# Toward an Accurate Analysis of Range Queries on Spatial Data

Ning An, Ji Jin, and Anand Sivasubramaniam, *Member*, *IEEE*

**Abstract**—Analysis of range queries on spatial (multidimensional) data is both important and challenging. Most previous analysis attempts have made certain simplifying assumptions about the data sets and/or queries to keep the analysis tractable. As a result, they may not be universally applicable. This paper proposes a set of five analysis techniques to estimate the selectivity and number of index nodes accessed in serving a range query. The underlying philosophy behind these techniques is to maintain an auxiliary data structure, called a density file, whose creation is a one-time cost, which can be quickly consulted when the query is given. The schemes differ in what information is kept in the density file, how it is maintained, and how this information is looked up. It is shown that one of the proposed schemes, called *Cumulative Density (CD)*, gives very accurate results (usually less than 5 percent error) using a diverse suite of point and rectangular data sets, that are uniform or skewed, and a wide range of query window parameters. The estimation takes a constant amount of time, which is typically lower than 1 percent of the time that it would take to execute the query, regardless of data set or query window parameters.

**Index Terms**—Spatial data, range query, selectivity estimation, node access estimation, histogram-based estimation, R-trees.

## 1 Introduction

Spatial databases are gaining importance and are becoming prevalent in numerous applications [31]. Geographical Information Systems (GIS), image processing, navigation/positioning, demography, epidemiology, terrain analysis, mining, military planning and logistics, computer-aided design, and robotics, are just a few of the domains that can benefit from spatial databases. Efficient storage, retrieval, and processing of spatial data to reduce processing and I/O costs is crucial in these applications.

Analysis of the performance of spatial operations [31] becomes even more essential with the choice of numerous index structures [6], [7], [11], [22], [12], [28], [32], [29]. Performance analysis can help better understand the suitability of a data structure for different input data sets (both size and spatial distribution). Given a data set, we could use analysis results to objectively choose between different indexing alternatives. After choosing an index structure, analysis results could then be used to efficiently build/layout/fine-tune the structure within the purview of its definition. Finally, analysis is extremely important for query optimization. The cost and number of data items that are retrieved by a query would be very useful to determine the execution plan of a query for best performance.

There are several interesting queries that could be posed to a spatial database. These include range queries (selecting items that overlap a given query window), nearest-neighbor queries, joins, and other topological queries. Of these, the range query is, perhaps, the most common, and has been widely used as the subject of analysis in other related studies [33], [8], [23], [34], [17], [35], [26] as well. With range queries, one is interested in finding out how many data items will be retrieved (*selectivity*) and what will be the I/O complexity (number of *nodes accessed* in the index structure) in servicing the query. These two measures reflect the I/O and CPU processing costs that would be incurred by the query, with the former factor usually being more dominant. This paper focuses on estimating (analyzing) the selectivity of range queries on spatial databases. Our proposed techniques can also be used to estimate the nodes that would be accessed in servicing a range query in the associated spatial index structure. We demonstrate this by using the packed R-tree [17] as a case-study.

There have been several previous attempts at analyzing R-tree performance for range queries [9], [33], [5], [7], [23], [34], [35], [26], [17]. Most of these studies are either limited by the kind of data sets that they can be applied to, or make some simplifying assumptions along the way to keep the model tractable. As a result, they can give inaccurate results when the data sets deviate from such assumptions. Further, many previous studies examine average case behaviors (average the errors in estimation over numerous query windows), and this may not necessarily reflect certain gross errors for specific windows (and such windows could be important for an application).

The underlying philosophy of the analysis techniques presented in this paper is that they should make little or no assumption about the data set. Any information that is data set dependent should be drawn from the data set itself. As a result, these techniques are universally applicable, regardless of the data set or the application that they are used for. The common theme between these different techniques is to use an auxiliary data structure, called the *density file*, which maintains sufficient information (*histograms*) about the data set that is necessary to conduct the estimation. The density

• *The authors are with the Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA 16802. E-mail: {an, jjin, anand}@cse.psu.edu.*

*Manuscript received 3 Apr. 2000; revised 18 June 2001; accepted 20 July 2001.*
*For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number 111841.*

file creation is a one-time cost. When the query is given, the density file is "quickly" looked up to determine selectivity and nodes accessed. There has been prior work [30], [24] on using histograms for estimating query performance in relational databases, but there are only a few forays [21], [2] on such techniques for spatial/multidimensional databases. Muralikrishna and DeWitt [21] is an early study that has proposed a technique for building equi-depth histograms for multidimensional point data sets. There has been a recent study [2], concurrently undertaken with the work that is presented in this paper, on different ways of maintaining histograms for spatial databases. The differences between these studies are explained in Section 2.

This paper considers both point (sizeless and shapeless objects) and rectangular data sets, and proposes five analysis schemes for the former and two for the latter (the rectangular schemes can be used for points as well without any loss of generality). It should be noted that rectangles can also be used to abstract more complex spatial objects (as Minimum Bounding Rectangles), and the proposed schemes can be used in such cases. These schemes differ in what information is maintained in the density file, how it is maintained, and how it is looked up for estimation. Using a diverse (both real and synthetic, that are uniform or skewed) suite of data sets and different query window parameters (size, location and aspect ratio), it is shown that one of our schemes (called *Cumulative Density*), gives very accurate estimations for selectivity and nodes accessed for each query window, with errors less than 5 percent. It gives much lower errors than most of the previously proposed techniques. It provides this accuracy at a (time) cost that is less than 1 percent of the actual query execution time. The storage overheads for maintaining the density file are tolerable as well.

The rest of this paper is organized as follows: The next section gives a quick overview of previous analysis attempts on range query performance. Section 3 presents the proposed analysis techniques for point and rectangular data sets. Section 4 gives results from the analyses using a spectrum of data sets and query windows. Finally, Section 5 summarizes the contributions of this paper and outlines directions for future work.

## 2 Related Work

Estimation of range query performance on spatial data has been shown to be extremely important [3], [21]. Consequently, there is a large body of literature on this topic. These techniques, however, are limited either to the kind of data sets that they can analyze (points, rectangles, etc.), and/or make simplifying assumptions about the data set (uniform, skewed following a certain rule, etc.) or query windows.

The first set of techniques [9], [33] make an assumption about the data set being more or less uniformly distributed in space. [9] presents the first known analysis of R-trees, by transforming objects into higher dimensional space. Theodoridi and Sellis [33] uses the size of the query window to find out how many leaf nodes will be covered by the query for the uniform data set. The internal nodes that will be touched can then be examined recursively to estimate the disk accesses. Since these studies assume uniform distribution of the data, they do not have to be concerned about the location of the query window. These estimations can, however, become inaccurate for skewed data sets.

Some other studies [8], [5], though not restricted to uniform data sets, focus specifically on point data sets. Faloutsos and Kamel [8] show that certain point data sets behave as mathematical fractals and calculate the fractal dimension of the data sets. Using this, they can find out the number of disk accesses at each level of the R-tree. However, this analysis is restricted to point data sets, and also requires that the R-tree be well-built and the aspect ratio of the MBRs of its nodes be close to 1.

Similarly, Proietti and Faloutsos [23] focus on region data sets for their analysis. They show that certain region data sets can be packed into MBRs having a quite uniform aspect ratio, and the areas of these MBRs obey the REGAL law. This observation is used to estimate the number of MBRs intersecting the given query window.

Finally, there are studies [34], [35], [26], [17] which have looked at both point and region data sets, that are both uniform and skewed. Theodoridi and Sellis [34], [35] use the concept of density (average number of data entries that contain a given point) to analyze R-tree performance. The density is calculated level-by-level, beginning at the leaf. This technique assumes that the density is uniform within the data set. The authors suggest that it could be extended to nonuniform data sets, by splitting the spatial extent into multiple regions, each of uniform density. This can, however, become a nontrivial decomposition. Further, their technique assumes that the node MBRs have unit aspect ratios.

With the bulk-loaded packed R-tree structure, Kamel and Faloutsos [17] propose a simple formula to estimate the number of pages (nodes accessed) that will be retrieved by a range query, which is a function of the query window size and the average size of the Minimum Bounding Rectangles (MBR) of the R-tree nodes. This formula would apply to any R-tree (built using any technique), as long as the MBRs of the nodes are available. Since this technique does not take query window location into account, it may not always be accurate, though the accuracy may be acceptable in the average case. Aref and Samet [4] later extended this model for selectivity estimation of spatial joins. A probabilistic model for different range query classes on spatial data structures, that is relatively independent of the choice of the data structure and data set, is conducted in [25]. This study makes a couple of assumptions in deriving the model: all query windows have unit aspect ratio, and all query windows are either of the same area or request the same number of data items. Such assumptions may not necessarily be valid for real applications.

Most of the above techniques fall under what has been characterized as parametric techniques [2], which try to mathematically model the data based on certain assumptions. Spatial data sets are likely to be very diverse. Consequently, not all of the above techniques can be used to analyze the performance of all data sets. We believe that an estimation technique should make little or no assumptions about the input data set. Any information that it would need should come from the data set being analyzed itself, and this information should be provided without

adding significant overheads. This is the underlying philosophy of this paper. Techniques adhering to this philosophy would be universally applicable, regardless of the data set or the application that it is being used for. These techniques typically use auxiliary data structures, called histograms, which partition the space into buckets and keep track of how many data items fall within each bucket.

A very recent study [2], undertaken concurrently with this work, has examined different ways of constructing histograms for spatial databases to estimate selectivity, and has proposed a novel scheme (called *MinSkew*) that is shown to give fairly accurate estimations. There are several similarities between some of their suggestions and the techniques presented here. For instance, our DH scheme uses the equi-area partitioning suggested in [2], with the difference that it does not optimize empty spaces/regions. Our DHC scheme is intended for such optimizations. The NDH scheme discussed here is almost identical to the R-tree index-based grouping suggested in [2]. However, there is a key difference between the two studies. Except for the NDH scheme, all the others in this paper use equi-width (equi-area) and nonoverlapping buckets unlike the ones used in [2]. As a result, it is rather straightforward in our schemes to find the relevant buckets for a query window. Query estimation is thus fast and has very low memory requirements. Estimation for the schemes in [2], on the other hand, requires a search to find the relevant buckets. As a result, those schemes try to keep the number of buckets relatively small so that they fit in main memory. The techniques detailed here do not have such restrictions, and we can potentially go for a large number of buckets for better accuracy. We are able to maintain nonoverlapping buckets even with rectangular data items, using a novel idea (derived from simple geometric properties of rectangles) whereby a rectangular object is counted in exactly one bucket. To our best knowledge, no previous study has pursued such an idea.

Another common observation about all the above studies is that the estimation accuracy is evaluated using average case behavior (i.e., numerous query windows are fed to the model and the error in estimation is averaged over all these queries). While this may be a viable approach to discuss the overall quality of different modeling techniques, it is important to note that there can be gross inaccuracies for certain specific windows (and such windows may be important workloads for an application). Instead, one should try to conduct studies with different query window sizes and locations, and try to understand the accuracy of the estimation for each of these windows.

## 3 Analysis Techniques

There are two main costs in searching for objects intersecting a rectangular query window. The first is the cost of computing the intersection between the data entries and the query window. The second is the cost of retrieving the items from the disks. The number of data items that will be retrieved (called the *selectivity (s)*) has a direct bearing on both these costs. Further, the retrieval cost will also depend on the *number of nodes (n)* in the index structure that will be touched by the query. In the rest of this discussion, we present a set of techniques for estimating the selectivity for point and rectangular spatial data sets. We also illustrate how these techniques can be used to estimate the number of nodes in the index structure that will be accessed, using the packed R-tree structure [17] as a case-study.

