Maintenance of Partial-Sum-Based Histograms

Kin-Fai Kan, David W. Cheung, Ben Kao
Department of Computer Science and Information Systems
University of Hong Kong
Hong Kong
{kfkan, dcheung, kao}@csis.hku.hk

Abstract

This paper introduces an efficient method for the maintenance of wavelet-based histograms built on partial sums. Wavelet-based histograms can be constructed from either raw data distributions or partial sums. The two construction methods have their own merits. Previous works have only focused on the maintenance of raw-data-based histograms. However, it is highly inefficient to apply directly their techniques to partial-sum-based histograms because a single data update would trigger changes of multiple partial sums, which in turn, would trigger large amounts of computation on the changes of the wavelet decomposition.

We present a novel technique to compute the effects of data updates on the wavelet decomposition of partial sums. Moreover, we point out some special features of the wavelet decomposition of partial sums and adapt a probabilistic counting technique for the maintenance of partial-sum-based histograms. Experimental results show that our maintenance method is efficient and its accuracy is robust to changing data distributions.

1. Introduction

Many DBMSs maintain histograms for selectivity estimation. There are a number of proposals in the literature on the construction of histograms including partition-based methods [9, 8] and transform-based methods [6, 5]. When the data of underlying relations are changed, the histograms will no longer reflect the current data distributions and significant estimation error could occur. In order to maintain the accuracy of selectivity estimation, it is important to bring histograms as up-to-date as possible. Several incremental methods have been proposed to maintain partition-based histograms [3, 1] and transform-based histograms [5, 7].

Wavelet-based histograms are first proposed by Matias et al. [6]. They possess several advantages. First, they require little storage cost and CPU cost at query optimization. Second, they offer more accurate selectivity estimation than traditional partition-based histograms. Third, unlike partition-based histograms, they can be extended naturally to multi-dimensional data. While wavelet-based histograms have several nice features, their maintenance is much more difficult than partition-based histograms because of the non-trivial mathematical transformation. In [6], two methods are suggested for the construction of wavelet-based histograms. One method is to perform wavelet decomposition on the raw data distribution while the other method is to perform wavelet decomposition on the extended cumulative data distribution (partial sums). The accuracy of the two methods varies with data distribution, query type, and the choice of an error measure [7]. One method performs better in some cases while the other method excels in others. Thus, both methods have their own merits.

Recently, Matias et al. [7] propose an efficient method for the dynamic maintenance of wavelet-based histograms. However, they only focus on the maintenance of histograms built on raw data distributions and do not deal with issues related to processing partial sums. In this paper, based on the method in [7], we develop a maintenance method for histograms built on partial sums. We present two techniques to figure out the effects of data updates on the wavelet decomposition of partial sums. The first technique is a direct extension of the results in [7]. The second technique is a novel one and it takes advantage of some nice features of the wavelet decomposition of partial sums. Moreover, we point out some special features of the wavelet decomposition of partial sums and adapt the probabilistic counting method in [7] for partial-sum-based histograms. Our techniques help complete the solution to the maintenance problem of wavelet-based histograms and make them a more attractive choice for future database optimizations.

The remaining of the paper is organized as follows. Section 2 introduces the basics of wavelet decomposition. Section 3 overviews the maintenance method proposed in [7]. Section 4 describes our techniques to maintain partial-sum-

based histograms incrementally. The experimental results are presented in Section 5. We conclude our study in Section 6.

2. Wavelet Decomposition

Wavelet decomposition is a mathematical tool for hierarchical decomposition of functions. As in [6, 7], we choose Haar wavelets as the wavelet functions. To illustrate how wavelets work, we start with a simple example. A detail treatment of wavelets can be found in [10].

Suppose we have a raw data distribution (frequency distribution) F of an attribute X of $N = 8$ values. We assume that the attribute X has domain $\{0, 1, \ldots, 7\}$.

$$F = [4, 8, 2, 0, 3, 3, 6, 8]\,.$$

The Haar wavelet decomposition of F can be computed as follows. We first average the data values pairwise to get a *lower-resolution* representation of the data with values

$$[7, 3, 1, 6]\,.$$

The average of the eighth and seventh values (8 and 6) is 7 and that of the sixth and fifth values (3 and 3) is 3, and so on. Some information is lost in this averaging process. To recover the values of the original data vector, we need to store the *detail coefficients* that capture the missing information. In Haar wavelets, detail coefficients are the pairwise differences of the original data values (divided by 2). For the eighth and seventh values (8 and 6), the detail coefficient is $(8-6)/2 = 1$. For the sixth and fifth values (3 and 3), the detail coefficient is $(3-3)/2 = 0$ and so on. It is easy to see that the original values can be recovered from the averages and the detail coefficients.

