Self-tuning Histograms: Building Histograms Without Looking at Data

Ashraf Aboulnaga*
Computer Sciences Department
University of Wisconsin - Madison
ashraf@cs.wisc.edu

Surajit Chaudhuri
Microsoft Research
surajitc@microsoft.com

Abstract

In this paper, we introduce self-tuning histograms. Although similar in structure to traditional histograms, these histograms infer data distributions not by examining the data or a sample thereof, but by using feedback from the query execution engine about the actual selectivity of range selection operators to progressively refine the histogram. Since the cost of building and maintaining self-tuning histograms is independent of the data size, self-tuning histograms provide a remarkably inexpensive way to construct histograms for large data sets with little up-front costs. Self-tuning histograms are particularly attractive as an alternative to multi-dimensional traditional histograms that capture dependencies between attributes but are prohibitively expensive to build and maintain. In this paper, we describe the techniques for initializing and refining self-tuning histograms. Our experimental results show that self-tuning histograms provide a low-cost alternative to traditional multi-dimensional histograms with little loss of accuracy for data distributions with low to moderate skew.

1. Introduction

Database systems require knowledge of the distribution of the data they store. This information is primarily used by query optimizers to estimate the selectivities of the operations involved in a query and choose the query execution plan. It could also be used for other purposes such as approximate query processing, load balancing in parallel database systems, and guiding the process of sampling from a relation. *Histograms* are widely used for capturing data distributions. They are used in most commercial database systems such as Microsoft SQL Server, Oracle, Informix, and DB2.

While histograms impose very little cost at query optimization time, the cost of building them and maintaining or rebuilding them when the data is modified has to be considered when we choose the attributes or attribute combinations for which we build histograms. Building a histogram involves scanning or sampling the data, and sorting the data and partitioning it into buckets, or finding quantiles. For large databases, the cost is significant enough to prevent us from building all the histograms that we believe are useful. This problem is particularly striking for *multi-dimensional* histograms that capture joint distributions of correlated attributes [MD88, PI97]. These histograms can be extremely useful for optimizing decision-support queries since they provide valuable information that helps in estimating the selectivities of multi-attribute predicates on correlated attributes. Despite their potential, to the best of our knowledge, no commercial database system supports multi-dimensional histograms. The usual alternative to multi-dimensional histograms is to assume that the attributes are independent, which enables using a combination of one-dimensional histograms. This approach is efficient but also *very* inaccurate. The inaccuracy results in a poor choice of execution plans by the query optimizer.

Self-tuning Histograms

In this paper, we explore a novel approach that helps reduce the cost of building and maintaining histograms for large tables. Our approach is to build histograms *not* by examining the data but by using feedback information about the execution of the *queries on the database (query workload).* We start with an initial histogram built with whatever information we have about the distribution of the histogram attribute(s). For example, we will construct an initial two-dimensional histogram from two existing one-dimensional histograms assuming independence of the attributes. As queries are issued on the database, the query optimizer uses the histogram to estimate selectivities in the process of choosing query execution plans. Whenever a plan is executed, the query execution engine can count the number of tuples produced by each operator. Our approach is to use this "free" feedback information to refine the histogram. Whenever a query uses the histogram, we compare the estimated selectivity to the actual selectivity and refine the histogram based on the selectivity estimation error. This incremental refinement progressively reduces estimation errors and leads to a histogram that is accurate for similar workloads. We call histograms built using this process *self-tuning histograms* or *ST-histograms* for short. This work was done in the broader context of the AutoAdmin project at Microsoft Research (http://research.microsoft.com/db/Autoadmin) that investigates techniques to make databases self-tuning.

ST-histograms make it possible to build higher dimensional histograms incrementally with little overhead, thus providing commercial systems with a low-cost approach to creating and maintaining such histograms. The ST-histograms have a low up-front cost because they are initialized without looking at the data. The refinement of ST-histograms is a simple low-cost procedure that leverages "free" information from the execution engine. Furthermore, we demonstrate that histogram refinement converges quickly. Thus, the overall cost of ST-histograms is much lower than that of traditional multi-dimensional histograms, yet the loss of accuracy is very acceptable for data with low to moderate skew in the joint distribution of the attributes.

* Work done while the author was at Microsoft Research

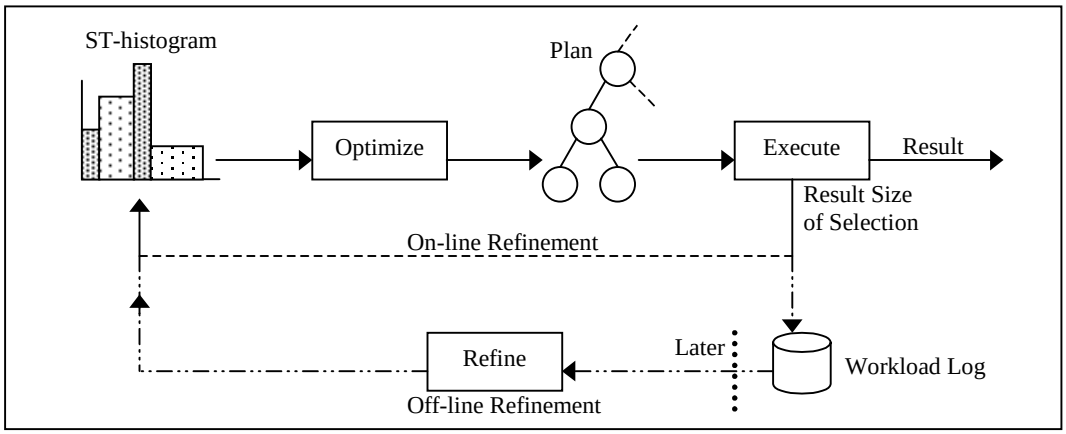

Figure 1: On-line and off-line refinement of ST-histograms

A ST-histogram can be refined *on-line* or *off-line* (Figure 1). In the on-line mode, the module executing a range selection immediately updates the histogram. In the off-line mode, the execution module writes every selection range and its result size to a *workload log*. Tools available with commercial database systems, e.g., Profiler in Microsoft SQL Server, can accomplish such logging. The workload log is used to refine the histogram in a batch at a later time. On-line refinement ensures that the histogram reflects the most up-to-date feedback information but it imposes more overhead during query execution than off-line refinement and can also cause the histogram to become a high-contention hot spot. The overhead imposed by histogram refinement, whether on-line or off-line, can easily be tailored. In particular, the histogram need not be refined in response to every single selection that uses it. We can choose to refine the histogram only for selections with a high selectivity estimation error. We can also skip refining the histogram during periods of high load or when there is contention for accessing it.

On-line refinement of ST-histograms brings a ST-histogram closer to the actual data distribution, whether the estimation error driving this refinement is due to the initial inaccuracy of the histogram or to modifications in the underlying relation. Thus, ST-histograms automatically adapt to database updates. Another advantage of ST-histograms is that their accuracy depends on how often they are used. The more a ST-histogram is used, the more it is refined, the more accurate it becomes.

Applications of Self-tuning Histograms

One can expect traditional histograms built by looking at the data to be more accurate than ST-histograms that "learn" the distribution without ever looking at the data. Nevertheless, ST-histograms, and especially multi-dimensional ST-histograms, are suitable for a wide range of applications.

As mentioned above, multi-dimensional ST-histograms are particularly attractive. Traditional multi-dimensional histograms, most notably MHIST-*p* histograms [PI97], are significantly more expensive than traditional one-dimensional histograms, increasing the value of the savings in cost offered by ST-histograms. Furthermore, ST-histograms are very competitive in terms of accuracy with MHIST-*p* histograms for data distributions with low to moderate skew (Section 5). Multi-dimensional ST-histograms can be initialized using traditional one-dimensional histograms and subsequently refined to provide a cheap and efficient way of capturing the joint distribution of multiple attributes. The other inexpensive alternative of assuming independence has been repeatedly demonstrated to be inaccurate (see, for example, [PI97] and our experiments in Section 5). Furthermore, note that building traditional histograms is an off-line process, meaning that histograms cannot be used until the system incurs the whole cost of completely building them. This is not true of ST-histograms. Finally, note that ST-histograms make it possible to inexpensively build not only two-dimensional, but also *n*-dimensional histograms.

