Equi-Depth Multi-Dimensional Histograms

by
M. Muralikrishna
David J. DeWitt

Computer Sciences Technical Report #733
December 1987

Equi-Depth Multi-Dimensional Histograms

M. Muralikrishna

David J. DeWitt

Computer Sciences Department
University of Wisconsin
Madison, Wisconsin 53706

This research was partially supported by the Defense Advanced Research Projects Agency under contract N00039-86-C-0578 and by the National Science Foundation under grants DCR-8512862, MCS82-01870, and MCS81-05904.

1. Abstract

Multi-dimensional queries commonly occur in databases dealing with geographical, image, and VLSI databases. A typical two dimensional query in a geographical database might involve finding all cities within certain latitudinal and longitudinal bounds. Several multi-dimensional index structures have been proposed in the literature. KDB trees [Robinson81], R-trees [Guttman84], and Grid files [Nievergelt84] are among the more popular ones. However, there has been no work in designing multi-dimensional histograms to aid in the optimization process using these multi-dimensional index structures. In order for an optimizer to select an appropriate access path for a multi-dimensional query, fairly accurate selectivity estimates must be available to it. Selectivity estimates are also useful in determining appropriate join methods that follow the selections.

In this paper we present an algorithm for generating equi-depth, multi-dimensional histograms. One might expect that the cost of building a d-dimensional histogram would be at least d times the cost of sorting the relation on a single attribute. We show, in our algorithm, that the sorting cost of building a d-dimensional histogram is significantly less than the cost of sorting the relation d times. We present a main memory data structure for storing the histograms and discuss two schemes for estimating the number of tuples that will be retrieved by a given query. Experimental results are presented that show the efficacy of our histograms. The usefulness of a sampling technique in generating histograms at a very low cost is also explored.

2. Related Work

The System R optimizer [Selinger79] used simple statistics, such as the minimum and maximum values in a given column, to estimate selectivity factors. Using such simple statistics will produce good selectivity estimates only if the attribute values are uniformly distributed. Since attribute values can have other distributions, it has become commonplace for relational query optimizers to use histograms for estimating selectivity factors. However, these histograms traditionally have the same width. Equi-width histograms also produce erroneous selectivity estimates if the attribute values are not uniformly distributed [Shapiro84]. The problem of building equi-depth histograms on a single attribute has been well studied in [Shapiro84]. It has been shown in [Shapiro84] that the way to control the maximum estimation error is to control the depth of each histogram and not its width. In other words, all histogram buckets must have the same depth and not the same width. It is necessary to sort the relation on the particular attribute in order to generate equi-depth histograms. The maximum selectivity estimation error can be arbi-

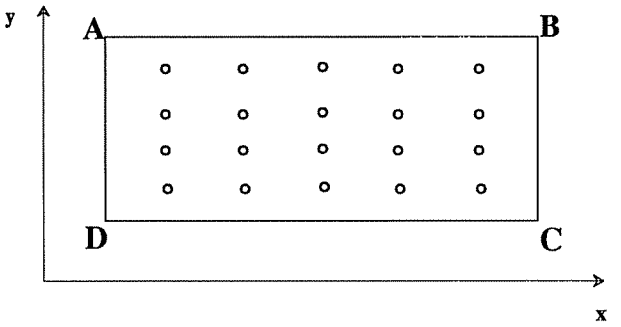
trarily reduced by increasing the number of equi-depth buckets.

In this paper, we will show that even in the case of queries involving multiple attributes, equi-depth histograms are superior to equi-width histograms.

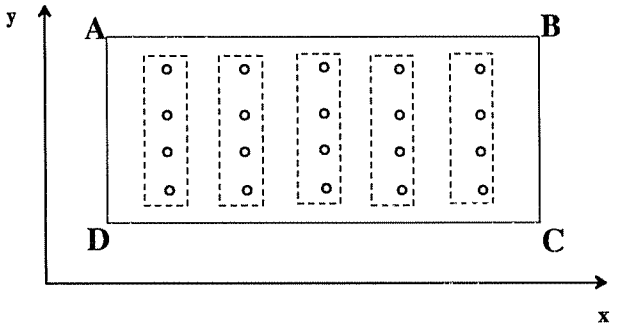
3. Generating Multi-Dimensional Histograms

Before we discuss our algorithm for generating multi-dimensional histograms, we must first describe what equi-depth multi-dimensional histograms will look like. Let us discuss this in the context of a 2-dimensional example. Assume a relation R with attributes x and y. Figure 1(A) shows a rectangle ABCD that represents the space of tuples of relation R. The points inside the rectangle represent the tuples. The problem of generating equi-depth histograms is equivalent to covering all the tuples in the tuple space with S rectangles such that each rectangle has the same number, viz., $\frac{|R|}{S}$ of tuples within it. Such rectangles are called equi-depth histograms or equi-depth buckets. We will hereafter use the terms bucket and histogram interchangeably. We will later show how the maximum estimation error is decreased by increasing S. Clearly, the problem of covering the tuple space with equi-depth buckets does not have a unique solution. For example, Figures 1(B) and 1(C) show two different solutions with 5 buckets, each bucket having 4 tuples. It clearly seems infeasible to design an algorithm that can come up with ad-hoc solutions, such as seen in Figures 1(B) and 1(C). Instead, we developed the following algorithm for determining the boundaries of the equi-depth buckets. We will describe the algorithm for the 2-dimensional case. The extension to higher dimensions is straightforward.

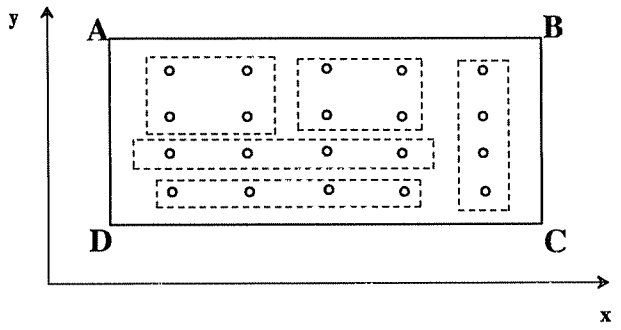
Let the number of buckets desired be $S = bucket_1 * bucket_2$. $Bucket_i$ will be used to denote the number of divisions along the ith attribute (dimension). Thus, the number of tuples in each bucket = $\frac{N}{S}$, where N is the total number of tuples. To simplify the following explanation, we will assume that $\frac{N}{S}$ is an integral number. We assume a sorting routine called SORT which takes three parameters. The first parameter is the attribute_number (1 or 2 in this case) on which the relation is to be sorted in ascending order. The second and the third parameters are respectively called low and high. Low and high are the serial numbers of two tuples in the relation, such that the tuples ranging from low through high are sorted on the attribute given by the first parameter. For example, the invocation SORT(2, 501, 1000) would sort tuples 501 through 1000 on the second attribute in ascending order. We describe the algorithm in words and then in pseudo-code.



