Selectivity Estimators for Multi-Dimensional Range Queries over Real Attributes

Dimitrios Gunopulos, UC Riverside, dg@cs.ucr.edu *
George Kollios, Boston University, gkollios@cs.bu.edu
Vassilis J. Tsotras, UC Riverside, tsotras@cs.ucr.edu †
Carlotta Domeniconi, UC Riverside, carlotta@cs.ucr.edu

Abstract

Estimating the selectivity of 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 defines buckets of variable size, and allows the buckets to overlap. The size of the cells is based on the local density of the data. The use of overlapping buckets allows a 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 approaches.

1 Introduction

Computing the selectivity of 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 [5, 9]. Hence, selectivity estimation techniques can be used to optimize top-k queries, as well.

*This research has been supported by NSF IIS-9907477, NSF 9984729 CAREER Award, and the US Dept. of Defense.

†This research has been supported by NSF IIS-9907477 and the US Dept. of Defense.

In data mining, 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 worth exploring, 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 intractable, and only an efficient approximation algorithm can make this data exploration task interactive.

In data warehousing, datasets are typically 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 [39] 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 climatic data like humidity, wind speed, and precipitation).

Real domains have two important characteristics. First, the number of possible queries is infinite in the case of real domains, but finite when considering a finite discrete domain. Of course the number of possible distinct query answers is finite in both cases, since the dataset is finite. 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 for the distribution function of the data. The methods suggested in the literature employ different techniques to find the density estimator for attributes with finite discrete domains. They include computing multi-dimensional histograms [29] [1] [20] [2], using the wavelet transformation [37] [23], SVD [29] or the discrete cosine transform [22], using kernel estimators [3], and sampling [27] [21] [13].

Since the approximate solution to a query must be derived quickly, the description of the approximation function is kept in memory. Typically, an optimizer would maintain a separate approximation function for each of many datasets. Hence function descriptions cannot be very 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 the attributes are independent, and therefore an estimator for multiple dimensions can be obtained by multiplying one-dimensional estimators.

Furthermore, we note that finding density estimators for combinations of attributes can be used to verify whether the independence assumption holds for a set of attributes or not. This is of independent importance for the query optimizer: many optimizers [32] compute query execution costs under the attribute independent assumption. If an optimizer can realize that this assumption does not hold for a set of attributes, more accurate statistics for this set should then be utilized.

1.1 Our Contribution

In this paper we give efficient techniques for finding density estimators for multidimensional datasets with real values. These techniques were originally introduced in [12] along with preliminary experimental results.

First, we describe GENHIST, an approach designed to find multi-dimensional histograms for datasets from real domains. The basic feature of our technique is the overlapping of histogram buckets. Similar to other approaches, GENHIST uses more and smaller buckets to approximate the data distribution where the data density is higher, and fewer and larger buckets where the density decreases. The difference is that these buckets are allowed to overlap, so that the data distribution at a given location is computed by considering all buckets that include this location.

Second, we show how to use multi-dimensional kernel density estimators to solve the multi-dimensional range query selectivity problem. Our solution 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, therefore producing a better approximation of the data density distribution.

Third, we present an extensive comparison between the proposed techniques (GENHIST and multi-dimensional kernel density estimators), and most of the existing approaches for estimating the selectivity of multi-dimensional range queries for real attributes (wavelet transform [37], multi-dimensional histogram MHIST [29], Min-Skew histograms [2], one-dimensional estimation techniques with the attribute independence assumption, and sampling [13]). We include the attribute independence assumption in our study as a baseline comparison. To the best of own knowledge this is the first work comparing a wide variety of selectivity estimators for multidimensional real valued data.

The experimental results show that we can efficiently build selectivity estimators for multi-dimensional datasets with real attributes. Although the accuracy of all techniques drops rapidly as the dimensionality increases, the estimators are quite accurate up to 10 dimensions. GENHIST is the most robust and accurate technique among the approaches that we have tested (MHIST, Min-Skew, kernels, sampling and Independence Assumption), for space dimensionalities between 3 and 10. Among the other techniques, multi-dimensional kernel estimators are quite competitive with GENHIST 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. GENHIST is introduced in section 4, while section 5 describes how to use kernel estimators for multi-dimensional data. Section 6 presents our experimental results and section 7 concludes the paper.

2 Problem Description

Let R be a relation (dataset) 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.

2.1 Range Query Approximation

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, namely $sel(R, Q)$, is the number of tuples in the interior of the hyper-rectangle.

Since n can be very large, the main problem in approximating $sel(R, Q)$ 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 defined in $[0, 1]^d$ and with the property

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

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:

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

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 utilize various error metrics (also used by [37]).

The *absolute error* of a query Q is simply the difference between the real value and the estimated value of its selectivity:

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

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 [37] in defining the relative error as the ratio of the absolute error over the maximum of the selectivity of Q and 1:

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

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

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

2.2 Aggregate Range Query Approximation

Consider the case that each tuple in the database is a point in a d-dimensional space that also has a weight. For example, this weight can be the temperature that a set of fixed sensors report. The set of attributes is then $\mathcal{A} = \{A_1, \ldots, A_d, W\}$. An aggregate range query has the form: "What is the sum (or the average) of the weights of all the points that are inside a range query Q"; here Q is again a range query defined on the d attributes $\{A_1, \ldots, A_d\}$. To allow the user to specify on-line which attribute is the weight, we define an aggregate range query as the sum of one given attribute for all tuples that satisfy the range query:

$$sum(R, Q, i) = \sum_{(x_1, \ldots x_d) \in R \text{ and } (\wedge_{1 \leq j \leq d}(a_j \leq x_j \leq b_j))} x_i$$

or the average of one attribute:

$$ave(R, Q, i) = \frac{sum(R, Q, i)}{sel(R, Q)}.$$

where the query range is specified as: $(a_1 \leq R.A_1 \leq b_1) \wedge \ldots \wedge (a_d \leq R.A_d \leq b_d)$. Using this general definition the user can specify the attribute A_i to aggregate, and chose the ranges of all attributes. If the user does not want to specify a range for attribute A_i, this range can be replaced by the maximum and minimum values of A_i. Following [33], we can approximate such an aggregate query using a density estimator f by computing the integral:

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

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.

Multi-dimensional density estimation techniques are typically generalizations of very successful one-dimensional density 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 [39].

3.1 One-dimensional Histograms

In system R [32], 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 [25]. 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-depth histogram with b buckets is NP-complete even in two dimensions [18], although the problem is easy to be solved in one dimension (in an equi-depth histogram the sum of the tuples in each bucket is approximately the same).

[29] 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 [29].