ST-histograms are also a suitable alternative when there is not enough time for updating database statistics to allow building all the desired histograms in the traditional way. This may happen in data warehouses that are updated periodically with huge amounts of data. The sheer data size may prohibit rebuilding all the desired histograms during the batch window. This very same data size makes ST-histograms an attractive option, because examining the *workload* to build histograms will be cheaper than examining the data and can be tailored to a given time budget.

The technique of ST-histograms can be an integral part of database servers as we move towards self-tuning database systems. If a self-tuning database system decides that a histogram on some attribute or attribute combination may improve performance, it can start by building a ST-histogram. The low cost of ST-histograms allows the system to experiment more extensively and try out more histograms than if traditional histograms were the only choice. Subsequently, one can construct a traditional histogram only if the ST-histogram does not provide the required accuracy.

Finally, an intriguing possible application of ST-histograms will be for applications that involve queries on remote data sources. With recent trends in database usage, query optimizers will have to optimize queries involving remote data sources not under their direct control, e.g., queries involving data sources accessed over the Internet. Accessing the data and building traditional histograms for such data sources may not be easy or even possible. Query results, on the other hand, are available from the remote source, making the technique of ST-histograms an attractive option.

The rest of this paper is organized as follows. In Section 2 we present an overview of the related work. Section 3 describes one-dimensional ST-histograms and introduces the basic concepts that lead towards Section 4 where we describe multi-dimensional ST-histograms. Section 5 presents an experimental evaluation of our proposed techniques. Section 6 contains concluding remarks.

2. Related Work

Histograms were introduced in [Koo80], and most commercial database systems now use histograms for selectivity estimation.

Although one-dimensional equi-depth histograms are used in most commercial systems, more accurate histograms have been proposed recently [PIHS96]. [PI97] extends the techniques in [PIHS96] to multiple dimensions. However, we are unaware of any commercial systems that use the MHIST-*p* technique proposed in [PI97]. A novel approach for building histograms based on wavelets is presented in [MVW98].

A major disadvantage of histograms is the cost of building and maintaining them. Some recent work has addressed this shortcoming. [MRL98] proposes a one-pass algorithm for computing approximate quantiles that could be used to build approximate equi-depth histograms in one pass over the data. Reducing the cost of maintaining equi-depth and compressed histograms is the focus of [GMP97]. Recall that our approach is not to examine the data at all, but to build histograms using feedback from the query execution engine. However, our technique for refining ST-histograms shares commonalities with the split and merge algorithm proposed in [GMP97]. This relationship is further discussed in Section 3.

In addition to histograms, another technique for selectivity estimation is sampling the data at query optimization time [LNS90]. The main disadvantage of this approach is the overhead it adds to query optimization.

The concept of using feedback from the query execution engine to estimate data distributions is introduced in [CR94]. In this paper, the data distribution is represented as a linear combination of "model functions". Feedback information is used to adjust the weighting coefficients of this linear combination by a method called recursive-least-square-error. This paper only considers one-dimensional distributions. It remains an open problem whether one can find suitable multi-dimensional model functions, or whether the recursive least-square-error technique would work well for multi-dimensional distributions. In contrast, we show how our technique can be used to construct multi-dimensional histograms as well as one-dimensional histograms. Furthermore, our work is easily integrated into existing systems because we use the same histogram data structures that are currently supported in commercial systems.

A different type of feedback from the execution engine to the optimizer is proposed in [KD98]. In this paper, the execution engine invokes the query optimizer to re-optimize a query if it believes, based on statistics collected during execution, that this will result in a better query execution plan.

3. One-dimensional ST-histograms

Although the main focus of our paper is to demonstrate that ST-histograms are low cost alternatives to traditional multi-dimensional histograms, the fundamentals of ST-histograms are best introduced using ST-histograms for single attributes. Single-attribute ST-histograms are similar in structure to traditional histograms. Such a ST-histogram consists of a set of buckets. Each bucket, *b*, stores the *range* that it represents, [*low*(*b*), *high*(*b*)], and the number of tuples in this range, or the *frequency*, *freq*(*b*). Adjacent buckets share the bucket endpoints, and the ranges of all the buckets together cover the entire range of values of the histogram attribute. We assume that the refinement of ST-histograms is driven by feedback from range selection queries.

A ST-histogram assumes that the data is uniformly distributed until the feedback observation contradicts the uniformity assumption. Thus, the refinement/restructuring of ST-histograms corresponds to weakening the uniformity assumption as needed in response to feedback information. Therefore, the lifecycle of a ST-histogram consists of two stages. First, it is *initialized* and then, it is *refined*. The process of refinement can be broken down further into two parts: (a) *refining* individual *bucket frequencies*, and (b) *restructuring* the histogram, i.e., moving the bucket boundaries. The refinement process is driven by a query workload (see Section 1). The bucket frequencies are updated with every range selection on the histogram attribute, while the bucket boundaries are updated by *periodically restructuring* the histogram. We describe each of these steps in the rest of the section.

3.1 Initial Histogram

To build a ST-histogram, *h*, on an attribute, *a*, we need to know the required number of histogram buckets, *B*, the number of tuples in the relation, *T*, and the minimum and maximum values of attribute *a*, *min* and *max*. The *B* buckets of the initial histogram are evenly spaced between *min* and *max*. At the time of initializing the histogram structure, we have no feedback information. Therefore, we make the uniformity assumption and assign each of the buckets a frequency of *T*/*B* tuples (with some provision for rounding)

The parameter *T* can be looked up from system catalogs maintained for the database. However, the system may not store minimum and maximum values of attributes in its catalogs. The precise value of the minimum and maximum is not critical. Therefore, the initialization phase of ST-histograms can exploit additional sources to project an estimate that may subsequently be refined. For example, domain constraints on the column, as well as the minimum and maximum values referenced in the query workload can be used for such estimation.

3.2 Refining Bucket Frequencies

The bucket frequencies of a ST-histogram are refined (updated) with feedback information from the queries of the workload. For every selection on the histogram attribute, we compute the absolute estimation error, which is the difference between the estimated and actual result sizes. Based on this error, we refine the frequencies of the buckets that were used in estimation.

The key problem is to decide how to distribute the "blame" for the error among the histogram buckets that overlap the range of a given query. In a ST-histogram, error in estimation may be due to incorrect frequencies in any of the buckets that overlap the selection range. This is different from traditional histograms in which, if the histogram has been built using a full scan of data and has not been degraded in accuracy by database updates, the estimation error can result only from the first or last bucket, and only if they partially overlap the selection range. Buckets that are totally contained in the selection range do not contribute to the error.

The change in frequency of any bucket should depend on how much it contributes to the error. We use the heuristic that buckets with higher frequencies contribute more to the estimation error than buckets with lower frequencies. Specifically, we assign the "blame" for the error to the buckets used for estimation *in proportion to their current frequencies*. An alternative heuristic, not studied in this paper, is to assign the blame in proportion to the current ranges of the buckets.

Finally, we multiply the estimation error by a *damping factor* between 0 and 1 to make sure that bucket frequencies are not modified too much in response to errors, as this may lead to over-sensitive or unstable histograms.

Figure 2 presents the algorithm for updating the bucket frequencies of a ST-histogram, *h*, in response to a range selection, [*rangelow*,*rangehigh*], with actual result size *act*. This algorithm

algorithm UpdateFreq
Inputs: *h*, *rangelow*, *rangehigh*, *act*
Outputs: *h* with updated bucket frequencies
begin
1 Get the set of *k* buckets overlapping the selection range, $\{b_1, b_2, \ldots, b_k\}$;
2 *est* = Estimated result size of selection using histogram *h*;
3 *esterr* = *act* – *est* ; /* Compute the absolute estimation error. */
4 /* Distribute the error among the buckets in proportion to frequency. */
5 **for** *i* = 1 **to** *k* **do**
6 $frac = \frac{\min(rangehigh, high(b_i)) - \max(rangelow, low(b_i)) + 1}{high(b_i) - low(b_i) + 1}$;
7 $freq(b_i) = \max(freq(b_i) + \propto * esterr * frac * freq(b_i) / est, \ 0)$;
8 **endfor**
end UpdateFreq

Figure 2: Algorithm for updating bucket frequencies in one-dimensional ST-histograms

is used for both on-line and off-line refinement. The algorithm first determines the histogram buckets that overlap the selection range, whether they partially overlap the range or are totally contained in it, and the estimated result size. The query optimizer usually obtains this information during query optimization, so we can save some effort by retaining this information for subsequently refining bucket frequencies.

