Dynamic Histograms: Capturing Evolving Data Sets

Donko Donjerkovic
Yannis Ioannidis
Raghu Ramakrishnan

Technical Report #1396

March 1999

Technical Report: CS-TR-99-1396. UW-Madison

Dynamic Histograms: Capturing Evolving Data Sets

Donko Donjerkovic, Yannis Ioannidis, Raghu Ramakrishnan
Department of Computer Sciences, University of Wisconsin–Madison
{donko,yannis,raghu}@cs.wisc.edu

Abstract

In this paper, we introduce *dynamic histograms*, which are constructed and maintained incrementally. We develop several dynamic histogram construction algorithms and show that they come close to static histograms in quality. Our experimental study covers a wide range of datasets and update patterns, including histogram maintenance in a shared-nothing environment. Building upon the insights offered by the dynamic algorithms, we also propose a new static histogram construction algorithm that is very fast and generates histograms that are close in quality to the highly accurate (but expensive to construct!) V-Optimal histograms.

1 Introduction

The cost of executing a relational operator is a function of the sizes of the tuple streams that are input to the operator, which for intermediate operators are in turn determined by selectivities of the previous operators. The more complex a query is, the more important it is to have precise intermediate size estimates. Otherwise, errors in the size estimates will grow intolerably (exponentially in the number of joins [2] in the worst case), and the optimizer's estimates may be completely wrong.

To estimate selectivities of query predicates, one needs to have some information about the data distributions of numerical attributes referred to in the predicates. There have been several proposals in the literature on how to maintain concise information about data distributions [14], including histograms, sampling, and parametric techniques. The most common technique in commercial systems is a histogram. *Histograms* are approximations to data distributions; they partition the data into subsets (called *buckets*) and maintain some aggregate information within each bucket.

Currently, histograms are static structures: they are created from scratch periodically and their creation is based on looking at the entire data distribution as it exists each time. This creates problems, however, as data stored in DBMSs usually varies with time. If new data arrives at a high rate and old data is likewise deleted, a histogram's accuracy may deteriorate fast as the histogram becomes older, and the optimizer's effectiveness may be lost. Hence, how often a histogram is reconstructed becomes very critical, but choosing the right period is a hard problem, as the following trade-off exists:

- If the period is too long, histograms may become outdated.
- If the period is too short, updates of the histogram may incur a high overhead.

In this paper, we propose what we believe is the most elegant solution to the problem, i.e., maintaining *dynamic histograms* within given limits of memory space. Dynamic histograms are continuously updateable, closely tracking changes to the actual data. We consider two of the best static histograms proposed in the literature [9], namely V-Optimal and Compressed, and modify them. The new histograms are naturally called *Dynamic V-Optimal (DVO)* and *Dynamic Compressed (DC)*. In addition, we modified V-Optimal's partition constraint to create the *Static Average-Deviation Optimal (SADO)* and *Dynamic Average-Deviation Optimal (DADO)* histograms.

We compare the effectiveness of dynamic histograms with Approximate Histograms [10], which are based on *Reservoir Sampling* [1]. Experimental results clearly show that the Dynamic Average-Deviation

Optimal histogram is the most effective approach to capturing evolving data sets. We study a wide range of datasets and data updating patterns, as well as histogram maintenance in a distributed shared-nothing environment, and include a comparison with the best known static histogram techniques.

The concept of dynamic histograms, the algorithms for maintaining such histograms, and the evaluation of their effectiveness is the main contribution of this paper. However, a second important contribution is a highly effective, inexpensively computed class of *static* histograms, called *Successive Similar Bucket Merge* (SSBM) histograms. Based upon the same intuitions underlying our dynamic histogram algorithms, the static SSBM histograms are comparable in quality to the highly accurate (but expensive to construct!) V-Optimal histograms. (We note that the static SADO histograms are also proposed here for the first time.)

The rest of this paper is organized as follows. We briefly discuss related work and review relevant histogram definitions in Section 2. We then present our first dynamic histogram, DC, in Section 3. We present the DVO and DADO dynamic histograms (and the static SADO histogram) in Section 4, and the highly effective SSBM static histogram in Section 5. Finally, we discuss dynamic histogram maintenance in a shared nothing environment in Section 8, and then outline future work.

2 Previous Work

Recent work on Approximate Histograms [10] has the same objectives as ours but takes a very different approach. It considers versions of Equi-Depth and Compressed histograms constructed from a reservoir sample [1]. The idea is to maintain a large reservoir sample (called the 'backing sample') on disk and a small approximate histogram in the main memory. Equi-Depth (and Compressed) histograms used for this purpose are approximate since they do not maintain equi-count buckets but the deviations in the bucket counts must not exceed certain threshold T. During the insertions or deletions of data, both in memory structure and the reservoir sample are updated. When the count in some bucket exceeds the threshold T, an attempt is made to split this bucket and merge one neighboring pair. If this cannot be done (because the merge would exceed T), the existing approximate histogram is discarded and a new one is built from the reservoir sample. We compare dynamic histograms with Approximate Histograms in later sections.

Another dynamic technique for approximating distributions is the Birch clustering algorithm [3]. Birch was originally designed for detection of clusters in large multidimensional data distributions, and later extended to approximate the data distributions themselves using *kernel theory* [4]. The basic building block of Birch is the *cluster*, which plays a role analogous to the bucket in a histogram. All Birch clusters have a common radius, which makes them similar to Equi-Width histogram buckets. It has been shown earlier [8] that Equi-Width histogram are in most cases inferior to Equi-Depth histogram, which are in turn inferior to the V-Optimal and Compressed histograms. Hence, for one-dimensional distributions, we expect the best histograms to be superior to Birch. We included Birch in our experimental study, and found that the best histograms indeed significantly outperformed Birch; due to lack of space, we do not discuss Birch further.

2.1 A Framework for Histograms

A histogram approximates a data distribution by partitioning it into buckets and summarizing each bucket by some concise information on the attribute values that fall in the bucket and their corresponding frequencies. Typically, each bucket has the minimum and (optionally) the maximum value in the bucket, a count of the data points it contains, and possibly the number of unique values it contains. This information is enough to generate an approximation of the distribution inside the bucket, assuming that within each bucket the following hold: points have fallen uniformly in the value range of the bucket (*uniform distribution assumption*); every value within the bucket value range has appeared in the data (*continuous value assumption*). Regarding the value distribution, there are indeed better assumptions than the continuous value assumption [9], but we have chosen this one due to its simplicity.

The following three dimensions are used to describe histograms in the histogram framework described in [9], which we use in our presentation: [1] *(1) Sort Parameter:* Conceptually, the data distribution elements

[1] For the referees' convenience, we have included a summary of this framework in Appendix A.

are sorted on their corresponding values of the histogram's sort parameter; the histogram buckets are then contiguous, non-overlapping groups in that sorted order. *(2) Source Parameter:* Used to determine where in the sort-parameter order bucket borders are placed. *(3) Partition Constraint:* A constraint on source parameter values that characterizes a histogram class.

3 Dynamic Compressed (DC) Histogram

A Compressed histogram stores some number of points in singleton buckets while the rest of them are partitioned as in an Equi-Sum histogram [8, 9]. In this paper, we concentrate on Compressed(V,F) histograms, i.e., the frequencies determine the buckets (source parameter). Therefore, singleton buckets are justified for values whose frequency exceeds N/n (N is the total number of points, n is the number of buckets). In the rest of this paper, we call singleton buckets *singular* and equi-depth buckets *regular.* Note that an Equi-Depth histogram is a special case of a Compressed histogram, with no singular buckets. A bucket of a Dynamic Compressed (DC) histogram records its left border and the number of points it contains.

The general idea behind all dynamic histograms is to relax histogram constraints up to a certain point, after which the histogram is reorganized in order to meet constraints. A DC histogram is constructed as follows. Initially, n distinct points are loaded into the histogram, each defining an individual bucket (n is the number of buckets we can maintain, given available memory). After this 'loading' phase, every new value is inserted into the appropriate bucket, possibly extending the leftmost or rightmost border to do so. When the regular buckets end up having radically different counts, thus violating the partition constraint of Compressed histograms, repartitioning occurs. Repartitioning is quite simple: it uses the point counts that are maintained in each bucket of the histogram and respecifies bucket boundaries so that the partition constraint is satisfied again. During this process, some regular buckets may become singular and vice versa, depending on whether or not they satisfy the criterion *count* $> N/n$, and, of course, have width equal to one.

