Paper Number 402

Approximating multi-dimensional aggregate range queries over real attributes

Dimitrios Gunopulos *
George Kollios †
Vassilis J. Tsotras ‡
Carlotta Domeniconi §

Abstract

Finding approximate answers to multi-dimensional range queries over real valued attributes has significant applications in data exploration and database query optimization. In this paper we consider the following problem: given a table of d attributes whose domain is the real numbers, and a query that specifies a range in each dimension, find a good approximation of the number of records in the table that satisfy the query.

The simplest approach to tackle this problem is to assume that the attributes are independent. More accurate estimators try to capture the joint data distribution of the attributes. In databases, such estimators include the construction of multi-dimensional histograms, random sampling, or the wavelet transform. In statistics, kernel estimation techniques are being used. Many traditional approaches assume that attribute values come from discrete, finite domains, where different values have high frequencies. However, for many novel applications (as in temporal, spatial and multimedia databases) attribute values come from the infinite domain of real numbers. Consequently, each value appears very infrequently, a characteristic that affects the behavior and effectiveness of the estimator. Moreover, real life data exhibit attribute correlations which also affect the estimator.

We present a new histogram technique that is designed to approximate the density of multi-dimensional datasets with real attributes. Our technique finds buckets of variable size, and allows the buckets to overlap. Overlapping buckets allow more efficient approximation of the density. The size of the cells is based on the local density of the data. This technique leads to a faster and more compact approximation of the data distribution. We also show how to generalize kernel density estimators, and how to apply them on the multi-dimensional query approximation problem.

Finally, we compare the accuracy of the proposed techniques with existing techniques using real and synthetic datasets. The experimental results show that the proposed techniques behave more accurately in high dimensionalities than previous ones.

*CS Dept., Univ. of California Riverside, Riverside, CA 92521, dg@cs.ucr.edu, Contact author. This research has been supported by NSF IIS-9907477 and the US Dept. of Defense.

†CS Dept., Polytechnic Univ., Brooklyn, NY 11201, gkollios@cs.ucr.edu.

‡CS Dept., Univ. of California Riverside, Riverside, CA 92521, tsotras@cs.ucr.edu. This research has been supported by NSF IIS-9907477 and the US Dept. of Defense.

§CS Dept., Univ. of California Riverside, Riverside, CA 92521, carlotta@cs.ucr.edu.

1 Introduction

Computing approximate answers to multi-dimensional range queries is a problem that arises in query optimization, data mining and data warehousing. The query optimizer requires accurate estimations of the sizes of intermediate query results in the evaluation of different execution plans. Recent work also shows that top-k queries can be mapped to multi-dimensional queries [4, 8], so selectivity estimation techniques can be used to optimize top-k queries.

The problem of approximating multi-dimensional range queries is also relevant for data mining applications. Answering range queries is one of the simpler data exploration tasks. In this context, the user defines a specific region of the dataset that is interested to explore, and asks queries to find the characteristics of this region (like the number of points in the interior of the region, the average value or the sum of the values of attributes in the region). Consider for example a dataset that records readings of different environmental variables, such as types of pollution, at various space locations. In exploring this dataset the user may be interested in answering range queries similar to: find how many locations exist for which the values of given pollution variables are within a specified range. The user may want to restrict the answers to a given geographical range too. The size of such datasets makes exact answers slow in coming, and only an efficient approximation algorithm can make this data exploration task interactive.

In data warehousing, datasets can be very large. Answering aggregate queries exactly can be computationally expensive. It is therefore very important to find approximate answers to aggregate queries quickly in order to allow the user to explore the data.

In this paper we address the problem of estimating the selectivity of multi-dimensional range queries when the datasets have numerical attributes with real values. The range queries we consider are intersections of ranges, each range being defined on a single attribute. In the multi-dimensional attribute space, the queries are then hyper-rectangles with faces parallel to the axes. Solving such a range query exactly involves counting how many points fall in the interior of the query. When the number of dimensions increases, recent results show [35] that the query time is linear to the size of the dataset.

We should emphasize that this problem is different from the traditional definition of selectivity estimation, where it has been generally assumed that each numerical attribute has a finite discrete domain. In many applications however the attribute domain is the real numbers. This is typically the case in spatial, temporal and multimedia databases where the objects are represented as feature vectors. For example consider a relation that records climatic data like humidity, wind speed, precipitation, at different points of interest and different times.

Real domains have two important consequences. First, the number of possible queries is infinite in the case of real domains, but finite when considering a finite discrete domain. In the case of real domains, the number of possible query answers is still finite, since the dataset is finite, but depends on the size of the dataset. Second, with real domains it is unlikely that many attribute values will appear more than once in the database. Nevertheless some techniques that have been developed to solve the discrete finite domain case are still applicable, if properly modified.

The main approach to solve the selectivity estimation problem has been to compute a non-parametric density estimator to the data distribution function. There is a number of methods suggested in the literature that use different ways to find the density estimator for attributes with finite discrete domains. They include computing multi-dimensional histograms [25] [1] [18] [2], using the wavelet transformation [33] [21], SVD [25] or the discrete cosine transform [20] on the data, using kernel estimators [3], and sampling [23] [19] [11].

Density estimator techniques attempt to define a function that approximates the data distribution. Since we must be able to derive the approximate solution to a query quickly, the description of the function must be kept in memory. Further, we may have to answer queries on many datasets, so the description cannot be large. The success of the different methods depends on the simplicity of the function, the time it takes to find the function parameters, and the number of parameters we store for a given approximation quality.

Multi-dimensional density estimation techniques are typically generalizations of very successful one-dimensional density estimators. In general, in one dimension, estimators of small size (histograms, kernels, sampling) can be used to effectively approximate the data distribution. Indeed, one of the techniques used to solve the multi-dimensional problem is to assume that attributes are independent, and therefore an estimator for multiple dimensions can be obtained by multiplying one-dimensional estimators.

Unfortunately, if the independence assumption does not hold, as it frequently happens, the problem of estimating the result of range queries in multiple dimensions becomes tougher as the dimensionality increases. One of the reasons is that the volume of the space increases exponentially with the dimensionality, and datasets of any size become sparse and do not allow accurate density estimation in any local area. This problem is referred to as the curse of dimensionality [35]. Finding density estimators for combinations of attributes also allows us to find out whether the independence assumption holds for a set of attributes or not. This is of independent importance for the query optimizer: many query optimizers [28] compute query execution costs under the attribute independent assumption. Knowing where this does not hold allows one to keep better statistics for these sets of attributes.

1.1 Our Contribution

There are three main contributions from this work:

First, we present a new technique, GENHIST, to find multi-dimensional histograms for datasets from real domains. The method has been designed to approximate the joint data distribution more efficiently than existing techniques. One of the basic characteristic in our technique is that the histogram buckets can be overlapping. The technique uses more and smaller buckets to approximate the data distribution where the density of points is larger and fewer and larger buckets where the density decreases.

Second, we show how to use multi-dimensional kernel density estimators to solve the multi-dimensional range query selectivity problem. Our technique generalizes in multiple dimensions the technique given in [3]. Kernel estimation is a generalization of sampling. Like sampling, finding a kernel estimator is efficient, and can be performed in one pass. In addition, kernel estimators produce a smoother density approximation function that is shown experimentally to better approximate the dataset density distribution.

Third, we present an extensive comparison between the new techniques (GENHIST, multi-dimensional kernel density estimators), and most of the existing techniques for estimating the selectivity of multi-dimensional range queries for real attributes (wavelet transform [33], multi-dimensional histogram MHIST [25], one-dimensional estimation techniques with the attribute independence assumption, and sampling [11]). We include the attribute independence assumption in our study as a baseline comparison.

The experimental results show that we can efficiently build selectivity estimators for multi-dimensional datasets with real attributes. Although the accuracy of all the techniques drops rapidly with the increase in dimensionality, the estimators are still accurate in 5 dimensions. GENHIST appears to be the most robust and accurate technique that we tested. Multi-dimensional kernel estimators are also competitive in accuracy. An advantage of kernel estimators is that they can be computed in one dataset pass (just like sampling). However, they work better than sampling for the dimensionalities we tried. Therefore, multi-dimensional kernel estimators are the obvious choice when the selectivity estimator must be computed fast.

In the next section (Section 2) we formally define the problem. In section 3 we briefly describe the multi-dimensional histogram and wavelet decomposition approaches. We present a new multi-dimensional histogram construction in section 4. We describe how to use kernel estimators for multi-dimensional data in section 5. Section 6 includes our experimental results, and we conclude in section 7.

2 Problem Description

Let R be a relation with d attributes and n tuples. Let $\mathcal{A} = \{A_1, A_2, \ldots, A_d\}$ be the set of these attributes. The domain of each attribute A_i is scaled to the real interval $[0,1]$. Assuming an ordering of the attributes, each tuple is a point in the d-dimensional space defined by the attributes. Let V_i be the set of values of A_i that are present in R. Since the values are real, each could be distinct and therefore $|V_i|$ can be as large as n.

The range queries we consider are of the form $(a_1 \leq R.A_1 \leq b_1) \wedge \ldots \wedge (a_d \leq R.A_d \leq b_d)$. All a_i, b_i are assumed to be in $[0,1]$. Such a query is a hyper-rectangle with faces parallel to the axes. The *selectivity* of the query $sel(R,Q)$ is the number of tuples in the interior of the hyper-rectangle.

Since n can be very large, the problem of approximating the selectivity of a given range query Q arises naturally. The main problem is how to preprocess R so that accurate estimations can be derived from a smaller representation of R without scanning the entire relation, or counting exactly the number of points in the interior of Q.

Let $f(x_1, \ldots x_d)$ be a d-dimensional, non-negative function that is defined in $[0,1]^d$ and has the property that

$$\int_{[0,1]^d} f(x_1, \ldots x_d) dx_1 \ldots dx_d = 1.$$

We call f a *probability density function*. The value of f at a specific point $\mathbf{x} = (x_1, \ldots x_d)$ of the d-dimensional space is the limit of the probability that a tuple exists in area U around $\mathbf{x}$ over the volume of U, when U shrinks to $\mathbf{x}$.