Both algorithms can be used directly on a dataset with real valued attributes. When MHIST-2 is run on real attribute data the processing time of the method increases. The reason is that with real data there are many possible positions to place a bucket boundary. Every different value which appears in the dataset corresponds to a possible boundary position that MHIST-2 tries to evaluate when it finds the best split. Another problem is that for real valued data, the different values in the dataset have frequency one because each represented value appears once. It is unlikely that good splits can be found for such a frequency distribution. To solve these problems we first use a regular partitioning of the space using a grid.

Min-Skew is a multi-dimensional histogram for spatial datasets (sets of rectangles on the plane) that was proposed in [2]. Min-Skew approximates the original dataset using a regular grid (bining) and computes the number of rectangles that fall inside each cell of the grid. It then partitions the array produced by the grid to create the buckets of the histogram. The buckets cover the whole data space and the goal is to minimize the variance (or spatial skew) inside each bucket. The optimal solution to that problem is provably NP-hard [26], and a technique to decrease the complexity of the problem is to search for a special type of partitioning called Binary Space Partitioning (BSP). Even in that case the complexity of the optimal algorithm increases exponentially with the dimensionality. Therefore, Min-Skew first finds for each bucket a good dimension to split using the one-dimensional marginal distributions. For each bucket the best split is found and finally the split that gives the highest reduction in variance is chosen. The variance is normalized with the number of objects inside each bucket. Although the technique was proposed for two-dimensional sets of rectangles, it can be easily extended to consider two or higher dimensional points, essentially by considering each point as a special case rectangle.

3.3 Discrete Decomposition Techniques

The d-dimensional data distribution of a dataset R with attributes $A_1, \dots A_d$ can be represented by a d dimensional array D with $\prod_{1 \le i \le d} |V_i|$ 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) [29], the wavelet transform [37], and recently the Discrete Cosine Transform (DCT) [22].

The basic operation of all decomposition techniques is to essentially perform a change of bases. Each value in the original array D is then 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 (i.e., $O(b)$ space). To estimate a value of D, the inverse transformation on D'' is computed. 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 ([29]). Wavelets and, recently, DCT have been shown to give good results in high dimensionalities [37] [23] [22]. In our comparisons we include the wavelet transform.

If the attributes have real values, the size of the array D can be n^d, where n is the size of R. Since each of these approaches involves operations on the array 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 $\{1, \dots, \xi\}$. We call the new dataset R'. We define the *frequency* of a tuple $\mathbf{t}= (t_1, \dots, t_d) \in \{1, \dots, \xi\}^d$ 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 array with D_ξ. We can then use the wavelet transform to obtain an approximation for R'.

Note however that queries do not have to fall on bucket boundaries. In one dimension, a query may contain a number of buckets 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^d 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

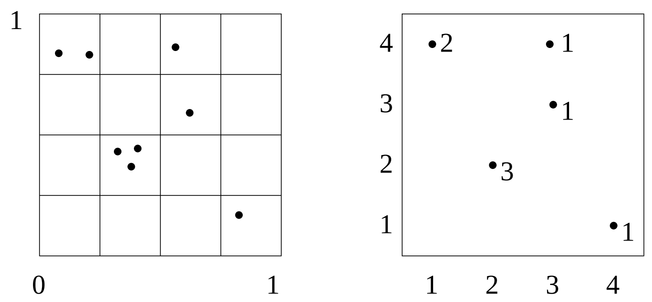

Figure 1: A 4-regular partitioning of a 2-dimensional dataset and the tuple frequencies.

completely inside the query, we just add up the approximate value. Otherwise we use the uniform distribution assumption to estimate the fraction of tuples that lie in the interior of the bucket and 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 array D_ξ^s of D_ξ. In the partial sum the value of an array slot is replaced by the sum of all preceding slots:

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

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, because this results to a smaller number of significant wavelet coefficients. This result agrees with [37], who also suggest that the partial sum method is more accurate because this operation smoothes up the data distribution.

3.4 Self-Tuning histograms

Another very interesting alternative to multi-dimensional histograms are the self-tuning histograms (STH) presented in [1]. The idea is to start with a very simple initial histogram, then use the 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, experimental results presented in [1] [4] show that STH histograms are less accurate than GENHIST or MHIST-2 for high dimensions (experiments in dimensionalities 2 to 4 were reported) and skewed data.

To improve the performance of STHs, the STHoles histogram technique was recently proposed [4]. STHoles are also build by analyzing query results, rather than by examining the dataset. They allow bucket nesting (which is a restricted variation of overlapping), and have been shown to perform comparably to GENHIST and superior to STHs and MHIST in experiments presented in [4].

Compared to traditional multidimensional histograms and discrete decomposition techniques, STHs and STHoles offer the significant advantage that they gracefully adapt to updates. On the other hand, the estimator is built over a sequence of queries to the dataset, which means that in the beginning no estimator is available, and the time it takes to create an accurate estimator depends on the distribution of the queries.

3.5 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 $\frac{|R|}{b} sel(S, Q)$ is used to estimate $sel(R, Q)$. Sampling is simple and easy to perform so the estimator can be computed fast. Computing the selectivity is also simple, although not necessarily more efficient than other methods. As a result it is widely used for estimating the selectivity [32] [6] [10] or for on-line aggregation [14]. 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 [38] [7] have rarely been applied in database problems. One of the reasons is that in statistics a dataset is considered as 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 [33]. This technique can be quite accurate if the clusters themselves can be accurately modeled by multi-dimensional Gaussian functions. However, even assuming that this is the case, 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 buckets is that ideally we need relatively small buckets to capture the variations in the data distribution accurately. However, assuming a constant number of buckets, when the dimensionality of the space increases the buckets of the partition must have increasingly larger one-dimensional projections. 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 range query in d dimensions can intersect $O(\xi^{d-1})$ buckets. In the 4 regular partitioning of a 2-dimensional space, a range query can intersect up to 12 of the 16 buckets; in the 4 regular partitioning of a 5-dimensional space, a range query can intersect up to 992 of the 1024 buckets, that is, all the buckets except the 2^5 buckets that do not touch any boundary.

An example of such a query is a range query where all the one-dimensional intervals cover the entire range of the attributes, except for very small intervals next to the minimum and the maximum values.

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. On the one hand, the accuracy within each bucket becomes better. On the other hand, each time a bucket is intersected partially, we have to make the assumption that the data distribution is uniform inside the bucket. When this assumption is not accurate, an additional error is introduced. This error can be additive and therefore can increase if the number of intersected buckets is increased.

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.

