Direction Relations and Two-Dimensional Range Queries: Optimization Techniques

Yannis Theodoridis[*] Dimitris Papadias[+] Emmanuel Stefanakis[*] Timos Sellis[*]

[*]Computer Science Division
Department of Electrical and Computer Engineering
National Technical University of Athens
Zographou, Athens, GREECE 15773
{theodor, stefanak, timos}@theseas.ntua.gr

[+] National Center for Geographic Information and Analysis &
Department of Spatial Information Science and Engineering
University of Maine
Orono, ME 04469-5711, USA
dimitris@spatial.maine.edu

Abstract

Despite the attention that *direction relations*, such as east, southeast etc., have attracted in several domains related to Spatial Databases and Geographic Information Systems (GIS), little work has been done on their formalization and efficient processing. In this paper we define direction relations between two-dimensional objects in different levels of qualitative resolution and we show how these relations can be efficiently retrieved in existing DBMSs using B- , KDB- and R- tree-based data structures. Since query processing involving direction relations maps into *range queries in two-dimensional space*, our work essentially studies optimization techniques for 2D ranges. We test the efficiency of alternative indexing methods through extensive experimentation and we present analytical models that estimate their performance. The analytical estimates are proved to be very close to the actual results and can be used by spatial query optimizers in order to decide the appropriate data structure for each type of range query. Although for our experiments we define a small set of representative relations motivated by geographic applications, the results of this paper are directly applicable to any type of direction relations (or range queries in general) that may arise in different application domains.

Keywords: Spatial Databases, Direction Relations, Range Queries, Data Structures, Performance Analysis, Query Optimisation.

A short version of this paper has been submitted to the 12th IEEE International Conference on Data Engineering [Theo95a]. Some parts of the performance analysis appear in the Proceedings of the 2nd Conference on Spatial Information Theory [Theo95c].

Yannis Theodoridis and Timos Sellis were partially supported by the Department of Research and Technology of Greece (PENED'91) and by the European Commission under the ESPRIT Basic Research project AMUSING (EP6881).
Dimitris Papadias was partially supported by NSF - IRI 9221276.

1. INTRODUCTION

The most common types of spatial relations in geographic applications include *direction* relations which describe order in space (e.g., south, northwest), *topological* relations that describe concepts of neighbourhood and incidence (e.g., overlap, disjoint), and *distance* relations (e.g., near, far). The above types of relations have been used in a wide range of topics, such as Spatial Query Languages [Papa95a], Reasoning [Egen91] and Consistency Checking Mechanisms [Grig95].

This paper describes implementations of direction relations in Spatial Database Management Systems (DBMS) and Geographical Information Systems (GIS). Despite the fact that direction relations constitute an important class of user queries, they have not been studied extensively in spatial access methods. Query processing and optimization techniques have mainly focused on window queries [Gutt84], topological relations of high resolution [Papa95b] and nearest neighbour queries [Rous95]. The main reason for this, is the lack of well-defined direction relations between actual objects.

In Geography, for example, although the linguistic terms used to describe the direction relation between some pairs of countries are undisputed (everybody will agree that Germany is *north* of Italy) this is not always the case. Consider, for instance, the map of Europe illustrated in Figure 1, and the query "find all countries *north* of Italy". Should France belong to the result, or is it *northwest* (but not *north*) of Italy? The answer to the above query depends on the definition of direction relations which may vary for each application.

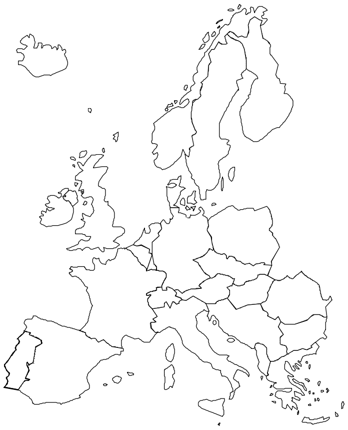


Fig. 1 Map of Europe

In this paper we define direction relations between two-dimensional objects in different levels of qualitative resolution to match the application needs. Our work extends previous attempts to formalise direction relations which have concentrated on point objects [Fran92, Papa94a] or Minimum Bounding Rectangles [Peuq87]. Then we show how the relations that we define can be transformed into *range queries* and retrieved in spatial DBMSs using different indexing methods. Essentially we compare the efficiency of indexing methods in two-dimensional range queries in the context of geographic applications. We also propose modifications of the traditional (spatial) data structures that yield improved performance for certain classes of queries.

The results of this paper are directly applicable to Spatial Databases and GISs where the formalization of spatial relations is crucial for user interfaces and query optimisation strategies. Although we deal with geographic examples, potential applications for direction relations (and 2D range queries in general) include other domains such as CAD and VLSI design. In these domains, the queries can be very similar, but the linguistic terms used to express them may vary (e.g., *above* instead of *north*).

The paper is organised as follows. In section 2 we define several direction relations between objects in 2D-space and describe how object approximations, commonly used in spatial access methods, can be used for the efficient retrieval of direction relations. In section 3 we discuss the retrieval of direction relations and 2D ranges using alternative access methods (B-trees, KDB-trees and R-trees) and we provide analytical formulas for their expected performance. Section 4 tests the different implementations and compares analytical predictions with the actual results; the relative performance of the indexing methods is also discussed. Section 5 is concerned with modifications of indexing methods that increase performance for some queries, and section 6 concludes with comments on future work.

2. DIRECTION RELATIONS AS RANGE QUERIES

Unlike the case of topological relations where there seems to exist a set of widely accepted relations [Egen91], there are no such definitions of direction relations. In order to define direction relations between extended objects we first define relations between points and apply these definitions using universally and existentially quantified formulas. We assume an absolute frame of reference and a pair of orthocanonic axes x and y. Our method is *projection based,* that is, direction relations are defined using projection lines vertical to the coordinate axes. An alternative approach is based on the *cone-shaped* concept of direction, i.e., direction relations are defined using angular regions between objects. Extensive studies of the two approaches can be found in [Fran95, Hern94].

For the rest of the paper, *p* denotes the *primary object* (the object to be located) and *q* the *reference object* (the object in relation to which the primary object is located-in the following examples the reference object is grey). Let p_i be a point of object *p* and q_j be a point of object *q*; p_{i-x} is the x- coordinate of point p_i, p_{i-y} is the y- coordinate of point p_i, and so on. If we draw projection lines from a reference point, the plane is divided into nine partitions (Figure 2), each corresponding to one direction relation. The symbol * in Figure 2 illustrates the reference point (it also corresponds to the relation *same_position*).

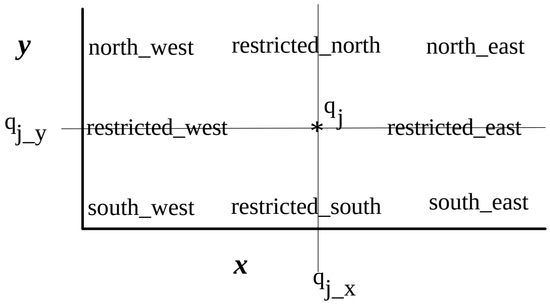


Fig. 2 Plane partitions using one point per object

Some of the definitions for the relations of Figure 2 are given below, while the rest can be defined in the same way:

$\mathit{north\_west}(p_i,q_j) \equiv (p_{i\_x}<q_{j\_x}) \wedge (p_{i\_y}>q_{j\_y})$, $\mathit{restricted\_north}(p_i,q_j) \equiv (p_{i\_x}=q_{j\_x}) \wedge (p_{i\_y}>q_{j\_y})$,

$\mathit{north\_east}(p_i,q_j) \equiv (p_{i\_x}>q_{j\_x}) \wedge (p_{i\_y}>q_{j\_y})$, $\mathit{restricted\_west}(p_i,q_j) \equiv (p_{i\_x}<q_{j\_x}) \wedge (p_{i\_y}=q_{j\_y})$,

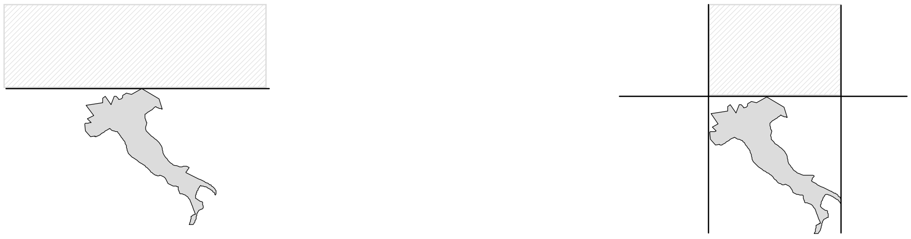
$\mathit{same\_position}(p_i,q_j) \equiv (p_{i\_x}=q_{j\_x}) \wedge (p_{i\_y}=q_{j\_y})$.

The above nine relations (including *same_position*) correspond to the *highest resolution* that we can achieve when reasoning in terms of points using the concept of projections, in the sense that they cannot be represented as disjunctions of other relations. Exactly one of the previous relations holds true between any pair of points. The relations are transitive and *same_position* is also symmetric. The rest form four pairs of converse relations (e.g. $\mathit{north\_east}(p_i,q_j) \Leftrightarrow \mathit{south\_west}(q_j, p_i)$). Additional direction relations can be defined using disjunctions of the nine relations; for example:

$\mathit{north}(p_i,q_j) \equiv \mathit{north\_west}(p_i,q_j) \vee \mathit{restricted\_north}(p_i,q_j) \vee \mathit{north\_east}(p_i,q_j)$,

$\mathit{south}(p_i,q_j) \equiv \mathit{south\_west}(p_i,q_j) \vee \mathit{restricted\_south}(p_i,q_j) \vee \mathit{south\_east}(p_i,q_j)$,

Using the above definitions between points we can define spatial relations between objects. The relation *strong_north* between objects *p* and *q*, for instance, denotes that all points of *p* are *north* of all points of *q*: $\mathit{strong\_north}(p,q) \equiv \forall p_i \; \forall q_j \; \mathrm{north}(p_i,q_j)$, that is all points of the primary object must be in the shaded area (acceptance area) of Figure 3a. The relation *strong_north* can be characterised as low resolution relation because its acceptance area is large. On the other hand, we can define a higher resolution version of *strong_north*, as : $\mathit{strong\_bounded\_north}(p,q) \equiv \forall p_i \forall q_j \; \mathit{north}(p_i, q_j) \wedge \forall p_i \exists q_j \; \mathit{north\_east}(p_i, q_j) \wedge \forall p_i \; \exists q_j \; \mathit{north\_west}(p_i, q_l)$. According to this relation, all points of object *p* must be in the region bounded by the horizontal line that passes from *q*' snorthmost point and by the vertical lines that also bound *q*. For example, although Belgium is *strong_north* of Italy, it does not satisfy the relation *strong_bounded_north* because it is not in the shaded area of Figure 3b.