By repeating the above process recursively on the lower resolution array containing the averages, we get the full decomposition. Table 1 shows the results.

Resolution	Averages	Detail Coefficients
8	[8, 6, 3, 3, 0, 2, 8, 4]	
4	[7, 3, 1, 6]	[1, 0, -1, 2]
2	[5, 3.5]	[2, -2.5]
1	[4.25]	[0.75]

Table 1. The wavelet decomposition

We define the *wavelet decomposition* (also known as *wavelet transform*) of a data vector F to be the single coefficient representing the overall average of the data values in F together with the detail coefficients in increasing order of resolution. The individual entries are called the *wavelet coefficients*. Thus, the wavelet decomposition of F is given by

$$\widehat{F} = [4.25,\ 0.75,\ 2,\ -2.5,\ 1,\ 0,\ -1,\ 2]\,.$$

No information is lost in wavelet decomposition. Given the decomposition, we can reconstruct the original data vector by recursively adding and subtracting the detail coefficients from the next-lower resolution ones. For compression purposes, the detail coefficients are often normalized; the coefficients at the lower resolutions are weighted more heavily than the coefficients at the higher resolutions. After normalization, a large number of wavelet coefficients would become very small in magnitude. We can discard the smaller coefficients without introducing large errors in the reconstructed data vector. Thus, we can use the most significant coefficients to approximate the original data vector. For convenience in discussion, we give the formulae for the unnormalized wavelet coefficients in this paper, although in practice, the coefficients are normalized in an online manner.

The wavelet decomposition procedure can be represented by the *error tree* [6]. The value of the root is the overall average of the data values. The values of the internal nodes (excluding the root) are the detail coefficients of the wavelet decomposition. The values of the leaves are the original data values. The internal nodes and the leaves are labeled separately. The internal nodes are labeled in the breadth-first traversal order while the leaves are labeled from right to left. The construction of the error tree mirrors the wavelet decomposition procedure and shows clearly the relationship between the wavelet coefficients and the original data values. First, the original data values are assigned to the leaves from right to left. Then the wavelet coefficients are computed and assigned to the internal nodes. Hence, the value of an internal node is the difference between the sum of the values of the leaves in its left subtree and that in its right subtree (divided by the number of its descendant leaves). Figure 1 is the error tree for the above example.

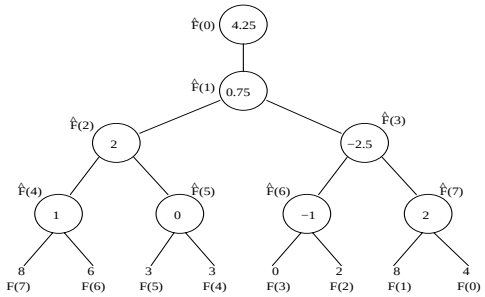


Figure 1. The error tree

We use $path\,(i)$ to denote the set of internal nodes along the path from leaf i to the root. We use $left\,(j)$ and

$right(j)$ to denote the left child and right child of internal node j. We use $leaves(j)$ to denote the set of leaves in the subtree rooted at internal node j. We use $L\_leaf(j)$ ($R\_leaf(j)$) to denote the index of the left most leaf (right most leaf) in the subtree rooted at internal node j. For any internal node j, its height in the tree is given by

$$height(j) = \begin{cases} \log_2 N - \lfloor \log_2 j \rfloor & \text{for } 1 \leq j < N; \\ \log_2 N & \text{for } j = 0, \end{cases} \quad (1)$$

where N is the length of the original data vector.

3. Maintenance of Raw-Data-Based Histograms

For the maintenance of wavelet-based histograms, Matias et al. [7] propose an efficient method based upon probabilistic counting. Their focus is on raw-data-based histograms, but some of their ideas are also useful for maintaining partial-sum-based histograms. In this section, we briefly discuss the main ideas of their maintenance method.

