Selectivity Estimators for Multi-Dimensional Range Queries over Real Attributes

Dimitrios Gunopulos *
George Kollios †
Vassilis J. Tsotras ‡

Technical Report UCR-CS-99-02
May 26th, 1999

Abstract

Estimating the selectivity of range queries is useful in data exploration and database query optimization. In this paper we consider the problem of estimating the selectivity of multi-dimensional range queries on attributes with real domains. Database approaches to solve this problem include the construction of multi-dimensional histograms, random sampling, or the construction of wavelets. In statistics, kernel estimation techniques are being used. Much of the work so far considers only numerical attributes with a small set of distinct possible values.

In this paper we give a new multi-dimensional histogram technique, GENHIST, that is designed to approximate the data distribution of real attributes more effectively than existing techniques. Our technique uses overlapping buckets with variable sizes. The size of the buckets is based on the local density of the data, and leads to a faster and more compact approximation of the data distribution. We evaluate and compare this and existing approaches. We present experiments using synthetic and TPC-D datasets with real attributes. The experiments show that our technique is more effective in higher dimensionalities than other techniques.

1 Introduction

Computing approximate answers to range queries is a problem that arises in database query optimization and data warehousing. The query optimizer requires accurate estimations of

*CS Dept., Univ. of California Riverside, Riverside, CA 92521, dg@cs.ucr.edu
†CS Dept., Polytechnic Univ., Brooklyn, NY 11201, gkollios@cs.ucr.edu.
‡CS Dept., Univ. of California Riverside, Riverside, CA 92521, tsotras@cs.ucr.edu.

the sizes of intermediate query results in the evaluation of different execution plans. 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 [21] that the query time is linear to the size of the dataset.

We should emphasize here that this problem is different from the traditional definition of selectivity estimation. In related work so far it has been generally assumed that each numerical attribute has a finite discrete domain of small cardinality, that is, each attribute can take one of a small number of distinct values. In many applications however the attribute domain is the real numbers, for example when we have spatial or temporal dimensions. This has two important consequences: First, the number of possible queries is infinite in the case of real domains, but finite in the case of 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, it is unlikely that many attribute values will appear more than once in the database in the case of real domains. 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 [16], using the wavelet transformation [20] [13], SVD [16] or the discrete cosine transform [12] on the data, using kernel estimators [1], and sampling [15] [11] [6].

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 hap-

pens, 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 dimension, 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. 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 [18] 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

We present a new technique, GENHIST, to find multi-dimensional histograms for datasets from real domains. The method has been designed to approximate the data distribution more efficiently than existing techniques. Like other histogram techniques, our technique uses grids to approximate the data distribution. It computes an approximation of the data distribution in the space, and then uses a constant number of buckets to approximate the data distribution. The basic characteristic in our technique is that the buckets can be of varying size, and 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.

We evaluate GENHIST and existing techniques for estimating the selectivity of multi-dimensional range queries for real attributes. The existing techniques we consider are the wavelet transform [20], multi-dimensional histograms [16], one-dimensional estimation techniques with the attribute independence assumption, multi-dimensional kernel estimators [1], and sampling [6].

We implemented and run experiments on wavelet transformations, multi-dimensional kernels, and sampling. Both the wavelet transform [20] and the histogram techniques have been described in the context of finite discrete attribute domains. To use them in our context we discretize the attribute domains. We also include the attribute independence assumption in our study as a baseline comparison.

The experimental results show that GENHIST is more accurate than existing techniques. The experiments also show that current selectivity estimation techniques work well only when the dimensionality of the space is low. In the accuracy results the error rates rise quickly with the increase of the dimensionality and this suggests that most techniques can be used effectively for dimensionalities of around 5. In addition we show that kernel estimators work better than sampling for the dimensionalities we tried, although sampling becomes more competitive as the dimensionality increases.

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 known statistical methods such as sampling or kernel estimators for multi-dimensional data in sec-

tion 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 [20]).