(a)

(b)

Fig. 3 Acceptance area for *strong_north* and *strong_bounded_north* relations

All relations that we define in this paper concern the direction *north*, and they are representative for other relations as well, in the sense that they refer to several levels of qualitative resolution. Depending on the application needs, a large number of direction relations between extended objects can be defined and implemented accordingly. The set of direction relations can be chosen so that several properties are satisfied: the relations can be pairwise disjoint, provide a complete coverage, form a relation algebra etc.

Notice that the relations that we study here do not satisfy any of these properties, since our goal is to show how direction relations of different resolution can be defined and retrieved in spatial data structures. Customised direction relations between extended objects that satisfy the above properties and correspond to certain application needs are straightforward to define by comparing point coordinates.

Although the previous discussion refers to actual two-dimensional objects, usually spatial access methods use approximations to efficiently retrieve candidates that could satisfy a query. In this paper we examine methods based on the traditional approximation of Minimum Bounding Rectangles (MBRs). MBRs are the most common approximations in spatial applications because they need only two points for their representation; in particular, each object q is represented as an ordered pair (q'_l,q'_u) of *representative points* that correspond to the lower left (q'_l) and the upper right point (q'_u) of the MBR q' that covers q. Figure 4 illustrates how the map of Figure 1 can be approximated by MBRs.

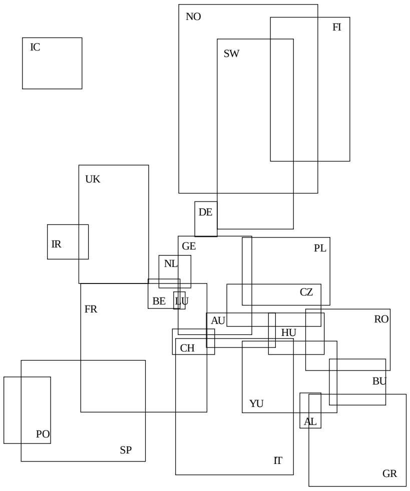


Fig. 4 MBR approximations for the map of Europe

Some access methods (R-trees) explicitly store MBRs, while others (B-trees and KDB-trees) compare object locations using representative points. In order to answer the query "find all objects p that satisfy the relation R with respect to an object q" we have to retrieve all MBRs p' that satisfy a set of *range constraints* R' with respect to the MBR q' of object q. Table 1 illustrates the direction relations on which we concentrate on this paper, and a mapping from direction relations R between actual objects to constraints R' between MBR representative points.

According to the last column of Table 1, each direction relation is transformed to a set of 1 up to 4 constraints that specify a range of the 2D space where the representative points of candidate MBRs should belong.

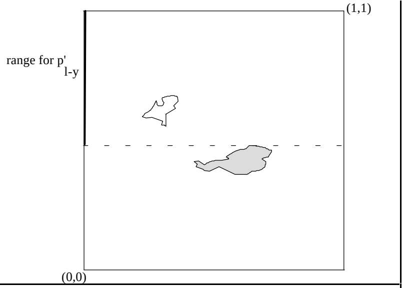
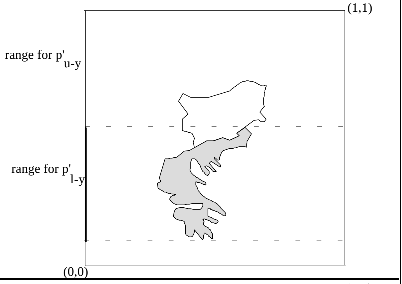
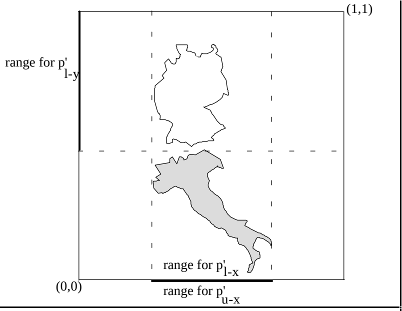
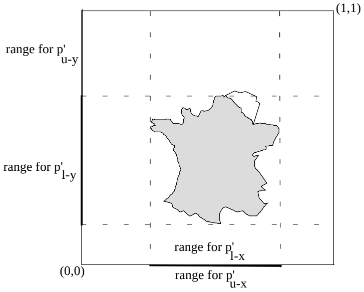
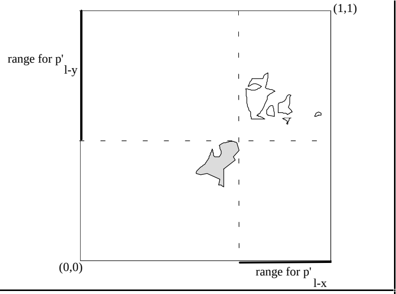
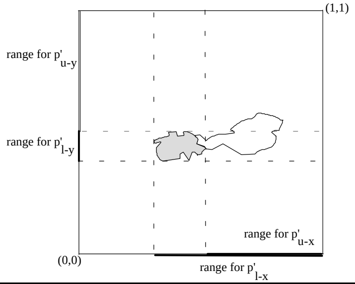
Relation between objects	**Example**	**Range Constraints for MBRs p' to be Retrieved**
*s*trong_*n*orth(p,q) = $\forall p_i\ \forall q_j$ north(p_i, q_j) example: *sn*(NL,AU)		1 constraint on y axis: $p'_{l\text{-}y} > q'_{u\text{-}y}$
*w*eak_*n*orth(p,q) = $\exists p_i \forall q_j$ north(p_i,q_j) ∧ $\forall p_i \exists q_j$ north(p_i,q_j) ∧ $\exists p_i \exists q_i$ south(p_i,q_i) example: *wn*(BU,GR)		2 constraints on y axis: $(q'_{l\text{-}y} < p'_{l\text{-}y} < q'_{u\text{-}y})$ $(p'_{u\text{-}y} > q'_{u\text{-}y})$
*s*trong_*b*ounded_*n*orth(p,q)= $\forall p_i \forall q_j$ north(p_i, q_j) ∧ $\forall p_i \exists q_j$ north_east(p_i, q_j) ∧ $\forall p_i\ \exists q_i$ north_west(p_i, q_i) example: *sbn*(GE,IT)	 (1,1) range for p'l-y range for p'l-x (0,0) range for p'u-x	1 constraint on y axis: $(p'_{l\text{-}y} > q'_{u\text{-}y})$ 2 constraints on x axis: $(q'_{l\text{-}x} < p'_{l\text{-}x} < q'_{u\text{-}x})$ $(q'_{l\text{-}x} < p'_{u\text{-}x} < q'_{u\text{-}x})$
*w*eak_*b*ounded_*n*orth(p,q)= $\exists p_i\ \forall q_j$ north(p_i, q_j) ∧ $\exists p_i\ \exists q_j$ south(p_i, q_j) ∧ $\forall p_i\ \exists q_j$ north_east(p_i, q_j) ∧ $\forall p_i \exists q_i$ north_west(p_i, q_i) example: *wbn*(BE,FR)	 (1,1) range for p'u-y range for p'l-y range for p'l-x (0,0) range for p'u-x	2 constraints on y axis: $(q'_{l\text{-}y} < p'_{l\text{-}y} < q'_{u\text{-}y})$ $(p'_{u\text{-}y} > q'_{u\text{-}y})$ 2 constraints on x axis: $(q'_{l\text{-}x} < p'_{l\text{-}x} < q'_{u\text{-}x})$ $(q'_{l\text{-}x} < p'_{u\text{-}x} < q'_{u\text{-}x})$
*s*trong_*n*orth_*e*ast(p,q)= $\forall p_i\ \forall q_i$ north_east(p_i, q_i) example: *sne*(DE,NL)		1 constraint on y axis: $(p'_{l\text{-}y} > q'_{u\text{-}y})$ 1 constraint on x axis: $(p'_{l\text{-}x} > q'_{u\text{-}x})$
*w*eak_*n*orth_*e*ast(p,q)= $\exists p_i \forall q_j$ north_east(p_i,q_j) ∧ $\exists p_i \exists q_j$south(p_i,q_j) ∧ $\forall p_i \exists q_i$north_east(p_i,q_i) example *wne*(AU,CH)	 (1,1) range for p'u-y range for p'l-y range for p'u-x (0,0) range for p'l-x	2 constraints on y axis: $(q'_{l\text{-}y} < p'_{l\text{-}y} < q'_{u\text{-}y})$ $(p'_{u\text{-}y} > q'_{u\text{-}y})$ 2 constraints on x axis: $(p'_{l\text{-}x} > q'_{l\text{-}x})$ $(p'_{u\text{-}x} > q'_{u\text{-}x})$

Table 1 Direction relations between objects and mapping to range constraints between MBRs

In particular the four potential ranges are:

1. Range of p'_l on axis y (e.g., $p'_{l\text{-}y} > q'_{u\text{-}y}$)
2. Range of p'_l on axis x (e.g., $q'_{l\text{-}x} < p'_{l\text{-}x} < q'_{u\text{-}x}$)
3. Range of p'_u on axis y (e.g., $p'_{u\text{-}y} > q'_{u\text{-}y}$)
4. Range of p'_u on axis x (e.g., $p'_{u\text{-}x} > q'_{u\text{-}x}$)

Some of the ranges are left unspecified for some direction relations. The relations are chosen in a way that the number of constraints and the axes involved differ. The performance of each indexing method is expected to be similar for relations with similar range constraints. Therefore the behaviour of an indexing method to any spatial relation can be predicted from other relations with similar range characteristics.

