Efficient Cost Models for Spatial Queries Using R-Trees

Yannis Theodoridis, *Member*, *IEEE*, Emmanuel Stefanakis, and Timos Sellis, *Member*, *IEEE*

Abstract—Selection and join queries are fundamental operations in Data Base Management Systems (DBMS). Support for nontraditional data, including spatial objects, in an efficient manner is of ongoing interest in database research. Toward this goal, access methods and cost models for spatial queries are necessary tools for spatial query processing and optimization. In this paper, we present analytical models that estimate the cost (in terms of node and disk accesses) of selection and join queries using R-tree-based structures. The proposed formulae need no knowledge of the underlying R-tree structure(s) and are applicable to uniform-like and nonuniform data distributions. In addition, experimental results are presented which show the accuracy of the analytical estimations when compared to actual runs on both synthetic and real data sets.

Index Terms—Spatial databases, access methods, query optimization, cost models, R-trees.

1 Introduction

Supporting large volumes of multidimensional (spatial) data is an inherent characteristic of modern database applications, such as Geographical Information Systems (GIS), Computer-Aided Design (CAD), Image and multimedia databases. Such databases need underlying systems with extended features (query languages, data models, indexing methods) as compared to traditional databases, mainly due to the complexity of representing and retrieving spatial data. Spatial Data Base Management Systems (SDBMS), in general, should 1) offer appropriate data types and query language to support spatial data and 2) provide efficient indexing methods and cost models on the execution of specialized spatial operations, for query processing and optimization purposes [15].

In the particular field of spatial query processing and optimization, during the last two decades, several data structures have been developed for point and nonpoint multidimensional objects in low-dimensional space to meet needs in a wide area of applications, including the GIS and CAD domains. Due to the large number of spatial data structures proposed (an exhaustive survey can be found in [14]), active research in this field has recently turned to the development of analytical models that could make accurate cost predictions for a wide set of spatial queries. Powerful analytical models are useful in three ways:

1. *Structure evaluation*: This allows us to better understand the behavior of a data structure under various input data sets and sizes.
2. *Benchmarking*: This can play the role of an objective comparison point when various proposals for efficient spatial indexing are compared to each other
3. *Query optimization*: This can be used by a query optimizer in order to evaluate the cost of a complex spatial query and its execution procedure.

Spatial queries addressed by users of SDBMS usually involve *selection* (point or range) and *join* operations. In the literature, most efforts toward the analytical prediction of the performance of spatial data structures have focused on point and range queries [13], [19], [28], [11], [37], while only recently on spatial join queries [17], [38]. Some proposals support both uniform-like and nonuniform data distributions, which is an important advantage, keeping in mind that modern database applications handle large amounts of real (usually nonuniform) multidimensional data.

In this paper, we focus on the derivation of analytical formulae for range and join queries based on R-trees [16], [4]; such models support data sets of various distributions and make cost prediction based on data properties only. The proposed formulae are shown to be efficient for several distributions of synthetic and real data sets with the relative error being around 10-15 percent for any type of data sets used in our experiments.

The rest of the paper is organized as follows: In Section 2, we provide background information about hierarchical tree structures for spatial data, in particular R-tree-based ones, and related work on cost analysis for R-tree-based methods. Section 3 presents cost models for the prediction of the R-tree performance for selection and join queries. In Section 4, comparison results of the proposed models are presented with respect to efficient R-tree implementations for different data distributions. An extended survey of related work appears in Section 5, while Section 6 concludes the paper.

2 Background

Multidimensional data was first involved in geographical applications (GIS, cadastral systems, etc.), CAD, and VLSI

- *Y. Theodoridis is with the Computer Technology Institute, PO Box 1122, GR-26110, Patras, Hellas. E-mail: ytheod@cti.gr.*
- *E. Stefanakis and T. Sellis are with the Department of Electrical and Computer Engineering, National Technical University of Athens, Zografou 15773, Athens, Hellas.*
  E-mail: {stefanak, timos}@dbnet.ece.ntua.gr.

Manuscript accepted 28 Sept. 1999.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number 110165.

Fig. 1. An example of a spatial application: a database of capitals and regions.

design. Recently, especially due to the development of extensible DBMS [34], spatial data management techniques have been applied to a wide area of applications, from image and multimedia databases [9] to data mining and warehousing [10], [31]. Fig. 1 is motivated by a geographical application, where the database consists of several relations (or classes, etc.) about Europe. In particular, Fig. 1 illustrates capitals and regions of European countries (i.e., point and region data).

The most common queries on such databases include point or range queries on a specified relation (e.g., "find all countries that contain a user-defined point" or "find all countries that overlap a user-defined query window") or join queries on pairs of relations (e.g., "find all pairs of countries and motorways that overlap each other").

2.1 Spatial Queries and Spatial Data Structures

The result of the *select operation* on a relation REL_1 using a query window q contains those tuples in REL_1, with the spatial attribute standing in some relation θ to q. On the other hand, the result of the *join operation* between a relation REL_1 and a relation REL_2 contains those tuples in the Cartesian product $REL_1 \times REL_2$, where the ith column of REL_1 stands in some relation θ to the jth column of REL_2.

In traditional (alphanumeric) applications, θ is often equality. When handling multidimensional data, θ is a spatial operator, including topological (e.g., *overlap*), directional (e.g., *north*), or distance (e.g., *close*) relationships between spatial objects. For each spatial operator, with *overlap* being the most common, the query object's geometry needs to be combined with each data object's geometry.

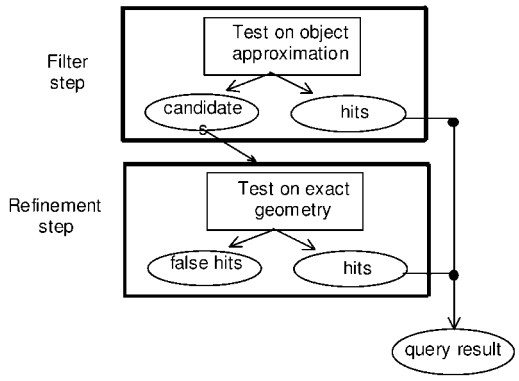


Fig. 2. Two-step spatial query processing: filter and refinement step.

However, the processing of complex representations, such as polygons, is very costly. For that reason, a two-step procedure for query processing, illustrated in Fig. 2, has been widely adopted [23]:

- *Filter step*: An approximation of each object, such as its *Minimum Bounding Rectangle* (MBR), is used in order to produce a set of candidates (and, possibly, a set of actual answers), which is a superset of the answer set consisting of actual answers and *false hits*.
- *Refinement step*: Each candidate is then examined with respect to its exact geometry in order to produce the answer set by eliminating false hits.

The filter step is usually based on multidimensional indices that organize MBR approximations of spatial objects [33]. In general, the relationship between two MBR approximations cannot guarantee the relationship between the actual objects; there are only few operators (mostly directional ones) that make the refinement step unnecessary [30].

On the other hand, the refinement step usually includes computational geometry techniques for geometric shape comparison [26] and, therefore, it is usually a time-consuming procedure since the actual geometry of the objects need to be checked. Although techniques for speeding-up this procedure have been studied in the past [5], the cost of this step cannot be considered as part of index cost analysis and, hence, it is not taken into consideration in the following.

Several methods for spatial indexing have been proposed in the past. They can be grouped in two main categories: indexing methods for points (also known as *point access methods*—PAMs) and indexing methods for nonpoint objects (also known as *spatial access methods*—SAMs). Well-known PAMs include the BANG file [12] and the LSD-tree [18], while, among the proposed SAMs, the R-tree [16], the Quadtree [33], and their variants are the most popular. In the next section, we describe the R-tree indexing method and its algorithms for search and join operations.

2.2 The R-tree

The R-tree was proposed by Guttman [16] as a direct extension of the B^+-tree [20], [7] in d dimensions. The data structure is a height-balanced tree that consists of intermediate and leaf nodes. A *leaf node* is a collection of entries

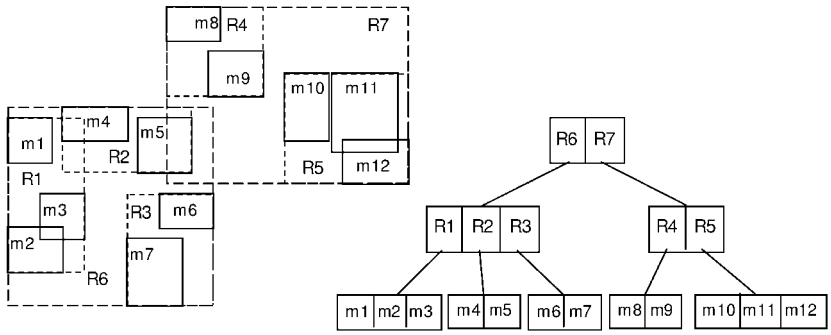


Fig. 3. Some rectangles organized to form an R-tree, and the corresponding R-tree.

of the form (oid, R), where oid is an object identifier used to refer to an object in the database and R is the MBR approximation of the data object. An *intermediate node* is a collection of entries of the form (ptr, R), where ptr is a pointer to a lower level node of the tree and R is a representation of the minimum rectangle that encloses all MBRs of the lower-level node entries.

