Geoinformatica manuscript No.
(will be inserted by the editor)

The largest empty rectangle containing only a query object in Spatial Databases

Gilberto Gutiérrez · José R. Paramá · Nieves Brisaboa · Antonio Corral

Received: date / Accepted: date

Abstract Let S be a set of n points in a fixed axis-parallel rectangle $R \subseteq \Re^2$, i.e. in the two-dimensional space (2D). Assuming that those points are stored in an R-tree, this paper presents several algorithms for finding the empty rectangle in R with the largest area, sides parallel to the axes of the space, and containing only a query point q. This point can not be part of S, that is, it is not stored in the R-tree.

All algorithms follow the basic idea of discarding part of the points of S, in such a way that the problem can be solved only considering the remaining points. As a consequence, the algorithms only have to access a very small portion of the nodes (disk blocks) of the R-tree, saving main memory resources and computation time.

We provide formal proofs of the correctness of our algorithms and, in order to evaluate the performance of the algorithms, we run an extensive set of experiments using synthetic and real data. The results have demonstrated the efficiency and scalability of our algorithms for different dataset configurations.

Keywords Spatial databases, Query, Indexing methods

A preliminary partial version of this work appeared in [1].

Gilberto Gutiérrez (✉)
Universidad del Bío-Bío, Computer Science and Information Technologies Department Chillán, Chile
E-mail: ggutierr@ubiobio.cl

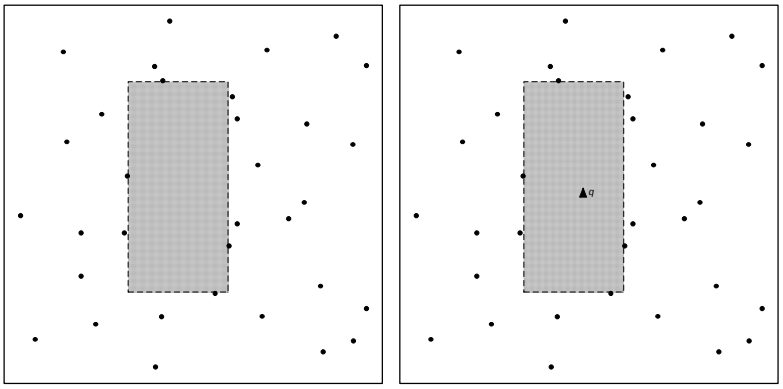
José R. Paramá · Nieves Brisaboa
University of A Coruña, Computer Science Department A Coruña, Spain
E-mail: {jose.parama,brisaboa}@udc.es

Antonio Corral
University of Almeria, Department of Languages and Computation Almeria, Spain
E-mail: acorral@ual.es

1 Introduction

In computational geometry, there is a research line that is aimed at finding empty geometric figures in a space that contains a set of points. For example, one of them is to find the largest empty axis-parallel rectangle in a space containing a set of points (see Figure 1(a)). A variant of the previous problem is to find the largest rectangle that only contains a given query point, assuming that the query point does not belong to the set of points in the space (see Figure 1(b)). This work deals with the latter variant.

More variants of this problem are those that find a circumference, a square, or a convex hull. In addition, in the case of rectangles and squares, another alternative is to consider figures with sides that are not parallel to the axes.



(a) Largest empty rectangle.

(b) Largest empty rectangle containing only a query point (the triangle point is the query point).

Fig. 1 Two variants of the problem of finding the largest empty rectangle.

The search for empty geometric figures with the largest area, or any other metric, has applications in several fields. Among them, we can cite very-large-scale integration design (VLSI), database management, operations research, wireless sensor network [2], geographical information systems (GIS) [3], and data mining [4].

As an example of application of the largest empty rectangle containing a query point in the GIS field, consider a entrepreneur trying to find a location for her/his business (restaurant, coffe shop, etc.). If she/he has a list of candidate positions for her/his business (available and affordable locals), she/he can provide those positions as the query points to a GIS system, which has the locations of business of the same sector as points. After running the queries, she/he obtains the largest empty rectangle around each query point, and then, she/he can choose a local with a large empty area around the query point, where the position of the local (query point) is quite centered in the

empty area. This means that the site has a considerable area around without competitors and, therefore, a high probability of having customers for whom the closest option is her/his business. Note that, the largest empty rectangle (without location constraints) would be in an area where the locals are too expensive, or in an area too far from her/his home, or any other constraint.

Another classic application would be when we have several candidate placements for a facility that implies risks (for example, a nuclear plant or a potential toxic chemical plant) [3]. If we have the location of human settlements as points and we provide those candidate sites for the facility as query points, we can choose a location with a large empty area around, and where the query point is quite centered. Observe that, if we do not use the query point, the largest empty rectangle can be in a foreign country, in a non appropriate geological soil, or in the sea (assuming that the space limits include one or more peninsulas).

Finally, in the case of VLSI, one might be interested in placing a large circuit close to a given component [2,5].

The spatial databases (SDBs) represent an important aid for GIS to manage large amounts of data. However, SDBs require the design of new data structures, spatial access methods, query languages, and algorithms to manage spatial information. In this sense, several new query types have been defined, among others, the window query, the intersection query, the nearest neighbor, and the spatial join [6,7]. Many of those query types are problems that were first tackled in the field of the computational geometry, where it is assumed that all spatial objects can be fit into main memory, and later, those problems were faced in the field of the SDBs. Following this path, several algorithms have been proposed considering that objects are stored in a multidimensional structure, in most cases an R-tree [8], for example; [9,10] present several algorithms that solve the k-pairs ($k \geq 1$) of nearest neighbors between two sets, [11] shows an algorithm to find the nearest neighbor to a given point, and [12] presents an algorithm to obtain the convex hull of a set of points stored in an R-tree.

The problem of finding the largest axis-parallel rectangle that only contains a given query point (see Figure 1(b)) has also been treated in the area of computational geometry. In this work, we face this problem from the perspective of SDBs, assuming that the points are stored in an R-tree. We present three basic algorithms and two combinations of them. All algorithms are based on the same idea, they extract a portion of the points in the space by taking advantage of the presence of an R-tree. Then, using only those points, a conventional computational geometry algorithm obtains the same result as if it were applied over the whole set of points. The idea is that the extracted set is much smaller than the whole set of points, and thus, the computation is faster and needs less main memory.

These algorithms take advantage of the extensive use of the R-tree in commercial database management systems [13] (Oracle [14], PostgreSQL [15], etc.), extending the usefulness of that data structure.

The first algorithm, called $q-$MER, was presented in [1]. The other two, called $q-\text{MER}_{D1}$ and $q-\text{MER}_{D2}$, are presented for the first time in this work. Since depending on the distribution of the data provided as input, one algorithm performs better than the others, in this paper we also present two new heuristics that choose the best algorithm for a given query and input data. In addition, this work present a more detailed empirical study with respect to that in [1].

We have run a set of experiments to evaluate the performance of our algorithms over large real and synthetic datasets. The results have demonstrated the efficiency of our algorithms in terms of pruning unnecessary branches on the R-tree in order to find the final result of the query and scalability for different dataset configurations. Moreover, our algorithms required less response time and storage resources than the naive approach of reading all the points from disk, store them in main memory, and use a computational geometry algorithm to obtain the result.

The outline of the paper is as follows: Section 2 presents some previous related work. Section 3 presents some basics and definitions. Section 4 presents our three basic algorithms and the two heuristics. Section 5 shows the results of our experiments. Finally, Section 6 shows our conclusions and directions for future work.

2 Related work

Given a set S of n points in a fixed axis-parallel rectangle $R \subseteq \Re^2$, the search for the largest empty geometric figure (circumference, square, rectangle, or convex hull) has been an active research field in last decades. Focusing on the problem of finding the largest rectangle with sides parallel to the axes of the space, two variants have been considered: (i) no information about the position of the figure is provided (see Figure 1(a)), and (ii) information about the position is provided (see Figure 1(b)).