In [7], it is noted that the error tree has some nice properties that we can make use of to compute the effects of data updates on the wavelet decomposition of the raw data distribution.

Lemma 1 [7] *For any leaf i in the error tree, we consider the effect of its value change*

$$\Delta(i) = F_{t_1}(i) - F_{t_0}(i) \quad (2)$$

from time t_0 to time t_1. For each j where internal node $j \in path(i)$, we have

$$\widehat{F}_{t_1}(j) = \begin{cases} \widehat{F}_{t_0(j)} + \frac{\Delta(i)}{2^{height(j)}} & \text{if } i \in leaves(left(j)); \\ \widehat{F}_{t_0(j)} - \frac{\Delta(i)}{2^{height(j)}} & \text{if } i \in leaves(right(j)). \end{cases} \quad (3)$$

Lemma 1 can be extended to the multi-dimensional case. For details, please refer to [7]. Based on Lemma 1, Matias et al. [7] proposes using a probabilistic counting technique to maintain raw-data-based histograms incrementally.

First, wavelet decomposition is performed on the raw data distribution. Only the $m + m'$ most significant wavelet coefficients (in absolute value) would be stored, where $m+m' \ll N$. The top m coefficients would be stored in the histogram H for selectivity estimation. The other m' coefficients would be stored in the auxiliary histogram H' on disk for future maintenance. It is assumed that an activity log L is used to log all insert, delete, and modify activities; the maximal size of L is Max_Log_Size. Any wavelet coefficient $\widehat{F}(j)$ can be characterized into one of three types: (a) $\widehat{F}(j) \in H$, (b) $\widehat{F}(j) \in H'$ and (c) $\widehat{F}(j) \notin H \bigcup H'$.

When a new entry is added to L, all type (a) coefficients affected by the entry are updated according to Equation 3. When the number of entries in L reaches Max_Log_Size, we process the entries in L. For each entry in L, all type (b) coefficients affected by it are updated according to Equation 3. If an entry in L affects a type (c) coefficient $\widehat{F}(j)$, a probabilistic counting technique [2] is used: A coin with probability $p(j)$ of heads is flipped. If it is a head, the magnitude of $\widehat{F}(j)$ is set to v (a value to be explained soon) and $\widehat{F}(j)$ will replace the smallest coefficient (in absolute value) in H'. After processing all the entries in L, H and H' are adjusted. Whenever the magnitude of the largest coefficient in H' exceeds a threshold value H'_Thresh, it will replace the smallest coefficient in H. After adjusting H and H', online update of H and logging will be restarted.

The values of the four parameters required by the above method are set as follows. The value of Max_Log_Size depends on how often we would like to detect significant coefficients that do not appear significant initially (we call them *surprising coefficients*). Denote the magnitude of the minimum coefficient in H by $min(H)$. The value of H'_Thresh depends on how aggressive we are in adjusting H over time. Since there is no advantage to replace a coefficient with another of a close magnitude, it is set to $c_1 \times min(H)$ in [7], where c_1 is a constant (typically in the range $[1.0, 3.0]$). The parameter v is the estimated magnitude of surprising coefficients and, in [7], it is set to $c_2 \times min(H)$, where c_2 is a constant (typically in the range $[0.2, 0.8]$). The value of $p(j)$ should be set such that its inverse equals to the number of insertions (deletions) at the attribute value of the log entry that are required to bring the magnitude of $\widehat{F}(j)$ from zero to v. Thus, in [7], according to Equation 3,

$$p(j) = \frac{1}{v \times 2^{height(j)}}. \quad (4)$$

4. Maintenance of Partial-Sum-Based Histograms

In this section, we describe our method to maintain partial-sum-based histograms. First we focus on how to compute the effects of data updates on the wavelet decomposition of partial sums. Then we discuss how to adapt the probability counting technique for the maintenance of partial-sum-based histograms.

4.1. The Naive Update Technique

One simple way to compute the effects of data updates on the wavelet decomposition of partial sums is to use the results of [7]. First, we figure out how a data update affects the partial sums. Then we computes how the changes

of partial sums affects the wavelet decomposition of partial sums using Lemma 1. We call this simple technique the *naive update technique*.

