Analyzing Range Queries on Spatial Data *

Ji Jin
jjin@cse.psu.edu

Ning An
an@cse.psu.edu

Anand Sivasubramaniam
anand@cse.psu.edu

Dept. of Computer Science & Engineering
The Pennsylvania State University
University Park, PA 16802

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 datasets 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%** *error) using a diverse suite of point and rectangular datasets, 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%** *of the time that it would take to execute the query, regardless of dataset or query window parameters.*

1. Introduction

Spatial databases are gaining importance and are becoming prevalent in numerous applications. Geographical Information Systems (GIS), navigation/positioning, image processing, demography, epidemiology, terrain analysis, mining, military planning and logistics, computer-aided design and robotics, are just 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 becomes even more essential with the choice of numerous index structures [BKSS90, FB74, Gut84, HN83, HSW89, Rob81, SRF87, Sam95]. Performance analysis can help better understand the suitability of a data structure for different input datasets (both size and spatial distribution). Given a dataset, 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 [TP95, FK94, PF99, TS96, KF93, TSS97, PSTW93] 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. We can also use these techniques 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 [KF93] as a case-study.

The underlying philosophy of the analysis techniques presented in this paper is that they make little or no assumption about the dataset. Any information that is dataset dependent should be drawn from the dataset itself. As a result, these techniques are universally applicable, regardless of the dataset 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 dataset that is necessary to conduct the estimation. The density file creation is a one-time cost. When the query is

* This research has been supported in part by NSF Career Award MIPS-9701475, EPA grant R825195-01-0, and NSF equipment grants CDA-9617315 and EIA-9818327.

given, the density file is "quickly" looked up to determine selectivity and nodes accessed.

Using 2-dimensional space as an example (extension to multi-dimensional space is straightforward), this paper proposes five analysis schemes for point (sizeless and shapeless objects) datasets, and two for rectangular datasets. 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. With a diverse (both real and synthetic, that are uniform or skewed) suite of datasets 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%. 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% of the actual query execution time. The storage overheads of 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 datasets. Section 4 gives results from the analyses using a spectrum of datasets and query windows. Finally, Section 5 summarizes the contributions of this paper.

2. Related work

Estimation of range query performance on spatial data has been shown to be extremely important [AS91, MD88]. Consequently, there is a large body of literature [FSR87, TP95, FK94, BF95, PF99, TS96, TSS97, PSTW93, KF93] on this topic. These techniques, however, are limited either to the kind of datasets that they can analyze (points, rectangles, etc.), and/or make simplifying assumptions about the dataset (uniform, skewed following a certain rule, etc.) or query windows. The reader is referred to [Jin99] for details on these techniques, and a comparison of the accuracy of these techniques with those presented here is given in [JAS99].

Most of these techniques fall under what has been characterized as parametric techniques [APR99], which try to mathematically model the data based on certain assumptions. Spatial datasets are likely to be very diverse. Consequently, not all of the above techniques can be used to analyze the performance of all datasets. We believe that an estimation technique should make little or no assumptions about the input dataset. Any information that it would need should come from the dataset 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 dataset 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 [APR99], undertaken concurrently with this work, has examined different ways of constructing histograms for spatial databases to estimate selectivity. 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 [APR99], 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 [APR99]. 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 non-overlapping buckets unlike the ones used in [APR99]. 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 [APR99], 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 non-overlapping 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 [KF93] as a case-study.

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 dataset, 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 dataset. 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.

3.2. Techniques for point datasets

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 dataset fall within this window (selectivity). The leaf nodes (of the packed R-tree index structure) 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. 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 h [Gri86] (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 (4^h) 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 (potentially non-contiguous) 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. 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 higher the order, higher would be the level of accuracy most of the time. 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. Specifically, this scheme attempts to compress/merge neighboring grid cells that have similar (including zero) densities.

We initially start with the density file of the DH scheme. We compare the density of each set of 4 consecutive grid cells (recall that Hilbert ordering recursively breaks down the space into 4 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 deviation (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 4 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, and so on. 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. Further, 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% 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 [MD88]. 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 [APR99]. 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) of a overlap between the buckets.

3.3. Techniques for rectangle datasets

Rectangle datasets pose a more difficult problem than point datasets since they contain size information in addition to position. The point dataset schemes may not necessarily work for rectangle datasets, 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 [KF93]), but that would need a high amount of computation. Further, 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 non-overlapping buckets. There are a couple of differences from the previous schemes. In the point dataset 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. The spatial extent is, instead, gridded into cells (4^h) 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 2-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 just keep 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 either, 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 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 ($DS(i,j)$); (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 Figure 1 for the rectangular dataset on the left side.