The first variant has been extensively studied. The first work is [16], where two algorithms are described: the first one takes $O(n^2)$ time and $O(n)$ space, the second one takes $O(n \log^2 n)$ expected time considering that the points are randomly arranged into the space. Later, Chazelle et al. [17], present a divide-and-conquer algorithm with $O(n \log^3 n)$ time complexity using $O(n \log n)$ space. An algorithm with similar time complexity is discussed in [18], this one using $O(n)$ space. Orlowski presented an algorithm [19] that takes $O(s \log n)$ time, where s is the number of maximal empty rectangles (or restricted rectangles). A *maximal empty rectangle* (MER) is a rectangle that (i) its edges are parallel to the axes of R, (ii) it lies wholly in R, (iii) no point of S is contained in its interior, and (iv) each edge contains, at least, a point of S or is contained in an edge of R. Moreover, that algorithm has an expected time $O(n \log n)$. A more recent approach [20] takes $O(n \log^2 n + s)$ time and $O(\log n)$ space by using a priority search tree.

There are also works in 3D, in this case the algorithms compute the largest empty axis-parallel cuboid [21,22].

The second variant was proposed in [23,24]. This algorithm performs a preprocessing step where the space is divided into a set of cells such that all points that fall in the same cell produce the same maximal empty rectangle containing the query point. These cells are stored in main memory organized into a data structure for objects in the two-dimensional space called range tree. The preprocessing stage takes storage $O(n^2 \log n)$ and time $O(n^2)$. To retrieve the MER corresponding to a query point q, an additional $O(\log n)$ time is needed. Another approach was presented in [5], this corresponds to a significant improvement in terms of preprocessing time and space with respect to that in [23,24]. Specifically, this algorithm requires $O(n\alpha(n)\log^3 n)$ storage to maintain the data structure (a segment tree) and $O(n\alpha(n)\log^4 n)$ time to build the structure, where the term $\alpha(n)$ is the slowly increasing inverse Ackermann function. Yet, the query time to find the MER that only contains q increases to $O(\log^4 n)$.

However the search of a circle with location constraints is an old problem. There are several variants: largest circumference only containing a query point [25,2], a query line [26], or even a $n - gon$ [3,27].

All the algorithms commented so far assume that the objects can be fit into main memory. Edmonds, et al. [4] face the problem of finding all the empty spaces left by a set of objects, assuming that the main memory does not have enough space to store all the objects. That algorithm takes $O(|X||Y|)$, where X e Y are the distinct values of the coordinates of the dataset. Yet, this work does not consider the case where the objects are stored in a multidimensional structure.

To the best of our knowledge, this problem has not been tackled in the field of SDBs. The closest problem might be the skyline query (which given a set of points S, the query returns all points in S that are not *dominated* by another point) [28,29] and its variations (dynamic and reverse skyline queries) [30]. Observe that the dominant points of the skyline query define an empty area that includes the origin of the space (0, 0). However, our empty areas do not have a skyline shape, but a simple corner of a city block, yet in four directions (northeast, northwest, southwest, and southeast) from the query point. Therefore our problem is related to the skyline problem, since both use the dominance relationship between elements stored in the R-tree (points and MBRs (Minimum Bounding Rectangles)). Furthermore, in order to speed up execution times, indexes (as R-tree) have been also used to avoid processing part of the points [28,29].

3 Preliminaries

3.1 R-tree

An R-tree is a generalization of B^+-trees designed for the dynamic indexation of a set of k-dimensional geometric objects. It is a hierarchical, height balanced multidimensional data structure, designed to be used in secondary storage. In inner levels the indexed objects are represented by the Minimum Bounding k-dimensional Rectangles (MBRs), which bound their children. In this paper, we focus on 2 dimensions, therefore these MBRs are rectangles with faces parallel to the coordinate axis and characterized by two points (p_{min} and p_{max}). By using the MBR instead of the exact geometrical representation of the object, its representational complexity is reduced to two points where the most important features of the spatial object (position and extension) are maintained. Consequently, the MBR is an approximation widely employed, and the R-trees belong to the category of data-driven access methods, since their structure adapts itself to the MBRs distribution in the space.

Definition 1 An MBR is a rectangle, which faces are parallel to the axis of the coordinate system and is characterized by two points p_{min} and p_{max}, where $p_{min} = (p_{min}.x,\ p_{min}.y)$ and $p_{max} = (p_{max}.x,\ p_{max}.y)$, such that $p_{min}.x \leq p_{max}.x$ and $p_{min}.y \leq p_{max}.y$. They correspond to the end-points of one of its major diagonals (e.g. the lower-left corner and the upper-right corner of such rectangle).

An R-tree satisfies the following rules. Leaves are on the same level. In our case, each leaf node contains the indexed real points, although it may contain pointers to the objects of the database together with their MBRs. Every inner node contains entries of the form $\langle MBR, ref \rangle$, where ref is a pointer to the child of the entry and MBR is the MBR that contains spatially the MBRs (or points if the child is a leaf node) contained in this child. An R-tree of class $(m; M)$ has the characteristic that every node, except possibly for the root, contains between m and M entries, where $m \leq \lceil M/2 \rceil$. The root contains at least two pairs. The R-tree nodes are implemented as disk pages.

Figure 2 depicts some MBRs, the indexed points, and the corresponding R-tree. Dotted lines denote the MBRs of the entries at the root node. The rectangles with solid lines are the MBRs in the entries of parent nodes of the leaves. Finally, points are the indexed objects in the leaves.

We consider that the leaf nodes are at level 0 and the root is at level h, that is, the tree has $h + 1$ levels.

Next we present two important properties of the MBRs of R-trees [11]:

Property 1 MBR enclosure property: This property establishes that an MBR of an R-tree entry always encloses the MBRs of its descendant entries.

Property 2 MBR face property: This property establishes that every face of any MBR of an R-tree node (at any level) touches at least one point of some spatial object in the spatial database.

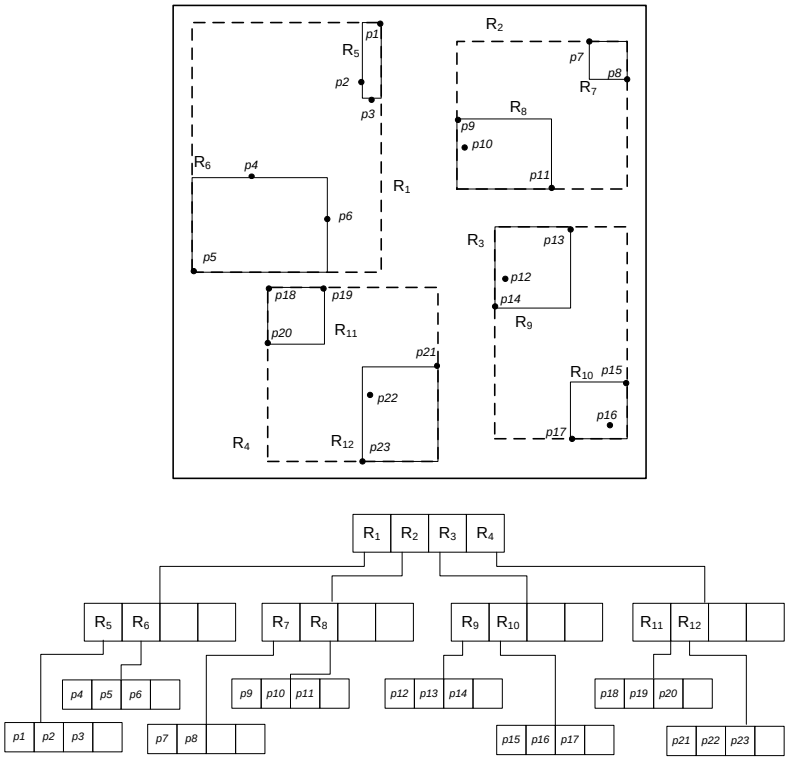


Fig. 2 An R-tree.

Many variations of R-trees have appeared in the literature (an exhaustive survey can be found in [13]). One of the most popular and efficient variations is the R*-tree [31]. The R*-tree added two major enhancements to the R-tree, when a node overflow is caused. First, rather than just considering the area, the node-splitting algorithm in R*-tree also minimized the perimeter and overlap enlargement of the minimum bounding rectangles. Minimizing the overlap tends to reduce the number of subtrees to follow for search operations. Second, R*-tree introduced the notion of *forced reinsertion* to make the shape of the tree less dependent to the order of insertions. When a node becomes overflowed, it is not split immediately, but a portion of entries of the node is reinserted from the top of the tree. The reinsertion provides two important improvements First, it can reduce the number of splits needed and; second, it is a technique for dynamically reorganizing the tree. With these two enhancements, the R*-tree generally outperforms R-tree and it is commonly accepted that the R*-tree is one of the most efficient R-tree variants. In this paper, we have chosen the R*-tree to perform our experimental study.