The main drawback of the naive update technique is that its computational cost depends on the number of partial sums that are affected by a data update. In the worst case, all partial sums would be affected by a data update and we need to compute how the changes of all partial sums affect the wavelet decomposition of partial sums.

4.2. The Fast Update Technique

The naive update technique does not perform well when the domain size of the attribute is large. Can we find some interesting properties so that the effects of data updates on the wavelet decomposition of partial sums can be computed more efficiently? As illustrated below, the wavelet decomposition of partial sums has some nice properties that we can make use of to compute its exact changes efficiently. We call this technique the *fast update technique*.

We make a key observation: *even if a change of $F(i)$ affects the partial sums that would, in turn, affect $\widehat{S}(j)$, it does not necessarily affect $\widehat{S}(j)$ eventually, because the effects of different partial sums could be cancelled out.*

As mentioned in Section 2, the value of an internal node in the error tree represents the difference between the sum of the values of the leaves in its left subtree and that in its right subtree (divided by the number of its descendant leaves). Expressing a detail coefficient of partial sums in terms of the partial sums, we have

$$\begin{aligned}\widehat{S}(j) &= \frac{1}{2d}\cdot\{[S(r+d-1)+S(r+d-2)+\cdots \\ &\quad +S(r)]-[S(r-1)+S(r-2)+\cdots \\ &\quad +S(r-d)]\} \end{aligned} \tag{5}$$

where $2d$ is the number of descendant leaves of internal node j in the error tree and leaf r corresponds to the rightmost leaf in the left subtree of internal node j. Substituting the partial sums by the raw frequencies, we have

$$\begin{aligned}\widehat{S}(j) &= \frac{1}{2d}\cdot[F(r+d-1)+2\cdot F(r+d-2)+\cdots \\ &\quad +d\cdot F(r)+(d-1)\cdot F(r-1)+\cdots \\ &\quad +F(r-d+1)] \end{aligned} \tag{6}$$

We can observe that the indices of the terms in Equations 5 and 6 coincide (excluding the last term in Equation 5). This observation implies that a change of $F(i)$ affects wavelet coefficient $\widehat{S}(j)$ only if leaf i is one descendant leaf of internal node j in the error tree. In other words, a change of $F(i)$ can only affect those wavelet coefficients of partial sums corresponding to the internal nodes in $path(i)$ in the error tree. Thus, the number of wavelet coefficients of partial sums that are affected by a single data update is at most the logarithm of the domain size of the attribute (i.e., the height of the error tree) plus one but not the domain size of the attribute. Based on the above observation, we can easily figure out which wavelet coefficients of partial sums are affected a data update using the error tree.

The remaining problem is to compute the changes of those wavelet coefficients affected by the data update. Equation 6 shows that the raw frequencies are multiplied by some constants. As the constants are not all the same, changing different raw frequencies by the same amount may affect wavelet coefficient $\widehat{S}(j)$ differently. From Equation 6, we observe that the constants follow an interesting pattern (starting with 1 from the left-most raw frequency $F(r+d-1)$; increasing by one at a time until $F(r)$; then decreasing by one at a time; ending with 1 for the rightmost raw frequency $F(r-d+1)$). Hence, we can derive the changes of the wavelet coefficients affected by data updates easily.

Now we look at how the overall average $\widehat{S}(0)$ would be affected by data updates. The overall average $\widehat{S}(0)$ is given by

$$\widehat{S}(0)=\frac{S(N-1)+S(N-2)+\cdots+S(0)}{N}. \tag{7}$$

Substituting the partial sums by the raw frequencies, we have

$$\widehat{S}(0)=\frac{F(N-1)+2\cdot F(N-1)+\cdots+N\cdot F(0)}{N} \tag{8}$$

It is obvious that changing any raw frequency would affect the overall average $\widehat{S}(0)$ and the effects can be figured out without any difficulty.