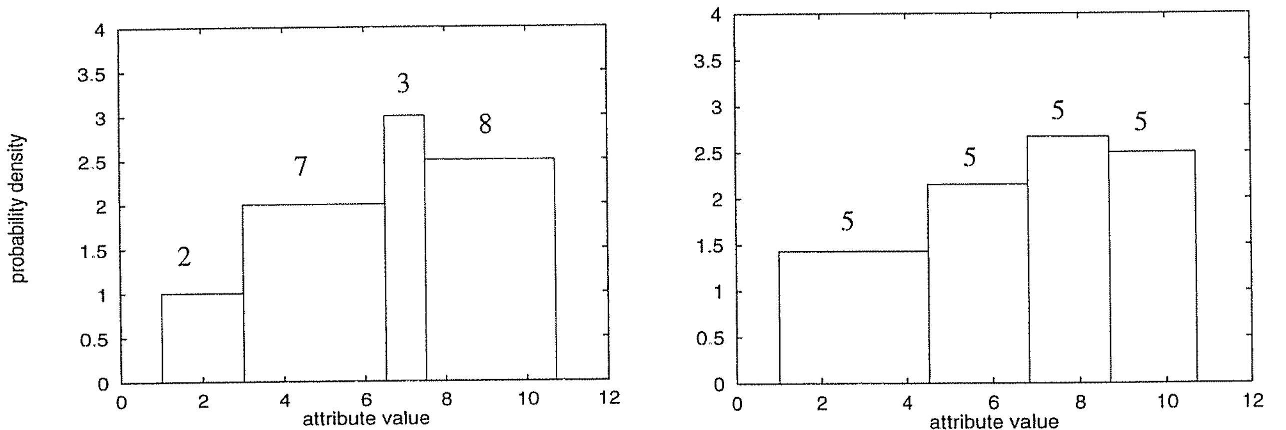

Figure 1: Bucket redistribution in DC histogram. Number above bucket is its area.

An example of repartitioning a DC histogram (with no singular buckets) is shown if Figure 1. Repartitioning is done in such a way that the total area and number of buckets remain the same. (Area of each bucket in Figure 1 is shown above the bucket.) Before repartitioning, the area underneath each bucket was significantly different. After repartitioning, all such areas are equalized. In this example, all buckets are regular both before and after repartitioning, but in general this is not the case.

The above captures the essential mechanics of the DC histogram. The question that remains is when repartitioning should be done. Doing it after every point received will result in both unacceptable performance degradation and probably poor histogram quality, since repartitioning introduces errors in the histogram due to the uniformity assumptions employed within each bucket. In fact, histogram quality may be in even higher jeopardy, since random oscillations in the order in which data is received could cause "false alarms", triggering the modification of the bucket boundaries unnecessarily.

What is needed is a test that will trigger repartitioning only when the Compressed partition constraint is *significantly* violated, i.e., when the counts in the regular buckets vary significantly. In statistical terms, repartitioning should occur when the following hypothesis (called the *null hypothesis*) is found to be false:

Counts in regular buckets are uniformly distributed.

The standard test used for this purpose is the *Chi-square* test. The Chi-square metric is

$$\chi^2 = \sum_i \frac{(c_i - C_i)^2}{c_i}, \tag{1}$$

where each (C_i) is the experimentally determined number of events of some category i, and c_i is the expected number of events according to the null hypothesis. In our context, (C_i) is the count of points in each regular bucket i, and c_i is the average count in regular buckets. A large value of χ^2 indicates that the null hypothesis is unlikely to hold. The *level of significance* α is the probability of rejecting a true null hypothesis, i.e., considering the null hypothesis as false when it is actually true. Given a value of χ^2, the significance level of the Chi-square test is given by the well known *Chi-square probability function* [7].

Based on the above, setting an upper bound on the deviation of a histogram from the Compressed partition constraint before repartitioning is triggered is equivalent to setting a lower bound α_{min} on the significance level. The lower α_{min} is set, the less frequent redistributions will be. Clearly, setting α_{min} to 0 would effectively freeze the initial histogram and never allow any repartitioning. On the other hand, setting α_{min} to 1 would trigger repartitioning after every insertion. In our performance evaluation, we have found that the algorithm is quite insensitive to the value of α_{min}, as long as it is much less than 1. In our experiments, we have set $\alpha_{min} = 10^{-6}$. Pseudo-code for the DC algorithm is presented in Figure 2.

```
Algorithm DC (data stream, number of buckets n) {

// Maintains an array of buckets that approximate numerical records
// seen on a potentially unbounded data stream.
// Each bucket stores its left border and the count c of points in it.
// The number of points read is denoted by N.

    read the first n distinct points;
    set the bucket borders between them;
    do until end of file {
        read the new point x;
        if x is beyond the range of end buckets {
            extend the appropriate regular bucket up to x;
        }
        insert x into appropriate bucket;
        if Probability(χ²) < α_min {
            degrade singular buckets with c < N/n to regular;
            redistribute the regular buckets to equalize their counts;
            promote regular buckets with width one and c > N/n to singular;
        }
    }
}
```

Figure 2: DC algorithm

3.1 Cost

Recall that N is the number of data points and n is the number of buckets. Processing a new point requires finding the appropriate bucket by binary search and increasing its counter (cost $O(\log n)$ for each point, $O(N \log n)$ total). Depending on the order in which data is read, the bucket borders may

occasionally be reorganized. Repartitioning essentially costs $O(n)$ each time, so assuming that it is done relatively infrequently, the overall cost of DC is $O(N \log n)$. Note that this is much lower than the cost of constructing a static Compressed histogram, which is $O(N \log N)$. For potentially lower accuracy, higher efficiency is obtained.

The space that is required by a DC histogram is the same as for its static counterpart. For each bucket, it stores its left border and the count of points in it. Its right border is assumed to be the left border of the next bucket. Altogether, the space requirement is

$$(n+1) * sizeof(data_type) + n * sizeof(counter_type)$$

4 Dynamic V-Optimal and DADO Histograms

A V-Optimal histogram minimizes (over all buckets) the variance between the source-parameter values within each bucket [8]. In this paper, we concentrate on V-Optimal(V,F) histograms, i.e., again the frequencies determine the buckets (source parameter). Therefore, the histograms considered minimize the quantity

$$\epsilon = \sum_{i=1}^{n} p_i V_i, \tag{2}$$

where n denotes the number of buckets, p_i is the number of frequencies in bucket i, and V_i is the variance of frequencies in the ith bucket. Expanding Eq. (2) further, we see that V-Optimal histograms essentially minimize

$$\epsilon = \sum_{i=1}^{n} \sum_{j} (f_{ij} - \overline{f}_i)^2, \tag{3}$$

where f_{ij} denotes frequency of the jth value in the ith bucket, and $\overline{f}_i$ is the average frequency in that bucket. Here, we assume that j ranges over all possible domain values within the ith bucket.

Note that unlike the Compressed partition constraint, we cannot check the V-Optimal constraint just by looking at the aggregate information that is typically stored in each bucket (left border and count, or equivalently average frequency). Eq. (3) requires that the set of the individual frequencies f_{ij} in each bucket be known. Clearly, storing the entire set is unrealistic and defeats the purpose of using histograms. One has to settle for some approximation that is, nevertheless, more detailed than the average frequency, since the latter would always generate zero ϵ in Eq. (3). Our approximation consists of dividing each bucket into two parts of equal value-range width, called *sub-buckets*, and storing the individual counts of points that belong in each sub-bucket. We have also tried other alternatives by combining different choices in the following dimensions:

- dividing each bucket into more than two parts;
- using *equi-depth* divisions instead of *equi-width* divisions.

Experimentation has shown that all alternatives with a small number of sub-buckets (two or three) have comparable performance, with finer subdivisions being worse. Intuitively, this trend is to be expected because, for example, with a large number of equi-width sub-buckets histograms become more equi-width than V-Optimal in nature. Given the empirical "optimality" of our approximation, we do not discuss any other alternatives in this paper.

With the above internal bucket structure in place, the DVO algorithm is able to approximate the dynamic minimization of Eq. (3) using two operations:

- *Splitting* a bucket along the sub-bucket border to generate two new buckets. The sub-buckets of each new bucket have equal counts. Since each of the new buckets has zero ϵ, splitting never increases ϵ.
- *Merging* two neighboring buckets to generate a single new bucket. The sub-bucket counts of the new bucket are calculated based on the counts and ranges of the original buckets. Straightforward manipulation of Eq. (3) shows that the ϵ of the new bucket is greater than or equal to the sum of the ϵ of the original buckets. Hence, merging never decreases ϵ.

Since the memory used for a histogram is fixed, repartitioning in the DVO algorithm consists of a split-merge pair, i.e., splitting a high-ϵ bucket and merging two neighboring buckets of similar characteristics. The resulting change $\Delta\epsilon$ in overall ϵ (Eq. (3)) is equal to

$$\begin{aligned} \Delta\epsilon &= \epsilon_M - \epsilon_S \\ &= \sum_j (f_{Mj} - \overline{f}_M)^2 - \sum_k (f_{Sk} - \overline{f}_S)^2, \end{aligned} \tag{4}$$

where j runs over all values in the two merged buckets, $\overline{f}_M$ is the average frequency of these values, k runs over all values of the split bucket, and $\overline{f}_S$ is the average frequency of these values. The bucket to be split and the two buckets to be merged are selected from all possible candidates so that $\Delta\epsilon$ is minimized. If $\min\Delta\epsilon$ is positive, repartitioning is not done.