Let M be the maximum number of entries in a node and let $m \leq M/2$ be a parameter specifying the minimum number of entries in a node. An R-tree satisfies the following properties:

1. Every leaf node contains between m and M entries unless it is the root.
2. For each entry (oid, R) in a leaf node, R is the smallest rectangle that spatially contains the data object represented by oid.
3. Every intermediate node has between m and M children unless it is the root.
4. For each entry (ptr, R) in an intermediate node, R is the smallest rectangle that completely encloses the rectangles in the child node pointed by ptr.
5. The root node has at least two children unless it is a leaf.
6. All leaves appear in the same level.

After Guttman's proposal, several researchers proposed their own improvements on the basic idea. Among others, Roussopoulos and Leifker [32] proposed the packed R-tree, for the case that data rectangles are known in advance (i.e., it is applicable only to static databases), Sellis et al. [35] proposed the R^+-tree, a variant which guarantees disjointness of nodes by introducing redundancy, and Beckmann et al. [4] proposed the R^*-tree, an R-tree-like structure that uses a rather complex, but more effective, grouping algorithm. Gaede and Guenther [14] offer an exhaustive survey of multidimensional access methods including several other variants of the original R-tree technique. As an example, Fig. 3 illustrates a set of data rectangles and the corresponding R-tree built on these rectangles (assuming maximum node capacity $M = 4$).

The processing of any type of spatial query can be accelerated when a spatial index (e.g., an R-tree) exists. Selection queries, for example, perform a traversal of the R-tree index: Starting from the root node, several tree nodes are accessed down to the leaves, with respect to the result of the overlap operation between q and the corresponding node rectangles. When the search algorithm for spatial selection (so-called SS) accesses the leaf nodes, all data rectangles that overlap the query window q are added into the answer set.

```
// Spatial Selection Algorithm for R-trees
SS (R1: R-tree node, q: rectangle)
01 BEGIN
02    FOR all Er1 in R1 DO
03       IF Er1.rect overlaps q THEN
04          IF R1 is a leaf page THEN
05                Output (Er1.oid)
06          ELSE
07                ReadPage (Er1.ptr);
08                SS (Er1.ptr, q)
09          END-IF
10       END-IF
11    END-FOR
12 END.
```

On the other hand, the join operation between two spatial relations REL_1 and REL_2 can be implemented by applying synchronized tree traversals on both R-tree indices. An algorithm based on this general idea, called *SpatialJoin1*, was originally introduced by Brinkhoff et al. in [3]. Two improvements of this algorithm were also proposed in the same paper toward the reduction of the CPU- and I/O-cost by taking into consideration faster main-memory algorithms, and better read schedules for a given LRU-buffer, respectively. Specifically, for two R-trees rooted by nodes R_1 and R_2, respectively, the spatial join procedure (algorithm *SJ*) is as follows:

```
// Spatial Join Algorithm for R-trees
SJ (R1, R2: R-tree nodes)
01 BEGIN
02    FOR all Er2 in R2 DO
03       FOR all Er1 in R1 DO
04          IF Er1.rect overlaps Er2.rect THEN
05             IF R1 and R2 are leaf pages THEN
06                Output (Er1.oid, Er2.oid)
07             ELSE IF R1 is a leaf node THEN
08                ReadPage (Er2.ptr);
09                SJ (Er1.ptr, Er2.ptr)
10             ELSE IF R2 is a leaf node THEN
11                ReadPage (Er1.ptr);
12                SJ (Er1.ptr, Er2.ptr)
```

```
13              ELSE
14                 ReadPage(Er1.ptr); ReadPage(Er2.ptr);
15                 SJ(Er1.ptr, Er2.ptr)
16              END-IF
17           END-IF
18        END-FOR
19     END-FOR
20 END.
```

In other words, a synchronized traversal of both R-trees is performed, with the entries of nodes R_1 and R_2 playing the roles of data and query rectangles, respectively, in a series of range queries.

For both operations, the total cost is measured by the total amount of page accesses in the R-tree index (the procedure *ReadPage* in the above algorithms). This procedure either performs an actual read operation on the disk or reads the corresponding node information from a memory-resident buffer, thus we distinguish between node and disk accesses in the analytical study that follows. (The distinction between node and disk accesses is a subject of cache policy. By definition, the number of node accesses is always greater than or equal to the number of actual disk accesses; the equality only holds for the case where no buffering scheme exists.)

3 Analytical Cost Models for Spatial Queries

Complex queries are usually transformed by DBMS query optimizers to a set of simpler ones and the execution procedure takes the partial costs into account in order to schedule the execution of the original query. Thus, query optimization tools that estimate access cost and selectivity of a query are complementary modules together with access methods and indexing techniques. Traditional optimization techniques usually include heuristic rules, which, however, are not effective in spatial databases due to the peculiarity of spatial data sets (multidimensionality, lack of total ordering, etc.). Other, more sophisticated, techniques include histograms and cost models. Although research on multidimensional histograms has recently appeared in the literature [24], in the spatial database literature, cost models for selectivity and cost estimation seem to be the most promising solutions.

Proposals in this area include models for spatial selection [13], [19], [28], [21] and spatial join [2], [17]. However, as will be discussed later, in Section 5, most of those proposals require *knowledge of index properties* and/or make a *uniformity assumption*, thus rendering them incomplete tools for the practical purposes of query optimization. In [37], [38], we proposed appropriate extensions to solve those problems, which are formally presented in the rest of the section. Throughout our discussion we use the list of symbols that appear in Table 1.

3.1 Selection Queries

Formally, the problem of the R-tree cost analysis for selection queries is defined as follows: Let d be the dimensionality of the data space and $WS = [0,1)^d$ the d-dimensional unit workspace. Let us assume that N_{R_1} data rectangles are stored in an R-tree index R_1 and a query asking for all rectangles that *overlap* a query window $q = (q_1, \ldots, q_d)$ needs to be answered. What is sought is a formula that estimates the average number NA of node accesses using only knowledge about data properties (i.e., without extracting information from the underlying R-tree structure).

Definition. *The* density D *of a set of* N *rectangles with average extent* $s = (s_1, \ldots, s_d)$ *is the average number of rectangles that contain a given point in* d*-dimensional space.*

Equivalently, D can be expressed as the ratio of the sum of the areas of all rectangles over the area of the available workspace. If we consider a unit workspace $[0,1)^d$, with area equal to 1, then the density $D(N,s)$ is given by the following formula:

$$D(N,s) = \sum_{N} \prod_{k=1}^{d} s_k = N \cdot \prod_{k=1}^{d} s_k. \tag{1}$$

Assume now an R-tree R_1 of height h_{R_1} (the root is assumed to be at level h_{R_1} and leaf-nodes are assumed to be at level 1). If $N_{R_1,l}$ is the number of nodes at level l and $s_{R_1,l}$ is their average size, then the expected number of node accesses in order to answer a selection query using a query window q is defined as follows (assuming that the root node is stored in main memory):

$$NA\_total(R_1, q) = \sum_{l=1}^{h_{R_1}-1} intsect\big(N_{R_1,l}, s_{R_1,l}, q\big), \tag{2}$$

where $intsect(N_{R_1,l}, s_{R_1,l}, q)$ is a function that returns the number of nodes at level l intersected by the query window q. In other words, (2) expresses the fact that the expected number of node accesses is equal to the expected number of intersected nodes at each level.

Lemma. *Given a set of* N *rectangles* $r_1, \ldots, r_N$ *with average extent* s *and a rectangle* r *with extent* q*, the average number of rectangles* r_i *intersected by* r *is:*

$$intsect(N, s, q) = N \cdot \prod_{k=1}^{d} (s_k + q_k). \tag{3}$$

Proof. The average number of a set of N rectangles with average extent s that intersect a rectangle r with extent q is equal to the number of a second set of N rectangles with average extent s' (where $s'_k = s_k + q_k, \forall k$) that contain a point in the workspace. The latter one, by definition, equals to the density D' of the second set of rectangles. Formally:

$$intsect(N, s, q) = intsect(N, s', 0) = D'(N, s') = N \cdot \prod_{k=1}^{d} s'_k,$$

where $s'_k = s_k + q_k$. □

Assuming that rectangle r represents a query window on R_1, we derive $NA\_total(R_1, q)$ by combining (2) and (3):

TABLE 1
List of Symbols

Symbol	Definition
d	number of dimensions
f	average R-tree node fanout
h_{R_i}	height of the R-tree R_i
N_{R_i}	cardinality - number of data rectangles indexed in the R-tree R_i
D_{R_i}	density of data rectangles indexed in the R-tree R_i
$N_{R_i,l}$	number of nodes of the R-tree R_i at level l
$D_{R_i,l}$	density of node rectangles of the R-tree R_i at level l
q_k	average extent of a query rectangle q on dimension k (k = 1, ..., d)
$s_{R_i,l,k}$	average extent of node rectangles of the R-tree R_i at level l on dimension k (k = 1, ..., d)

$$NA\_total(R_1, q) = \sum_{l=1}^{h_{R_1}-1} \left\{ N_{R_1,l} \cdot \prod_{k=1}^{d} \left(s_{R_1,l,k} + q_k\right) \right\}. \quad (4)$$

In order to reach our goal, we have to express (4) as a function of the data properties N_{R_1} (number of data rectangles or, in other words, cardinality of the data set) and D_{R_1} (density of the data set) or, equivalently, express the R-tree properties h_{R_1}, $N_{R_1,l}$, and $s_{R_1,l,k}$ as functions of the data properties N_{R_1} and D_{R_1}.