Lemma 2 *For any attribute value i, we consider the effects of its frequency change $\Delta(i)$ from time t_0 to time t_1. For each wavelet coefficient $\widehat{S}(j)$, where internal node $j \in path(i)$ in the error tree, we have*

$$\widehat{S}_{t_1}(j)=\widehat{S}_{t_0}(j)+\frac{weight(j,\,i)\cdot\Delta(i)}{2^{height(j)}}, \tag{9}$$

where

$$weight(j,\,i)=\begin{cases} d-r & for\ 1\le j<N; \\ N-i & for\ j=0. \end{cases} \tag{10}$$

$$d = 2^{height(j)-1} \tag{11}$$

$$r = \left|\frac{L\_leaf(j)+R\_leaf(j)+1}{2}-i\right| \tag{12}$$

and N is the domain size of the attribute.

From Lemma 2, we obtain the following:

Theorem 1 *For any data update, we can compute the value change of each affected wavelet coefficient of partial sums in constant time using a closed-form formula.*

4.3. The Maintenance Method

The maintenance method of raw-data-based histograms can be divided into two parts. Part 1 updates the wavelet coefficients in the present histogram H and the auxiliary histogram H' using Lemma 1. Part 2 detects surprising coefficients that are currently not in the histograms but become significant after data updates by probabilistic counting. The probabilistic counting technique requires only one scan over the log and has no storage overhead. It is shown to be very effective in detecting surprising coefficients [7]. For the maintenance of partial-sum-based histograms, apart from the histogram for selectivity estimation, we also keep an auxiliary histogram on disk so that we have reasonable amount of extra information on the possible candidates that may get into the histogram. To update the wavelet coefficients in the histograms, we can use the fast update technique presented above. Below we discuss how to adapt the probabilistic counting technique to detect surprising coefficients for partial-sum-based histograms.

For the raw-data-based case, a wavelet coefficient may be positive, negative or zero. A significant coefficient may be a large positive number or a negative number with a large magnitude. Thus, we can change an insignificant coefficient to a significant one by either incrementing or decrementing its value. The maintenance method of raw-data-based histograms will treat an insignificant coefficient as surprising if the coefficient is estimated to have been incremented or decremented by a certain amount. Both insert and delete operations can increment and decrement the values of coefficients and hence are handled similarly. For the partial-sum-based case, the situation is quite different. From Equations 6 and 8, we can see that a wavelet coefficient of partial sums is a weighted sum of raw frequencies of attribute values and must be non-negative. Hence, a significant wavelet coefficient of partial sums must be a large positive number and we cannot make a coefficient surprising by decrementing its value. We will treat an insignificant coefficient as surprising if the coefficient is estimated to have been incremented by a certain amount. Furthermore, from Equation 9, we know that any insert operation can only increment some coefficients whereas any delete operation can only decrement some coefficients. As a result, we handle insert operations and delete operations differently. When the size of the log L reaches Max_Log_Size, we start to process the log entries sequentially. If a log entry is a delete operation, we just update the affected coefficients in H'. If a log entry is an insert operation, in addition to updating the affected coefficients in H', a probabilistic count technique is used to decide if any affected coefficients outside the current histogram and auxiliary histogram should be placed in the auxiliary histogram H'.

Before performing probabilistic counting, we need to determine the probability of flipping a head. As mentioned in the probabilistic counting method, we should set the probability to a value such that the reciprocal of that value equals to the number of insert (or delete) operations at the attribute value of the current log entry needed to bring the affected coefficient from zero to a certain threshold v. According to Equation 9, we can derive the probability of flipping a head

$$p(j) = \frac{weight(j,i)}{v \times 2^{height(j)}}. \tag{13}$$

With the above ideas, we can maintain wavelet-based histograms built on partial sums incrementally. Since our method only performs probabilistic counting for insert operations, we call it the *semi-probabilistic-counting method (semi-PC)*. Note that we can extend the fast update technique and hence the semi-PC method to the multi-dimensional case. For details, please refer to [4].

5. Performance Studies

In this section, we report on performance studies on the efficiency of the two update techniques and the accuracy of the semi-PC method. All experiments were conducted on a computer with four Intel Pentium III Xeon 700MHz CPUs with 4096M main memory running Solaris. All programs were coded in C++.