Because MBRs differ from the actual objects they enclose, they are not always adequate to express the relation between the actual objects. For this reason, spatial queries involve the following two step strategy: First a *filter step* based on MBRs is used to rapidly eliminate MBRs of objects that could not possibly satisfy the query and select a set of potential candidates. Then during a *refinement step* each candidate is examined (by using computational geometry techniques) and false hits are detected and eliminated. Unlike topological relations, where the refinement step is the rule [Papa95b], the only direction relations of Table 1 that need a refinement step are *weak_bounded_north* and *weak_north_east*. For the rest of the relations all retrieved MBRs correspond to objects that satisfy the query. Details about the retrieval of direction relations using MBRs can be found in [Papa94b]. In the next sections we will show how the above results can be applied to indexing techniques available in commercial DBMSs.

3. RETRIEVAL OF DIRECTION RELATIONS

The retrieval of direction relations in existing DBMSs could be accomplished by maintaining traditional indexes (e.g. B-trees), or, alternatively, by incorporating Abstract Data Types (ADTs) with specialised indexes defined by external code (e.g. KDB-trees or R-trees). Application developers can decide which is the most appropriate for their application needs. In the rest of the section we describe how B-, KDB- and R-trees can be used to retrieve direction relations and we provide analytical formulas that estimate the performance of the three methods. Existing formulas for the expected performance of the above structures focus on traditional retrieval (match or range queries on B-trees and KDB-trees, and overlap queries on R-trees). Our work extends previous work and estimates the expected cost (i.e., number of disk accesses) for the retrieval of direction relations and the corresponding range queries. In the following discussion we assume that both data (primary) and query (reference) rectangles follow a random (uniform-like) distribution over the unit square address space.

3.1. Retrieval of Direction Relations using B-trees

The first solution for the retrieval of direction relations includes the maintenance of a group of 4 alphanumeric indexes, such as B-trees. In particular, the implementation used in existing systems is the B^+-tree index [Knut73] and, in this paper, we will think of B^+-trees when we use the term B-trees. Each index corresponds to one of the four numbers: p'_{l-x}, p'_{l-y}, p'_{u-x}, p'_{u-y}, where p'_{l-x} stands for the x-coordinate of the lower point of p', p'_{l-y} for the y-coordinate of the lower point, and so on. Obviously, some relations imply search on one B-tree while others imply search on more B-trees.

In general, the processing of a query of the form "find all objects *p* that satisfy a given direction relation with respect to object *q*" using B-trees involves the following steps:

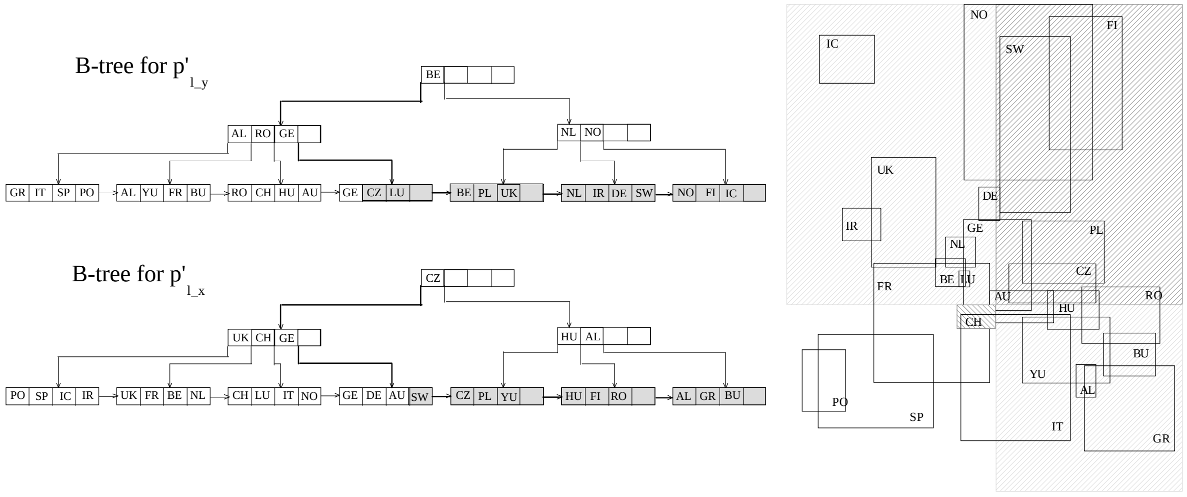
Step 1. Depending on the relation to be retrieved, select the B-trees involved from the set of four indexes. This procedure uses the last column of Table 1 to determine the appropriate B-trees to be searched.

Step 2. Search each index involved to find the corresponding answer sets.

Step 3. If more than one index is involved, find the intersection set. A "realistic" assumption is that this procedure is executed in main memory.

Step 4. If necessary, follow a refinement step for the selected object IDs.

As an example, consider the query: "Find the countries *p strong_north_east* of Switzerland (CH)". In this case, two B-trees, for p'_{l-y} and p'_{l-x} parameters, need to be searched, because only p'_{l-y} and p'_{l-x} are involved in the range constraints for *strong_north_east*. The two trees are illustrated in Figure 5a[1] where each label indicates the appropriate coordinate for the corresponding object *p*. The sets {CZ' ,LU' ,BE' ,PL' , UK' ,NL' ,IR' ,DE' ,SW' ,NO' ,FI' ,IC' }and {SW' ,CZ' ,PL' ,YU' ,HU' ,FI' ,RO' ,AL' ,GR' ,BU} are the two answer sets. The intersection set {CZ' ,PL' ,SW' ,FI' }contains the object IDs that satisfy the query (illustrated as the dark shaded area in Figure 5b). A refinement step is not needed for the retrieval of *strong_north_east*, that is, all retrieved MBRs correspond to objects that satisfy the query.



(a) (b)

Fig. 5 Retrieval of relation *strong_north_east* using B-trees

In order to provide analytical formulas for the performance of each direction relation we will use Table 1. The constraints of the last column of Table 1 can be characterised by a range *r*, $0 \le r \le 1$. We consider, for example, the constraint $q'_{l-y}<p'_{l-y}<q'_{u-y}$ as a query with range $r=q'_{u-y}-q'_{l-y}$. If we suppose that the data keys

[1] For illustration reasons, in the examples of the tree structures hereafter we assume a branching factor of 4, i.e., each node contains at most four entries.

are stored in a B-tree index of height *h* with *L* leaf nodes then the average cost *C(r)* for the retrieval of a constraint with range *r* is [Come79]

$$C(r) = h + r \cdot L \qquad (1)$$

In order to compute the expected cost *C(r)* for the retrieval of a constraint characterised by a range *r* we need to provide equations for the parameters of Eq. 1, namely *h, L, r.* Suppose now that *m* is the maximum number of entries in a B-tree node, *c* is the average capacity of a node, and *N* is the total number of keys stored in the leaf nodes. We have the following equations [Bato81] for the average *h* and *L* values in order to use them in Eq. 1:

$$h = 1 + \left\lceil \log_{c \cdot m} \frac{N}{m} \right\rceil \qquad (2)$$

$$L = N \big/ c \cdot m \qquad (3)$$

What remains in order to have a complete expression for Eq. 1 is the value of parameter *r* which depends on the particular constraint. If we assume that the size of a data object MBR is $p_x \bullet p_y$ and the size of a query object MBR is $q_x \bullet q_y$ then we can provide (Table 2) the values of parameters *r* according to possible range constraints (the constraints refer only to x- coordinate since constraints for y- coordinate can be expressed in a similar way).

Constraint	**Average range r**
$p'_{l\text{-}x} < q'_{l\text{-}x}$ or $p'_{u\text{-}x} > q'_{u\text{-}x}$ or $p'_{l\text{-}x} > q'_{l\text{-}x}$ or $p'_{u\text{-}x} < q'_{u\text{-}x}$	$r = (1 - q_x) / 2$
$q'_{l\text{-}x} < p'_{l\text{-}x} < q'_{u\text{-}x}$ or $q'_{l\text{-}x} < p'_{u\text{-}x} < q'_{u\text{-}x}$	$r = q_x$
$p'_{l\text{-}x} > q'_{u\text{-}x}$ or $p'_{u\text{-}x} < q'_{l\text{-}x}$	$r = (1 - (2 \bullet p_x + q_x)) / 2$
$p'_{l\text{-}x} < q'_{u\text{-}x}$ or $p'_{u\text{-}x} > q'_{l\text{-}x}$	$r = (1 + q_x) / 2$

Table 2 Average values for range *r* of a constraint

Using information from Table 2 and Eq. 2 and 3 we can estimate the expected cost for each constraint (Eq. 1) and, summing up, the expected cost *C(R,k)* for the retrieval of a spatial relation *R* with *k* constraints (the value of *k* is between 1 and 4), i.e.,

$$C(R,k) = \sum_{i=1}^{k} C(r_i) = \sum_{i=1}^{k} (h + r_i \cdot L) \qquad (4)$$

The performance of the retrieval mechanism using B-trees depends significantly on the particular direction relation because the number of B-trees to be searched is equal to the number of range constraints that are involved in the definition of the relation. *Strong_north*, for example, involves only one constraint ($p'_{l\text{-}y} > q'_{u\text{-}y}$), while *weak_bounded_north* contains four constraints ($q'_{l\text{-}y} < p'_{l\text{-}y} < q'_{u\text{-}y}$) $\wedge$ ($p'_{u\text{-}y} > q'_{u\text{-}y}$) $\wedge$ ($q'_{l\text{-}x} < p'_{l\text{-}x} < q'_{u\text{-}x}$) $\wedge$ ($q'_{l\text{-}x} < p'_{u\text{-}x} < q'_{u\text{-}x}$). As it will be shown in section 4 this fact makes *strong_north* much more efficient than *weak_bounded_north* to process.

3.2. Retrieval of Direction Relations using KDB-trees

The second solution for the retrieval of direction relations includes the maintenance of a group of 2 two-dimensional point indexes, such as KDB-trees [Robi81]. The two indexes correspond to the MBR representative points p'_l and p'_u. KDB-trees, like B-trees, are multiway, height-balanced trees, which consist of leaf and intermediate nodes. Point data are stored in leaf nodes and intermediate nodes partition the space in *disjoint* regions which totally contain the corresponding entries. In order to retrieve a direction relation using a KDB-tree we have to specify the appropriate query window that contains the answer set and recursively search the tree nodes that intersect the query window. Table 3 presents the corresponding query windows for each direction relation, assuming the unit segment [0,1] as global space per axis. Query windows Q_l and Q_u correspond to the KDB-trees for the lower and upper MBR corner points, respectively, and the format is a set of four values that correspond to the four coordinates (Q_{l-x}, Q_{l-y}, Q_{u-x}, Q_{u-y}) of Q (Q_l or Q_u).