(A)



(B)



(C)

Figure 1.

First, the entire relation (tuples 1 through N) is first sorted on the first attribute. We then form $bucket_1$ partitions of equal size. The first partition consists of tuples 1 through $\frac{N}{bucket_1}$; the second partition consists of tuples $(\frac{N}{bucket_1} + 1)$ through $2 * \frac{N}{bucket_1}$, etc. We call these partitions **primary** partitions. We then sort each of these primary partitions on the second attribute and then divide each primary partition into $bucket_2$ **secondary** partitions.

The important point is that the secondary partitions that are formed from a single primary partition are completely enclosed within that parent primary partition. We thus form a total of ($bucket_1$ * $bucket_2$) number of secondary partitions, each containing $\frac{N}{bucket_1 * bucket_2}$ number of tuples. Each of these secondary partitions corresponds to a bucket and vice versa. Each bucket may be represented by the coordinates of its left-bottom and right-top corners. The left-bottom x(y)-coordinate of a bucket is simply the lowest value of the first (second) attribute of the tuples in the corresponding secondary partition. Similarly, the right-top x(y)-coordinate of a bucket is the highest value of the first (second) attribute of the tuples in the corresponding secondary partition.

We now present the pseudo-code version of the algorithm.

Algorithm /* To generate equi-depth 2-dimensional histograms */

SORT (1, 1, N) /* sort the whole relation on the first attribute */

FOR i = 1 TO $bucket_1$ DO
BEGIN
low = (i - 1) * $\frac{N}{bucket_1}$ + 1;

high = i * $\frac{N}{bucket_1}$;

SORT (2, low, high); /* Sort on the second attribute */

END_FOR

capacity = $\frac{N}{bucket_1 * bucket_2}$;

bucket_no = 0;

FOR j = 1 TO $bucket_1$ DO
BEGIN
FOR k = 1 TO $bucket_2$ DO
BEGIN
bucket_no = bucket_no + 1

/* find serial numbers of the first and last tuples in the partition*/

first_tuple_id = (bucket_no - 1) * capacity + 1;

last_tuple_id = bucket_no * capacity;

FIND_COORDINATES (bucket_no, first_tuple_id, last_tuple_id);

END_FOR
END_FOR

End Algorithm

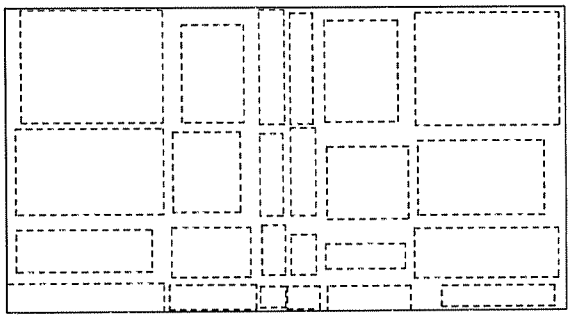
Figure 2 shows an example of 2-dimensional histograms. The values of the first attribute are normally distributed and those along the second attribute have a zipfian [Zipf49] distribution. Equi-depth histograms 'capture' the notion of distribution of the tuples very elegantly. Note that the entire tuple space is not covered by the histograms. This is because there were not any tuples in those spaces not covered by any histogram.

The above algorithm can be easily extended to higher dimensions. For example, extending to three dimensions, we would need to sort each of the ($bucket_1$ * $bucket_2$) secondary partitions on the third attribute and divide each secondary partition into $bucket_3$ **tertiary** partitions. Again, each of the tertiary partitions are completely enclosed within the parent secondary partition. We would then have a total of ($bucket_1$ * $bucket_2$ * $bucket_3$) number of partitions in a strict hierarchy, each having the same number of tuples. Each of these tertiary partitions corresponds to a 3-dimensional bucket, whose coordinates can be found in the manner described above.

A natural question that arises is: What is the cost of building these histograms? Let the number of dimensions be 3, and the number of data pages in the relation be Z. We will assume that all the tuples in the relation are of the same size. The cost of sorting the whole relation to obtain the $bucket_1$ number of equi-sized primary partitions is

$$C_1 = C * Z * \log(Z)$$

where C is some constant. Each of these primary partitions is sorted to give $bucket_2$ number of secondary partitions. The sorting cost at this stage is therefore given by



An example of two-dimensional, equi-depth histograms.

Figure 2.

$$C_2 = \text{bucket}_1 * C * (\frac{Z}{\text{bucket}_1}) * \log(\frac{Z}{\text{bucket}_1})$$

$$= C * Z * \log(\frac{Z}{\text{bucket}_1})$$

Similarly, at the third stage, the sorting cost is given by

$$C_3 = \text{bucket}_1 * \text{bucket}_2 * C * (\frac{Z}{\text{bucket}_1 * \text{bucket}_2}) * \log(\frac{Z}{\text{bucket}_1 * \text{bucket}_2})$$

$$= C * Z * \log(\frac{Z}{\text{bucket}_1 * \text{bucket}_2})$$

Thus, the total sorting cost = $C_1 + C_2 + C_3$

$$= C * Z \left[\log(Z) + \log(\frac{Z}{\text{bucket}_1}) + \log(\frac{Z}{\text{bucket}_1 * \text{bucket}_2}) \right]$$

Notice that the sorting cost decreases at each stage. Generalizing to d-dimensions, and assuming the same number of divisions at each stage (= b), the total sorting cost is

$$C * Z * \log \left[\frac{Z^d}{b^{\frac{d(d-1)}{2}}} \right]$$

At some stage, it is quite possible that the partitions become sufficiently small enough that they can each fit in main memory. If this happens, the sorting cost will be further reduced.

4. A Storage Structure for Multi-Dimensional Histograms: The H-tree

A multi-dimensional query corresponds to finding all tuples that have attribute values within the bounds of the multi-dimensional box specified by the query (the query box).

Definition: An **f-bucket** is a bucket that is completely enclosed within the query box. □

Definition: An **p-bucket** is a bucket that partially overlaps the query box. □

Let S be the total number of equi-depth buckets with $\frac{N}{S}$ tuples per bucket. For a given query box, let

f = total number of f-buckets.

p = total number of p-buckets.