For a given f with these properties, to find the selectivity of query $(a_1 \leq R.A_1 \leq b_1) \wedge \ldots \wedge (a_d \leq R.A_d \leq b_d)$ we compute the integral of f in the interior of the query Q:

$$sel(f,Q) = \int_{[a_1,b_1]\times\ldots\times[a_d,b_d]} f(x_1, \ldots x_d) dx_1 \ldots dx_d.$$

For a given R and f, f is a good *estimator* of R with respect to range queries if for any range query Q, the selectivity of Q on R and the selectivity of Q on f multiplied by n are similar. To formalize this notion, we define the following error metrics (also used by [33]).

For a given query Q, we define the *absolute error* of Q to be simply the difference between the real value and the estimated value:

$$\epsilon_{abs}(Q,R,f) = |sel(R,Q) - n\ sel(f,Q)|.$$

The *relative error* of a query Q is generally defined as the ratio of the absolute error over the selectivity of the query. Since in our case a query can be empty, we follow [33] in defining the relative error as the ratio of the absolute error over the maximum of the selectivity of Q and 1:

$$\epsilon_{rel}(Q,R,f) = \frac{|sel(R,Q) - n\ sel(f,Q)|}{\max(1, sel(R,Q))}.$$

Finally the *modified relative error* of a query Q is the absolute error over the minimum of the actual selectivity and the estimated selectivity of Q (since this can be zero, we again have to take the maximum with 1):

$$\epsilon_{rel\_mod}(Q,R,f) = \frac{|sel(R,Q) - n\ sel(f,Q)|}{\max(1, \min(sel(R,Q), n\ sel(f,Q)))}.$$

To represent the error of a set of queries, we define the *p-norm average error*. Given a query workload $\{Q_1, \ldots, Q_k\}$ comprising of k queries, R, f, and an error metric ϵ that can be any of the above, the p-norm average error for this workload is:

$$\parallel \epsilon \parallel_p = (\frac{1}{k} \sum_{1 \le i \le k} \epsilon(Q_i, R, f)^p)^{\frac{1}{p}}.$$

We can also define different aggregate queries such as the sum on one attribute: $sum(R,Q,i) = \sum_{(x_1,\ldots,x_d) \in R \cap Q} x_i$, or the average $ave(R,Q,i) = \frac{\sum_{(x_1,\ldots,x_d) \in R \cap Q} x_i}{sel(R,Q)}$. Following [29], we can approximate such a query using the density estimator f:

$$sum(f,Q,x_i) = \int_Q x_i\ f(x_1,\ldots,x_d) dx_1 \ldots dx_d$$

3 Multi-dimensional Density Estimators

In this section we briefly examine existing techniques to estimate the selectivity of a query. We group them into histograms (one- and multi-dimensional), discrete decomposition techniques, and statistical estimators.

3.1 One-dimensional Histograms

In system R [28], density estimators for each attribute are combined under the attribute independence assumption to produce a multi-dimensional density estimator. To estimate the selectivity of a multi-dimensional query as a fraction of the size of relation R, first the query is projected on each attribute and the selectivity of each one-dimensional query is estimated, and then the selectivities are multiplied. Typically one-dimensional histograms are used. This technique is still widely employed.

3.2 Multi-dimensional Histograms

Multi-dimensional histograms were introduced in [22]. Multi-dimensional histograms attempt to partition the data space into b non-overlapping buckets. In each bucket, the data distribution is assumed to be uniform. Partitioning a d-dimensional space into buckets is a difficult problem: finding an optimal equi-sum histogram with b buckets is NP-complete even in two dimensions [16], although the problem is easy to be solved in one dimension (in an equi-sum histogram the sum of the tuples in each bucket is approximately the same).

[25] presents two new algorithms, PHASED and MHIST-2, the second being an "adaptive" version of the first. In PHASED, the order in which the dimensions are to be split is decided only once at the beginning and

arbitrarily; in MHIST-2, at each step, the most "critical" attribute is chosen for partition. For a MaxDiff histogram, at each step, MHIST-2 finds the attribute with the largest difference in source values (e.g., spread, frequency, or area) between adjacent values, and places a bucket boundary between those values. Therefore, when frequency is used as a source parameter, the resulting MaxDiff histogram approximates the minimization of the variance of values' frequencies within each bucket. Of the two techniques, MHIST-2 is shown to be more accurate and performs better than previous approaches [25].

Both algorithms can be used directly on a dataset with real valued attributes. However, when we use real valued data, all the different values in the dataset have frequency of 1 because each value that appears, appears exactly once. We run our experiments with MHIST-2 on real attribute data, however, the processing time of the method increases because the number of possible positions among which a bucket boundary can be chosen increases as well.

Another very interesting alternative to multi-dimensional histograms is the self-tuning histograms (STH) recently presented in [1]. The idea is to start with a very simple initial histogram, then use feedback from a query execution engine regarding the actual selectivity of range queries, and refine the histograms accordingly. Thus, the cost of building these histograms is low since it is independent of the data size. However ST-histograms are less accurate than MHIST-2 for high dimensions and skewed data [1].

3.3 Discrete Decomposition Techniques

The d-dimensional data distribution of a dataset R with attributes $A_1, \ldots A_d$ can be represented by a d dimensional array D with $\prod_{1 \leq i \leq d} |V_i|$ slots (recall that V_i is the set of distinct values of attribute A_i). The value in each slot is the number of times this value combination appears in R.

One approach to find an approximation of the joint data distribution is to approximate the array D directly. A number of decomposition techniques have been proposed in the literature to find such an approximation. These include the Singular Value Decomposition (SVD) [25], the wavelet transform [33], and recently the Discrete Cosine Transform (DCT) [20].

The basic operation of all decomposition techniques is to essentially perform a change of bases. Each value in the original array D is now computed as a combination of the new basis. For the three methods mentioned, this combination is linear. The coefficients in this combination are stored in a new array D' that has the same size as D. The important observation is that many coefficients in the new array D' may be close to zero. Therefore the contribution of each of these coefficients in the evaluation of a value of D is small, and they can be set to zero with little loss of accuracy.

These techniques compute the transformation and keep the b largest coefficients, for a given input parameter b. The remaining coefficients are set to zero. This results in an array D'' with b non zero values (so we need $O(b)$ space to store it). To estimate a value of D we compute the inverse transformation on D''.

The accuracy depends on the distribution of the values of the coefficients. If the transformation results in many large coefficients, the error will be large for all practical values of b. The feasibility of the approach depends on the time it takes to perform the transformation and the inverse transformation.

Of the three techniques we mentioned, SVD can only be used in two dimensions ([25]). Wavelets and, recently, DCT have been shown to give good results in high dimensionalities [33] [21] [20]. In our comparisons we include the wavelet transform.

If the attributes have real values, the size of the matrix D can be n^d, where n is the size of R. Since each of these approaches involves operations on the matrix D, we cannot use the raw data, and therefore we perform a ξ *regular partitioning* of the data space first. Hence, we partition the domain of each attribute into ξ non-overlapping intervals. The width of each interval is $1/\xi$, so that we obtain an equi-width partitioning, resulting into ξ^d d-dimensional non-overlapping buckets that cover the entire data space.

If the value x_i of a tuple $\mathbf{x} \in R$ falls in the j-th interval of attribute A_i, then it is replaced by j. Thus the domain of each attribute becomes $\{1, \ldots, \xi\}$. We call the new dataset R'. We define the *frequency* of a tuple $\mathbf{t}= (t_1, \ldots, t_d) \in \{1, \ldots, \xi\}^d$ to be the number of tuples $\mathbf{x} \in R$ that are mapped to $\mathbf{t}$ (Figure 1). We denote the resulting ξ^d size matrix with D_ξ. We can then use the wavelet transform to obtain an approximation for R'.

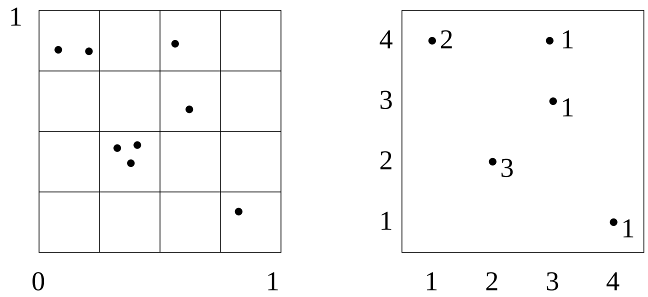

Figure 1: A 4-regular partitioning of a 2-dimensional dataset and the tuple frequencies.

Note however that queries do not have to fall on interval boundaries. In one dimension, a query may contain a number of intervals completely, and partially intersect at most two. In two dimensions a query can intersect up to $4(\xi - 1)$ buckets. In general, in d dimensions a query can intersect $O(d\ \xi^{d-1})$ buckets (out of a total number of ξ buckets). We assume that the points are uniformly distributed in the interior of a bucket.

To estimate the selectivity of a query, we use the approximation of R' to find the approximate values for every bucket of the ξ regular partitioning that intersects the query. If the bucket is completely inside the query, we just add up the approximate value. If it isn't we use the uniform distribution assumption to estimate the fraction of tuples that lie in the interior of the bucket that also lie in the interior of the query. In this case the data distribution is approximated by the average data density within each bucket. The *average data density* (or simply data density) of a bucket is defined as the number of tuples in the bucket over the volume of the bucket.

It becomes clear that the quality of the approximation critically depends on how closely the actual point distribution resembles the uniform distribution inside each bucket.

We can compute the wavelet decomposition of either D_ξ or the partial sum matrix D^s_ξ of D_ξ. In the partial sum the value of a matrix slot is replaced by the sum of all preceding slots:

$$D^s_\xi[i_1, \ldots, i_d] = \sum_{j_1 \le i_1, \ldots, j_d \le i_d} D_\xi[j_1, \ldots, j_d].$$

We ran experiments to determine which of the two methods should be used. The results indicate that wavelets on the partial sum matrix provide a more accurate approximation for our datasets. Also [33] suggests that partial sum method is more accurate because this operation smoothes up the data distribution.

3.4 Statistical Estimators

The simplest statistical method for selectivity estimation is sampling. One finds a random subset S of size b of the tuples in R. Given a query Q, the selectivity $sel(S, Q)$ is computed. The value $\frac{n}{b} sel(S, Q)$ is used to estimate $sel(R, Q)$. Sampling is simple and efficient, and so it is widely used for estimating the selectivity [28] [5] [9] or for on-line aggregation [12]. Sampling can be used to estimate the selectivity of a query regardless of the dimensionality of the space, and can directly be applied to real domains.

