Supporting Direction Relations in Spatial Database Systems

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

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

[+]Department of Computer Science and Engineering
University of California, San Diego
La Jolla, California, USA 92093-0114
dimitris@cs.ucsd.edu

Abstract Despite the attention that direction relations have attracted in domains such as Spatial Query Languages, Image and Multimedia Databases, Reasoning and Geographic Applications, they have not been extensively applied in spatial access methods. 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 several indexing techniques. Then we describe our implementations and provide experimental results on B- and R- tree-based data structures to support direction relations efficiently.

Keywords: Direction relations, Spatial Data Structures, Query Optimisation.

1. INTRODUCTION

There has been an increasing interest recently about the representation and processing of spatial relations in Spatial Database Systems (for an extensive discussion see [Papa94c]). This interest has focused on several topics such as Reasoning [Fran95], Consistency Checking [Egen93] and Spatial Query Languages [Papa95a]. In this paper we deal with the retrieval of spatial relations in DBMSs used for geographic, or more generally, spatial, applications. In particular we concentrate on the retrieval of *direction relations* using B- and R- tree-based data structures.

Direction relations (*north, northeast*) deal with order in space. Unlike the case of topological relations where there seems to exist a set of widely accepted relations [Egen90], there are no such definitions of direction relations. Consider, for instance, the map of Europe illustrated in Figure 1, and the query "find all countries *northwest* of Greece". Although such queries are very common, the answer is not always trivial. Most people will argue that Germany is northwest of Greece but what about Italy? There are parts of Italy that are northwest of all parts of Greece, but is this sufficient for stating that Italy is northwest of Greece?

Yannis Theodoridis was partially supported by the Department of Research and Technology of Greece (PENED'91) and Dimitris Papadias by NSF - IRI 9221276.
We would like to thank Timos Sellis for carefully reading this paper and providing useful comments.
Correspondence should be addressed to Dimitris Papadias.

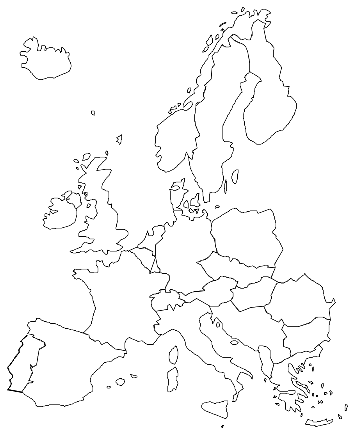


Fig. 1 Map of Europe

The answer to this query depends on the definition of direction relations which may vary for each application. 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, Muke90]. Then we show how the relations that we define can be efficiently retrieved in existing DBMSs. Although for other types of spatial relations, such as topological relations, there has been extensive work in spatial data structures and specialised indexing methods have been proposed [Gutt84, Sell87, Beck90, Papa95b], limited work has focused on direction relations (see [Papa94b] for related work).

The results of this paper are directly applicable to Spatial Databases and Geographic Information Systems (GISs) where the formalization of spatial relations is crucial for user interfaces and query optimisation strategies. Furthermore, the importance of direction relations has been pointed out by several researchers in areas including Multimedia Databases [Sist94], Spatial Reasoning [Glas92], Cognitive Science [Jack83] and Linguistics [Hers86].

The paper is organised as follows. In section 2 we define several direction relations between objects in 2D-space. Section 3 describes how MBR approximations, commonly used in spatial access methods, can be used in the retrieval of direction relations. Sections 4 and 5 discuss the retrieval of direction relations using alternative access methods (B- and R- trees respectively). Section 6 is concerned with extensions to access methods that increase performance for some queries. Section 7 compares the different implementations illustrating experimental results, and section 8 concludes with comments on future work.

2. 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 formulae. 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 following discussion 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, q_j be a point of object q, and *X* and *Y* be functions that give the *x* and y coordinate of a point. 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*).

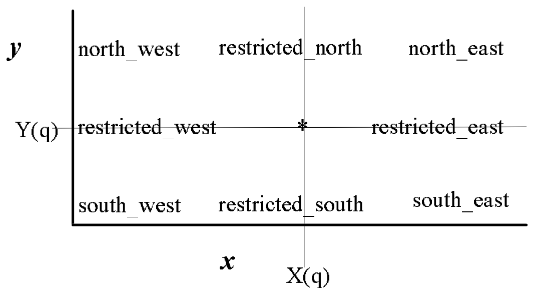


Fig. 2 Plane partitions using one point per object

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

north_west$(p_i,q_j) \equiv X(p_i)<X(q_j) \wedge Y(p_i)>Y(q_j)$, restricted_north$(p_i,q_j) \equiv X(p_i)=X(q_j) \wedge Y(p_i)>Y(q_j)$,

north_east$(p_i,q_j) \equiv X(p_i)>X(q_j) \wedge Y(p_i)>Y(q_j)$, restricted_west$(p_i,q_j) \equiv X(p_i)<X(q_j) \wedge Y(p_i)=Y(q_j)$,

same_position$(p_i,q_j) \equiv X(p_i)=X(q_j) \wedge Y(p_i)=Y(q_j)$.