Clearly, the following holds:

$$f * \frac{N}{S} \leq \text{actual number of tuples in the query box} \leq (f+p) * \frac{N}{S}.$$

Whatever the method we use to estimate the number of tuples in the query box, we will be interested in determining the exact values of f and p for the given query box. One possible scheme is to check every bucket and see if it is an

f-bucket or a p-bucket or neither. This process will obviously become increasingly inefficient for larger values of S, even when the histograms are stored in main memory. In addition, S can grow exponentially with the number of dimensions. We would like a main memory data structure that will enable us to search significantly less than S buckets for f-buckets and p-buckets. At the same time, the memory requirements for the data structure should grow only linearly with S. Fortunately, the R-tree index structure proposed in [Guttman84] is very close to what is needed. The R-tree mechanism is used to retrieve data items efficiently according to their spatial locations. For our case, the equi-depth buckets correspond to the data items in the leaves of the R-tree. In order to enhance the performance of the search process, we will use a very close variant of the R-tree that exploits the strict hierarchy of partitions obtained during the process of generating the equi-depth buckets. The variant will be called the H-tree (**H**istogram tree), so as to distinguish it from the R-tree. Unlike the dynamic R-tree, the H-tree will be a static structure that is built once when the histograms are first computed. If the histograms become dated, the H-tree will need to be built again[1]. The H-tree will always be height balanced with the height equal to the number of dimensions. Each level corresponds to the respective dimension. For example, the root node corresponds to the first dimension, the second level to the second dimension and so on.

There are two kinds of nodes in the H-tree:

1. The internal nodes (including the root node), and
2. The leaf nodes.

For ease of notation, we will assume that the data type of the attribute along the kth dimension is DATA_TYPE_k. Let d be the total number of dimensions. An internal node of the H-tree at the kth ($1 \leq k \leq d-1$) level is an array of records and can be characterized by the following definitions:

```
TYPE Internal_node_element_k =

RECORD
{
        low_point, high_point : DATA_TYPE_k;
        next : POINTER;
}

TYPE Internal_node_k =

ARRAY [1 .. buckets_k] OF Internal_node_element_k;
```

[1]We feel that it would be very inefficient to dynamically update the histograms after each addition or deletion.

A leaf node is an array of records and can be characterized by the following definitions:

TYPE **Leaf_node_element** =

```
RECORD
{
        low_coordinate_1, high_coordinate_1 : DATA_TYPE_1;
        low_coordinate_2, high_coordinate_2 : DATA_TYPE_2;
        ...             ...
        low_coordinate_d, high_coordinate_d : DATA_TYPE_d;
}
```

TYPE **Leaf_node** =

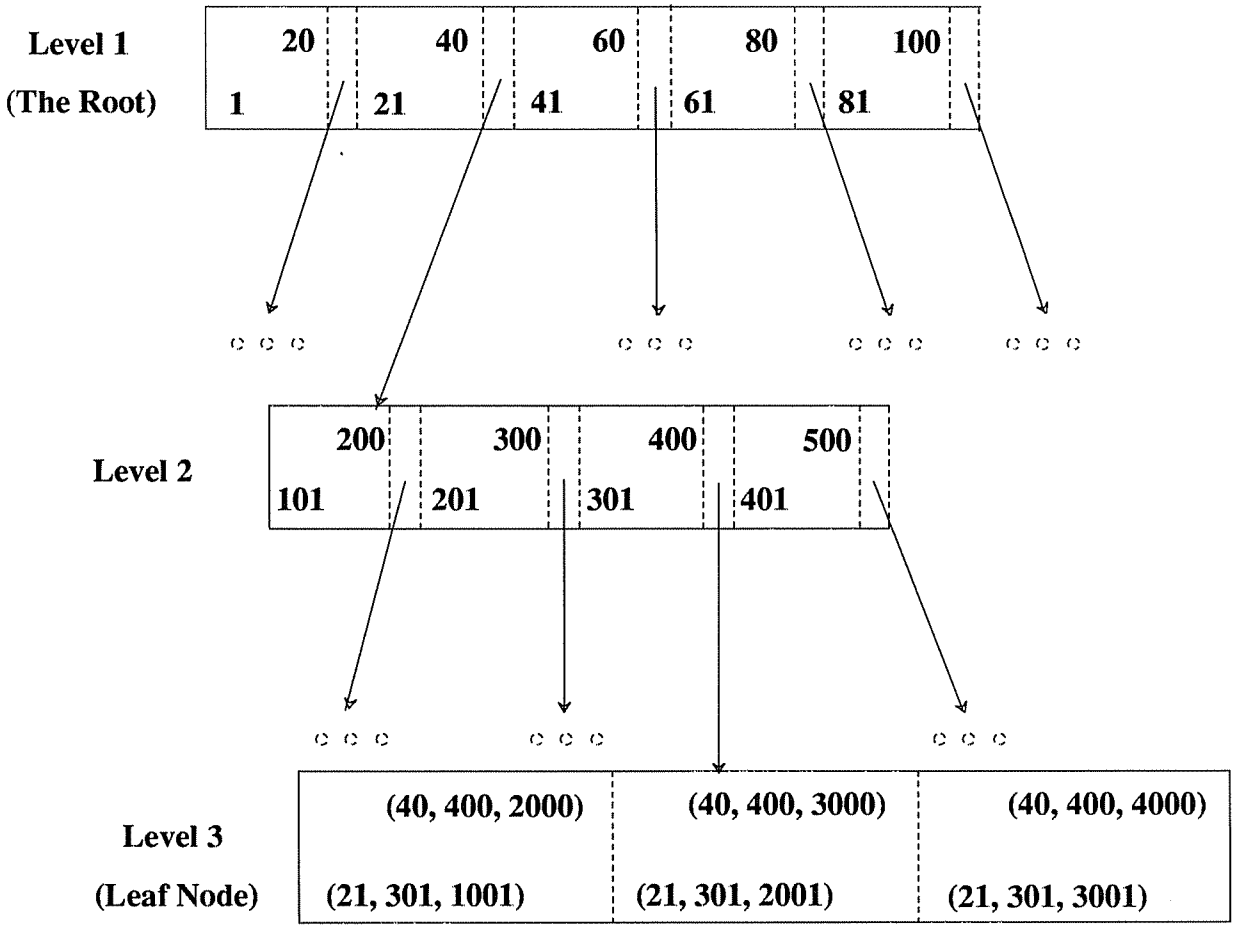
ARRAY [1 .. $buckets_d$] OF Leaf_node_element;

4.1. The Search Algorithm

The search algorithm is recursive and similar to that of the search mechanism in the R-tree. We will illustrate it with a three dimensional example. Let the attribute values of the first attribute range from 1 to 100; the attribute values of the second attribute range from 101 to 500; and the attribute values of the third attribute range from 1001 to 4000. Let $bucket_1 = 5$, $bucket_2 = 4$, and $bucket_3 = 3$. Assuming that the attribute values along each dimension are perfectly uniformly distributed, then the resulting H-tree will be as shown in Figure 3. The numbers shown in Figure 3 represent the values of the fields in the respective records (as defined above).