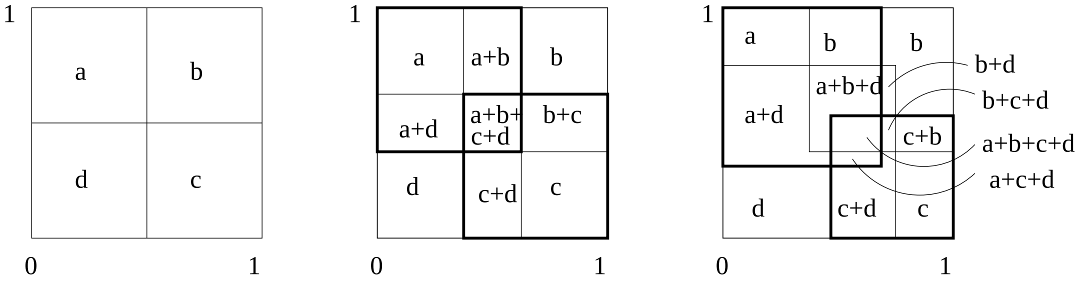

Figure 2: The same 2-dimensional space is partitioned first into 4 regions, using 4 non-overlapping buckets (average densities are a, b, c and d), then into 9 regions, using 4 overlapping equal-sized buckets (densities are a, b, c, d, a+b, b+c, c+d, d+a, a+b+c+d), and finally into 13 regions, using 4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

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.

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 (Figure 3).

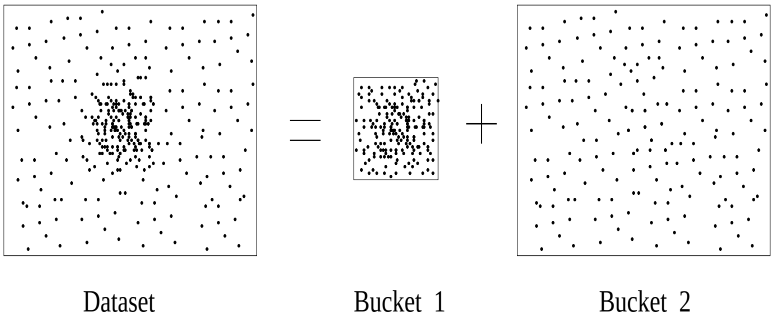

Figure 3: An example where the density of a two-dimensional dataset can be efficiently approximated by two overlapping buckets.

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. 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 (Figure 4).

Thus GENHIST can be classified as a generalization of the biased histograms [30]. 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).

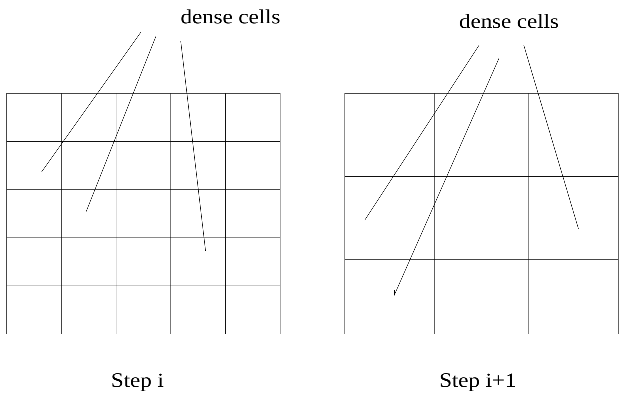

Figure 4: At step i, a set of grid cells (in the 5-regular grid in this example) are selected to be added in the GENHIST estimator. At step $i+1$, a new set of cells, from a coarser grid (3-regular in this example) are selected to be added in the GENHIST estimator. Buckets from different grids can overlap.

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 5 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 $b_\xi = \lceil b/\log_{\frac{1}{\alpha}} \xi \rceil$ 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 $1/\log_{\frac{1}{\alpha}} \xi$.

4.2 Storing the GENHIST estimator

GENHIST uses only cells from the grids as buckets in the estimator. This offers an important advantage: each bucket can be stored using only two numbers. For a given initial value of ξ, and a given α, the sequence of grids that the algorithm is using is deterministic. Therefore all grid cells can be totally ordered in a single ordering. Since each bucket c is a cell in one of the grids, we can use one number to identify the location of each bucket in the total ordering of cells. This allows us to

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 within each bucket divided by n).
3. Find the $b_\xi = \lceil b/\log_{\frac{1}{\alpha}} \xi \rceil$ 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 neighboring buckets:
      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 5: The GENHIST algorithm

find the bounds of the bucket. We use another number to keep the density of c, $d_c - av_c$.

4.3 Running Time

The algorithm makes $\log_{\frac{1}{\alpha}} \xi$ iterations. The number of iterations is therefore constant if ξ and α are constants. Each iteration performs one pass over the data, and takes linear time. Therefore the running time of the algorithm is linear to the size of the data. One pass over the data is performed each time the size of the grid changes. Since ξ is set to a small constant, the actual number of passes over the data is 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 this bucket is in the hash table, we increment its counter, otherwise we insert the bucket 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.

4.4 Eliminating Writes

Implementing step 4.b.i 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 on the disk.

Assume that we are scanning the dataset R at the i-th iteration, and in the previous $i-1$ iterations we have already computed a set of buckets that we are keeping in the estimator. Each of these buckets was a grid cell in some previous iteration. During the i-th scan of dataset R we want to determine the number of points that fall in the interior of each cell in the grid. However, each of the buckets already in the estimator overlaps with some of the cells in the grid we are using in the i-th scan. To count accurately how many points fall in each cell of the grid of the i-th iteration, we have to account for the points that have been removed from the interior of each bucket already in the estimator. For each such bucket B_j in the estimator we keep the total number of 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.i (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. The following lemma derives from the above discussion.

Lemma 1 *The expected density of a bucket that is computed using this process in the i-th iteration is equal to the expected density of the bucket if we had physically removed from the dataset the required number of points in the previous iterations.*

Proof: We 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 $\frac{r(B_i)}{tot(B_i)}+(1-\frac{r(B_i)}{tot(B_i)})\frac{r(B_j)}{tot(B_j)}$
□

4.5 Estimating the selectivity of range queries using GENHIST

The estimation of the selectivity of a range 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.

4.6 Estimating the selectivity of aggregate range queries using GENHIST

To estimate the selectivity of an aggregate range query Q and an attribute A_i, we have to compute the integral

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

For GENHIST the approximation f is the sum of b constant functions, corresponding to the b buckets $B_1, \ldots B_b$ of the estimator. Each function is non-zero only within the area of the cell. The integral is equal to

$$sum(f, Q, i) = \sum_{B \in G} \int_{B \cap Q} B_{count} x_i dx_1 \ldots dx_d$$

which can be computed analytically in $O(db)$ time.