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., north_east$(p_i,q_j) \Leftrightarrow$ south_west(q_j, p_i)). Additional direction relations can be defined using disjunctions of the nine relations; for example:

north$(p_i,q_j) \equiv$ north_west$(p_i,q_j) \vee$ restricted_north$(p_i,q_j) \vee$ north_east(p_i,q_j),

south$(p_i,q_j) \equiv$ south_west$(p_i,q_j) \vee$ restricted_south$(p_i,q_j) \vee$ south_east(p_i,q_j),

same_level$(p_i,q_j) \equiv$ restricted_west$(p_i,q_j) \vee$ same_position$(p_i,q_j) \vee$ restricted_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: strong_north $\equiv \forall p_i\ \forall q_j$ 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 : strong_bounded_north(p,q) $\equiv \forall p_i \forall q_j$ north(p_i, q_j) $\wedge \forall p_i \exists q_j$ north_east(p_i, q_j) $\wedge \forall p_i \exists q_j$ 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's northmost 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.

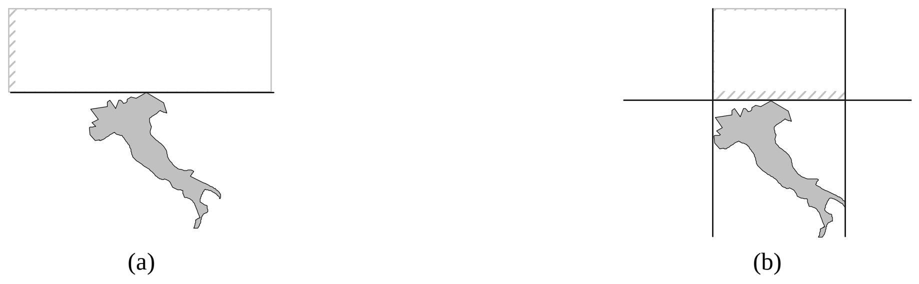


Fig. 3 Acceptance area for strong_north and strong_bounded_north relations

Table 1 illustrates several direction relations between objects using examples from the map of Figure 1. All of these relations 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. The definition of pairwise disjoint relations is not a difficult task, but it is beyond the scope of this paper, which aims to show how direction relations of different resolution between extended objects can be defined and retrieved in spatial data structures. Furthermore, since the relations of Table 1 are concerned with direction north only, they do not provide a complete coverage. Customised direction relations between extended objects that satisfy the above properties and correspond to certain application needs are straightforward to develop using the definitions between points.

Although the previous discussion refers to actual two-dimensional objects, usually spatial access methods store approximations that need only a few points for their representation, instead of the objects themselves. Such approximations are used 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). The next section discusses how MBRs can be used for the retrieval of direction relations between actual objects.

3. DIRECTION RELATIONS AND MINIMUM BOUNDING RECTANGLES

Minimum Bounding Rectangles have been used extensively to approximate objects in Spatial Data Structures and Spatial Reasoning 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. While MBRs demonstrate some disadvantages when approximating non convex or diagonal objects, they are the most commonly used approximations in spatial applications. Figure 4 illustrates how the map of Figure 1 can be approximated by MBRs.

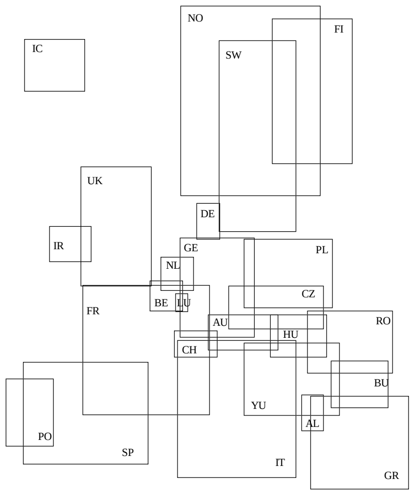
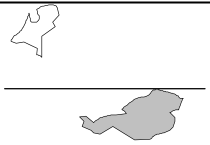


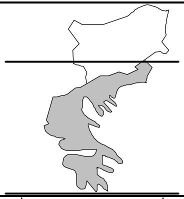
Fig. 4 MBR approximations for the map of Europe

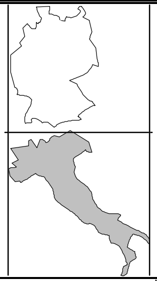
Some access methods (e.g., R-trees) explicitly store MBRs, while others (e.g., B-trees) compare object locations using representative points. In order to answer the query “find all objects p that satisfy the relation R with respect to an object q” we have to retrieve all MBRs p' that satisfy the relation R' with respect to the MBR q' of object q. Table 1 illustrates the mapping from direction relations R between actual objects to relations R' between MBR representative points.

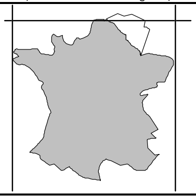
Because MBRs differ from the actual objects they enclose, they are not always adequate to express the relation between the actual objects. For this reason, spatial queries involve the following two step strategy: First a *filter step* based on MBRs is used to rapidly eliminate MBRs of objects that could 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* [Papa94b]. For the rest of the relations all retrieved MBRs correspond to objects that satisfy the query.