More sophisticated kernel estimation statistical techniques [34] [6] have rarely been applied in database problems. One of the reasons is that in statistics a dataset is considered an instance of a probability distribution function, and the goal is to approximate the probability distribution function itself. On the other hand, in databases the goal is simply to approximate the dataset. Recently [3] used kernels to estimate the selectivity of one-dimensional range queries on metric attributes.

One similar statistical technique is clustering the dataset and using a Gaussian function to model each cluster [29]. This technique can be quite accurate if the clusters themselves can be accurately modeled by multi-dimensional Gaussian functions. Even assuming that this is the case however, the technique requires clustering the dataset, a task that is much less efficient than simple sampling.

4 A New Multi-Dimensional Histogram Construction

In this section we present a new density estimation technique, GENHIST (for GENeralized HISTograms). As in other histogram algorithms, we want to produce a density estimator for a given dataset R using rectangular buckets. The important difference is that we allow the buckets to overlap.

Histogram techniques typically partition the space into buckets and assume that the data distribution inside each bucket is uniform (if uniformity within each bucket is not assumed, additional information about the data distribution must be kept, at a higher storage cost). The problem with partitioning the space into a fixed, small, number of buckets is that the volume of the space increases exponentially when the dimensionality increases, or alternatively, the number of buckets that are required to partition the space increases exponentially with the dimensionality. Even a partitioning scheme that partitions each attribute into 4 one-dimensional buckets, generates $4^5 = 1024$ 5-dimensional buckets. Since the data points become sparser in higher dimensions it is very likely that the actual data distribution deviates significantly from the uniform distribution within each of these buckets. The problem becomes severe in higher dimensions because the number of buckets a query can partially intersect increases exponentially with the dimensionality. For example, consider a ξ regular partitioning of a d dimensional space. A query that is the intersection of two $(d-1)$-dimensional hyper-planes intersects ξ^{d-1} buckets. In the 4 regular partitioning of a 5-dimensional space example above, such a query partially intersects 256 out of 1024 total buckets.

Clearly, to achieve acceptable accuracy a technique has to either ensure that the data distribution within each bucket is close to uniform, or ensure that each bucket contains a small number of points and therefore the error for each bucket is small. Note that non-overlapping partitions into b buckets allow only b different values for estimating the data density. To increase the number of values, one has to increase the number of buckets. This has conflicting results. The accuracy within each bucket becomes better, but the overall accuracy may decrease because a query can now partially intersect more buckets.

Our approach to solve this problem is to allow overlapping buckets. The intuition is the following. As in previous approaches, we assume that within each bucket the data distribution can be approximated by the average data density of the bucket. But when two buckets overlap, in their intersecting area we assume that the data density is the sum of the two densities. If more than two buckets overlap, the data density in their intersecting area will be approximated by the sum of the data densities of the overlapping buckets. Clearly, for our scheme to work, we have to be careful when we compute the average density within each bucket. In particular, if a tuple lies in the intersection of many buckets, we must count it in the computation of the average density of only one of the buckets.

A simple two dimensional example shows that we can achieve more using overlapping buckets (Fig. 2). In the example we partition $[0, 1]$ using four buckets. If the buckets are non-overlapping, this results into a partitioning into 4 regions. We have to assume that the data distribution is uniform within each region. If we use 4 overlapping buckets of the same size, we can partition $[0, 1]$ into a total of 9 regions. Although we again keep only 4 numbers, each of the 9 regions is the intersection of different buckets and its density is estimated differently. Moreover, if we use 4 rectangular buckets of different sizes, we can partition $[0, 1]$ into 13 regions, each with a different estimated density. The number of intersections increases exponentially with the dimensionality. This implies that the advantage of overlapping buckets increases with the dimensionality. On the other hand, this exponential increase makes the problem of finding the optimal size and location of the buckets and the values of the average densities in each bucket, computationally hard. To tackle this problem, we present an efficient heuristic algorithm that works in linear time and performs a constant number of database passes.

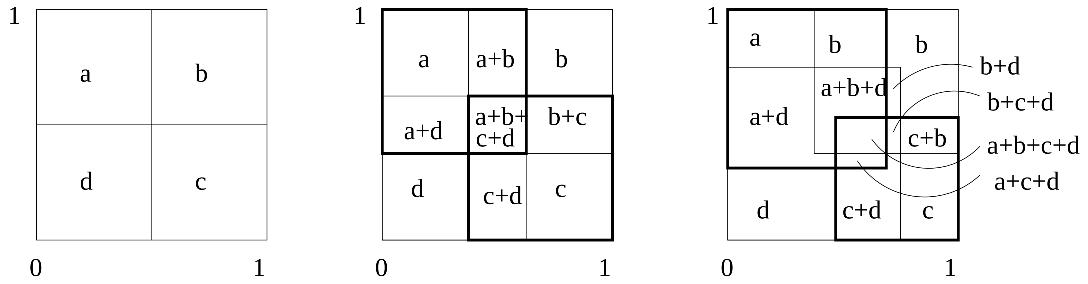

Figure 2: The same 2-dimensional space is partitioned first into 4 regions, using 4 non-overlapping buckets (average densities are a, b, c and d), then into 9 regions, using 4 overlapping equal-sized buckets (densities are a, b, c, d, a+b, b+c, c+d, d+a, a+b+c+d), and finally into 13 regions, using 4 overlapping buckets of different sizes.

4.1 Heuristic for finding generalized multi-dimensional histograms

Our heuristic approach partitions a d-dimensional space using b overlapping buckets of different sizes. The main idea of the algorithm is to iteratively compute an approximation of the density function using a grid. In each iteration, the algorithm tries to find and approximate the dense areas. Our approach for finding dense areas is to partition the space using a regular grid, and find which buckets have the larger average density. Buckets with high count are in areas where the density is large. However, instead of removing the entire dense buckets, we only remove enough points from each bucket so that the density of this bucket is approximately equal to its surrounding area. Tuples are removed at random to ensure that the density decreases uniformly to the level of the density of neighboring buckets.

This is important for two reasons: (i) The density of the entire dataset is now smoother, because the high bumps have been removed. This means that the remaining bumps are also smoother, and can be approximated using a coarser grid. (ii) In the successive iteration we have to approximate the new smoother data density in the entire data space. To solve this problem we naturally use the same technique, namely, using a grid to find dense buckets.

Due to the overall smoothing effect, a coarser grid is used in each successive iteration. The buckets with the largest densities are kept in the estimator, along with their average density values set to the fraction of points removed from each. Clearly, buckets produced from different iterations can overlap. This ensures that the density of regions in the intersection of many buckets is correctly estimated by adding up the average densities of each one.

Thus GENHIST can be classified as a generalization of the biased histograms [26]. Biased histograms keep in singleton buckets the values with highest frequencies, and partition the remaining values in a number of buckets. Like biased histograms, GENHIST uses buckets to approximate the areas where the density is highest. In addition, it repeats the same process after the smoothing step.

The possible bucket overlapping effectively partitions the data space into a much larger number of regions than simply the number of buckets (Fig. 2). In this respect the technique is similar to the wavelet transform. Just as the wavelet transform provides more resolution in the areas where the variation in the frequencies is highest, GENHIST provides more detail in the areas where more points are concentrated (the areas of higher density).

We next describe the algorithm in detail. There are three input parameters: the initial value of ξ, the number of buckets we keep at each iteration, and the value of α that controls the rate by which ξ decreases. We describe how to set these parameters after the presentation of the algorithm. The output of the algorithm is a set of buckets E along with their average densities. This set can be used as a density estimator for R. Figure 3 gives the outline of the algorithm.

We use $\alpha = (1/2)^{1/d}$ to ensure that at each iteration we use roughly half as many buckets to partition the space as in the preceding operation (the new buckets have approximately twice the volume of the previous ones). Unless we remove more than half the tuples of R in each iteration, the average number of tuples per bucket increases slightly as we decrease the number of buckets. This is counterbalanced by the overall smoothing effect we achieve in each iteration. S counts the number of points we remove during an iteration. If this number is large ($\frac{|R|}{|R|+S} < \alpha^d$), we decrease ξ faster, and we do not allow the average bucket density to decrease between operations.

The number of buckets that we remove in each iteration is constant. Since ξ is replaced by $\lfloor \alpha\xi \rfloor$ in each operation, we expect to perform approximately $\log_{\frac{1}{\alpha}} \xi$ iterations, and in each iteration we keep approximately $\lceil b/\log_{\frac{1}{\alpha}} \xi \rceil$ buckets. The value of b is provided by the user.

The choice of ξ is important. If ξ is set too large, the buckets in the first iterations are practically empty. If ξ is set too low, then we loose a lot of details in the approximation of the density function. Since we have to provide b buckets, we set ξ so that in the first iteration the percentage of the points that we remove from R is at least $1/\log_{\frac{1}{\alpha}} \xi$.

Given a d-dimensional dataset R with n points and input parameters b, ξ, and α,

1. Set E to empty.
2. Compute a ξ regular partitioning of $[0,1]^d$, and find the average density of each bucket (i.e., number of points with in each bucket divided by n).
3. Find the b_ξ buckets with highest density.
4. For each bucket c of the b_ξ buckets with highest density:
   (a) Let d_c be the density of c.
   Compute the average density av_c of c's neighboring buckets.
   (b) If the density of c is larger than the average density av_c of its neighbors:
      i. Remove from R a randomly chosen set of $(d_c - av_c)n$ tuples that lie in c.
      ii. Add bucket c into the set E and set its density to $d_c - av_c$.
5. Set $S = \sum_{c \in b_\xi}(d_c - av_c)n$ (S is the number of removed points).
   Set $\alpha' = \min((\frac{|R|}{|R|+S})^{\frac{1}{d}}, \alpha)$.
   Set $\xi = \lfloor \alpha' \, \xi \rfloor$.
6. If R is empty, return the set of buckets E.
   else if R is non empty and $\xi > 1$ return to Step 2.
   else if $\xi \leq 1$ add bucket $[0,1]^d$ with density $\frac{|R|}{n}$ to E and output the set of buckets E.