5 Multi-dimensional kernel density estimators

The problem of estimating an underlying data distribution has been studied extensively in statistics [31] [38]. Kernel estimators are statistical techniques which approximate a probability distribution, and are thus applicable to address the query selectivity problem.

Kernel estimation is a generalized form of sampling. The basic step is to produce a uniform random sample from the dataset. As in random sampling, 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 over the area nearby it, 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 over this area. Therefore, a given location in space gets contribution from each kernel centered at a point that is close enough so that the corresponding 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 $\int_{[0,1]} k_i(x)dx = 1$. 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. [31] 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

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

One dimensional kernel estimators were examined in [3]. 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

$$\sigma(f, Q) = \int_{[0,1]^d \cap Q} f(x_1, \ldots, x_d) = \frac{1}{n} \sum_{t_i \in S} \int_{[0,1]^d \cap Q} k(x_1 - t_{i_1}, \ldots, x_d - t_{i_d}) dx_1 \ldots dx_d.$$

It has been shown that the shape of the kernel function does not affect the approximation substantially [7]. It is the standard deviation, or bandwidth, of the function that is important. Therefore, we choose a kernel function that it is easy to integrate. The Epanechnikov kernel function has this property [7]. 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 (Figure 6).

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.

We need to face two problems 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 it covers lies outside the data (and the query) space. The result is that these points distribute less than a unit weight over 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 [38]. No efficient solution exists for finding the optimal bandwidths. To get an initial estimate for the bandwidth we use Scott's rule [31] 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 under the assumption that the data distribution is a multi-dimensional normal and so in many cases smoothes the function too much. To solve the second

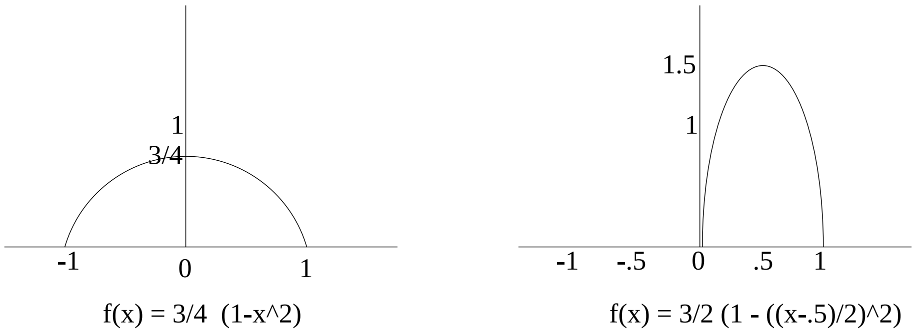

Figure 6: The one-dimensional Epanechnikov kernel, with $B = 1$, centered around the origin, and, with $B = 2$, centered at 0.5.

problem, we project the parts of the kernel function that lie outside $[0,1]^d$ back into the data space [38]. 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:

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

$$\frac{1}{n} \int_{[a_1,b_1]\times\ldots\times[a_d,b_d]} \sum_{1 \le i \le b} (\frac{3}{4})^d \frac{1}{B_1 B_2 \ldots B_d} \prod_{1 \le j \le d} (1 - (\frac{x_j - X_{ij}}{B_j})^2)) \, dx_1 \ldots dx_d =$$

$$\frac{1}{n} (\frac{3}{4})^d \frac{1}{B_1 B_2 \ldots B_d} \sum_{1 \le i \le b} \int_{[a_1,b_1]} (1 - (\frac{x_1 - X_{i1}}{B_1})^2) \ldots \int_{[a_d,b_d]} (1 - (\frac{x_d - X_{id}}{B_d})^2) \, dx_d \ldots dx_1$$

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

5.2 Running Time

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

1. A random sample of size n is taken (where n is an input parameter).
2. An approximation of the standard deviation for each attribute is computed.

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 used in our experimental study, it is always better to use a multi-dimensional kernel estimator rather that random sampling for selectivity estimation.

6 Experimental Results

In our experiments we want to compare the behavior of the different selectivity estimation techniques 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 highly correlated. For example, the precipitation and humidity readings in climatic data are definitely correlated attributes. 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, 5, 8 and 10 dimensions. Interestingly, after 5 dimensions, accuracy dropped significantly for all methods in the correlated datasets we experimented with.

Third, we want to examine how the accuracy of the various methods varies as a function of the available space for the estimator. However, since we expect that a system will have to store many estimators in memory, we concentrate on the accuracy of estimators of small size.

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, MHIST-2, and Min-Skew.

Random sampling is a simple and widely used technique. In particular, we want to compare sampling against kernels, to measure the improvement we gain by 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. [37] 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 and the Min-Skew histogram since it can be used for non-spatial datasets as well.

In the sequel we describe in more detail our implementation of the different techniques.

6.1.1 GENHIST Histograms

We implemented the GENHIST algorithm as described in Section 4, using a main memory hash table, to maintain statistics for every bucket. In our implementation we only considered buckets that contain more than 0.1% of the remaining points. We varied the initial value of ξ between 8 and 20. We stored two numbers for each bucket in the estimator: We used one number to identify the location of each bucket in the total ordering of cells, and another number to keep the number of tuples in the bucket.

6.1.2 Wavelet Decomposition

To implement the wavelets method we followed the approach presented in [37]. We used the Haar wavelets as our wavelet basis functions. In the first step we performed a ξ regular partitioning of the data space with ξ equal to 32, and then we computed the partial sum matrix D^s_ξ. We used $\xi = 32$ because wavelets operate on arrays that are a power of 2 long, and the next larger choice, 64, creates a very large partial sum array even in 5 dimensions. We used the standard wavelet decomposition of a d-multidimensional array. That is, we performed a one-dimensional wavelet transform on the first dimension and we replaced the original values with the resulting coefficients. Then we did the same for the second dimension, treating the modified matrix as the original matrix, and we continued up to d dimensions. We performed thresholding after normalization: we weighed the wavelet coefficients, and we kept 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 ran experiments both using partial sums and using the original matrix. Our datasets are not very sparse, and the partial sum method performed better. Therefore, we report the partial sum matrix results only. Nevertheless, the partial sum matrix method cannot be used in dimensionalities higher than five since the space overhead is high. For example, for a 8-dimensional dataset and using $\xi = 8$, we get an array of $8^8 = 2^{24}$ cells. For dimensionalities higher than 5 we should use the original matrix. However, the performance of this approach degenerates in 8 and 10 dimensions. As is pointed out in [24], the accuracy of the two alternatives varies with dataset and query type. It seems that for the real-valued datasets that we used the original matrix approach is not appropriate. We obtained the code for wavelets from [15].