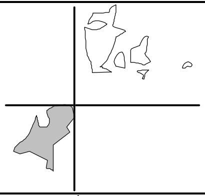
Relation between objects	**Example**	**Definition**	**Corresponding relation between representative points**

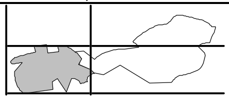
strong_north(p,q)
- sn -
 	
example:
sn(NL,AU)
		$\forall p_i\ \forall q_j$ north(p_i, q_j)	north(p'_l, q'_u)

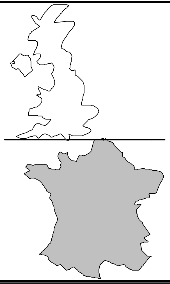
weak_north(p,q)
- wn -
 	
example:
wn(BU,GR)
		$\exists p_i \forall q_j$ north$(p_i,q_j) \wedge$
		$\forall p_i \exists q_j$ north$(p_i,q_j) \wedge$
		$\exists p_i \exists q_j$ south(p_i,q_j)
			north$(p'_u, q'_u) \wedge$
			north$(p'_l, q'_l) \wedge$
			south(p'_l, q'_u)

strong_bounded_north(p,q)
- sbn -
 	
example:
sbn(GE,IT)
		$\forall p_i \forall q_j$ north$(p_i, q_j) \wedge$
		$\forall p_i \exists q_j$ north_east$(p_i, q_j) \wedge$
		$\forall p_i\ \exists q_j$ north_west(p_i, q_j)
			north$(p'_l, q'_u) \wedge$
			north_east$(p'_l, q'_l) \wedge$
			north_west(p'_u, q'_u)

weak_bounded_north(p,q)
- wbn -
 	
example:
wbn(BE,FR)
		$\exists p_i\ \forall q_j$ north$(p_i, q_j) \wedge$
		$\exists p_i\ \exists q_j$ south$(p_i, q_j) \wedge$
		$\forall p_i\ \exists q_j$ north_east$(p_i, q_j) \wedge$
		$\forall p_i \exists q_j$ north_west(p_i, q_j)
			north$(p'_u, q'_u) \wedge$
			south$(p'_l, q'_u) \wedge$
			north_east$(p'_l, q'_l) \wedge$
			north_west(p'_u, q'_u)

strong_north_east(p,q)
- sne -
 	
example:
sne(DE,NL)
		$\forall p_i\ \forall q_j$ north_east(p_i, q_j)	north_east(p'_l, q'_u)

weak_north_east(p,q)
- wne -
 	
example
wne(AU,CH)
		$\exists p_i \forall q_j$ north_east$(p_i,q_j) \wedge$
		$\exists p_i \exists q_j$ south$(p_i,q_j) \wedge$
		$\forall p_i \exists q_j$ north_east(p_i,q_j)
			north_east$(p'_u, q'_u) \wedge$
			north_east$(p'_l, q'_l) \wedge$
			south(p'_l, q'_u)

just_north(p,q)
- jn -
 	
example
jn(UK,FR)
		$\forall p_i\ \forall q_j$ (north$(p_i,q_j) \vee$
		same_level$(p_i,q_j)) \wedge$
		$\exists p_i\ \exists q_j$ same_level$(p_i,q_j) \wedge$
		$\exists p_i\ \exists q_j$ north(p_i,q_j)
			same_level(p'_l, q'_u)

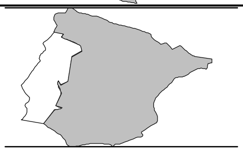
north_south(p,q)
- ns -
 	
example
ns(SP,PO)
		$\exists p_i \forall q_j$ north$(p_i,q_j) \wedge$
		$\exists p_i \forall q_j$ south(p_i,q_j)
			north$(p'_u, q'_u) \wedge$
			south(p'_l, q'_l)

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

In the next sections we will show how the above results can be applied to indexing techniques available in commercial DBMSs. The retrieval of direction relations in existing DBMSs could be accomplished by maintaining traditional indexes (e.g., B-trees[1]), or, alternatively, by incorporating Abstract Data Types (ADTs) with specialised indexes defined by external code (e.g., R-trees). Furthermore, when using extended-relational systems, like Postgres [Ston86], both indexing methods are available (or easily included) and application developers can decide which is the most appropriate for their application needs.

4. IMPLEMENTATION 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. Each index corresponds to one of the four numbers: $p'_{l\text{-}x}$, $p'_{l\text{-}y}$, $p'_{u\text{-}x}$, $p'_{u\text{-}y}$, where $p'_{l\text{-}x}$ stands for the x-coordinate of the lower point of p', $p'_{l\text{-}y}$ 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. For instance, the query "find all objects p that are *strong_north* of object q" is transformed to the constraint $north(p'_l, q'_u)$ or, in other words, $p'_{l\text{-}y} > q'_{u\text{-}y}$, which is a simple range query in the corresponding B-tree. On the other hand, other queries (such as *strong_north_east*) need to search two or more B-trees and, in a second phase, to compute the intersection of the intermediate answer sets. Table 2 presents the constraints needed for the retrieval of each direction relation using the set of four B-trees.