The R-tree [11], [18], proposed as an extension to the B-Tree structure, is one of the most popular spatial data structures. Many variants of the R-Tree, such as $R+$-Tree [32] and $R^*$-Tree [6] have been proposed. They differ in the algorithm that is used for insertion, specifically in splitting a node of the tree when its subtree is filled. They attempt to give better balanced (and efficient) trees by dynamically adapting to the insertion pattern/sequence. However, these structures can become inefficient when the database of spatial items is static (and known a priori). In such cases, one should use bulk-loading techniques rather than insert item by item to build the data structure. Roussopoulos and Leifker [27] use packed R-trees for such static databases to lower response times. Further, Kamel and Faloutsos [17] suggest using Hilbert value (a linearization technique for multidimensional space [10], [13]) as the index for the bulk-loaded R-tree. This would help optimize (localize) the number of nodes traversed in serving a query. Typically, such R-trees are built in a bottom-up fashion, level by level.

There are three main reasons for using bulk-loaded, packed R-trees to illustrate node access estimation. First, our research project [1] is exploring efficient database support for geographic information (GIS), and the data sets in this domain are usually static. Many other applications use static spatial data sets as well, and our techniques would apply to all these domains. Second, packed R-trees have high space utilization (close to 100 percent) compared with their dynamic counterparts. This reduces the size of the tree, localizes (optimizes) the nodes traversed in serving a query, and reduces response times. Finally, without loss of generality, bulk-loaded R-trees help us keep our analysis of node access estimation relatively simple. It is for these reasons that other similar studies [17] analyzing the performance of R-trees have used bulk-loaded bottom-up R-trees as well. The reader should note that the use of a packed R-tree to illustrate node access estimation does not in any way mitigate the generalization of our schemes. One could very well feed the bounding boxes of the nodes in the index structure to the selectivity estimation process to find the number of nodes that will be accessed. In fact, our rectangle data set schemes follow this approach. The bulk-loaded R-tree, that we use, is built from a sorted list (based on Hilbert ordering as suggested in [17]) of the data items in a bottom-up fashion.

For the data sets, we focus on points and rectangles, which capture interesting facets of spatial information. Points are shapeless/sizeless objects, while this is not the case for rectangles. Rectangles can also be used to represent/abstract other objects (such as lines, polylines, polygons, etc.). In such cases, the rectangles represent the Minimum Bounding Box of the objects, and there needs to be an additional step (called the refinement step) to process the retrieved rectangles (from the filtering step) and find out which ones actually pertain to the query. The techniques presented here will then apply to an analysis

TABLE 1
Common Symbols and Definitions

| Symbols | Definitions |
|---|---|
| $h$ | Hilbert order ($4^h$ cells) for point datasets |
| | Level of gridding ($4^h$ cells) for rectangle datasets |
| $S_e$ | Size in bytes of a density value |
| F | Node capacity or fanout |
| $N_s$ | Dataset size |

of the performance of the filtering step. It has often been argued that perhaps this is a more dominant cost of a range query since it requires I/O.

### 3.1 Overview

Our techniques can be briefly summarized as follows: We construct an auxiliary data structure (which we call the *density file*) from the original data set, in addition to the R-Tree at the time of building. This density file contains sufficient information—how many data items are contained in different regions/cells of the spatial extent—about the data set. When a query is given, the density file is *quickly* looked up to procure the necessary information. The size of the density file, the time for constructing this file, and the time for looking up the required information are the issues that one needs to keep in mind, as will be discussed for each technique. The techniques differ in what information is kept in the density file, how it is kept, and how the information is looked up. Some common symbols/definitions that are used in the following discussion are given in Table 1.

### 3.2 Techniques for Point Data Sets

Point data has only position information, making them easier to process. When a (rectangular) query window is given, we need to find out how many points of the data set fall within this window (selectivity). The leaf nodes that the selected points fall on is determined by the Hilbert values of the points. Subsequently, we use a recursive procedure to find out how many internal nodes of the R-tree will be accessed to get to these leaf nodes.

The common theme in the following schemes is to first partition the spatial extent into grid cells in the same way that is used to assign a Hilbert ordering (number) for the data items [10]. The density file is then just a histogram of the number of points that fall within a specific Hilbert range. It is important for the reader to note that the Hilbert order [10] (the level to which the spatial extent is recursively broken down) will have an important effect on the size of the density file as well as on the accuracy of the estimation. The schemes differ in how the histogram is maintained within the density file.

#### 3.2.1 The Density Histogram (DH) Scheme

The DH scheme uses a straightforward representation of the density information. The density file contains the number of points that fall within each Hilbert grid cell, and the file is maintained in increasing Hilbert cell order. The Hilbert cell number can be used as an offset into the file to directly get the corresponding density (number of points within this cell). The size of the file is thus dependent on the number of grid cells, which in turn is a function of the Hilbert order that is used to recursively break down the spatial extent. For a given query window, the selectivity ($s$) and number of nodes traversed ($n$) can be estimated as follows.

**Selectivity**. The query window is broken down into a sequence of Hilbert ranges that it covers, based on the Hilbert order that has been used to create the density file. There is an approximation being made here in aligning/extending the query window to the boundaries of grid cells. It should also be noted that there is a sequence of (potentially noncontiguous) Hilbert ranges that are generated from the query window. These ranges are looked up in the density file to find out how many points fall within each range (density), and then the densities are added up to get the selectivity. To make it more efficient, we keep cumulative densities (sum of densities from grid cell 0 until that grid cell) in the density file, so that finding the density within a range will require looking up just two values (the ends of the range) and subtracting one from the other.

**Nodes Accessed**. We could find the number of nodes accessed by a query if we had some knowledge about which leaf nodes the selected data items reside on. It is rather easy to figure out this information from the packed R-tree algorithm since the leaf nodes contain data items sorted by Hilbert order (and each leaf node contains the same number of data items). As with the selectivity method, we can break the query window into Hilbert ranges. For a range, we could look up the density (specified as a cumulative density from grid cell 0) information for the lower end of the range. This number divided by the number of data items pointed to by a leaf node would specifically identify the leaf node where the range starts. Similarly, the density information for the end of the range can be used to find out on which leaf node the range ends. Once we identify all the leaf nodes that will be accessed, we can use the same method to recursively move up the tree to find out what nodes will be accessed at each level.

#### 3.2.2 The Density Histogram Compression (DHC) Scheme

The problem of the DH Scheme is the storage space. For each Hilbert cell, a (cumulative) density value is stored in the file, regardless of whether there are any data items present in that cell or not. The size of the density file grows exponentially with the Hilbert order. Note that the higher Hilbert order would be the level of accuracy most of the time. Hence, if one wants to tolerate a relatively low estimation error, then the DH scheme would not be a very scalable alternative. The DHC scheme tries to compress the density file information of the DH scheme with the same underlying algorithms used to find selectivity and nodes accessed.

When examining the density file of the DH scheme carefully, one can find characteristics that can help optimize space requirements. First, when the Hilbert order is high, the spatial extent gets divided into very small grid cells, with a large number of cells having no data items within them. Second, neighboring grid cells tend to have similar densities, and densities change more gradually from one

cell to another. This is usually the case for data sets with points uniformly distributed in the spatial extent. Even for skewed data sets, we can have regions within which the grid cells have similar densities.

These observations suggest that we can reduce the size of the density file by compressing the storage required to represent neighboring grid cells with similar densities. The scheme can be explained as follows: We initially start with the density file of the DH scheme. We compare the density of each set of four consecutive grid cells (recall that Hilbert ordering recursively breaks down the space into four regions). If they are similar (close enough), then they are represented by only one entry in the density file. The "similarity" check is done by comparing the coefficient of variation (standard deviation divided by mean of the density values for the four cells) with a certain threshold. If the value is less than the threshold, then the four cells are combined into one value, else they are left as four cells. This procedure is then recursively carried out for the next lower level of the Hilbert order, etc. The recursion stops when either there are no more grid cells to be merged, or when the number of grid cells to be considered is just one.

This scheme is expected to lower the size of the density file compared to the DH scheme, albeit at a higher cost required to build the density file. Furthermore, finding the offset in the file for a particular grid cell is no longer straightforward as in the DH scheme. A slightly higher price has to be paid during estimation. We use a simple index structure to improve the performance of the lookup operation on the compressed density file. Other than this, the selectivity and nodes accessed estimation algorithms are the same as for the DH scheme.

#### 3.2.3 The Node-Based Density Histogram (NDH) Scheme

Another way of reducing the size of the density file is by keeping the information at a slightly coarser level. In the DH scheme, the information was maintained at a grid cell granularity, thus (potentially) giving a finer level of accuracy in estimating both selectivity and nodes accessed. Instead, we maintain the file on an R-tree leaf node basis (i.e., one entry in the density file for each leaf node of the tree, with each entry containing the Hilbert range of the cells covered by that node together with the number of data items within that range). The selectivity and nodes accessed are calculated as follows.

**Selectivity**. Convert the query window into a sequence of Hilbert ranges as before. Next, we find the leaf nodes that intersect these ranges from the density file. Experimentally, we have found that using a binary search within the density file to find the leaf node that intersects with the start of the first range, and then a linear search from that point for subsequent ranges works rather well. For each intersecting leaf node, we approximate the number of data items that would be retrieved as $x$ percent of the items within that node, where $x$ is the percentage of the node's Hilbert range that intersects the query window. This approximation assumes that the data items within a leaf node are uniformly distributed within the Hilbert range of that node. By summing this number over all the intersecting leaf nodes, we get the required selectivity.

**Nodes Accessed**. The same algorithm used in the DH scheme, which calculates the internal nodes that are accessed after the leaf nodes have been identified, is used here as well.

It should be noted that the density file of NDH essentially maintains equi-depth histograms [21]. Each bucket corresponds to an R-tree node, with the fanout determining the depth of the bucket. A similar scheme has been used as one of the options in a more recent study [2]. However, since the points are sorted by Hilbert order in our approach, there is not much (consecutive buckets may at most have one Hilbert value in common) overlap between the buckets.

#### 3.2.4 Comparing Point Data Set Schemes

Before we proceed to rectangular data set schemes, we give a brief summary of the three schemes for points in Table 2. We use four criteria in discussing each scheme. These include the size of the auxiliary data structure (density file) that they use, the time for creating/building this auxiliary data structure (which is a one-time cost incurred at the time of building the R-tree), the time taken for estimating the selectivity and nodes accessed, and the accuracy of the estimation.

DH requires a large density file ($4^h * S_e$), which grows exponentially with the Hilbert order. DHC compresses this file, based on the similarity between successive grid cells. The degree of compression would depend on the variation of the number of data items that fall in successive grid cells, together with the threshold that is used to qualify this variation as significant to warrant extra storage. The density file size is independent of Hilbert order for the NDH scheme, clearly making it a winner in terms of the size criterion. The density file size in NDH ($\frac{N_s}{F} * S_e * 2$) is directly proportional to the number of leaf nodes in the corresponding R-tree, with each leaf node requiring two Hilbert values (to indicate a range).

To create the density file in the DH scheme, we need to histogram the points based on which Hilbert grid cell they fall on. While one could use this straightforward approach for creating the density file, the performance may be rather poor since successive data items may fall on different grid cells (if the data set is not sorted) resulting in a lot of I/O operations (the file can get quite large). Instead, we first generate the Hilbert values for the points (which would be done in any case to build the bottom-up R-tree), sort them based on these values, and then histogram them to avoid thrashing effects. The creation of the density file for DHC would be even more expensive since the density file has to be compressed based on the coefficient of variation. NDH takes very little time to build the auxiliary structure. Since the points are already sorted based on Hilbert order (to build the R-tree), we can just examine the two extreme points of each leaf node to find out its range.