6.1.3 MHIST-2

We ran MHIST-2, using the binary code provided by the authors [29]. The binary code worked for up to 4 dimensions, so we could not run experiments for higher dimensional data. As above, we first performed a regular ξ partitioning with ξ between 5 and 30 and then we ran MHIST-2 using the resulting dataset (which can have up to ξ^d size) as input. 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, and for the best ξ in each case.

6.1.4 Min-Skew Histograms

The implementation of Min-Skew is based on the description in [2]. The method as originally proposed addresses the selectivity estimation problem when the input is two-dimensional (spatial) rectangles. To extend the algorithm for larger dimensionalities is straightforward because it starts with a uniform grid of regions. In [2] the authors note that Min-Skew is most accurate when the optimal grid size instead of progressive refinement is used. Since the number of regions in the grid is a very important parameter for Min-Skew, in our experiments we considered grids with different numbers of regions. In particular, we used regular grids with ξ between 3 and 30. We ran the experiments for all different grid sizes and we used the best grid size to compare with the other methods. In most cases, the best value of ξ is between 3 and 10. In our implementation the size of the main memory requirement for Min-Skew increases as ξ^d. This is not an issue for the two-dimensional (spatial) datasets for which the method was originally designed for. Nevertheless, it is a consideration for higher dimensionalities.

In is worth noting here that Min-Skew is significantly different from GENHIST. GENHIST creates overlapping buckets for the density estimator; Min-Skew creates non-overlaping buckets (that form a partition of the space). All the buckets in the GENHIST estimator are cells in some grid; Min-Skew may combine regions into larger buckets. GENHIST approximates the very dense regions first, using smaller buckets; Min-Skew starts from a very simple partitioning and then splits buckets into smaller buckets. GENHIST uses multiple overlapping grids; Min-Skew uses one grid (if progressive refinement is used, Min-Skew may entirely replace a region in the grid with smaller regions that partition the same area).

6.1.5 Multi-dimensional Kernels

For the kernels method we used the Epanechnikov product kernel. We select a bandwidth using 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.

6.1.6 Attribute Value Independence (AVI)

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 used a specific method to compute the selectivity for each attribute then we expect the error rates to be 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).

Essentially, the AVI line in the following experiments represents the upper bound on selectivity estimation accuracy that one can achieve if there is no use of any information on correlations between attributes, but unlimited space is allowed to estimate one-dimensional selectivities.

6.2 Datasets

6.2.1 Synthetic datasets

We generated 3, 4, 5, 8 and 10-dimensional datasets with many clusters, and therefore significant correlations between the attributes. In addition, the distribution was non-uniform in each attribute. Each attribute took real values in the interval $[0, 1]$.

We used 3 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 generated clusters are in the $(d-1)$ or $(d-2)$-dimensional subspaces. This means that the d-way correlation is small in datasets of Type 2. 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 [37] [23], and is a popular benchmark. We used a projection of 3 to 5 numerical attributes. In particular we used the tables LINEITEM and PARTSUPP. First we performed a join between the two tables using the foreign keys and then we used the attributes QUANTITY, EXTENDEDPRICE, DISCOUNT, TAX and AVAILQTY to generate our datasets. We did not change the resulting dataset in any way.

All datasets include 10^6 points.

6.2.2 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 590,000 points, and each point has 54 attributes, 10 of which are numerical. We normalize the numerical attributes to a $[0, 1]$ range, so the values become real values. We use subsets of 3, 4, 5, and 8 numerical attributes for our experiments. We also use the entire set of 10 numerical attributes. We normalize the coordinates for each dimension to $[0, 100]$ and therefore the values are stored as float numbers. In this dataset the distribution of the attributes is non-uniform, and there are correlations between pairs of attributes.

6.3 Query workloads

To evaluate the techniques we created workloads of 4 types of queries. For each dataset we create a workload 1 containing random queries with selectivity approximately 10%, and a workload 2 with random queries having selectivity approximately 1%. These workloads comprise 10^4 queries each. We chose the two numbers to estimate the performance of the techniques for relatively large queries (10% selectivity) and for relatively small queries (1% selectivity). We expected that all techniques

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

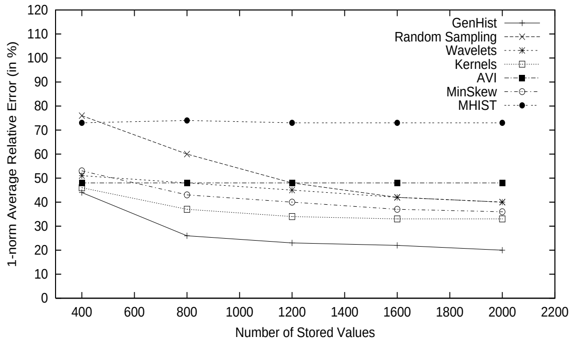

Figure 7: DS1 dataset, query workload 3, 4-dim.

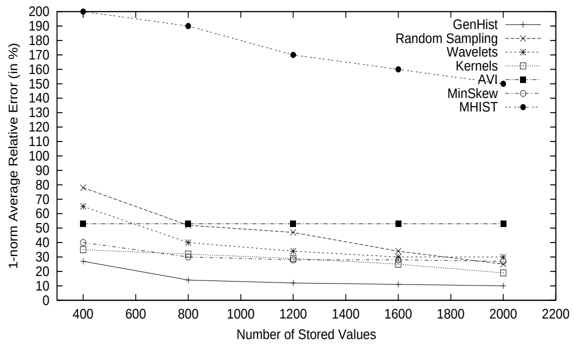

Figure 8: Forest Cover dataset, query workload 3, 4-dim.

would perform significantly better for large queries. Note that it is difficult to generate large numbers of queries of very small selectivity in high dimensional spaces. Therefore, we did not experiment with query workloads where the queries had smaller selectivity than 1%. Instead we created a third workload (workload 3) of 20,000 queries of the form $(R.A_1 < a_1) \wedge \ldots \wedge (R.A_d < a_d)$ for a randomly chosen point $(a_1, \ldots, a_d) \in [0,1]^d$. Such queries have very diverse selectivities, including very small ones. For the datasets in 10 dimensions we also create a workload 4 containing random queries with selectivity approximately 1%, and with the interval for two randomly chosen attributes set to the whole range $[0,1]$.

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

6.4 Experimental comparison of the accuracy of different methods

We performed an extensive study to evaluate the performance of the methods for 3-, 4-, 5-, 8- and 10-dimensional data with the following exceptions. For MHIST-2 the available binary could run for up to 4 dimensions. We could run wavelets for up to 5 dimensions because the size of the partial sum array (32^d) became prohibitive in higher dimensionalities. Our extension of Min-Skew could run up to 8 dimensions (over that the main memory requirements became prohibitive). For 3-dimensional datasets there were small differences in accuracy between the techniques. Interesting changes started appearing at 4 dimensions and are described below.