The height h_{R_1} of an R-tree R_1 with average node capacity (fanout) f that stores N_{R_1} data rectangles is given by the following formula [13]:

$$h_{R_1} = 1 + \left\lceil \log_f \frac{N_{R_1}}{f} \right\rceil. \quad (5)$$

Since a node organizes, on the average, f rectangles, we can assume that the average number of leaf nodes (i.e., $l = 1$) is $N_{R_1,1} = N_{R_1}/f$, and so on. Hence, in general, the average number of R-tree nodes at level l is

$$N_{R_1,l} = N_{R_1}/f^l. \quad (6)$$

A second assumption in our analysis is that we consider square node rectangles (i.e., $s_{R_1,l,1} = \cdots = s_{R_1,l,d}, \forall l$). This *squaredness assumption* is a reasonable property for a "good" R-tree [19]. According to that assumption and (1) and (6), the average extent $s_{R_1,l,k}$ of node rectangles at level l is a function of their density $D_{R_1,l}$:

$$D_{R_1,l} = N_{R_1,l} \cdot \prod_{k=1}^{d} s_{R_1,l,k} = \frac{N_{R_1}}{f^l} \cdot \left(s_{R_1,l,k}\right)^d \Rightarrow$$
$$s_{R_1,l,k} = \left(D_{R_1,l} \cdot \frac{f^l}{N_{R_1}} \right)^{1/d}. \quad (7)$$

What remains to be estimated is $D_{R_1,l}$. Suppose that, at level l, $N_{R_1,l}$ nodes with average size $\left(s_{R_1,l,k}\right)^d$ are organized in $N_{R_1,l+1}$ parent nodes with average size $\left(s_{R_1,l+1,k}\right)^d$. As described in [39], each parent node groups on the average f child nodes and $f^{1/d}$ out of them are responsible for the size of the parent node along each direction. The centers of the $N_{R_1,l}$ node rectangles' projections are assumed to be equally distanced and this distance (denoted by $t_{l,k}$) depends on $N_{R_1,l}^{1/d}$ nodes along each direction. Hence, the average extent $s_{R_1,l+1,k}$ of a parent node along each direction is calculated by:

$$s_{R_1,l+1,k} = \left(f^{1/d} - 1\right) \cdot t_{l,k} + s_{R_1,l,k}, \quad (8)$$

where $t_{l,k}$ is:

$$t_{l,k} = 1/{N_{R_1,l}}^{1/d}, \forall l, \quad (9)$$

denoting the distance between the centers of two consecutive rectangles' projections on dimension k. (To derive (9), we divided the extent of the unit workspace by the number of different node projections on dimension k.)

Lemma. *Given a set of $N_{R_1,l}$ node rectangles with density $D_{R_1,l}$, the density $D_{R_1,l,k}$ of their projections is derived by the following formula:*

$$D_{R_1,l,k} = \left(D_{R_1,l} \cdot {N_{R_1,l}}^{d-1}\right)^{1/d}. \quad (10)$$

Proof. Straightforward by applying (1), (6), and (7) (for proof details see [39]). □

Using (8), (9), and (10), the density $D_{R_1,l+1}$ of node rectangles at level $l+1$ is a function of the density $D_{R_1,l}$ of node rectangles at level l:

$$D_{R_1,l+1} = \left(1 + \frac{D_{R_1,l}^{1/d} - 1}{f^{1/d}}\right)^d. \quad (11)$$

Essentially, by using (11), the density at each level of the R-tree can be calculated as only a function of the density D_{R_1} of the data rectangles (which, according to the terminology adopted, can be named $D_{R_1,0}$).

At this point, the original goal of our analysis has been reached. By combining (4), (5), (6), (7), and (11), the expected number $NA\_total(R_1, q)$ of node accesses for a selection query can be computed by using only the data set properties N_{R_1} and D_{R_1}, the typical R-tree parameter f, and the query window q.

3.2 Join Queries

According to the discussion of Section 2.2, the processing cost of a join query is equal to the total cost of a set of appropriate range queries, according to the procedure described in the algorithm *SJ*. In this section, we propose a formula that estimates the cost of a join query.

Formally, the problem of R-tree cost analysis for join queries is defined as follows: Let d be the dimensionality of the workspace and $WS = [0,1)^d$ the d-dimensional unit workspace. Let us assume two spatial data sets of cardinality N_{R_1} and N_{R_2}, with the corresponding MBR approximations of spatial data being stored in two R-tree indices R_1 and R_2, respectively. In correspondence with Section 3.1, the goal of the cost analysis in this section is a formula that would efficiently estimate the average number of nodes accessed in order to process a join query between the two data sets, based on the knowledge of the data properties only and without extracting information from the corresponding R-tree structures.

Let the height of an R-tree R_i $(i = 1, 2)$ be equal to h_{R_i} and assume that both root nodes are stored in main memory. At each level l_i, $1 \le l_i \le h_{R_i} - 1$, R_i contains N_{R_i,l_i} nodes of average size s_{R_i,l_i}, each node consisting of a set of entries E_{R_i,l_i}. Thus, in order to find which pairs of entries are overlapping and downward traverse the two structures, we compare entries E_{R_1,l_1} with entries E_{R_2,l_2} (line 04 of the spatial join algorithm *SJ*).

The cost of the above comparison at each level is the sum of two factors, which correspond to the costs for the two R-trees, namely $NA(R_1, R_2, l_1)$ and $NA(R_2, R_1, l_2)$, respectively. The above factors can be calculated by considering that the entries of R_1 (respectively, R_2) play the role of the data set (respectively, a set of query windows q); then we can derive the pairs of intersecting nodes for each R-tree ((3) in Section 3.1) in order to estimate the total access cost. Since we do not consider the existence of a buffering scheme, the access costs for both R-trees R_1 and R_2 at the corresponding levels l_1 and l_2 are equal (since an equal number of nodes are accessed, as can be extracted by algorithm *SJ*, line 14). Line 04 of the *SJ* algorithm is repeatedly called at each level of the two trees down to the leaf level of the shorter tree R_2 (without loss of generality, it is assumed that $h_{R_1} \ge h_{R_2}$).

Formally, for the upper $h_{R_2} - 1$ levels of the two R-trees:

$$\begin{aligned} NA(R_1, R_2, l_1) &= NA(R_2, R_1, l_2) \\ &= \sum_{N_{R_2,l_2}} intsect\big(N_{R_1,l_1}, s_{R_1,l_1}, s_{R_2,l_2}\big) \Rightarrow \ldots \\ &\quad \Rightarrow NA(R_1, R_2, l_1) \\ &= NA(R_2, R_1, l_2) \\ &= N_{R_2,l_2} \cdot N_{R_1,l_1} \cdot \prod_{k=1}^{d} \big(s_{R_1,l_1,k} + s_{R_2,l_2,k}\big), \end{aligned} \tag{12}$$

where $h_{R_1} - h_{R_2} + 1 \le l_1 \le h_{R_1} - 1$ and $1 \le l_2 \le h_{R_2} - 1$.

In [38], we provide a slightly different formula: in particular, instead of the factor $\big(s_{R_1,l_1,k} + s_{R_2,l_2,k}\big)$, an upper bound $\min\big\{1, \big(s_{R_1,l_1,k} + s_{R_2,l_2,k}\big)\big\}$ is presented. Since, in the whole discussion of the present section, we discuss unit workspace, such a bound is already a presupposition in (3), (4), and (12).

After the leaf nodes of the short tree R_2 have been accessed, l_2 is fixed to value 1 (i.e., denoting the leaf level) and the propagation of R_1 continues down for another $h_{R_1} - h_{R_2}$ levels. Overall, (13) estimates the total cost in terms of R-tree node accesses for a spatial join:

$$NA\_total(R_1, R_2) = \sum_{l_1=1}^{h_{R_1}-1} \{NA(R_1, R_2, l_1) + NA(R_2, R_1, l_2)\}, \tag{13}$$

where

$$l_2 = \begin{cases} l_1 - (h_{R_1} - h_{R_2}), & h_{R_1} - h_{R_2} + 1 \le l_1 \le h_{R_1} - 1 \\ 1, & 1 \le l_1 \le h_{R_1} - h_{R_2}. \end{cases}$$

The involved parameters in (13) are:

- h_{R_i}, calculated by (5),
- N_{R_i,l_i}, calculated by (6) as a function of the actual cardinality N_{R_i} of the corresponding data set, and
- $s_{R_i,l_i,k}$, calculated by (7) as a function of the density D_{R_i,l_i} of node rectangles at level l_i (which, in turn, is calculated by (11) as a function of the actual density D_{R_i} of the corresponding data set).

Qualitatively, (13) estimates the cost of a join query between two spatial data sets based on their primitive properties only, namely *number* and *density* of data rectangles, in correspondence with the analysis for range queries (Section 3.1). Note that (13) is symmetric with respect to the two indices R_1 and R_2. The same conclusion is drawn by the study of the algorithm *SJ* since the number of node accesses is equal to the number of *ReadPage* calls (line 14) which, in turn, are the same for both trees. The equivalence of the two indices is not the case when a simple *path buffer* (i.e., a buffer that stores the most recently visited path for each R-tree) is introduced, as we will discuss in the next section.