Figure 3: The GENHIST algorithm

4.1.1 Running Time

The running time of the algorithm is linear in the size of R. One pass over the data is performed each time the value of ξ changes. Since ξ is originally a small constant, the number of passes over the data is also small. In our experiments the number of passes was between 5 and 10. During each pass, to compute the number of points that fall in each bucket, we use a hashing scheme: non-empty buckets are kept in a hash table. For each point, we compute the slot it should be in and probe the hash table. If the bucket is there, we increment its counter, otherwise we insert it into the hash table. This simple scheme allows us to compute the bucket counts for large values of ξ in memory, even when the dataset has millions of points.

Implementing step 4.b.ii of the algorithm can slow down the process, because we have to designate that some points in the dataset are deleted, and to do so we have to modify the dataset.

The following technique allows us to estimate accurately, at each step of the algorithm, the number of remaining points that fall in each bucket, without having to write at the disk at all.

Assume that we are scanning the dataset D at the i-th iteration, and, in the previous $i-1$ iterations we have already computed a set of buckets that we will keep in the estimator. During the i-th scan of dataset D

we want to determine the number of points that fall in the interior of each bucket in the grid, assuming that some points have been removed from the interior of each bucket in the estimator. For each such bucket B_j in the estimator we keep the total number dataset points that lie in its interior (let that number be $tot(B_j)$), and the number of points we would remove from B_j in step 4.b.ii (let that number be $r(B_j)$.

During the scan of D, if a point p lies in a bucket B_j that we have already included in the estimator, then, with probability $\frac{r(B_j)}{tot(B_j)}$, we do not use this point in the computation of densities of the grid buckets.

Lemma 1 *The expected density of a bucket in the i-th iteration that is computed using this process is equal to the expected density of a bucket if we had removed the same number of points at random in the previous iteration*

Proof: (Sketch) We only have to show that the probability that a point is not counted in the i-th iteration is the same as the probability that the same point has been removed in an earlier iteration. This is true if the point lies in the interior of only one bucket in the estimator. If it is not, then we have to consider the buckets in the order that they were added to the estimator. Assume that p is in the interior of B_i and B_j, and B_i was added before B_j. Then we do not use p with probability $\frac{r(B_i)}{tot(B_i)} + (1 - \frac{r(B_i)}{tot(B_i)})\frac{r(B_j)}{tot(B_j)}$ □

4.2 Estimating the selectivity of range queries using GENHIST

The estimation of the selectivity of a query using overlapping buckets is similar to the non-overlapping case. We check every bucket against the query to see if there is any overlap. We add up the overlapping contributions, again assuming uniform distribution inside each bucket. More formally, given a set of buckets $G = (B_1, \ldots, B_b)$ and a range query Q, the estimated selectivity of the query is:

$$sel(G,Q) = \sum_{B \in G} \frac{B_{count}}{Vol(B)} Vol(Q \cap B),$$

where B_{count} is the number of points inside bucket B, $Vol(B)$ is the volume of bucket B, and $Vol(Q \cap B)$ is the volume of the intersection between query Q and bucket B.

5 Multi-dimensional kernel density estimators

For the problem of computing the query selectivity, all the proposed techniques compute a density estimation function. Such function can be thought as an approximation of the probability distribution function, of which the dataset at hand is an instance. It follows that statistical techniques which approximate a probability distribution, such as kernel estimators, are applicable to address the query estimation problem. The problem of estimating an underlying data distribution has been studied extensively in statistics [27] [34].

Kernel estimation is a generalized form of sampling. The basic step is to produce a uniform random sample from the dataset. As in random sample, each sample point has weight one. In kernel estimation however, each point distributes its weight in the space around it.

A *kernel function* describes the form of the weight distribution. Generally, a kernel function distributes most of the weight of the point in the area very close the point, and tapers off smoothly to zero as the distance from the point increases. If two kernel centers are close together, there may be a considerable region where the non-zero areas of the kernel functions overlap, and both distribute their weight in this area. Therefore, a given location in the data space gets contributions from each kernel point that is close enough to this location so that its respective kernel function has a non zero value. Summing up all the kernel functions we obtain a density function for the dataset.

Let us consider the one dimensional case first. Assume that R contains tuples with one attribute A whose domain is $[0,1]$. Let S be a random subset of R (our sample). Also assume that there is a function $k_i(x)$ for each tuple t_i in S, with the property that $\int_{[0,1]} k(x)dx = 1$. Then the function

$$f(x) = \frac{1}{n} \sum_{t_i \in S} k_i(x - t_i)$$

is an approximation of the underlying probability distribution according to which R was drawn.

To approximate the selectivity of a query Q of the form $a \leq R.A \leq b$, one has to compute the integral of the probability function f in the interval $[a,b]$:

$$\sigma(f,Q) = \int_{[a,b]} f(x) = \frac{1}{n} \sum_{t_i \in S} \int_{[a,b]} k_i(x - t_i).$$

As defined, kernel estimation is a very general technique. [27] shows that any non-parametric technique for estimating the probability distribution, including histograms, can be recast as a kernel estimation technique for appropriately chosen kernel functions.

In practice the functions $k_i(x)$ are all identical. The approximation can be simplified to

$$f(x) = \frac{1}{n} \sum_{t_i \in S} k(x - t_i).$$

To use kernels in d-dimensions we have to provide a d-dimensional kernel function.

For a dataset R, let S be a set of tuples drawn from R at random. Assume there exists a d dimensional function $k(x_1, \ldots, x_d)$, the *kernel function*, with the property that

$$\int_{[0,1]^d} k(x_1, \ldots, x_d) dx_1 \ldots dx_d = 1$$

The approximation of the underlying probability distribution of R is

$$f(x) = \frac{1}{n} \sum_{t_i \in S} k(x_1 - t_{i_1}, \ldots, x_d - t_{i_d}),$$

and the estimation of the selectivity of a d-dimensional range query Q is

$$sel(f,Q) = \int_{[a,b]^d \cap Q} f(x_1, \ldots, x_d) = \frac{1}{n} \sum_{t_i \in S} \int_{[a,b]^d \cap Q} k(x_1 - t_{i_1}, \ldots, x_d - t_{i_d}) dx_1 \ldots dx_d.$$

It has been shown that the shape of the kernel function does not affect the approximation substantially [6]. What is important is the standard deviation of the function, or, its bandwidth. Therefore, we choose a kernel function that it is easy to integrate. The Epanechnikov kernel function has this property [6]. The d-dimensional Epanechnikov kernel function centered at 0 is

$$k(x_1, \ldots, x_d) = (\frac{3}{4})^d \frac{1}{B_1 B_2 \ldots B_d} \prod_{1 \leq i \leq d} (1 - (\frac{x_i}{B_i})^2)$$

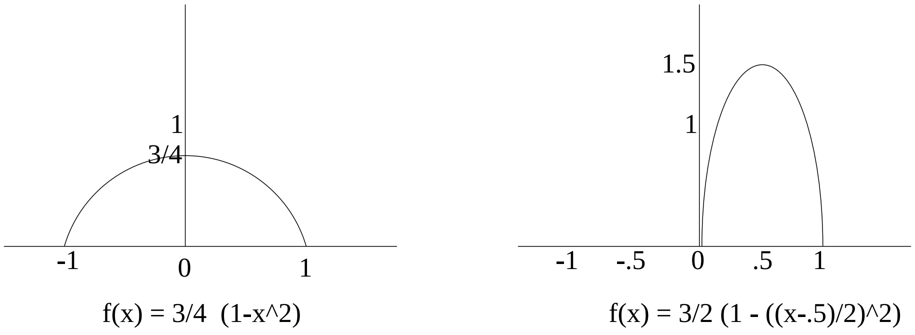

Figure 4: The one-dimensional Epanechnikov kernel, with $B = 1$, centered around the origin, and, with $B = 2$, centered at 0.5.

if, for all i, $|\frac{x_i}{B_i}| < 1$, and 0 otherwise (Figure 4).

The d parameters $B_1, \ldots, B_d$ are the bandwidth of the kernel function along each of the d dimensions. The magnitude of the bandwidth controls how far from the sample point we distribute the weight of the point. As the bandwidth becomes smaller, the non-zero diameter of the kernel becomes smaller.

There are two problems that we have to solve before we can use the multi-dimensional kernel estimation method. The first is setting the bandwidth parameters. The second problem is that in high dimensions many tuples, and therefore many samples, are likely to be close to one of the faces of the $[0,1]^d$ cube. If a kernel is close to the space boundary and its bandwidth goes beyond it, then part of the volume that is covered by the kernel function lies outside the data (and the query) space. The result is that these points distribute less than weight 1 to the area around them, and so $\int_{[0,1]^d} f(x_1, \ldots, x_d) dx_1 \ldots dx_d < 1$. This problem is referred to as the boundary problem.

Both problems have been addressed before in statistics [34]. No efficient solution exists for finding the optimal bandwidths. To get an initial estimate for the bandwidth we use Scott's rule [27] in d-dimensional space: $B_i = \sqrt{5}\, s_i\, |S|^{-\frac{1}{d+4}}$, where s_i is the standard deviation of the sample on the i-th attribute. This rule is derived using the assumption that the data distribution is a multi-dimensional normal and so it oversmoothes the function. To solve the second problem, we project the parts of the kernel function that lie outside $[0,1]^d$ back into the data space. The complexity of this projection increases with the dimensionality, because each d-dimensional corner of $[0,1]^d$ partitions $\mathcal{R}^d$ into 2^d quadrangles, and we have to find the intersection of the kernel function with each quadrangle.

5.1 Computing the selectivity

Since the d-dimensional Epanechnikov kernel function is the product of d one-dimensional degree-2 polynomials, its integral within a rectangular region can be computed in $O(d)$ time:

$$sel(f, [a_1, b_1] \times \ldots \times [a_d, b_d]) = \frac{1}{n} \int_{[a_1,b_1]\times\ldots\times[a_d,b_d]} \left(\sum_{1 \le i \le b} k_i(x_1, \ldots, x_d) dx_1 \ldots dx_d \right) =$$

$$\frac{1}{n}\int_{[a_1,b_1]\times\ldots\times[a_d,b_d]}\sum_{1\leq i\leq b}(\frac{3}{4})^d\frac{1}{B_1B_2\ldots B_d}\prod_{1\leq j\leq d}(1-(\frac{x_j-X_{ij}}{B_j})^2))\,dx_1\ldots dx_d =$$

$$\frac{1}{n}(\frac{3}{4})^d\frac{1}{B_1B_2\ldots B_d}\sum_{1\leq i\leq b}\int_{[a_1,b_1]}(1-(\frac{x_1-X_{i1}}{B_1})^2)\ldots\int_{[a_d,b_d]}(1-(\frac{x_d-X_{id}}{B_d})^2)\,dx_d\ldots dx_1$$

It follows that, for a sample of $|S|$ tuples, $sel(f,Q)$ can be computed in $O(d|S|)$ time.