3.2 Definitions and properties

Next, we introduce some definitions and properties that will be used later.

Definition 2 Let S be a set of points in a fixed axis-parallel rectangle $R \subseteq \Re^2$ (in the plane) and a query point $q \notin S$ such that $q \cap R \neq \emptyset$. An axis-parallel rectangle is called a maximal empty rectangle only containing the query point (QMER) if:

(i) it contains the query point,
(ii) its edges are parallel to the axes of R,
(iii) it lies wholly in R,
(iv) no point of S is contained in its interior, and
(v) each edge contains, at least, a point of S or is contained in a edge of R.

Here, the notion of an axis-parallel rectangle assumes that its geometric structure is bounded by four sides and thus, a QMER in 2D can be given by two points: its lower-left corner and its upper-right corner (as an MBR).

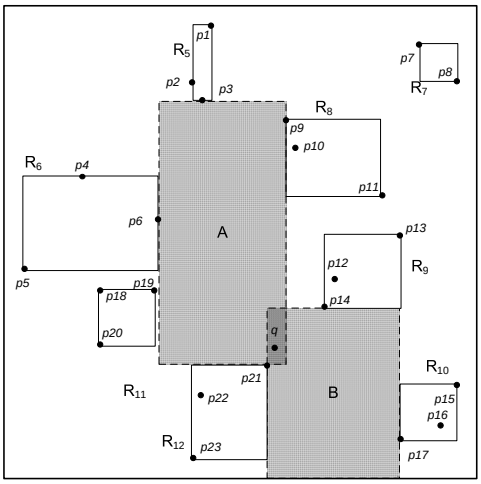


Fig. 3 Two QMERs.

A QMER is a rectangle only containing the query point that can not be enlarged, either due to edges contained in the space limits or edges that contain a point. Figure 3 displays two QMERs (marked with letters A and B), there are many others, but in order to avoid an overloaded figure, only those QMERs are shown.

Observe that the QMER B can not grow in any direction: to the south, the space ends; to the west, the QMER can not be enlarged, otherwise B would contain the point p_{21}; to the north, the QMER founds a barrel in p_{14}; and finally to the east, p_{17} represents an obstacle to the growth of B. This does not mean that B is the largest empty rectangle containing q, in fact A is larger.

In order to obtain the largest empty rectangle, all computational geometry algorithms compute MERs (QMERs in the variants with query point) somehow, and then they search the largest MER (QMER), which is the desired result.

Definition 3 Given a set of points S in a fixed axis-parallel rectangle $R \subseteq \Re^2$ stored in an R-tree and a query point $q \notin S$ such that $q \cap R \neq \emptyset$. A candidate empty rectangle containing the query point (QCER) is a QMER computed using a set of points (called C) that are obtained from the MBRs of the R-tree, instead of using the real points in S.

QCERs are computed taking advantage of the knowledge of the position and extension of the objects in the space present in the R-tree. QCERs are equal or upper bounds of real QMERs, this allows the $q-$MER algorithm to compute only QMERs inside QCERs. By doing this, we divide the problem of computing the largest empty rectangle containing the query point over the set of points S into much smaller problems. Each one computes the largest empty rectangle containing the query point inside a QCER, which is usually much smaller than the complete space R. Although, we have to repetitively run the computational geometry algorithm (once for each QCER), since those executions have as input a much smaller set of points, this implies that the total execution time is lower.

Definition 4 Given a set of points S in a fixed axis-parallel rectangle $R \subseteq \Re^2$, which is stored in an R-tree, and a query point $q \notin S$ and $q \cap R \neq \emptyset$, the largest maximal empty rectangle containing the query point q ($LQMER$) is the QMER, also in R, with the largest area.

Definition 5 We say that an edge of a rectangle is *supported* by a point p, if that edge contains p.

For example, observe in Figure 3 that the upper edge of the QMER A is supported by the point p_3, since it contains that point. That is, the edge contains the point, but the point is not inside A. If an edge of a QMER is supported by a point, then such edge can not be moved in such a way that the QMER grows, otherwise, the rectangle will contain that point, and hence it would not be a QMER, since only the query point is allowed inside a QMER.

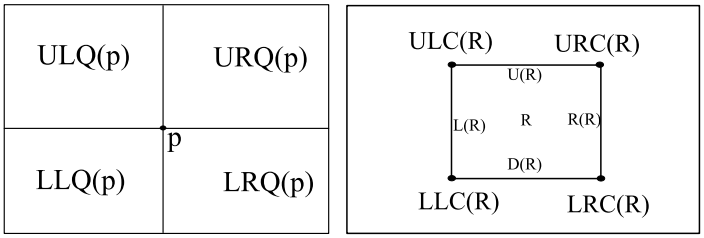
Given a rectangle $R \subseteq \Re^2$ that contains a point p, Table 1 shows some definitions (see Figure 4 for a graphic description). These definitions are auxiliary definitions that will be used along the paper.

Definition 6 Let $R \subseteq \Re^2$ be a rectangle that contains three points: a query point q, $p_i(p_i.x, p_i.y)$, and $p_j(p_j.x, p_j.y)$, being $i \neq j$:

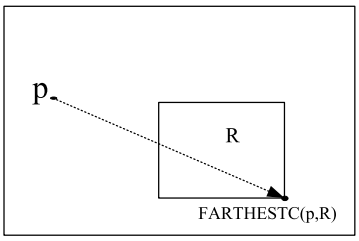
- If p_i and p_j are in $URQ(q)$. p_i *dominates* p_j in $URQ(q)$ if $p_i.x \leq p_j.x$ and $p_i.y \leq p_j.y$.
- If p_i and p_j are in $ULQ(q)$. p_i *dominates* p_j in $ULQ(q)$ if $p_j.x \leq p_i.x$ and $p_i.y \leq p_j.y$.

$ULQ(p)$	the Upper-Left *quadrant* of p is the rectangle bounded by point p and the Upper-Left corner of R
$URQ(p)$	the Upper-Right *quadrant* of p is the rectangle bounded by point p and the Upper-Right corner of R
$LLQ(p)$	the Lower-Left *quadrant* of p is the rectangle bounded by point p and the Lower-Left corner of R
$LRQ(p)$	the Lower-Right *quadrant* of p is the rectangle bounded by point p and the Lower-Right corner of R
$ULC(R)$	is the Upper-Left *corner* of R
$URC(R)$	is the Upper-Right *corner* of R
$LLC(R)$	is the Lower-Left *corner* of R
$LRC(R)$	is the Lower-Right *corner* of R
$L(R)$	the line segment that connects $ULC(R)$ with $LLC(R)$
$R(R)$	the line segment that connects $URC(R)$ with $LRC(R)$
$U(R)$	the line segment that connects $ULC(R)$ with $URC(R)$
$D(R)$	the line segment that connects $LLC(R)$ with $LRC(R)$
$FARTHESTC(p, R)$	the corner of R such that $dist(p, FARTHESTC(p, R))$ is the largest euclidean distance between p and any corner of R

Table 1 Definitions.



(a) Quadrants defined by a point. (b) Elements defined by a rectangle.

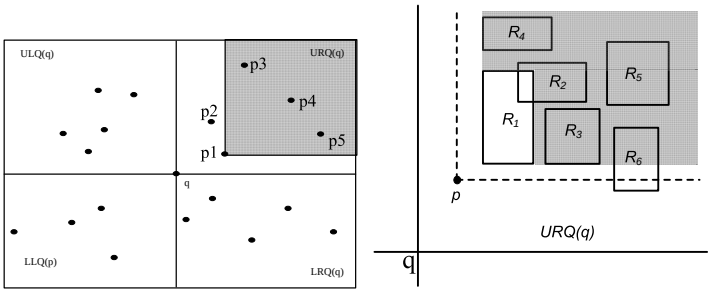


(c) A distance relationship between a rectangle and a point.