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 [20] 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:

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

3 Multi-dimensional Density Estimators

In this section we briefly examine existing techniques to estimate the selectivity of a query. We group them in two categories, histograms and discrete decomposition techniques.

Most of the techniques we discuss work for attributes with discrete finite domains only. To use real domain attributes it is necessary to discretize the attributes. Hence partition the domain of each attribute into ξ non-overlapping intervals. The width of each interval is $1/\xi$, so that we have an equi-width partitioning. This partitioning results into ξ^d d-dimensional non-overlapping buckets that cover the entire data space; we call it the ξ regular partitioning of the 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 can then use any of the existing techniques, such as multi-dimensional histograms or the wavelet transform, to obtain an approximation for R'.

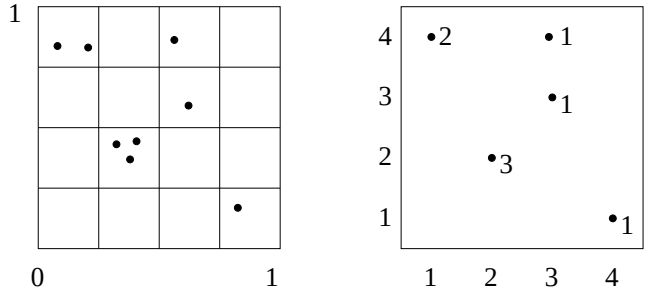

Figure 1: A 4-regular partitioning of a 2-dimensional dataset and the tuple frequencies.

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 4ξ 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.

3.1 One-dimensional Histograms

In system R [18], density estimators for each attribute were 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 project the query on each attribute and estimate the selectivity of each one-dimensional query, and then multiply the selectivities. Typically one-dimensional histograms are used. This technique is still widely employed.

3.2 Multi-dimensional Histograms

Multi-dimensional histograms were introduced by [14]. 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 [10], although the problem is easy to solve in one dimension (in an equi-sum histogram the sum of the tuples in each bucket is approximately the same).

[16] presents two new algorithms, PHASED and MHIST-2, that attempt to produce MaxDiff histograms. In a MaxDiff histogram there is a bucket boundary between two values if the difference of the frequencies of the two values is large. Therefore MaxDiff minimizes the variance of frequencies of values within each bucket. Of the two techniques, MHIST-2 is shown to be more accurate and performs better than previous approaches [16]. In MHIST-2, the criterion used for splitting an area into different bucket takes the frequency distribution of each attribute into account.

Both algorithms can be used directly on a dataset with real valued attributes. However when we use real data all the different values that exist in the dataset have frequency of 1 because each value that appears, appears exactly once. It is unlikely that good splits can be found for such a frequency distribution. To solve this problem we must use a ξ regular

partitioning of the space.

3.3 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 to 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) [16], the wavelet transform [20] and recently the Discrete Cosine Transform (DCT) [12].

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 rest of the coefficients are set to zero. This results to 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.

From the three techniques we mention, SVD can only be used in two dimensions ([16]). Wavelets and, recently, DCT has been shown to give good results in high dimensionalities [20] [13] [12]. Due to time constraints we only include the wavelet transform in our comparisons.

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, but we perform a ξ regular partitioning of the data space first. We call the resulting ξ^d size matrix D_ξ.

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

$$D_\xi^s[i_1, \ldots, i_d] = \sum_{j_1 \leq i_1, \ldots, j_d \leq i_d} D_\xi[j_1, \ldots, j_d]$$

We run 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. Also [20] suggests that partial sum method is more accurate because this operation smoothes up the data distribution. Therefore we report the partial sum matrix results in our comparison experiments.

4 A New Multi-Dimensional Histogram Construction

In this section we present a new density estimation technique, GENHIST (for GENeralized HISTograms). Like previous histogram techniques 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.

All histogram techniques 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 larger, 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 before 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 the area of their intersection we assume that the data density is the sum of the two densities. If more that two buckets overlap, the data density in the area of the intersection 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 only

count it in the computation of the average density of one of the buckets only.