6.4.1 4-dimensional data

We first present the performance of all methods for 4-dimensional data in Figures 7 and 8. We plot the 1-norm average relative error for each method for different values of the available storage space to store the estimator. For brevity we show only the DS1 (type 1), and the Forest Cover datasets, using query workload 3. We got similar results for the other datasets and query workloads. All techniques perform worse in 4 dimensions than in 3 dimensions. The largest drop in overall accuracy is for MHIST-2.

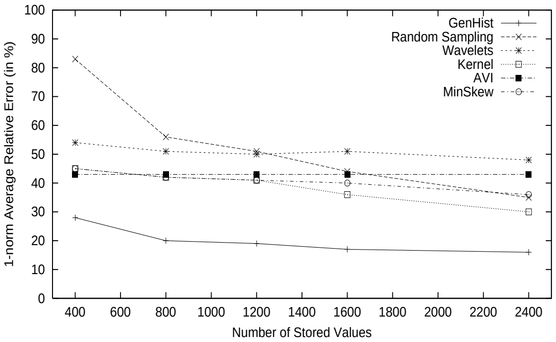

Figure 9: DS1 dataset, query workload 2, 5-dim.

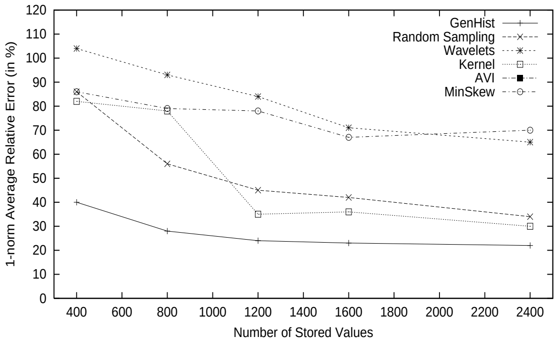

Figure 10: DS2 dataset, query workload 2, 5-dim.

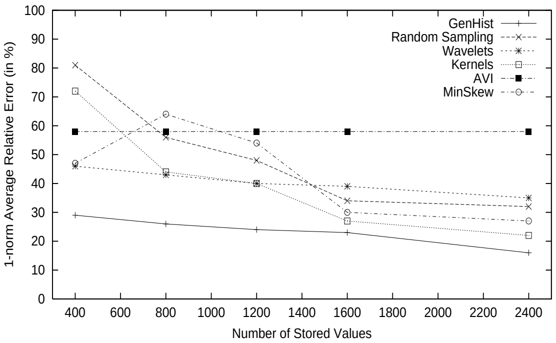

Figure 11: Forest Cover dataset, query workload 2, 5-dim.

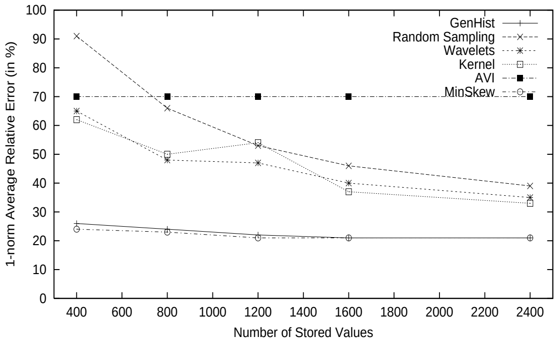

Figure 12: TPC-D dataset, query workload 2, 5-dim.

6.4.2 5-dimensional data

For the experiments in 5 dimensions, we show four datasets ($DS1$, $DS2$, TPC-D, and Forest Cover dataset) using query workloads 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 $DS1$.

Figures 9-12 depict the 1-norm average relative error of each method for query workload 2 (1% queries) while Figures 13-16 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 1, multi-dimensional kernels are more accurate. For the TPC-D dataset on query workload 2 Min-Skew is the most accurate technique. However, even in these cases GENHIST is very close in accuracy to the best technique, and is in fact the only technique among those that we tried that is the most accurate or close to the most accurate in all the experiments.

Multi-dimensional kernels also perform well in general. An interesting observation is that kernels perform better than random sampling for all datasets.

Wavelets perform better than other methods when the space used to store the approximation is

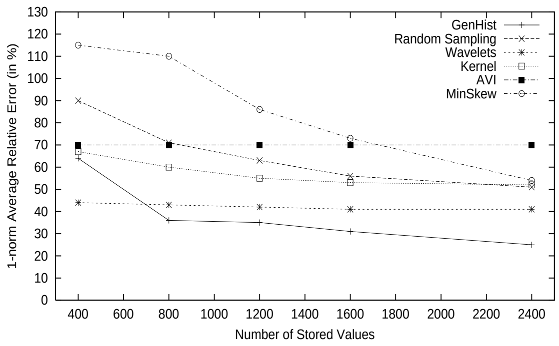

Figure 13: DS1 dataset, query workload 3, 5-dim.

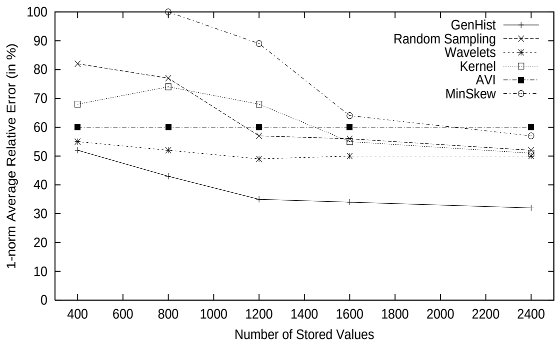

Figure 14: DS2 dataset, query workload 3, 5-dim.

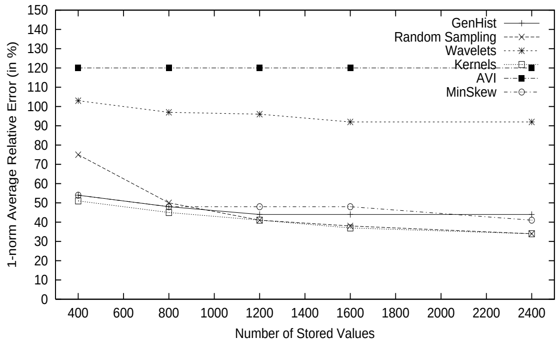

Figure 15: Forest Cover dataset, query workload 3, 5-dim.