5.1. Data Distributions

The data used in the experiments are similar to those of [7]. We model the data using an extensive set of Zipf distributions [11]. The Zipf distribution was chosen because it is an well-accepted model on the skewness of real-life data. Without loss of generality, we use an attribute with the integer value domain. The z value in the Zipf distribution for the frequency distribution was varied from 0 to 2. The larger the z value, the more skew the frequency distribution is ($z = 0$ corresponds to the uniform distribution and $z = 2$ is already very skew). The frequencies are assigned to the attribute values according to three different types of correlations: positive (the bigger the attribute value, the higher the frequency), negative (the bigger the attribute value, the lower the frequency) and random. We refer a Zipf distribution with the parameter z and correlation X as $zipf\ (z,\ X)$.

5.2. Efficiency of The Two Update Techniques

We experimentally study the efficiency of the naive update technique and the fast update technique. The data we

used are one-dimensional. In the experiment, the spreads of the attribute value set follow the $cusp\_min$ distribution (decreasing spreads for the first half of the elements followed by increasing spreads) with Zipf parameter $z = 1.0$ and the number of distinct values of the attribute in the database (the value set size) is 512. The base dataset contains 100K tuples from the $zipf\,(1.0,\, negative)$ distribution. The update sequence contains 100K insertions from the $zipf\,(1.5,\, positive)$ distribution. Note that most of insertions are inserted at large attribute values and do not affect many partial sums. Hence, that gives a big advantage to the naive update technique. To evaluate the efficiency of the two update techniques, we first compute a partial-sum-based histogram from the base dataset and the histogram contains 20 coefficients. Then we read the update sequence and update the histogram accordingly. Without loss of generality, we do not maintain any auxiliary histogram and do not perform probabilistic counting in this experiment. We measure the time required to process the update sequence by the update techniques. Table 2 shows the results when the numbers of possible values of the attribute (the domain sizes) are 4096 and 8192. The fast update technique outperforms the naive update technique by a wide margin in both cases. Also, as the domain size is increased from 4096 to 8192, the run-time of the fast update technique remains about 5.5 seconds while that of the naive update technique almost doubles.

Domain Size	Fast	Naive
4096	5.47	3175.55
8192	5.48	5986.41

Table 2. Run-time of the two techniques (in seconds)

5.3. Methods of Comparisons

We implement the following three methods for the maintenance of partial-sum-based histograms and compare their accuracy.

1. *Ideal Method*. This method corresponds to recomputing the histograms from scratch upon updates to the database.

2. *Semi Probabilistic Counting Method (Semi-PC)*. This is the method described in Section 4.

3. *Static Method*. In the beginning, the histogram is computed from the base dataset. From then on, the values of the coefficients of the histogram would be updated accordingly but the choice of coefficients of the histogram would remain unchanged.

5.4. Parameter Settings

The parameters of the semi-PC method are set as follows. Parameters c_1 and c_2 depend on how aggressively we want to shake up the present histogram and auxiliary histogram. Small c_1 and c_2 result in aggressive shake-up of the histograms while large c_1 and c_2 result in a conservative policy. In our experiments, we find that the maintenance method performs well and is stable for a variety of update distributions as long as we choose c_1 and c_2 from reasonable ranges. The appropriate ranges for c_1 and c_2 are 1.0–2.0 and 0.4–0.8, respectively. We use default values $c_1 = 1.5$ and $c_2 = 0.5$ in our experiments. Another important parameter of the maintenance method is the size of the auxiliary histogram m'. Obviously, a large m' would give better accuracy but slower performance. However, our experiments show that it is not necessary to keep a big set of coefficients in the auxiliary histogram to achieve reasonable accuracy. We set the default value of m' to the size of the histogram m in our experiments. Lastly, we choose Max_Log_Size between 1% and 5% of the base data size.

5.5. Error Measure

The ideal method always chooses the set of the most significant coefficients to form the histogram. With proper normalization, the Haar basis is orthogonal. For any orthogonal wavelet basis, choosing the largest (in absolute value) wavelet coefficients is provably optimal in minimizing the 2-norm of the absolute error when considering the reconstruction of the original signal values [10]. Since we are now dealing with the wavelet decomposition of partial sums, the ideal method minimizes the 2-norm of the absolute error for partial sums:

$$\|e\|_2 = \sqrt{\frac{1}{N} \sum_{1 \le i \le N} (S_i - S_i')^2}, \qquad (14)$$

where N is the number of all partial sums and S_i (S_i') is the actual (estimated) value of a partial sum. Thus, a useful measure to evaluate the accuracy of a maintenance method is the 2-norm average absolute error for all partial sums using the maintained histogram after a sequence of updates. This measure clearly tells how close a histogram maintained using a particular method is to the one maintained using the ideal method.