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. As a result the advantage of overlapping buckets increases with the dimensionality.

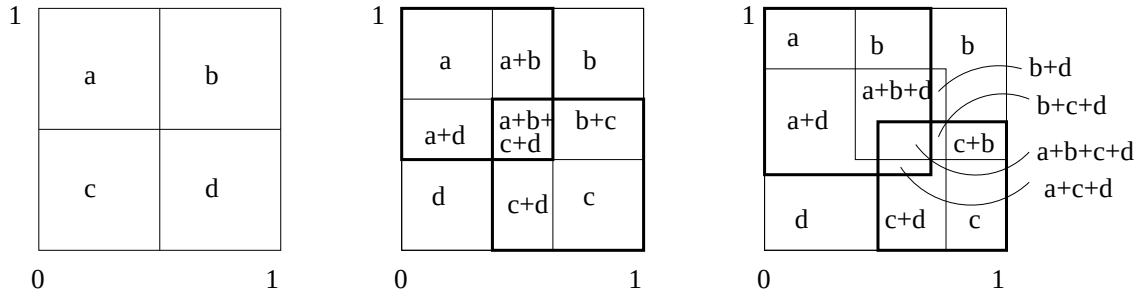

Figure 2: The same 2-dimensional space is partitioned first using 4 non overlapping buckets (average densities a, b, c and d), then, using 4 overlapping but equal-sized buckets, into 9 regions (densities are a, b, c, d, a+b, b+c, c+d, d+a, a+b+c+d), and finally into 13 regions, using 4 overlapping unequal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

Estimating 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.

4.1 Generalized multi-dimensional histograms

We now give a heuristic approach to partition a d-dimensional space using b overlapping buckets of different sizes.

The algorithm iteratively computes an approximation of the density function using a grid, by finding how many tuples fall in the interior of each bucket. Buckets with high count are in areas where the density is large. The largest are kept in the estimator, and the number of tuples in their interior is reduced. Tuples are removed at random to ensure that the density decreases uniformly to the level of the density of neighbor buckets. Consequently each iteration results to an overall smoothing of the data distribution. We can then use larger buckets to approximate the smother data distribution in the next iteration.

The description of the algorithm includes a number of input parameters. These are 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 result of the algorithm is a set of buckets E with their average density. This set can be used as a density estimator for R.

The outline of the algorithm GENHIST follows.

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.

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 can be difficult. 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 detail 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 $1/\log_{\frac{1}{\alpha}} \xi$.

The running time of the algorithm is linear to 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.

5 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 [18] [2] [5] or for on-line aggregation [7]. Sampling can be used to estimate the selectivity of a query regardless of the dimensionality of the space and can be applied to real domains as is.

More sophisticated statistical techniques 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.

In the problem of computing the query selectivity, all the proposed techniques compute a density estimation function. Such function can be thought of as an approximation to an appropriate probability distribution function of which the dataset at hand is an instance. It follows that statistical techniques, such as kernel estimation, that approximate the probability distribution are applicable in the query estimation problem. Recently [1] introduced kernels to the database community and used them to estimate the selectivity of range queries on metric attributes. At the time of this writing we didn't have access to [1] so we follow [3] [4] in the implementation of kernel estimators.

Kernel estimation is a generalized form of sampling. Each sample point has weight one, but it distributes its weight in the space around it. A kernel function describes the form of the weight distribution.

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 [3]. What is important is the standard deviation of the function, or, its bandwidth. So we choose a kernel function that it is easy to integrate. The Epanechnikov kernel function has this property [3]. The d-dimensional Epanechnikov kernel function centered at 0 is:

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

if, for all i, $|\frac{x_i}{B_i}| < 1$, and 0 otherwise. 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. We initially set all B_i equal to 0 (in this case kernel estimation degenerates to sampling). We then perform a number of passes progressively increasing the bandwidth and checking whether the approximation accuracy increases. We stop at a local minimum.