Fig. 4 Definitions.

- If p_i and p_j are in $LRQ(q)$. p_i *dominates* p_j in $LRQ(q)$ if $p_i.x \leq p_j.x$ and $p_j.y \leq p_i.y$.
- If p_i and p_j are in $LLQ(q)$. p_i *dominates* p_j in $LLQ(q)$ if $p_j.x \leq p_i.x$ and $p_j.y \leq p_i.y$.

Definition 7 Let $R \subseteq \Re^2$ be a rectangle that contains two points: a query point q and p_i.



(a) Involving points. (b) Involving points and MBRs.

Fig. 5 Dominance relationships and areas.

- If p_i is in $ULQ(q)$ the *dominance area* of p_i is the rectangle bounded by p_i and the Upper-Left corner of R
- If p_i is in $URQ(q)$ the *dominance area* of p_i is the rectangle bounded by p_i and the Upper-Right corner of R
- If p_i is in $LLQ(q)$ the *dominance area* of p_i is the rectangle bounded by p_i and the Lower-Left corner of R
- If p_i is in $LRQ(q)$ the *dominance area* of p_i is the rectangle bounded by p_i and the Lower-Right corner of R

In Figure 5(a), p_1 dominates p_3, p_4, and p_5 in the $URQ(q)$ quadrant, yet it does not dominate p_2. The grey shaded area is the dominance area of p_1 in $URQ(q)$.

Definition 8 Let $R \subseteq \Re^2$ be a rectangle that contains a query point q, a point p_i, and an MBR R_j:

- p_i *dominates* R_j in $ULQ(q)$, if p_i dominates $LRC(R_j)$.
- p_i *dominates* R_j in $URQ(q)$, if p_i dominates $LLC(R_j)$.
- p_i *dominates* R_j in $LLQ(q)$, if p_i dominates $URC(R_j)$.
- p_i *dominates* R_j in $LRQ(q)$, if p_i dominates $ULC(R_j)$.

Definition 9 Let $R \subseteq \Re^2$ be a rectangle that contains a query point q and two MBRs, R_i and R_j, being $i \neq j$:

- R_i *dominates* R_j in $ULQ(q)$, if $LRC(R_i)$ dominates R_j and $R_i \cap R_j = \emptyset$ (i.e. R_j does not overlap with R_i and R_j is dominated by the lower-right corner of R_i).
- R_i *dominates* R_j in $URQ(q)$, if $LLC(R_i)$ dominates R_j and $R_i \cap R_j = \emptyset$.
- R_i *dominates* R_j in $LLQ(q)$, if $URC(R_i)$ dominates R_j and $R_i \cap R_j = \emptyset$.
- R_i *dominates* R_j in $LRQ(q)$, if $ULC(R_i)$ dominates R_j and $R_i \cap R_j = \emptyset$.

Definition 10 Let $R \subseteq \Re^2$ be a rectangle that contains a query point q and an MBR R_i fully inside a quadrant x. The *dominance area* of R_i covers the space of x where any MBR completely within that area is dominated by R_i.

In Figure 5(b) is shown the dominance relationships between points and MBRs. p dominates all MBRs except R_6. For the MBRs, we can see in grey the dominance area of R_1 in $URQ(q)$, and we can deduce that R_1 dominates R_3, R_4, and R_5 (equivalently, R_3, R_4, and R_5 are dominated by R_1), while R_2 and R_6 are not dominated by R_1.

Dominance relationships are the key for the algorithms $q-\text{MER}_{D1}$ and $q-\text{MER}_{D2}$. Basically, when a point or an MBR dominates another point or MBR, these algorithms can discard those dominated items, since a QMER can not have edges supported by dominated points. However, the dominant elements are kept, because those points or points inside those MBRs might support the edges of a QMER.

As a remainder, Table 2 briefly describes the definitions not covered by Table 1.

$QMER$	maximal empty rectangle only containing the query point
$QCER$	candidate maximal empty rectangle, a QMER computed from a set of points C extracted from the R-tree
Supported	an edge of a rectangle is supported by a point p if that edge contains p
$LQMER$	largest maximal empty rectangle containing the query point
p_i dominates p_j in $URQ(q)$	$p_i.x \leq p_j.x$ and $p_i.y \leq p_j.y$
p_i dominates p_j in $ULQ(q)$	$p_j.x \leq p_i.x$ and $p_i.y \leq p_j.y$
p_i dominates p_j in $LRQ(q)$	$p_i.x \leq p_j.x$ and $p_j.y \leq p_i.y$
p_i dominates p_j in $LLQ(q)$	$p_j.x \leq p_i.x$ and $p_j.y \leq p_i.y$
Dominance area of p_i in quadrant $URQ(q)$	the rectangle bounded by p_i and the Upper-Right corner of R
Dominance area of p_i in quadrant $ULQ(q)$	the rectangle bounded by p_i and the Upper-Left corner of R
Dominance area of p_i in quadrant $LLQ(q)$	the rectangle bounded by p_i and the Lower-Left corner of R
Dominance area of p_i in quadrant $LRQ(q)$	the rectangle bounded by p_i and the Lower-Right corner of R
p_i dominates R_j in $ULQ(q)$	p_i dominates $LRC(R_j)$
p_i dominates R_j in $URQ(q)$	p_i dominates $LLC(R_j)$
p_i dominates R_j in $LLQ(q)$	p_i dominates $URC(R_j)$
p_i dominates R_j in $LRQ(q)$	p_i dominates $ULC(R_j)$
R_i dominates R_j in $ULQ(q)$	$LRC(R_i)$ dominates R_j and $R_i \cap R_j = \emptyset$
R_i dominates R_j in $URQ(q)$	$LLC(R_i)$ dominates R_j and $R_i \cap R_j = \emptyset$
R_i dominates R_j in $LLQ(q)$	$URC(R_i)$ dominates R_j and $R_i \cap R_j = \emptyset$
R_i dominates R_j in $LRQ(q)$	$ULC(R_i)$ dominates R_j and $R_i \cap R_j = \emptyset$
Dominance area of R_i in quadrant x	the space of x where any MBR completely within that area is dominated by R_i

Table 2 Additional definitions. p_i and p_j are points and R_i and R_j are MBRs.

4 Algorithms

In this section we present our algorithms. We show the rationale behind them and we prove their concreteness by means of the appropriate lemmas, theorems, and corollaries. Our algorithms assume that the points of S are stored in an R-tree and are based on the general idea of discarding points using the properties of R-trees and the dominance relationships defined in Section 3.2.

4.1 $q-$MER algorithm

This algorithm requires two main steps:

1. First, it computes a set of QCERs, which, in turn, requires two substeps:
   (a) *The computation of the set of points C*: QCERs are QMERs computed using as input a set of points C extracted from the R-tree, rather than computing them from the real points in S. For the computation of C, the algorithm processes the MBRs in parent nodes of the leaves of the R-tree. For each MBR, one or two points can be added to C. Those points are the most distant points to the query point q that could be located in that MBR (that is, it is likely those points are not part of the real set of points). Figure 6 displays an example. From the MBRs in parent nodes of leaves (the rectangles) and the query point q, $q-$MER produces the set of points $C = \{c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, c_{10}\}$.
   When the MBR is completely inside one quadrant, the algorithm adds only one point. For example, R_5 produces the point c_1, since it is the farthest point with respect to q that could be located in that MBR, but, c_1 is probably not part of the set of points actually stored in the R-tree. When the MBR overlaps two quadrants, two points are added. For example, the processing of R_6 produces two points c_4 and c_5, those are the most distant points in each of the two quadrants that intersect with R_6: c_4 is the farthest point with respect to q that could be located in the part of R_6 in the $ULQ(q)$ quadrant, and c_5 the one with the same properties in the $LLQ(q)$ quadrant. The rest of points are computed similarly.
   (b) *The actual computation of the QCERs*: A computational geometry algorithm is run (in our experiments, we used the Orlowski's algorithm [19]) using the set C as input. We use the variant of the computational geometry algorithm that obtains all QMERs, instead of only computing the largest one, that is, in our case, we obtain all the QCERs. Figure 7(a) shows a QCER (again, there are others, but to simplify the figure, we only show one), denoted as A'. The points of C are the barrels that do not allow to expand A' in any direction. Figure 7(b) shows a real QMER A overlaid over the QCER A' and the real points. Observe that A' is an upper bound of A since is computed using the farthest points with respect to q that could be located in the MBRs of the R-tree.