3.3 Introducing a Path Buffer

Extending previous analysis, we introduce a simple buffering mechanism that maintains a path buffer for the underlying tree structure(s). The existence of such a buffer mainly affects the performance of the tree index that plays the role of the query set (namely R_2), as will be discussed in

detail in this section, since the search procedure (algorithm *SJ*), reads R_2 entries less frequently than R_1 entries. With respect to that, we assert that the cost of a selection query in terms of disk accesses $DA\_total(R_1, q)$ can be considered equal to $NA\_total(R_1, q)$, as formulated in (4), without loss of accuracy and, therefore, provide no further analysis for path buffering. Moreover, the effect of a buffering mechanism (e.g., an LRU buffer) has been already addressed in the literature [21] and, according to experimental results, very low buffer size (such as that of a path buffer) causes almost zero impact on point and range query performance. On the other hand, even a simple path buffer scheme highly affects the actual cost of a join query.

As already mentioned, by examining algorithm *SJ*, we conclude that the existence of such a buffering scheme mainly affects the computation of the cost for R_2 because its entries constitute the outer loop of the algorithm and, hence, are less frequently updated. As for R_1, since its entries constitute the inner loop of the algorithm, the respective cost computation is not considerably affected by the existence of a path buffer. These arguments are more formally described through the following alternative cases (for more details through an example, see [39]):

1. Suppose that an entry E_{R_2,l_2} of tree R_2 at level l_2 overlaps with m entries of a node E_{R_1,l_1+1}, m entries of a different node E'_{R_1,l_1+1}, etc., of the tree R_1. E_{R_2,l_2} is kept in main memory during its comparison with all entries of node E_{R_1,l_1+1} and will be again fetched from disk, hence, recomputed in $DA(R_2, R_1, l_2)$ cost, when its comparison with the entries of node E'_{R_1,l_1+1} starts. As a result, the number of actual disk accesses of the node rooted by E_{R_2,l_2} is equal to the number of the nodes of R_1 at level $l_1 + 1$ (i.e., the parent level) having rectangles intersected by E_{R_2,l_2}; formally, it is $intsect(N_{R_1,l_1+1}, s_{R_1,l_1+1}, s_{R_2,l_2})$.
2. On the other hand, an entry E_{R_1,l_1} of tree R_1 at level l_1 is recomputed in $DA(R_1, R_2, l_1)$ as soon as it overlaps with an entry E_{R_2,l_2} of tree R_2 with a single exception: In case E_{R_1,l_1} is the last member of the intersection set of E_{R_2,l_2} and, at the same time, it is the first member of the intersection set of its consecutive E'_{R_2,l_2}. The above exception happens rarely; moreover, it is hardly modeled since no order can be assumed among R-tree node entries.

Since the cost for the tree that plays the role of the query (data) set is affected in a high (low) degree, we distinguish between two different cases: R_1 that plays the role of the data set being 1) taller (or of equal height) or 2) shorter than R_2. In the first case ($h_{R_1} \geq h_{R_2}$), the propagation of R_1 down to its lower levels adds no extra cost (in terms of disk accesses) to the "query" tree R_2 that has already reached its leaf level. In the second case ($h_{R_1} < h_{R_2}$), each propagation of the "query" tree R_2 down to its lower levels adds equal cost to the "data" tree R_1 (denoting that buffer existence does not affect the cost of R_1).

Overall, with respect to the above discussion, the access cost of each tree at a specific level l_i is calculated according to the following formulae:

$$DA(R_1, R_2, l_1) \approx NA(R_1, R_2, l_1) \tag{14}$$

$$\begin{aligned} DA(R_2, R_1, l_2) &= \sum_{N_{R_2,l_2}} intsect\big(N_{R_1,l_1+1}, s_{R_1,l_1+1}, s_{R_2,l_2}\big) \\ &= N_{R_2,l_2} \cdot N_{R_1,l_1+1} \cdot \prod_{k=1}^{d} \big(s_{R_1,l_1+1,k} + s_{R_2,l_2,k}\big) \end{aligned} \tag{15}$$

and the total cost is given by (16) (note that if $h_{R_1} \geq h_{R_2}$, then $l_2 = l_1 - (h_{R_1} - h_{R_2})$, else $l_1 = l_2 - (h_{R_2} - h_{R_1})$.

$$DA\_total(R_1, R_2) = \begin{cases} \sum_{l_1=h_{R_1}-h_{R_2}+1}^{h_{R_1}-1} \{DA(R_1, R_2, l_1) & \\ \quad + DA(R_2, R_1, l_2)\} & \text{if } h_{R_1} \geq h_{R_2} \\ \quad + \sum_{l_1=1}^{h_{R_1}-h_{R_2}} DA(R_1, R_2, l_1), & \\ \sum_{l_2=h_{R_2}-h_{R_1}+1}^{h_{R_2}-1} \{DA(R_1, R_2, l_1) & \\ \quad + DA(R_2, R_1, l_2)\} & \text{if } h_{R_1} < h_{R_2} \\ \quad + \sum_{l_2=1}^{h_{R_2}-h_{R_1}} \{2 \cdot DA(R_1, R_2, l_1)\}, & \end{cases} \tag{16}$$

Again, the involved parameters h_{R_i}, N_{R_i,l_i}, and $s_{R_i,l_i,k}$ are calculated as functions of N_{R_i} and D_{R_i}. Note that, in contrast to (13), (16) is sensitive to the two indices, R_1 and R_2. The experimental results of Section 4 also strengthen this observation.

In the above analysis, we have taken two cases into consideration: adopting either zero or a simple path buffer scheme. A more complex buffering scheme (e.g., an LRU buffer of predefined size) would surely achieve a lower value for $DA\_total()$. However, its effect is beyond the scope of this paper (see [17], [21] for related work).

3.4 Support for Nonuniform Data Sets

The proposed analytical model assumes data uniformity in order to compute the density of the R-tree node rectangles at a level $l + 1$ as a function of the density of the child node rectangles at level l (11). In particular, in order to derive (8) and (9), the centers of the $N_{R_1,l}$ node rectangles' projections were assumed to be equally distanced. This *uniformity assumption* [13] leads to a model that could be efficient for uniform-like data distributions, but hardly applicable to nonuniform distributions of data, which are the rule when dealing with real applications.

In order to adapt the model in a way that would efficiently support several types of data sets (uniform or nonuniform ones), we reduce the *global* uniformity assumption of the analytical model (i.e., by considering the whole workspace) to a *local* uniformity assumption (by considering a small subarea of the workspace) according to the following motivation: The densityof a data set is involved in the cost formulae as a single number D_{R_i}. However, for nonuniform data sets, density is a varying parameter, graphically a surface in d-dimensional space. Such a surface could show strong deviations from point to point of the workspace, compared to the average value. For example, in

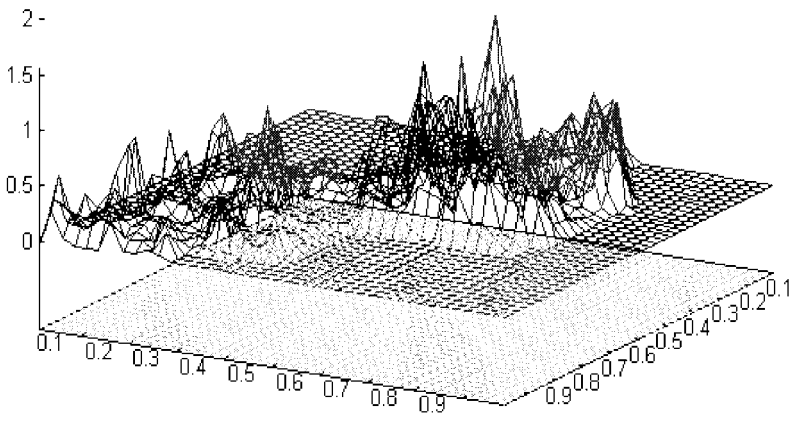


Fig. 4. The density surface of a real data set.

Fig. 4, the *density surface* of a real data set, called *Lbeach*, is drawn (see Fig. 5 in Section 4 for an illustration of the *Lbeach* data set [6]).

The average density of this data set is $D_{avg} = 0.13$. However, as extracted from Fig. 4, the actual density varies from $D = 0.0$ (in zero-populated areas, such as the upper-left and bottom-right corners) up to $D = 2.0$ (in high-populated areas), with respect to the reference point. Evidently, using the D_{avg} value in a cost formula would usually provide inaccurate estimations. On the other hand, a satisfactory illustration of the density surface provides more accurate D values with respect to a specified query window q. Although the cardinality of a data set could be very large, a near-to-the-actual density surface can usually be computed by examining a sample of the data set (efficient sampling algorithms have been proposed, among others, by Vitter in [40], [41]).

Based on the above idea, the proposed cost formulae could efficiently support either uniform or nonuniform data distributions after the following adaptations:

1. The average density D_{R_i} of the data set is replaced by the actual density D'_{R_i} of the data set within the area of the specified query window q.
2. The amount N_{R_i} of the data set is transformed to N'_{R_i}, which is computed as follows:

$$N'_{R_i} = \left(D'_{R_i}/D_{R_i}\right) \cdot N_{R_i}.$$

Note that, in discrete space, the average density of a data set is always $D_{R_i} > 0$, even for point data sets; $D_{R_i} = 0$ corresponds to zero-populated areas only.