Let the query box of interest be given by ((31, 325, 1250), (50, 375, 2500)). Walking through the elements in the root, we find that the range (31, 50) overlaps with the second and third element in the root. Following the second pointer, to the second level, we find that (325, 375) overlaps only with the third entry. Following the third pointer into the third level, and searching through the elements at the leaf level, we find that the first and second entries overlap with (1250, 2500). Both these buckets, ((21, 301, 1001), (40, 400, 2000)) and ((21, 301, 2001), (40, 400, 3000)) are p-buckets with respect to the query box. Backing up to the second level and then to the root, we follow the third pointer in the root node down the H-tree in a similar fashion.

It must be observed that, like R-Tree traversals but unlike B-tree traversals, more than one subtree under a node may have to be searched. Hence, in the worst case, the whole H-tree may be traversed. However, in practice, query boxes will generally be small in comparison to the size of the entire tuple space. Only those buckets in the vicinity of the query box will be searched. If the number of entries at each node is large (> 20), binary search can



A Three dimensional H-Tree.

Figure 3.

be used at each node instead of a linear search.

The storage requirements of the H-tree are dominated by the Leaf nodes. The number of Leaf Node elements is exactly the same as the total number of buckets. In addition, for a fixed number of dimensions and a particular set of attributes, the size of a Leaf Node element is fixed. The size of the H-tree thus grows linearly with the number of buckets. Assuming d = 3 and four-byte integer attributes, each leaf_node_element will have six integer fields and thus will be 24 bytes in size. If S = 10 x 10 x 10, the H-tree will occupy slightly over 1000 * 24 = 24,000 bytes of memory.

5. Estimation Schemes

Consider a relation with N tuples. For a given query box, let 'act_tuples' denote the actual number of tuples within the box. Let 'est_tuples' denote the estimated number of tuples within the box by some estimation scheme. Consider the two evaluation metrics, D and R, defined as follows: $D = \frac{|act\_tuples - est\_tuples|}{N}$ and $R = \frac{est\_tuples}{act\_tuples}$, (act_tuples $\neq 0$). If the estimation scheme is good, we would expect D to be close to 0, and R to be close to 1. In judging how good an estimation scheme is, we will consider only D for the following reason. Consider the following two scenarios:

1. act_tuples = 10; est_tuples = 100;
2. act_tuples = 1000; est_tuples = 10000;

In either case R = 10. However, $D = \frac{90}{N}$ in the first case and $D = \frac{9000}{N}$ in the second case. D reflects the magnitude of the error in the estimated selectivity of the query. Since cost formulas in query optimizers depend heavily on the estimated selectivity factors, D is a much better metric than R. Assuming that $\frac{90}{N}$ is fairly small, it is unlikely that the access path chosen in the first scenario, based on the estimated selectivity, would be different from the optimal access path. On the other hand, it is quite possible that the access path chosen in the second scenario might be different from the optimal access path, since $\frac{9000}{N}$ is 100 times larger than $\frac{90}{N}$. Therefore, we will use only parameter D for judging the quality of an estimation scheme.

We now describe two schemes for estimating the value of est_tuples for a given query box. The first scheme, viz., the **Half Scheme**, is conservative in that it only attempts to reduce the worst case error. The second scheme, viz., the **Uniform Scheme**, as we will demonstrate below, performs much better on the average. Theoretically, the worst case error possible using the Uniform Scheme is twice that of in the Half Scheme. However, in practice, we have found that the maximum error of the Uniform Scheme is significantly less than the maximum error attained by the Half Scheme. After describing the two schemes, we will present experimental results that confirm these statements.

5.1. The Half Scheme

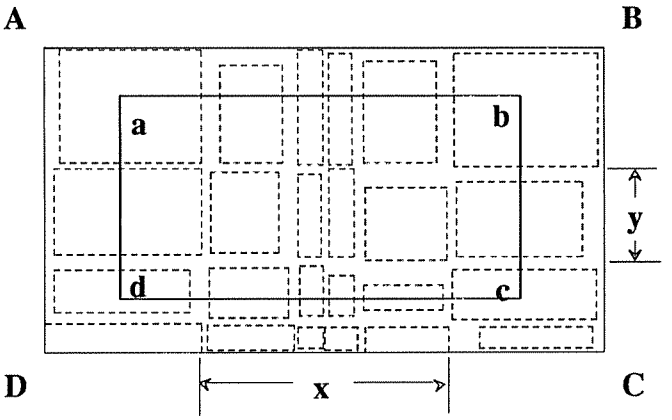
Given a query box, we know that the following holds:

$$f * \frac{N}{S} \leq \text{actual number of tuples in the query box} \leq (f+p) * \frac{N}{S},$$

where f(p) is the number of f(p)-buckets for the given query box. In other words,

$$\frac{f}{S} \leq \text{actual selectivity} \leq \frac{(f+p)}{S}.$$

If we choose the estimated selectivity to be $\frac{(f + \frac{p}{2})}{S}$, which is the mid-point of the two extremes, our estimation error can never be larger than $\frac{p}{2S}$. In other words, for every partially overlapping bucket, we will assume that half of the tuples within it are also within the given query box. How large can p get? Figure 4 shows an example of 2-dimensional (d = 2) histograms. Clearly[2], f = x * y and p = (x + 2) * (y + 2) - x * y = 2 * (x + y) + 4. Note that the



Rectangle ABCD is the entire tuple space.

Rectangle abcd is a query box.

x = 4, y = 1.

bucket(1) = 6, bucket(2) = 4.