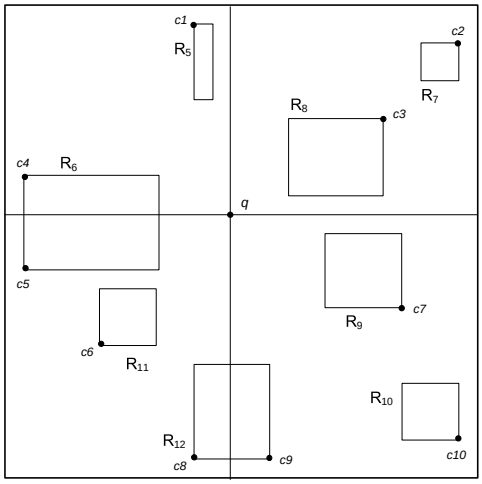
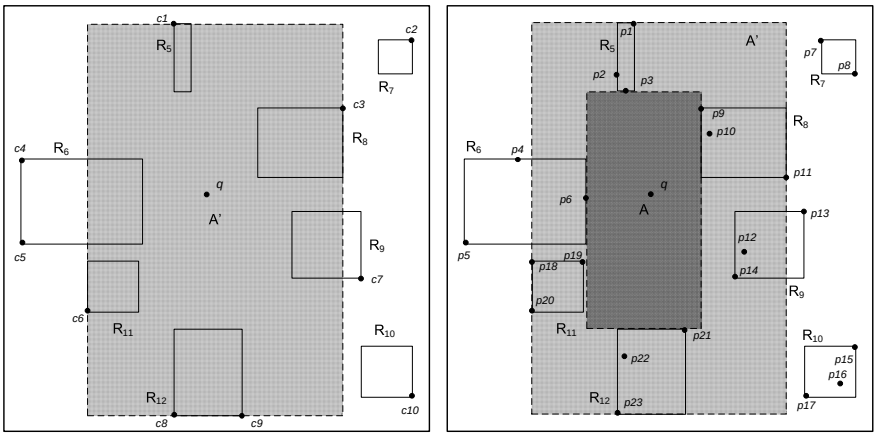


Fig. 6 The set of points C.



(a) A QCER.

(b) A QCER (A') and its corresponding QMER (A).

Fig. 7 The computation of QCERs.

2. In the second step, QCERs are processed according to their area, from largest to smallest. For each QCER, our algorithm accesses the leaves of the R-tree that contain the real points that intersect with such a QCER. Those real points, that we call C', are used to obtain a *candidate solution*, by means of the same computational geometry algorithm used to obtain the QCERs, this time with the variant that computes only the largest QMER. This candidate solution is the real largest empty rectangle containing the query point that is equal to or contained into the processed QCER. As the processing of QCERs progresses, the candidate solutions may improve previous ones. For example, when the QCER A' of Figure 7 is processed,

it is necessary to access the children of the entries containing the MBRs R_5, R_8, R_6, R_9, R_{11}, and R_{12}. Then $q-$MER inserts in C' the points that intersect with A', and processes C' with the computational geometry algorithm. Finally, if the obtained candidate solution (A) is better than the previous ones, then it passes to be considered the best candidate solution found so far.

Next, we describe in detail each of these steps.

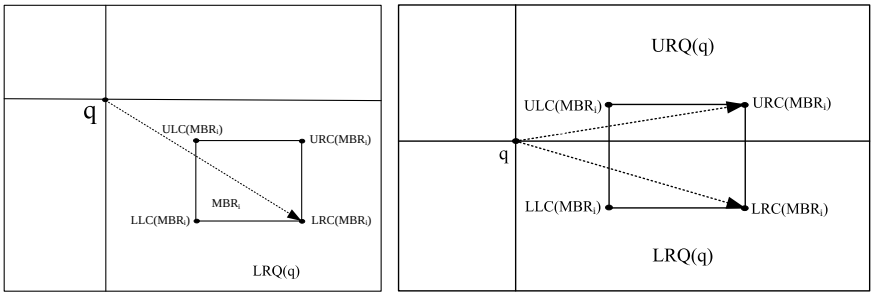
4.1.1 Obtaining the QCERs

Algorithm 1 First step of $q-$MER.

```
1: step1(q, R-tree T)
2: INPUT: q {the query point and the R − tree}
3: OUTPUT: L_QCER {a set of QCERs}
4: Let C = ∅ {a set of points}
5: for each node n parent of the leaves of T do
6:    for each MBR MBR_i in n do
7:       if q is not inside MBR_i then
8:          if MBR_i intersects with only one of the quadrants ULQ(q), LLQ(q), URQ(q) or
            LRQ(q)) then
9:             add FARTHESTC(q, MBR_i) to C
10:         else
11:            if MBR_i intersects with ULQ(q) and LLQ(q) then
12:               add ULC(MBR_i) and LLC(MBR_i) to C
13:            else if MBR_i intersects with LLQ(q) and LRQ(q) then
14:               add LLC(MBR_i) and LRC(MBR_i) to C
15:            else if MBR_i intersects with URQ(q) and ULQ(q) then
16:               add URC(MBR_i) and ULC(MBR_i) to C
17:            else if MBR_i intersects with LRQ(q) and URQ(q) then
18:               add LRC(MBR_i) and URC(MBR_i) to C
19:            end if
20:         end if
21:      end if
22:   end for
23: end for
24: return L_QCER = ComputeQCER(C, q)
```

As it can be seen in Algorithm 1, the first step of $q-$MER obtains zero, one, or two points from each processed MBR, depending on three cases.

1. The first case is when the considered MBR_i is completely inside one of the quadrants defined by the query point (see Figure 8(a)). In this case, the algorithm produces the point of the farthest corner of MBR_i with respect to the query point, i.e. $FARTHESTC(q, MBR_i)$. In Figure 8(a), it is supposed that MBR_i is in $LRQ(q)$, and therefore the point $LRC(MBR_i)$ is added to the set of points C.
2. Another treated case is when the MBR_i intersects with two of the quadrants defined by the query point (see Figure 8(b)). In this case, two points are added to the set C, those in the farthest corners of MBR_i with respect to the query point. In Figure 8(b), MBR_i intersects with quadrants $URQ(q)$ and $LRQ(q)$, and therefore the algorithm adds $URC(MBR_i)$ and $LRC(MBR_i)$ to C.



(a) Example of the first case. (b) Example of the second case.

Fig. 8 The two cases tackled by the first step of $q-$MER.

3. The last case appears when the query point is inside the considered MBR_i. For this situation, we had three options:
   (a) The first option is to split C in two sets of points $C_1 = C \cup \{URC(MBR_i), LLC(MBR_i)\}$ and $C_2 = C \cup \{ULC(MBR_i), LRC(MBR_i)\}$. Now each set should continue the whole process independently. This apparently does not represent a big issue. The problem arises when the query point is in more than one MBR. In this case the number of set of points increases rapidly, since for C_1 two new sets should be created (C_{11} and C_{12}), and the same for C_2. Furthermore, since for each set of points C_i, several QCERs could be created, if the number of sets of points grows fast, the same will happen with the number of QCERs.
   (b) Another option is to access the leaf node corresponding to MBR_i and add to C all real points it contains. This significantly increases the number of points in C and thus, the number of QCERs to be processed. Observe again, that the query point might be inside several MBRs, and then all the points in the leaves corresponding to the entries of those MBRs should be added to C.
   (c) No point is added to C.

   We chose the third option, since we experimentally found that other two options increase the computation time, whereas the benefits in the filtering capability were not significant.

Once we have the set of points C, the algorithm runs the *ComputeQCER* to obtain the QCERs. Next we prove that $LQMER$ can not be larger than, at least, one of these QCERs.

Lemma 1 *Let S be a set of points in a fixed axis-parallel rectangle $R \subseteq \Re^2$, which is stored in an R-tree, and a query point $q \notin S$ such that $q \cap R \neq \emptyset$. Let L_{QCER} the list of QCERs obtained by the first step of q-MER. $LQMER$ can not be larger than one of the QCERs in L_{QCER}.*

Proof: It is clear that $LQMER$ should be one of the QMERs computed from the points in S.