The above description captures the essential mechanics of the DVO histogram. The question that remains is when repartitioning should be done. This is determined by comparing $\min\Delta\epsilon$ with some upper bound beyond which repartitioning will not be triggered. This upper bound has to be non-positive, as positive values of $\min\Delta\epsilon$ imply that the original histogram is better than any other one can obtain by a split-merge pair. In our experiments, we have made the most aggressive choice and set this upper bound equal to 0.

The definition of $\min\Delta\epsilon$ does not lend itself to efficient calculation. Fortunately, one can show that identifying the triple of buckets involved in the split-merge operations (one to be split and two to be merged) that generates $\min\Delta\epsilon$ (if $\min\Delta\epsilon < 0$) can be found in linear time.

Theorem 4.1 *If* $\min\Delta\epsilon < 0$, *then the bucket to split is the one with the largest* ϵ.

Similarly, one can prove that the pair to be merged is the one that has the minimal combined ϵ.

Pseudo-code for the DVO algorithm is presented in Figure 3. It uses the efficient $\min\Delta\epsilon < 0$ evaluation algorithm mentioned above. As before, the input parameter is the number of buckets n.

4.1 Dynamic Average-Deviation Optimal (DADO) Histogram

The original definition of the V-Optimal histogram calls for the minimization of the sum of the squares of the deviations of frequencies from their average (Eq. (3)). Nevertheless, our experimental results (see, for example, Fig. 9) indicate that better results are achieved by minimizing the sum of the absolute values of these deviations instead:

$$\epsilon = \sum_{i=1}^{n} \sum_j |f_{ij} - \overline{f}_i|. \tag{5}$$

A histogram with this partition constraint is called Static Average-Deviation Optimal (SADO) and the corresponding dynamic version is called Dynamic Average-Deviation Optimal (DADO). Eq. (5) is a more robust estimate of the deviations than Eq. (2). In other words, ϵ as defined in Eq. (5) is less sensitive to the frequency outliers. (Outliers are the data points far from the average.) We expect input data to possibly have large random oscillations in frequencies and consequently outliers may be common. This would explain why the DADO histogram is on average better than the DVO histogram, especially in the skewed distributions. There are no random oscillations during the construction of static histograms and therefore there is essentially no difference between the static V-optimal and the static Average-Deviation optimal histograms. The DADO algorithm, and all the results presented above remain the same as the DVO version provided that the square is replaced by absolute values.

4.2 Example of DADO (or DVO) Operation

Figure 4a shows a DADO histogram with five buckets. There are two sub-buckets (counters) per bucket, and bucket borders are marked by vertical lines. The second bucket from the left has high ϵ because its counters are very different. Assume that the next point read is number 3, which is inserted into the left sub-bucket of the second bucket. Recalculating $\min\Delta\epsilon$ produces a negative value, because the ϵ in the second bucket is larger than the combined ϵ in the third and fourth buckets. This triggers a split of the

```
Procedure DVO (data stream, # of buckets n) {
// Maintains an array of buckets that approximate numerical records
// seen on a potentially unbounded data stream
// Each bucket stores its left border and counters for its two sub-buckets.
    read first n points and create buckets between them;
    do until end of file {
        read next point (x);
        if x is beyond the range of the end buckets {
            Create a new bucket just for this point; // borrow one bucket
            Bucket b = findBestToMerge(buckets);
            merge b and b.next; // one bucket less
        } else {
            insert x into appropriate bucket;
            Bucket s = findBestToSplit(buckets);
            Bucket m = findBestToMerge(buckets);
            if (ε(s) > ε(m + m.next)) {
                split s;
                merge m and m.next;
            }
        }
    }
}
Procedure findBestToSplit (DVO_histogram) {
// Return the bucket with highest ε
}
Procedure findBestToMerge (DVO_histogram) {
// Return the bucket b such that the ε of b bucket combined
// with its successor is smallest among all the pairs.
}
```

Figure 3: DVO (DADO) algorithm

second bucket and a merge of the third and fourth buckets. Fig. 4b shows the result. Note that the counters in the merged bucket are deduced from the old configuration while the counters in each of the buckets that resulted from the split are set equal.

4.3 Note on Buckets vs. Sub-Buckets

By looking at the typical DADO histogram (Figure 4), one may argue that what we call "buckets" may be eliminated from the algorithm completely and what we call "sub-buckets" may be considered the true buckets. This is certainly a valid alternative view of the problem, but one needs to be cautious. The DADO histogram does not treat sub-buckets symmetrically; for example, large ϵ of sub-bucket counters within a single bucket is considered harmful, but large ϵ between two neighboring sub-buckets that belong to distinct buckets is ignored. This asymmetry is actually the main strength of the DADO algorithm, as it tends to place bucket borders at points where there are great differences in frequencies, and conversely, it spreads buckets across the smooth parts of the distribution. Therefore, even if one treats sub-buckets as the basic building blocks, one would still need some notion of "super-structure" in order to capture the AD-Optimal partition constraint; some notion of hierarchy in the bucket structure is necessary.

4.4 Cost

Clearly, the main loop of DADO is executed once for each point read, i.e., N times. The key operation in that loop in terms of cost is the identification of buckets s and m (Figure 3), which as we have shown, can

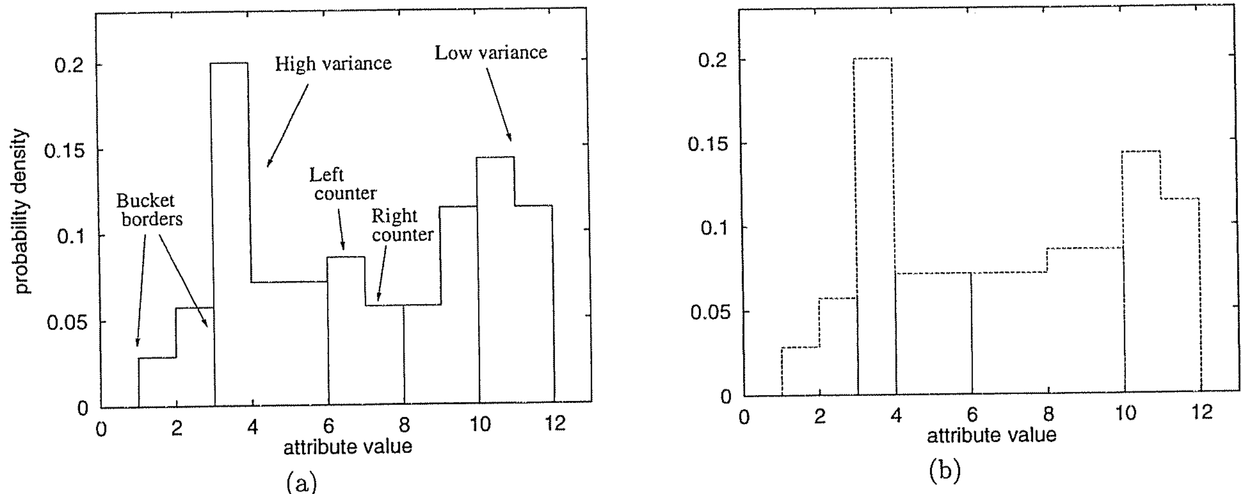

Figure 4: Example of DADO split & merge reorganization; (a) before, (b) after

be done in time linear in the number of buckets n. Hence, the overall cost of DADO is $O(Nn)$, larger than the cost of DC, which is $O(N \log n)$.

The space required by a DADO histogram is also slightly larger than that of a DC histogram (for the same number of buckets), because it stores its left border and two counters, one for each sub-bucket. Therefore, the total space requirement is

$$(n+1) * sizeof(data_type) + 2 * n * sizeof(counter_type)$$

5 A New Static Histogram Algorithm

The DADO merge technique suggests a new way of constructing static histograms, as described next. Initially, load all the data in an exact histogram (one bucket for each non empty distinct point). After the loading phase, successively merge the neighboring buckets with the smallest ϵ_M, as defined in Eq. 4. The algorithm starts merging the most similar buckets first (with small ϵ_M) and proceeds until the requested target number of buckets is left unmerged. We call such (static) histograms *Successive Similar Bucket Merge* (SSBM) histograms.

Performance comparison of SSBM histograms to various static histograms is shown later in Figs. 9, 10, 11, and 12. We found that SSBM is comparable in performance to V-optimal(V, F) histogram, which is one of the best known histograms, but is much cheaper to construct. The cost of SSBM construction is quadratic in the number of distinct attribute values, rather than exponential (as for V-optimal).