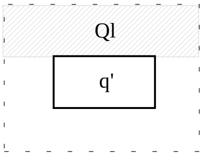
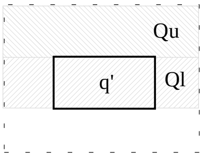
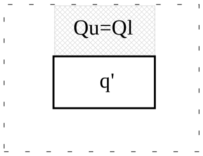
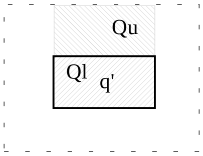
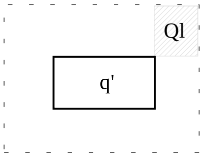
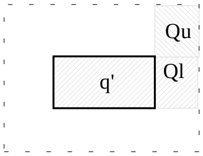
Relation	**Range query** Q_l	**Range query** Q_u	**Representation**
strong_north(p,q)	$(0, q'_{u-y}, 1, 1)$		
weak_north(p,q)	$(0, q'_{l-y}, 1, q'_{u-y})$	$(0, q'_{u-y}, 1, 1)$	
strong_bounded_north(p,q)	$(q'_{l-x}, q'_{u-y}, q'_{u-x}, 1)$	$(q'_{l-x}, q'_{u-y}, q'_{u-x}, 1)$	
weak_bounded_north(p,q)	$(q'_{l-x}, q'_{l-y}, q'_{u-x}, q'_{u-y})$	$(q'_{l-x}, q'_{u-y}, q'_{u-x}, 1)$	
strong_north_east(p,q)	$(q'_{u-x}, q'_{u-y}, 1, 1)$		
weak_north_east(p,q)	$(q'_{l-x}, q'_{l-y}, 1, q'_{u-y})$	$(q'_{u-x}, q'_{u-y}, 1, 1)$	

Table 3 Query windows for the retrieval of direction relations using KDB-trees

The relations that involve only one of the two representative points (i.e., only one range query, such as *strong_north* and *strong_north_east*) imply search on one KDB-tree while the rest imply search on both KDB-trees and the computation of the intersection.

In general, the processing of a query of the form "find all objects *p* that satisfy a given direction relation with respect to object *q*" using KDB-trees involves four steps similar to the ones for B-trees:

Step 1. Depending on the relation to be retrieved, select the KDB-trees involved from the set of two indexes. This procedure involves Table 3.

Step 2. Search each index involved to find the corresponding answer sets.

Step 3. If both indexes are involved, find the intersection set.

Step 4. If necessary, follow a refinement step for the selected object IDs.

Consider again the query: "Find the countries *p strong_north_east* of Switzerland (CH)". In this case, only one KDB-tree (for p'_l data point) needs to be searched. It is illustrated in Figure 6a where each label indicates the lower left point for each object *p*. The intermediate nodes that are selected for propagation are nodes X and Y (1st level), D, F and G (2nd level) i.e., the ones that intersect the shaded area of Figure 6b (these nodes are grey in the tree structure of Figure 6a). Among the leaves of the intermediate nodes D, F and G, the entries that lie within the shaded area (i.e., satisfy the range query) are CZ' , PL' , SW' and FI' .

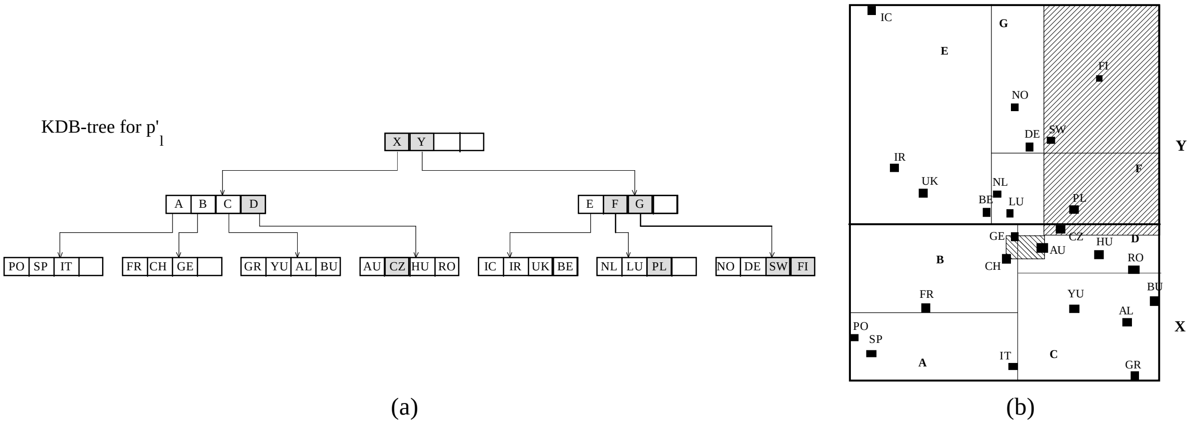


Fig. 6 Retrieval of relation *strong_north_east* using KDB-trees

Currently, there do not exist analytical models that estimate the retrieval cost of KDB-trees. Robinson [Robi81] estimates that the expected cost of a window query with size Q should be, where P is the total number of nodes (pages) of the tree structure and the global space is assumed to be the unit square [0,1]. However, this estimation is oversimplified. In order to facilitate cost analysis, we consider a KDB-tree as a simple R-tree with two properties: point data, and non-overlapping nodes.

According to [Page93, Falo94], the expected retrieval cost (i.e., number of disk accesses) of an overlap query with size $q_x \cdot q_y$ using R-trees (and consequently KDB-trees) is:

$$C(q_x,q_y)=\sum_{j=1}^{h}\left\{\frac{N}{(c\cdot m)^{j}}\cdot\left(n_{j,x}+q_x\right)\cdot\left(n_{j,y}+q_y\right)\right\} \tag{5}$$

where *h* is the height of the tree structure, *m* is the maximum number of entries in a KDB-tree node, *c* is the average capacity of a node, *N* is the total number of data entries and $n_{j,x} \bullet n_{j,y}$ is the average node size at level *j* (the root is assumed at level *j*=*h* and the leaf-nodes at level *j*=1).

The expression for computing the height *h* of the R-tree (or KDB-tree) is [Falo87]

$$h = 1 + \left\lceil \log_{c \cdot m} \frac{N}{c \cdot m} \right\rceil, \qquad (6)$$

the expected number of nodes N_j at level *j* is

$$N_j = \frac{N}{(c \cdot m)^j} \qquad (7)$$

and the average node sizes $n_{j,x}$ and $n_{j,y}$ are given by

$$n_{j,x} = n_{j,y} = \left(1 \big/ N_j \right)^{1/2} \qquad (8)$$

Using information from Table 3 and Eq. 6, 7 and 8 we can estimate the cost for each query window Q_l and Q_u (Eq. 5) and, summing up, the expected cost *C(R)* for the retrieval of a direction relation *R*:

$$C(R) = C(Q_{l-x}, Q_{l-y}) \cdot \begin{Bmatrix} 1, \text{if } Q_l \text{ exists} \\ 0, \text{otherwise} \end{Bmatrix} + C(Q_{u-x}, Q_{u-y}) \cdot \begin{Bmatrix} 1, \text{if } Q_u \text{ exists} \\ 0, \text{otherwise} \end{Bmatrix} \qquad (9)$$

3.3. Retrieval of Direction Relations using R-trees

The R-tree data structure [Gutt84] is a height-balanced tree, which consists of intermediate and leaf nodes. The MBRs of the actual data objects are assumed to be stored in the leaf nodes of the tree. Intermediate nodes are built by grouping rectangles at the lower level. An intermediate node is associated with some rectangle which encloses all rectangles that correspond to lower level nodes. The fact that R-trees permit overlap among node entries sometimes leads to unsuccessful hits on the tree structure. The R^+-tree [Sell87] and the R^*-tree [Beck90] methods have been proposed to address the problem of performance degradation caused by the overlapping regions or excessive dead-space respectively.

In order to retrieve objects that satisfy a direction relation with respect to a reference object we have to specify the MBRs that could enclose such objects (Table 1) and then to search the intermediate nodes that contain these MBRs. Table 4 presents the constraints for the intermediate nodes for each direction relation of Table 1. Notice that the same relation between intermediate nodes and the reference MBR holds for all the levels of the tree structure. For instance, the intermediate nodes that could enclose other intermediate nodes *P* that satisfy the general constraint ($P_{u\_x} > q'_{u\_y}$) should also satisfy the same constraint. This conclusion is applicable to all direction relations.

Relation	**Constraints for the Intermediate Nodes P to be Searched**
strong_north(p,q)	$(P_{u\_y} > q'_{u\_y})$
weak_north(p,q)	$(P_{u\_y} > q'_{u\_y}) \wedge (P_{l\_y} < q'_{u\_y})$
strong_bounded_north(p,q)	$(P_{u\_y} > q'_{u\_y}) \wedge (P_{l\_x} < q'_{u\_x}) \wedge (P_{u\_x} > q'_{l\_x})$
weak_bounded_north(p,q)	$(P_{u\_y} > q'_{u\_y}) \wedge (P_{l\_y} < q'_{u\_y}) \wedge (P_{l\_x} < q'_{u\_x}) \wedge (P_{u\_x} > q'_{l\_x})$
strong_north_east(p,q)	$(P_{u\_y} > q'_{u\_y}) \wedge (P_{u\_x} > q'_{u\_x})$
weak_north_east(p,q)	$(P_{u\_y} > q'_{u\_y}) \wedge (P_{l\_y} < q'_{u\_y}) \wedge (P_{u\_x} > q'_{u\_x})$

Table 4 Constraints for intermediate nodes of R-trees

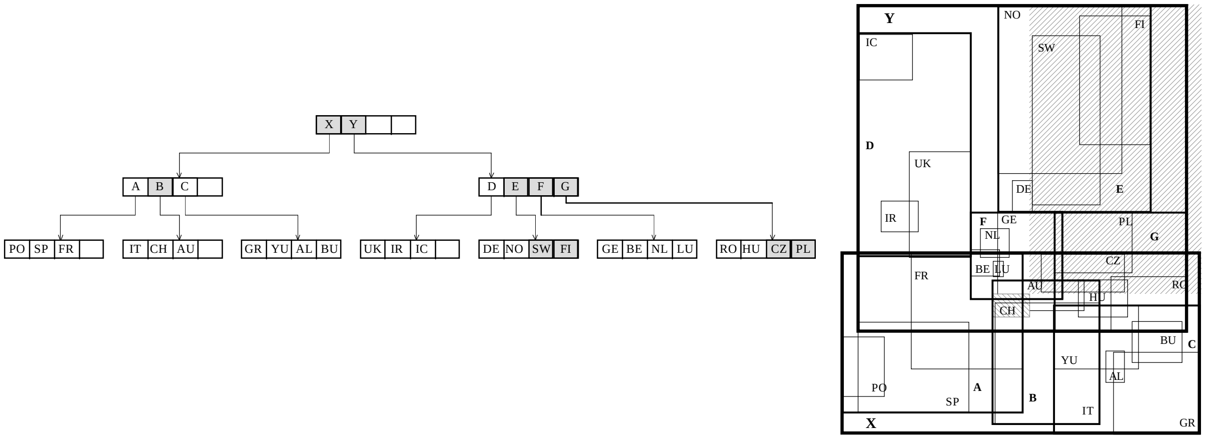
In general, the processing of a query of the form "find all objects *p* that satisfy a given direction relation with respect to object *q*" using R-trees involves the following steps:

Step 1. Starting from the top node, exclude the intermediate nodes *P* which could not enclose MBRs that satisfy the direction relation and recursively search the remaining nodes. This procedure involves Table 4.