Relation	**Constraints on the $p'_{l\text{-}x}$, $p'_{l\text{-}y}$, $p'_{u\text{-}x}$, $p'_{u\text{-}y}$ parameters**	**# of constraints**
strong_north(p,q)	$p'_{l\text{-}y} > q'_{u\text{-}y}$	1
weak_north(p,q)	$(q'_{l\text{-}y} < p'_{l\text{-}y} < q'_{u\text{-}y}) \wedge (p'_{u\text{-}y} > q'_{u\text{-}y})$	2
strong_bounded_north(p,q)	$(p'_{l\text{-}y} > q'_{u\text{-}y}) \wedge (q'_{l\text{-}x} < p'_{l\text{-}x} < q'_{u\text{-}x}) \wedge (q'_{l\text{-}x} < p'_{u\text{-}x} < q'_{u\text{-}x})$	3
weak_bounded_north(p,q)	$(q'_{l\text{-}y} < p'_{l\text{-}y} < q'_{u\text{-}y}) \wedge (p'_{u\text{-}y} > q'_{u\text{-}y}) \wedge (q'_{l\text{-}x} < p'_{l\text{-}x} < q'_{u\text{-}x}) \wedge (q'_{l\text{-}x} < p'_{u\text{-}x} < q'_{u\text{-}x})$	4
strong_north_east(p,q)	$(p'_{l\text{-}y} > q'_{u\text{-}y}) \wedge (p'_{l\text{-}x} > q'_{u\text{-}x})$	2
weak_north_east(p,q)	$(q'_{l\text{-}y} < p'_{l\text{-}y} < q'_{u\text{-}y}) \wedge (p'_{u\text{-}y} > q'_{u\text{-}y}) \wedge (p'_{l\text{-}x} > q'_{l\text{-}x}) \wedge (p'_{u\text{-}x} > q'_{u\text{-}x})$	4
just_north(p,q)	$p'_{l\text{-}y} = q'_{u\text{-}y}$	1
north_south(p,q)	$(p'_{u\text{-}y} > q'_{u\text{-}y}) \wedge (p'_{l\text{-}y} < q'_{l\text{-}y})$	2

Table 2 Constraints for the retrieval of direction relations using 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:

[1]To be more precise, 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.

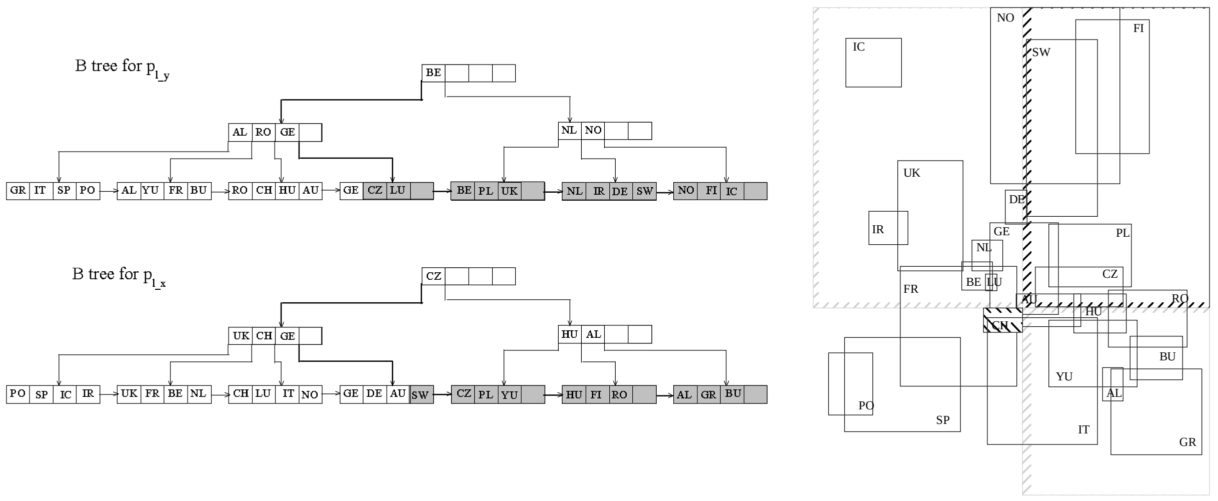
Step 1. Depending on the relation to be retrieved, select the B-trees involved from the set of four indexes. This procedure involves Table 2.

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[2].