5.6. Accuracy for Maintaining Histograms

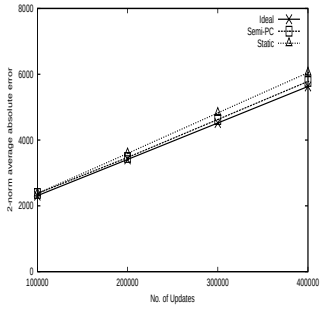
We experimentally study the accuracy of the three methods for a wide range of base data and update sequences. We

consider three classes of data updates, namely pure insertion, pure deletion, and mixed insertions and deletions. Below we show the results using one-dimensional data. The histogram size m is set to 20. The domain size is 4096 and the value set size is 512.

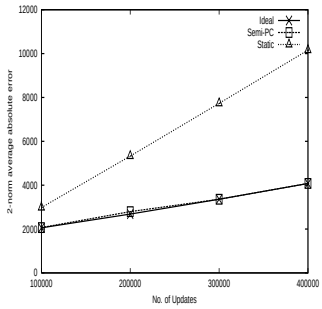
We first compare the accuracy of the three methods for pure insertion. Figure 2(a) shows one typical case when the distribution of the insertion sequence is similar to that of the base dataset. The base dataset contains 100K tuples from the $zipf\ (1.0,\ negative)$ distribution; the insertion sequence comes from the $zipf\ (1.2,\ negative)$ distribution. In this case, the set of significant coefficients would either remain the same or have minor changes after updates. The results show that the semi-PC method offers almost the same accuracy as the ideal method and the static method despite of its probabilistic nature. Figure 2(b) shows one typical case when the distribution of the insertion sequence is different from that of the base dataset. The base dataset contains 100K tuples from the $zipf\ (1.0,\ negative)$ distribution; the insertion sequence comes from the $zipf\ (1.5,\ random)$ distribution. In this case, the set of significant coefficients would become very different after updates. The results show that as the number of updates increases, the accuracy of the semi-PC method remains close to that of the ideal method but the gap between the static method and the ideal method widens.

We also compare the three methods for pure deletion. Intuitively, pure deletion is very challenging to the semi-PC method because it can no longer detect surprising coefficients outside the histograms. The semi-PC method would become the same as the static method except that the semi-PC method also maintains an auxiliary histogram from which it may be able to find candidates to replace small coefficients in the present histogram. Fortunately, since deletions can only decrease the values of wavelet coefficients, wavelet coefficients that are very small initially are unlikely to become members of the top m coefficients after deletions. In addition, even if some wavelet coefficients that are very small initially become members of the top m coefficients after deletions, they are not very likely to play a significant role because of their small values. As a result, we can achieve pretty good accuracy by keeping some coefficients that are initially "quite large" in the auxiliary histogram. Our experiments show that we do not need to keep a very large auxiliary histogram to have a reasonable accuracy. The appropriate range for m' is m to $2m$.

We study the effects of deleting different amount of data from the base dataset. Figure 3 depicts the accuracy of the methods in one set of our experiments. We generate three base datasets. Each of them consists a portion from the $zipf\ (1.0,\ negative)$ distribution and a portion from the $zipf\ (1.5,\ positive)$ distribution. The latter portion is used for deletion and its size ranges from 30% to 70% of the base



(a) Similar distributions



(b) Different distributions

Figure 2. Accuracy of various methods for pure insertion

dataset. The results show that the accuracy of the semi-PC method is very close to that of the ideal method when 30% and 50% is deleted from the base data. Although the semi-PC method degrades a bit when 70% is deleted, its accuracy is still close to that of the ideal method. On the other hand, the accuracy of the static method is close to that of the ideal method when 30% is deleted but the static method deteriorates quickly when 50% or 70% is deleted.