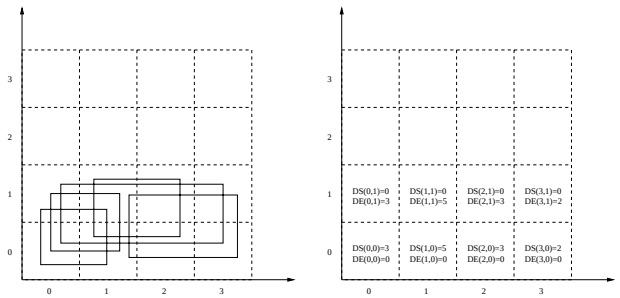

Figure 1. Density Information in ID

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:

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

Figure 2 illustrates this observation with an example. The query window covers $(1, 2, 2, 3)$, for the dataset with

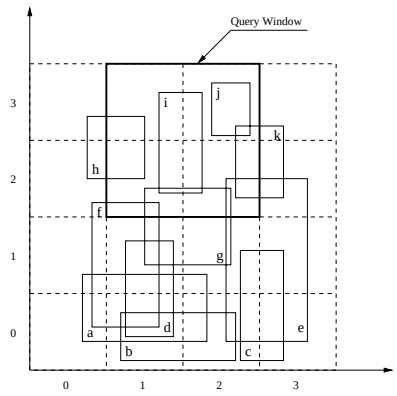

Figure 2. Rectangles in a Query Window

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 can 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.

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

For example, in Figure 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,

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

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

This scheme, however, is not always accurate. It does not differentiate between two neighboring rectangles with edges that fall on adjacent columns of a row, and a single large rectangle covering both columns (see [Jin99]).

Nodes Accessed: We cannot use selectivity information directly to estimate the nodes accessed (as in the point dataset 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 dataset 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 earlier. 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 dataset schemes, and we 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:

$$S(xl,0,xh,yh) = BS'(xh,yh) - BE'(xl-1,yh) \quad (6)$$

$$E(xl,0,xh,yh) = US'(xh,yh) - UE'(xl-1,yh) \quad (7)$$

$$N(qxl,qyl,qxh,qyh) = BS'(qxh,qyh) - BE'(qxl-1,qyh) - [US'(qxh,qyl-1) - UE'(qxl-1,qyl-1)] \quad (8)$$

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 (4 disk accesses and 3 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 of the ID scheme mentioned earlier.

It is also interesting to note that a point dataset can be viewed as a special class of rectangular data (of size 0). Consequently, *the rectangle dataset schemes can be used to estimate selectivity and nodes accessed of point datasets as well*. We have used the CD scheme for estimation of point datasets in the following evaluation studies, and compare it with the point dataset schemes. The reader is referred to [Jin99] for a detailed comparison of the five schemes in terms of both space and time complexity, as well as accuracy.

4. Evaluating the analysis techniques

To evaluate the different schemes described in the previous section, we conduct extensive experiments with several point and rectangle datasets, and several query windows. These studies have been conducted on a 170 MHz SUN UltraEnterprise 1 server. In the following discussion, we briefly discuss the datasets considered, examine the metrics/criteria used for comparing the schemes, and present the results for the point and rectangle datasets.

4.1. Datasets

We have considered a wide spectrum of point and rectangular datasets, 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 datasets (such as the Tiger [Mar86] data), while others have been synthetically generated. Due to space limitations, we are not able to present the results for all of them or show them pictorially here. The reader is referred to [Jin99] for further information. In this paper, we present results for (a) two point datasets: **TOP**: Topological Point dataset taken from [Pre], with $478,786$ points following an interesting pattern (points are arranged in regular rows with significant gaps between successive rows), **CFD**: Computation Fluid Dynamics dataset taken from [Mav95], with $208,688$ points that are clustered; and (b) two rectangular datasets: **PAR**: a dataset containing the MBRs of rivers of Pennsylvania from the TIGER database [Mar86], with $30,218$ rectangles, **CAR**: a dataset containing the MBRs of the streets of California from the TIGER database, with $248,643$ rectangles.