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 closer to the space boundary than its bandwidth, 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 $\int_{[0,1]^d} f(x_1, \ldots, x_d)dx_1 \ldots dx_d < 1$. To solve it 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.

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

6 Experimental Results

In this section we report the results of our experimental comparison of the different selectivity estimation methods. The performance of all methods was compared using 5-dimensional

datasets. For every technique we examine the effect that additional available space for storing the estimator has to the accuracy of the approximation. In addition we evaluate how the dimensionality increase affects the accuracy of GENHIST, by using 3-, 4- and 5-dimensional datasets.

To evaluate the techniques we created datasets using 4 different distributions and we created workloads of 3 types of queries. For each workload we compute the average absolute error $\| e^{abs} \|_1$ and the average relative $\| e^{mod} \|_1$ error.

6.1 Datasets

We generated datasets in 3, 4, and 5 dimensions. All datasets include 10^6 points.

We used 2 synthetic data generators and TPC-D benchmark data [19].

Dataset Generator 1 creates clustered datasets. We call these datasets 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 are harder 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. 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.

6.2 Query workloads

We evaluate the techniques using random queries of fixed size, and random queries that are rectangles anchored to the origin. 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 $(a_1, \ldots, a_d) \in [0,1]^d$.

6.3 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 [20]. 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 16 and then we compute the partial sum matrix D^s_ξ. We use 16 because wavelets operate on arrays that are a power of 2 long, and the next larger choice, 32, 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 obtained the code for wavelets from [8].

For the kernels method we used the Epanechnikov product kernel. To set the bandwidth we start form 0 and increase the bandwidths until we reach a local optimal. In experiments we needed usually only a few iterations. In figure 19 we plot the relative error of the kernel method as a function of the bandwidth for a specific dataset and query workload.

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).

First we perform an extensive study to evaluate the performance of different methods for 5-dimensional data. We use four data sets and apply query workloads 1,2, 3 on each one. The two data sets $DS1$ and $DS2$ are of type 1, dataset $DS3$ is of type 2 and TPC-D is a type 3 dataset. The difference between $DS1$ and $DS2$ is that the clusters cover less space and therefore are comparatively denser in $DS1$.

In figures 3-14 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 3-6 show the results for the four 5-dimensional datasets for query workload 3 (queries anchored to the origin). Similarly figures 7-10 show the results for query workload 1 (10% queries) and figures 6.3-14 show the results for for query workload 3 (1%).

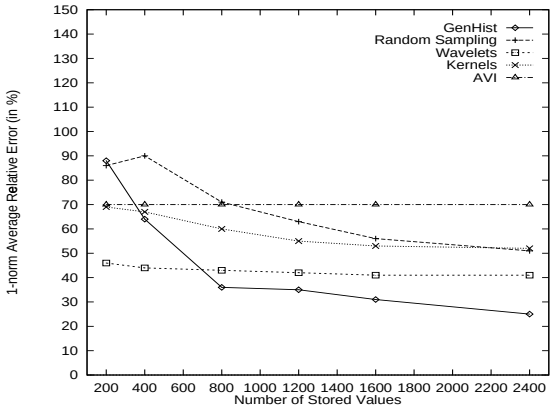

Figure 3: DS1 dataset, query workload 3, 5-dim, all methods.

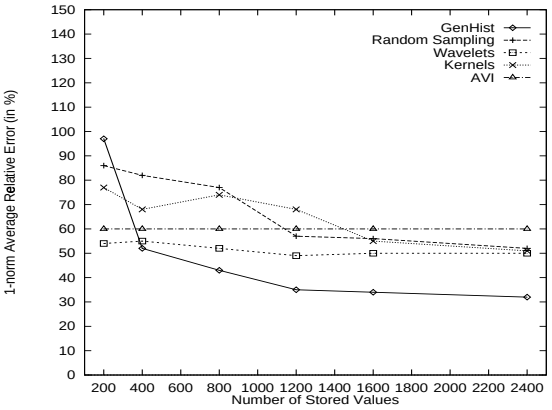

Figure 4: DS2 dataset, query workload 3, 5-dim, all methods.