We are going to show that each QCER $QCER_i$ is supported by points of C that ensure the existence of points in the real set of points (S) within the limits of $QCER_i$ and, at least one of them is inside each of the quadrants defined by q. Therefore those points allow the creation of a QMER within $QCER_i$.

Let MBR_p be an MBR in an entry of a node of the R-tree, which is parent of leaves. Let p_{ru} be a point extracted from MBR_p by the *step1* of $q-$MER, that is $p_{ru} \in C$, and let us suppose without loss of generality that p_{ru} is in $URQ(q)$.

Let us consider that $QCER_i$ has an edge supported by p_{ru}. Assume that there is a QMER $QMER_j$ with three edges supported by *real* points inside or contained by the edges of $QCER_i$. We are going to prove that there is a real point p_u within the limits of $QCER_i$ that allows $QMER_j$ keep its area inside $QCER_i$. We have two cases:

1. *$QCER_i$ has its right side supported by p_{ru}.*
   By Property 2 (MBR face property), the existence of MBR_p requires the presence of at least one *real* point contained by $U(MBR_p)$ ($p_u \in S$). Since p_{ru} is the rightmost point of $U(MBR_p)$, $p_u.x \leq p_{ru}.x$, then $QMER_j$ can have an edge supported by p_u, and that edge is within the limits of $QCER_i$.
2. *$QCER_i$ has its upper side supported by p_{ru}.*
   In this case $p_u.y = p_{ru}.y$, then again, $QMER_j$ can have an edge supported by p_u, and that edge is within the limits of $QCER_i$.

Then, we have shown, that the existence of $QCER_i$ implies the existence of a point (p_u in this case) that allows the creation of a $QMER_j$ within the area covered by $QCER_i$.

Observe that, since we assumed that p_{ru} is in $URQ(q)$, if $QCER_i$ has its lower or left edge supported by p_{ru}, $QCER_i$ will not contain q and thus it would not be a QCER.

The only exception to this proof is when the *step1* does not produce points in a quadrant, in such a case we assume that the algorithm includes in C the farthest corner of such quadrant with respect to the query point. Thus $QCER_i$ can not be shortened by a point of C in $URQ(q)$, and hence a QMER can not expand beyond the area of $QCER_i$ in $URQ(q)$.

Finally, the proof can be extended to points in the rest of quadrants with similar reasonings. □

In Figure 7(a), by Property 2, $U(R_8)$ must contain a point of S, and in our example that point is p_9 (see Figure 7(b)), yet could be the case that $c_3 \in S$. p_9 and any point in $U(R_8)$, excepting c_3, would shorten the candidate solution with respect to the QCER A'. Even if $c_3 \in S$ and is the only point in $U(R_8)$, in our example, the obligatory point in $L(R_8)$ would shorten A with respect to A'.

4.1.2 Computing the rectangle with the largest area containing q

Algorithm 2 shows the second step of $q-$MER, which obtains the largest QMER. The set of QCERs obtained from the first step are stored in a heap (binary max-heap, called H_{QCER}) where the QCER with the largest area is at the top.

The algorithm starts by checking the QCER with the largest area. The function *RemoveMax()* extracts the top of the heap. Now the corresponding candidate solution is computed by accessing the real points stored in the leaves of the R-tree. We run the computational geometry algorithm *computeER*[1] with the real points that intersect the considered QCER. To obtain them, we check all the MBRs of nodes that are parents of leaves and intersect with the current QCER. Moreover, from the points inside those MBRs, we only consider those that actually intersect with the considered QCER.

The result of running *computeER* is stored in a temporary object (TMPQMER). The function *area* computes the area of TMPQMER, and if its area is greater than that of the current largest QMER ($MaxMer$), then TMPQMER becomes $MaxMer$.

The process ends when the heap becomes empty or the area of the QCER at the top of the heap is smaller than that of the current $MaxMer$.

Algorithm 2 Second step of $q-$MER

```
1: Step2(Heap H_QCER, point q, R-tree T)
2: INPUT: {heap with the QCERs, query point, and the R − tree}
3: OUTPUT:MaxMer {the largest rectangle containing only q}
4: Let a = 0 {The area of the candidate solution currently stored at MaxMer}
5: repeat
6:    Let C' = ∅ {A set of points}
7:    Let QCER = H_QCER.RemoveMax() {Extracts the first QCER of the heap H_QCER}
8:    for each entry e in nodes of T parent of leaves whose MBR intersects with QCER do
9:       Obtain the leaf node Node pointed by the entry e
10:      for each point r ∈ Node that intersects with QCER do
11:         Let C' = C' ∪ r
12:      end for
13:   end for
14:   Let TMPQMER = computeER(C', q) {Computes LQMER considering the points in
      C'}
15:   if area(TMPMER) > a then
16:      Let MaxMer = TMPQMER
17:      Let a = area(TMPMER)
18:   end if
19: until (H_QCER.isEmpty()) OR (area(QCER)< a)
20: return MaxMer
```

Theorem 1 *Given a set of points S in a fixed axis-parallel rectangle $R \subseteq \Re^2$, which is stored in an R-tree, and a query point $q \notin S$ and $q \cap R \neq \emptyset$. Let $MaxMer$ the output of algorithm $q-MER$. $MaxMer$ is $LQMER$.*

[1] *computeER* only computes $LQMER$, instead of computing all QMERs.

Proof: *Step2* computes the largest QMER inside each QCER obtained by *Step1*. By Lemma 1, no QMER can be larger than one of the QCERs computed by *Step1*, then the largest QMER computed by *Step2* is $LQMER$. □

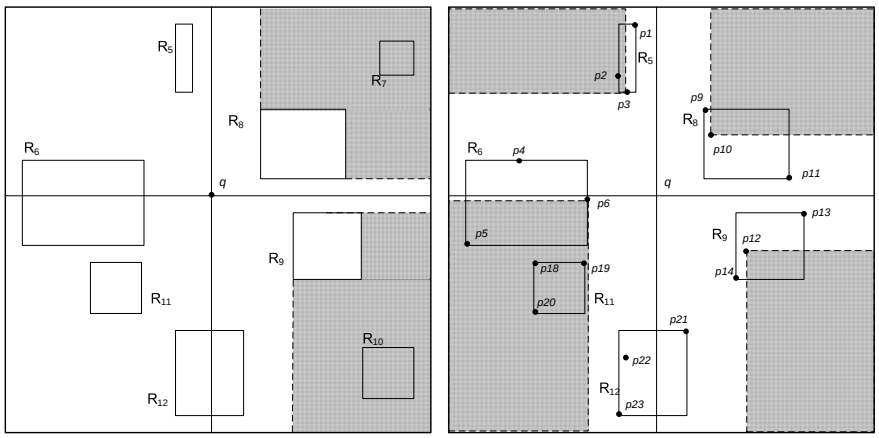
4.2 Dominance-based algorithms

4.2.1 $q-MER_{D1}$ algorithm

This algorithm processes the R-tree level by level from the root to the leaves discarding MBRs in inner levels and points in the leaves. When an MBR is discarded, the algorithm does not take into account its children, which are not further inspected by taking advantage of the Property 1.

The algorithm basically works in inner levels as follows. Considering a level l, for each of the four quadrants defined by q, the algorithm chooses the MBR of l that is completely contained in that quadrant and whose farthest corner with respect to q is nearest to that point. That is, for each quadrant $x \in \{LRQ(q), LLQ(q), ULQ(q), URQ(q)\}$, the algorithm chooses the MBR MBR_x such that:

$$dist(q, FARTHESTC(q, MBR_x)) = min\{dist\ (q, FARTHEST(q, MBR_i))\} \quad \forall MBR_i \text{ completely inside } x \tag{1}$$



(a) Discarding complete MBRs in intermediate levels.

(b) Discarding points in the last level.

Fig. 9 $q-\text{MER}_{D1}$ procedure.