Next, the algorithm computes the absolute estimation error, denoted by *esterr* (line 3 in Figure 2). The error formula distinguishes between overestimation, indicated by a negative error and requiring the bucket frequencies to be lowered, and underestimation, indicated by a positive error and requiring the bucket frequencies to be raised. As mentioned earlier, the blame for this error is assigned to histogram buckets in proportion to the frequencies that they contribute to the result size. We assume that each bucket contains all possible values in the range that it represents, and we approximate all frequencies in a bucket by their average (i.e., we make the continuous values and uniform frequencies assumptions [PIHS96]). Under these assumptions, the contribution of a histogram bucket to the result size is equal to its frequency times the fraction of the bucket overlapping the selection range. This fraction is the length of the interval where the bucket overlaps the selection range divided by the length of the interval represented by the bucket (line 6). To distribute the error among buckets in proportion to frequency, each bucket is assigned a portion of the absolute estimation error, *esterr*, equal to its contribution to the result size, $frac * freq(b_i)$, divided by the total result size, *est*, damped by a damping factor, $\propto$ (line 7). We experimentally demonstrate in Section 5 that the refinement process is robust across a wide range of values for $\propto$, and we recommend using values of $\propto$ in the range 0.5 to 1.

3.3 Restructuring

Refining bucket frequencies is not enough to get an accurate histogram. The frequencies in a bucket are approximated by their average. If there is a large variation in frequency within a bucket, the average frequency is a poor approximation of the individual frequencies, no matter how accurate it is. Specifically, high frequency values will be contained in high frequency buckets, but they may be grouped with low frequency values in these buckets. Thus, in addition to refining the bucket frequencies, we must also restructure the buckets, i.e., move the bucket boundaries to get a better partitioning that avoids grouping high frequency and low frequency values in the same buckets. Ideally, we would like to make high frequency buckets as narrow as possible. In the limit, this approach separates out high frequency values in singleton buckets of their own, a common objective for histograms (e.g., see [PIHS96]). Therefore, we choose buckets that currently have high frequency and *split* them into several buckets. Splitting induces the separation of high frequency and low frequency values into different buckets, and the frequency refinement process later adjusts the frequencies of these new buckets. In order to ensure that the number of buckets assigned to the ST-histogram does not increase due to splitting, we need a mechanism to reclaim buckets as well. To that end, we use a step of *merging* that groups a run of consecutive buckets with similar frequencies into one bucket. Thus, our approach is to *restructure* the histogram periodically by merging buckets and using the buckets thus freed to split high frequency buckets. Restructuring may be triggered using a variety of heuristics. In this paper, we study the simplest scheme where the restructuring process is invoked after every *R* selections that use the histogram. The parameter *R* is called the *restructuring interval*.

To merge buckets with similar frequencies, we first have to decide how to quantify "similar frequencies". We assume that two bucket frequencies are similar if the difference between them is less than *m* percent of the number of tuples in the relation, *T*. *m* is a parameter that we call the *merge threshold*. In most of our experiments, $m \leq 1\%$ was a suitable choice. We use a greedy strategy to form a run of adjacent buckets with similar frequencies and collapse them into a single bucket. We repeat this step until no further merging is possible that satisfies the merge threshold condition (Steps 2–9 in Figure 3).

We also need to decide which "high frequency" buckets to split. We choose to split the *s* percent of the buckets with the highest frequencies. *s* is a parameter that we call the *split threshold*. In our experiments, we used s=10%. Our heuristic distributes the reclaimed buckets among the high frequency buckets in proportion to frequency. The higher the frequency of a bucket, the more extra buckets it gets.

Figure 3 presents the algorithm for restructuring a ST-histogram, *h*, of *B* buckets on a relation with *T* tuples. The first step in histogram restructuring is greedily finding runs of consecutive buckets with similar frequencies to merge. The algorithm repeatedly finds the pair of adjacent runs of buckets such that the maximum difference in frequency between a bucket in the first run and a bucket in the second run is the minimum

```
algorithm RestructureHist
Inputs:   h
Outputs:  restructured h
begin
1   /* Find buckets with similar frequencies to merge. */
2   Initialize B runs of buckets such that each run contains one histogram bucket;
3   For every two consecutive runs of buckets, find the maximum difference in frequency between a bucket in the
4       first run and a bucket in the second run;
5   Find the minimum of all these maximum differences, mindiff;
6   if mindiff ≤ m * T then
7       Merge the two runs of buckets corresponding to mindiff into one run;
8       Look for other runs to merge.  Goto line 3;
9   endif
10
11  /* Assign the extra buckets freed by merging to the high frequency buckets. */
12  k = s * B;
13  Find the set, {b_1, b_2, ..., b_k} of buckets with the k highest frequencies that were not chosen to be
14      merged with other buckets in the merging step;
15  Assign the buckets freed by merging to the buckets of this set in proportion to their frequencies;
16
17  /* Construct the restructured histogram by merging and splitting. */
18  Merge each previously formed run of buckets into one bucket spanning the range represented by all the buckets
19      in the run and having a frequency equal to the sum of their frequencies;
20  Split the k buckets chosen for splitting, giving each one the number of extra buckets assigned to it earlier.
21      The new buckets are evenly spaced in the range spanned by the old bucket and the frequency of the old
22      bucket is equally distributed among them;
end RestructureHist
```

Figure 3: Algorithm for restructuring one-dimensional ST-histograms

over all pairs of adjacent runs. The two runs are merged into one if this difference is less than the threshold $m*T$, and we stop looking for runs to merge if it is not. This process results in a number of runs of several consecutive buckets. Each run is replaced with one bucket spanning its entire range, and with a frequency equal to the total frequency of all the buckets in the run. This frees a number of buckets to allocate to high frequency buckets during splitting.

Splitting starts by identifying the s percent of the buckets that have the highest frequencies and are not singleton buckets. We avoid splitting buckets that have been chosen for merging since their selection indicates that they have similar frequencies to their neighbors. The extra buckets freed by merging are distributed among the buckets being split in proportion to their frequencies. A bucket being split, b_i, gets $freq(b_i)$ / *totalfreq* of the extra buckets, where *totalfreq* is the total frequency of the buckets being split. To split a bucket, it is replaced with itself plus the extra buckets assigned to it. These new buckets evenly divide the range of the old bucket, and the frequency of the old bucket is evenly distributed among them.

Splitting and merging are used in [GMP97] to redistribute histogram buckets in the context of maintaining approximate equi-depth and compressed histograms. The algorithm in [GMP97] merges pairs of buckets whose total frequency is less than a threshold, whereas our algorithm merges *runs* of buckets based on the differences in their frequency. Our algorithm assigns the freed buckets to the buckets being split in proportion to the frequencies of the latter, whereas the algorithm in [GMP97] merges only one pair of buckets at a time and can, thus, split only one bucket into two. A key difference between the two approaches is that in [GMP97], a sample of the tuples of the relation is continuously maintained (the "backing sample"), and buckets are split at their approximate medians computed from this sample. On the other hand, our approach does not examine the data at any point, so we do not have information similar to that represented in the backing sample of [GMP97]. Hence, our restructuring algorithm splits buckets at evenly spaced intervals, without using any information about the data distribution within a bucket.

Figure 4 gives an example of histogram restructuring. In this example, the merge threshold is such that algorithm RestructureHist merges buckets if the difference between their frequencies is within 3. The algorithm identifies two runs of buckets to be merged, buckets 1 and 2, and buckets 4 to 6. Merging these runs frees three buckets to assign to high frequency buckets. The split threshold is such that we split the two buckets with the highest frequencies, buckets 8 and 10. Assigning the extra buckets to these two buckets in proportion to frequency means that bucket 8 gets two extra buckets and bucket 10 gets one extra bucket.

Splitting may unnecessarily separate values with similar, low frequencies into different buckets. Such runs of buckets with similar low frequencies would be merged during subsequent restructuring. Notice that splitting distorts the frequency of a bucket by distributing it among the new buckets. This means that the histogram may lose some of its accuracy by restructuring. This accuracy is restored when the bucket frequencies are refined through subsequent feedback.

In summary, our model is as follows: The frequency refinement process is applied to the histogram, and the refined frequency information is periodically used to restructure the histogram. Restructuring may reduce accuracy by distributing frequencies among buckets during splitting but frequency refinement restores, and hopefully increases, histogram accuracy.