TABLE 2
Point Data Set Schemes at a Glance

| | DH | DHC | NDH |
|---|---|---|---|
| Density File Size | $4^h * S_e$ | Smaller than DH Depends on $h$, threshold and dataset patterns | $\frac{F}{N_s} * S_e * 2$ |
| File Creation Time | Time to generate Hilbert values + Time to sort + Time to histogram | Larger than DH | Smaller than DH |
| Estimation Time | Time to convert query window to Hilbert ranges + Time to lookup density file for each range | Higher than DH | Higher than DH |
| Estimation Accuracy | Depends on $h$ | Depends on $h$ and threshold | Depends on $h$, uniformity in leaf nodes |

In all three schemes, the first step in estimation is to break up the given query window into a sequence of Hilbert ranges. This would itself take up time that is dependent on the Hilbert order. Next, selectivity is determined by examining the density file for these ranges. This is a straightforward lookup (using the Hilbert value as an offset) of the density file for DH. DHC requires extra time since there is another index structure within this file to get to the information. NDH requires binary and some linear searches within this file to find intersecting leaf nodes, which perhaps is the more expensive of the three. Subsequently, estimation of nodes accessed uses the same algorithm in all three schemes.

Finally, the estimation accuracy is largely dependent on the Hilbert order that is used for linearizing the data set and breaking down the query window for all three schemes. For the DHC scheme, it is also dependent on the threshold that is used to detect variation in densities. In the NDH scheme, the accuracy of the selectivity is dependent on the spatial distribution of the data points contained within each leaf node of the R-tree.

## 3.3 Techniques for Rectangle Data Sets

Rectangle data sets pose a more difficult problem than point data sets since they contain size information in addition to position information. The point data set schemes may not necessarily work for rectangle data sets because of this extra dimension to the problem. One could think about extending two-dimensional Hilbert space into three or more dimensions (as was mentioned in [17]), but that would need a high amount of computation. Furthermore, it is not straightforward to convert a query window into three-dimensional Hilbert ranges while still maintaining the spatial relationships, i.e., objects that fall into those ranges should spatially intersect the query window.

We propose two schemes below that use simple geometrical properties of rectangles to address this problem, while still providing nonoverlapping buckets. There are a couple of differences from the previous schemes. In the point data set schemes, Hilbert space and ordering was used to grid the spatial extent. In the following two schemes, we do not really care because there is no need to get a linearization of the spatial extent. Instead, the spatial extent is gridded into cells by just drawing a number of vertical (columns) and horizontal (rows) lines. A cell is then denoted by its row and column. The density file is looked up by using a two-dimensional offset (a row and column number).

The second difference from the point schemes is in what each entry of the density file contains. We cannot keep just the center point information of the rectangular items (i.e., each cell contains the number of rectangles whose center points fall within that cell) since this would loose the size information. Neither can we record how many rectangles intersect each cell since we would end up double/multiple counting the rectangles in estimation. In the following two schemes, we illustrate what we need to maintain within each grid cell to avoid the multiple counting of rectangles without sacrificing size information.

### 3.3.1 The Incremental Density (ID) Scheme

In this scheme, the density file keeps track of the information on an incremental basis. Specifically, for each grid cell we keep two values: a) the number of rectangles whose bottom side/edge falls (intersects) on that cell ($DS(i,j)$), and b) the number of rectangles whose top side/edge falls (intersects) on that cell ($DE(i,j)$). This is shown pictorially on the right side of Fig. 1 for the rectangular data set on the left side.

**Selectivity**. We can find out how many rectangles ($N(qxl, qyl, qxh, qyh)$) intersect with a query window $(qxl, qyl, qxh, qyh)$ (the lower-left corner is $(qxl, qyl)$ and the upper-right corner is $(qxh, qyh)$) as follows: Let $S(xl, yl, xh, yh)$ denote the number of rectangles that start in region $(xl, yl, xh, yh)$ (i.e., whose lower edges intersect with this region), and let $E(xl, yl, xh, yh)$ denote the number of rectangles whose top edges intersect with this region. We can then use the following equation to calculate $N$:

$$N(qxl, qyl, qxh, qyh) = S(qxl, 0, qxh, qyh) - E(qxl, 0, qxh, qyl - 1). \quad (1)$$

Fig. 2 illustrates this observation with an example. The query window covers $(1, 2, 2, 3)$, for the data set with 11 rectangles numbered $a$ through $k$. For this example, $S(1, 0, 2, 3) = 11$ (i.e., all the 11 rectangles), and $E(1, 0, 2, 1) = 4$ (i.e., rectangles $a$, $b$, $c$, and $d$). So, $N(1, 2, 2, 3) = 11 - 4 = 7$.

We determine $S$ and $E$, from $DS$ and $DE$, respectively, as follows: Let $RS([xl, xh], j)$ represent the number of rectangles whose lower edges intersect with grid cells of row $j$ between columns $xl$ and $xh$.

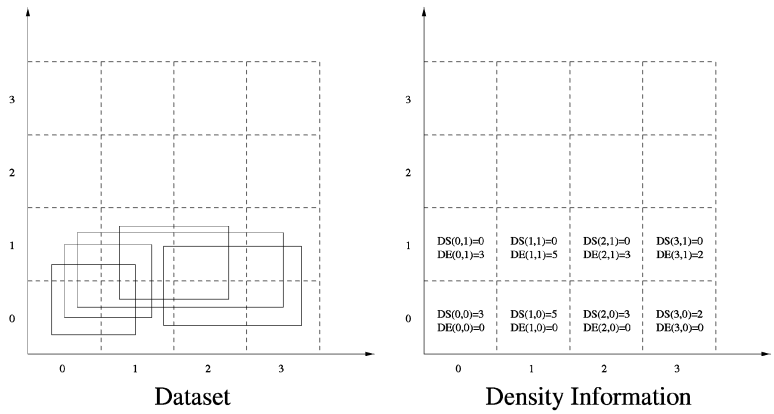


Fig. 1. Densitity information in the ID Scheme.

$$RS([xl, xh], j) = DS(xl, j) + \sum_{i=xl+1}^{xh} MAX(DS(i, j) - DS(i-1, j), 0). \quad (2)$$

For example, in Fig. 1, $RS([0, 2], 0) = 3 + (5 - 3) + 0 = 5$, which means that there are five rectangles whose lower edges intersect with grid cells (0,0), (0,1), and (0,2). The following equation can then be used to calculate $S$,

$$S(xl, yl, xh, yh) = \sum_{j=yl}^{yh} RS([xl, xh], j). \quad (3)$$

Similarly, let $RE([xl, xh], j)$ represent the number of rectangles whose top edges intersect with grid cells of row $j$ between columns $xl$ and $xh$. We can then calculate $E$ from $DE$ as follows:

$$RE([xl, xh], j) = DE(xl, j) + \sum_{i=xl+1}^{xh} MAX(DE(i, j) - DE(i-1, j), 0) \quad (4)$$

$$E(xl, yl, xh, yh) = \sum_{j=yl}^{yh} RE([xl, xh], j). \quad (5)$$

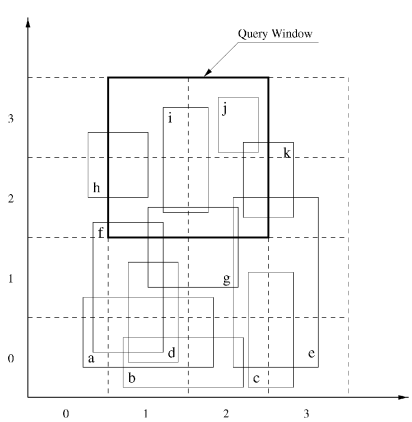


Fig. 2. Calculating rectangles in a query window.

$RS$ and $RE$, however, are not always accurate. For example, let us look at a situation where two neighboring rectangles have edges that fall on adjacent columns of a row as shown in Fig. 3a. This scheme will not differentiate between this and a single large rectangle (shown in Fig. 3b). In both cases, this scheme will result in the same density information. The CD scheme, to be discussed next, does not suffer from this inaccuracy.

**Nodes Accessed**. We cannot use selectivity information directly to estimate the nodes accessed (as in the point data set schemes) because the density information is not maintained as Hilbert grids, i.e., after the selectivity is obtained, we do not know the Hilbert values for the data items. Although it is possible to divide the universe using Hilbert order as in the point data set schemes, we would need extra storage and longer computation times (the above equations for selectivity are based on simple geometric properties of rectangles, and there needs to be a level of translation before they can be used if we used Hilbert gridding). Instead, we use an alternate solution using the property that each node in the R-tree can itself be represented by a rectangle in the spatial extent (the Minimum Bounding Rectangle covering its subtree). It is thus sufficient to examine if this MBR intersects the query window to find out if this node would be accessed. As a result, we maintain the MBRs of all the R-tree nodes (this doubles the space requirement if we use the same degree of gridding as with the selectivity), and use these MBRs themselves as the data for the above selectivity procedure. This would directly give us the number of MBRs (nodes) that intersect the query window.

It should be noted that since we are not using Hilbert grids, or making any other assumptions about the way the R-tree is built, the ID scheme is independent of the algorithm that is used to create the R-tree.

### 3.3.2 The Cumulative Density (CD) Scheme

The main problem with the ID scheme is in the time it takes for estimation, and to a lesser extent the inaccuracy that was pointed out in Fig. 3. We need to go through each row between 0 and $qyh$, and check the columns in the $[qxl, qxh]$ range to serve a query $(qxl, qyl, qxh, qyh)$. This becomes expensive with a fine level of gridding (which would increase accuracy), or with large query windows. There is a similar problem with the point data set schemes, and we

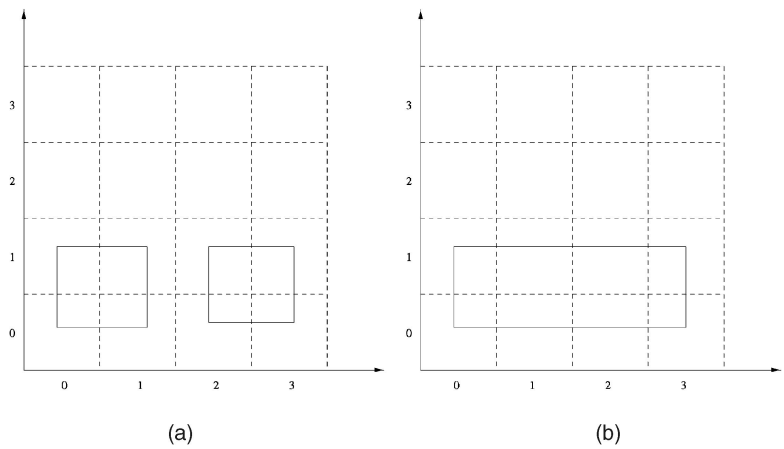


Fig. 3. Errors in estimation for ID. (a) Data set A and (b) data set B.

have used a cumulative density information to alleviate this problem there. This technique can be used here as well, which gives us the CD scheme.

We grid the spatial extent as in the ID scheme and we keep four values for each cell $(i, j)$:

- $BS'(i,j)$ (if $BS(i,j)$ is the number of rectangles whose lower-left corners lie in the range $(0,j)$ to $(i,j)$, $BS'(i,j) = \sum_{x=0}^{j} BS(i,x)$);
- $BE'(i,j)$ (if $BE(i,j)$ is the number of rectangles whose lower-right corners lie in the range $(0,j)$ to $(i,j)$, $BE'(i,j) = \sum_{x=0}^{j} BE(i,x)$);
- $US'(i,j)$ (if $US(i,j)$ is the number of rectangles whose upper-left corners lie in the range $(0,j)$ to $(i,j)$, $US'(i,j) = \sum_{x=0}^{j} US(i,x)$);
- $UE'(i,j)$ (if $UE(i,j)$ is the number of rectangles whose upper-right corners lie in the range $(0,j)$ to $(i,j)$, $UE'(i,j) = \sum_{x=0}^{j} UE(i,x)$).