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. They are illustrated in Figure 5a where each label indicates the appropriate coordinate for the corresponding object p, namely p'_{l-y} and p'_{l-x}. 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 3b). 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

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 constraints that are involved in the definition of the relation. *Just_north*, for example, involves only one *exact matching* constraint ($p'_{l-y} = q'_{u-y}$), while *weak_bounded_north* contains four *partial matching* constraints ($q'_{l-y} < p'_{l-y} < q'_{u-y}$) $\wedge$ ($p'_{u-y} > q'_{u-y}$) $\wedge$ ($q'_{l-x} < p'_{l-x} < q'_{u-x}$) $\wedge$ ($q'_{l-x} < p'_{u-x} < q'_{u-x}$). As it will be shown in section 7 this fact makes *just_north* one order of magnitude more efficient than *weak_bounded_north* to process. In the next section we describe another implementation based on R-trees which is more suitable for relations involving a large number of constraints.

[2]A "realistic" assumption is that this procedure is executed in main memory.

5. IMPLEMENTATION 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 this paper we use R*-trees because we found them to have consistently better performance in the retrieval of direction relations than both R- and R^+- trees.

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 3 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 $north(P_u, q'_u)$ should also satisfy the same constraint. This conclusion is applicable to all direction relations.

Relation	**Intermediate Nodes P to be Searched**
strong_north(p,q)	$north(P_u, q'_u)$
weak_north(p,q)	$north(P_u, q'_u) \wedge south(P_l, q'_u)$
strong_bounded_north(p,q)	$north(P_u, q'_u) \wedge west(P_l, q'_u) \wedge east(P_u, q'_l)$
weak_bounded_north(p,q)	$north(P_u, q'_u) \wedge south(P_l, q'_u) \wedge west(P_l, q'_u) \wedge east(P_u, q'_l)$
strong_north_east(p,q)	$north(P_u, q'_u) \wedge east(P_u, q'_u)$
weak_north_east(p,q)	$north(P_u, q'_u) \wedge south(P_l, q'_u) \wedge east(P_u, q'_u)$
just_north(p,q)	$north(P_u, q'_u) \wedge (south(P_l, q'_u) \vee samelevel(P_l, q'_u))$
north_south(p,q)	$north(P_u, q'_u) \wedge south(P_l, q'_l)$

Table 3 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 3.

Step 2. Among the leaf nodes retrieved, select the ones that satisfy the direction relation. This procedure involves Table 1.

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

Figure 6 shows how the MBRs of Figure 4 are grouped and stored in an R-tree. We assume a branching factor of 4, i.e., each intermediate node contains at most four entries. 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 6a). 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'.

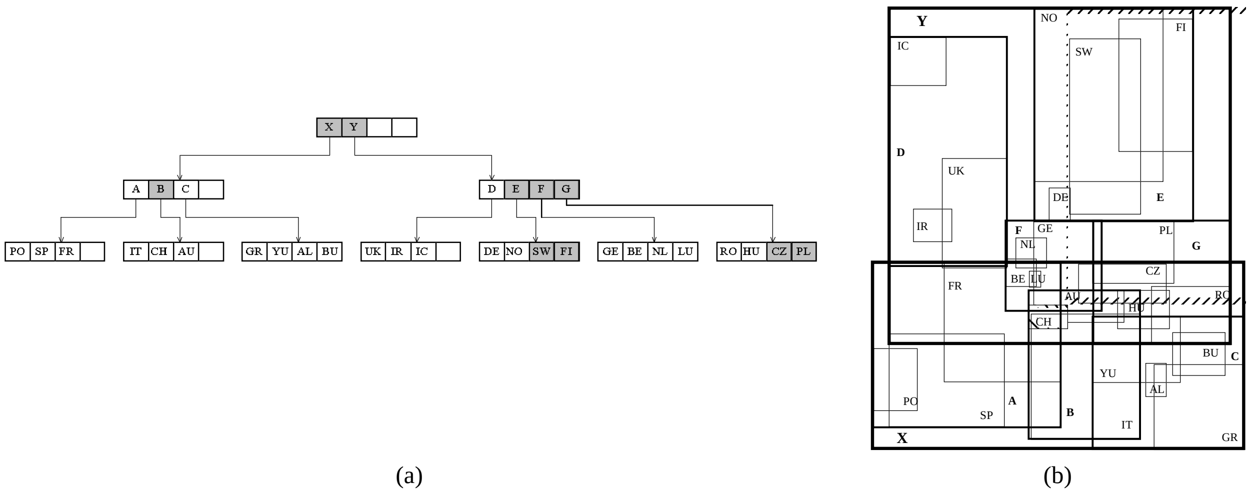


Fig. 6 Retrieval of relation strong_north_east using R-trees

Intuitively R-trees perform better than B-trees in cases where many constraints are involved in the definition of the direction relation of interest. The implementation of section 7 demonstrates that this is actually the case when four constraints are involved, while for one constraint B-trees have a better performance. For the intermediate relations though (two or three constraints) there is no clear winner and modified versions of the two data structures can yield better results. In the next section we present extensions of B-trees and R -trees that facilitate efficiency for some types of queries.

6. EXTENSIONS OF B-TREES AND R-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 4, 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-x}-tree that keeps p'_{l-x} as primary component and p'_{u-x} as secondary component, and, similarly, B_{u-x}-tree, B_{l-y}-tree and B_{u-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-y}-tree (condition to be fulfilled: $q'_{l-y} < p'_{l-y} < q'_{u-y}$; refinement condition: $p'_{u-y} > q'_{u-y}$) or a B_{u-y}-tree (condition to be fulfilled: $p'_{u-y} > q'_{u-y}$; refinement condition: $q'_{l-y} < p'_{l-y} < q'_{u-y}$).