In this section, we provided analytical formulae for the cost estimation of selection or join queries on spatial data sets organized by disk-resident R-tree indices. The proposed cost models are based on primitive data properties only, without any knowledge of the corresponding R-trees. In the next section, we evaluate our model by comparing the analytical estimations with experimental results on synthetic and real data sets in one- and two-dimensional space.

4 Evaluation of the Cost Models

The evaluation of the analytical formulae proposed in Section 3 is based on a variety of experimental tests on *synthetic* and *real* data sets illustrated in Fig. 5. Synthetic one- and two- dimensional data sets consist of random (Fig. 5a) and skewed (Fig. 5b) distributions of varying cardinality N ($20\text{K} \leq N \leq 80\text{K}$) and density D ($0.2 \leq D \leq 0.8$), and have been constructed by using random number generators. Real two-dimensional data sets are parts of the TIGER/Line database of the U.S. Bureau of Census [6]. In particular, we have used two TIGER data sets: 1) the *LBeach data set*, consisting of 53,143 line segments (stored as rectangles) indicating roads of Long Beach, California (Fig. 5c) and 2) the *MGcounty data set*, consisting of 39,221 line segments (stored as rectangles) indicating roads of Montgomery County, Maryland (Fig. 5d).

For the experimental tests we built R*-trees [4] and performed several spatial joins using the above data sets. All experimental results were run on an HP700 workstation with 256 Mbytes of main memory. On the other hand, the analytical estimations of node accesses for selection queries were based on (4) and the node (respectively, disk) cost estimations for join queries were based on (13) (respectively, (16)) with the average capacity of the tree indices set to the typical 67 percent value.

4.1 Experiments on Uniform-like Data Sets

We present several test results in order to evaluate the cost estimation of (4) for selection (either point or range) queries. Fig. 6 illustrates the results for two random data sets ($N = 40\text{K}$ and 80K, respectively, both with density $D = 0.1$). The relative error was always below 10 percent for the two experiments illustrated in Fig. 6, as well as for the rest experiments with random data sets.

As a further step, we evaluated the analytical formulae for join query estimation, presented in Section 3, on various R-tree combinations. Fig. 7 illustrates the experimental and analytical results of node and disk accesses (denoted by *NA* and *DA*) for 1) one- and 2) two-dimensional random data sets, respectively, for all N_{R_1}/N_{R_2} combinations.

The nonlinearity of the plots in Fig. 7b is due to the fact that all R-tree indices are not of equal height; the height of

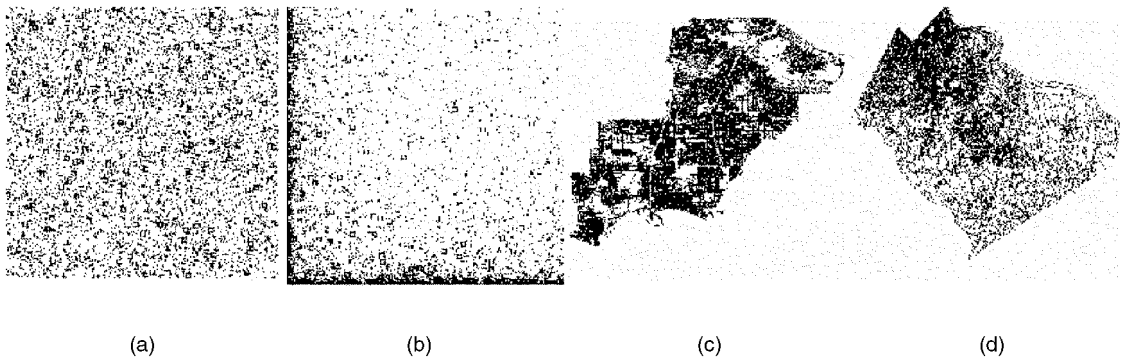


Fig. 5. Two-dimensional data sets used in the experiments: (a) synthetic random (uniform-like) data, (b) synthetic skewed (Zipf) data, (c) LBeach real data, (d) MGcounty real data.

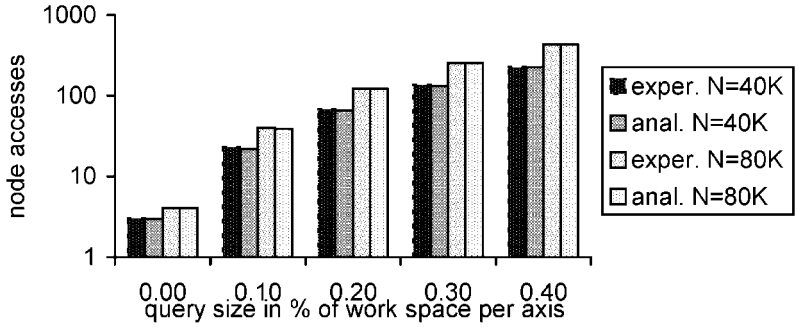


Fig. 6. Performance comparison for selection queries on uniform-like data ($d = 2$, $D = 0.1$, $N = 40\text{K}$ or 80K, log scale).

the two-dimensional indices of cardinality $20\text{K} \leq N \leq 40\text{K}$ (respectively, $60\text{K} \leq N \leq 80\text{K}$) is equal to $h = 3$ (respectively, $h = 4$), while the height of all one-dimensional indices is equal to $h = 3$. According to our experiments, it also turns out that the cost formulae for the estimation of disk accesses DA are nonsymmetric with respect to the trees R_1 and R_2, a fact that has been already mentioned during the presentation of the cost models in Section 3.

The comparison results confirm that, for tree indices of equal height, the choice of the smaller (larger) index to play the role of the "query" ("data") tree is the best choice for the effectiveness of the SJ algorithm, which, however, is not a general rule for trees of different height, as illustrated in Fig. 8 (areas 2 and 3 in Fig. 8 do not obey this rule).

Summarizing the results for join queries on random (uniform-like) data sets, we conclude that:

1. When no buffering scheme is considered, the accuracy of the estimation is high (the relative error never exceeds 10 percent).
2. When a path buffer is adopted then the estimated cost of R_2 is always very close to the actual cost (relative error usually below 5 percent), while the estimated cost of R_1 is usually within 10-15 percent of the experimental result. The accuracy of the estimation concerning R_2 (i.e., the "query" tree in the join procedure) is expected since the existence of a buffer has been taken into account in (15), while (14) assumes that the buffer existence does not affect R_1 (i.e., the "data" tree in the join procedure), an assumption that reduces the accuracy of the estimation for the access cost of R_1. However, as already mentioned in Section 3.2, the exception to the rule is hardly modeled.

4.2 Nonuniform Data Sets

As explained in Section 3, a transformation of the actual density of each nonuniform data set is necessary in order to reduce the impact of the *uniformity assumption* of the underlying analytical model from *global* (i.e., assuming the global workspace) to *local* (i.e., assuming a small subarea of the workspace). In other words, instead of considering the average density D_{avg} of a data set, the cost formulae ((4), (13), and (16)) consider the values of the density surface $D(x, y)$ that correspond to the appropriate areas of the workspace. For experimentation purposes, we extracted a density surface for each nonuniform data set using a grid of 40×40 cells, i.e., a step of 0.25 percent of the workspace per axis.

Fig. 9 illustrates average results for selection queries on skewed ($d = 2$, $D = 0.1$, $N = 40\text{K}$ or 80K) and real (LBeach and MGcounty) data sets. The analytical results (respectively, experimental results) are plotted with dotted (respectively, solid) lines. The relative error is usually around 10-15 percent and this was the rule for all data sets that we tested.

The flexibility of the proposed analytical model on nonuniform distributions of data, using the *density surface*, is also extracted from the results of our experiments. Fig. 10 illustrates the results for range queries (query rectangles of size 0.1^2) around nine representative points on the above skewed and real data sets ($d = 2$). Note that the analytical estimate always follows the experimental value for each

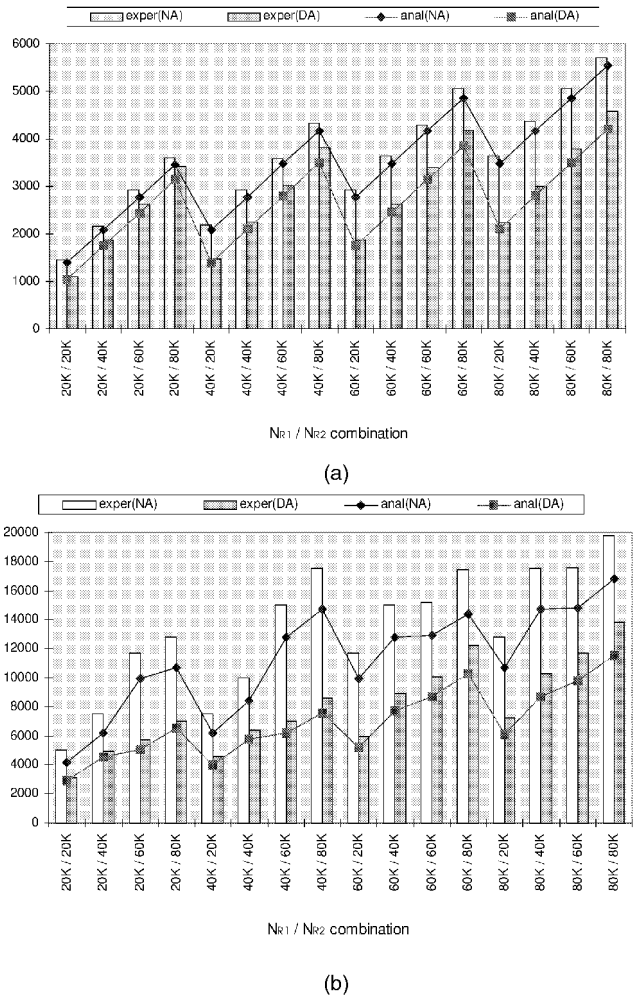


Fig. 7. Experimental vs. analytical *NA* and *DA* costs for join queries on uniform-like data: (a) $d = 1$, (b) $d = 2$.