The selectivity $N(qxl, qyl, qxh, qyh)$ can then be calculated as follows:

$$RS([xl, xh], j) = BS(xh, j) - BE(xl - 1, j), \tag{6}$$

$$S(xl, 0, xh, yh) = BS'(xh, yh) - BE'(xl - 1, yh), \tag{7}$$

$$RE([xl, xh], j) = US(xh, j) - UE(xl - 1, j), \tag{8}$$

$$E(xl, 0, xh, yh) = US'(xh, yh) - UE'(xl - 1, yh), \tag{9}$$

$$\begin{aligned} N(qxl, qyl, qxh, qyh) &= S(qxl, 0, qxh, qyh) \\ &\quad - E(qxl, 0, qxh, qyl - 1) \\ &= BS'(qxh, qyh) - BE'(qxl - 1, qyh) \\ &\quad - [US'(qxh, qyl - 1) \\ &\quad - UE'(qxl - l, qyl - 1)]. \end{aligned} \tag{10}$$

Instead of examining all density values in the range $[xl, xh]$, we need to access only two values for calculating S, and two for E. Thus, the estimation of selectivity and nodes accessed (a similar method of calculating selectivity with the MBRs of the R-tree nodes as explained with ID can be used to find out nodes accessed) require constant (four disk accesses and three arithmetic operations) time. This time does not depend on the query window size nor the level of gridding (we can use a very fine level for higher accuracy without compromising on estimation time). Further, the CD scheme avoids the inaccuracies shown in Fig. 3 of the ID scheme.

#### 3.3.3 Comparing Rectangle Data Set Schemes

A quick summary of the two rectangle data set schemes is given in Table 3, comparing the size of the density file that is generated, the time taken for generating this information, the time taken for estimation, and the accuracy of the estimation.

The size of the density file in both schemes ($2 * 2 * S_e * 4^h$ for ID and $2 * 4 * S_e * 4^h$ for CD) depends on the number of grid cells ($4^h$). ID uses two variables for each cell, while CD requires four. In addition to selectivity, the density information has to be maintained for the node MBRs for node access estimation, which doubles the size of the density file. The time taken for generating this density file is proportional to the number of grid cells (for the rectangles and for the MBRs of the R-tree nodes). The CD scheme takes at least twice the time taken by the ID scheme since it maintains twice the number of variables with each cell and its variables are cumulative variables.

Estimation is expensive for the ID scheme since it is proportional to the size of the query window as well as the level of gridding. On the other hand, the CD scheme is independent of these factors and takes a constant time for selectivity and node access estimation. The CD scheme is also expected to be more accurate than ID because cumulative information can lower estimation errors in some cases as was illustrated earlier.

It is also interesting to note that a point data set can be viewed as a special class of rectangular data (of size 0). Consequently, *the rectangle data set schemes can be used to estimate selectivity and nodes accessed of point data sets as well.* We have used the CD scheme for estimation of point data sets in the following evaluation studies and compare it with the point data set schemes.

### 3.4 Dynamically Updating Density File

While we have discussed the approach for creating the density/histogram information for all the above schemes, we would also like to point out that any updates (inserts/deletes) to the data set can also be reflected in the density information.

TABLE 3
Rectangle Data Set Scheme at a Glance

| | ID | CD |
|---|---|---|
| Density File Size | $2*2*S_e*4^h$ | $2*4*S_e*4^h$ |
| File Creation Time | Time to generate density of rectangles + Time to generate density of node MBRs | At least twice as ID build time |
| Estimation Time | Proportional to $h$ and window size | Constant |
| Estimation Accuracy | Higher $h$ gives higher accuracy | Depends on $h$, more accurate then ID |

In NDH, where the histogram directly corresponds to the leaf nodes of the R-tree, updating an entry in the leaf node only requires updating the corresponding bucket of the histogram. When the update results in changing the leaves of the index structure, then a similar redistribution has to be performed for the histogram information as well. The ID scheme requires touching only the grid cells that are affected by the update, and is thus fairly efficient as far as dynamic updates are concerned.

On the other hand, schemes such as DH, NDH, and CD are a little more expensive since they cumulate the information. One way of alleviating this cost is to not reflect the changes to the density information immediately, but to let them accumulate. At some time when there are a large enough number of updates, to make a significant impact on the accuracy of the estimation, they can be carried out simultaneously to amortize the cost. In fact, this approach can be used for all the schemes to keep dynamic update costs fairly low.

## 4 Evaluating the Analysis Techniques

To evaluate the different schemes described in the previous section, we conduct extensive experiments with several point and rectangle data sets, and several query windows. These studies have been conducted on a 170 MHz SUN Ultra Enterprise 1 server (the choice of the machine is immaterial since we are comparing relative performance of schemes on the same system). In the following discussion, we briefly discuss the data sets considered, examine the metrics/criteria used for comparing the schemes, and present the results for specific query windows as well as in the average case. Our schemes are also compared with some of the previously proposed ones.

### 4.1 Data Sets

We have considered a wide spectrum of point and rectangular data sets that are either uniformly distributed in space or exhibit some kind of clustering (we use the terms clustered and skewed synonymously in this paper). Some of them have been obtained from actual/real data sets (such as the Tiger [19] data), while others have been synthetically generated. The reader is referred to [16] for the results on all the data sets and, in this paper, we present results for the following data sets:

- Four point data sets:
  - **SUP**: Synthetic Uniform Points, containing $240,000$ points,
  - **SCP**: Synthetic Clustered Points, with $240,000$ points clustered around one location,
  - **TOP**: Topological Point data set taken from [25], with $478,786$ points following a rather regular pattern (points are arranged in regular rows with significant gaps between successive rows), and
  - **CFD**: Computation Fluid Dynamics data set taken from [20], with $208,688$ points that are clustered.
- Four rectangular data sets:
  - **SUR**: Synthetic Uniform Rectangles, with $500,000$ uniformly distributed rectangles,
  - **SCR**: Synthetic Clustered Rectangles, with $240,000$ rectangles that are clustered around the center,
  - **PAR**: a data set containing the MBRs of rivers of Pennsylvania from the TIGER database [19], with $30,218$ rectangles, and
  - **CAR**: a data set containing the MBRs of the streets of California from the TIGER database, with $248,643$ rectangles.

A pictorial view of these data sets is given in Figs. 4 and 5.

We have considered various query windows whose locations are randomly picked with the condition that they cover both sparse and clustered regions of the given spatial data set. These query windows also have different aspect ratios and sizes (1 percent, 5 percent, 25 percent, 50 percent, and 100 percent). The reader is referred to [16] for more detailed information about these queries. We believe that the chosen data sets and query windows capture sufficiently diverse workloads with interesting properties to stress the pros and cons of different schemes.

### 4.2 Metrics/Criteria

We have developed a bulk-loaded packed R-tree based on Hilbert order [10], [13] for each of the above data sets, which we use for comparison. We use six criteria for discussing the pros and cons of each scheme:

- *Selectivity Estimation (s)*. This measures the accuracy of the scheme in estimating the number of data items retrieved for the specific query. It is expressed as a percentage error with respect to the number of items retrieved by the query on the actual R-tree.
- *Selectivity Estimation Time (st)*. This measures the time taken by a scheme to estimate selectivity for the specific query. It is expressed as a percentage of the time to execute the query on the actual R-tree.

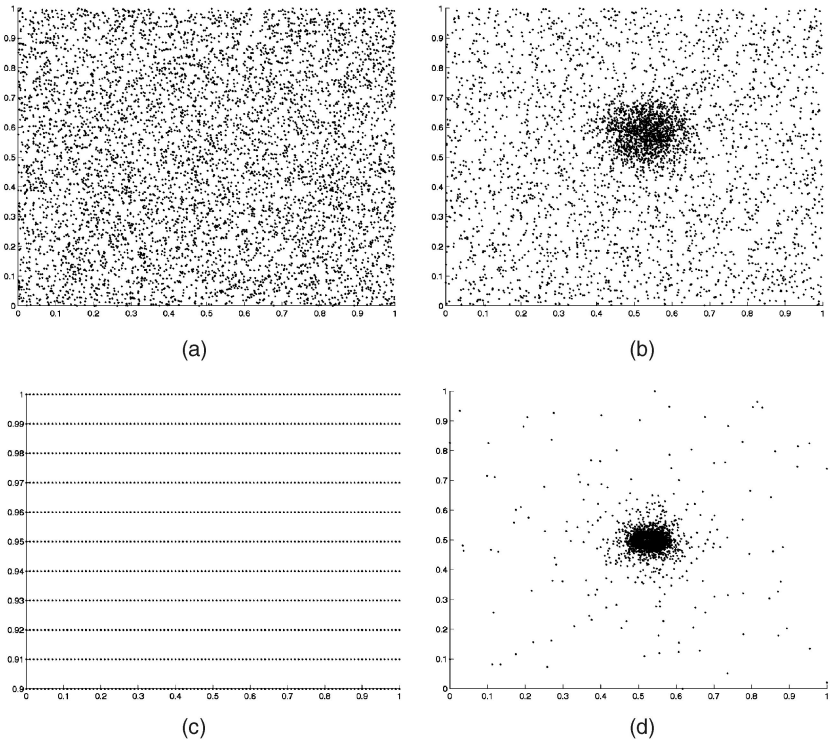


Fig. 4. Point Data Sets. (a) SUP, (b) SCP, (c) TOP, and (d) CFD.

- *Node Access Estimation (n)*. This measures the accuracy of the scheme in estimating the number of nodes accessed/touched in serving the specific query (which is an indication of the I/O costs). It is expressed as a percentage error with respect to the number of nodes touched by the query on the actual R-tree.
- *Node Access Estimation Time (nt)*. This measures the time taken by a scheme to estimate the number of nodes accessed/touched by the specific query. It is expressed as a percentage of the time to execute the query on the actual R-tree.
- *Density File Size (d)*. This is a measure of the storage overhead (in bytes) to maintain the density information required by each scheme. It is expressed as a percentage of the storage taken by the actual R-tree.
- *Time for Building Density File (dt)*. This measures the time taken by a scheme to create the density file. It is expressed as a percentage of the time taken to build the actual R-tree.

The reader should note that a relatively small $s$ and $n$ is preferable with low $st$ and $nt$. Though one would like to have a low $dt$ and $d$ as well, it should be noted that $dt$ is a one-time cost, and $d$ may not be a big issue these days with ever increasing disk storage capacities (as long as density files are not larger or become a large fraction of the actual data set/R-tree).

### 4.3 Point Data Set Results

Fig. 6 shows a part of the results for the different schemes with the point data sets. We present representative results from four query windows for each data set. We have obtained results for each scheme using different levels/orders ($h = 5, 6, 7, 8, 9$) for the density file. Instead of presenting all those results, for each workload-scheme combination, we present only those for the lowest level/order (since this will have the lowest $d$ and $dt$) which gives a satisfiable degree (less than 5 percent error) of accuracy. In case, none of these levels gives an error lower than 5 percent, then we give the results for $h = 9$. Consequently, different schemes could have different orders/levels ($h$) for a particular workload (a scheme with a higher $h$ usually indicates that it does not do as well as a scheme with a lower $h$). The reader is referred to [14] for actual values if needed.