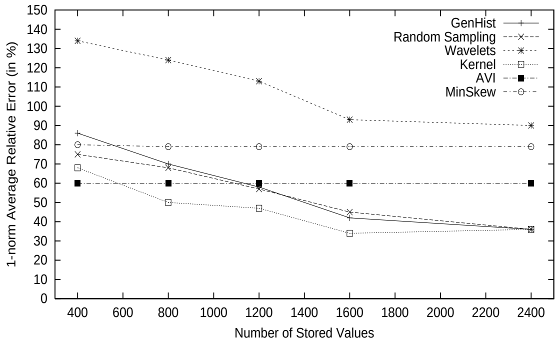

Figure 16: TPC-D dataset, query workload 3, 5-dim.

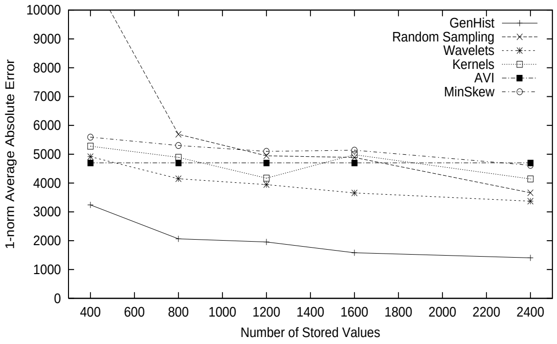

Figure 17: DS1 dataset, query workload 3, 5-dim, Average Absolute error.

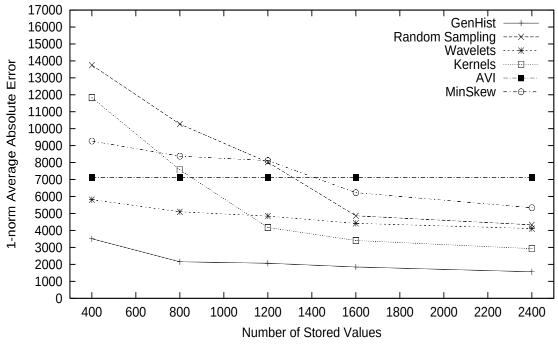

Figure 18: DS2 dataset, query workload 3, 5-dim, Average Absolute error.

very small. But increasing the available space (that is, using more wavelet coefficients to approximate the density function) offers small improvement. The reason is that beyond a certain point, which is reached quickly in our experiments, many coefficients have approximately the same magnitude. Thus, using more coefficients gives little help.

Min-Skew histograms perform well in two cases. For the TPCD dataset and workload 2 and the Forest Cover dataset and workload 3. In the other experiments Min-Skew is inferior to GENHIST and in most cases it is also inferior to multi-dimensional kernels.

In Figures 17 and 18 we plot the Average Absolute error for DS1 and DS2 datasets and query workload 3. Since the relative performance of the various methods is similar, in the rest of this paper we report only the Average Relative Error.

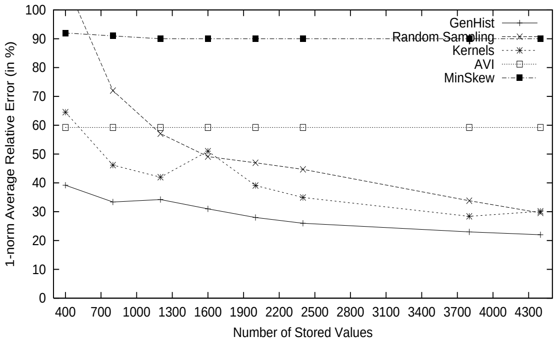

Figure 19: Forest Cover dataset, query workload 2, 8-dim.

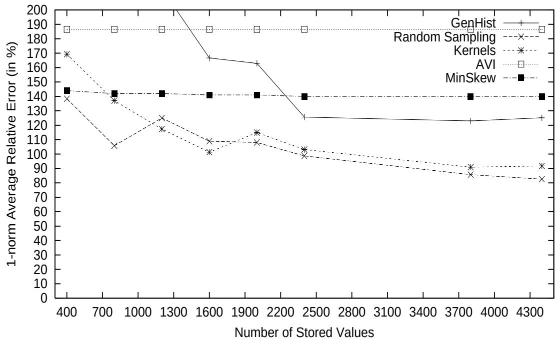

Figure 20: Forest Cover dataset, query workload 3, 8-dim.

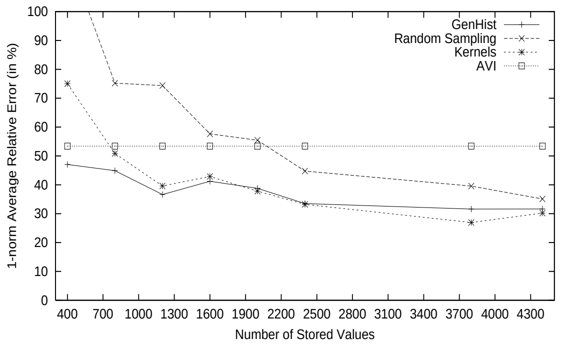

Figure 21: Forest Cover dataset, query workload 2, 10-dim.

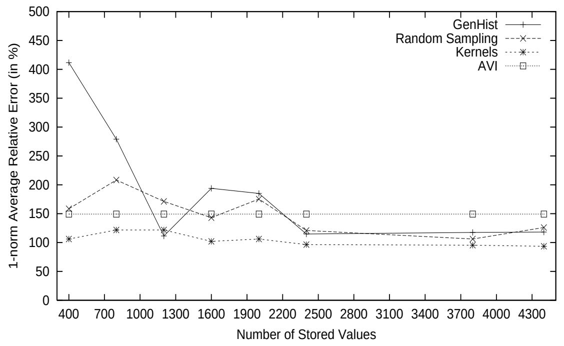

Figure 22: Forest Cover dataset, query workload 3, 10-dim.

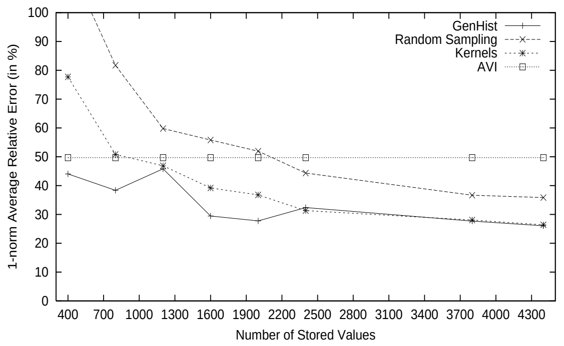

Figure 23: Forest Cover dataset, query workload 4, 10-dim.