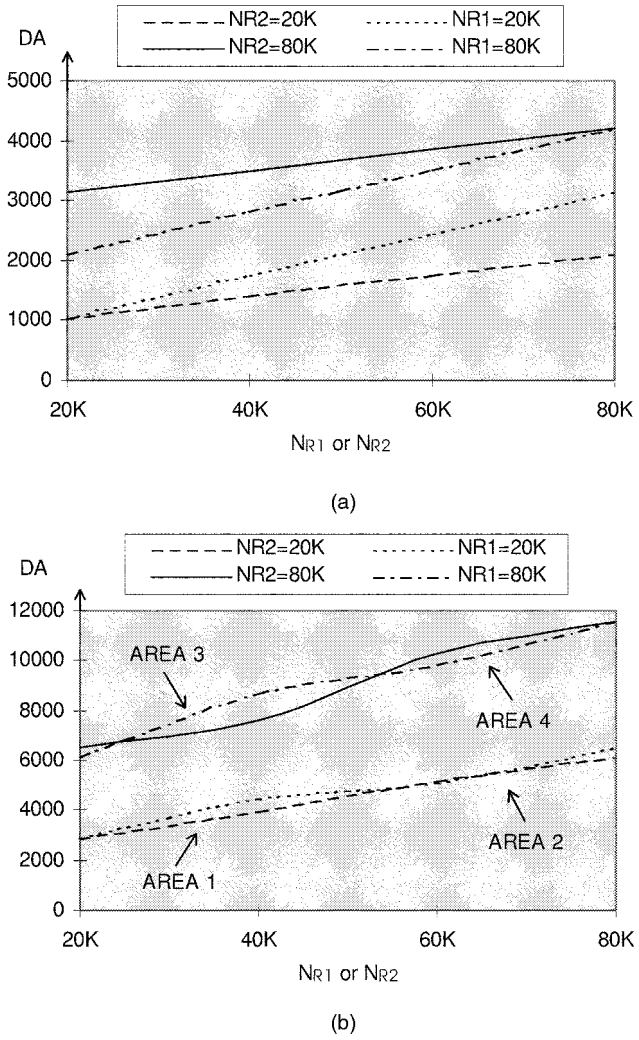


Fig. 8. Analytical *DA* costs for join queries on uniform-like data for varying cardinality N_{R_1} or N_{R_2}: (a) $d = 1$, (b) $d = 2$.

representative point (the same behavior appears for point queries also [39]).

The evaluation of the model for join queries also includes a wide set of experiments. Fig. 11a illustrates weighted average costs on skewed data sets for varying density D ($d = 2$). (The weighted average number of disk accesses, denoted by $w.DA$ is computed by multiplying each measured cost DA_i with a factor inversely proportional to the corresponding cardinality N_i, i.e., $w.DA = \frac{1}{4} \cdot \sum_{i=1}^{4} w_i \cdot DA_i$, where $w_i = 10\text{K}/N_i$ in our experiments, in order to achieve fair-minded portions for both low- and high-populated indices). Apart from synthetic data sets, we also used the two real data sets described earlier. Fig. 11b illustrates the corresponding experimental and analytical results. The labels _lb_ and _mg_ (respectively, _lb'_ and _mg'_) denote the actual (respectively, after mirroring of x- and y-axis) LBeach and MGcounty data sets. In general, a relative error below 20 percent appears for all nonuniform data combinations.

Summarizing the experiments, Table 2 lists the average relative errors of the actual results compared to the predictions of the proposed model.

4.3 The Benefit When Using a Path Buffer

As discussed in Section 3.3, the larger the buffer size in a DBMS, the lower the access cost for a selection or join query. However, the benefit for spatial selection queries by using a simple path buffer is not clearly measurable; as already mentioned, according to a related work [21], when the buffer size is close to zero then no significant performance gain is achieved.

On the other hand, a path buffer clearly affects the performance of join queries, as the gaps between the lines that represent *NA* and *DA* in Figs. 7 and 11 indicate. This difference is illustrated in Fig. 12 with *NA* values being fixed to value 100 percent and, hence, *DA* values showing the relative performance gain.

Significant savings of 10-30 percent appear for one-dimensional data. Recalling that all one-dimensional data sets of our experiments generated equal height trees, one can observe that the smaller the "query" tree, the higher the gain becomes. For two-dimensional data sets, the performance gain increases up to a 50 percent level. The above conclusion is also true in this case, showing, however, a less

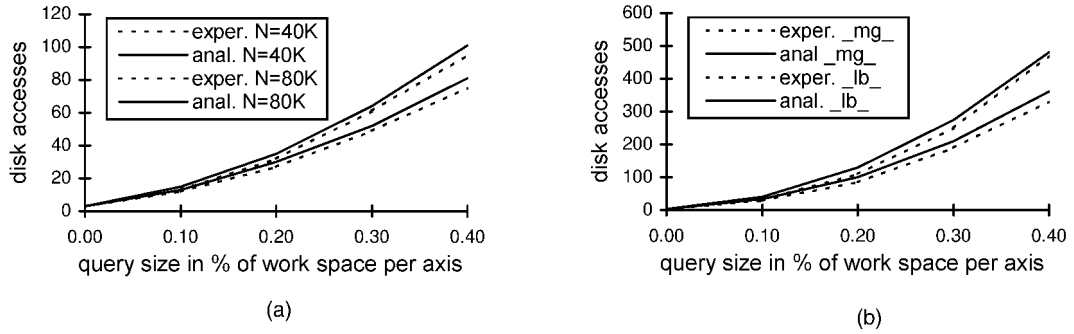


Fig. 9. Performance comparison for selection queries on (a) skewed and (b) real data.

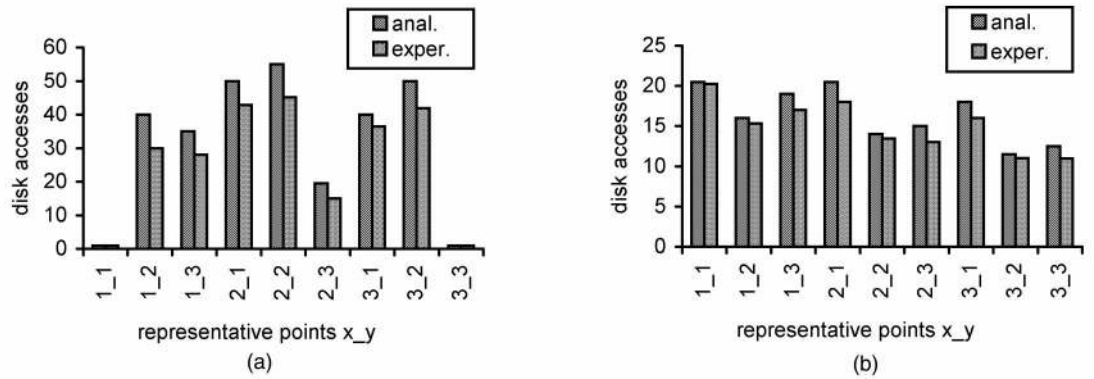


Fig. 10. Performance comparison for selection queries around representative points: range queries on (a) skewed data, (b) real data.

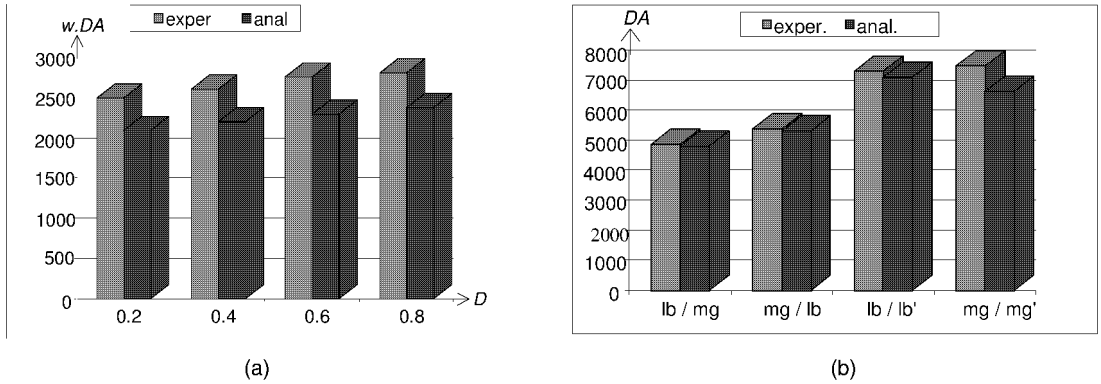


Fig. 11. Experimental vs. analytical *NA* and *DA* costs for join queries on (a) skewed and (b) real data.

uniform behavior, which is due to the different index heights.

5 Related Work

In the survey of this section, we present previous work on analytical performance studies for spatial queries using R-trees. Several findings from those proposals have been used as starting points for consequent studies and our analysis as well.