6 How to Measure the Quality of a Histogram?

Clearly, there are many ways to evaluate the performance of a histogram, and in general there is no definitive answer to the question of which algorithm is the best. One can only say which algorithm is best under certain test data and evaluation metrics. In the following subsections we present our test distributions and the metric that we believe to be the most suited for comparing different histograms.

6.1 Performance Parameters: Data Distribution and Memory

We evaluated the algorithms using a parametrizable data distribution, in order to measure trends in their relative behavior as the parameters are changed.

Many real distributions obey the Zipf [15] and Normal laws. Motivated by this, we have created distributions that contain clusters of data, characterized by the position of their center, their size, and shape.

The Zipf law governs positions and sizes of clusters. Correlation between cluster sizes and separations was chosen from three alternatives: no correlation, positive correlation, and negative correlation. The shape of clusters was chosen from three distributions: uniform, normal, and exponential. The width of clusters was parameterized by variable standard deviation σ. We did not detect significant variations in algorithm performance along the following dimensions: correlation between cluster sizes and separations, the shape of clusters, and the number of clusters. Therefore, all the performance results presented in this paper have the following data distribution parameters fixed: (1) CS, cluster shape (distribution), fixed to Normal. (2) SF, spread frequency correlation, fixed to random. (3) C, the number of distinct clusters, fixed to 2000 or to 50. Parameters that were varied are listed below:

- Z, the Zipf parameter of the skew in the distribution of cluster sizes.
- S, the Zipf parameter of the skew in the distribution of cluster centers.
- σ, the standard deviation within the clusters, if zero, each cluster has a single value.

By changing these parameters, it is possible to capture many different distributions. Essentially, our test distributions look very much like the ones from [9], with the addition of a parameterized cluster width (σ). Finally, we note that we varied the amount of memory (M) available for histogram construction.

6.2 What is a Good Evaluation Metric?

To evaluate the quality of a histogram we compared the original data distribution to the approximate data distribution represented by the histogram, using a *goodness-of-fit* test. The two most widely used goodness-of-fit tests are the Chi-Square and Kolmogorov-Smirnov tests. The Chi-Square test is usually used when the data involves categories (e.g., colors), and the Kolmogorov-Smirnov (KS) test [12] is the most generally accepted test for numeric distributions.

It should be emphasized that we only use the statistical metrics associated with the respective tests, and not their associated probability functions. That is, we just measure goodness-of-fit without worrying about the significance of the deviation. Significance is irrelevant because we are interested only in the relative performance of the algorithms.

The KS statistic for two distributions is defined as

$$D = \max_{-\infty < x < \infty} \mid P_1(x) - P_2(x) \mid \tag{6}$$

where $P(x)$ is the cumulative distribution function.

One can always group the data into bins and use the Chi-Square statistic on the the numeric distribution. Grouping involves a loss of information and moreover the choice of how to group the data is arbitrary. KS statistic does not require any unnecessary categorization and is also independent of any kind of re-parameterizations of the domain axis.

The KS statistic has an intuitive interpretation: It is the maximum error in selectivity of a range predicate posed against the histogram rather than the original data. Unfortunately, there is no similarly intuitive interpretation for the Chi-Square statistic.

We have also tried the error metric suggested in [9],

$$E = \frac{100}{N} \sum_{\text{all queries}} \frac{|S_q - S_q^{'}|}{S_q} \tag{7}$$

where S_q and $S_q^{'}$ are the actual and the estimated size of the query result, respectively. This error metric, although different from KS, gave similar results in terms of relative performance of our histogram construction algorithms. We preferred the KS statistic because the metric in Eq. (7) obviously depends on the test set of queries. For example, Eq. (7) would give different error for the set of range queries ($attribute < value$) depending how *value* is distributed. Two obvious choices for *value* distribution are uniform and the data distribution itself. Also, open range queries ($attribute < value$) would give different results than close range queries ($low < attribute < high$). To avoid any bias towards any of these queries, we have turned to the KS statistic as an established test for comparing two distributions.

7 Performance Evaluation

We have evaluated all algorithms along the three broad classes of tests:

1. Parameterized synthetic distributions:
   (a) random insertions
   (b) sorted insertions
   (c) random insertions intermixed with random deletions
   (d) random insertions followed by random deletions
   (e) sorted insertions followed by sorted deletions
2. Real-world distribution. This data was collected by a mail order company through the period of time. The data contains only insertion in approximately random order.

In order to make fair comparisons, *all of the algorithms were given the same amount of main memory.* In addition, we gave the Approximate Compressed (AC) histogram the disk space equal to twenty times the main memory (by default), following the suggestion by authors of [10]. (We have also tried other disk-space factors, as shown later in Fig. 14.) To obtain the best quality AC histogram possible, we set the γ performance parameter to -1. This causes recomputation at every update, which in general gives the best quality histogram, but the worst performance in terms of speed, according to [10].

We compare all algorithms on a parameterized family of data distributions. The test data was a file of 100,000 integer numbers, spread over the interval [0..5000] according to the parameters of the distribution (S, Z, ...). All the histograms are initially empty and are populated by numbers drawn from the test distribution in a random order. After all insertions are done, the histogram distribution is compared to the original distribution using KS statistic as an error metric. We chose the following reference distribution: ($S = 1$, $Z = 1$, $\sigma = 2$, Memory = 1 KB), and changed all the parameters, one at a time. Every test configuration was generated ten times (by starting from a different seed for the random number generators used in data set generation) and evaluated based on the average of ten measured KS statistics.

7.1 Random Insertions From Synthetic Distributions

Figures 5, 6, 7, and 8 show the performance of various histograms under a workload of random insertions, and can be summarized as follows:

DADO histogram has the best behavior among dynamic histograms. The algorithm works better for very high skews because in this case it can afford to create buckets with only one value in them, thus capturing large parts of distribution with almost perfect precision. On average, maximal error of the 1KB DADO histogram is below 0.5% of the relation size.

DVO histogram has similar behavior to the DADO histogram but it is consistently worse in quality. A likely reason for this was given in Section 4.1.

AC histogram has on average worse behavior than both DADO and DVO (in spite of the advantage of extra disk space and the favorable setting of $\gamma = -1$). Oscillations in quality for various S, Z, σ, and M are results of random fluctuations. Notice that for larger values of memory M (Figure 8), the AC histogram becomes even less effective compared to the DADO histograms.

DC histogram behaves roughly as its static counterpart with larger errors, as expected. In general, errors are small for low skews (because any histogram is good at uniform distributions) and high skews (because singular buckets capture most of the data). Along the same lines, errors are small for low σ because it effectively increases the skew of the distribution. Similarly, errors are low at large σ because it makes distribution uniform. The DC histogram has the most difficulty with intermediate σ. We found that this large increase in error for intermediate skews are accompanied by large number of border relocations. Border relocations certainly introduce errors because new border positions are calculated under assumption of uniform distribution within each bucket. Distributions with larger