We compare the methods when the update sequence contains both insertions and deletions. Figure 4(a) shows one typical case. In this case, we generate three datasets containing 100K tuples. The first dataset comes from the $zipf\ (1,\ negative)$ distribution. The second dataset comes from the $zipf\ (1.5,\ positive)$ distribution. The third dataset comes from the $zipf\ (1.5,\ random)$ distribution. We combine the first dataset and the second dataset to form the base dataset. The second dataset is used as the deletion sequence and the third dataset is used as the insertion sequence. We combine the deletion sequence and the insertion sequence and arrange the entries in a random order. The results show that as the number of updates increases,

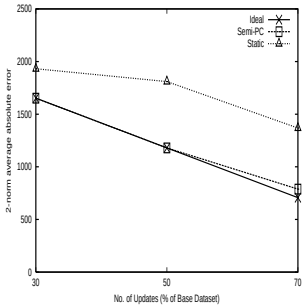


Figure 3. Accuracy of various methods for pure deletion

the accuracy of the semi-PC method stays close to that of the ideal method but the gap between the static method and the ideal method widens. Figure 4(b) shows another typical case. We generate the data as described above except that all insertions are arranged to occur before deletions. The results show that as the number of updates increases, the accuracy of the semi-PC method stays close to that of the ideal method but the gap between the static method and the ideal method widens.

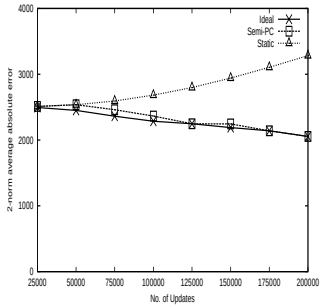
We have also performed experiments using two-dimensional datasets and three-dimensional datasets, and obtained similar results. For details, please refer to [4].

6. Conclusions

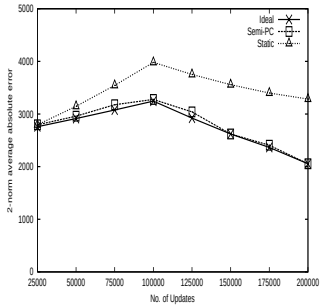
In this paper, we have proposed an efficient method for the maintenance of wavelet-based histograms built on partial sums. We show that a naive method of applying previous results on the maintenance of wavelet-based histogram is too slow to be practical. We propose a novel technique, the fast update technique, to compute the effects of data updates on the wavelet decomposition of partial sums and present a method, the semi-PC method, to maintain partial-sum-based histograms. Our performance studies show that the fast update technique is efficient and the semi-PC method performs well for a variety of data distributions and update sequences. One direction for future work is to address the maintenance of histograms based upon linear wavelets which offer more accurate approximation than Haar wavelets.

References

[1] D. Donjerkovic, Y. Ioannidis, and R. Ramakrishnan. Dynamic histograms: Capturing evolving data sets. Technical report, Department of Computer Science, University of Wisconsin-Madison, 1999.



(a) entries in random order



(b) all inserts before deletes

Figure 4. Accuracy of various methods for mixed distribution

[2] P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. 31(2):182–209, Oct. 1985.
[3] P. B. Gibbons, Y. Matias, and V. Poosala. Fast incremental maintenance of approximate histograms. In *VLDB*, 1997.
[4] K. F. Kan. Maintenance of partial-sum-based histograms. Master's thesis, University of Hong Kong, 2002.
[5] J. Lee, D. Kim, and C. Chung. Multi-dimensional selectivity estimation using compressed histograms. In *SIGMOD*, 1999.
[6] Y. Matias, J. S. Vitter, and M. Wang. Wavelet-based histograms for selectivity estimation. In *SIGMOD*, 1998.
[7] Y. Matias, J. S. Vitter, and M. Wang. Dynamic maintenance of wavelet-based histograms. In *VLDB*, 2000.
[8] V. Poosala and Y. Ioannidis. Selectivity estimation without the attribute value independence assumption. In *VLDB*, 1997.
[9] V. Poosala, Y. Ioannidis, P. Haas, and E. Shekita. Improved histograms for selectivity estimation of range predicates. In *SIGMOD*, 1996.
[10] E. J. Stollnitz, T. D. Derose, and D. H. Salesin. *Wavelets for Computer Graphics*. Morgan Kauffmann, 1996.
[11] G. K. Zipf. *Human Behaviour and the Principle of Least Effort*. Addison-Wesley, Reading, MA, 1949.