We have considered different query window sizes (1%, 5%, 25%, 50% and 100% of the spatial extent), aspect ratios and locations (that cover both sparse and clustered regions of the extent) for each of these datasets. We believe that the chosen datasets 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 [Gri86, Jag90] for each of the above datasets, 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 an absolute 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.
- *Node Access Estimation (n)*: This measures the accuracy of the scheme in estimating the number of nodes accessed/touched in serving the specific query. It is expressed as an absolute 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 with ever increasing disk storage capacities (as long as density files are not larger or become a large fraction of the actual dataset/R-tree).

4.3. Point dataset results

Figure 3 shows a part of the results for the different schemes. We present representative results from four query windows for each dataset (the window coordinates are given in the Figure). 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% error) of accuracy. In case, none of these levels gives an error lower than 5%, then we give the results for h=9. Consequently, different

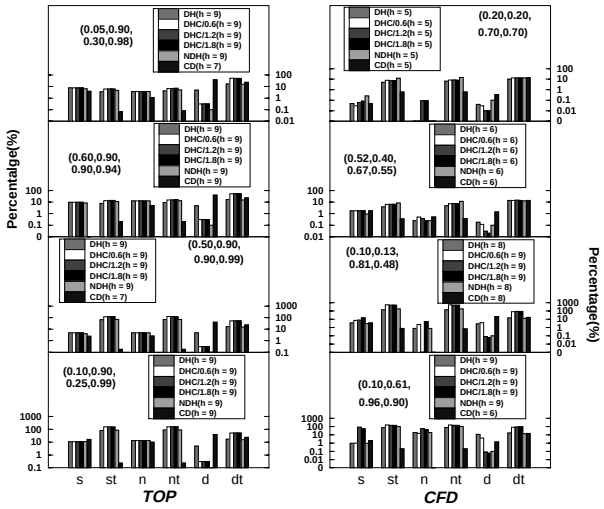

Figure 3. Point Datasets

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), and the corresponding value is shown in the legends.

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 modify the query window to align the query window with 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 area of query window compared to a coarser level of gridding (see [Jin99] 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 [Jin99]). 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 dataset (where several points could lie on grid cell boundaries because of the nature of the dataset) 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% 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). 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. Further, even clustered datasets, have some semblance of uniformity within small/isolated regions 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 Figure 3. Accuracy for thresholds of 1.2 and 1.8, is not acceptable in several cases, and only 0.6 even comes close to the other schemes.

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 datasets and query windows, the DH scheme estimations were less than 10% error at order/level 6, and less than 5% error at order/level 8 [Jin99]. A similar observation holds for CD as well. Hence, level 8 seems to be an appropriate operating point.

Apart from the nature of the dataset (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 dataset. 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 dataset size on estimation accuracy have been studied in [Jin99].

Estimation Time (nt and st): For some of the datasets, 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 $20 - 50\%$ of the query time itself. 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 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 dataset and order/level. As a result, the CD estimation time rarely exceeds even 1% 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 chapter 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 datasets considered, the size goes only as high as 10% 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 datasets (like CFD), where there are regions in space with little or no points that can be merged. The CD scheme requires nearly 8 times the size of the DH scheme for a given level. This is because there are 4 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%). Further, this size is independent of the order/level of gridding.

The DH, DHC and NDH schemes require 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 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 dataset results

As in the point dataset results, Figure 4 shows the results of the ID and CD schemes on PAR and CAR 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%. This is because of approximation errors that were explained earlier. 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%. It achieves this goal with a constant estimation time that is usually much less than 1% 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 dataset size. It is purely a function of the level/order. The reason why d is quite high for PAR is because the dataset is itself quite small (the corresponding R-tree is smaller) compared to CAR. The same argument holds for the density file creation time (dt). In summary, as the dataset size grows larger, the space overhead of the CD scheme gets smaller (as a percentage).

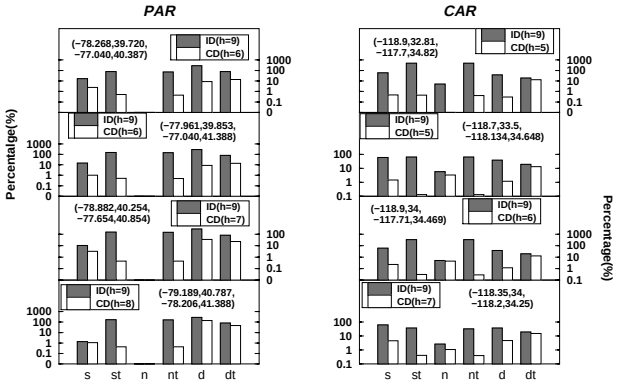

Figure 4. Rectangle Datasets

4.5. 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 [APR99, KF93, TS96, PF99], 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, we have also run numerous query windows (1000) over the datasets, and present the average (mean) estimation error for the CD scheme with level 9 in Table 1 expressed as a percentage of the actual mean (s and n), together with the minimum and maximum 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% error).