5.2 Running Time

Computing a kernel density estimator with b kernels can be done in one dataset pass, during which two functions are performed:

1. Take a random sample of size b (where b is an input parameter.
2. Compute an approximation of the standard deviation for each attribute.

Kernel estimation has the very important advantage that the estimator can be computed very efficiently, in one dataset pass. Therefore, the cost of computing a kernel estimator is comparable to the cost of finding a random sample. In addition, for the dimensionalities we tried in our experimental study, it is always better to use a multi-dimensional kernel estimator rather that random sampling for selectivity estimation.

In the next section we describe a technique to adaptively set the bandwidths while still making only one dataset pass.

5.3 Adaptive kernel estimates

Current work in statistics on the problem of setting the bandwidth parameters optimally has addressed only the one dimensional problem satisfactorily [27]. Approximately optimal bandwidth parameters in the multi-dimensional case have been obtained only for the special case where the following conditions are all true: (i) the attributes are independent, (ii) the distribution on each dimension is a Gaussian and (iii) all bandwidths, in all kernels and all dimensions are to be set at the same value [27].

We present a technique to estimate the optimal locations where to place kernels, by detecting clusters in the data, i.e., locations where the data density is high. The technique also allows to adaptively tune the bandwidths of the kernels along the direction of each attribute, according to the extension of the cluster along each direction, which is in turn proportional to the local variance of the data along each direction. As a consequence, the combination of the resulting kernels should closely resemble the underlying distribution of the data.

Our approach combines clustering with nearest-neighborhood information: it clusters the sample data according to the extension (radius) of their $k-$nearest neighborhoods, that we denote with $r(N_k)$. Such extension, in fact, can be seen as a measure of the local density of the data: the smaller the $r(N_k)$ values the higher the density.

The idea consists in finding bumps in the underlying density function by detecting areas where the $r(N_k)$ values are smaller than the average of the $r(N_k)$ values computed over the whole data space. This process allows us to detect areas where the density is higher. In order to distinguish the single clusters within these areas, we partition them into regions, in which the distributions of the $r(N_k)$ values are close to uniform. Each resulting region is a rectangular box surrounding the data points in the interior.

We fit one kernel to each region by combining the kernel of each data point in the interior of the region, each of them having the bandwidth set equal to $r(N_k)$. The extensions of the regions along each dimension, and consequently the bandwidths of the corresponding kernel, along each dimension, tune to the local density of the data. Where the density is high, the region will be small and the kernel will reach a high pick. Where the density is low, the region will be larger and the kernel will be flatter and wider.

To perform this operation efficiently we first approximate the dataset using a uniform random sample. Since this sample will be discarded after the bandwidths are computed, we can afford to use a much larger sample than the number of kernels, and still perform all operations in memory. The sampling can be done in the same pass that the kernels are selected, computing the bandwidth of the kernels still requires only one pass of the dataset.

6 Experimental Results

In our experiments we want to compare the behavior of the different selectivity estimation on synthetic and real-life datasets with real valued attributes. There are three issues we want to address through the experiments.

First, one characteristic of the applications we have in mind (GIS, temporal and multimedia applications) is that attributes are also highly correlated. For example the precipitation and humidity readings in climatic data are definitely correlated attributes. This is a more interesting problem since if attributes are independent one could simply multiply one-dimensional estimators. Therefore, we created synthetic datasets that experienced significant correlations among attributes. In addition, our real-life datasets (Forest Cover and multimedia data) also have correlations among attributes.

Second, we want to evaluate the accuracy of the various methods as the dimensionality increases. We thus try datasets with 3, 4 and 5 dimensions. Interestingly, at 5 dimensions, accuracy dropped significantly for all methods in the correlated datasets we experimented with.

Third, we want to examine the behavior of the various methods when additional space for storing the estimator is available (in particular, how the accuracy of the approximation is affected by the extra space).

6.1 Techniques

We compare the new techniques (GENHIST and multi-dimensional kernels) with the following existing techniques: random sampling, one-dimensional estimation with the attribute independence assumption, wavelet transform, and MHIST-2.

Random sampling is a simple and widely used technique. In particular, we want to compare sampling against kernels, to measure the improvement we gain using kernels.

We use the Attribute Value Independence (AVI) assumption as a baseline. Any multi-dimensional technique has to be compared to AVI, to see if it is able to take into account correlations between attributes, and therefore improve performance. On the other hand, if there is no correlation among the attributes, we can measure how much we lose by using a multi-dimensional technique.

We also consider the wavelet transform, since Vitter et al. [33] show that wavelets perform better than MHIST-2 and other multi-dimensional techniques, for discrete valued attributes.

Finally, we consider MHIST-2, as the current state of the art representative of multi-dimensional histogram approaches for density estimation.

6.2 Synthetic datasets

We generate 3, 4, and 5 dimensional datasets with many clusters, and therefore significant correlations between the attributes. In addition, the distribution is non-uniform in each attribute.

We used 2 synthetic data generators:

Dataset Generator 1 creates clustered datasets (called Type 1). The number of clusters is a parameter, set to 100 in our experiments. Each cluster is defined as a hyper-rectangle, and the points in the interior of the cluster are uniformly distributed. The clusters are randomly generated, and therefore can overlap. This creates more complicated terrains. Datasets of Type 1 contain 10% to 20% uniformly distributed error.

Dataset Generator 2 is similar to the previous one, but the clusters we generate are in the $(d-1)$ or $(d-2)$-dimensional subspaces. This means that in datasets of Type 2 d-way correlation is small. Therefore these datasets present more difficulties for any algorithm that tries to approximate the joint data distribution in the d-dimensional space. Type 2 datasets contain 50 clusters and 10% to 20% uniformly distributed error.

Dataset Generator 3 is the TPC-D benchmark data. TPC-D data have been used by [33] [21], and is a popular benchmark. We use a projection of 3 to 5 numerical attributes. In particular we used the tables LINEITEM and PARTSUPP. First we perform a join between the two tables using the foreign keys and then we use the attributes QUANTITY, EXTENDEDPRICE, DISCOUNT, TAX and AVAILQTY to generate our datasets.

All datasets include 10^6 points.

6.3 Real datasets

We use the Forest Cover Dataset from the UCI KDD archive[1]. This was obtained from the US Forest Service (USFS). It includes 590000 points, and each point has 54 attributes, 10 of which are numerical. We use subsets of three, four or five numerical attributes for our experiments. In this dataset the distribution of the attributes is non-uniform, and there are correlations between pairs of attributes. We have also used multimedia datasets which have shown similar results and therefore are not reported for brevity.

6.4 Query workloads

To evaluate the techniques we created workloads of 3 types of queries. For each dataset we create a workload 1 of random queries with selectivity approximately 10%, and a workload 2, of random queries with selectivity approximately 1%. These workloads comprise 10^4 queries each. We also create a workload 3 of 20000 queries of the form $(R.A_1 < a_1) \wedge \ldots \wedge (R.A_d < a_d)$ for a randomly chosen point $(a_1, \ldots, a_d) \in [0,1]^d$.

For each workload we compute the average absolute error $\| e_{abs} \|_1$ and the average relative $\| e_{mod} \|_1$ error.

[1]available from kdd.ics.uci.edu/summary.data.type.html

6.5 Experimental comparison of the accuracy of different methods

We implemented GENHIST algorithm as described in Section 4, using a main memory hash table, to maintain statistics for every bucket. In our implementation we only consider buckets that contain more than 0.1% of the remaining points. We vary the initial value of ξ between 16 and 20. For a given value of ξ, we can use only two numbers to store a bucket. We use one number to store the location of each bucket, and another one to keep the number of tuples in the bucket.

To implement the wavelets method we followed the approach presented in [33] in our comparison experiments. We used the Haar wavelets as our wavelet basis functions. In the first step we perform a ξ regular partitioning of the data space with ξ equal to 32, and then we compute the partial sum matrix D_{ξ}^{s}. We use $\xi = 32$ because wavelets operate on arrays that are a power of 2 long, and the next larger choice, 64, creates a very large 5-dimensional partial sum array. We use the standard wavelet decomposition of a d-multidimensional array. That is, we perform a one-dimensional wavelets transform on the first dimension and we replace the original values with the resulting coefficients. Then we do the same for the second dimension, treating the modified matrix as the original matrix, and we continue up to d dimensions. We perform thresholding after normalization, that is, first we weight the wavelets coefficients, and then we keep the C most important among them (with largest absolute value). To store a coefficient we used two numbers, one to store the bucket number and the other one to store its value. We run experiments both using partial sums and using the original matrix. Our datasets are not sparse, and the partial sum method performed better. Therefore, we report the partial sum matrix results only. We obtained the code for wavelets from [13].

We ran MHIST-2, using the binary code provided by the authors [25]. We used MaxDiff as partition constraint, the attribute values as sort parameter, and frequency as source parameter in our experiments. We also tested the Area as source parameter, and obtained slightly worse results than for frequency. Therefore, we report the results obtained for frequency.

For the kernels method we used the Epanechnikov product kernel. We select a bandwidth using the Scott's rule. The storage requirements of this method is the same as sampling. That is, we store for each sample the value of each attribute (thus for 5 dimensions we used $5t$ numbers to store t samples). The results we present in experiments for kernel and sampling are averages for five different runs with randomly chosen sample sets.

Finally, we used the Attribute Value Independence (AVI) assumption as a baseline. We did not use any particular method to keep statistics for each attribute separately, but we computed the selectivity of every query in each dimension exactly. Thus the results presented here are the optimal results for this method. If we use a method to compute the selectivity for each attribute then we expect the error rates to be a higher. (However in one dimension there are methods with high accuracy, so the error is not expected to be much higher, given of course that we use sufficient space).

We performed an extensive study to evaluate the performance of different methods for 3-, 4-, and 5-dimensional data (except for MHIST-2 since the available binary could run for up to 4 dimensions). For 3-dimensional datasets there were small differences in accuracy between the techniques. Interesting changes started appearing at 4 and then 5 dimensions and are described below.

