%)						
	V−opt	LSLS	LSCSG	SAP0	SAP1	BSW	MINHERR
20	22.95	4.91	4.66	61.12	57.37	**2.57**	4.49
25	11.59	4.76	3.56	56.84	61.02	**2.49**	2.75
30	8.83	4.6	2.39	49.27	50.18	2.34	**2.32**
35	7.97	4.43	2.1	49.82	50.4	2.23	**1.81**
40	7.21	4.15	1.69	44.62	44.54	2.19	**1.63**
45	4.54	2.57	**0.95**	42.35	46.03	2.05	0.96
50	3.76	2.13	0.8	46.61	44.15	1.87	**0.79**

Figure 6. Approximate SUM queries

that SAP0 and SAP1 are less competitive. This is because that SAP0 and SAP1 are specially designed for a evenly distributed value space.

In summary, our experiment results clearly suggested that our algorithm MINHERR lead to the best performance among these 7 algorithms. The minimum reduction for the average relative error, by using our MINHERR, is about 50% comparing with the other techniques; however, this excludes our earlier algorithm LSCSG for the database 2. For the database 2, MINHERR is just slightly better than LSCSG on average. This is because that in the database 2, the distribution of areas varies sharply and takes the dominant role in data partitioning algorithm; and thus the advantage of MINHERR over LSCSG fades away.

5 Conclusion and Remarks

In this paper, we presented a framework for minimizing the average approximate errors on aggregates. It tends to be computational infeasible to get an optimal solution generally even restricted to a uniform query pattern. We then concentrated on a mathematic analysis to obtain an upperbound for the average error, which is composed of the dominant parts in the average error. Consequently, we developed a near optimal efficient algorithm to solve the problem. According to our results, we will be able to choose the number of buckets to control an approximation accuracy in contrast to the existing results for general data distribution. The experiment results showed that our algorithm significantly out-performs the existing techniques.

As a future study, we will investigate the problem when query patterns are not necessarily uniform. we will also investigate the problem in a data stream environment.

References

[1] S. Acharya, P. B. Gibbons, and V. Poosala. *Congressional samples for approximate answering of group-by queries*, In SIGMOD Conference, pages 487-498, 2000.

[2] N. Alon, P. B. Gibbons, Y. Matias, and M. Szegedy, *Tracking Join and Self-Join Size in Limited Storage*, ACM PODS, 1999.

[3] R. E. Bellman. *The theory of dynamic programming.*, Bull. Amer Math Soc, 60:503-516, 1954.

[4] F. Buccafurri, L. Pontri, D.Rosci, and D. Sacca. *Improving range query estimation on histograms*, In 18th ICDE, 628-638, 2002.

[5] A. Dobra, M. Garofalakis, J. Gehrke, and R. Rastogi. *Processing Complex Aggregate Queries over Data Streams*, In SIGMOD Conference, 2002.

[6] M. Garofalakis and P. B. Gibons. *Approximate query processing: Taming the terabytes.*, VLDB, 2001.

[7] P. B. Gibbons and Y. Matias. *New Sampling-based summary statistics for improving approximate query answers.*, SIGMOD pages 331-342, 1998.

[8] A. Gilbert, Y. Kotidis, S. Muthukrishnan and M. Strauss. Optimal and Approximate Computation of Summary Statistics for Range Aggregates. *In Proceedings of PODS*, 227-236, 2001.

[9] A. Graps. *An introduction to wavelets.*, IEEE Computational Science and engineering, 2:50-61, Summer 1995.

[10] J. Han and M. Kamber, *Data Mining: Concepts and Techniques*, Morgan Kaufman, 2001.

[11] Y. E. Ioannidis and V. Poosala, *Histogram-Based Approximation Set-Valued Query Answers*, VLDB'99, 174-185, 1999.

[12] H. V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. C. Sevcik and T. Suel. *Optimal histograms with quality guarantees.* VLDB, 1998.

[13] H. V. Jagadish, H. Jin, B. C. Ooi and K. L. Tan. *Global optimization of histograms.*, SIGMOD, 2001.

[14] R. P. Kooi. *The optimization of queries in relational database.*, PhD thesis, Case Western Reserver University, Sep 1980.

[15] A. C. Knig and G. Weikum. *Combining histograms and parametric curve fitting for feedback-driven query result-size estimation.*, VLDB, 1999.

[16] D. D. Wackerly, W. Mendenhall and R. L. Scheaffer. *Mathematical Statistics with Application.* Duxbury Press, 1995.

[17] Y. Matias, J. S. vitter, and M. Wang. *Wavelet-based histograms for selectivity estimation.* SIGMOD pages 448-459, 1998.

[18] V. Poosala, P. J. Haas, Y. E. Ioannidis and E. J. Shekita. *Improved histograms for selectivity estimation of range predicates.* SIGMOD 1996

[19] G. Piatetsky-Shapiro and C. Connell. *Accurate estimation of the number of tuples satisfying a condition.* SIGMOD 1984.

[20] R. Ramakrishnan and J. Gethrke, *Database Management Systems*, McGraw Hill, 2000.

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

[22] Q. Zhang and X. Lin. *On Linear-Spline Based Histograms*, WAIM 2002, *LNCS Springer-Verlag*, 354-366, 2002.