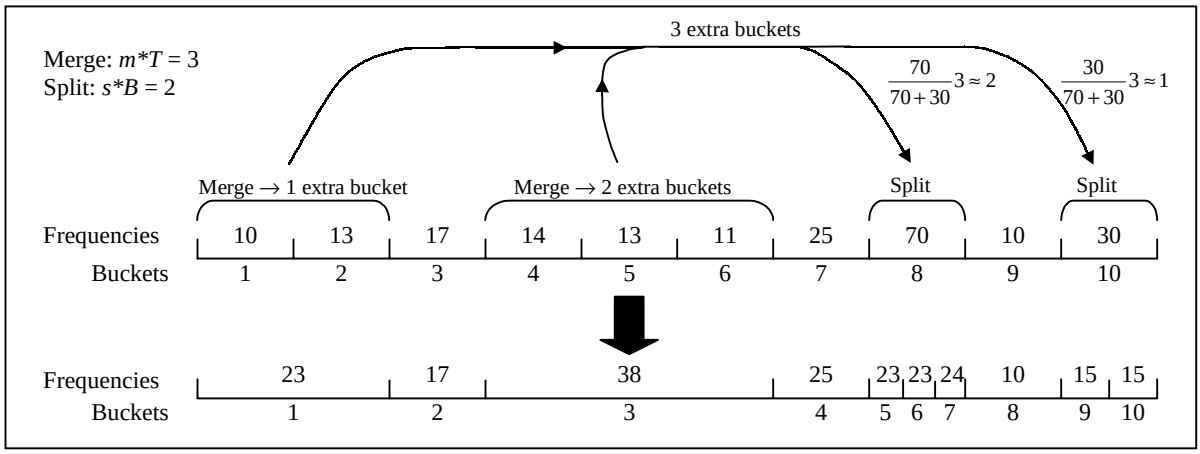

Figure 4: Example of histogram restructuring

4. Multi-dimensional ST-histograms

In this section, we present multi-dimensional (i.e., multi-attribute) ST-histograms. Our goal is to build histograms representing the joint distribution of multiple attributes of a single relation. These histograms will be used to estimate the result size of conjunctive range selections on these attributes, and are refined based on feedback from these selections. Using accurate one-dimensional histograms for all the attributes is not enough, because they do not reflect the correlation between attributes. In this section, we discuss the special considerations for multi-dimensional histograms.

Working in multiple dimensions raises the issue of how to partition the multi-dimensional space into histogram buckets. The effectiveness of ST-histograms stems from their ability to pinpoint the buckets contributing to the estimation error and "learn" the data distribution. The partitioning we choose must efficiently support this learning process. It must also be a partitioning that is easy to construct and maintain, because we want the cost of ST-histograms to remain as low as possible. To achieve these objectives, we use a *grid partitioning* of the multi-dimensional space. Each dimension of the space is partitioned into a number of partitions. The partitions of a dimension may vary in size, but the partitioning of the space is always fully described by the partitioning of the dimensions.

We choose a grid partitioning due to its simplicity and low cost, even though it does not offer as much flexibility in grouping values into buckets as other partitionings such as, for example, the MHIST-*p* histogram partitioning [PI97]. The simplicity of a grid partitioning allows our histograms to have more buckets for a given amount of memory. It is easier for ST-histograms to infer the data distribution from feedback information when working with a simple high-resolution representation of the distribution than it is when working with a complex low-resolution representation. Furthermore, we doubt that the simple feedback information used for refinement can be used to glean enough information about the data distribution to justify a more complex partitioning.

Each dimension, *i*, of an *n*-dimensional ST-histogram is partitioned into B_i partitions. B_i does not necessarily equal B_j for $i \neq j$. The partitioning of the space is described by *n* arrays, one per dimension, which we call the *scales* [NHS84]. Each array element of the scales represents the range of one partition, [*low*,*high*]. In addition to the scales, a multi-dimensional ST-histogram has an *n*-dimensional matrix representing the grid cell frequencies, which we call the *frequency matrix*. Figure 5 presents an example of a 5×5 two-dimensional ST-histogram and a range selection that uses it.

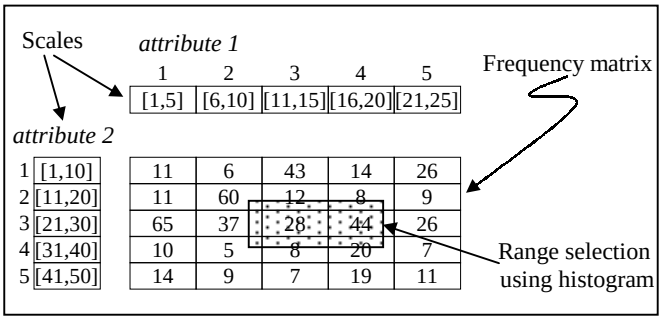

Figure 5: A 2d ST-histogram and a range selection using it

4.1 Initial Histogram

To build a ST-histogram on attributes, a_1, a_2,…, a_n, we can assume complete uniformity and independence, or we can use existing one-dimensional histograms but assume independence of the attributes as the starting point.

If we start with the uniformity and independence assumption, we need to know the minimum and maximum values of each attribute a_i, min_i and max_i. We also need to specify the number of partitions for each dimension, B_1, B_2,…, B_n. Then, each dimension, *i*, is partitioned into B_i equally spaced partitions, and the *T* tuples of the relation are evenly distributed among all the buckets of the frequency matrix. This technique is an extension of one-dimensional ST-histograms.

Another way of building multi-dimensional ST-histograms is to start with traditional one-dimensional histograms on all the multi-dimensional histogram attributes. Such one-dimensional histograms, if they are available, provide a better starting point than assuming uniformity and independence. In this case, we initialize the scales by partitioning the space along the bucket boundaries of the one-dimensional histograms, and we initialize the frequency matrix using the bucket frequencies of the one-dimensional histograms and assuming that the attributes are independent. Under the independence assumption, the initial frequency of a cell of the frequency matrix is given by $freq[j_1, j_2, \ldots, j_n] = \frac{1}{T^{n-1}} \prod_{i=1}^{n} freq_i[j_i]$, where $freq_i[j_i]$ is the frequency of bucket j_i of the histogram for dimension *i*.

4.2 Refining Bucket Frequencies

The algorithm for refining bucket frequencies in the multi-dimensional case is identical to the one-dimensional algorithm presented in Figure 2, except for two differences. First, finding the histogram buckets that overlap a selection range (line 1 in Figure 2) now requires examining a multi-dimensional structure. Second, a bucket is now a multi-dimensional cell in the frequency matrix, so the fraction of a bucket overlapping the selection range (line 6) is equal to the *volume of the region* where the bucket overlaps the selection range divided by volume of the region represented by the whole bucket (Figure 5).

4.3 Restructuring

Periodic restructuring is needed only for multi-dimensional ST-histograms initialized assuming uniformity and independence. ST-histograms initialized using traditional one-dimensional histograms do not need to be periodically restructured, assuming that the one-dimensional histograms are accurate. This is based on the assumption that the partitioning of an accurate traditional one-dimensional histogram built by looking at the data is more accurate when used for multi-dimensional ST-histograms than a partitioning built by splitting and merging.

As in the one-dimensional case, restructuring in the multi-dimensional case is based on merging buckets with similar frequencies and splitting high frequency buckets. The required parameters are also the same, namely the restructuring interval, *R*, the merge threshold, *m*, and the split threshold, *s*. Restructuring changes the partitioning of the multi-dimensional space *one dimension at a time*. The dimensions are processed in any order, and the partition boundaries of each dimension are modified independent of other dimensions. The algorithm for restructuring one dimension of the multi-dimensional ST-histogram is similar to the algorithm in Figure 3. However, merging and splitting in multiple dimensions present some additional problems.

For an *n*-dimensional ST-histogram, every partition of the scales in any dimension identifies an (*n*-1)-dimensional "slice" of the grid (e.g., a row or a column in a two-dimensional histogram). Thus, merging two partitions of the scales requires merging two slices of the frequency matrix, each containing several buckets. Every bucket from the first slice is merged with the corresponding bucket from the second slice. To decide whether or not to merge two slices, we find the maximum difference in frequency between any two corresponding buckets that would be merged if these two slices are merged. We merge the two slices only if this difference is within $m*T$ tuples. We use this method to identify *runs* of partitions to merge.