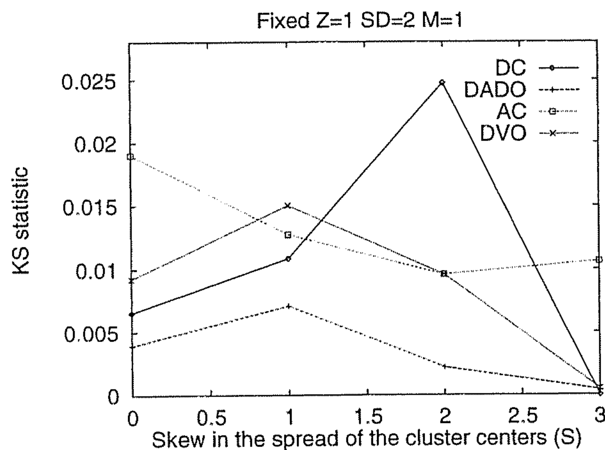

Figure 5: KS statistic as a function of S

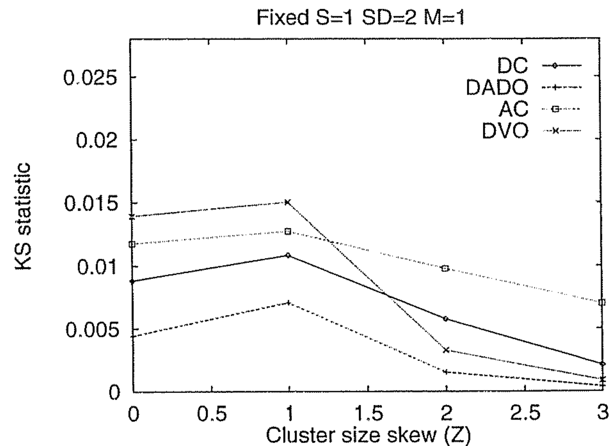

Figure 6: KS statistic as a function of Z

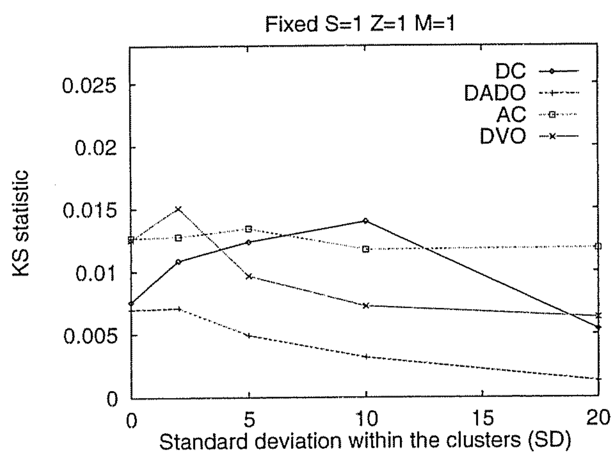

Figure 7: KS statistic as a function of σ

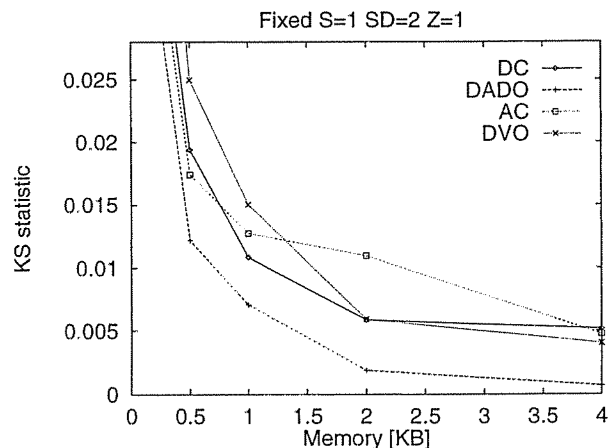

Figure 8: Error vs. available memory

skews have large random oscillations that cause unnecessary border relocations. The Chi-square test helps to reduce border relocations due to random oscillations but does not eliminate them completely. Unnecessary relocations are the primary source of errors of the DC histogram.

The above results were obtained using random insertions from the underlying distributions.

Since the experiments presented above indicate that the DADO algorithm has the best performance, it is of interest to compare it to the best static algorithms, such as Compressed (SC) and V-Optimal (SVO). In addition, we have implemented the new static Average-Deviation Optimal (SADO) and Successive Similar Bucket Merge (SSBM) algorithms. All the static histograms are of the same size as memory given to the DADO histogram. Construction of a SC histogram requires sorting of the input data set and for this purpose it was given as much memory (disk space) as needed. Similarly, the SVO, SADO and SSBM histograms require enough memory to fit all the data points. Results of this comparison are shown in Figures 9, 10, 11, and 12. From these figures we can see that the DADO algorithm comes close to the performance of its static counterpart and is very comparable to the SC algorithm. Optimizing for Average-Deviation or Variance seems not to make any difference in the static case but it makes a significant difference in the dynamic case, as explained in Sec. 4.1. Performance of the SSBM histogram is comparable to the SVO, however the cost of constructing the SSBM histogram is much smaller (quadratic in the number of distinct attribute values) than that of SVO (exponential in the number of buckets). Comparison of execution times is shown in Fig. 13.

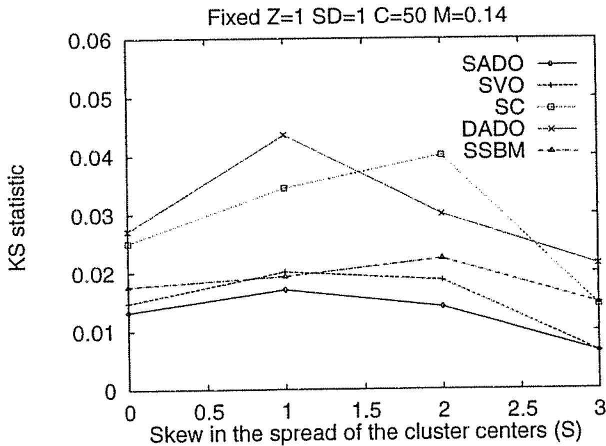

Figure 9: KS statistic as a function of S

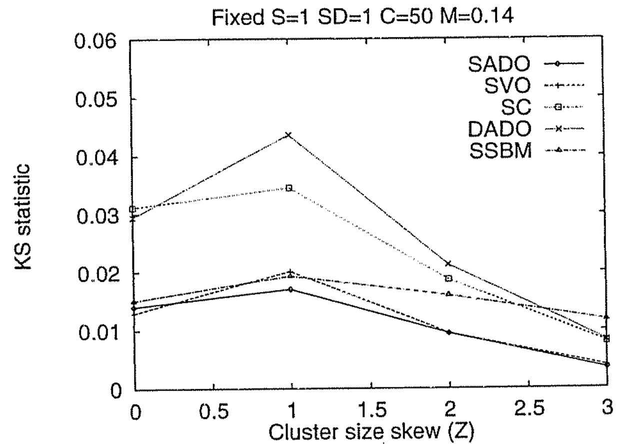

Figure 10: KS statistic as a function of Z

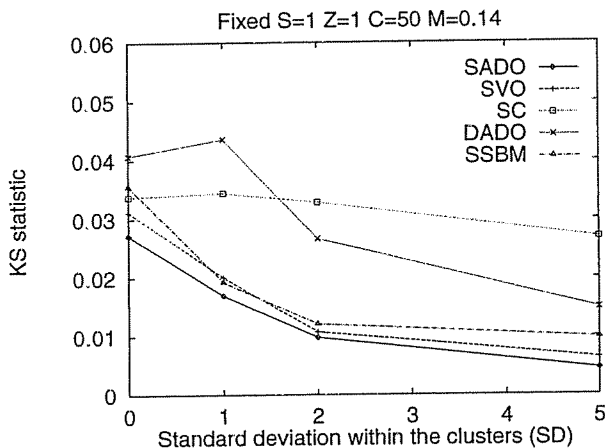

Figure 11: KS statistic as a function of σ

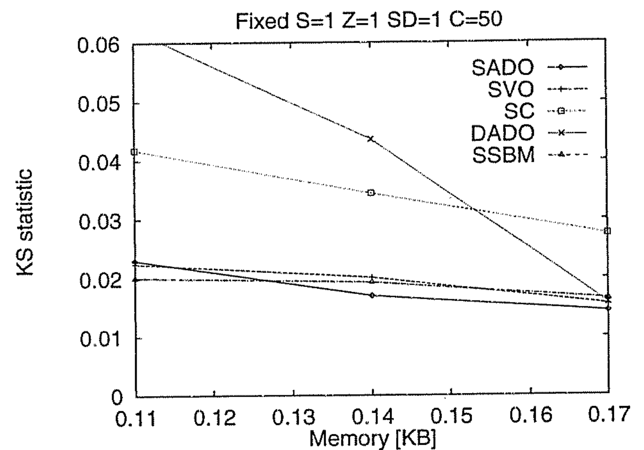

Figure 12: Error vs. available memory

Finally, we have performed disk-space sensitivity analysis of the approximate compressed histogram and the results are shown in Fig. 14. Numbers after "AC" denote the factor by which the disk space is larger than main memory space. Although the performance of the AC histogram increases as the disk-space factor increases, it is still worse than that of the DADO histogram, even for a factor as big as 60. Notice that the quality of the AC histogram slowly converges to that of the SC histogram as the size of the sample approaches the data size.

7.2 Sorted Insertions From Synthetic Distributions

In this section we present the results with insertions given in the same order as the domain values they represent (Fig. 15). Fig. 15 is a reproduction of Fig. 6 except for sorted insertion of data. Sorted insertions are more difficult to capture correctly for DADO and DC histograms because the distribution of received points is constantly changing. This is in contrast to random insertions, where the distribution of received points is static, modulo random oscillations. On the other hand, AC histogram with $\gamma = -1$ handles sorted insertions with the same precision as random insertions. This is because the reservoir sampling is "blind" on the input order, and when $\gamma = -1$ histogram is recomputed at any modification of the reservoir sample. We conclude from these results that although the performance of the best dynamic histogram DADO worsens with sorted input, it is still comparable or better than the AC histogram.