**Accuracy** ($s$ **and** $n$). As can be expected, the DH and NDH schemes give better accuracy with higher $h$. This is specifically the case for selectivity estimation since these schemes align the query window to the grid cells. In general, higher $h$ will reduce the change in query window that is needed, giving higher accuracy. There are certain situations where a finer level of gridding can result in a higher percentage change in an area of query window compared to a coarser level of gridding (see [16] for some examples). The overall trend, however, indicates that there will be a higher level of accuracy with these schemes with higher $h$. In fact, one could theoretically hypothesize that, at very high $h$, these schemes should come very close to the actual R-tree results. But, we find that the estimated selectivities and nodes accessed are much lower. After a closer examination, we found that this is due to some points falling directly on cell boundaries. Such points are histogrammed into only one of the buckets to avoid double-counting (see [16]). Consequently, a query which is aligned to that grid cell and does not include the Hilbert value assigned to the point, will not count the point in its estimation. This also explains why the TOP data set (where

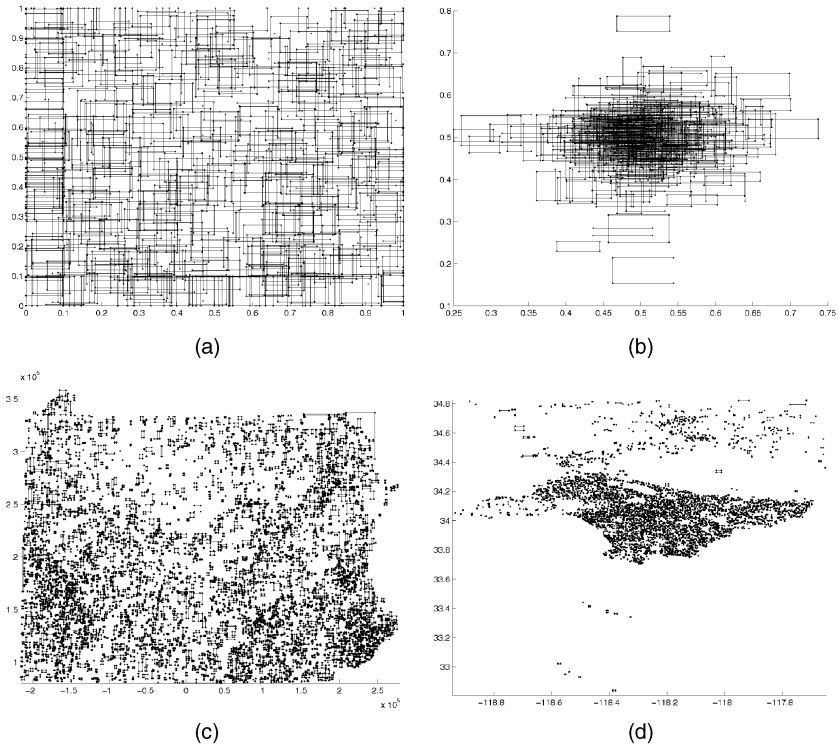


Fig. 5. Rectangular Data Sets. (a) SUR, (b) SCR, (c) PAR, and (d) CAR.

several points could lie on grid cell boundaries because of the nature of the data set) shows unstable results. Despite this, we find that these inaccuracies are not a major problem from our experiments, and we can still get less than 5 percent error in most cases.

In terms of node access estimation, NDH uses the same technique as the DH scheme, and gives similar results. For selectivity estimation, NDH assumes that data items are uniformly distributed in space within each leaf node (or at least at the nodes which are at the boundaries of a Hilbert range covered by the query). However, this assumption does not seem to hurt accuracy very much. This is, perhaps, because of two factors. The assumption is made only for the nodes that partially overlap a Hilbert range covered by a query, and this number is relatively low (as a percentage) compared to the total number of selected data items. Furthermore, even clustered data sets have some semblance of uniformity within small/isolated regions (such as those covered by a single leaf node of the R-tree) and can be approximated accordingly.

The performance of the DHC schemes with three different thresholds (0.6, 1.2, and 1.8) for the coefficient of deviation are given in Fig. 6. It can be seen that accuracy for thresholds of 1.2 and 1.8 is not acceptable in several cases, and only 0.6 comes close to the other schemes.

As mentioned in the previous section, the rectangle schemes can be used for point data sets as well. In the tables, we have shown the results for the point data sets using CD. We can observe that CD gives similar accuracies as DH for selectivity estimation and higher precision for node access estimation. This is because it uses the actual R-tree information (MBRs of nodes) to estimate this information. So, any approximation made in determining selectivities is not carried over to node estimation.

For most combinations of data sets and query windows, the DH scheme estimations were less than 10 percent error at order/level 6, and less than 5 percent error at order/level 8 [16]. A similar observations holds for the DHC, NDH, and CD schemes as well. Hence, level 8 seems to be an appropriate operating point.

Apart from the nature of the data set (uniform/clustered) and level of gridding, three other factors have an important effect on the accuracy of these schemes, namely, the location of query window, the size of the query window, and the size of the data set. The query window (0.10,0.61,0.96,0.90) covers a relatively sparse area of the CFD data set. Consequently, the number of nodes accessed for this window is quite low. This small value can result in a higher percentage error (even though the deviation from the value may not be much in absolute terms). Similarly, a window located on a dense area may cause more aberrations when the window is aligned to the nearby grid cell boundaries, resulting in a larger number of data points being included/excluded than actual. A small query window can also give low selectivities and node accesses, and even a minor deviation from the actual number can mean a large percentage error. These factors, together with the impact of data set size on estimation accuracy, have been studied in [16].

**Estimation Time** ($nt$ **and** $st$). For some of the data sets, the estimation times for the DH and NDH schemes are quite expensive relative to the actual time taken for serving the query. All the benefits of estimating R-tree performance will be lost, if the estimation time is as high as 20 to 50 percent of the query time itself. The higher the order (for better accuracy), the higher is this time since the query window needs to be broken into several finer Hilbert ranges, and these ranges need to be looked up in the density file. The

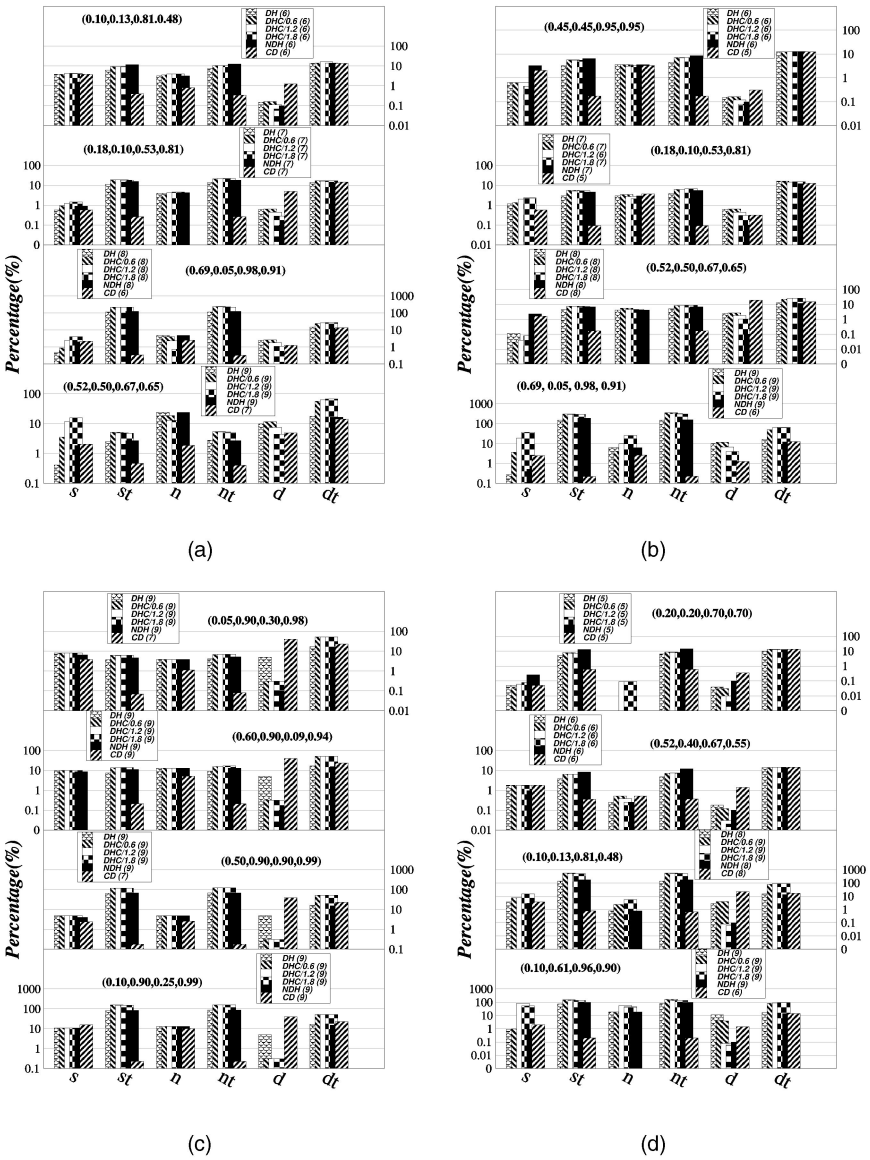


Fig. 6. Point Data Sets. (a) SUP, (b) SCP, (c) TOP, and (d) CFP.

estimation times for the DHC scheme are even worse since it needs to look up index information to get to the appropriate densities. However, as the selectivity becomes larger, the estimation time as a percentage of the query time gets smaller, and these schemes may not be as bad in such cases. The CD scheme is clearly a winner for this criteria because it takes a constant amount of time regardless of the data set and order/level. As a result, the CD estimation time never exceeds even 1 percent of the query execution time for any of the workloads.

**Density File Size and Creation Time** ($d$ **and** $dt$). The theoretical observations in the previous section about the density file size are well borne out by the experimental results. The size quadruples in the DH scheme as we move to the next order/level. However, for the data sets considered, the size goes only as high as 10 percent of the space occupied by the actual R-tree (and that too in only a few cases). Given that space is not really a severe problem (as important as time), the DH scheme may not be a bad choice. Moving to the DHC scheme, we observe that the savings due to compression is not very significant. At low thresholds, there are not many nearby grid cells to merge, and the index that is necessitated for this file can offset any gains due to compression. There is some saving in DHC for certain clustered data sets (like CFD), where there are regions in space with little or no points that can be merged. The CD scheme requires nearly eight times the size of the DH scheme. This is because there are four variables stored for each grid cell (DH requires only one), and we need to maintain the information for not only the data points, but also for the node MBRs. In terms of $d$, the NDH scheme is the winner. The space that it requires is directly proportional to the number of R-tree nodes, and this is much smaller than the actual space in bytes taken by the R-tree (much less than 1 percent). Furthermore, this size is independent of the order/level of gridding.

The DH, DHC, and NDH schemes require that the Hilbert values be assigned to all the data points and then externally sorted before they are histogrammed. DHC requires additional time for compression, which seems to be significantly higher from the results. The CD scheme

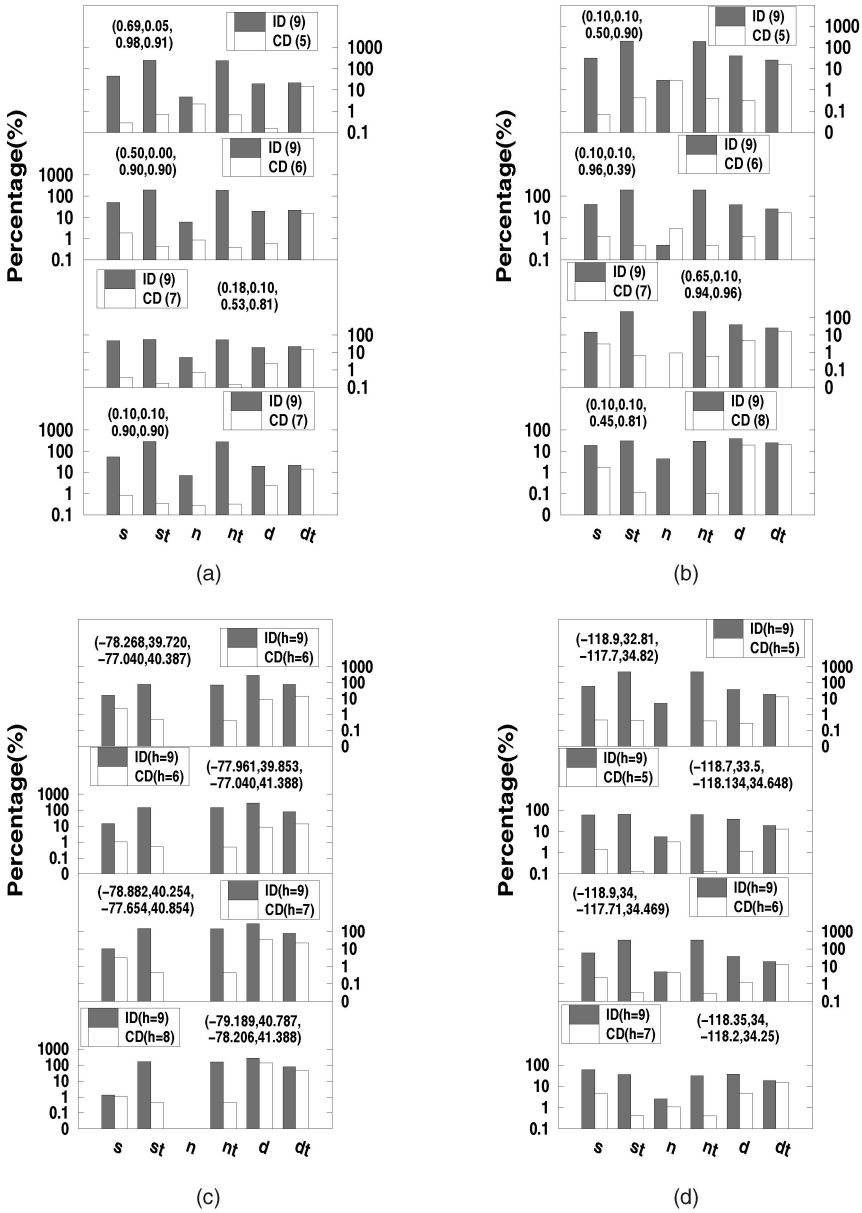


Fig. 7. Rectangle data sets. (a) SUR (universe = [0,0,1,1]). (b) SCR (universe = [0,0,1,1]), (c) PAR (universe = [-79.901,39.7198, -76.8602, 41.6181]), and (d) CAR (universe = [-118.9440, 32.8062, -117.6480, 34.8232]).

does not require Hilbert values to be generated or sorted, but it requires additional time to calculate the cumulative information. These two factors more or less compensate each other, and the density file creation for CD takes roughly the same time as DH/NDH.