The high frequency partitions of any dimension are split by assigning them the extra partitions freed by merging *in the same dimension*. Thus, restructuring does not change the number of partitions in a dimension. To decide which partitions to split in any dimension and how many extra partitions each one gets we use the *marginal frequency distribution* along this dimension. The marginal frequency of a partition is the total frequency of all buckets in the slice of the frequency matrix that it identifies. Thus, the marginal frequency of partition j_i in dimension i is given by $f_i(j_i) = \sum_{j_1=1}^{B_1} \cdots \sum_{j_{i-1}=1}^{B_{i-1}} \sum_{j_{i+1}=1}^{B_{i+1}} \cdots \sum_{j_n=1}^{B_n} freq[j_1, j_2, \ldots, j_n]$. As in the one-dimensional case, we split the *s* percent of the partitions in any dimension with the highest marginal frequencies, and we assign them the extra partitions in proportion to their current marginal frequencies.

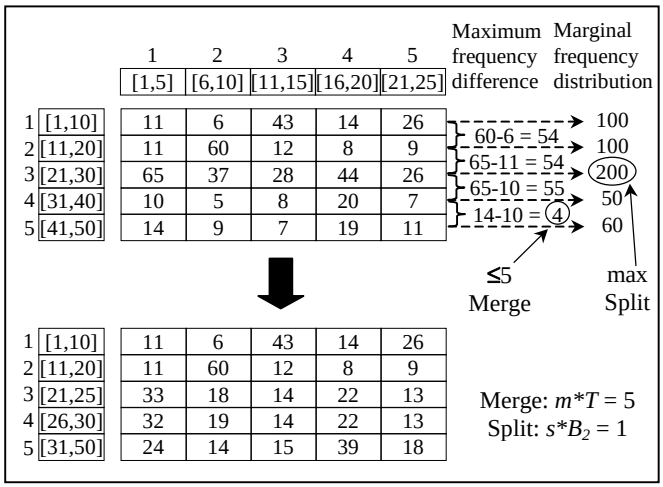

Figure 6: Restructuring the vertical dimension

Figure 6 demonstrates restructuring the histogram in Figure 5 along the vertical dimension (attribute 2). In this example, the merge threshold is such that we merge two partitions if the maximum difference in frequency between buckets in their slices that would be merged is within 5. This condition leads us to merge partitions 4 and 5. The split threshold is such that we split one partition along the vertical dimension. We compute the marginal frequency distribution along the vertical dimension and identify the partition with the maximum marginal frequency, partition 3. Merging and splitting (with some provisions for rounding) result in the shown histogram.

5. Experimental Evaluation

In this section, we present an experimental evaluation of our techniques using synthetic data sets and workloads. We investigate the accuracy and efficiency of one and multi-dimensional ST-histograms. In particular, we are interested in the accuracy of ST-histograms for data distributions with *varying degrees of skew*, and for *workloads with different access patterns*. We examine whether histogram refinement *converges* to an accurate state, or whether it oscillates in response to refinement. Another important consideration is how well ST-histograms *adapt to database updates*, and how efficiently they use the available memory. Due to space limitations, we present only a subset of the experiments conducted.

5.1 Setup for Experiments

5.1.1 Data Sets

We present the results of experiments using one to three-dimensional integer data sets. The results for higher dimensional data sets are similar. The one-dimensional data sets have 100K tuples and the multi-dimensional data sets have 500K tuples. Each dimension in a data set has *V* distinct values drawn randomly from a domain ranging from 1 to 1000. *V* = 200, 100, and 10, for 1, 2, and 3 dimensions, respectively. For multi-dimensional data sets, the number of distinct values and the domains of all dimensions are identical, and the value sets of all dimensions are generated independently. Frequencies are generated according to the Zipfian distribution [Zip49] with parameter *z* = 0, 0.5, 1, 2, and 3. *z* controls the skew of the distribution, with *z*=0 representing a uniform distribution (no skew). For one-dimensional data sets, the frequencies are assigned at random to the values. For multi-dimensional data

sets, the frequencies are assigned at random to *combinations* of values using the technique proposed in [PI97], namely assigning the generated frequencies to randomly chosen cells in the joint frequency distribution matrix.

5.1.2 Query Workloads

We use workloads consisting of random range selection queries in one or more dimensions. Each workload consists of 2000 independent selection queries. Most experiments use *random workloads*, in which the corner points of each selection range are independently generated from a uniform distribution over the entire domain. Some experiments use *workloads with locality of reference*. The attribute values used for selection range corner points in these workloads are generated from *piecewise uniform* distributions in which there is an 80% probability of choosing a value from a *locality range* that is 20% of the domain. The locality ranges for the different dimensions are independently chosen at random according to a uniform distribution.

5.1.3 Histograms

Unless otherwise stated, we use 100, 50, and 15 buckets per dimension for 1, 2, and 3 dimensional ST-histograms, respectively. For multi-dimensional ST-histograms, we use the same number of buckets in all dimensions, resulting in two and three-dimensional histograms with a total of 2500 and 3375 buckets. The one, two, and three-dimensional ST-histograms occupy 1.2, 10.5, and 13.5 kilobytes of memory, respectively. Our traditional histograms of choice are MaxDiff(V,A) histograms for one dimension, and MHIST-2 MaxDiff(V,A) histograms for multiple dimensions. These histograms were recommended in [PIHS96] and [PI97] for their accuracy and ease of construction. We compare the accuracy of ST-histograms to traditional histograms of these types occupying the same amount of memory.

We consider a wider range of memory allocation than most previous works (e.g., [PIHS96], [PI97], and [MVW98]) because of current trends in memory technology. We also demonstrate that our techniques are effective across a wide range of available memory (Section 5.7).

Note that the cost of building and maintaining traditional histograms is a function of the size of the relation (or the size of the sample used to build the histogram). In contrast, the cost of ST-histograms is independent of the data size and depends on the size of the query workload used for refinement.

5.1.4 Refinement Parameters

Unless otherwise stated, the parameters we use for restructuring the histogram (Section 3.3) are a restructuring interval, R=200 queries, a merge threshold, m=0.025%, and a split threshold, s=10%. For frequency refinement (Section 3.2), we use a damping factor, $\propto$=0.5 for one dimension, and $\propto$=1 for multiple dimensions.

5.1.5 Measuring Histogram Accuracy

We use the *relative estimation error* (abs(actual result size - estimated result size) / actual result size) to measure the accuracy of query result size estimation. To measure accuracy over an entire workload, we use the *average relative estimation error* for all queries in the workload, ignoring queries whose actual result size is zero.

One important question is with respect to which workload should we measure the accuracy of a ST-histogram. Recall that the premise of ST-histograms is that they are able to adapt to feedback from query execution. Therefore, for our evaluation we generate workloads that are statistically similar, but not the same as the training workload.

Unless otherwise stated, our experiments use *off-line* histogram refinement. Our steps for verifying the effectiveness of ST-histograms for some particular data set are:

1. Initialize a ST-histogram for the data set.
2. Issue the query workload that will be used to refine the histogram and generate a workload log. We call this the *refinement workload*.
3. Refine the histogram off-line based on the generated workload log.
4. After refinement, issue the refinement workload again and compute the estimation error. Verify that the error after refinement is less than the error before refinement.
5. Issue a different workload in which the queries have the same distribution as the workload used for refinement. We call this the *test workload*. We cannot expect the workload issued before refinement to be repeated exactly after refinement, but we can reasonably expect a workload with similar statistical characteristics. The ultimate test of accuracy is whether the ST-histogram performs well on the test workload.

5.2 Accuracy of One-dimensional ST-histograms

In this section, we experimentally study the effectiveness of one-dimensional ST-histograms for a wide range of data skew (z) using random workloads and the procedure outlined in Section 5.1.5. We demonstrate that ST-histograms are always better than assuming uniformity, and that they are competitive with MaxDiff(V,A) histograms in terms of accuracy except for highly skewed data sets.