Then for each quadrant x, the algorithm discards all the MBRs of l dominated by MBR_x. In the example of Figure 9(a), from the MBRs fully inside $URQ(q)$, since R_8 dominates R_7, the algorithm discards R_7. In $LRQ(q)$,

R_9 dominates R_{10}, which is discarded. In the other two quadrants, there are not dominance relationships.

After this, the MBRs that are not discarded are replaced by the MBRs in their children, that is, the algorithm continues in the next level of the R-tree. The process described above continues while it does not reach the parent level of the leaves of the R-tree. When this condition becomes true, for each of the MBRs that have not been discarded, the algorithm obtains from the children of their entries, the *real* points they contain. Now the algorithm computes the nearest neighbor nn_x point for each of the quadrants $x \in \{LRQ(q), LLQ(q), ULQ(q), URQ(q)\}$, which is the *real* nearest point to q in the quadrant x considering only the points that were not discarded by the filtering of MBRs in previous steps. Next, from the remaining points, the algorithm discards in each quadrant x, those points dominated by nn_x. Finally, the points that were not discarded are provided as input to *ComputeER*, which obtains the final answer.

In Figure 9(b), in the $ULQ(q)$ quadrant, $nn_{ULQ(q)} = p_3$, then p_2 is discarded since that point is dominated by p_3. However, since p_3 does not dominate p_1 and p_4. Those points, as well as p_3, are provided as input to *ComputeER*. Algorithm 3 presents a detailed description.

Algorithm 3 $q-\text{MER}_{D1}$

1: $q-\text{MER}_{D1}$(point q, R-tree T)
2: INPUT: {the query point and the R-tree}
3: OUTPUT: $MaxMer$ $\{LQMER\}$
4: Let MBR_{root} the MBR that includes all the MBRs at the root of the R-tree T
5: Let E,E_x be sets of MBRs
6: Let C be a set of points
7: Insert in E the element MBR_{root}
8: Let $l = h$ {h is the height of the R-tree T}
9: **while** $l \geq 0$ **do**
10: **if** $l > 0$ **then**
11: **for each** MBR_i in E **do**
12: Substitute MBR_i in E by the MBRs in the child node corresponding to its entry {the node in the next level pointed by the entry containing MBR_i is read}
13: **end for**
14: **for each** $x \in (LLQ(q), LRQ(q), URQ(q), ULQ(q))$ **do**
15: Let E_x the set of MBRs of E that are completely inside the quadrant x
16: Let MBR_x an MBR in E_x such that
$dist(q, FARTHESTC(q, MBR_x)) = min\{dist(q, FARTHEST(q, MBR_i))\}$
$\forall MBR_i$ in E_x
17: Discard from E all the MBRs of E_x that are dominated by MBR_x
18: **end for**
19: **else**
20: Let $C = \emptyset$
21: **for each** MBR_i in E **do**
22: Add to C the points in the children of MBR_i
23: **end for**
24: **for each** $x \in (LLQ(q), LRQ(q), URQ(q), ULQ(q))$ **do**
25: From the points in C, Let nn_x the nearest point to q in quadrant x
26: Remove from C the points dominated by nn_x in the quadrant x
27: **end for**
28: Let $MaxMer$ = ComputeER(C, q)
29: **return** $MaxMer$
30: **end if**
31: Let $l = l - 1$
32: **end while**

Line	content of E	content of C
7	MBR_{root}	$\emptyset$
12	R_1,R_2,R_3, R_4	$\emptyset$
12	R_5,R_6,R_7, R_8,R_9,R_{10}, R_{11},R_{12}	$\emptyset$
17	R_5,R_6,~~R_7~~, R_8,R_9,~~R_{10}~~, R_{11},R_{12}	$\emptyset$
22	R_5,R_6, R_8,R_9, R_{11},R_{12}	p_1, p_2, p_3, p_4, p_5, p_6, p_9, p_{10}, p_{11}, p_{13}, p_{12}, p_{14}, p_{18}, p_{19}, p_{20}, p_{21}, p_{22}, p_{23}
26	R_5,R_6, R_8,R_9, R_{11},R_{12}	p_1, ~~p_2~~, p_3, p_4, ~~p_5~~, p_6, p_9, p_{10}, p_{11}, p_{13}, p_{12}, p_{14}, ~~p_{18}~~, p_{19}, ~~p_{20}~~, p_{21}, p_{22}, p_{23}

Table 3 A trace of $q-\mathrm{MER}_{D1}$.

Using the example of Figure 9, whose R-tree is shown in Figure 2, Table 3 shows a trace of the Algorithm 3. The algorithm starts considering the MBR (MBR_{root}) that encloses the MBRs at the root node, which is immediately substituted by those MBRs (second row of Table 3). The MBRs R_1, R_2, R_3, and R_4 are the rectangles with dotted lines in Figure 2. Since there is no dominance relationships between those MBRs, they are replaced by their children (third row of Table 3). Then the dominance relationships between R_8 and R_7 discards R_7 and that between R_9 and R_{10} discards R_{10} (fourth row). Since we have reached the parent level of leaves, those MBRs are used to build a first version of the set of points C, which contains the points of each MBR in E (fifth row). Next, the dominance relationships between points remove points: p_3 discards p_2, and p_6 discards p_5, p_{18}, and p_{20} (sixth row). Finally, the remaining points in C are provided, along with the query point, to the computational geometry algorithm *ComputeER* to obtain the final result.

Lemma 2 *Let S be a set of points S in a fixed axis-parallel rectangle $R \subseteq \Re^2$, which is stored in an R-tree, and let q be a query point $q \notin S$ such that $q \cap R \neq \emptyset$. Given a point $p_i \in S$, there can not be a QMER $QMER_{p_j}$, which overlaps the dominance area of p_i.*

Proof: Without loss of generality let us suppose that p_i is in $URQ(q)$.

Assume that $QMER_{p_j}$ overlaps the dominance area of p_i. Therefore, it should have an edge supported by a point p_j such that p_i dominates p_j, and thus p_j should be in $URQ(q)$ as well. Without loss of generality suppose that edge is the upper edge.

Observe that when considering the $URQ(q)$ quadrant, if $QMER_{p_j}$ has its lower or left edge supported by p_j, it will not be a QMER, since it would not contain q. Therefore, $QMER_{p_j}$ should have its left edge to the left of q and the lower edge below q.

Since $p_i.y \leq p_j.y$, $p_i.x \leq p_j.x$ and $QMER_{p_j}$ has its left edge to the left of q and the lower edge below q, if $QMER_{p_j}$ overlaps the dominance area of p_i, then $QMER_{p_j}$ will contain p_i, and thus it would not be a QMER. We reach a contradiction because we supposed that $QMER_{p_j}$ overlaps the dominance area of p_i. □

Corollary 1 *Let S be a set of points in a fixed axis-parallel rectangle $R \subseteq \Re^2$, which is stored in an R-tree. Given a query point $q \notin S$ and two MBRs R_i*

and R_j of the R-tree, where $i \neq j$, all contained in R, if R_i dominates R_j in a quadrant defined by q and a QMER $QMER_a$ has an edge e supported by a point p_w, $p_w \cap R_i \neq \emptyset$, there can not be another QMER $QMER_b$ with an edge supported by a point p_k, $p_k \cap R_j \neq \emptyset$.

Proof: Without loss of generality let us suppose that R_i and R_j are in $URQ(q)$. By Property 2, there should be a point p_d in $D(R_i)$ and another one p_l in $L(R_i)$. By the definition of dominance between MBRs, every point in R_j is dominated by p_d and/or by p_l, therefore by Lemma 2, $QMER_b$ can not exist. □

Theorem 2 *Given a set of points S in a fixed axis-parallel rectangle $R \subseteq \Re^2$, which is stored in an R-tree, and a query point $q \notin S$ and $q \cap R \neq \emptyset$. Algorithm $q-MER_{D1}$ obtains LQMER.*

Proof: When the algorithm removes an MBR R_j when processing inner levels, that MBR is dominated by another MBR R_i ($i \neq j$) in that quadrant, therefore by Corollary 1, R_j can be removed safely because any QMER will have its edges supported by points of R_i rather than having them supported by points in R_j.