Workload	Selectivity(%)			Node Access(%)		
	Min	Max	**Avg**	Min	Max	**Avg**
TOP-uni	0	72.73	**2.34**	0	44.85	**1.70**
CFD-uni	0	54.55	**1.93**	0	31.28	**1.09**
PAR-uni	0	53.85	**1.23**	0	35.71	**1.05**
CAR-uni	0	50.00	**1.50**	0	100.00	**1.24**
TOP-skew	0	158.70	**5.52**	0	65.58	**3.61**
CFD-skew	0	14.58	**0.12**	0	13.58	**0.13**
PAR-skew	0	25.10	**0.64**	0	20.00	**0.95**
CAR-skew	0	8.28	**0.71**	0	21.05	**0.92**

Table 1. Average Case Errors for CD

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% to 25% of the spatial extent. In *uni*, the center point of the query window is uniformly distributed within the spatial extent. In *skew*, the center point of the query window is located based on a distribution of the points (in point datasets) or rectangle center-points (in rectangle datasets) in the actual dataset i.e. a spatial Cumulative Distribution Function (CDF) of the dataset has been obtained and the query window location is drawn 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. Regardless of the workload used for average case estimation, we still find that CD gives very good estimates (typically less than 5% error) over all the datasets as in the previous individual query estimations.

4.6. Summary

In summary, the point dataset results clearly show that CD (and DH to a certain extent) does the best, giving fairly accurate results (less than 5% error for CD) in a short time (in CD it takes much less than 1% of the query execution time for estimation). For the rectangular datasets, CD is the clear winner. For the datasets 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 datasets get larger (which is when analysis is really meant to be useful), the storage overhead as a percentage of the dataset size becomes smaller. Further, one is usually interested in lowering estimation times rather than space overheads.

We have also compared accuracy of the DH and CD schemes with several previously proposed analysis techniques [KF93, TS96, PF99, APR99]. CD gives much better accuracy than these techniques and the reader is referred to [JAS99] for further details on these comparisons.

5. Concluding remarks

This paper has presented a novel set of schemes to analyze range query performance on spatial data. Three of these schemes can analyze point datasets, and the other two can be used for both point and rectangular datasets. These schemes make very little assumptions about the dataset, 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, the density file is "quickly" looked up to get sufficient information about the dataset 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 datasets, which fall under both uniform and skewed classifications, this paper has shown that one of the schemes, called *Cumulative Density* (CD) scheme, gives very accurate results, with errors that are much lower (usually less than 5% errors) than many previously proposed analysis techniques. This accuracy is observed 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 4 disk accesses and 3 arithmetic operations), regardless of dataset or query window parameters. This time tends to be typically lower than 1% 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.

This paper has opened several interesting directions for future research. One issue 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). We have been examining different real datasets and our examination suggests that we do not really need to go to high levels (relative to the dataset size) with the CD scheme to limit estimation errors within reasonable bounds. Further, we are trying to develop techniques that can be used to decide on the level at the time of density file creation using workload information. We are also trying to find out if the proposed schemes can be extended (or new ones can be developed) to more complicated queries, such as spatial joins.

References

[APR99] S. Acharya, V. Poosala, and S. Ramaswamy. Selectivity Estimation in Spatial Databases. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, pages 13–24, May 1999.

[AS91] W.G. Aref and H. Samet. Optimization for Spatial Query Processing. In *Proceedings of the 17th International Conference on Very Large Data Bases*, pages 81–90, Barcelona, Catalonia, 1991.

[BF95] A. Belussi and C. Faloutsos. Estimating the Selectivity of Spatial Queries Using the 'Correlation' Fractal Dimension. In *Proceedings of 21th International Conference on Very Large Data Bases*, pages 299–310, Zurich,Switzerland, 1995.

[BKSS90] N. Beckmann, H.P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. In *Proceedings of the 1990 ACM-SIGMOD Conference*, June 1990.

[FB74] R. Finkel and J. Bentley. Quad trees: a data structure for retrieval on composite keys. *Acta Informatica*, 4:1–9, 1974.