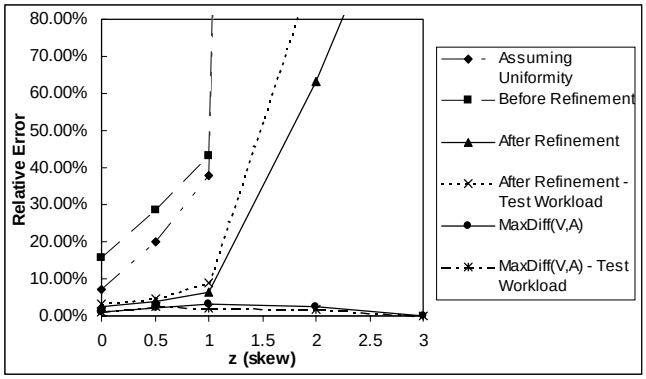

Figure 7: One-dimensional data, random workload

Figure 7 presents the estimation errors for a random refinement workload on one-dimensional data sets with varying z. For each data set, the figure presents the estimation error for the random refinement workload assuming a uniform distribution and using the initial ST-histogram constructed assuming uniformity. The estimation errors in these two cases are different due to rounding errors during histogram initialization. The figure also presents the average relative estimation error for the random refinement workload using the refined ST-histogram when this workload is issued again after it is used for refinement. It also presents the error for a statistically similar test workload using the refined ST-histogram. Finally, the figure presents the estimation errors for the refinement and test workloads using a traditional MaxDiff(V,A) histogram occupying the same amount of memory as the ST-histogram.

Histogram refinement results in a significant reduction in estimation error for all values of *z*. This reduced error is observed for both the refinement workload and the test workload indicating a true improvement in histogram quality. Thus, ST-histograms are always better than assuming uniformity. The MaxDiff(V,A) histograms are more accurate than the ST-histograms. This is expected because MaxDiff(V,A) histograms are built based on the true distribution determined by examining the data. However, for low values of *z*, the estimation errors using refined ST-histograms are very close to the errors using MaxDiff(V,A) histograms, and are small enough for query optimization purposes.

MaxDiff(V,A) histograms are considerably more accurate than ST-histograms only for highly skewed data sets ($z \geq 2$). This is expected because as *z* increases, the data distribution becomes more difficult to capture using simple feedback information. At the same time, the benefit of MaxDiff(V,A) histograms is maximum for highly skewed distributions [PIHS96].

5.3 Accuracy of Multi-Dimensional ST-histograms

In this section, we show that multi-dimensional ST-histograms initialized using traditional one-dimensional histograms are much more accurate than assuming independence. We also compare the performance of such ST-histograms and MHIST-2 histograms. In particular, we demonstrate that these ST-histograms are more accurate than MHIST-2 histograms for low to moderate values of *z* (i.e., low correlation). This is an important result because it indicates that ST-histograms are better than MHIST-2 histograms in both cost and accuracy for data distributions with low to medium correlation. For this paper, we only present the results of our experiments with ST-histograms initialized using traditional histograms. Experiments with the less accurate ST-histograms initialized assuming uniformity and independence have similar results.

Figures 8 and 9 present the results of using multi-dimensional ST-histograms initialized using MaxDiff(V,A) histograms and assuming independence for random workloads on two and three-dimensional data set with varying *z*. The information presented is the same as in Figure 7, except that we do not show the estimation error assuming uniformity because one would never assume uniformity when one-dimensional histograms are available, and we compare the performance of the ST-histograms against multi-dimensional MHIST-2 histograms instead of one-dimensional MaxDiff(V,A) histograms. Since the ST-histograms are initialized using MaxDiff(V,A) histograms, using them before refinement is the same as using the one-dimensional histograms and assuming independence.

The refined ST-histograms are more accurate than assuming independence, and the benefit of using them (i.e., the reduction in error) increases as *z* increases. ST-histograms are not as accurate as MHIST-2 histograms for high *z*, especially in three dimensions. This indicates that inferring joint data distributions based on simple feedback information becomes increasingly difficult with increasing dimensionality. As expected, MHIST-2 histograms are very accurate for high *z* [PI97], but we must bear in mind that the cost of building multi-dimensional MHIST-2 histograms is much more than the cost of building one-dimensional MaxDiff(V,A) histograms. Furthermore, this cost increases with increasing dimensionality.

Notice, though, that ST-histograms are more accurate than MHIST-2 histograms for low *z*. This is because MHIST-2 histograms use a complex partitioning of the space (as compared to ST-histograms). Representing this complex partitioning

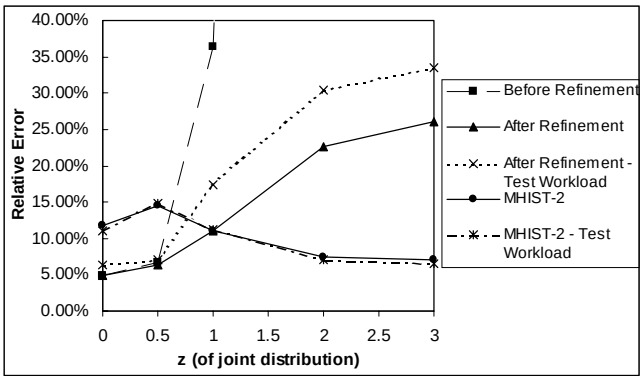

Figure 8: Two-dimensions, starting with MaxDiff(V,A)

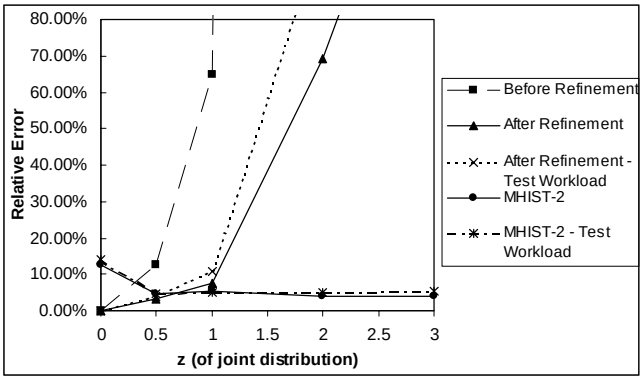

Figure 9: Three-dimensions, starting with MaxDiff(V,A)

requires MHIST-2 histograms to have complex buckets that consume more memory than ST-histogram buckets. Consequently, ST-histograms have more buckets than MHIST-2 histograms occupying the same amount of memory. For low *z*, the complex partitioning of MHIST-2 histograms does not increase accuracy because the joint distribution is close to uniform so any partitioning is fine. On the other hand, the large number of buckets in ST-histograms allows them to represent the distribution at a finer granularity leading to higher accuracy. This result demonstrates the value of multi-dimensional ST-histograms for database systems. For data with low to moderate skew, ST-histograms provide an effective way of capturing dependencies between attributes at a low cost.

	Equi-width		Equi-depth		MaxDiff(V,A)	
z	Before	After	Before	After	Before	After
0	4.27%	5.47%	6.41%	6.65%	4.93%	4.95%
0.5	6.77%	5.84%	8.67%	8.21%	6.64%	6.35%
1	37.61%	11.64%	39.94%	12.61%	36.37%	11.08%
2	562.36%	518.33%	615.06%	78.36%	435.54%	22.57%
3	530.71%	233.32%	383.76%	48.26%	460.71%	26.07%

Table 1: Starting with different types of 1d histograms

Table 1 presents the estimation errors for random workloads on two-dimensional data sets with varying *z* using ST-histograms built starting with traditional one-dimensional histograms. The errors are shown before refinement and after off-line refinement using the same random workloads. All one-dimensional histograms have 50 buckets. In addition to MaxDiff(V,A) histograms, the table presents the errors when we start with equi-width histograms, which are the simplest type of histograms, and

when we start with equi-depth histograms, which are currently used by many commercial database systems. The table shows that ST-histograms are equally effective for all three types of one-dimensional histograms.

5.4 Effect of Locality of Reference in the Query Workload

An interesting issue is studying the performance of ST-histograms on workloads with locality of reference in accessing the data. Locality of reference is a fundamental concept underlying all database accesses, so one would expect real life workloads to have such locality. Moreover, purely random workloads provide feedback information about the entire distribution, while workloads with locality of reference provide most of their feedback about a small part of the distribution. We would like to know how effective this type of feedback is for histogram refinement. In this section, we demonstrate that ST-histograms perform well for workloads with locality of reference. We also demonstrate that histogram refinement adapts to changes in the locality range of the workload.