All the preceding error measurements were obtained after a given data set was completely read into a

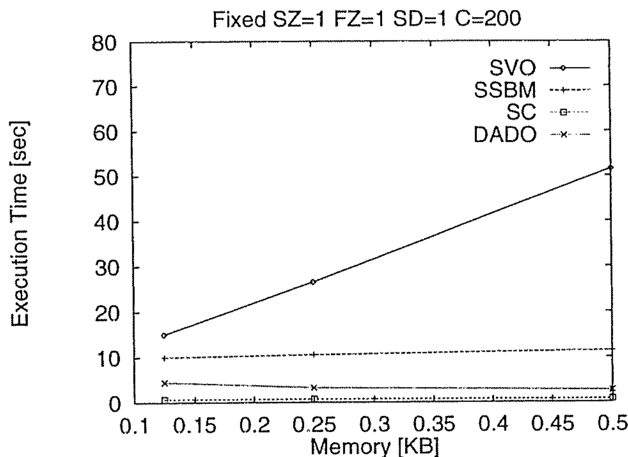

Figure 13: Typical execution times

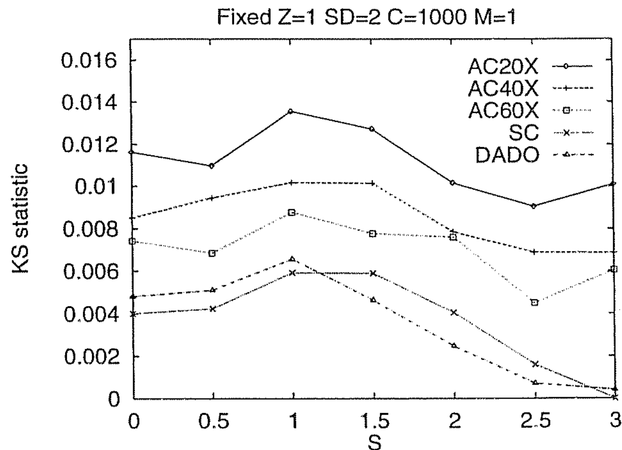

Figure 14: Sensitivity to available disk space

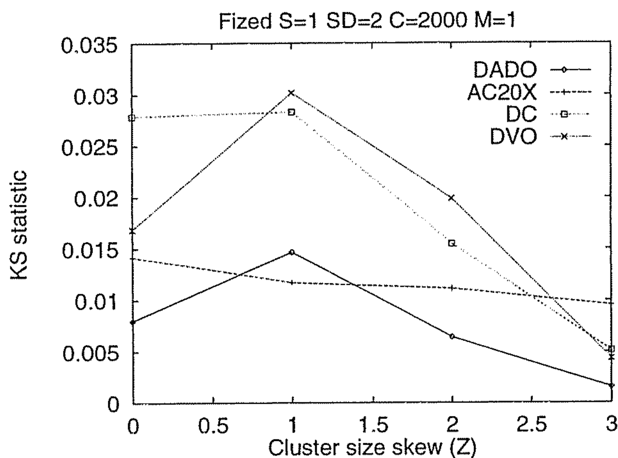

Figure 15: Sorted insertions.

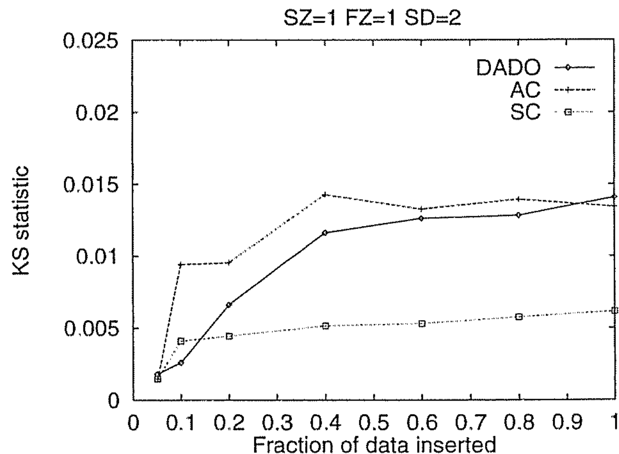

Figure 16: Error vs. volume of inserts

specific histogram. The following section presents the performance of histograms while the data is being read.

7.2.1 Histogram's Precision Degradation as the Data Size Increases

The following discussion relates to any histogram (static or dynamic) that maintains borders and point counts for each bucket. Initially, as only a few tuples are inserted into a histogram, the distribution is captured precisely. Histograms remain accurate as long as we can store each distinct point in a separate bucket and still have enough buckets to represent empty spaces between these points. When even more distinct points are inserted, histograms become an approximation to the data distribution. Initially, approximation becomes worse with the increase of the data size, but in general we expect this approximation to stabilize. For example, the maximal error of equi-depth histogram, measured by the KS statistics, is limited to $1/\beta$ where β is the number of buckets. In the following experiment, we have measured the error of various histograms for certain fractions of the data loaded. I.e., we measure the error when 5% of the data is read, 10% and so on.

Results of our experiments are shown in Fig. 16 and conforms the preceeding analysis. In these experiments we have measured KS statistics for 1KB DADO, Approximate Compressed (with 20KB disk space) and Static Compressed histograms for the reference distribution described in Section 6.1. Data was given in sorted order and the error was recorded for fractions of data being read. Each point represents

the average of 10 measurements.

It can be seen from Fig 16 that DADO histogram reaches a stable point after which the error does not significantly increase with additional insertions.

7.3 Deletions and Insertions

Deletion is simply the inverse of insertion and is naturally handled by decrementing the appropriate counters in the buckets of the DADO or DC histograms. Random (uniform) deletions do not significantly affect the performance of these two algorithms. On the other hand, frequent random deletions in general deteriorate performance of the AC histogram because they reduce the size of the backing sample. This trend is clearly shown in Fig. 17.

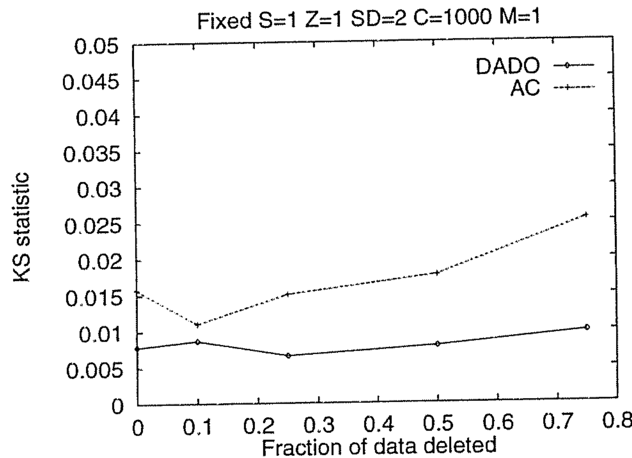

Figure 17: Error vs. volume of random deletes

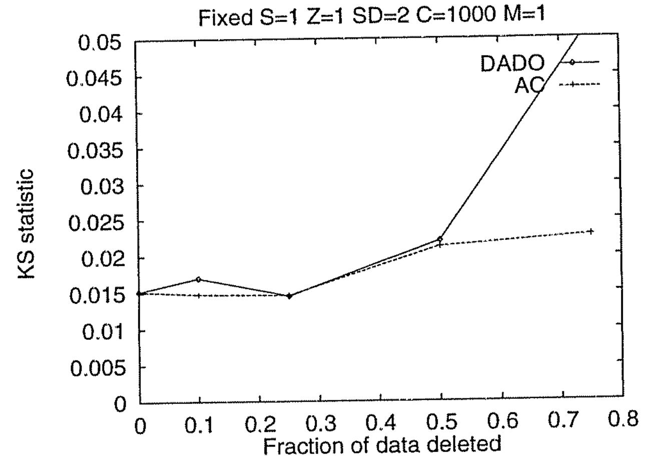

Figure 18: Random deletes after sorted inserts

However, we found that performance of DADO for deletions that follow sorted insertions suffers when the majority of data is deleted. In this case, counters in some buckets of the DADO histogram become zero and no additional points can be removed from them. This essentially means that some data was spilled over to the neighboring buckets. Since the point must be removed from somewhere, we find the closest bucket to the deleted point and decrement its counter. This policy works when the bucket overspilling is likely to occur to the left or to the right of the bucket in question (which is the case for random insertions). Unfortunately, for sorted insertions, bucket overspilling occurs only to the bucket closer to the center of the histogram, since the insertions always happen at the edge of the histogram. We were not able to correct this problem, but it is our opinion that the circumstance under which it occurs (sorted insertions and heavy deletions) are rare.

7.3.1 Histogram's Precision Degradation as the Data Size Increases

We monitored the performance of DADO histogram through time, in a similar fashion as in Sec. 7.2.1, with the addition of 25% deletion rate. In this experiment, the data was inserted in sorted order, after every insertion one tuple was chosen randomly to be deleted with the probability of 25%. We omit the results, which are similar to the experiments without deletions (Fig. 16).

7.4 Real-World Data

We have measured performance of all algorithms on a real data trace obtained from a mail order company. The data file contains 61,105 records (240 KB) that represent dollar amounts for each order and is shown in Figure 19. The results are shown in Figure 19. As expected, this figure does not deviate much from the corresponding figure for synthetic data (Fig. 8). It is interesting that KS statistic for DADO is very good for low memory (less than 1KB) but does not drop quite as quickly as $1/M$. This may be caused

by the very "spiky" nature of the data. We observe that while the DADO histogram has captured the outline of the data quickly, it obviously needs much more memory to capture this many spikes. The same observation appears to partially hold for DC also.