[FK94] C. Faloutsos and I. Kamel. Beyond Uniformity and Independence: Analysis of R-trees Using the Concept of Fractal Dimension. In *Proceedings of the 13th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 4–13, Minneapolis,Minnesota, 1994.

[FSR87] C. Faloutsos, T.K. Sellis, and N. Roussopoulosand. Analysis of Object Oriented Spatial Access Methods. In *Proceedings of the 1987 ACM-SIGMOD conference*, pages 426–439, San Francisco, California, 1987.

[Gri86] J.G. Griffiths. An Algorithm for Displaying a Class of Space-filling Curves. *Software - Practice and Experience (SPE)*, 16:403–411, 1986.

[Gut84] A. Gutman. R-trees: A Dynamic Index Structure for Spatial Searching. In *Proceedings of the 1984 ACM-SIGMOD Conference*, June 1984.

[HN83] K. Hinrichs and J. Nievergelt. The grid file: a data structure designed to support proximity queries on spatial objects. In *Proceedings of the International Workshop on Graph Theoretic Concepts in Computer Science*, pages 100–113, 1983.

[HSW89] A. Henrich, H.W. Six, and P. Widmayer. The LSD tree: spatial access to multidimensional point and non-point data. In *Proceedings of the 15th International Conference on Very Large Databases (VLDB)*, pages 45–53, 1989.

[Jag90] H.V. Jagadish. Linear Clustering of Objects with Multiple Atributes. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, pages 332–342, Atlantic City, NJ, 1990.

[JAS99] 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 University, June 1999.

[Jin99] J. Jin. Techniques for Analyzing Range Queries on R-Trees. Master's thesis, Dept. of Computer Science & Engineering, Penn State Univ., May 1999.

[KF93] I. Kamel and C. Faloutsos. On Packing R-trees. In *Proceedings of the 2nd ACM Intl. Conf. on Information and Knowledge Management*, pages 490–499, Washington, DC, 1993.

[Mar86] R.W. Marx. The TIGER System: Automating the Geographic Structure of the United States Census. *Government Publications Review*, 13:181–201, 1986.

[Mav95] D.J. Mavriplis. An Advancing Front Delaunay Triangulation Algorithm Designed for Robustness. *Journal of Computational Physics*, pages 90–101, 1995.

[MD88] M. Muralikrishna and D.J. DeWitt. Equi-Depth Histograms For Estimating Selectivity Factors For Multi-Dimensional Queries. In *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data*, pages 28–36, Chicago, Illinois, 1988.

[PF99] G. Proietti and C. Faloutsos. I/O Complexity for Range Queries on Region Data Stored Using an R-tree. In *Proceedings of the 15th International Conference on Data Engineering*, pages 628–635, Syndey, Austrialia, 1999.

[Pre] International Research Institute For Climate Prediction. IRI/LDEO Climate Data Library. http://ingrid.ldgo.columbia.edu/.

[PSTW93] B. Pagel, H-W. Six, H. Toben, and P. Widmayer. Towards an Analysis of Range Query Performance in Spatial Data Structures. In *Proceedings of the 12th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 214–221, Washington,DC, 1993.

[Rob81] J.T. Robinson. The k-d-b-tree: a search structure for large multidimensional dynamic indexes. In *Proceedings of the 1981 ACM-SIGMOD Conference*, pages 10–18, 1981.

[Sam95] H. Samet. Spatial Data Structures. *Modern Database Systems*, pages 361–385, 1995.

[SRF87] T.K. Sellis, N. Roussopoulos, and C. Faloutsos. The R+-Tree: A Dynamic Index for Multi-Dimensional Objects. In *Proceedings of the 13th International Conference on Very Large Data Bases*, pages 507–518, Brighton, England, 1987.

[TP95] Y. Theodoridis and D. Papadias. Range Queries Involving Spatial Relations: A Performance Analysis. In *Proceedings of COSIT '95*, pages 537–551, Semmering, Austria, 1995.

[TS96] Y. Theodoridis and T.K. Sellis. A Model for the Prediction of R-tree Performance. In *Proceedings of the 15th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 161–171, Montreal, Canada, 1996.

[TSS97] Y. Theodoridis, E. Stefanakis, and T. Sellis. An Efficient Cost Model for Spatial Queries Using R-trees. Technical Report KDBSLAB-TR-97-01, Department of Electrical and Computer Engineering, Computer Science Division, National Technical University of Athens, February 1997.