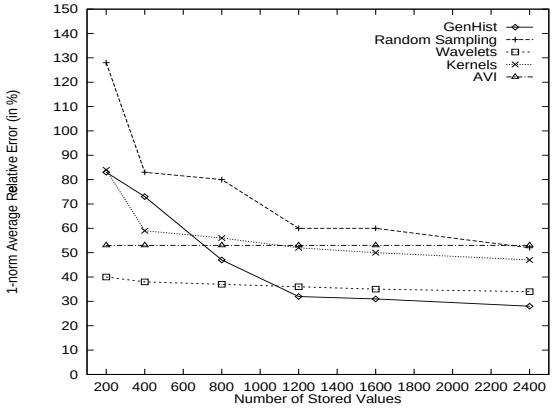

Figure 5: DS3 dataset, query workload 3, 5-dim, all methods.

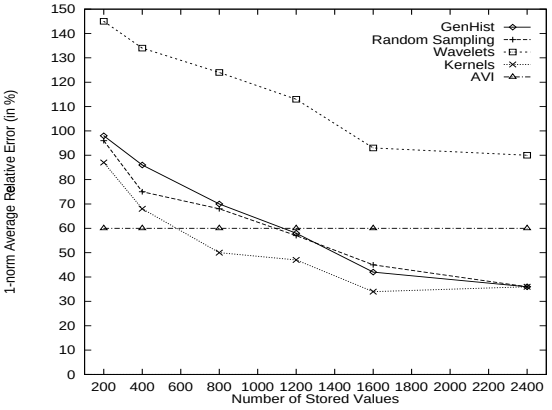

Figure 6: TPC-D dataset, query workload 3, 5-dim, all methods.

In general the GENHIST method performs better than all other methods when the available space increases. Only for the TPC-D dataset and query workload 3 is another method (kernels) more accurate.

Kernels also 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 very small improvement. The reason is that after some point, which is reached quickly in our experiments, many coefficients have approximately the same non-negligible magnitude. Thus using some more coefficients gives little help. We also note that wavelets do not perform well for the TPC-D dataset for query

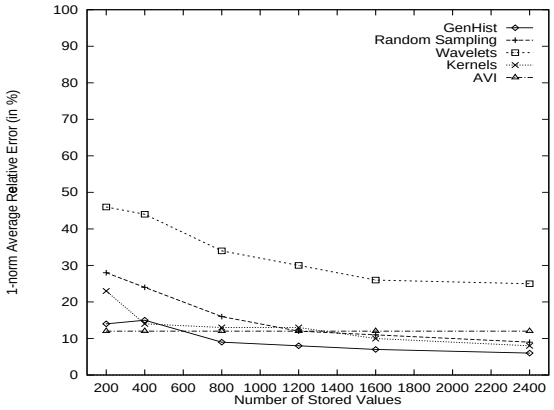

Figure 7: DS1 dataset, query workload 1, 5-dim, all methods.

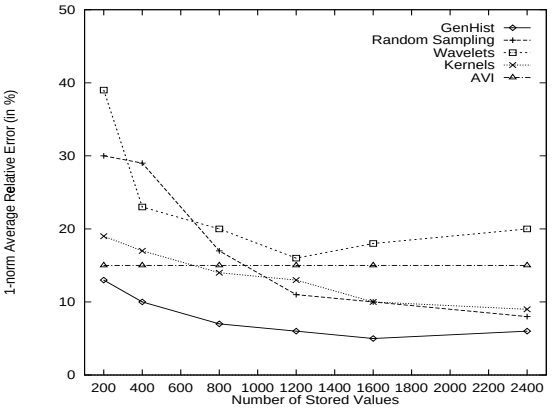

Figure 8: DS2 dataset, query workload 1, 5-dim, all methods.