Step 2. Among the leaf nodes retrieved, select the ones that satisfy the range constraints of Table 1.

Step 3. If necessary, follow a refinement step for the selected MBRs.

Figure 7 shows how the MBRs of Figure 4 are grouped and stored in an R-tree. At the lower level MBRs of countries (denoted by two letters) are grouped into seven intermediate nodes A, B, C, D, E, F and G. At the next level, the seven nodes are grouped into two larger nodes X and Y. Consider now the query "Find the countries *p strong_north_east* of Switzerland (CH)". The intermediate nodes that are selected for propagation are nodes X and Y (1st level), B, E, F and G (2nd level) i.e., the ones that have the representative point P_u within the shaded area (these nodes are grey in the tree structure of Figure 7a). Among the leaves of the intermediate nodes B, E, F and G, the ones that satisfy the constraint *north_east(p'$_l$, CH'$_u$)*, are SW' , FI' , CZ' and PL' .

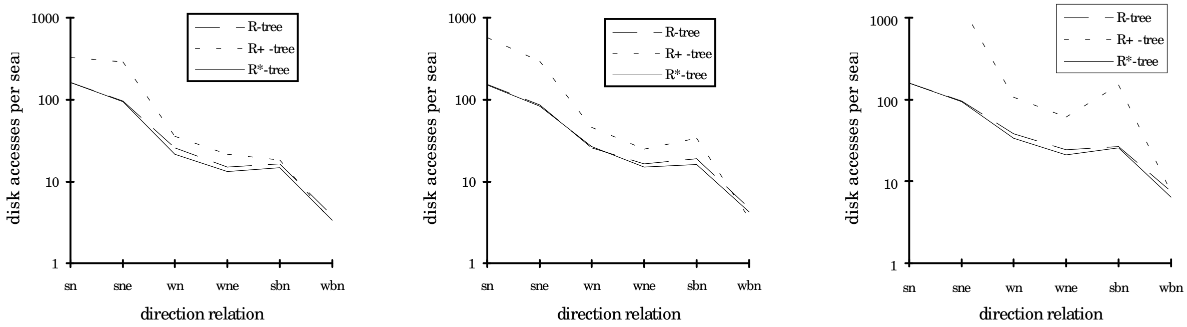


(a)

(b)

Fig. 7 Retrieval of relation *strong_north_east* using R-trees

We implemented and tested the most popular R-tree variations on the retrieval of direction relations using MBRs of variable sizes[2]. Figure 8 illustrates the number of disk accesses (in logarithmic scale) for each direction using small, medium and large MBRs respectively. According to Figure 8, R*-trees have consistently better performance than both R- and R^+- trees (R^+- trees are unsuitable for direction relations). Therefore, in the rest of the paper we use R*-trees (and wherever we use the term R-tree, we imply R*-tree).



(a) small data (b) medium data (c) large data

Fig. 8 Performance comparison of R-tree variants on direction relations

Most of the work in the literature has dealt with the expected performance of R-trees for processing overlap queries, i.e., the retrieval of data objects *p* that share common area with a query window *q*. Eq. 5 describes the expected cost of retrieval for both R- and KDB- trees, where the height of the tree is given by Eq. 6.

The fact that the data objects MBRs and R-tree nodes at each level are possibly overlapping make the computation of the R-tree nodes size different and not straightforward as the KDB-tree one (Eq. 8). If N_j is the number of nodes at level *j*, and d_j is the density[3] of these nodes then the average node sizes $n_{j,x}$ and $n_{j,y}$ are given by the following equations [Theo95b]:

$$n_{j,x}=n_{j,y}=\left(D_j/N_j\right)^{1/2} \tag{10}$$

where

$$D_j=\left\{1+\frac{(D_{j-1})^{1/2}-1}{(c\cdot m)^{1/2}}\right\}^2 \tag{11}$$

and

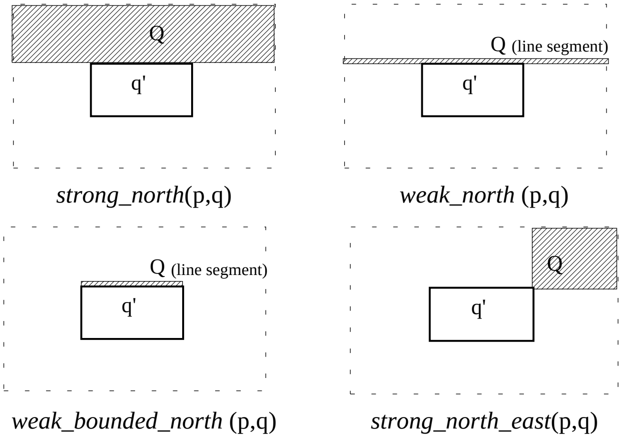
$$N_j=\frac{N_{j-1}}{c\cdot m} \tag{12}$$

Therefore, D_j and N_j can be computed recursively using D_0 and N_0 which denote the density *D* and the amount *N* respectively of the data object MBRs. Qualitatively, this means that we can estimate the retrieval cost of an overlap query just with the knowledge of the data set and the query window.

[2] Details about the implementation procedures can be found in section 4.

[3] We define *density D* of a set of *N* objects to be the sum of the object areas s_i divided by the global area: $D=\frac{\sum_{i=1}^{N} s_i}{global\ area}$

Since Eq. 5 expresses the expected performance of R-trees on overlap queries with a query window *q*, in order to estimate the cost of a direction relation *R*(*p*,*q*) we need the following transformation: *R*(*p*,*q*) ⇒ overlap(*p'*,*Q*). In other words, the retrieval of a direction relation using R-trees is equivalent (in terms of cost) to the retrieval of an overlap query using an appropriate query window *Q*. The necessary transformation *Q* for each direction relation *R* should take into consideration the corresponding constraint of the intermediate nodes because only these nodes are important when estimating the retrieval cost [Papa95b]. The appropriate query windows *Q* for each relation are illustrated in Figure 9. Each query window is a transformation of the corresponding constraints presented in Table 4.



strong_north(p,q) *weak_north* (p,q) *strong_bounded_north*(p,q)

weak_bounded_north (p,q) *strong_north_east*(p,q) *weak_north_east*(p,q)

Fig. 9 Query windows for the estimation of the retrieval cost

Using information from Figure 9 and Eq. 10, 11 and 12 we can estimate the expected cost for the query window *Q* (Eq. 5) which equals to the expected cost *C(R)* for the retrieval of a direction relation *R*:

$$C(R)=C(Q_x,Q_y)=\sum_{j=1}^{h}\left\{\frac{N}{(c\cdot m)^j}\cdot(n_{j,x}+Q_x)\cdot(n_{j,y}+Q_y)\right\} \qquad (13)$$

Intuitively R-trees perform better than other indexing methods in cases where many constraints are involved in the definition of the direction relation of interest. The experimental results of the next section demonstrate that this is actually the case when four constraints are involved, while for one constraint B-trees have a better performance and for the intermediate relations (two or three constraints) KDB-trees usually win.