We first present the performance of all six methods for 4-dimensional data. We show the DS1 (type 1), TPC-D (type 3), and the Forest Cover datasets, using query workload 1 and 3 on each one. We plot the 1-norm average relative error for each method for different values of the available storage space to store the estimator. Figures 5-7 show the results for the three 4-dimensional datasets for query workload 1 (10% queries). Similarly, Figures 8-10 show the results for query workload 3 (queries anchored to the origin). All techniques perform worse in 4 dimensions than in 3 dimensions. The largest drop in overall accuracy is for MHIST-2.

For the experiments in 5 dimensions, we show four datasets ($DS1$, $DS2$, TPC-D, and Forest Cover dataset) using query workloads 1, 2 and 3 on each one. The difference between $DS1$ and $DS2$ is that the clusters cover less space and therefore are comparatively denser in $DS1$.

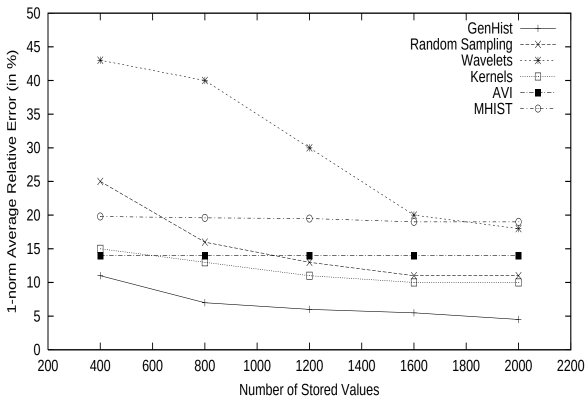

Figure 5: DS1 dataset, query workload 1, 4-dim.

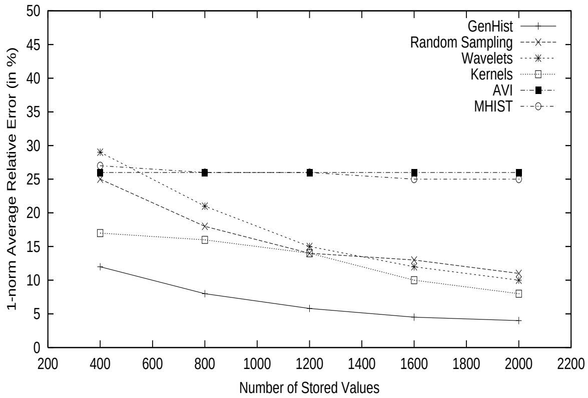

Figure 6: Forest Cover dataset, query workload 1, 4-dim.

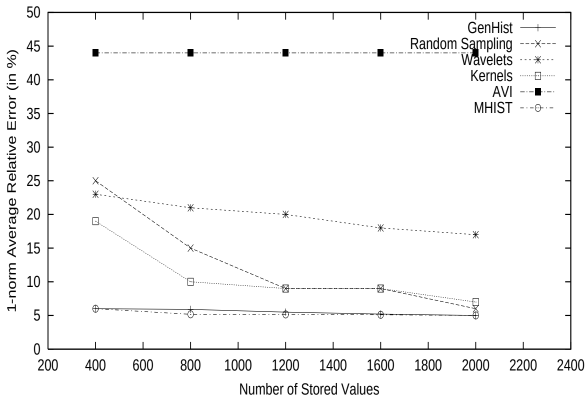

Figure 7: TPC-D dataset, query workload 1, 4-dim.

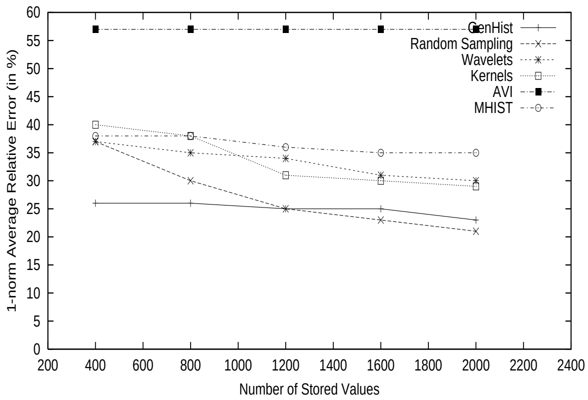

Figure 8: TPC-D dataset, query workload 3, 4-dim.

In Figures 11-22 we show the results. We plot the 1-norm average relative error for each method for different values of the available storage space to store the estimator. Figures 11-14 (10% queries) show the results for the four 5-dimensional datasets for query workload 1. Similarly, Figures 15-18 show the results for query workload 2 (1% queries) and Figures 19-22 show the results for query workload 3.

In general the GENHIST method performs better than all other methods when the available space increases. For the TPC-D dataset on query workload 3 and for the Forest Cover dataset on query workload

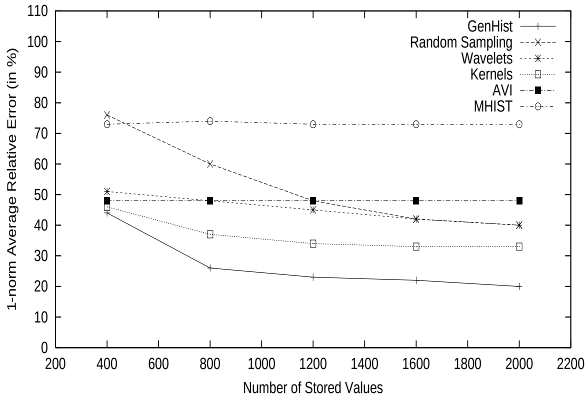

Figure 9: DS1 dataset, query workload 3, 4-dim.

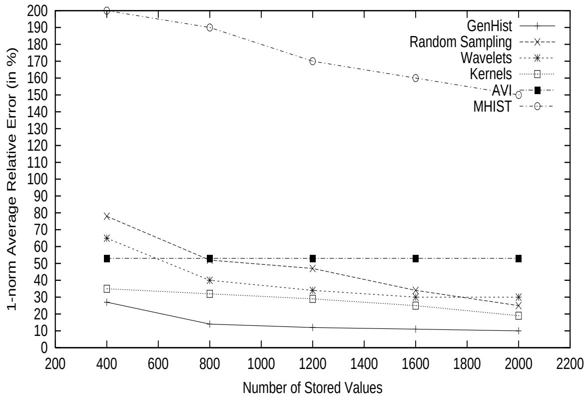

Figure 10: Forest Cover dataset, query workload 3, 4-dim.

1, multi-dimensional kernels are more accurate. Multi-dimensional kernels generally perform well. An interesting observation is that kernels perform better than random sampling for all datasets.

Wavelets perform better than the other methods when the space used to store the approximation is very small. But increasing the available space (that is, using more wavelets coefficients to approximate the density function) offers small improvement. The reason is that after some point, which is reached quickly in our experiments, many coefficients have approximately the same magnitude. Thus, using more coefficients gives little help.

The results for query workloads 1 and 2 can be used to evaluate the impact of the query size on the accuracy of the selectivity estimators (Figures 11-18). Clearly the relative error rate increases when the query size decreases. This is to be expected from the definition of the relative error. Even a small difference in absolute terms between the estimated and the actual query size can lead to a large relative error if the actual query size is small. Thus the performance of all methods decreases significantly from workloads 1 to workloads 2.

It is clear that in 5 dimensions most methods do not offer very high accuracy, for small queries in particular. GENHIST, the most accurate of the methods we tested, offers an accuracy of 20% to 30% for queries of size 1%. In addition, the curves for all methods are rather flat, so accuracy is unlikely to increase a lot even if we allocate much more space.

However, all meaningful queries in high dimensions are likely to be small. For example, in six dimensions a range query of the form $(R.A_1 < 0.5) \wedge \ldots \wedge (R.A_6 < 0.5)$ only covers $\frac{100}{2^6}\% \approx 2\%$ of the space. We consider 5 dimensions to be close to the limit at which we can still expect an accurate estimation to the selectivity problem.

In another set of experiments we consider how the increase on the dimensionality affects the accuracy of GENHIST. In Figures 23 and 24 we plot the average relative error for GENHIST, for 3, 4 and 5 dimensions, as a function of the space used for dataset $DS1$ and workloads 1 and 3. The results, not surprisingly, show the significant degradation in the accuracy that accompanies the increase in space dimensionality.

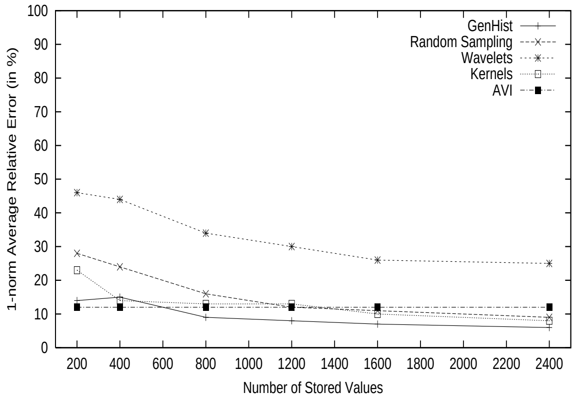

Figure 11: DS1 dataset, query workload 1, 5-dim.

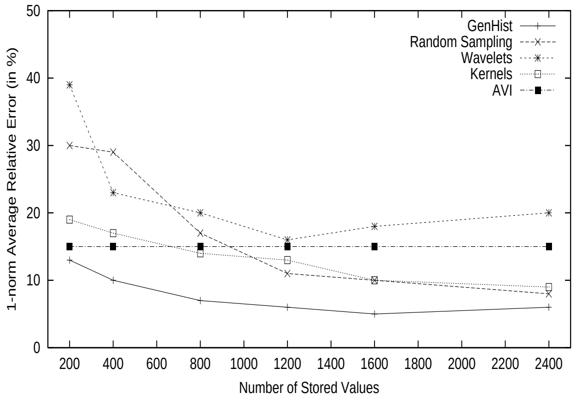

Figure 12: DS2 dataset, query workload 1, 5-dim.