The processing of a direction relation query involves the four steps described in section 4. 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 MBRs distribution and size). We propose 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-y}-tree is chosen for *strong_north* and (arbitrarily for) *north_south* relations.
- *Selection based on the location and size of the query object:* The query object divides each axis of the work space into five ranges: $A = (-\infty, q_l)$; $B = q_l$; $C = (q_l, q_u)$; $D = q_u$; $E = (q_u, +\infty)$. 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-y}-tree is chosen for *north_south* relation only if range A is shorter that range E; otherwise the B_{u-y}-tree is chosen.
- *Selection based on the location and size of the query object and the distribution of MBR corners:* This scheme may select the most effective index. This 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 three schemes have been implemented and tested (see [Stef95] for detailed experimental results). As a general conclusion, the third scheme outperforms the other two schemes. The second scheme may provide an improvement in performance when the data set consists of small MBRs and, as a consequence, the assumption of the uniform distribution of MBR corners over the space is true.

On the other hand, 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*, *north_south*, etc., 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.

To obtain a more efficient retrieval of direction relations that involve only one axis of the space, two one-dimensional R-trees are needed to index the MBR extents over each axis separately. The set of the two trees can also support the retrieval of direction relations that involve both axes, such as *strong_north_east*. 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.

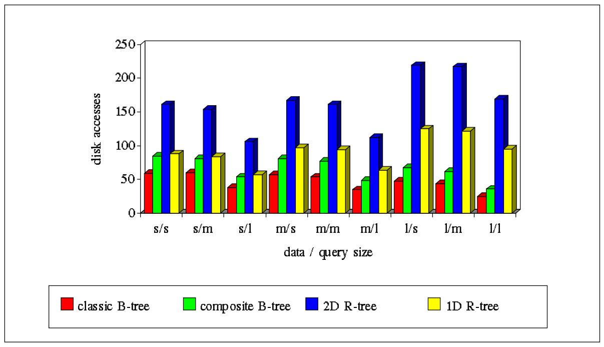
After the description of the alternative implementations of direction relations in practical systems using several indexing techniques we compare the efficiency of retrieval. It is not easy to claim a-priori which solution is the best for each direction relation but it is more sensible to claim that particular structures are more suitable for some queries.

7. PERFORMANCE COMPARISON

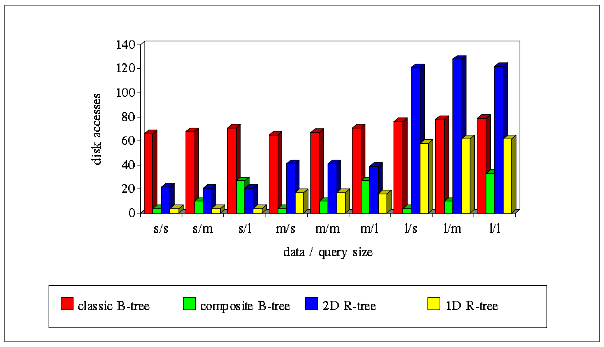
In the previous sections we have argued that the performance of each procedure depends significantly on the particular direction relation. In this section we present several experimental results that justify our argument. In order to experimentally quantify the performance, we created tree structures by inserting 10000 randomly generated MBRs. We tested three *data files*:

- the first file contains *small* MBRs: the size of each rectangle is at most 0.02% of the global area
- the second file contains *medium* MBRs: the size of each rectangle is at most 0.1% of the global area
- the third file contains *large* MBRs: the size of each rectangle is at most 0.5% of the global area.

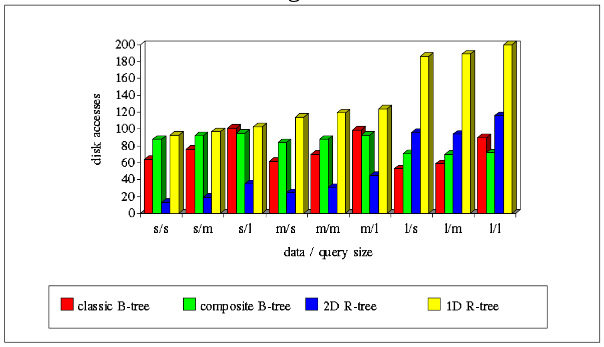
The search procedure used three *query files* for each data file containing 100 rectangles, also randomly generated, with similar size properties as the data rectangles. We used the previous data files for the retrieval of direction relations in classic B-trees, composite B-trees, 2D R-trees and 1D R-trees. The performance of each structure per relation (for each data / query size combination) is illustrated in Figure 7.