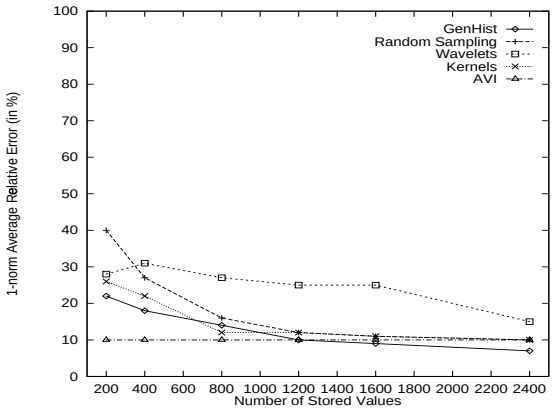

Figure 9: DS3 dataset, query workload 1, 5-dim, all methods.

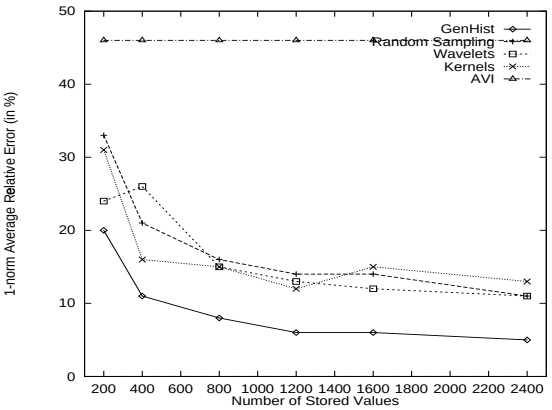

Figure 10: TPC-D dataset, query workload 1, 5-dim, all methods.

workload 3.

Note that AVI performs reasonably well for the $DS3$ dataset. The reason is that this data set has clusters in lower dimensions, so there is a weak 5-way correlation. In fact for large queries (figure 9) AVI performs better than other methods (except GENHIST here). It is clear that we have to be careful when we approximate the selectivity of multi-dimensional range queries. If the attributes are not correlated, using the AVI assumption can give better results than using a more sophisticated method.

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 7-14). 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

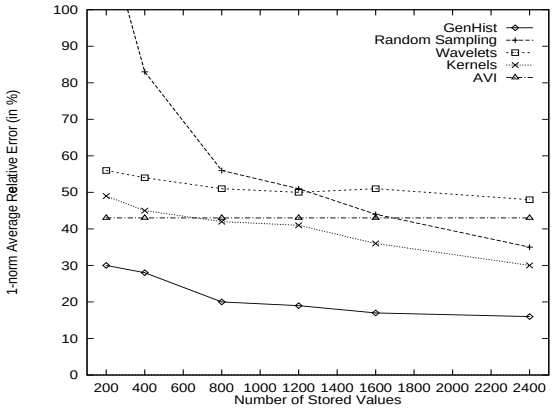

Figure 11: DS1 dataset, query workload 2, 5-dim, all methods.

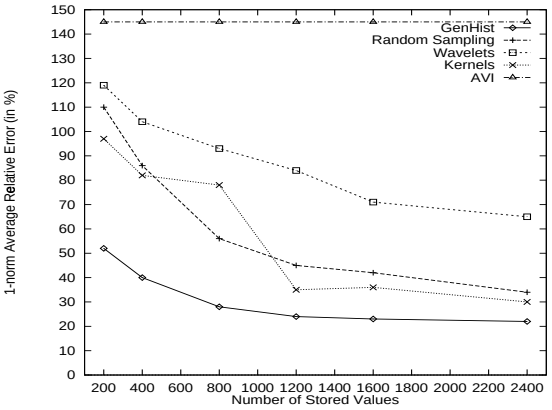

Figure 12: DS2 dataset, query workload 2, 5-dim, all methods.