### 4.4 Rectangular Data Set Results

As in the point data set results, Fig. 7 shows the results of the ID and CD schemes on the four rectangular data sets with different query windows.

The ID scheme has significant errors for selectivity and node estimation even at level 9. In fact, we need to go higher than level 12 to get errors within 5 percent. This is because of approximation errors that were explained earlier using Fig. 3. The estimation times are very high as well, and in many cases even exceed the actual query execution times on the R-tree. These results suggest that ID is not a feasible approach, and we do not discuss it further.

On the other hand, the CD scheme appears to give remarkably accurate results for both selectivity and node access estimation. We need to go only up to levels 8 or 9 to get the errors within 5 percent. It achieves this goal with a constant estimation time that is usually much less than 1 percent of the actual query execution time. The reader should note that the storage taken by the density file for CD (for DH and ID as well) is independent of the data set size. It is purely a function of the level/order. So, the storage (in bytes) taken by the density file at level 8 for CD in the PAR data set is the same as the storage taken at level 8 in the SUR data set. The reason why the percentages are quite high for PAR is because the data set is itself quite small (the corresponding R-tree is smaller) compared to CAR. The same argument holds for the density file creation time. In summary, as the data set size grows larger, the space overhead of the CD scheme gets smaller (as a percentage).

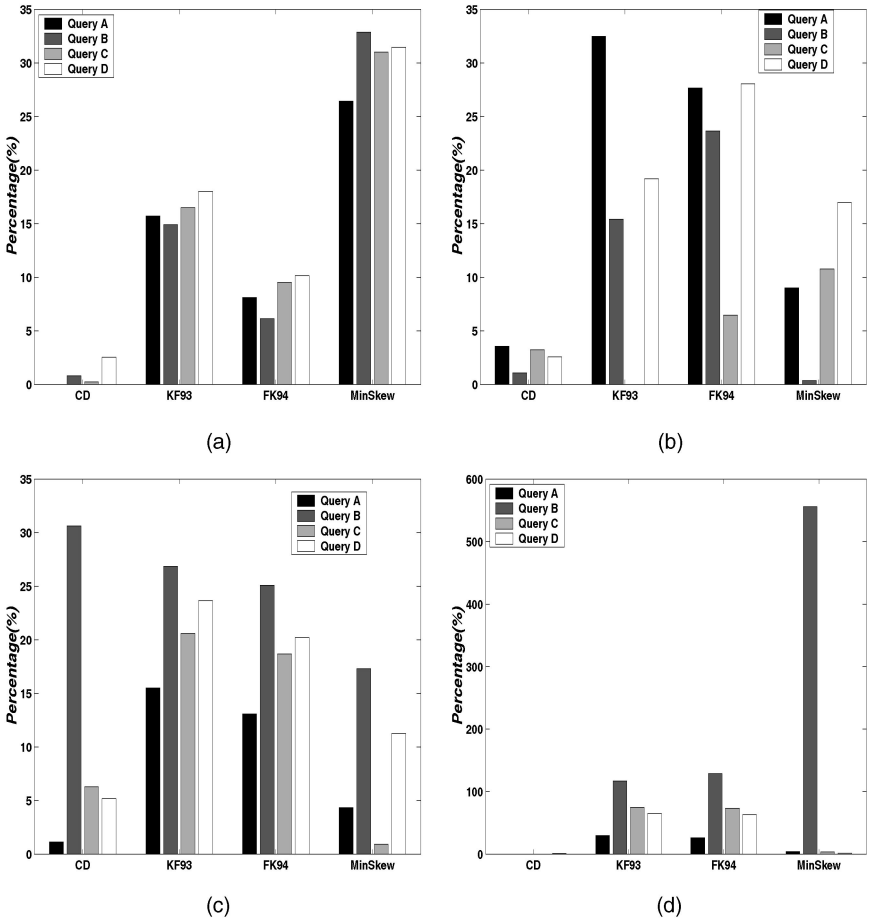


Fig. 8. Comparing accuracy of node access estimation for point data sets with previous research (a) SUP (universe = [0,0,1,1]): QueryA = [0.18, 0.10, 0.53, 0.81], QueryB = [0.10, 0.13, 0.81, 0.48], QueryC = [0.45, 0.45, 0.95, 0.95], and QueryD = [0.69, 0.05, 0.98, 0.91]. b) SCP (universe = [0,0,1,1]): QueryA = [0.18, 0.10, 0.53, 0.81], QueryB = [0.10, 0.13, 0.81, 0.48], QueryC = [0.45, 0.45, 0.95, 0.95], and QueryD = [0.69, 0.05, 0.98, 0.91]. (c) TOP (universe = [0,0,9,1,1]): QueryA = 0.05, 0.90, 0.30, 0.98], QueryB = [0.10, 0.96, 0.90, 0.99], QueryC = [0.20, 0.92, 0.60, 0.98], and (d) CFP (universe = [0,0,1,1]): QueryA = 0.05, 0.90, 0.30, 0.98], QueryB = [0.10, 0.96, 0.90, 0.99], QueryC = [0.20, 0.20, 0.70, 0.70], and QueryD = [0.52, 0.10, 0.81, 0.96].

### 4.5 Comparison with Previous Research

In summary, the point data set results clearly show that CD (and DH to a certain extent) does the best, giving fairly accurate results (less than 5 percent error for CD) in a short time (in CD it takes much less than 1 percent of the query execution time for estimation). CD is the clear winner in the case of the rectangle data sets. For the data sets that were considered, we need to go only up to levels 8 or 9 for these schemes, and the density storage overheads are not overly demanding at these levels. As the data sets get larger (which is when analysis is really meant to be useful), the storage overhead as a percentage of the data set size becomes smaller.

We have also compared node access estimation accuracy ($n$) of CD with three previously proposed analysis techniques for point data sets and rectangular data sets in Figs. 8 and 9. The reader is referred to [14] for actual values and to [16] for details on the formulae/algorithms used in estimation for the previously proposed techniques. We have presented the comparisons over several query windows (the average case results, which have been used in previous studies, are presented in the next section) to bring out the quirks of the different techniques.

The KF93 scheme in [17] gives reasonable accuracy for uniform point data sets. Since it does not consider the location of the query window, it gives higher errors (some even larger than 100 percent) for some windows on clustered data sets. The FK94 scheme in [8] gives a little better accuracy for uniform data sets compared to [17]. But, it is equally worse for the clustered data sets. If the query windows on such data sets were to be focussed on the clustered area (as an application may demand), these schemes would give gross approximations. The MinSkew scheme [2] is not very accurate for the SUP data set, but is better than [17], [8] for the other point data sets.

For the rectangle data sets, the KF93 [17] and PF99 [23] schemes give similar accuracies. The TS96 scheme in [34] gives a little better accuracy, coming closer to the actual R-tree results. This is because in our implementation of this scheme, we actually go through the data set to find the density of the region covered by the query window (though this is not really practical) as explained in [16] instead of assuming uniform densities. MinSkew does better than KF93 for the clustered rectangular data sets (SCR and CAR) as expected.

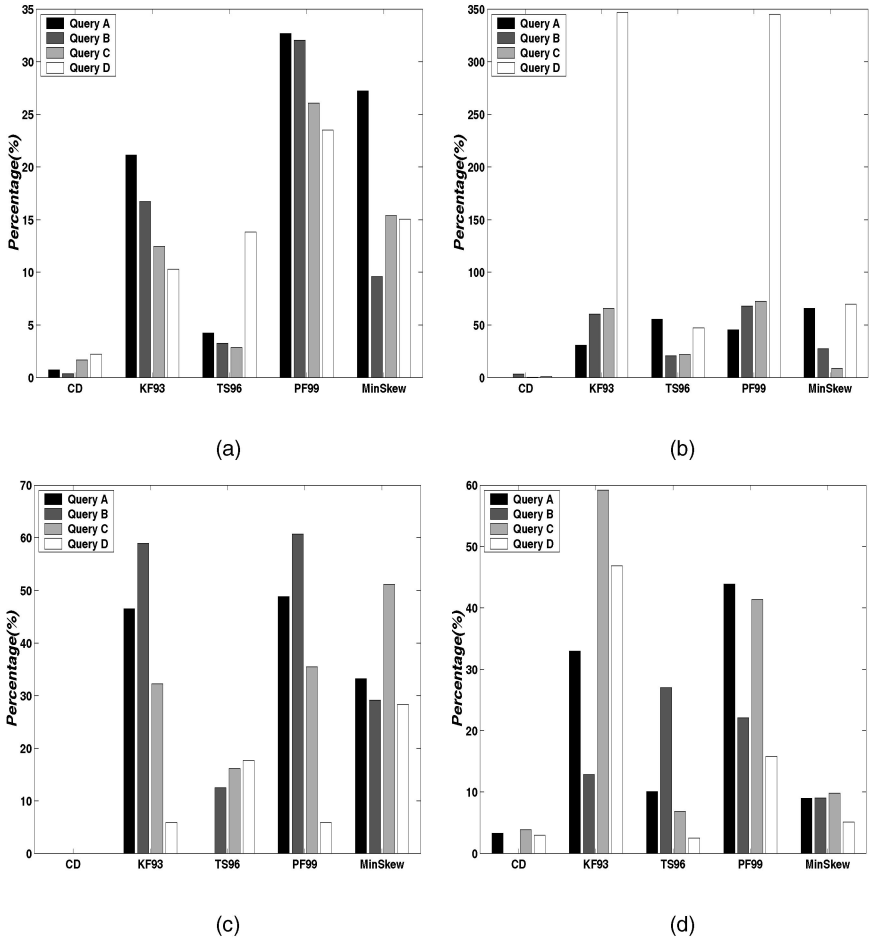


Fig. 9. Comparing accuracy of node access estimation for rectangle data sets with previous research. (a) SUR (universe = [0,0,1,1]): QueryA = [0.18, 0.10, 0.53, 0.81], QueryB = [0.10, 0.13, 0.81, 0.48], QueryC = [0.45, 0.45, 0.95, 0.95], and QueryD = [0.69, 0.05, 0.98, 0.91]. (b) SCR (universe = [0,0,1,1]): QueryA = [0.10, 0.10, 0.45, 0.81], QueryB = [0.20, 0.50, 0.91, 0.85], QueryC = [0.40, 0.12, 0.90, 0.62], and QueryD = [0.65, 0.10, 0.94, 0.96]. (c) PAR (universe = [79.901, 39.7198, -76.8602, 41.6181]): QueryA = [-79.901, 39.7198, -79.189, 40.854], QueryB = [-78.268, 39.7198, -77.040, 40.387], QueryC = [-78.882, 40.254, -77.654, 40.854], and QueryD = [-79.189, 40.788, -78.206, 41.388]. (d) CAR (universe = [-118.9440, 32.8062, -117.6480, 34.8232]): QueryA = [-118.7, 33.5, -118.134, 34.648], QueryB = [-118.8, 33.4, -117.652, 33.966], QueryC = [-118.7, -117.891, 34.809], and QueryD = [-118.2, 33.5, -117.731, 34.82].