Figure 4.

estimation error for small query boxes will be smaller. p will assume its largest value when x is $bucket_1$ - 2, and y is $bucket_2$ - 2. The largest value of p is therefore 2 * ($bucket_1$ + $bucket_2$) - 4. The largest estimation error is thus = $\frac{p}{2S} = \frac{bucket_1 + bucket_2 - 2}{bucket_1 * bucket_2} \approx \frac{1}{bucket_1} + \frac{1}{bucket_2}$. If $bucket_1 = bucket_2 = 5$, the maximum estimation error = 32%. Similarly we can show that the largest estimation error in d dimensions

$$\approx \sum_{i=1}^{d} \frac{1}{bucket_i}$$

Thus, for a fixed number of buckets S, we can easily show (by differentiating the expression above and equating it to zero and solving for $bucket_i$, i = 1, d) that the maximum estimation error is minimized when

$$bucket_i = S^{\frac{1}{d}}, \; i = 1, d.$$

5.2. The Uniform Scheme

In this scheme, the estimated number of tuples for a given query box is calculated by the following formula:

$$est\_tuples = \frac{N}{S}(f + \sum_{i=1}^{p} fract(i))$$

where

$$fract(i) = \frac{Size\ of\ ((ith\ p\text{–}bucket) \cap (the\ query\ box))}{(Size\ of\ the\ ith\ p\text{–}bucket)}$$

where if d = 2, the size refers to the area; if d = 3, the size refers to the volume. Clearly,

$$f * \frac{N}{S} < est\_tuples < (f + p) * \frac{N}{S}, \text{ since } 0 < fract(i) < 1 \text{ for } i = 1, p.$$

Essentially, we are assuming that tuples are uniformly distributed in each of the p-buckets. As might be expected, the validity of this assumption will be enhanced as the size of each bucket becomes smaller, regardless of the actual distribution of the tuples. As we will demonstrate below, our experimental results bear this out. It is possible that the error in the estimate can be as large as $\frac{p}{S}$. However, this seems to rarely happen in practice.

6. The Experiments

Each relation had 104,000 tuples. All attributes were integers with attribute values varying from a minimum of 1 to a maximum[3] of 241. The values of each attribute were generated independently of each other and had one of

[2]A similar analysis can be carried out for any value of d. In particular, for d = 3, we have p = (x + 2) * (y + 2) * (z + 2) - x * y * z.

[3]These bounds were chosen arbitrarily. In fact, we replaced the bound 241 by 10001 and repeated some of the experiments. There was no significant change in the results.

the following three distributions: normal(n) (with a mean of 121 and a standard deviation of 50), uniform(u), or zipfian(z). We generated a total of 9 (= 3 * 3) tuple spaces in the 2-dimensional experiments and a total of 27 (= 3 * 3 * 3) in the 3-dimensional experiments. For each tuple distribution, we performed two types of experiments.

The objective of the first series of experiments was to observe the maximum estimation error obtained by the two schemes. A total of 5000 large, square query boxes were generated such that they almost occupied the entire tuple space. Using these large boxes, we measured the maximum estimation error for both the estimation schemes. The results of the first series of experiments (on two dimensions) are presented in Table 1. The corresponding 3-dimensional results are presented in Section 8. Most of the tables presented here have a format similar to that of Table 1. The first column indicates the distributions of the attributes along each of the dimensions. For example, an entry "n z" indicates a normal distribution for the first attribute and a zipfian distribution for the second attribute.

distr	stats	5x5		10x10		20x20	
		Half	Uniform	Half	Uniform	Half	Uniform
n n	max	**28.3+**	**16.1+**	**14.3+**	**6.6+**	6.5-	**2.2-**
	avg	**18.3**	**8.1**	**7.3**	**2.8**	**2.3**	**0.6**
	std.dev	7.5	3.9	4.2	1.5	1.6	0.5
n u	max	27.6+	6.6+	13.6+	3.0+	5.8+	1.2-
	avg	12.1	3.4	4.9	1.2	1.7	0.4
	std.dev	7.4	1.8	3.4	0.8	1.2	0.3
n z	max	23.6+	11.1+	11.0+	3.8+	5.1+	1.6+
	avg	8.8	1.6	4.3	0.6	1.7	0.2
	std.dev	5.9	1.5	2.8	0.6	1.1	0.2
u n	max	27.7+	6.6+	13.8+	3.0+	6.4+	1.2-
	avg	12.2	3.5	5.1	1.2	1.9	0.3
	std.dev	7.4	1.7	3.4	0.8	1.3	0.2
u u	max	28.2-	0.7-	13.2+	0.6+	**7.1-**	0.5-
	avg	9.0	0.2	3.8	0.1	1.5	0.1
	std.dev	6.9	0.1	2.9	0.1	1.2	0.1
u z	max	22.7+	6.1-	10.6+	1.5+	5.3+	1.3+
	avg	6.4	1.1	3.1	0.3	1.3	0.2
	std.dev	5.0	0.9	2.3	0.2	0.9	0.2
z n	max	23.6+	11.0+	12.3+	4.6+	5.7-	1.4-
	avg	9.0	1.7	4.6	0.8	1.8	0.3
	std.dev	5.9	1.6	2.9	0.7	1.2	0.2
z u	max	22.6+	6.1-	11.8+	2.0+	4.8+	0.9+
	avg	6.5	1.1	3.3	0.4	1.4	0.2
	std.dev	5.1	1.0	2.4	0.4	1.0	0.2
z z	max	21.1+	9.1-	9.7+	2.2+	4.1+	1.0+
	avg	4.0	1.0	2.4	0.3	1.1	0.1
	std.dev	3.7	1.0	2.0	0.3	0.8	0.2

Table 1: Estimation Errors For Large Boxes By The Two Schemes.

For a given tuple distribution, we varied the number of equi-depth buckets from 25 (5x5) to 100 (10x10) and finally to 400 (20x20). In each case, we calculated the actual number of tuples and the estimated number of tuples within a query box by each of the two schemes. This was repeated for each of the 5000 large query boxes. For each scheme, the magnitude of the maximum percentage deviation in selectivity over these 5000 boxes was calculated. A positive (negative) sign besides each number in the "max" row indicates that the actual number of tuples was greater than or equal to (less than) the estimated number of tuples. We also present the average ("avg" row) and the standard deviation ("std.dev" row) of the percentage magnitudes of the deviations. The maximum deviation and the maximum average deviation for each column are indicated in boldface.

There are some obvious conclusions we can draw from Table 1. We know from Section 5.1 that the maximum percentage error in estimating the selectivity by the Half scheme is 32% for the 5x5 case. The corresponding numbers for the 10x10 and the 20x20 case are 18% and 9.5% respectively. The maximum estimation error in each column (in Table 1) under the Half Scheme are close to the theoretically possible limits. On the other hand, the corresponding maximum estimation errors obtained by the Uniform Scheme are about one-half to one-third of those obtained by the Half Scheme.