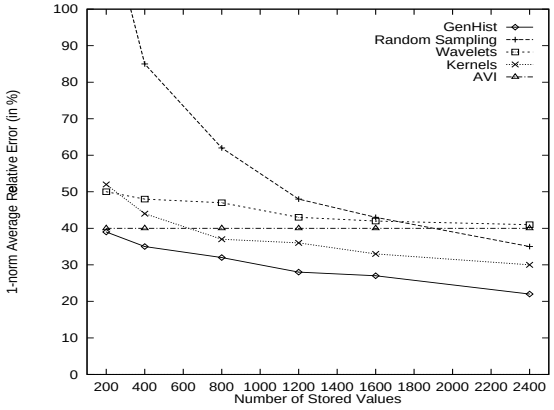

Figure 13: DS3 dataset, query workload 2, 5-dim, all methods.

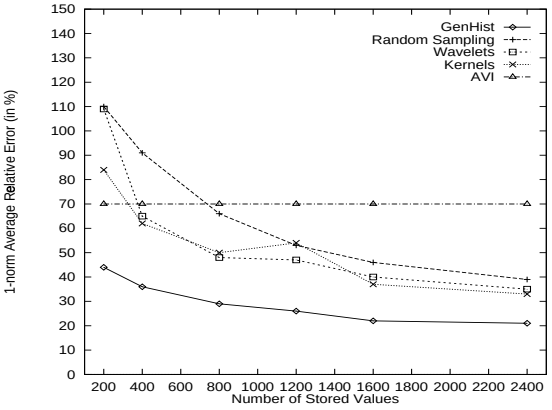

Figure 14: TPC-D dataset, query workload 2, 5-dim, all methods.

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 that 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

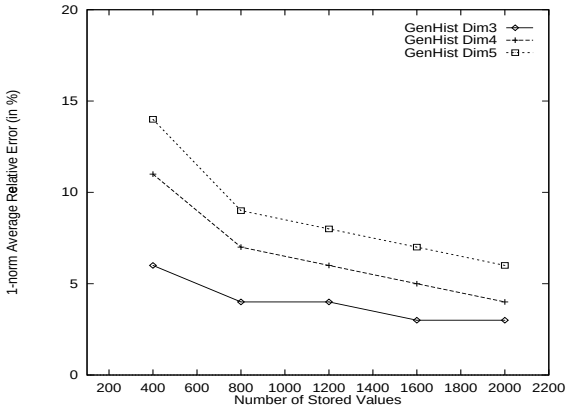

Figure 15: DS1 dataset, query workload 1, 3 to 5 dim, GENHIST.

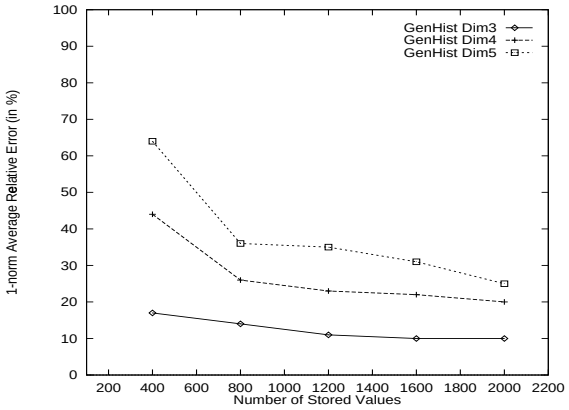

Figure 16: DS1 dataset, query workload 3, 3 to 5 dim, GENHIST.

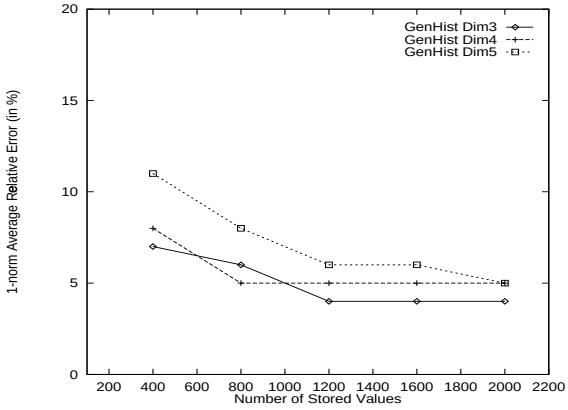

Figure 17: TPC-D dataset, query workload 1, 3 to 5 dim, GENHIST.