The CD scheme gives the best accuracies in most cases. Our schemes take the query window size, location, and data set distribution into consideration in making the estimations. They neither assume any particular distribution (uniform, clustered, clustered following a certain rule, etc.) for the data set, nor do they make any assumptions about the size and location of query window.

### 4.6 Average Case Accuracy

One of the points we are trying to make is that it is important to examine individual estimation errors with a spectrum of different query windows as we have done in the previous experiments. Averaging the errors over a large number of windows as many previous studies have done [2], [17], [34], [23] could hide some of the gross inaccuracies for certain specific query windows (and these windows could be important for a particular application). To address any concerns that the reader may have with this approach (and any concerns that the reader may have about the importance/representativeness of the query windows considered until now), we have also run numerous query windows (1,000) over the data sets, and present the average (mean) estimation error for the CD (level 9), KF93 and MinSkew schemes in Table 4, together with the minimum, maximum, and standard deviation of these errors. The reader should note that the maximum error captures the results for just one of the query windows, and this is usually for a window which has very low selectivity. A very low selectivity can result in large percentage errors even if the absolute number is not significantly different from the actual value (for instance, an estimate of 2 for a selectivity of 1 will give 100 percent error). When many query windows have low selectivities/node accesses in the workload, the corresponding mean errors are high as well. If $E_i$ is the estimation for the actual value $X_i$, then the mean error percentage ($\bar{d}$) is calculated as

$$\bar{d} = \frac{\sum_{i=1}^{N} R_i}{N}$$

and the standard deviation of error percentage ($\sigma_d$) is calculated as

TABLE 4
Estimation Errors in the Average Case for CD, KF93 and MinSkew

| Workload | Scheme | Selectivity | | | | Node Access | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Min(%) | Max(%) | $d$ | $\sigma_d$(%) | Min(%) | Max(%) | $d$ | $\sigma_d$ |
| TOP - Uni | CD | 0 | 72.73 | **2.34** | 8.74 | 0 | 44.85 | **1.70** | 6.63 |
| | KF93 | 0 | 86.10 | **11.28** | 11.87 | 0.04 | 135.68 | **12.79** | 12.19 |
| | MinSkew | 0 | 73.40 | **1.25** | 4.96 | 0 | 100 | **25.65** | 16.56 |
| TOP - Skew | CD | 0 | 158.70 | **5.52** | 17.87 | 0 | 65.58 | **3.61** | 10.69 |
| | KF93 | 0 | 118.87 | **19.86** | 22.28 | 0.1 | 84.82 | **15.49** | 11.68 |
| | MinSkew | 0 | 219.21 | **1.70** | 8.25 | 0 | 84.93 | **19.36** | 15.11 |
| CFD - Uni | CD | 0 | 54.55 | **1.93** | 3.16 | 0 | 31.28 | **1.09** | 3.68 |
| | KF93 | 0.15 | 13467.63 | **3230** | 2596.01 | 1.08 | 2754.6 | **697.92** | 594.86 |
| | MinSkew | 0.34 | 70400 | **2918.67** | 5394.05 | 0 | 3030.76 | **399.75** | 613.20 |
| CFD - Skew | CD | 0 | 14.58 | **0.12** | 0.69 | 0 | 13.58 | **0.13** | 0.77 |
| | KF93 | 57.99 | 7553.67 | **138.47** | 474.69 | 30.39 | 1601.29 | **95.12** | 96.61 |
| | MinSkew | 8.97 | 20702.08 | **143.93** | 1006.22 | 0 | 1888.24 | **52.05** | 118.47 |
| PAR - Uni | CD | 0 | 53.85 | **1.23** | 2.56 | 0 | 35.71 | **1.05** | 2.53 |
| | KF93 | 0.47 | 32160.24 | **420.51** | 1789.09 | 0.06 | 6153.75 | **251.62** | 441 |
| | MinSkew | 0 | 2784.62 | **36.19** | 159.04 | 0 | 350 | **37.90** | 22.09 |
| PAR - Skew | CD | 0 | 25.10 | **0.64** | 1.14 | 0 | 20.00 | **0.95** | 1.71 |
| | KF93 | 0.07 | 272.60 | **36.70** | 22.82 | 0.25 | 233.66 | **33.22** | 19.37 |
| | MinSkew | 0 | 118 | **9.25** | 10.45 | 0 | 100 | **46.16** | 18.33 |
| CAR - Uni | CD | 0 | 50.00 | **1.50** | 3.24 | 0 | 100.00 | **1.24** | 4.08 |
| | KF93 | 0.1 | 2616680 | **9839.30** | 117907 | 0.09 | 27059 | **883.42** | 2396.45 |
| | MinSkew | 0 | 85685.71 | **271.21** | 2939.75 | 0 | 1000 | **109.52** | 126.12 |
| CAR - Skew | CD | 0 | 8.28 | **0.71** | 0.85 | 0 | 21.05 | **0.92** | 1.59 |
| | KF93 | 0 | 305 | **69.51** | 22.11 | 0.1 | 225.08 | **66.75** | 18.75 |
| | MinSkew | 0.07 | 367.60 | **37.26** | 26.24 | 0 | 378.13 | **37.51** | 46.86 |

$$\sigma_d = \sqrt{\frac{\sum_{i=1}^{N}(R_i - \bar{d})^2}{N}},$$

where $R_i = \frac{|E_i - X_i|}{X_i} \times 100$. Those with a $X_i = 0$ are excluded from these calculations.

For the query windows, we use two workloads (*uni* and *skew*). In both these workloads, the aspect ratio of the query window is uniformly varied between 0.33 to 3.0, and the size is varied between 0.1 percent to 25 percent of the spatial extent. In *uni*, the center point of the query window is uniformly distributed within the spatial extent.

Our *skew* workload is based on the hypothesis that the region with more data will draw more queries. We first obtain a spatial Cumulative Distribution Function (CDF) of the points (in point data sets) or center-points (in rectangle data sets) of the target data set. The workload picks the center points of the query windows from the probability density function determined by this CDF.

Many previous studies have used workloads similar to *uni* to conduct average case experiments, and we feel that *skew* may be a better approximation to actual workloads (we can only hypothesize at this point, and we plan to investigate workload characterization in our future work). Regardless of the workload used for average case estimation, we still find that CD gives very good estimates (typically less than 5 percent error) over all the data sets as in the previous individual query estimations. Its results are significantly more accurate than the corresponding estimations by KF93 and MinSkew in most cases.

### 4.7 What Level/Order Should We Use?

There is an issue with regard to the proposed schemes that has not yet been investigated, which is important to all histogram/density based techniques. This is regarding how we can decide on the level/order for the density file (the query window parameters are not known at the time the density file is created). A larger level/order usually improves the accuracy of estimation, albeit at a higher storage cost. Previous histogram-based studies [2] have used an experimental approach with average case studies of different data sets to find a good operating point. We use a similar approach, and the average errors used with numerous (1,000) query windows randomly located at different regions of the spatial extent for each of three different sizes (0.1 percent, 1 percent, and 10 percent of the spatial extent) is given in Fig. 10 for the CD scheme on four real data sets. As expected, the average error decreases with increasing levels, and the percentage error is lower with larger query windows. Though the inferences that can be drawn are data set dependent, we find that over all the data sets that we have considered (including the ones presented in [16], the errors become reasonably small (typically lower than 5-10 percent) beyond levels 9 or 10. Knowledge of the data set and/or possible query windows for the given data set can help decide on a level in a better fashion than using a workload with average case behavior. It should be again noted that the estimation time for the CD scheme is relatively independent of the level of gridding (unlike other schemes [2]), allowing us to use a reasonably large gridding level as long as storage for the density file does not become a big issue.

## 5 Concluding Remarks and Future Work

This paper has presented a novel set of schemes to analyze range query performance on spatial data. Three of these schemes can analyze point data sets, and the other two can be used for both point and rectangular data sets. These schemes make very little assumptions about the data set, and use an auxiliary data structure (histograms), called a density file, which can be constructed when the index structure is created (one-time cost). When a query is given,

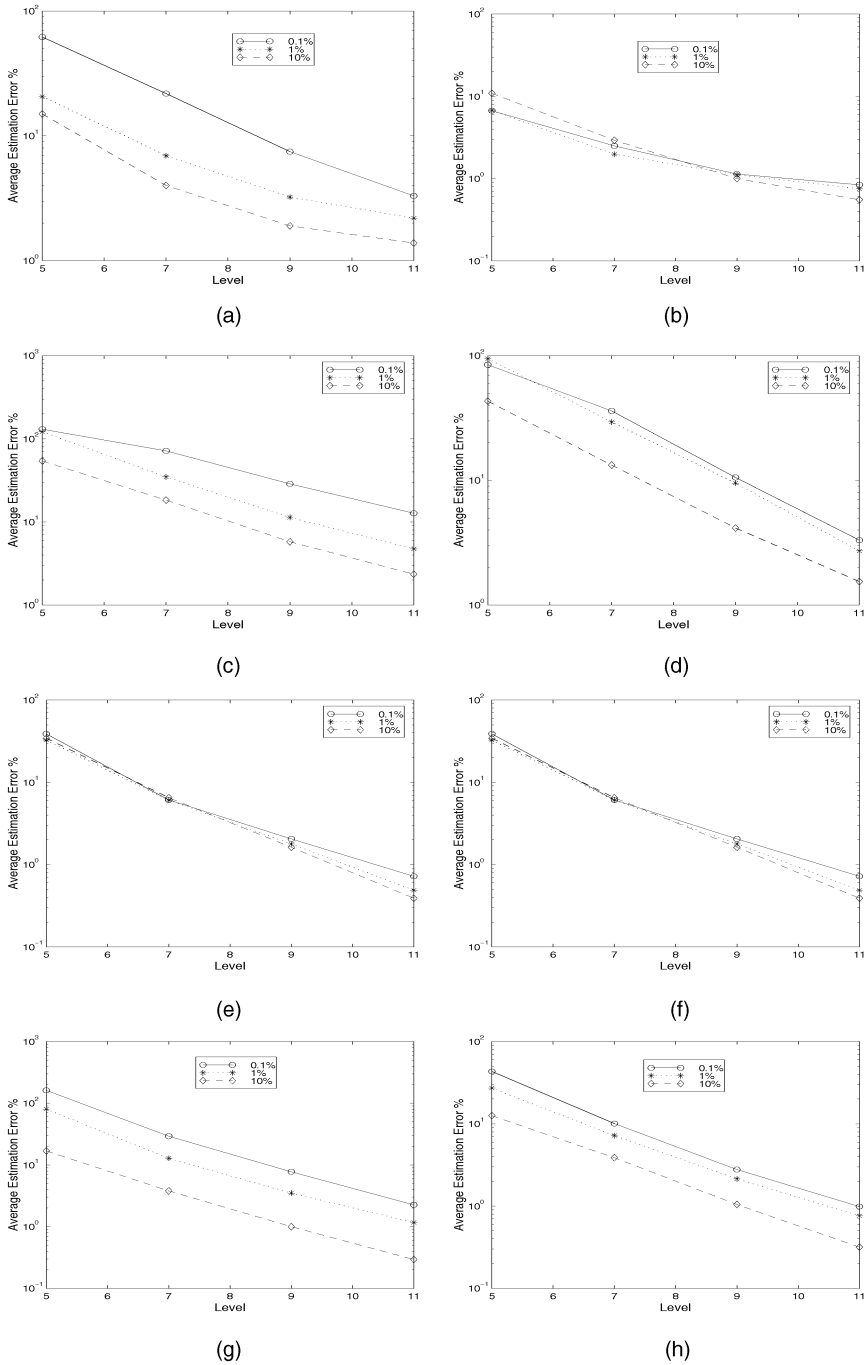


Fig. 10. Impact of gridding level on accuracy of estimation using CD. The average error is shown for three query window sizes (0.1, 1, and 10 percent of spatial extent). (a) CFD Selectivity, (b) CFD Node Access, (c) TOP Selectivity, (d) TOP Node Access, (e) CAR Selectivity, (f) Car Node Access, (g) PAR Selectivity, and (h) PAR Node Access.

the density file is "quickly" looked up to get sufficient information about the data set which is then used to calculate the selectivity. We have also illustrated how this information can be used to estimate the number of nodes that would be accessed in the index structure using the packed R-tree as a case-study.