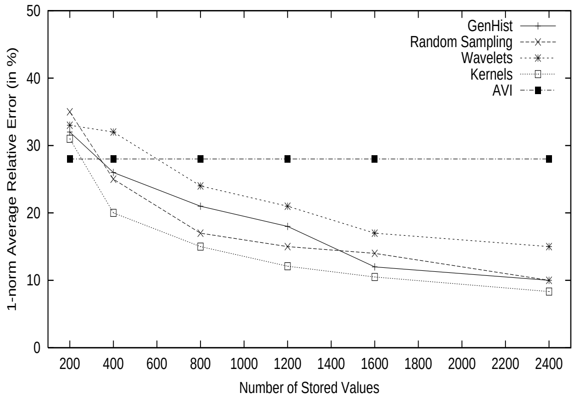

Figure 13: Forest Cover dataset, query workload 1, 5-dim.

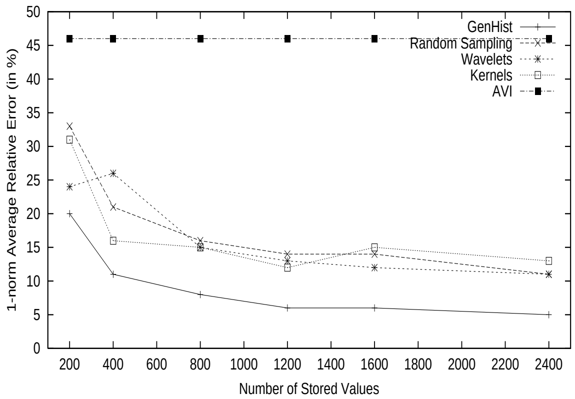

Figure 14: TPC-D dataset, query workload 1, 5-dim.

6.6 Comparison of running times

Random sampling and kernels require only one pass through the data where GENHIST requires 5 to 10 passes. Therefore a drawback of GENHIST compared to random sampling and kernel estimators, is that the construction time is larger. However, for the kernel estimators is not very easy to tune the bandwidth and it may require additional time to achieve a good value for this. The construction time for the wavelets estimators is much larger because in the worst case matches the time for external memory sorting.

The time to compute the selectivity of a range query is approximately the same for most methods, since we have to test the query against all buckets in GENHIST or sample points in random sampling and kernels. Even for a multi-dimensional histogram with non overlapping buckets, a d-dimensional range query can intersect $O(b^{\frac{d-1}{d}})$ buckets (where b is the number of buckets). However for the wavelets approach this time is larger because we have to compute the inverse transform first.

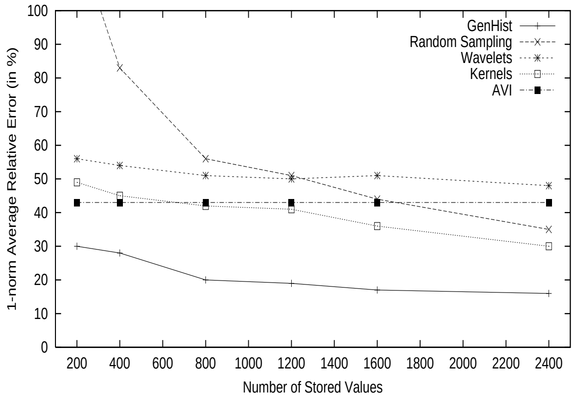

Figure 15: DS1 dataset, query workload 2, 5-dim.

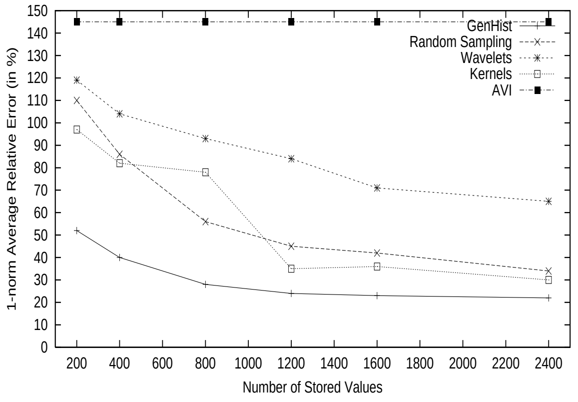

Figure 16: DS2 dataset, query workload 2, 5-dim.

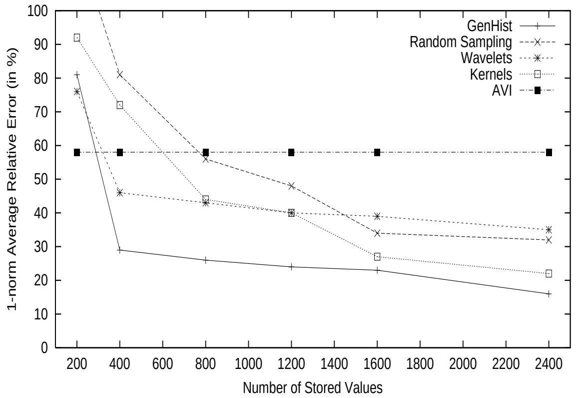

Figure 17: Forest Cover dataset, query workload 2, 5-dim.

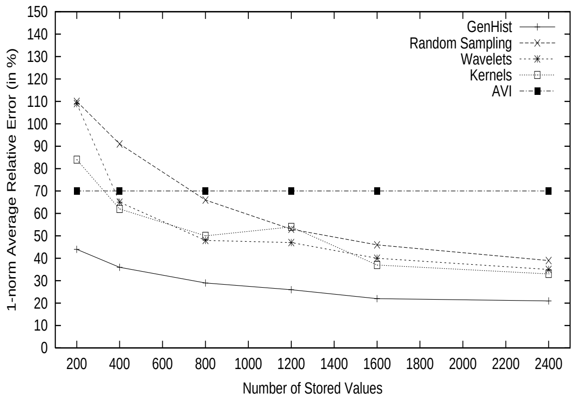

Figure 18: TPC-D dataset, query workload 2, 5-dim.

7 Conclusions

In this paper we have addressed the problem of estimating the selectivity of a multi-dimensional range query when the query attributes have real domains and exhibit correlations. In this environment each value appears very infrequently. Most of previous work considers only discrete-valued attributes with a small set of different values, which implies that the frequency of each value is high.

The contributions of the paper are: (1) We propose a new generalized histogram technique GENHIST to solve the problem. GENHIST differs from earlier partitioning techniques because it uses overlapping buckets of varying sizes. (2) For the same problem we generalize a kernel estimator technique to many dimensions. (3) We perform an experimental study to evaluate and compare the GENHIST technique and the multi-dimensional kernel estimators over real attributes, with a number of existing techniques: attribute independence assumption, wavelet decomposition, MHIST-2 and sampling.

Conclusions we can draw from our experimental results include: (1) GENHIST typically outperforms other techniques in the range of space dimensionality (3 to 5) that we run experiments on. GENHIST can be

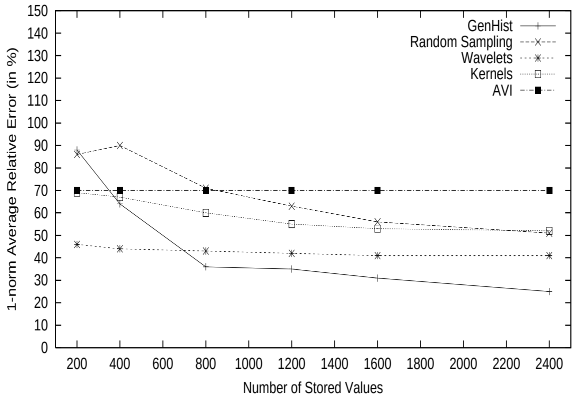

Figure 19: DS1 dataset, query workload 3, 5-dim.

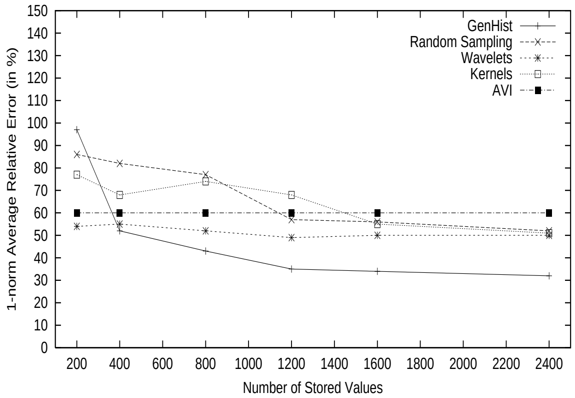

Figure 20: DS2 dataset, query workload 3, 5-dim.

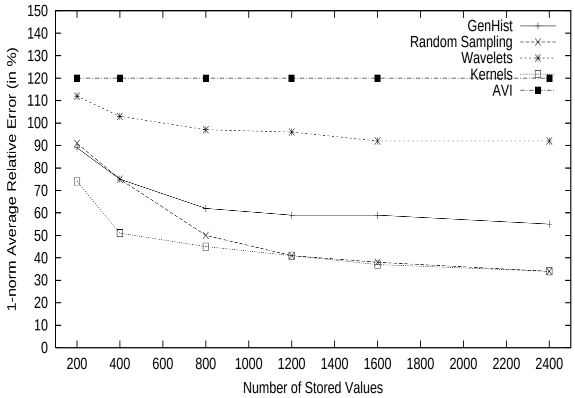

Figure 21: Forest Cover dataset, query workload 3, 5-dim.

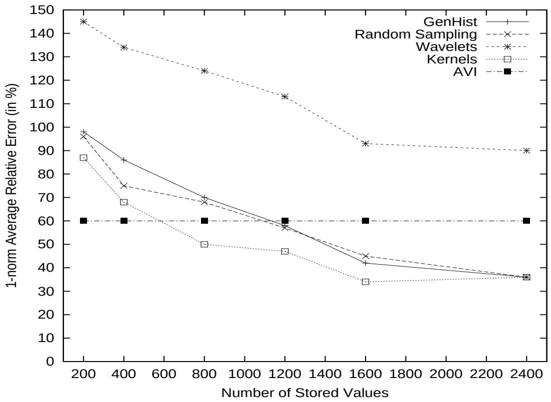

Figure 22: TPC-D dataset, query workload 3, 5-dim.

thought as a multi-dimensional histogram that allows for overlapping partitioning. The experiments show that overlapping partitioning shows an improvement over non-overlapping partitioning.

(2) Multi-dimensional kernel estimators offer good accuracy, and very fast construction time. The kernel estimator approach outperformed pure sampling in all our experiments.

(3) For the real-valued and correlated datasets we have used, the accuracy of all techniques decreases when dimensionality increases. However some of the techniques we examine (and in particular GENHIST and the multi-dimensional kernels) can be used effectively in 5-dimensional spaces.