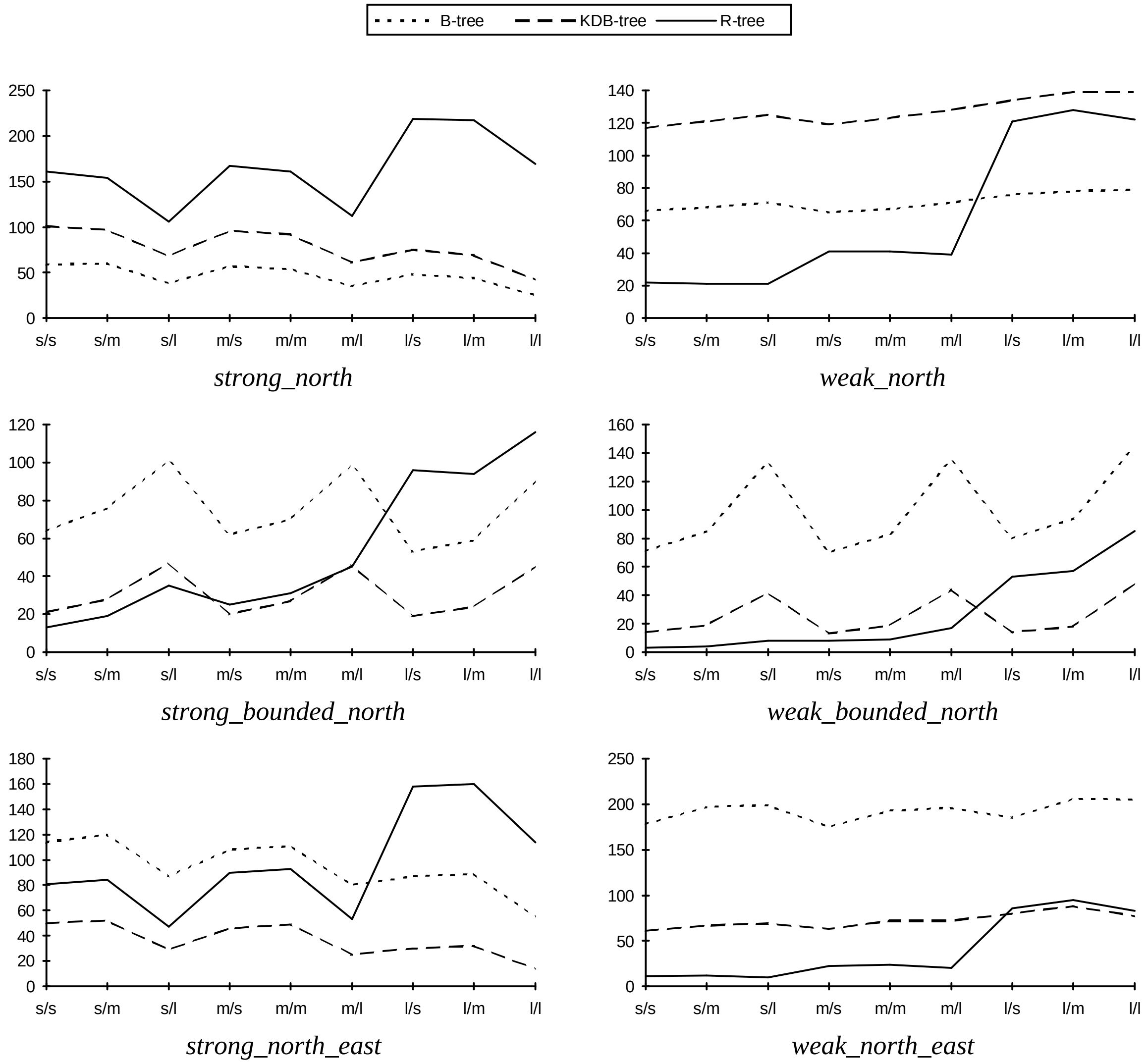
4. PERFORMANCE COMPARISON

In the previous section we have argued that the performance of each indexing method depends significantly on the particular range. In this section we present several experimental results that justify our argument. In order to experimentally quantify the performance, we created tree structures by inserting 10,000 randomly generated MBRs. We tested three *data files*:

- the first file contains *small* MBRs: the average area of each rectangle is 0.003% of the global area
- the second file contains *medium* MBRs: the average area of each rectangle is 0.25% of the global area

- the third file contains *large* MBRs: the average area of each rectangle is 5% of the global area.

The search procedure used three *query files* for each data file containing 100 rectangles, also randomly generated, with similar size properties as the data rectangles. We used the same data/query files for the retrieval of direction relations with B-trees, KDB-trees and R-trees. The performance of each data structure per relation (number of disk accesses for each data/query size combination) is illustrated in Figure 10.



strong_north

weak_north

strong_bounded_north

weak_bounded_north

strong_north_east

weak_north_east

Fig. 10 Performance Comparison (disk accesses per data/query size combination)

As a first conclusion we notice the way that the data and query size affect the performance of the data structures. The performance of B-trees and KDB-trees depends significantly on the query size while the opposite happens for R-trees where the data size is the main factor responsible for the high or low performance of the structure. The main conclusions that can be drawn from Figure 10 are:

- B-tree is the most efficient structure for 1-constraint relations only (e.g. *strong_north*). When more range constraints (even on the same axis) are involved it is not competitive.

- KDB-tree is the most competitive index when two constraints for the same representative point are involved (e.g. *strong_north_east*). It is also very competitive when both representative points are involved and the corresponding query windows are too selective (e.g. *strong_bounded_north*, *weak_bounded_north*).
- R-tree is the winner when four constraints are involved (*weak_bounded_north*, *weak_north_east*). However, when the data rectangles are large, R-tree is not able to organise the data efficiently and, therefore, it usually loses against B-tree or KDB-tree even for relations involving many constraints.

From the above results, it is evident that there is not an overall winner but the performance of the structures is affected by the number of constraints involved and the size of the data and query rectangles.

The analytical models proposed in section 3 can be used as guidelines to spatial query processors and optimisers in order to select appropriate indexes for use in answering spatial queries. In order to evaluate the quality of the proposed analytical approximations, we compared the expected and the experimental cost for the retrieval of the direction relations. For the analytical estimation of the B-tree, KDB-tree and R-tree indexing mechanisms we used the formulas of Eq. 4, 9 and 13, respectively, and the following typical values:

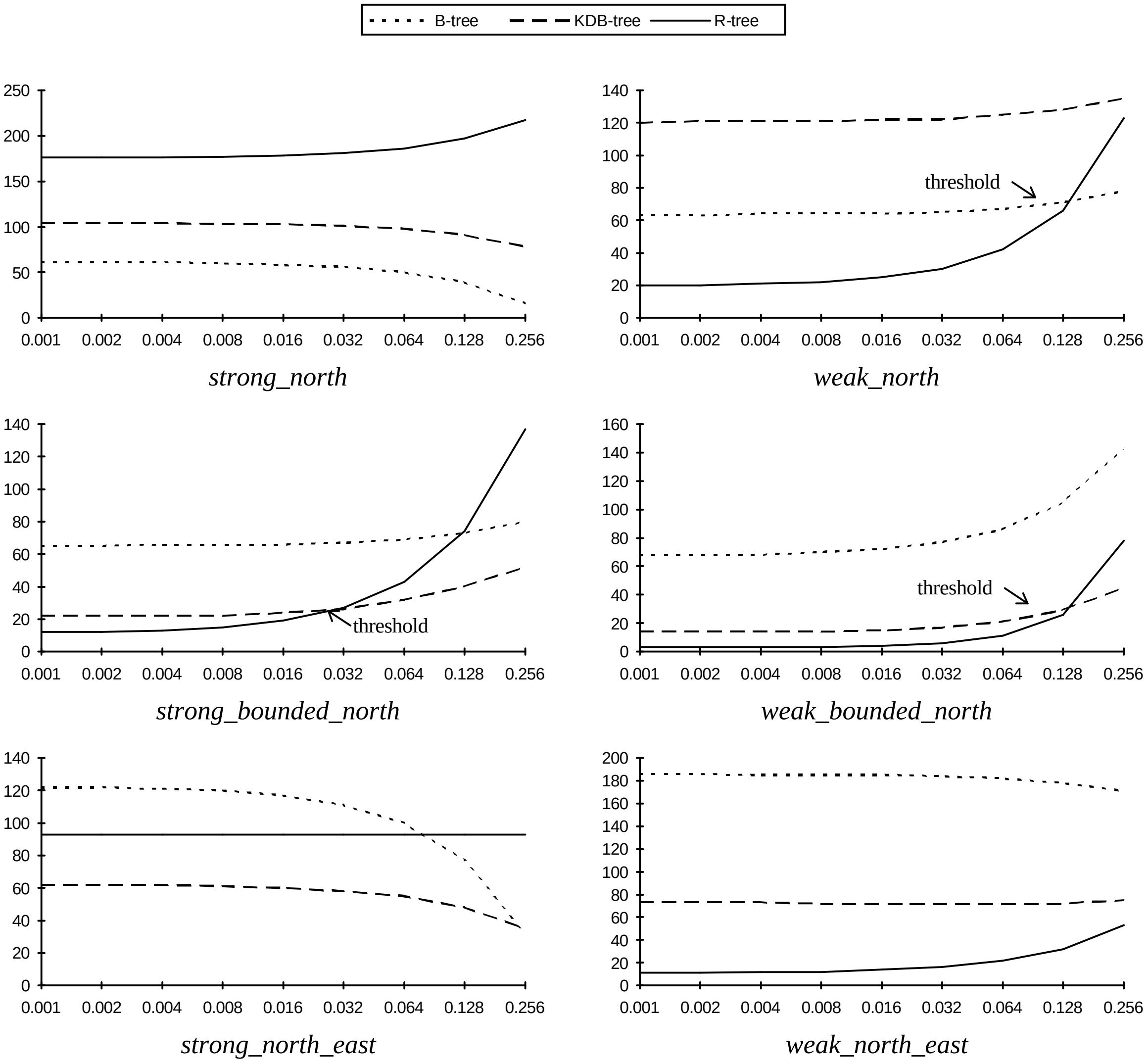
- number of data $N = 10,000$,
- average node capacity $c = 0.67$,
- maximum number of entries in a node $m = 126$ or 84 or 50 (for B-trees, KDB-trees and R-trees, respectively).

Table 5 lists the relative errors of the actual results compared to the predictions of the proposed analytical models for the various data / query size combinations.

	Relative Error		
Direction Relation	B-trees	KDB-trees	R-trees
strong_north	0% - 25%	0% - 30%	0% - 25%
weak_north	0% - 10%	0% - 10%	0% - 10%
strong_bounded_north	0% - 20%	5% - 15%	5% - 30%
weak_bounded_north	0% - 10%	5% - 15%	0% - 30%
strong_north_east	0% - 25%	5% - 30%	0% - 20%
weak_north_east	0% - 15%	0% - 15%	5% - 25%
TOTAL Average	5%	10%	15%

Table 5: Relative error in estimating the performance of the tree structures.

The estimations of the performance of all the indexing methods are very close to the experimental results. In most cases, the relative error is lower than 15%. Only for large data and relations with large answer sets, such as *strong_north* and *strong_north_east* the relative error is higher (20%-30%). Such analytical models are useful to spatial query optimisers because they permit the prediction of the cost. Since the most appropriate index for some relations (*strong_bounded_north*, *weak_bounded_north* etc.) is not known beforehand, but depends on the size of the data and query MBRs, the analytical models can be used to select the most suitable method. Figure 11 illustrates the expected performance of the three data structures for the direction relations implemented. For the analytical estimations we used data and query MBRs with the sizes of each side varying from 0.001 up to 0.256 of the global unit space (i.e., the area of each rectangle varied from 0.001^2 up to 0.256^2 of the global area) and assumed that the data and query MBRs are equal-sized, a property which usually appears in real geographic (and other) applications.



strong_north

weak_north

strong_bounded_north

weak_bounded_north

strong_north_east

weak_north_east

Fig. 11 Analytical Performance Comparison (disk accesses per data-query size)

According to Figure 11, the decision of the most efficient method depends on the data and query size for three out of six the direction relations (*weak_north*, *strong_bounded_north* and *weak_bounded_north*). R-trees perform better for these relations when the size is small but, after a specified threshold, KDB-trees (or B-trees for the *weak_north* relation) are the best choice. The knowledge of such a threshold is very important for a spatial query optimiser. It is also important to notice that the ordering of the three data structures per relation remains unchangeable for the experimental and the analytical results of Figures 10 and 11, respectively, since query optimisers primarily need to know about the best solution (i.e., the best data structure out of three) and, secondarily, about the exact retrieval cost.