With a diverse suite of real and synthetic data sets, which fall under both uniform and skewed classifications, this paper has shown that one of the schemes, called *Cumulative Density* (CD), gives very accurate results with errors that are much lower (usually less than 5 percent errors) than many previously proposed analysis techniques. This accuracy is obtained over a spectrum of query window sizes, locations, and aspect ratios, and not just in the average case. This makes the CD scheme more universally applicable. The estimation with the CD scheme takes a constant amount of time (at most four disk accesses and three arithmetic operations), regardless of data set or query window parameters. This time tends to be typically lower than 1 percent of the time that it would take to actually execute the query using an R-tree. As a result, the CD scheme is very practical and would be extremely useful in a query optimizer.

The CD scheme can be used to estimate the selectivity of both point and rectangular data sets. It can be used for other spatial objects, provided their bounding boxes are available (in which case, it can be used to measure the cost of a filtering step). In addition, the CD scheme can be used to estimate the number of nodes accessed by a range query in any index structure (not just an R-tree), provided the bounding boxes of the nodes of the structure are made available. We can use these bounding boxes as input to the selectivity procedure of CD to find out the number of nodes that would be accessed in the index structure.

This paper has opened interesting directions for future research. While the techniques presented here could be used for multidimensional data sets, there is a concern of the space required to maintain the histograms. However, the CD technique is still promising since its estimation costs remain constant. We plan to characterize the workload (data sets, query window parameters, etc.) of different spatial databases for a better choice of the gridding level, which could also help us address the storage issue. We are trying to find out if the proposed schemes can be extended (or new ones can be developed along the lines of what has been presented here) to more complicated queries, such as spatial joins. Finally, we would like to incorporate these schemes into a spatial database query optimizer.

## ACKNOWLEDGMENTS

The authors would like to thank Swarup Acharya of Bell Laboratories for his efforts to educate us on the details of the MinSkew algorithm/implementation. This is a substantially extended version of a preliminary paper [15] that appeared in the Proceedings of the IEEE International Conference on Data Engineering, March 2000. The extensions include a more detailed performance study as well as experimental comparisons with several previous estimation techniques. This research has been supported in part by the US National Science Foundation Career Award MIPS-9701475, US NSF grant CCR-9900701, EPA grant R825195-01-0, and US NSF equipment grants CDA-9617315 and EIA-9818327.

## REFERENCES

[1] N. An, R. Lu, L. Qian, A. Sivasubramaniam, and T. Keefe, "Storing Spatial Data on a Network of Workstations," *Cluster Computing, The J. Networks, Software Tools & Applications: Special Issue on I/O in Shared-Storage Clusters,* vol. 2, no. 4, pp. 259-270, 1999.

[2] S. Acharya, V. Poosala, and S. Ramaswamy, "Selectivity Estimation in Spatial Databases," *Proc. 1999 ACM SIGMOD Int'l Conf. Management of Data,* pp. 13-24, May 1999

[3] W.G. Aref and H. Samet, "Optimization for Spatial Query Processing," *Proc. 17th Int'l Conf. Very Large Data Bases,* pp. 81-90, Sept. 1991.

[4] W.G. Aref and H. Samet, "A Cost Model for Query Optimization Using R-Trees," *Proc. ACM GIS Workshop,* pp. 60-67, 1994.

[5] A. Belussi and C. Faloutsos, "Estimating the Selectivity of Spatial Queries Using the 'Correlation' Fractal Dimension," *Proc. 21th Int'l Conf. Very Large Data Bases,* pp. 299-310, Sept. 1995.

[6] N. Beckmann, H.P. Kriegel, R. Schneider, and B. Seeger, "The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles," *Proc. 1990 ACM-SIGMOD Conf.,* June 1990.

[7] R. Finkel and J. Bentley, "Quad Trees: A Data Structure for Retrieval on Composite Keys," *Acta Informatica,* vol. 4, pp. 1-9, 1974.

[8] C. Faloutsos and I. Kamel, "Beyond Uniformity and Independence: Analysis of R-Trees Using the Concept of Fractal Dimension," *Proc. 13th ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems,* pp. 4-13, June 1994.

[9] C. Faloutsos, T.K. Sellis, and N. Roussopoulosand, "Analysis of Object Oriented Spatial Access Methods," *Proc. 1987 ACM-SIGMOD Conf.,* pp. 426-439, Dec. 1987.

[10] J.G. Griffiths, "An Algorithm for Displaying a Class of Space-Filling Curves," *Software—Practice and Experience (SPE),* vol. 16, pp. 403-411, 1986.

[11] A. Gutman, "R-Trees: A Dynamic Index Structure for Spatial Searching," *Proc. 1984 ACM-SIGMOD Conf.,* June 1984.

[12] A. Henrich, H.W. Six, and P. Widmayer, "The LSD Tree: Spatial Access to Multidimensional Point and Non-Point Data," *Proc. 15th Int'l Conf. Very Large Databases (VLDB),* pp. 45-53, Aug. 1989.

[13] H.V. Jagadish, "Linear Clustering of Objects with Multiple Atributes," *Proc. 1990 ACM SIGMOD Int'l Conf. Management of Data,* pp. 332-342, May 1990.

[14] J. Jin, N. An, and A. Sivasubramaniam, "Analyzing Range Queries on Spatial Data," Technical Report CSE-99-005, Dept. of Computer Science and Engineering, The Pennsylvania State Univ., June 1999.

[15] J. Jin, N. An, and A. Sivasubramaniam, "Analyzing Range Queries on Spatial Data," *Proc. IEEE Int'l Conf. Data Eng.,* pp. 525-534, Mar. 2000.

[16] J. Jin, "Techniques for Analyzing Range Queries on R-Trees," master's thesis, Dept. of Computer Science & Eng., Penn State Univ., May 1999.

[17] I. Kamel and C. Faloutsos, "On Packing R-Trees," *Proc. 2nd ACM Int'l Conf. Information and Knowledge Management,* pp. 490-499, Nov. 1993.

[18] S.T. Leutenegger and M.A. Lopez, "The Effect of Buffering on the Performance of R-Trees," *Proc. 14th Int'l Conf. Data Eng.,* pp. 164-171, Feb. 1998.

[19] R.W. Marx, "The TIGER System: Automating the Geographic Structure of the United States Census," *Government Publications Rev.,* vol. 13, pp. 181-201, 1986.

[20] D.J. Mavriplis, "An Advancing Front Delaunay Triangulation Algorithm Designed for Robustness," *J. Computational Physics,* pp. 90-101, 1995.

[21] M. Muralikrishna and D.J. DeWitt, "Equi-Depth Histograms For Estimating Selectivity Factors For Multi-Dimensional Queries," *Proc. 1988 ACM SIGMOD Int'l Conf. Management of Data,* pp. 28-36, June 1988.

[22] J. Nievergelt, H. Hinterberger, and K.C. Sevcik, "The Grid File: An Adaptable, Symmetric Multikey File Structure," *Proc. Int'l Workshop Graph Theoretic Concepts in Computer Science,* vol. 9, no. 1, pp. 38-71, Mar. 1984.

[23] G. Proietti and C. Faloutsos, "I/O Complexity for Range Queries on Region Data Stored Using an R-Tree," *Proc. 15th Int'l Conf. Data Eng.,* pp. 628-635, Mar. 1999.

[24] V. Poosala, "Histogram-Based Estimation Techniques in Databases," PhD thesis, Univ. of Wisconsin-Madison, 1997.

[25] Int'l Research Inst. For Climate Prediction, IRI/LDEO Climate Data Library, http://ingrid.ldgo.columbia.edu/.

[26] B. Pagel, H.-W. Six, H. Toben, and P. Widmayer, "Towards an Analysis of Range Query Performance in Spatial Data Structures," *Proc. 12th ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems,* pp. 214-221, May 1993.

[27] N. Roussopoulos and D. Leifker, "Direct Spatial Search on Pictorial Databases Using Packed R-Trees," *Proc. 1985 ACM SIGMOD Int'l Conf. Management of Data,* pp. 17-31, May 1985.

[28] J.T. Robinson, "The K-D-B-Tree: A Search Structure for Large Multidimensional Dynamic Indexes," *Proc. 1981 ACM-SIGMOD Conf.,* pp. 10-18, Apr. 1981.

[29] H. Samet, "Spatial Data Structures," *Modern Database Systems,* pp. 361-385, 1995.

[30] G.P. Shapiro and C. Connel, "Accurate Estimation of the Number of Tuples Satisfing a Condition," *Proc. 1984 ACM-SIGMOD Conf.,* pp. 256-276, June 1984.

[31] S. Shekhar, S. Chawla, S. Ravada et al. "Spatial Databases—Accomplishments and Research Needs," *IEEE Trans. Knowledge and Data Eng.,* vol. 11, no. 1, pp. 45-55, 1999.

[32] T.K. Sellis, N. Roussopoulos, and C. Faloutsos, "The R+-Tree: A Dynamic Index for Multi-Dimensional Objects," *Proc. 13th Int'l Conf. Very Large Data Bases,* pp. 507-518, Sept. 1987.

[33] Y. Theodoridi and D. Papadias, "Range Queries Involving Spatial Relations: A Performance Analysis," *Proc. COSIT '95*, pp. 537-551, Sept. 1995.
[34] Y. Theodoridi and T.K. Sellis, "A Model for the Prediction of R-Tree Performance," *Proc. 15th ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems*, pp. 161-171, June 1996.
[35] Y. Theodoridis, E. Stefanakis, and T. Sellis, "An Efficient Cost Model for Spatial Queries Using R-Trees," Technical Report KDBSLAB-TR-97-01, Dept. Electrical and Computer Eng., Computer Science Division, Nat'l Technical Univ. of Athens, Feb. 1997.

**Ning An** received the BS and MS degree in computer science from Lanzhou University, P.R. China, in 1993 and 1996, respectively. He is currently a PhD candidate in the Department of Computer Science and Engineering at The Pennsylvania State University. His main research interests are in spatial databases and location-aware computing.

**Ji Jin** received the BE and ME degree in computer engineering from Shanghai JiaoTong University, Shanghai, China, in 1992 and 1994, respectively. Then, he received the MS degree in computer Science from The Pennsylvania State University in August 1999. His area of interest are in geographical information systems and database applications.

**Anand Sivasubramaniam** received the BTech degree in computer science from the Indian Institute of Technology, Madras, in 1989, and the MS and PhD degrees in computer science from the Georgia Institute of Technology in 1991 and 1995, respectively. He has been on the faculty at The Pennsylvania State University since the fall 1995, where he is currently an associate professor. Dr. Sivasubramaniam's research interests are in computer architecture, operating systems, performance evaluation, and spatial databases. His research has been funded by US National Science Foundation through several grants (including the CAREER award), and support from industries including IBM, Unisys, and Microsoft. He has published extensively in several leading journals and conferences, and is currently serving as an associate editor of the *IEEE Transactions on Computers*. He is a member of the IEEE, IEEE Computer Society, and ACM.

▷ **For more information on this or any computing topic, please visit our Digital Library at** http://computer.org/publications/dlib.