In future work we plan to perform experiments for higher dimensions. It is not clear how GENHIST, multi-dimensional histograms, wavelets and other decomposition techniques, and kernels will perform relative to each other or relative to sampling as the dimensionality increases. However we do not expect any technique to perform effectively when the dimensionality of the space approaches 10. As evidence for this, [34] reports that it is difficult to achieve good accuracy with kernel estimators when the dimensionality is larger than 5. We conjecture that sampling will outperform any of these techniques for dimensionality of around 10, but that the error will be too large to make the technique practical.

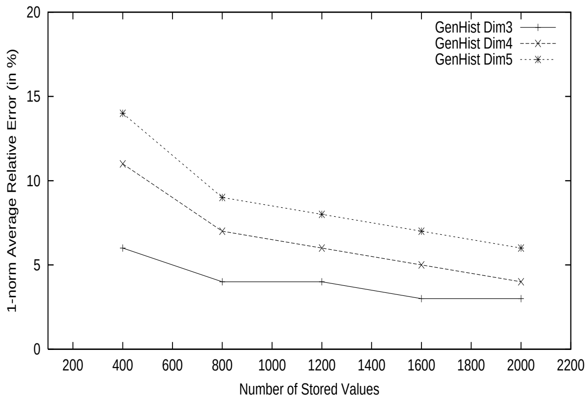

Figure 23: DS1 dataset, query workload 1, 3 to 5 dim, GENHIST.

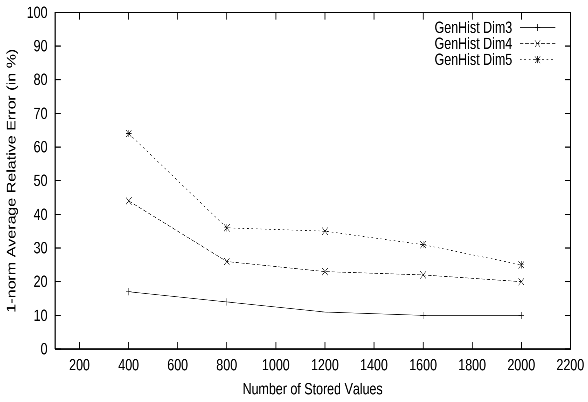

Figure 24: DS1 dataset, query workload 3, 3 to 5 dim, GENHIST.

An interesting problem is to compare how the various query estimators are maintained under different update loads. Updating in random sampling can be achieved using techniques from [10]. Such techniques can be extended to apply to kernel estimators, too. Most work for maintaining histograms has concentrated on the one dimensional case [10], although recently [1] proposed a technique for maintaining multi-dimensional histograms. Maintaining GENHIST is similar to maintaining other multi-dimensional histograms: an insertion or deletion will affect only one bucket. In particular for GENHIST if the updated point is in the interior of more than one bucket we chose to update the bucket that is the smallest in size.

8 Acknowledgement

We would like to thank Johannes Gehrke for providing the code of a dataset generator, and Vishy Poosala for providing the binary for the MHIST algorithm.

References

[1] A. Aboulnaga, S. Chaudhuri. Self-tuning Histograms: Building Histograms Without Looking at Data. in *Proc. of the 1999 ACM SIGMOD Intern. Conf. on Management of Data*, June 1999.

[2] S. Acharya, V. Poosala, S. Ramaswamy. Selectivity Estimation in Spatial Databases. in *Proc. of the 1999 ACM SIGMOD Intern. Conf. on Management of Data*, June 1999.

[3] B. Blohsfeld, D. Korus, B. Seeger. A Comparison of Selectivity Estimators for Range Queries on Metric Attributes. In *Proc. of the 1999 ACM SIGMOD Intern. Conf. on Management of Data*, June 1999.

[4] S. Chaudhuri and L. Gravano Evaluating Top-K Selection Queries. In *Proc. 25th Intern. Conf. on Very Large Data Bases (VLDB-99)*, Edinburgh, Scotland, Sept. 1999.

[5] S. Chaudhuri, R. Motwani, V.R. Narasayya. Random Sampling for Histogram Construction: How much is enough? In *Proc. of the 1998 ACM SIGMOD Intern. Conf. on Management of Data*, June 1998.

[6] N. A.C. Cressie. *Statistics For Spatial Data.* Wiley & Sons, 1993.

[7] P.J. Digglediggle A kernel method for smoothing point process data. *Applied Statistics*, 34, 138-147.

[8] D. Donjerkovic and R. Ramakrishnan. Probabilistic Optimization of Top N Queries. In *Proc. 25th Intern. Conf. on Very Large Data Bases (VLDB-99)*, Edinburgh, Scotland, Sept. 1999.

[9] P.B. Gibbons, Y. Matias. New Sampling-Based Summary Statistics for Improving Approximate Query Answers. In *Proc. of the 1998 ACM SIGMOD Intern. Conf. on Management of Data*, June 1998.

[10] P.B. Gibbons, Y. Matias and V. Poosala. Fast Incremental Maintenance of Approximate Histograms. In *Proc. of 23rd Intern. Conf. on Very Large Data Bases*, August 1997.
[11] P.J. Haas, A.N. Swami. Sequential Sampling Procedures for Query Size Estimation. In *Proc. of the 1992 ACM SIGMOD Intern. Conf. on Management of Data*, June 1992.
[12] J.M. Hellerstein, P.J. Haas, H. Wan. Online Aggregation. In *Proc. of the 1997 ACM SIGMOD Intern. Conf. on Management of Data*, May 1997.
[13] Imager Wavelet Library. www.cs.ubc.ca/nest/imager/contributions/bobl/wvlt/top.html
[14] Y. Ioannidis and V. Poosala. Histogram-Based Approximation of Set-Valued Query-Answers. In *Proc. 25th Intern. Conf. on Very Large Data Bases (VLDB-99)*, Edinburgh, Scotland, Sept. 1999.
[15] H. V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K.C. Sevcik, T. Suel. Optimal Histograms with Quality Guarantees. In *Proc. of 24rd Intern. Conf. on Very Large Data Base*, August 1998.
[16] S. Khanna, S. Muthukrishnan, M. Patterson. On Approximating Rectangle Tiling and Packing. In *Proc. of 9th Annual Symp. on Discrete Algorithms (SODA)*, 1998.
[17] A. Konig and G. Weikum Combining Histograms and Parametric Curve Fitting for Feedback-Driven Query Result-size Estimation. In *Proc. 25th Intern. Conf. on Very Large Data Bases (VLDB-99)*, Edinburgh, Scotland, Sept. 1999.
[18] F. Korn, T. Johnson and H. Jagadish. Range Selectivity Estimation for Continuous Attributes. In *Proc. of 11th Int. Conf. on SSDBMs*, 1999.
[19] R.J. Lipton, J.F. Naughton. Practical Selectivity Estimation through Adaptive Sampling. In *Proc. of the 1990 ACM SIGMOD International Conference on Management of Data*, May 1990.
[20] J. Lee, D. Kim and C. Chung. Multi-dimensional Selectivity Estimation Using Compressed Histogram Information. in *Proc. of the 1999 ACM SIGMOD Intern. Conf. on Management of Data*, June 1999.
[21] Y. Matias, J. Scott Vitter, M. Wang. Wavelet-Based Histograms for Selectivity Estimation. In *Proc. of the 1998 ACM SIGMOD Intern. Conf. on Management of Data*, June 1998.
[22] M. Muralikrishna, D.J. DeWitt. Equi-Depth Histograms For Estimating Selectivity Factors For Multi-Dimensional Queries. In *Proc. of the 1988 ACM SIGMOD Intern. Conf. on Management of Data*, June 1988.
[23] F. Olken, D. Rotem. Random Sampling from database Files:A Survey. In *Proc. of 5th International Conference on Statistical and Scientific Database Management*, Charlotte, N.C., 1990.
[24] V. Poosala, V. Ganti. Fast Approximate Answers to Aggregate Queries on a Data Cube. In 11th Intern. Conf. on Scientific and Statistical Database Management, Cleveland, 1999.
[25] V. Poosala and Y.E. Ioannidis. Selectivity Estimation Without the Attribute Value Independence Assumption. In *Proc. of 23rd Intern. Conf. on Very Large Data Bases*, August 1997.
[26] V. Poosala, Y.E. Ioannidis, P. J. Haas, E.J. Shekita. Improved Histograms for Selectivity Estimation of Range Predicates In *Proc. of the 1996 ACM SIGMOD Intern. Conf. on Management of Data*, May 1996.
[27] D. Scott. *Multivariate Density Estimation: Theory, Practice and Visualization.* Wiley & Sons, 1992.
[28] P.G. Selinger, M.M. Astrahan, D.D. Chamberlin, R.A. Lorie, T.G. Price. Access Path Selection in a Relational Database Management System. In *Proc. of the 1979 ACM SIGMOD Intern. Conf. on Management of Data*, June 1979.
[29] Jayavel Shanmugasundaram, Usama Fayyad, Paul Bradley. Compressed Data Cubes for OLAP Aggregate Query Approximation on Continuous Dimensions In *Fifth ACM SIGKDD Inter. Conf. on Knowledge Discovery and Data Mining*, August 1988.
[30] B.W. Silverman. Density Estimation for Statistics and Data Analysis. *Monographs on Statistics and Applied Probability*, Chapman & Hall 1986.
[31] TPC benchmark D (decision support), 1995.
[32] J. S. Vitter and M. Wang. Approximate Computation of Multidimensional Aggregates of Sparse Data Using Wavelets. in *Proc. of the 1999 ACM SIGMOD Intern. Conf. on Management of Data*, June 1999.
[33] J.S. Vitter, M. Wang, B. R. Iyer. Data Cube Approximation and Histograms via Wavelets. In *Proc.of the 1998 ACM CIKM Intern. Conf. on Information and Knowledge Management*, November 1998.
[34] M.P. Wand and M.C. Jones. Kernel Smoothing. *Monographs on Statistics and Applied Probability*, Chapman & Hall 1995.
[35] R. Webber, H.-J. Schek and S. Blott. A Quantitative Analysis and Performance Study for Similarity Search Methods in High-Dimensional Spaces. In *Proc. of 24rd Intern. Conf. on Very Large Data Base*, August 1998.