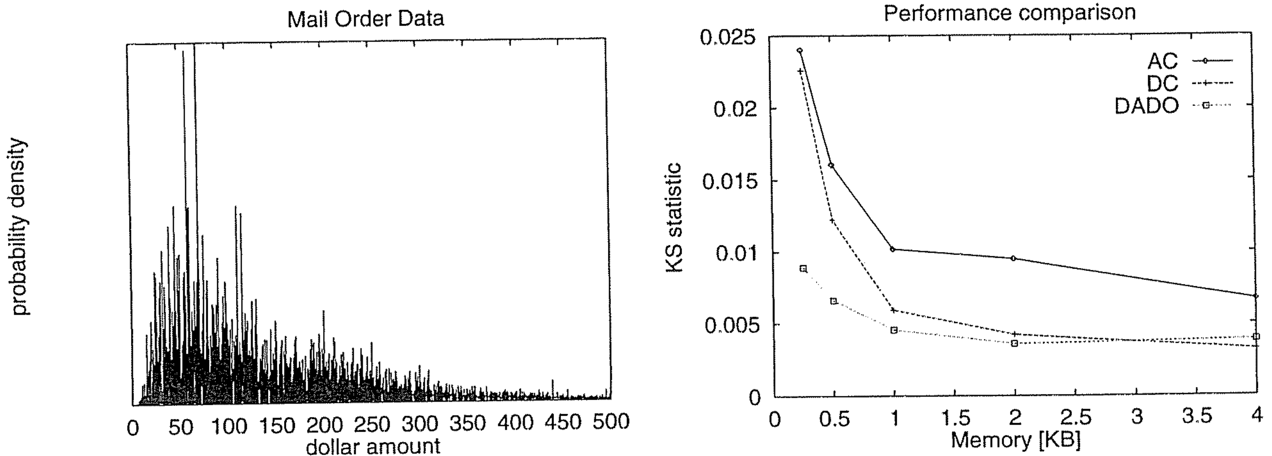

Figure 19: Mail Order Data

8 Global Histograms in a Shared-nothing Environment

In a distributed environment (such as the World Wide Web) large unions of tables with the same schema have been proposed as a model of scalable semantic integration [16]. Unions of tables are also present in the shared-nothing architecture for parallel databases, because data in such systems are partitioned across all the nodes.

Assuming that each union member has a histogram, it is desirable to build a global histogram, at the union level, using a limited amount of memory. A union histogram can be built by simply superimposing member histograms. The final histogram has a bucket border wherever either of the input histograms has a bucket border. Notice that this process does not involve any loss of information (the final histogram is as precise as the member histograms) and can be applied to an y histogram (static or dynamic). However, a composite histogram constructed using superposition may have a large number of buckets, and in some situations it may be desirable to reduce its size. To reduce the number of buckets, one can simply treat the histogram as a data set to be partitioned and use any partitioning strategy, such as equi-depth or V-optimal.

In addition to the approach of building the global histogram by merging the local histograms, described above, we can merge all the data first and then construct the global histogram directly. We now evaluate this tradeoff carefully.

We assume that histograms are of SSBM(V, F) class, and the merging technique used is also SSBM. For this purpose, all the histograms were given the same amount of memory M (by default 250 bytes), variations of which are shown in Fig. 20. Union members have data which is distributed within some range according to a Zipf distribution parametrized by Z_{Freq}, by default 1 (see Fig. 21). The attribute range of each union member is uniformly and randomly distributed. Number of data in each member is a zipf distribution with parameter Z_{Site}, by default 0 (see Fig. 23. Finally, the number of members N_{Site} was varied from the default value of 5 (see Fig. 22). Based on these figures, we conclude that the resulting histograms from each alternative are approximately of the same quality.

We also performed a similar experiment in which the local site histograms were given memory equal to M_{union}/N_{Site}, where M_{union} denotes the memory given to the final union histogram. In this case, the second alternative (histogram + union) performs worse, even though the aggregate amount of memory is the same for both alternatives. This is a consequence of the randomized partitions across sites: if the partitioning was disjoint, the performance of the two alternatives would be the same. These results are included in Appendix B.

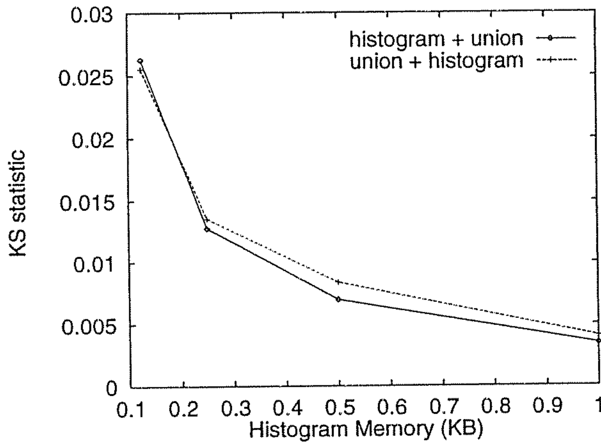

Figure 20: Error vs. histogram size

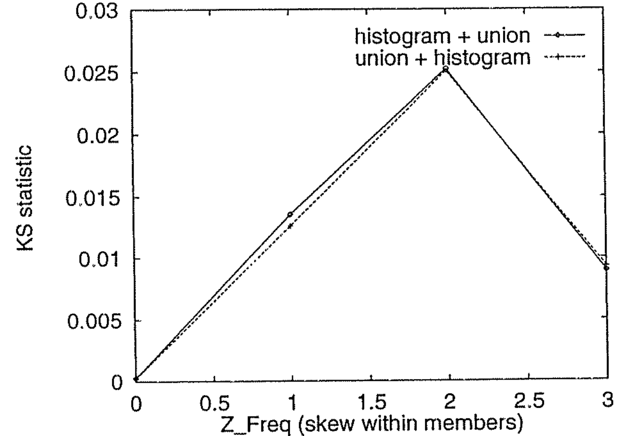

Figure 21: Error vs. intrasite data skew

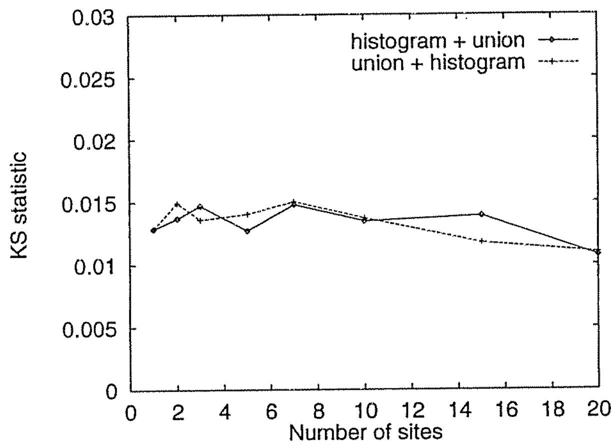

Figure 22: Error vs. number of sites

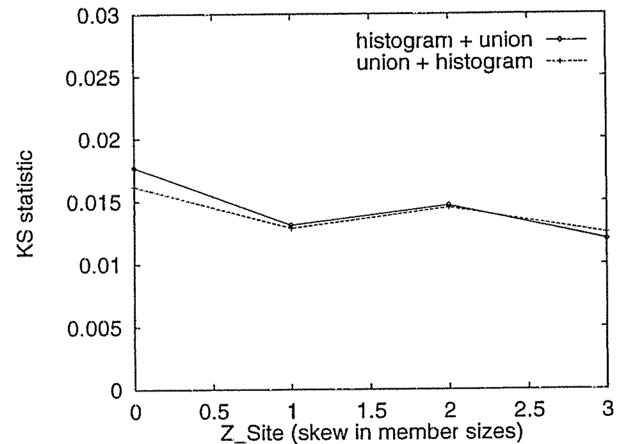

Figure 23: Error vs. skew in site size

9 Conclusion

We have developed two new histograms that can be incrementally maintained: DC and DADO. The DADO histogram showed stable behavior and came very close to the best static histograms in terms of how well they approximated the data distribution. Its performance is superior to that of Approximate Compressed and DC histograms when given the same amount of main memory, and furthermore, its error rate declines faster than the sampling error with increases in available memory. Dynamic histograms adapt equally well to both insertions and deletions of new data. We believe that the above observations are reliable indications that the Dynamic Average-Deviation Optimal histogram is probably the most robust and effective alternative for capturing evolving data sets. We also introduced a new static histogram, SSBM, that is close to the highly accurate V-Optimal histogram in estimation quality, but is far cheaper to compute. The most important direction of our future work is the extension of the DC and DADO algorithms to more than one dimension.