6.4.3 8 and 10-dimensional datasets

For the experiments in 8 and 10 dimensions, we use the DS1 and Forest Cover datasets with query workloads 2 and 3. The results are very similar and we report here only the results for the Forest Cover dataset in Figures 19 - 23. Also for the 10 dimensional datasets we used the workload 4 which consists of lower dimensional range queries of cardinality close to 1%.

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. As the results show, the accuracy of the estimation of the selectivity drops further. For example the error for workload 3 is much larger in dimensionalities 8 and 10 (Figures 20 and 23) because these queries are formed by taking a random point in the space, and so their volume and therefore their selectivity is very small in expectation.

Nevertheless, the GENHIST and the multidimensional kernel estimators are in general more accurate than random sampling or AVI. Note however that Min-Skew is performing worse than AVI for query workload 2 in 8 dimensions. We do not report results for Min-Skew in 10 dimensions because

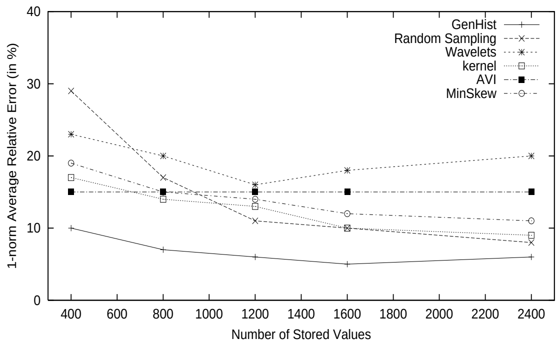

Figure 24: DS2 dataset, query workload 1, 5-dim.

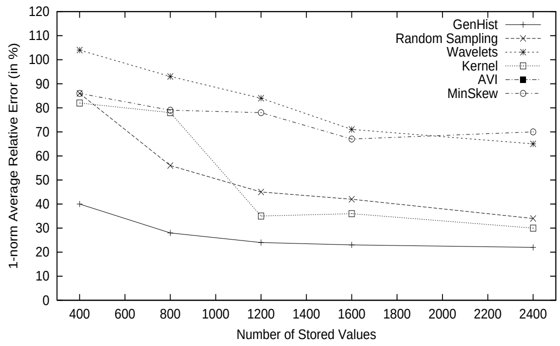

Figure 25: DS2 dataset, query workload 2, 5-dim.

in our implementation the array that approximates the original dataset becomes very large even for small values of ξ and it cannot be kept in main memory.

6.4.4 Impact of query size on accuracy

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 24 and 25). 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.

Clearly at 5 dimensions the accuracy of all methods decreases, 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.

6.4.5 Stability of the techniques

In Table 1 we report the standard deviation values of the 1-norm Average Relative Error for Random Sampling and Kernels, computed over the five runs we perform for these methods. We use the DS1 dataset (5, 8 and 10 dimensions), query workload 2, and sizes 800 and 2400 of the estimator. We observe that the standard deviation values for Kernels are smaller in all cases, and their values reduce as the size of the estimator increases.

6.5 Comparison of running times

Random sampling and multidimensional kernel estimators require only one pass through the data whereas 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 larger because of the computation of the transformation for each row of the matrix. Note that in the worst case the construction time matches the time for external memory sorting [37].

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.

We compare here the running times of the algorithms tested for the 5 dimensional datasets. We used the DS1 dataset, 20,000 queries of type 3 (anchored queries), and we set the size of the estimator to 2000 stored values (8kbytes).

The results for construction and estimation times are shown in Table 2. As it is expected, Random Sampling and Kernels give the shortest construction times. Random sampling has also the best estimation time. However, the estimation times for the other methods are very close (of the same magnitude).

Method	800, 5-d	2400, 5-d	800, 8-d	2400, 8-d	800, 10-d	2400, 10-d
Random Sampling	0.068	0.036	0.08	0.04	0.14	0.07
Kernels	0.024	0.006	0.04	0.02	0.07	0.02

Table 1: Standard deviation values of the 1-norm Average Relative Error for Random Sampling and Kernels, DS1 dataset, query workload 2, Number of Stored Values 800 and 2400, 5-8-10-dim.

Method	Construction Time	Estimation Time
Random Sampling	30	7
Kernels	31	35
GenHist	550	30
Wavelets	650	41
Min-Skew	500	28

Table 2: Running times in seconds.

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.

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, Min-Skew 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 10) that we run experiments on. GENHIST can be thought of as a multi-dimensional histogram that allows for overlapping partitioning.

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

(3) For the real-valued and correlated datasets we have used, the accuracy of all techniques decreases when dimensionality increases. However the GENHIST and the multi-dimensional kernel estimators are more robust and accurate even in 10-dimensional spaces.

An interesting future 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 [11]. Such techniques can be extended to apply to kernel estimators, too. Most work for maintaining histograms has concentrated on the one dimensional case [11], 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 Acknowledgment

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] N. Bruno, S. Chaudhuri, L. Gravano. STHoles: A Multidimensional Workload-Aware Histogram In Proc. of the 2001 ACM SIGMOD Int. Conf. on Management of Data, 2001.

[5] 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.

[6] 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.

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

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

[9] 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.

[10] 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.

[11] 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.

[12] D. Gunopulos, G. Kollios, V. Tsotras, C. Domeniconi. Approximating multi-dimensional aggregate range queries over real attributes *Proc. of the 2000 ACM SIMGOD Int. Symp. on Management of Data*, 2000.

[13] 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.

[14] 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.

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

[16] 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.

[17] 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.

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

[19] 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.

[20] F. Korn, T. Johnson and H. Jagadish. Range Selectivity Estimation for Continuous Attributes. In *Proc. of 11th Int. Conf. on SSDBMs*, 1999.

[21] 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.

[22] 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.

[23] 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.

[24] Y. Matias, J. Scott Vitter, M. Wang. Dynamic Maintenance of Wavelet-Based Histograms. In *Proc. of the 26th VLDB Conf., Cairo, Egypt*, September 2000.

[25] 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.

[26] S. Muthukrishnan, V. Poosala, T. Suel. On Rectangular Partitionings in Two Dimensions: Algorithms, Complexity, and Applications. In *Proc. of the ICDT 1999*, pp. 236-256, Jerusalem, 1999.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] D. Scott. *Multivariate Density Estimation: Theory, Practice and Visualization.* Wiley & Sons, 1992.

[32] 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.

[33] 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.

[34] B.W. Silverman. Density Estimation for Statistics and Data Analysis. *Monographs on Statistics and Applied Probability*, Chapman & Hall 1986.

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

[36] 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.

[37] 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.

[38] M.P. Wand and M.C. Jones. Kernel Smoothing. *Monographs on Statistics and Applied Probability*, Chapman & Hall 1995.

[39] 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.