The earlier attempt to provide an analysis for R-tree-based structures appeared in [13]. Faloutsos et al. proposed a model that estimates the performance of R-trees and R^+-trees for selection queries, assuming, however, uniform distribution of data and *packed* trees (i.e., all the nodes of the tree are full of data). The formulae for the height of an R-tree as a function of its cardinality and fanout (5) and for the average size of a parent node as a function of the average size of the child nodes and the average distance between two consecutive child nodes' projections (8) were originally proposed in [13].

Later, Kamel and Faloutsos [19] and Pagel et al. [28] independently presented a formula (actually a variation of (2)) that calculates the average number of page accesses in an R-tree accessed by a query window as a function of the average node size and the query window size. Since, for practical query optimization purposes, R-tree properties such as the average node size cannot be assumed to be known in advance, that analysis is *qualitative*, i.e., it cannot provide an accurate cost prediction but rather presents the

TABLE 2
Average Relative Error for the Cost Estimation of Selection and Join Queries

Type of data set	point queries	range queries	join queries
Random	0%-10%	0%-5%	0%-10%
Skewed	0%-15%	0%-10%	0%-20%
Real	0%-15%	0%-20%	0%-20%

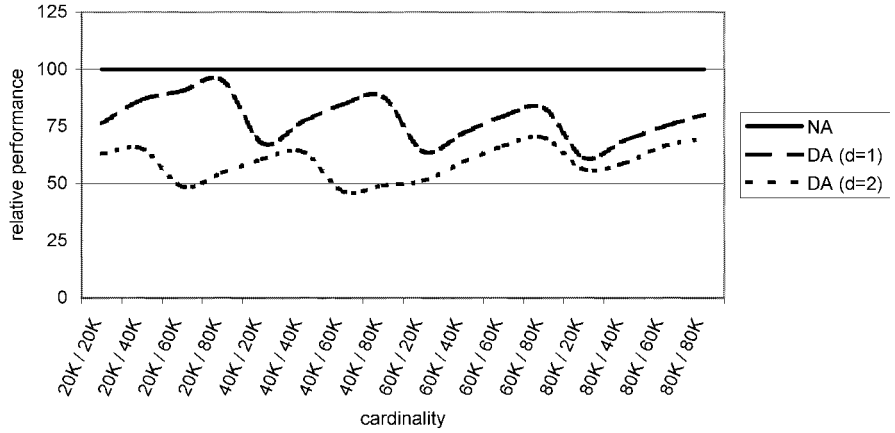


Fig. 12. Relative performance gain when using a path buffer: uniform-like data, (a) $d = 1$, (b) $d = 2$.

effect of three parameters, namely *area*, *perimeter*, and *number* of objects, on the R-tree performance. In those papers, the influence of the node perimeters was revealed, thus helping one to understand the efficiency of the R*-tree, which was the first R-tree variant to take the node perimeter into consideration during the index construction procedure [28]. Extending the work of [28], Pagel et al. [29] proposed an optimal algorithm that establishes a lower bound result for *static* R-tree performance. They also showed by experimental results that the best known static and dynamic R-tree variants, the *packed* R-tree [19] and the R*-tree respectively, perform about 10-20 percent worse than the lower bound. The impact of the three parameters (*area*, *perimeter*, and *number* of objects) was further discussed in [27], where performance formulae for various kinds of range queries, such as intersection, containment, and enclosure queries, were derived.

Faloutsos and Kamel [11] extended the previous formula to actually predict the number of disk accesses using a property of the data set, called the *fractal dimension*. The fractal dimension fd of a point data set can be mathematically computed and constitutes a simple way to describe nonuniform data sets by using just a single number. The formula constitutes the first attempt to model R-tree performance for nonuniform distributions of data (including the uniform distribution as a special case: $fd = d$) superceding the analysis in [13] that assumed uniformity. However, the model is applicable to point data sets only and, thus, cannot handle a number of spatial applications involving region data.

Since previous work focused on the number of nodes visited (NA is our analysis) as a metric of query performance, the effect of an underlying buffering mechanism has been neglected, although it is a real cost parameter in query optimization. Toward this direction, Leutenegger and Lopez [21] modified the cost formula of [19] introducing the size of an LRU buffer. Comparison results on different R-tree algorithms showed that the analytical estimations were very close to the experimental cost measures. A discussion of the appropriate number of R-tree levels to be pinned argued that pinning may mostly benefit point queries and, even then, only under special conditions.

Considering join queries, Aref and Samet [2] proposed analytical formulae for cost and selectivity, based on the R-tree analysis of [19]. The basic idea of [2] was the consideration of the one data set as the underlying database and the other data set as a source for query windows in order to estimate the cost of a spatial join query based on the cost of range queries. Experimental results showing the accuracy of the selectivity estimation formula were presented in that paper.

Huang et al. [17] recently proposed a cost model for spatial joins using R-trees. Independently of [38], it was the first attempt to provide an efficient formula for join performance by distinguishing two cases: considering either zero or nonzero buffer management. Using the analysis of [19], [28] as a starting point, it provides two formulae, one for each of the above cases. The efficiency of the proposed formulae was shown by comparing analytical estimations with experimental results for varying buffer size (with the relative error being around 10-20 percent). However, contrary to [38], that model assumes knowledge of R-tree properties in the same way that [19], [28] do.

Compared to related work, our model provides robust analytical formulae for selection and join cost estimation using R-trees, which:

1. do not need knowledge of the underlying R-tree structure(s) since they are only based on primitive data properties (cardinality N and density D of the data set), and
2. turn out to be accurate after a wide set of experimental results on both uniform-like and nonuniform data sets consisting of either point or nonpoint objects.

6 Conclusion

Selection and join queries are the fundamental operations supported by a DBMS. In the spatial database literature, there exist several techniques for the efficient processing of those operations, mainly based on the R-tree structure. However, for query optimization purposes, efficient cost models should be also available in order to make accurate cost estimations under various data distributions (either uniform or nonuniform ones).

In this paper, we presented a model that predicts the performance of R-tree-based structures for selection (point or range) queries and extended this model to support join queries as well. The proposed cost formulae are functions of data properties only, namely, *cardinality* and *density* in the workspace, and, therefore, can be used without any knowledge of the R-tree index properties. They are applicable to point or nonpoint data sets and, although they make use of the *uniformity assumption*, they are also adaptive to nonuniform distributions, which usually appear in real applications, by reducing its effect from global to local level (i.e., maintaining a *density surface* and assuming uniformity on a small subarea of the workspace).

Experimental results on both synthetic and real (TIGER/Line data [6]) data sets showed that the proposed analytical model is very accurate, with the relative error being usually around 10-15 percent when the analytical estimate is compared to cost measures using the R*-tree, perhaps the most efficient R-tree variant. In addition, for join query processing, a path buffer was considered and the analytical formula was adapted to support it. The performance saving due to the existence of such a buffering mechanism was highly affected by the sizes (and height) of the underlying indices and reached up to 50 percent for two-dimensional data sets. The proposed formulae and guidelines could be useful tools for spatial query processing and optimization purposes, especially when complex spatial queries are involved.

In this paper, we focused on the *overlap* operator between spatial objects. Any spatial operator could be used instead. For instance, one of the topological operators (*meet*, *covers*, *contains*, etc.) [8] or any of the 13^d possible directional operators between two d-dimensional objects [1], [30]. The idea could be handling such relations as range queries with an appropriate transformation of the query window. We have already adopted this idea for selection queries in order to estimate the cost of direction relations between two-dimensional objects in GIS applications [36] and combinations of topological and direction relations between three-dimensional (spatiotemporal) objects in large multimedia applications [42].

In a different direction, the work for (pairwise) join queries that appears in this paper has been extended to support multiway joins. Papadias et al. [25] provide a cost model for alternative multiway join algorithms, which is shown to be efficient for different types of join queries (chain, clique, etc.). Mamoulis and Papadias [22] also adopt the idea of the density surface, presented in Section 3.4, to estimate the size of a multiway join.

Acknowledgments

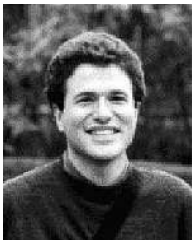
This research was partially supported by the EC funded project "CHOROCHRONOS: A Research Network for Spatiotemporal Database Systems" under the TMR program. The authors thank the reviewers for their detailed comments.