References

[1] J. S. Vitter. *Random Sampling with a Reservoir.* ACM Transact on Math. Software, Vol. 11, 1985.

[2] Yannis Ioannidis and Stavros Christodoulakis. *On the propagation of errors in the size of join results.* Proceedings of ACM SIGMOD, pages 268-277, 1991

[3] T. Zhang, R. Ramakrishnan, M. Livny. *Birch: An Efficient Data Clustering Method for Very Large Databases.* Proceedings of ACM SIGMOD, Montreal, Canada, 1996

[4] Miron Livny, Raghu Ramakrishnan, Tian Zhang. *Fast Density and Probability Estimations Using CF-Kernel Method for Very Large Databases.* Tech. Report, CS Dept. UW-Madison, July, 1996.

[5] V. Poosala and Y. Ioannidis. *Selectivity Estimation Without the Attribute Value Independence Assumption.* Proc. 23rd International VLDB Conference, Athens, Greece, August 1997, pp 486-495.

[6] M. Muralikrishna and D. J. DeWitt. *Equi-depth histograms for estimating selectivity factors for multi-dimensional queries.* Proc. of the 1988 ACM-SIGMOD Conf., pages 28-36.

[7] William H. Press, Brian P. Flannery, Saul A. and William T. Vetterling. *Numerical Recipes in C: The Art of Scientific Computing* Cambridge University Press, 1993.

[8] Y. Ioannidis and V. Poosala. *Balancing histogram optimality and practicality for query result size estimation.* Proc. of ACM SIGMOD Conf, pgs 233-244, May 1995.

[9] Viswanath Poosala, Yannis Ioannidis, Peter Haas, and Eugene Shekita. *Improved histograms for selectivity estimation of range predicates.* Proc. of ACM SIGMOD Conf, pgs 294-305, June 1996.

[10] P. B. Gibbons, Y. Matias, V. Poosala. *Fast Incremental Maintenance of Approximate Histograms.* Proc. of VLDB Conf., pgs 466-475, August 1997.

[11] P. B. Gibbons, Y. Matias, V. Poosala. *Fast Incremental Maintenance of Approximate Histograms.* Technical report, Bell Laboratories, Murray Hill, NJ, June 1997.

[12] F. J. Massey. *The Kolmogorov-Smirnov test for goodness-of-fit.* J. Amer. Statist. Assoc., 46:68-78, 1951

[13] A. N. Kolmogorov. *Confidence limits for an unknown distribution function.* Ann. Math. Statist., 12:461-463, 1941.

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

[15] G. K. Zipf. *Human Behavior and the Principle of Least Effort.* Addison-Wesley, Reading, MA, 1949.

[16] Raghu Ramakrishnan and Avi Silberschatz. *Scalable Integration of Data Collection on the Web.* Technical Report: CS-TR-98-1376, University of Wisconsin-Madison, June, 1998.

A Histogram Definitions

Of the four orthogonal dimensions that define the space of all histograms [9], the following three are critical for our work:

1. *Sort Parameter:* For a data distribution element, this is a function of the corresponding attribute value and/or frequency. Conceptually, the data distribution elements are sorted on their corresponding values of the histogram's sort parameter; the histogram buckets are then contiguous, non-overlapping groups in that sorted order.

2. *Source Parameter:* For a data distribution element, this is also a function of the corresponding attribute value and/or frequency. It is used in determining exactly where in the sort-parameter order bucket borders are placed.

3. *Partition Constraint:* This is a constraint that the source parameter values must satisfy to determine a unique histogram. It is typically a mathematical formula.

According to this classification, we identify a histogram using the following notation: *PartitionConstraint(Sort parameter, Source parameter)*. Some of the typical sort and source parameters of a data distribution element are the attribute value (V), the frequency (F), and the spread (S), which is the distance of the attribute value from the next largest value in the distribution. The following are some of the most important partition constraints:

- *Equi-Sum*: The sum of the source values in each bucket is equal;
- *Compressed*: Some number of the highest source values are stored individually in singleton buckets; the rest of the source values are partitioned based on an Equi-Sum histogram; and
- *V-Optimal*: The quantity $\sum n_j V_j$ is minimized, where n_j is the number of elements in the jth bucket and V_j is the variance of the source values in the jth bucket.

The traditional *Equi-Width* histogram is essentially the *Equi-Sum(V, S)* histogram, because it partitions the attribute value axis (V) so that the value range of each bucket is equal (S). Also, the traditional *Equi-Depth* histogram is essentially the *Equi-Sum(V, F)* histogram, because it partitions the attribute value axis (V) so that each bucket has the same number of elements (F). In this work, we concentrate on *V-Optimal(V, F)* and *Compressed(V, F)* histograms, which we simply refer to as Compressed and V-Optimal, respectively.

B Additional Shared-Nothing Results

The following graphs show the results we obtained in the shared nothing environment when local histograms are constructed with proportionally less memory than the global histogram. The results are summarized in Section 8.

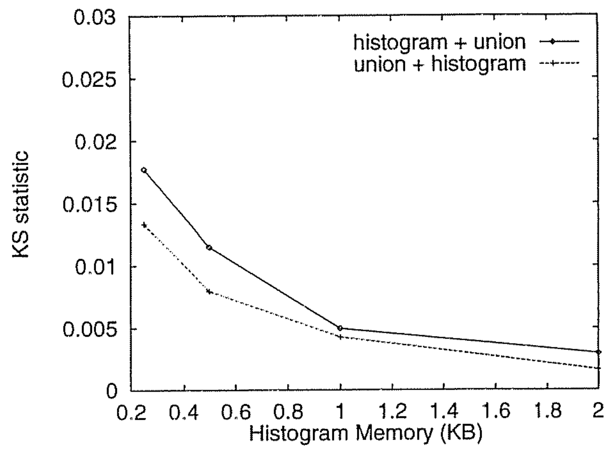

Figure 24: Error vs. histogram size

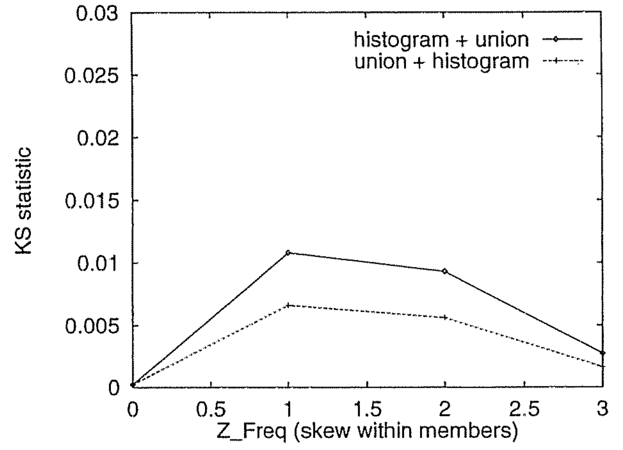

Figure 25: Error vs. intrasite data skew

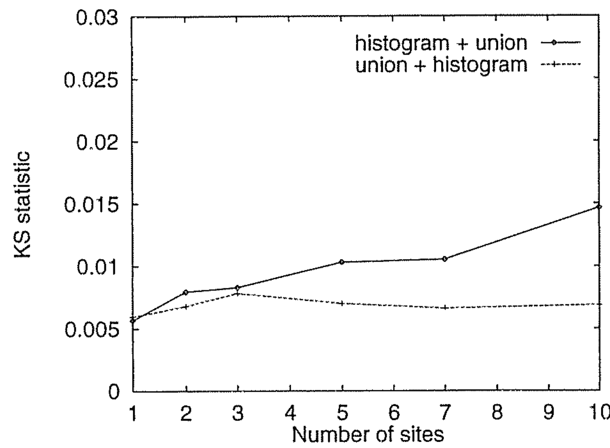

Figure 26: Error vs. number of sites

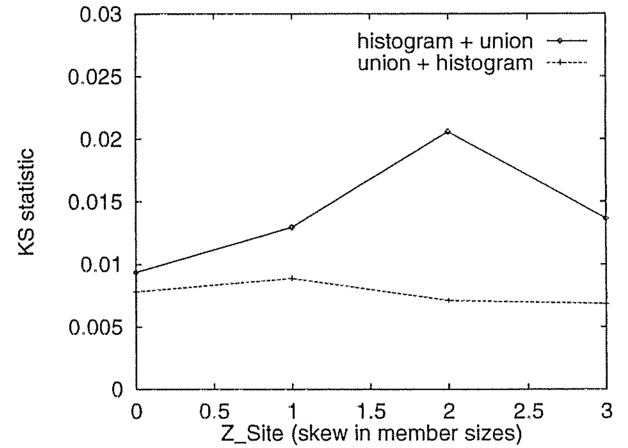

Figure 27: Error vs. skew in site size