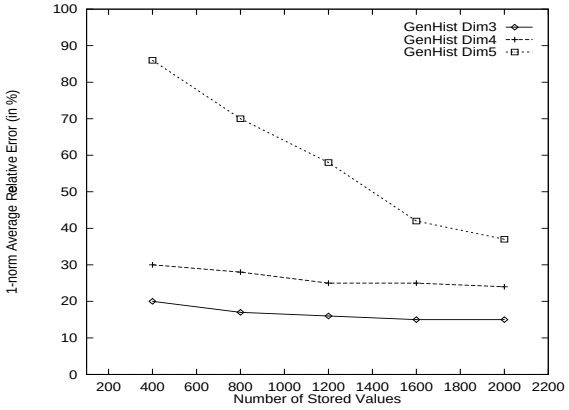

Figure 18: TPC-D dataset, query workload 3, 3 to 5 dim, GENHIST.

the accuracy of GENHIST. In figures 15 and 16 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. In figures 17 and 18 we give the same graphs for the TPC-D dataset. The results, not surprisingly, show the significant degradation in the accuracy that accompanies the increase in space dimensionality.

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. Most of previous work considers only numerical attributes with a small set of different values.

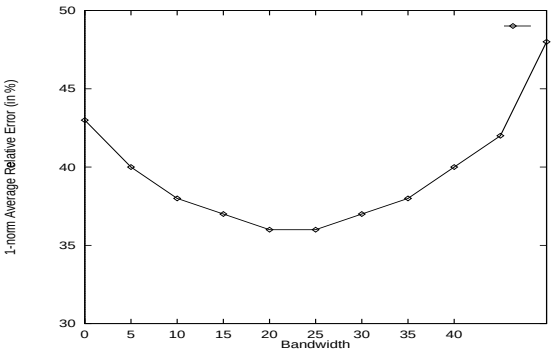

Figure 19: For dataset DS1, 320 samples, and query workload 2 (1% queries), it can be seen that the actual optimal value for the bandwidths B_i is approximately 0.2.

The contributions of the paper are: 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. We perform an experimental study to evaluate and compare this technique and a number of existing techniques: attribute independence assumption, wavelet decomposition, multi-dimensional kernel estimators and sampling.

Conclusions we can draw from our experimental results include: GENHIST typically outperforms other techniques in the range of space dimensionality (3 to 5) that we run experiments. The experiments show that overlapping partitioning is an improvement over non-overlapping partitioning.

Current techniques do not scale well when the dimensionality increases. However some techniques we examine and particularly GENHIST and kernel estimators can be used effectively in 5-dimensional spaces. The kernel estimator approach outperformed pure sampling in all experiments we did.

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. We conjecture that sampling will outperform any of these techniques for dimensionality of around 10, but that the error will be too large to be practical.

8 Acknowledgement

We would like to thank Johannes Gehrke for providing the code of a dataset generator.

References

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

[2] 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.

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

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

[5] 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.

[6] 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.

[7] 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.

[8] Imager Wavelet Library. www.cs.ubc.ca/nest/imager/contributions/bobl/wvlt/top.html

[9] 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.

[10] S. Khanna, S. Muthukrishnan, M. Patterson. On Approximating Rectangle Tiling and Packing. In *Proc. of 9th Annual Symp. on Discrete Algorithms (SODA)*, 1998.

[11] 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.

[12] J. Lee, D. Kim and C. Chung. Multi-dimensional Selectivity Estimation Using Compressed Histogram Information. To appear in *Proc. of the 1999 ACM SIGMOD Intern. Conf. on Management of Data*, June 1999.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] TPC benchmark D (decision support), 1995.

[20] 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.

[21] 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.