The objective of the second series of experiments was to study the average behavior of the Uniform Scheme. A total of 5000 square query boxes were generated such that a large percentage of boxes had small areas (or volumes). This reflects real life situations wherein a large percentage of queries retrieve only a small amount of data. The coordinates of the query boxes were chosen uniformly randomly. Table 2 gives the distribution of the areas of the boxes. The number of tuples in each query box was estimated by the Uniform Scheme only. The results of these experiments are displayed in Table 3. We generated histograms of equal depth as well as histograms of equal width. When buckets are of equal width, each bucket has a different number of tuples. When using equi-width histograms and the Uniform Scheme of estimation, the number of tuples within a query box is calculated by the following formula:

$$\text{est\_tuples} = \sum_{i=1}^{f+p} (\text{fract}(i) * \text{occupancy}(i))$$

where occupancy(i) is the number of tuples in the ith bucket. The one advantage of building equi-width buckets is that the relation never has to be sorted. However, as Table 3 shows, the maximum deviations obtained by the the equal-width scheme are very high. In the 20x20 case, the maximum average deviation is only 0.25% for equi-depth histograms as opposed to 7.38% for equi-width histograms.

range of areas	Number of boxes
0 - 2499	1069
2500 - 4999	446
5000 - 7499	327
7500 - 9999	265
10000 - 12499	232
12500 - 14999	214
15000 - 17499	189
17500 - 19999	182
20000 - 22499	147
22500 - 24999	186
25000 - 27499	161
27500 - 29999	175
30000 - 32499	148
32500 - 34999	168
35000 - 37499	137
37500 - 39999	132
40000 - 42499	158
42500 - 44999	138
45000 - 47499	98
47500 - 49999	126
50000 - 52499	138
52500 - 54999	100
55000 - 57499	97
57500 - 59999	27

Table 2: Distribution Of The Areas Of Query Boxes.

A natural question that arises at this point is the following: how valid are the results presented in Table 3, especially those under the Equi-depth columns? To find out, we used the **method of batch means** [Sargent76]. We generated 20 batches consisting of 1000 query boxes each such that the sizes of the boxes in each batch formed a zipfian distribution. The boxes were located randomly. In each batch we estimated the number of tuples in each query box using the Uniform Scheme. For each of these batches, we calculated the average percentage deviation (as before) for each tuple distribution and each equi-depth histogram configuration. In every case, we found that with 90% confidence, the variance of the average was less than 8% of the average. In fact, out of a total of 27 cases, the variance of the average was less than 6% of the average in 23 of the cases. In all cases, the variance itself did not exceed 0.152%.

7. Building Histograms by Random Sampling

In situations where sorting a relation may be considered expensive or where only a quick estimate of the selectivity is required, we can resort to building equi-depth histograms using a small sample of tuples taken from the

distr	stats	5x5		10x10		20x20	
		Eq.Dep.	Eq.Wid.	Eq.Dep.	Eq.Wid.	Eq.Dep.	Eq.Wid.
n n	max	**15.9+**	5.5-	**6.1+**	9.9-	**1.4-**	18.4-
	avg	**3.2**	1.2	**1.0**	3.4	**0.3**	**7.4**
	std.dev	3.3	1.1	1.1	3.0	0.3	6.3
n u	max	6.6+	4.6-	2.8+	9.0-	1.2-	17.6-
	avg	1.5	1.1	0.5	3.1	0.2	6.5
	std.dev	1.6	1.1	0.6	2.7	0.2	5.6
n z	max	11.2+	35.4-	3.2+	31.3-	1.3+	30.6-
	avg	0.8	6.6	0.2	3.9	0.1	3.6
	std.dev	0.8	10.8	0.3	7.5	0.1	5.8
u n	max	6.5+	4.5-	2.7+	8.9-	1.1-	17.5-
	avg	1.5	1.0	0.5	3.0	0.1	6.4
	std.dev	1.5	1.1	0.6	2.6	0.1	5.6
u u	max	0.6-	3.3-	0.5+	7.5-	0.3-	16.4-
	avg	0.1	1.2	0.1	2.7	0.1	5.7
	std.dev	0.1	1.1	0.1	2.4	0.1	5.0
u z	max	6.5-	33.9-	1.5-	29.7-	1.2-	29.8-
	avg	0.8	6.2	0.2	3.5	0.1	3.3
	std.dev	0.8	10.2	0.2	7.0	0.1	5.5
z n	max	8.5+	35.4-	3.7+	31.2-	1.4-	30.7-
	avg	0.8	6.7	0.2	**3.9**	0.1	3.8
	std.dev	0.8	10.8	0.3	7.5	0.1	5.9
z u	max	6.5-	34.0-	2.0+	29.8-	0.9+	30.0-
	avg	0.9	6.3	0.2	3.6	0.1	3.4
	std.dev	0.8	10.2	0.3	7.1	0.1	5.6
z z	max	11.3-	**48.6-**	2.6-	**41.3-**	1.0-	**37.2-**
	avg	0.6	**6.8**	0.2	3.4	0.1	2.2
	std.dev	0.9	12.9	0.2	8.4	0.1	5.7

Table 3: Estimation Errors For Zipfian Boxes By The Uniform Scheme.

base relation. We adopted the **random sampling technique without replacement** [Gibbons76] to obtain our sample. A random sample satisfies the property that, for a finite population and a fixed sample size n, every element in the population has the same chance of being included in the sample and every combination of of n elements has an equal chance of being the sample selected. During the sampling process, the same tuple is not picked more than once.

7.1. The Kolmogorov Statistic

Let α be the proportion of tuples in the population (relation) that satisfy a certain property. In our case, this property is that they lie within a certain query box. Let β be the proportion of tuples in the sample that lie within the same query box. Then the Kolmogorov's statistic [Gibbons76] tells us that $|\alpha - \beta| \leq d$ with probability $\geq p$ if the sample size is at least n. d is called the precision and p is the confidence. Given the values of p and d, n can be

found using standard tables. One such is reproduced here from [Gibbons76], page 73.

p/d	0.80	0.90	0.95	0.98	0.99
0.05	458	596	740	937	1063
0.10	115	149	185	231	266
0.15	51	67	83	105	119

Minimum sample size required to estimate with precision d and confidence p.

Thus, when the sample size is chosen to be 1063, we can say that $|\alpha - \beta| \leq 0.05$ with confidence ≥ 0.99. For a fixed confidence, the sample size is inversely proportional to the square of the precision.

In our experiments[4], we chose our sample size to be 1200. Thus with confidence[5] 0.99, we can say that $|\alpha - \beta| \leq 0.0471$. We present the results of our experiments in Table 4. The 5000 query boxes used for this test were the same as those used in generating Table 3. The numbers in the first three columns were obtained by building equi-depth histograms using the tuples obtained in the sample. In the last column (titled 'sample'), the estimated number of tuples within a query box was calculated using the following formula:

$$\text{est\_tuples} = \beta * N,$$

where N is the number of tuples in the whole relation. β was the actual fraction of tuples in the sample that was within the query box. In other words, we assumed that the fraction of tuples in the sample (that was within the query box) was the same as the fraction of tuples in the entire population (that was within the same query box).