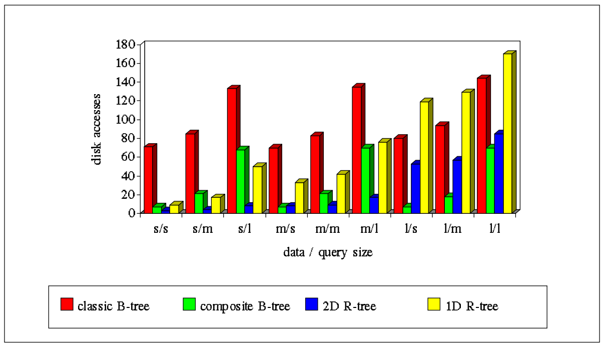
strong_north



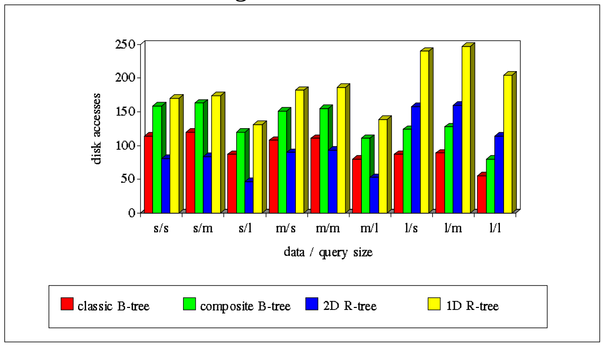
weak_north



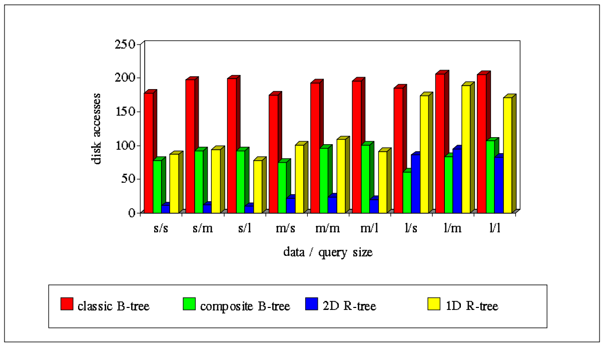
strong_bounded_north



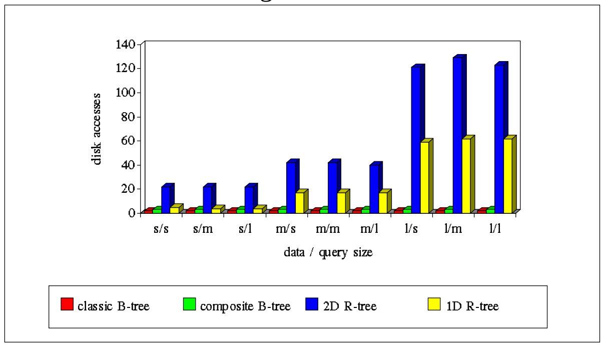
weak_bounded_north



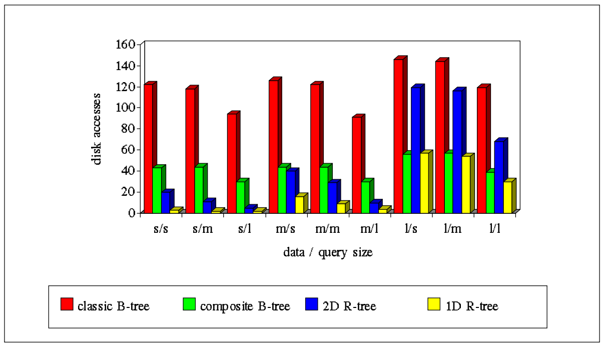
strong_north_east



weak_north_east



just_north



north_south

Fig. 7 Performance Comparison

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 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 results extracted by Figure 7 are the following:

- The classic B-tree is the most efficient structure for 1-constraint relations only (e.g., *strong_north*, *just_north*). In that case the classic B-tree outperforms even the (dedicated) composite B-tree because of its higher compactness[3].When more constraints (even for the same axis) are involved it is not competitive.
- The composite B-tree is the most efficient method for large data when more than one constraint is involved (e.g., *weak_north* and *weak_bounded_north* for large data files), since R-trees are unable to index such data efficiently. On the other hand, it is sensitive to query size; large queries are not handled efficiently by composite B-trees.
- The two-dimensional R-tree is the winner when three or four constraints are involved (see *strong_bounded_north*, *weak_bounded_north*), assuming that data rectangles are not large. In such cases, R-trees are not able to organise the data efficiently.
- The one-dimensional R-tree is the winner when two constraints along the same axis are involved (see *weak_north* relation), with the exception of large data files because of the reason mentioned above.

From the above results, it is evident that there is not an overall winner but the performance of the structures is affected significantly by the number of constraints involved, the size of the data rectangles and the query size. However, the above results can be used as guidelines to spatial query processors and optimisers in order to select appropriate indexes for use in answering spatial queries.