References

[1] J.F. Allen, "Maintaining Knowledge about Temporal Intervals," *Comm. ACM,* vol. 26, no. 11, pp. 832-843, 1983.
[2] W.G. Aref and H. Samet, "A Cost Model for Query Optimization Using R-Trees," *Proc. Second ACM Workshop Advances in GIS (ACM-GIS),* 1994.
[3] T. Brinkhoff, H.-P. Kriegel, and B. Seeger, "Efficient Processing of Spatial Joins Using R-trees," *Proc. ACM SIGMOD Conf. Management of Data,* 1993.
[4] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles," *Proc. ACM SIGMOD Conf. Management of Data,* 1990.
[5] T. Brinkhoff, H.-P. Kriegel, R. Schneider, and B. Seeger, "Multi-Step Processing of Spatial Joins," *Proc. ACM SIGMOD Conf. Management of Data,* 1994.
[6] Bureau of the Census, *TIGER/Line Census Files,* Mar. 1991.
[7] D. Comer, "The Ubiquitous B-Tree," *ACM Computing Surveys,* vol. 11, no. 2, pp. 121-137, June 1979.
[8] M.J. Egenhofer and R. Franzosa, "Point Set Topological Relations," *Int'l J. Geographical Information Systems,* vol. 5, pp. 161-174, 1991.
[9] C. Faloutsos, *Searching Multimedia Databases by Content.* Kluwer Academic, 1996.
[10] C. Faloutsos, H.V. Jagadish, and N. Sidiropoulos, "Recovering Information from Summary Data," *Proc. 23rd Int'l Conf. Very Large Data Bases (VLDB),* 1997.
[11] C. Faloutsos and I. Kamel, "Beyond Uniformity and Independence: Analysis of R-Trees Using the Concept of Fractal Dimension," *Proc. 13th ACM Symp. Principles of Database Systems (PODS),* 1994.
[12] M. Freeston, "The BANG File: A New Kind of Grid File," *Proc. ACM SIGMOD Conf. Management of Data,* 1987.
[13] C. Faloutsos, T. Sellis, and N. Roussopoulos, "Analysis of Object Oriented Spatial Access Methods," *Proc. ACM SIGMOD Conf. Management of Data,* 1987.
[14] V. Gaede and O. Guenther, "Multidimensional Access Methods," *ACM Computing Surveys,* vol. 30, no. 2, pp. 123-169, 1998.
[15] R.H. Gueting, "An Introduction to Spatial Database Systems," *VLDB J.,* vol.3, no. 4, pp. 357-399, Oct. 1994.
[16] A. Guttman, "R-Trees: A Dynamic Index Structure for Spatial Searching," *Proc. ACM SIGMOD Conf. Management of Data,* 1984.
[17] Y.-W. Huang, N. Jing, and E.A. Rundensteiner, "A Cost Model for Estimating the Performance of Spatial Joins Using R-trees," *Proc. Ninth Int'l Conf. Scientific and Statistical Database Management (SSDBM),* 1997.
[18] A. Henrich, H.-W. Six, and P. Widmayer, "The LSD Tree: Spatial Access to Multidimensional Point and Non Point Objects," *Proc. 15th Int'l Conf. Very Large Data Bases (VLDB),* 1989.
[19] I. Kamel and C. Faloutsos, "On Packing R-Trees," *Proc. Second Int'l Conf. Information and Knowledge Management (CIKM),* 1993.
[20] D. Knuth, *The Art of Computer Programming, vol. 3: Sorting and Searching.* Addison-Wesley, 1973.

[21] S.T. Leutenegger and M.A. Lopez, "The Effect of Buffering on the Performance of R-Trees," *Proc. 14th IEEE Int'l Conf. Data Eng. (ICDE),* 1998.
[22] N. Mamoulis and D. Papadias, "Integration of Spatial Join Algorithms for Processing Multiple Inputs," *Proc. ACM SIGMOD Conf. Management of Data,* 1999.
[23] J. Orenstein, "Redundancy in Spatial Databases," *Proc. ACM SIGMOD Conf. Management of Data,* 1989.
[24] V. Poosala and Y. Ioannidis, "Selectivity Estimation without the Attribute Value Independence Assumption," *Proc. 23rd Int'l Conf. Very Large Data Bases (VLDB),* 1997.
[25] D. Papadias, N. Mamoulis, and Y. Theodoridis, "Processing and Optimization of Multi-Way Spatial Joins Using R-Trees," *Proc. 18th ACM Symp. Principles of Database Systems (PODS),* 1999.
[26] F.P. Preparata and M.I. Shamos, *Computational Geometry.* Springer-Verlag, 1985.
[27] B.-U. Pagel and H.-W. Six, "Are Window Queries Representative for Arbitrary Range Queries?" *Proc. 15th ACM Symp. Principles of Database Systems (PODS),* 1996.
[28] B.-U. Pagel, H.-W. Six, H. Toben, and P. Widmayer, "Towards an Analysis of Range Query Performance," *Proc. 12th ACM Symp. Principles of Database Systems (PODS),* 1993.
[29] B.-U. Pagel, H.-W. Six, and M. Winter, "Window Query-Optimal Clustering of Spatial Objects," *Proc. 14th ACM Symp. Principles of Database Systems (PODS),* 1995.
[30] D. Papadias and Y. Theodoridis, "Spatial Relations, Minimum Bounding Rectangles, and Spatial Data Structures," *Int'l J. Geographic Information Science,* vol. 11, no. 2, pp. 111-138, 1997.
[31] N. Roussopoulos, Y. Kotidis, and M. Roussopoulos, "Cubetree: Organization of and Bulk Updates on the Data Cube," *Proc. ACM SIGMOD Conf. Management of Data,* 1997.
[32] N. Roussopoulos and D. Leifker, "Direct Spatial Search on Pictorial Databases Using Packed R-trees," *Proc. ACM SIGMOD Conf. Management of Data,* 1985.
[33] H. Samet, *The Design and Analysis of Spatial Data Structures.* Addison-Wesley, 1990.
[34] M. Stonebraker and D. Moore, *Object Relational Databases: The Next Wave.* Morgan Kaufmann, 1996.
[35] T. Sellis, N. Roussopoulos, and C. Faloutsos, "The R^+-Tree: A Dynamic Index for Multidimensional Objects," *Proc. 13th Int'l Conf. Very Large Data Bases (VLDB),* 1987.
[36] Y. Theodoridis, D. Papadias, E. Stefanakis, and T. Sellis, "Direction Relations and Two-Dimensional Range Queries: Optimisation Techniques," *Data and Knowledge Eng.,* vol. 27, no. 3, pp. 313-336, Sept. 1998.
[37] Y. Theodoridis and T. Sellis, "A Model for the Prediction of R-tree Performance," *Proc. 15th ACM Symp. Principles of Database Systems (PODS),* 1996.
[38] Y. Theodoridis, E. Stefanakis, and T. Sellis, "Cost Models for Join Queries in Spatial Databases," *Proc. 14th IEEE Int'l Conf. Data Eng. (ICDE),* 1998.
[39] Y. Theodoridis, E. Stefanakis, and T. Sellis, "Efficient Cost Models for Spatial Queries Using R-trees," Chorochronos Technical Report Series, CH-98-05, July 1998, http://www.dbnet.ntua.gr/~choros/publications.html.
[40] J.S. Vitter, "Faster Methods for Random Sampling," *Comm. ACM,* vol. 27, no. 7, pp. 703-718, July 1984.
[41] J.S. Vitter, "Random Sampling with Reservoir," *ACM Trans. Math. Software,* vol. 11, pp. 37-57, Mar. 1985.
[42] M. Vazirgiannis, Y. Theodoridis, and T. Sellis, "Spatio-Temporal Composition and Indexing for Large Multimedia Applications," *ACM/Springer Multimedia Systems,* vol. 6, no. 4, pp. 284-298, July 1998.

Yannis Theodoridis received a BEng (1990) in electrical engineering and a PhD (1996) in electrical and computer engineering, both from the National Technical University of Athens. Currently, he is a senior researcher at the Computer Technology Institute (CTI) in Patras, Greece. He has published more than 20 papers in refereed scientific journals and conference proceedings, including *Algorithmica*, *ACM Multimedia Systems Journal*, ACM SIGMOD/PODS Conference, and IEEE Conference on Data Engineering. His research interests include spatial and spatiotemporal databases, multimedia systems, access methods, query optimization, and benchmarking. He is a member of the IEEE.



Emmanuel Stefanakis obtained the PhD degree in computer science from the Department of Electrical and Computer Engineering, National Technical University of Athens, Greece, in 1997; the MScE degree from the Department of Geodesy and Geomatics Engineering, University of New Brunswick, Canada, in 1994; and the DiplEng degree (with distinction) from the Department of Rural and Surveying Engineering, National Technical University of Athens, Greece, in 1992. He is currently a postdoctoral researcher in the Knowledge and Database Systems Laboratory at the National Technical University of Athens, Greece. He has received several scholarships and awards in Greece and Canada and has published several times in international journals, books, and conferences. Since 1991, he has been involved in several research projects dealing with database and GIS applications. His research interests include database systems, geographic information systems, and digital mapping.

Timos Sellis received the BSc degree in electrical engineering in 1982 from the National Technical University of Athens, Athens, Greece. In 1983, he received the MSc degree from Harvard University and, in 1986, the PhD degree from the University of California at Berkeley, where he was a member of the INGRES group, both in computer science. In 1986, he joined the Department of Computer Science of the University of Maryland, College Park, as an assistant professor, and became an associate professor in 1992. In 1992, he became an associate professor in the Computer Science Division of the National Technical University of Athens (NTUA), in Athens, Greece, where he is currently a full professor. Dr. Sellis is also the head of the Knowledge and Database Systems Laboratory at NTUA. His research interests include extended relational database systems, data warehouses, and spatial, image, and multimedia database systems. He has published more than 100 articles in refereed journals and international conferences in the above areas. Dr. Sellis is a recipient of a Presidential Young Investigator (PYI) award for 1990-1995 and of the VLDB 1997 10 Year Paper Award together with N. Roussopoulos and C. Faloutsos. He is a member of the editorial boards of the *International Journal on Intelligent Information Systems: Integrating Artificial Intelligence and Database Technologies* and *Geoinformatica*. He is a member of the IEEE.