Comparing the equi-depth columns of Table 3 with the corresponding ones in Table 4, we see that the sampling technique performs very well. We calculated the differences between the corresponding "avg" values in Table 3 and Table 4. The maximum difference between the corresponding "avg" entries were as follows: 0.588% (5x5), 0.892% (10x10), and 1.005% (20x20). Thus the estimates obtained from the histograms built using the sample are well within the tolerance expected (4.71%).

8. The Three Dimensional Results

We conducted the same series of experiments on three dimensions as those on two dimensions. The objective of the first series of experiments, as before, was to observe the maximum estimation error obtained by the two

[4] 1200 is the smallest multiple of 400 (20 * 20) larger than 1063.

[5] $0.0471 \approx (1063 * (0.05)^2/1200)^{0.5}$.

distr	stats	5x5 Eq.Dep.	10x10 Eq.Dep.	20x20 Eq.Dep.	Sample
n n	max	**15.9+**	**5.5+**	**4.0+**	**4.1+**
	avg	**3.1**	**1.0**	0.7	0.7
	std.dev	3.3	0.9	0.6	0.6
n u	max	8.0+	3.9+	3.0+	2.6+
	avg	1.7	0.6	0.5	0.4
	std.dev	1.7	0.5	0.4	0.4
n z	max	11.6+	4.5+	3.4+	3.3+
	avg	0.7	0.6	0.5	0.6
	std.dev	0.7	0.5	0.4	0.5
u n	max	6.5+	3.2+	3.7-	3.3-
	avg	1.5	0.9	0.9	0.9
	std.dev	1.2	0.8	0.8	0.8
u u	max	2.3-	3.9-	3.8-	4.1-
	avg	0.7	1.0	**1.1**	**1.0**
	std.dev	0.7	0.9	0.9	0.9
u z	max	4.6-	2.4-	2.5+	2.6+
	avg	0.6	0.3	0.5	0.5
	std.dev	0.6	0.3	0.4	0.5
z n	max	8.6+	3.3+	2.3+	2.3+
	avg	0.7	0.3	0.4	0.4
	std.dev	0.8	0.3	0.3	0.3
z u	max	6.4-	2.9-	3.0+	2.8+
	avg	1.2	0.8	0.7	0.7
	std.dev	1.1	0.6	0.6	0.5
z z	max	8.7-	2.3+	2.4+	2.4+
	avg	0.7	0.3	0.2	0.2
	std.dev	0.8	0.3	0.3	0.3

Table 4: Estimation Errors By Sampling With The Uniform Scheme.

schemes. We generated 5000 large cubic boxes such that they almost occupied the entire tuple space. The results are presented[6] in Table 5. The first column indicates the distributions of the attributes along the three dimensions. For a given tuple distribution, we varied the number of equi-depth buckets from 125 (5x5x5) to 1000 (10x10x10) and finally to 8000 (20x20x20). As demonstrated in Section 5.1, we can easily calculate that the maximum percentage error in estimating the selectivity by the Half Scheme for each of the above bucket configurations. For the 5x5x5 case, the maximum percentage error in estimating the selectivity by the Half Scheme is 39.2%. The corresponding numbers for the 10x10x10 and 20x20x20 case are 24.4% and 13.55% respectively. The maximum estimation error in each column of Table 5 under the Half scheme is close to the theoretically possible limits for many tuple distributions. On the other hand, both the maximum and the average estimation errors obtained by the

[6]For all the tables in this section, rather than present all the 27 rows, we have presented the results for only those rows (tuple distributions) that have either a maximum deviation or the maximum average deviation under some column.

distr	stats	5x5x5		10x10x10		20x20x20	
		Half	Uniform	Half	Uniform	Half	Uniform
n n n	max	34.2+	**20.4+**	19.4+	**8.9+**	7.4+	**4.2-**
	avg	**22.2**	**9.6**	**10.1**	**3.6**	**2.9**	**1.1**
	std.dev	9.4	5.3	5.7	2.0	2.1	0.8
u n n	max	35.1+	11.3+	20.3+	5.0+	**7.5+**	3.1-
	avg	16.9	5.6	7.8	2.1	2.3	0.7
	std.dev	9.7	3.0	5.1	1.2	1.8	0.6
u u u	max	**36.7+**	0.7-	**21.5+**	0.5-	7.1+	0.8-
	avg	11.9	0.3	4.9	0.2	1.4	0.2
	std.dev	9.2	0.2	4.5	0.1	1.4	0.1

Table 5: Estimation Errors For Large Boxes By The Two Schemes.

Uniform Scheme are significantly smaller than those obtained by Half Scheme. Again, we conclude that the Uniform Scheme performs better than the Half Scheme.

The objective of the second series of experiments was to observe the average behavior of the Uniform Scheme. We generated a total of 5000 cubic query boxes such that a large percentage of them had small volumes. This reflects real life situations wherein a large percentage of queries retrieve only a small amount of data. Table 6 gives the distribution of the volumes of the boxes. We generated histograms of equal depth as well as histograms of equal width. We estimated the number of tuples in each query box by the Uniform Scheme and the results are presented in Table 7.

From Table 7, we can see that the maximum deviations obtained with equi-width buckets are very high. In the 20x20x20 case, the maximum average deviation is only 0.299% when estimating with equi-depth buckets. On the other hand, the the maximum average deviation in the 20x20x20 case is over 9% when estimating with equi-width buckets. Notice that the maximum deviation is around 50% in the "z z z" row under all the equi-width columns.

In order to find out how valid the results under the equi-depth columns in Table 7 are, we used the method of batch means. We generated 20 batches of 1000 query boxes each such that the volumes of the boxes in each batch had approximately a zipfian distribution. The boxes were located randomly. In each batch we estimated the number of tuples in each query box using the Uniform Scheme. For each of these batches, we calculated the average percentage deviation for each tuple distribution and each equi-depth histogram configuration. All statements that follow in this paragraph assume a confidence of 90%. Out of a total of 81 cases, the variance of the average was less

range of volumes	Number of boxes
0 - 499999	1623
500000 - 999999	451
1000000 - 1499999	328
1500000 - 1999999	230
2000000 - 2499999	200
2500000 - 2999999	184
3000000 - 3499999	135
3500000 - 3999999	144
4000000 - 4499999	146
4500000 - 4999999	117
5000000 - 5499999	121
5500000 - 5999999	100
6000000 - 6499999	93
6500000 - 6999999	104
7000000 - 7499999	75
7500000 - 7999999	88
8000000 - 8499999	108
8500000 - 8999999	77
9000000 - 9499999	59
9500000 - 9999999	91
10000000 - 10499999	57
10500000 - 10999999	98
11000000 - 11499999	47
11500000 - 11999999	68
12000000 - 12499999	81
12500000 - 12999999	63
13000000 - 13499999	60
13500000 - 13999999	52