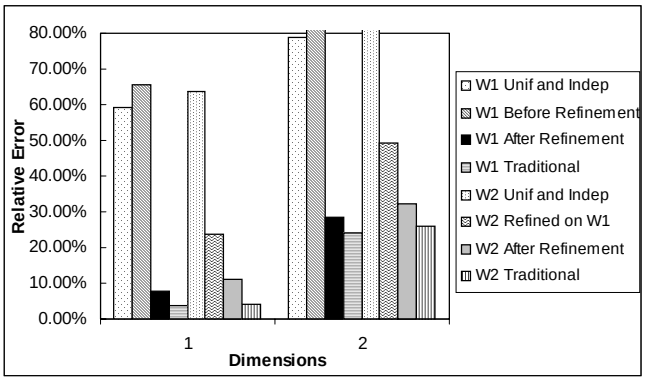

Figure 10: Workloads with locality of reference, z=1

Figure 10 presents the estimation errors for workloads with an 80%-20% locality for one and two-dimensional data sets with z=1. The first four bars for each data set present the errors for a workload, *W1*. The first two bars respectively show the errors assuming uniformity and independence, and using an initial ST-histogram representing the uniformity and independence assumption. The bars are not identical because of rounding errors. The third bar shows the error using the ST-histogram when issuing *W1* again after it is used for refinement. The fourth bar shows the error for *W1* using a traditional histogram. It is clear that refinement considerably improves estimation accuracy, making the ST-histogram almost as accurate as the traditional histogram. This improvement is also observed on test workloads that are statistically similar to *W1*. Next, we keep the refined histogram and change the locality of reference of the workload. We issue a new workload, *W2*, with a different locality range. The next four bars in Figure 10 present the estimation errors for *W2*. First, we issue *W2* and use the ST-histogram refined on *W1* for result size estimation (sixth bar). This histogram is not as accurate for *W2* as it was for *W1*, but it is better than assuming uniformity and independence. This means that refinement was still able to infer some information from the 20% of the queries of *W1* that lie outside the locality range. When we refine the histogram on *W2* and issue it again, we see that the ST-histogram becomes as accurate for *W2* as it was for *W1* after refinement. This improvement is also seen for workloads that are statistically similar to *W2*.

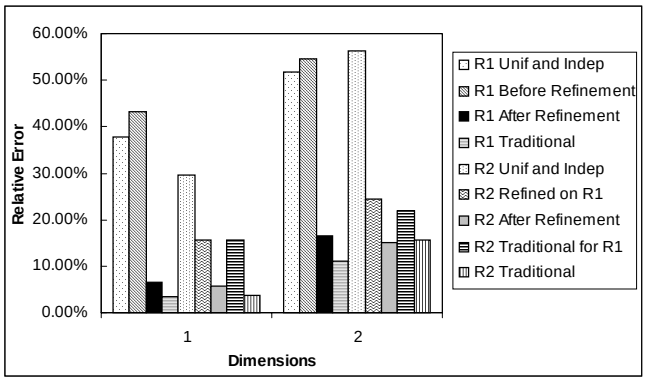

Figure 11: Adapting to database updates, z=1

5.5 Adapting to Database Updates

The results of this section demonstrate that although ST-histograms do not examine data, the feedback mechanism enables these histograms to adapt to updates in the underlying relation. Figure 11 presents the estimation errors for one and two-dimensional data sets with z=1 using random workloads. The first four bars present the estimation errors for the original relation before update, which we denote by *R1*. We update the relation by deleting a random 25% of its tuples and inserting an equal number of tuples following a Zipfian distribution with z=1. We denote this updated relation by *R2*. We retain the traditional and ST-histograms built for *R1* and re-issue the same random workload on *R2*. The fifth and sixth bars in Figure 11 are the estimation error for this workload on *R2* assuming uniformity and independence, and using the ST-histogram that was refined for *R1*, respectively. The histogram is not as accurate as it was for *R1*, which is expected, but it is still more accurate than assuming uniformity and independence. The seventh bar shows the error using the ST-histogram for *R2* after refinement using the same workload. Refinement restores the accuracy of the ST-histogram and adapts it to the updates in the relation. We also observe this improvement in error for statistically similar test workloads. The last two bars in Figure 11 present the estimation error for the random workload issued on *R2* using the traditional histograms for *R1* and *R2*, respectively. As expected, updating the relation reduces histogram accuracy, and rebuilding the histogram restores this accuracy.

5.6 Refinement Parameters

In this section, we investigate the effect of the refinement parameters: R, m, and s for restructuring and $\propto$ for updating bucket frequencies.

Table 2 presents the average relative estimation errors for random test workloads using ST-histograms that have been refined off-line using *other* random refinement workloads for one to three-dimensional data sets with varying z. For each data set, the error is presented if the histogram is not restructured during refinement, and if it is restructured with R=200, m=0.025%, and s=10%. Restructuring has no benefit for low z, but as z increases the need for restructuring becomes evident. Thus, restructuring extends the range of data skew for which ST-histograms are effective.

Figure 12 presents the estimation errors for random workloads and workloads with locality of reference on one to three-dimensional data sets with z=1 using ST-histograms that have been refined off-line using *other* statistically similar refinement workloads for $\propto$=0.01 to 1. The estimation errors are relatively

Dims	1		2		3	
z	No Restruct.	Restruct.	No Restruct.	Restruct.	No Restruct.	Restruct.
0	3.34%	3.05%	10.43%	10.78%	38.60%	38.86%
0.5	4.44%	4.54%	10.65%	10.62%	40.55%	38.64%
1	9.39%	8.94%	22.03%	21.41%	62.02%	51.45%
2	130.52%	95.09%	318.08%	77.22%	1098.09%	583.48%
3	306.79%	271.75%	327.39%	109.67%	14982.1%	4500.9%

Table 2: Error with and without restructuring

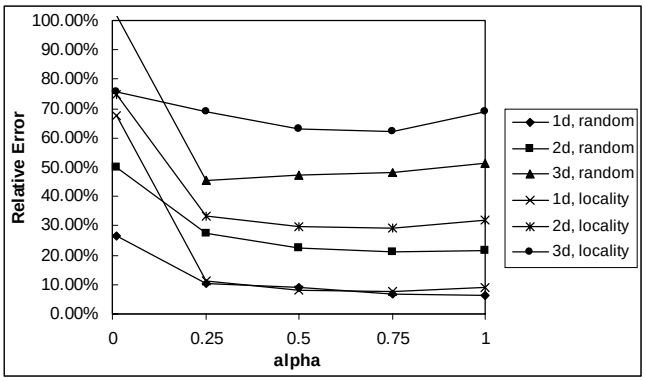

Figure 12: Effect of ∝, *z*=1

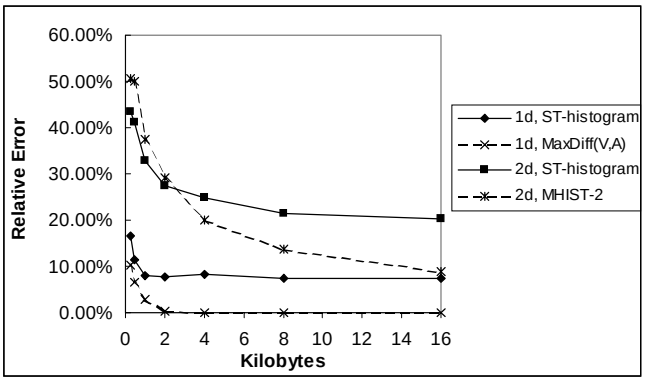

Figure 13: Effect of available memory

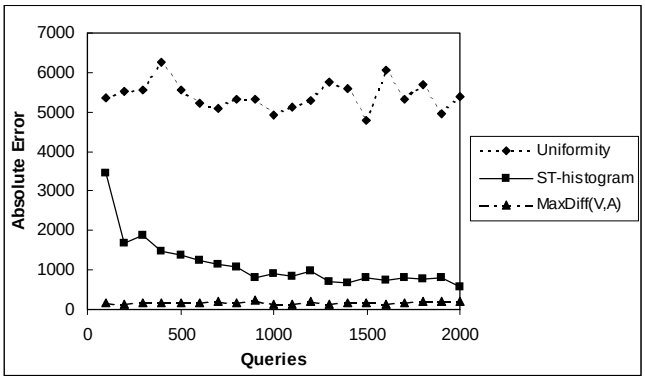

Figure 14: On-line refinement, one dimension, *z* =1

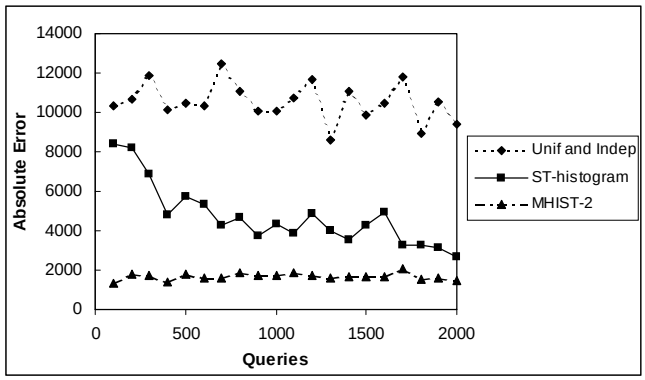

Figure 15: On-line refinement, two dimensions, *z* =1

flat for a wide range of ∝. Thus, the benefit of trying to find the optimal ∝ is low. We recommend using a fixed ∝ or varying it according to a simple algorithm. For our experiments we use ∝=0.5 for the one-dimensional case and ∝=1 for the multi-dimensional case.