In general, usually there is no overall winner even for a specified direction relation. Therefore, we believe that modifications or extensions of the three data structures could be more suitable for certain range queries. In the next sections we propose such extensions of B-trees, KDB-trees and R-trees.

5. EXTENSIONS

One- and multi-dimensional data structures have been designed to efficiently answer some particular types of queries. B-trees index alphanumeric data in order to answer match or range queries, KDB-trees are efficient for match or range queries on multi-dimensional points and R-trees were designed for overlap queries on multi-dimensional MBRs. However, direction relations between spatial objects in 2D space do not follow directly in one of these categories and, therefore, in section 3 we presented the appropriate procedures in order to use one of these "classic" data structures. In this section we propose extensions of the three data structures that facilitate efficiency for some 2D range queries.

5.1. Extensions of B-trees

In the case of B-trees, we propose schemes for the maintenance of information regarding the MBR extents in the index; we call the proposed structures *composite B-trees*. As shown in section 3, the number of B-trees to be searched is equal to the number of constraints that compose the query. This inconvenience may be overcome if the B-tree accommodates additional information regarding the MBRs in its *leaf nodes*. That is, a composite key may be maintained instead of a simple numerical value (i.e., the coordinate value of one MBR corner). This composite key consists of the coordinate values of the two MBR corners in one axis (x- or y- axis) or even of all four coordinates of the two MBR corners. One of these coordinates is the primary component, and based on its value the B-tree is built. The rest of the coordinates are used for the elimination of irrelevant MBRs.

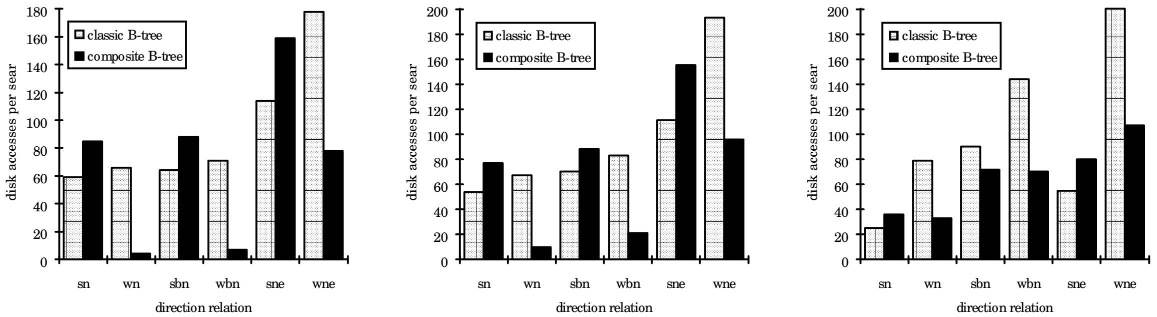
The key of the composite B-tree considered here consists of two values that represent the lower and upper coordinates of each MBR in one of the two axes. One of these values is the primary component. This scheme reduces each MBR into two line segments which represent its extents over the two dimensions of the space and are indexed separately. Clearly, the efficient retrieval of direction relations can be achieved when four composite B-trees are maintained: $B_{l\text{-}x}$-tree that keeps $p'_{l\text{-}x}$ as primary component and $p'_{u\text{-}x}$ as secondary component, and, similarly, $B_{u\text{-}x}$-tree, $B_{l\text{-}y}$-tree and $B_{u\text{-}y}$-tree. Some relations imply the search of only one B-tree (such as *strong_north*), while the rest imply exactly two searches (i.e., one for each axis such as *strong_north_east*).

In addition, provided that the information regarding the MBRs' extents over each axis are maintained in two indices, the approach to satisfy a query is not unique. For instance, relation *weak_north* requires no index to be searched for the x-axis, but it may be satisfied by searching either a $B_{l\text{-}y}$-tree (condition to be fulfilled: $q'_{l\text{-}y} < p'_{l\text{-}y} < q'_{u\text{-}y}$; refinement condition: $p'_{u\text{-}y} > q'_{u\text{-}y}$) or a $B_{u\text{-}y}$-tree (condition to be fulfilled: $p'_{u\text{-}y} > q'_{u\text{-}y}$; refinement condition: $q'_{l\text{-}y} < p'_{l\text{-}y} < q'_{u\text{-}y}$).

The processing of a direction relation query involves the four steps described in sub-section 3.1. During step 2 the search of each index is based on the primary component of the composite key while, in the leaf nodes, the possible refinement condition for the secondary component is considered to eliminate irrelevant MBRs. The selection of the most effective B-tree by a query processor or optimiser is not always trivial. In general, the selection should take into account: a) the direction relation involved; b) the query window size and position; and c) the distribution of MBR corners over the work space (this depends on the MBR distribution and size). We have implemented and tested three schemes for the selection of the most effective B-tree:

- *Selection based on the relation involved:* This scheme selects the index based on the relation using statistical results. For example, the $B_{l\text{-}y}$-tree is chosen for *strong_north*.
- *Selection based on the location and size of the query object:* The query object divides each axis of the unit work space into five ranges: A = [0, q_l); B = q_l; C = (q_l, q_u); D = q_u; E = (q_u, 1]. This scheme assumes that the MBR corners are uniformly distributed over the work space and selects the index based on the shortest range or sum of ranges involved. For example, the $B_{l\text{-}y}$-tree is chosen for *weak_north* relation only if range C is shorter that range E; otherwise the $B_{u\text{-}y}$-tree is chosen.
- *Selection based on the location and size of the query object and the distribution of MBR corners:* This scheme is obtained by maintaining an array (directory) with information about the number of lower and upper coordinates over the work space using a pre-determined resolution. For a given query object the lower and upper coordinates that fall within the five ranges A, B, C, D and E are computed, so that the number of segments to be retrieved for the two candidate indices is derived. The composite B-tree with the fewest segments is then chosen.

The third scheme proved to be the most efficient. A comparison of the classic B-tree and the third scheme of the composite B-tree is illustrated in Figure 12.



(a) small data / query (b) medium data / query (c) large data / query

Fig. 12 Performance comparison of the classic and the composite B-tree

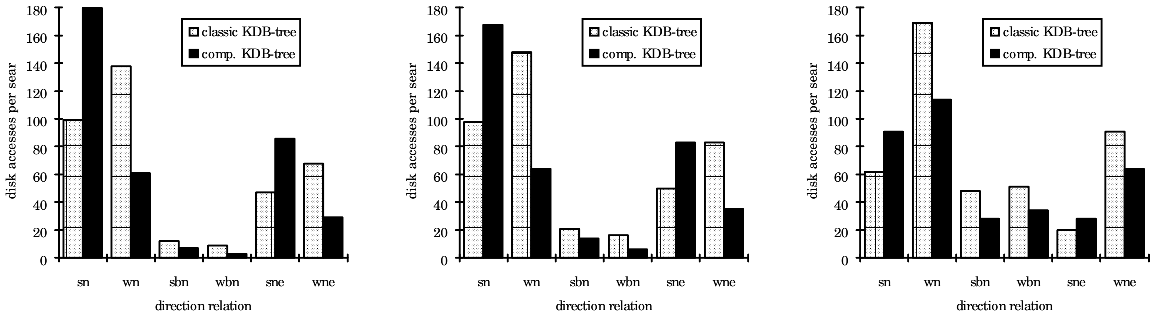
As a general conclusion, the composite B-tree outperforms the classic B-tree when the constraints involved imply access to less indexes than the classic method. For example, the retrieval of *weak_north* implies access to one composite B-tree instead of two classic B-trees, the retrieval of *weak_north_east* implies access to two composite B-trees instead of four classic B-trees, and so on. Compared to the other methods, the composite B-tree is the most efficient one for large data when more than one constraints are involved (e.g. *weak_north* and *weak_bounded_north* for large data files), since R-trees are unable to index such data efficiently, but it is sensitive to query size; large query windows are not handled efficiently by composite B-trees [Theo95a].

5.2. Extensions of KDB-trees

A similar approach can be considered for KDB-trees. KDB-trees handle two-dimensional data efficiently when the search procedure involves one of the two-representative points of the MBRs. However, most of the direction relations involve both points and, as a consequence, the intersection of two answer sets should be computed (step 3 in sub-section 3.2). The determination of both the answer sets and their intersection can be a highly time-consuming procedure in large databases. We propose the maintenance the opposite MBR corner, along with the corner on which indexing is based, in the leaf nodes of a composite KDB-tree. The additional point will serve the fast elimination of irrelevant MBRs. Clearly, the efficient retrieval of direction relations can be achieved when two *composite KDB-trees* are maintained: a KDB$_l$-tree where indexing is based on the lower left corners of the MBRs; and, a KDB$_u$-tree where indexing is based on the upper right corners of the MBRs.

A range relation query may be satisfied by searching one of the two composite KDB-trees. Similarly to the composite B-tree the selection of the most effective tree is not trivial, and should consider: a) the direction relation involved; b) the query window size and position; and c) the distribution of MBRs over the work space. The three schemes of the composite B-tree can also be applied for the selection of the most efficient composite KDB-tree, provided that linear ranges are substituted by area ranges.

We have implemented and tested the three schemes of composite KDB-trees and found again the third one to be the most efficient. A comparison of the classic KDB-tree and the third scheme is illustrated in Figure 13.