8. CONCLUSION

This paper describes implementations of direction relations in Spatial DBMSs. Direction relations constitute a new type of query for spatial access methods which so far have been concerned with disjoint/overlap relations [Same89], topological relations of high resolution [Papa95b] and distance relations such as nearest neighbour queries [Rous95]. 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 define direction relations between points and we use these definitions as a basis for relations between objects. For the purposes of this paper we use a set of eight object relations, but a large number of additional ones can be defined using the point relations. Then we show how these relations can be retrieved in existing DBMSs using B-trees and R-trees. We also propose extensions of these data structures, called composite B-trees and 1D R-trees, to facilitate efficiency for some types of queries.

[3]The capacity of leaf-nodes in the classic B-tree is 126 keys while composite B-trees keep 84 keys in the leaf-nodes.

For the implementation of the proposed techniques we used data and query files of different sizes. The main conclusion that arises from the experimental comparison of the alternative techniques is that there is not a data structure that performs better in all queries but the performance depends on the following factors:

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

It is possible that in actual systems, two or more of the previous structures can be used alternatively in conjunction with an optimiser that chooses the most suitable one according to the input query.

Future work can be done on the definition and implementation of direction relations. Experimental findings from Cognitive and Environmental Psychology[4] can be used as guidelines for the direction relations that people evoke in everyday reasoning. Although so far the psychological results are too vague to be helpful in defining direction relations in actual systems, ongoing research can lead to a set of well defined and psychologically sound relations. Note that such a set exists for topological relations [Egen90]. Progress can also be achieved in specialised data structures that have a better performance for queries involving direction relations or combinations of several types of spatial information (e.g., “find the five closest buildings in the area northeast of the (burning) factory”.

REFERENCES

[Beck90] Beckmann, N., Kriegel, H.P., Schneider, R., Seeger, B., “The R*-tree: an Efficient and Robust Access Method for Points and Rectangles”, In the Proceedings of ACM-SIGMOD Conference, 1990.

[Egen90] Egenhofer, M., Herring, J., “A Mathematical Framework for the Definitions of Topological Relationships”, In the Proceedings of the 4th International Symposium on Spatial Data Handling (SDH), 1990.

[Egen93] Egenhofer, M., Sharma, J., “Assessing the Consistency of Complete and Incomplete Topological Information”, Geographical Systems, Vol. 1, pp. 47-68, 1993.

[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”, to appear in the International Journal of Geographic Information Systems.

[Glas92] Glasgow, J.I., Papadias, D., “Computational Imagery”, Cognitive Science, Vol. 16, pp. 355-394, 1992.

[Gutt84] Guttman, A., “R-trees: a Dynamic Index Structure for Spatial Searching”, In the Proceedings of ACM-SIGMOD Conference, 1984.

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

[Hers86] Herskovits, A., “Language and Spatial Cognition”, Cambridge University Press, 1986.

[Jack83] Jackendoff, R., “Semantics and Cognition”, MIT Press, 1983

[4]A survey and an experimental study regarding the use of direction relations in cognitive spatial reasoning at geographic scales can be found in [Mark92].

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

[Mark92] Mark, D., “Counter-Intuitive Geographic 'Facts': Clues for Spatial Reasoning at Geographic Scales”, In the Proceedings of International Conference GIS - From Space to Territory: Theories and Methods of Spatio-Temporal Reasoning in Geographic Space, 1992.

[Muke90] Mukerjee, A., Joe, G., “A Qualitative Model for Space”, Technical Report TAMU 90-005, Texas A&M University, 1990.

[Papa94a] Papadias, D., Frank, A.U., Koubarakis, M., “Constraint-Based Reasoning in Geographic Databases: The Case of Symbolic Arrays”, In the 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”, In the Proceedings of the 5th Conference on Database and Expert Systems Applications (DEXA), 1994.

[Papa94c] Papadias, D., Sellis, T., “The 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”, to appear in the Journal of Visual Languages and Computing, Special issue on Visual Query Systems.

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

[Peuq87] Peuquet, D., Ci-Xiang, Z., “An Algorithm to Determine the Directional Relationship between Arbitrarily-Shaped Polygons in the Plane”, Pattern Recognition, Vol. 20(1), 1987, pp. 65-74.

[Rous95] Roussopoulos, N., Kelley, F., Vincent, F., “Nearest Neighbor Queries”, In the Proceedings of ACM-SIGMOD Conference, 1995.

[Same89] Samet, H., “The Design and Analysis of Spatial Data Structures”, Addison-Wesley, 1989.

[Sell87] Sellis, T., Roussopoulos, N., Faloutsos, C., “The R$^+$-tree: A Dynamic Index for Multi-Dimensional Objects”, In the Proceedings of the 13th Very Large Data Bases Conference, 1987.

[Stef95] Stefanakis, E., Theodoridis, Y., “B-trees and Spatial Relations in Two-Dimensional Space”, Technical Report, KDBSLAB-TR-95-01, National Technical University of Athens, 1995.

[Ston86] Stonebraker, M., Rowe, L., “The Design of Postgres”, In the Proceedings of ACM-SIGMOD Conference, 1986.

[Sist94] Sistla, P., Yu, C., Haddad, R., “Reasoning about Spatial Relationships in Picture Retrieval Systems”, In the Proceedings of the 20th Very Large Data Bases Conference, 1994.