5.7 Effect of Available Memory

Figure 13 presents the estimation errors for random test workloads on data sets with *z*=1 in one and two-dimensions using ST-histograms that have been refined on *other* random refinement workloads, and using traditional histograms occupying the same amount of memory as the ST-histograms. The errors are presented for histograms using 0.25 to 16 kilobytes of memory. ST-histograms are accurate and comparable to traditional histograms for the whole range of available memory, and their accuracy increases with increasing memory. For the two-dimensional case, ST-histograms are better than MHIST-2 histograms when the amount of available memory is small. An MHIST-2 histogram has fewer buckets than a ST-histogram using the same memory. For low memory, the MHIST-2 histogram has too few buckets to cleverly partition the space. The ST-histogram makes better use of what little memory is available by partitioning the space into more buckets and capturing the distribution at a finer granularity.

5.8 On-line Refinement and Convergence

The histogram refinement process and the resulting refined ST-histogram are the same whether we use on-line or off-line refinement. Thus, even though our experiments use off-line refinement, our conclusions are valid for on-line refinement as well. In this section, we switch to on-line refinement to study convergence. Convergence is important for both off-line and on-line refinement, but it is more important and easier to observe for on-line refinement. In addition to studying convergence, we also compare the performance of on-line and off-line refinement.

We issue a random workload one query at a time, recording the estimation error and incrementally refining the ST-histogram after each query. The goal of the refinement process is to reduce the *absolute* estimation error using the histogram. To verify that refinement does indeed reduce this error, we compute the average error of every 100 queries assuming uniformity and independence, using the ST-histogram, and using a traditional histogram. These errors are plotted in Figures 14 and 15 for one and two-dimensional data sets with *z*=1. The figures show that ST-

histogram refinement converges fairly rapidly. These results support our argument that ST-histograms have a low cost. The simple histogram refinement process has to be performed only a small number of times before the histogram becomes sufficiently accurate. The results also demonstrate that our choice of using 2000 queries for the workloads has no effect on our conclusions because refinement converges well before 2000 queries.

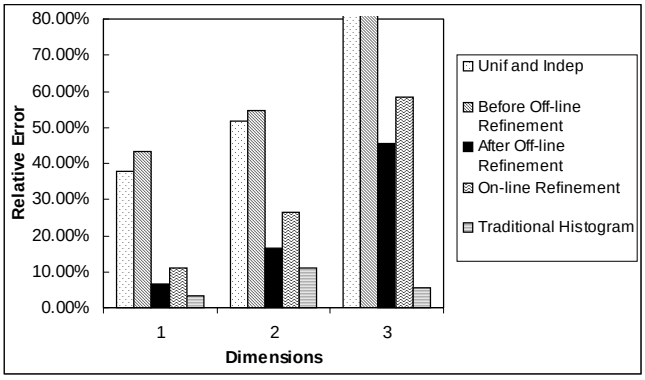

Figure 16: On-line vs. off-line refinement, z =1

Figure 16 compares on-line and off-line refinement. It presents the estimation errors for random workloads on one- to three-dimensional data set with z=1. For each data set, the errors are presented assuming uniformity and independence, using an unrefined ST-histogram, using a ST-histogram that has been refined off-line on the same workload, using a ST-histogram that is being refined on-line with every query, and using a traditional histogram. If we use on-line refinement, the queries see a progressively more accurate histogram. Only after the very last query in the workload is the histogram equivalent to a histogram that has been refined off-line using this workload. On the other hand, if we refine a histogram off-line on a workload and issue the same workload a second time, the entire workload will experience a fully refined histogram the second time it is issued. The interesting result in Figure 16 is that both these scenarios result in comparable estimation errors. Thus, one should choose the refinement method more suitable to the system architecture and expected usage situations, not based on the desired accuracy.

6. Conclusions

In this paper, we introduced a novel way of building histograms at a low cost based on feedback from the query execution engine. ST-histograms use such feedback and *do not* look at the data. Multi-dimensional ST-histograms are particularly attractive, since they provide a low-cost alternative to traditional multi-dimensional structures proposed in the literature that are often prohibitively expensive for large databases (true of many data warehouses). Multi-dimensional ST-histograms are almost as accurate as traditional MHIST-2 histograms for a wide range of data distributions, and sometimes even more accurate, while costing much less to build and maintain.

ST-histograms are better than assuming uniformity and independence for all values of data skew and are comparable in accuracy to traditional histograms for low to medium skew. However, for high data skew, ST-histograms are less accurate than MHIST-2 histograms. Thus, ST-histograms are suitable for low to medium data skew, and the high cost of building traditional multidimensional histograms can be justified only for high data skew.

To combine the best of both worlds, we can start by initializing a ST-histogram and refining it for a preset number of queries representative of the workload on the database. If the training sequence fails to reduce the error to an acceptably low level, then we should consider building a traditional histogram to capture the high skew.

Acknowledgement

We thank Vivek Narasayya for useful discussions and for his help with the experiments.

7. References

[CR94] C.M. Chen and N. Roussopoulos. Adaptive selectivity estimation using query feedback. In *Proceedings of the ACM SIGMOD Conference*, pages 161–172, 1994.

[GMP97] P.B. Gibbons, Y. Matias, and V. Poosala. Fast incremental maintenance of approximate histograms. In *Proceedings of the 23rd International Conference on Very Large Databases*, pages 466–475, 1997.

[KD98] N. Kabra and D.J. DeWitt. Efficient mid-query re-optimization of sub-optimal query execution plans. In *Proceedings of the ACM SIGMOD Conference*, pages 106–117, 1998.

[Koo80] R.P. Kooi. *The optimization of queries in relational databases*. PhD thesis, Case Western Reserve University, September 1980.

[LNS90] R.J. Lipton, J.F. Naughton, and D.A. Schneider. Practical selectivity estimation through adaptive sampling. In *Proceedings of the ACM SIGMOD Conference*, pages 1–11, 1990.

[MD88] M. Muralikrishna and D.J. DeWitt. Equi-depth histograms for estimating selectivity factors for multi-dimensional queries. In *Proceedings of the ACM SIGMOD Conference*, pages 28–36, 1988.

[MRL98] G.S. Manku, S. Rajagopalan, and B.G. Lindsay. Approximate medians and other quantiles in one pass and with limited memory. In *Proceedings of the ACM SIGMOD Conference*, pages 426–435, 1998.

[MVW98] Y. Matias, J.S. Vitter, and M. Wang. Wavelet-based histograms for selectivity estimation. In *Proceedings of the ACM SIGMOD Conference*, pages 448–459, 1998.

[NHS84] J. Nievergelt, H. Hinterberger, K.C. Sevcik. The grid file: an adaptable, symmetric multikey file structure. *ACM Transactions on Database Systems* 9(1):38–71, March 1984.

[PIHS96] V. Poosala, Y.E. Ioannidis, P.J. Haas, and E.J. Shekita. Improved histograms for selectivity estimation of range predicates. In *Proceedings of the ACM SIGMOD Conference*, pages 294–305, 1996.

[PI97] V. Poosala and Y.E. Ioannidis. Selectivity estimation without the attribute value independence assumption. In *Proceedings of the 23rd International Conference on Very Large Databases*, pages 486–495, 1997.

[Zip49] G.K. Zipf. *Human behaviour and the principle of least effort*. Addison-Wesley, Reading, MA, 1949.