Table 6: Distribution Of The Volumes Of Query Boxes.

distr	stats	5x5x5		10x10x10		20x20x20	
		Eq.Dep.	Eq.Wid.	Eq.Dep.	Eq.Wid.	Eq.Dep.	Eq.Wid.
n n n	max	**19.2**+	9.0-	**8.2**+	15.8-	**3.2**-	29.4-
	avg	**3.6**	1.3	**1.2**	**4.2**	0.3	**9.3**
	std.dev	4.4	1.7	1.6	4.5	0.4	9.6
n z z	max	11.6-	49.8-	2.9-	44.7-	0.7-	44.5-
	avg	0.4	**6.2**	0.1	3.4	0.0	2.6
	std.dev	0.9	12.8	0.2	9.0	0.1	7.0
z z z	max	14.7-	**56.6**-	2.9-	**50.1**-	0.5-	**47.6**-
	avg	0.3	5.4	0.1	2.8	0.0	1.8
	std.dev	1.0	13.0	0.2	9.0	0.1	6.7

Table 7: Estimation Errors For Zipfian Boxes By The Uniform Scheme.

than 5% of the average in 1 case, less than 7% of the average in 42 cases, and less than 9% of the average in 72 cases. In seven other cases, the variance of the average was between 9% and 12% of the average. In only two of the remaining cases, the variance of the average was between 16% and 17% of the average. However, the variances

in these two cases were extremely small (0.008% and 0.016%). In fact, in all of the 81 cases the variance never exceeded 0.19%.

The objective of the last experiment was to demonstrate the effectiveness of the random sampling technique in building equi-depth histograms. We chose a sample size of 8000 as this was the smallest multiple of 20 * 20 * 20. The results are presented in Table 8. The format of Table 8 is similar to that of Table 4.

Comparing the equi-depth columns of Table 7 with the corresponding ones in Table 8, we see that the sampling technique performs very well. We calculated the differences between the corresponding "avg" values in Table 7 and Table 8. The maximum difference between the corresponding "avg" entries were as follows: 0.513% (5x5x5), 0.509% (10x10x10), and 0.422% (20x20x20). Thus, the estimates obtained from the histograms built using the sample are well within the tolerance expected[7] (1.82%).

It is not a coincidence that the third column is identical to the fourth column in Table 8. Since the number of buckets (S = 8000) is equal to the sample size, each bucket consists of exactly one tuple. Let n denote the set of tuples (f-buckets) within any given query box. There will be no p-buckets for any query box since each bucket is consists of a single tuple. Each bucket, by definition, has exactly $\frac{N}{S}$ tuples in it. Therefore, the number of tuples estimated to be within a query box, by the Uniform Scheme is given by $n \frac{N}{S}$. On the other hand, the fraction of tuples in the sample that lie within a query box is $\frac{n}{S}$. Since we assumed that the fraction of tuples in the sample that

distr	stats	5x5x5 Eq.Dep.	10x10x10 Eq.Dep.	20x20x20 Eq.Dep.	Sample
n n n	max	**19.4+**	**5.8+**	1.1+	1.1+
	avg	**3.6**	**0.9**	0.2	0.2
	std.dev	4.4	1.2	0.2	0.2
u u u	max	1.9-	2.6-	2.0-	2.0-
	avg	0.6	0.6	**0.5**	**0.5**
	std.dev	0.6	0.6	0.5	0.5
u z z	max	9.8-	1.6+	**2.2+**	**2.2+**
	avg	0.5	0.2	0.2	0.2
	std.dev	0.8	0.2	0.3	0.3

Table 8: Estimation Errors By Sampling With The Uniform Scheme.

[7] $0.0182 \approx (1063 * (0.05)^2/8000)^{0.5}$.

lie within the query box is the same as the fraction of tuples in the population that also lie within the query box, the estimated number of tuples in the population that lie within the query box is equal to $\frac{n}{S}$ N.

9. Conclusions

In this paper, we have demonstrated that the concept of equi-depth histograms extends very nicely to the case of multi-dimensional attributes. We presented an efficient algorithm for generating multi-dimensional histograms, the cost of which is significantly less than sorting the relation d times where d is the number of attributes. We described a main memory data structure for storing these histograms that facilitates searching for the relevant buckets. We also explored the effectiveness of the random sampling technique in building equi-depth multi-dimensional histograms at a very low cost.

Finally, we would like to point out that besides being able to compute selectivity factors accurately, a query optimizer also needs an estimate of the number of pages that will be fetched from secondary storage in order to retrieve all the tuples within a query box. Estimating the number of pages accessed when using a clustered index on a single-attribute is straightforward. The System-R query optimizer [Selinger79] assumes that the number of pages accessed when using a non-clustered index is equal to the number of tuples retrieved. We could adopt the same solution when estimating the number of pages accessed when using a multi-attribute index such as the KDB-tree or the Grid file. However, this assumption is very conservative. In the future we would like to design a data structure that can be used in conjunction with an index structure such as the K-D-B tree or the Grid file for estimating the number of pages that will be fetched when retrieving tuples within a given query box.

10. References

Gibbons76.
J.D. Gibbons, *Nonparametric methods for quantitative analysis*, Holt, Rinehart and Winston, New York (1976).

Guttman84.
A. Guttman, "R-Trees: A Dynamic Index Structure for Spatial Searching," *Proc. ACM SIGMOD Conf.*, pp. 47-57 (June 1984).

Nievergelt84.
J. Nievergelt, H. Hinterberger, and K. C. Sevcik, "The grid file: An adaptable, symmetric multikey file structure," *ACM Trans. on Database Systems* **9**(1) pp. 38-71 (March 1984).

Robinson81.
J. Robinson, "The K-D-B Tree: A Search Structure for Large Multidimensional Dynamic Indexes," *Proc.*

of the ACM SIGMOD, (1981).

Sargent76.
R.G. Sargent, "Statistical Analysis of Simulation Output Data," *Proc. ACM Symp. on the Simulation of Computer Systems IV*, pp. 39-50 (August 1976).

Selinger79.
P. Griffiths Selinger, M.M. Astrahan, D.D. Chamberlin, R.A. Lorie, and T.G. Price, "Access Path Selection in a Relational Database Management System," *Proc. ACM SIGMOD Conf.*, (June 1979).

Shapiro84.
G.P. Shapiro and C. Connell, "Accurate estimation of the number of tuples satisfying a condition," *Proc. of ACM SIGMOD*, pp. 256-276 (June 1984).

Zipf49.
G. K. Zipf, *Human Behavior and the Principle of Least Effort*, Addison-Wesley, Cambridge, M.A. (1949).