(a) small data / query

(b) medium data / query

(c) large data / query

Fig. 13 Performance comparison of the classic and the composite KDB-tree

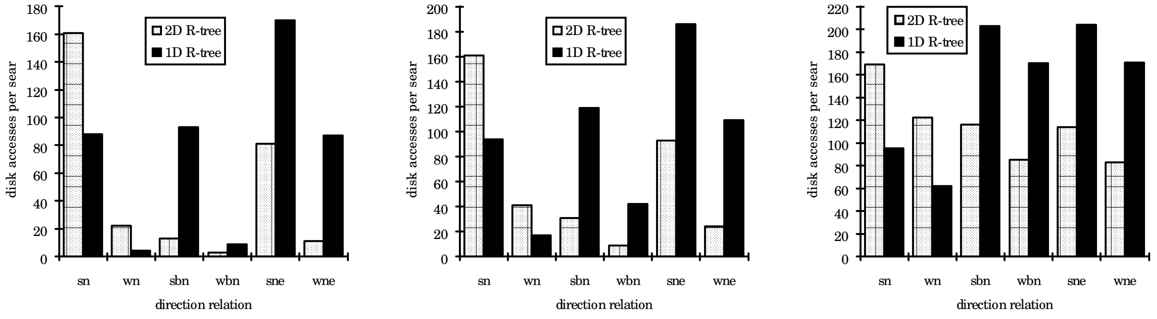
As a general conclusion, the composite KDB-tree outperforms the classic KDB-tree in most cases. The opposite happens only for the *strong_north* and *strong_north_east* relations where only one index is accessed (see Table 3). Compared to the other methods, the composite KDB-tree is the most efficient one for the *weak_north* relation and small or medium data, where the corresponding Q_l query window is too selective, but it is insufficient for the rest ones.

5.3. Extensions of R-trees

R-trees handle two-dimensional data efficiently when the search procedure involves both axes of the work space. However, several direction relations, such as *strong_north* and *weak_north*, involve search on only one axis. In such cases, the information regarding the other axis, which is maintained in the two-dimensional R-tree is useless. Clearly, a one-dimensional R-tree (i.e., segment tree), which is an index of the MBR extents along the axis of interest, would be more efficient because it is more compact (tree nodes accommodate a larger number of entries) and effective (the MBR extents along the other axis do not affect the maintenance of the index) than the two-dimensional R-tree.

For the retrieval of direction relations that involve both axes, two one-dimensional R-trees are needed to index the MBR extents over each axis separately. Each index provides a set of MBRs, and the intersection of the two sets composes the qualified set. In such cases though, the two-dimensional R-tree is expected to be more efficient.

We have implemented and tested the one-dimensional R-tree in comparison to the classic two-dimensional version. The results are illustrated in Figure 14.



(a) small data / query (b) medium data / query (c) large data / query

Fig. 14 Performance comparison of the two-dimensional and the one-dimensional R-tree

According to Figure 14, the one-dimensional R-tree outperforms the two-dimensional R-tree only for the *strong_north* and *weak_north* relations. In the other cases, where two one-dimensional indexes need to be searched, it is insufficient. *Weak_north* is the only relation where the one-dimensional R-tree is the overall winner, provided that data MBRs are not large.

6. CONCLUSION

This paper describes implementations of direction relations for spatial database systems. Direction relations constitute a new type of query for spatial access methods which so far have been concerned with disjoint/overlap relations, topological relations of high resolution and nearest neighbour queries. Despite the fact that direction queries are of equal importance to previous ones they have not been extensively implemented, mainly because of the lack of well-defined direction relations between actual objects.

In this work we defined direction relations between points and we used these definitions as a basis for relations between objects. For the purposes of this paper we used a set of six object relations, but a large number of additional ones can be defined using the point relations. Then we showed how these relations can be transformed to range queries and retrieved in existing DBMSs using three alternative indexing techniques: B-trees, KDB-trees and R-trees and compared their efficiency with data and query files of various sizes. We also provided analytical formulas for their expected performance which, in most cases, proved to be very accurate for all the indexing methods (relative error usually lower than 15%), a fact that renders the derived formulas suitable for query optimisers.

The main conclusion that arises from the experimental and analytical comparison of the alternative indexing techniques is that there is not a single data structure that performs best in all queries but the performance depends on the following factors:

- the number and the type of range constraints involved in the definition of the direction relations of interest,
- the data size (i.e., the size of the primary MBRs) and
- the query size (i.e., the size of the reference MBRs).

Besides the classic implementations of the three data structures, extensions of them (called *composite B-trees*, *composite KDB-trees* and *1D R-trees*) were also implemented in order to facilitate efficiency for some types of queries. The guidelines to a spatial query optimiser that should choose the most suitable indexing method for a specific input query are summarised in Table 6.

Thesee guidelines can be extended to include other spatial data structures (e.g. Grid files) and other direction relations. Progress can also be achieved in specialised data structures for queries that involve direction relations or combinations of several types of spatial information (“find the 5 nearest land parcels northeast of the lake”).

Direction Relation **(as a set of range constraints)**	**B-trees**		**KDB-trees**		**R-trees**	
	classic	comp.	classic	comp.	2D	1D
1 constraint on one axis (e.g., *strong_north*)	√	√				
2 constraints on one axis (e.g., *weak_north*)		√				√ [1]
1 constraint on the first axis PLUS 1 constraint on the second axis (e.g., *strong_north_east*)			√	√		
1 constraint on the first axis PLUS 2 constraints on the second axis (e.g., *strong_bounded_north*)			√	√	√ [1]	
2 constraints on the first axis PLUS 2 constraints on the second axis (e.g., *weak_bounded_north, weak_north_east*)			√ [2]	√ [2]	√ [1]	
Remarks	[1] may not be efficient for large data [2] may not be efficient for large queries					

Table 6 Decision rules for the efficient retrieval of direction relations

REFERENCES

[Bato81] Batory, D.S., "B+ Trees and Indexed Sequential Files: A Performance Comparison", Proceedings of ACM SIGMOD International Conference on Management of Data, 1981.

[Beck90] Beckmann, N., Kriegel, H.P., Schneider, R., Seeger, B., "The R*-tree: an Efficient and Robust Access Method for Points and Rectangles", Proceedings of ACM SIGMOD International Conference on Management of Data, 1990.

[Come79] Comer, D., "The Ubiquitous B-Tree", ACM Computing Surveys, Vol. 11(2), pp. 121-137, 1979.

[Egen91] Egenhofer, M. "Reasoning about Binary Topological Relations", Proceedings of the 2nd International Symposium on Large Spatial Databases (SSD), Springer Verlag LNCS, 1991.

[Falo87] Faloutsos, C., Sellis, T., Roussopoulos, N., "Analysis of Object Oriented Spatial Access Methods", Proceedings of ACM SIGMOD International Conference on Management of Data, 1987.

[Falo94] Faloutsos, C., Kamel, I., "Beyond Uniformity and Independence: Analysis of R-trees Using the Concept of Fractal Dimension", Proceedings of the 13th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS), 1994.

[Fran92] Frank, A.U., "Qualitative Spatial Reasoning about Distances and Directions in Geographic Space", Journal of Visual Languages and Computing, Vol. 3, pp. 343-371, 1992.

[Fran95] Frank, A.U., "Qualitative Spatial Reasoning: Cardinal Directions as an Example", International Journal of Geographic Information Systems (to appear).

[Grig95] Grigni, M., Papadias, D., Papadimitriou, C., "Topological Inference", Proceedings of the International Joint Conference of Artificial Intelligence, 1995.

[Gutt84] Guttman, A., "R-trees: a Dynamic Index Structure for Spatial Searching", Proceedings of ACM SIGMOD International Conference on Management of Data, 1984.

[Hern94] Hernandez, D., "Qualitative Representation of Spatial Knowledge", Springer Verlag LNAI, 1994.

[Knut73] Knuth, D., "The Art of Computer Programming, vol 3: Sorting and Searching", Addison-Wesley, 1973.

[Page93] Pagel, B., Six, H., Toben, H., Widmayer, P., "Towards an Analysis of Range Query Performance", Proceedings of the 12th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS), 1993.

[Papa94a] Papadias, D., Frank, A.U., Koubarakis, M., "Constraint-Based Reasoning in Geographic Databases: The Case of Symbolic Arrays", Proceedings of the 2nd ICLP Workshop on Deductive Databases, 1994.

[Papa94b] Papadias, D., Theodoridis, Y., Sellis, T., "The Retrieval of Direction Relations Using R-trees", Proceedings of the 5th International Conference on Database and Expert Systems Applications (DEXA), 1994.

[Papa94c] Papadias, D., Sellis, T., "Qualitative Representation of Spatial Knowledge in two-dimensional Space", Very Large Data Bases Journal, Special Issue on Spatial Databases, Vol 3(4), pp. 479-516, 1994.

[Papa95a] Papadias, D., Sellis, T., "A Pictorial Query-by-Example Language", Journal of Visual Languages and Computing, Special issue on Visual Query Systems, Vol. 6(1), 53 -72, 1995.

[Papa95b] Papadias, D., Theodoridis, Y., Sellis, T., Egenhofer, M., "Topological Relations in the World of Minimum Bounding Rectangles: a Study with R-trees", Proceedings of ACM SIGMOD International Conference on Management of Data, 1995.

[Robi81] Robinson, J.T., "The K-D-B-Tree: A Search Structure for Large Multidimensional Dynamic Indexes", Proceedings of ACM SIGMOD International Conference on Management of Data, 1981.

[Rous95] Roussopoulos, N., Kelley, F., Vincent, F., "Nearest Neighbor Queries", Proceedings of ACM SIGMOD International Conference on Management of Data, 1995.

[Sell87] Sellis, T., Roussopoulos, N., Faloutsos, C., "The R^+-tree: A Dynamic Index for Multi-Dimensional Objects", Proceedings of the 13th International Conference on Very Large Data Bases (VLDB), 1987.

[Theo95a] Theodoridis, Y., Papadias, D., Stefanakis. E., "Supporting Direction Relations in Spatial Database Systems", submitted for publication, 1995. Also available as Technical Report KDBSLAB-TR-95-02, Knowledge and Database Systems Laboratory, National Technical University of Athens, Greece, anonymous ftp site: ftp://dbnet.ece.ntua.gr:/pub/papers/reports/ 1995/tr9502.ps.Z.

[Theo95b] Theodoridis, Y., Sellis, T., "On the Performance Analysis of Multi-dimensional R-tree-based Data Structures", submitted for publication, 1995. Also available as Technical Report KDBSLAB-TR-95-03, Knowledge and Database Systems Laboratory, National Technical University of Athens, Greece, anonymous ftp site: ftp://dbnet.ece.ntua.gr:/pub/papers/reports/ 1995/tr9503.ps.Z.

[Theo95c] Theodoridis, Y., Papadias, D., "Range Queries Involving Spatial Relations: A Performance Analysis", In the Proceedings of the 2nd International Conference on Spatial Information Theory (COSIT), 1995.

[Yao78] Yao, A.C., "On Random 2-3 Trees", Acta Informatica, Vol. 9(2), pp. 159-168, 1978.