Similarly, when the algorithm removes points in the leaves, any discarded point p_k is dominated by another point p_w in the same quadrant, then by Lemma 2, p_k can be safely discarded because there can not be a QMER with an edge supported by p_k rather than by p_w, otherwise that QMER will overlap the dominance area of p_w. □

4.2.2 $q-MER_{D2}$ algorithm

This algorithm is a variant of the previous one. $q-\text{MER}_{D2}$ begins computing for each quadrant defined by q, the nearest point with respect to q ($nn_{URQ(q)}$, $nn_{LLQ(q)}$, $nn_{ULQ(q)}$, and $nn_{LRQ(q)}$), by using a combination of *window query* and *nearest neighbor query* in each quadrant. These searches take logarithmic time since we can use the R-tree.

Now the algorithm works similarly to $q-\text{MER}_{D1}$. The R-tree is traversed level by level from the root to the leaves. In non-leaf levels, the algorithm discards the MBRs fully inside one quadrant, which are dominated by the nearest neighbor of that quadrant. In the last level, it discards real points in nodes descendant of the MBRs that were not discarded in previous levels and that are dominated by the nearest neighbor of its quadrant. Finally, again, the remaining points are provided to *computeER* to obtain the final solution.

For example, suppose that $q-\text{MER}_{D2}$ processes the MBRs and points of Figure 10. Focusing on the $LLQ(q)$ quadrant, the algorithm discards the MBR R_{11} given that is fully inside $LLQ(q)$, and it is dominated by $nn_{LLQ(q)} = p_6$. R_6 and R_{12} can not be discarded because they are not fully inside $LLQ(q)$. When the algorithm reaches the leaves, p_5 is discarded since it is dominated by $nn_{LLQ(q)}$, whereas the points that are not dominated by $nn_{LLQ(q)}$ (p_{22} and p_{23}) are kept.

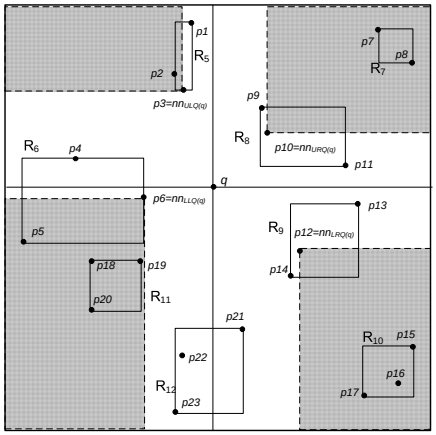


Fig. 10 $q-\text{MER}_{D2}$ procedure.

Line	content of E	content of C
7	MBR_{root}	$\emptyset$
12	R_1,R_2,R_3, R_4	$\emptyset$
12	R_5,R_6,R_7, R_8,R_9,R_{10}, R_{11},R_{12}	$\emptyset$
17	R_5,R_6,~~R_7~~, R_8,R_9,~~R_{10}~~, ~~R_{11}~~,R_{12}	$\emptyset$
22	R_5,R_6, R_8,R_9, R_{12}	p_1, p_2, p_3, p_4, p_5, p_6, p_9, p_{10}, p_{11}, p_{13}, p_{12}, p_{14}, p_{21}, p_{22}, p_{23}
26	R_5,R_6, R_8,R_9, R_{12}	p_1, ~~p_2~~, p_3, p_4, ~~p_5~~, p_6, p_9, p_{10}, p_{11}, p_{13}, p_{12}, p_{14}, p_{21}, p_{22}, p_{23}

Table 4 A trace of $q-\text{MER}_{D2}$.

Algorithm 4 describes the algorithm in detail.

Using the example of Figure 10, Table 4 shows a trace of the Algorithm 4. Again, the algorithm starts considering the MBR at the root node (MBR_{root}). Next, the nearest neighbor to q in each quadrant is computed ($nn_{LRQ(q)} = p_{12}$, $nn_{LLQ(q)} = p_6$, $nn_{ULQ(q)} = p_3$, $nn_{URQ(q)} = p_{10}$). Now, MBR_{root} is substituted by the MBRs it contains (R_1, R_2, R_3, and R_4 of the second row of Table 4), which correspond to the rectangles with dotted lines in Figure 2. Since there are no dominance relationships between the nearest neighbors of q and R_1, R_2, R_3, and R_4, such MBRs are substituted by the MBRs in their children (third row of Table 4). Then the dominance relationships between p_{10} and R_7 discards R_7, that between p_{12} and R_{10} discards R_{10}, and p_6 dominates and discards R_{11} (fourth row). In the level containing parent of leaves, MBRs are used to build a first version of the set of points C, which contains the points of each MBR in E (fifth row). Next, the dominance relationships between points performs the last prune: p_3 discards p_2 and p_6 discards p_5 (sixth row). Finally, the remaining points in C are provided, along with the query point, to the computational geometry algorithm *ComputeER* to obtain the final solution.

Algorithm 4 $q-\text{MER}_{D2}$

```
1: q−MER_D2(point q, R-tree T)
2: INPUT: q and T {the query point and the R-tree}
3: OUTPUT:MaxMer {Largest empty rectangle only containing q}
4: Let MBR_root the MBR that includes all the MBRs at the root of the R-tree T
5: Let E and E_x be sets of MBRs
6: Let C be a set of points
7: Insert in E the element MBR_root
8: for each x ∈ (LLQ(q), LRQ(q), URQ(q), ULQ(q)) do
9:    Let nn_x be the nearest neighbor to q in quadrant x.
10: end for
11: Let l = h − 1 {h is the height of the R-tree}
12: while l ≥ 0 do
13:    if l > 0 then
14:       for each MBR_i in E do
15:          Substitute MBR_i in E by the MBRs in the child node corresponding to its entry
             {the node in the next level pointed by the entry containing MBR_i is read}
16:       end for
17:       for each x ∈ (LLQ(q), LRQ(q), URQ(q), ULQ(q)) do
18:          Let E_x the set of MBRs of E that are completely inside the quadrant x
19:          Discard from E all the MBRs in E_x dominated by nn_x
20:       end for
21:    else
22:       Let C = ∅
23:       for each MBR_i in E do
24:          Add to C the points in the children of MBR_i
25:       end for
26:       for each x ∈ (LLQ(q), LRQ(q), URQ(q), ULQ(q)) do
27:          Remove from C the points dominated by nn_x in the quadrant x
28:       end for
29:       Let MaxMer = ComputeER(C, q)
30:       return MaxMer
31:    end if
32:    Let l = l − 1
33: end while
```

Theorem 3 *Given a set of points S in a fixed axis-parallel rectangle $R \subseteq \Re^2$, which is stored in an R-tree, and a query point $q \notin S$ and $q \cap R \neq \emptyset$. Algorithm $q-MER_{D2}$ computes LQMER.*

Proof: When the algorithm removes an MBR R_j when processing inner levels, that MBR is dominated by the nearest neighbor (nn_x) in that quadrant, therefore since, by construction, nn_x dominates all the points in R_j, by Lemma 2, R_j can be removed safely because any QMER will have its edges supported by nn_x, rather than by any point in R_j.

Similarly, when the algorithm removes points in the leaves level, any discarded point p_k is dominated by nn_x in the same quadrant, then by Lemma 2, p_k can be safely discarded. □

4.3 Combination of the algorithms

As we will see, our experiments show that depending on the data distribution, one of the algorithms performs better than the others. To solve this problem, we propose two combinations of the previous basic algorithms. Each combination chooses one of the basic algorithms depending on a heuristic that tries to anticipate which algorithm will work better with the input data. One

combination includes the $q-$MER and $q-\text{MER}_{D1}$, whereas the other combines $q-$MER and $q-\text{MER}_{D2}$.

In the case of the combination $q-\text{MER}+q-\text{MER}_{D1}$, the heuristic uses the MBRs in the entries of the root node of the R-tree and computes for each quadrant $x \in (LLQ(q), LRQ(q), URQ(q), ULQ(q))$ the MBR MBR_x that satisfies the Formula 1 (see page 19). Then the heuristic computes the dominance area created by each MBR_x in its corresponding quadrant. The percentage of space (R) covered by these dominance areas are stored in a variable called δ_{heur1} $(0 \leq \delta_{heur1} \leq 1)$.

In the case of the combination $q-\text{MER}+q-\text{MER}_{D2}$, the heuristic computes the nearest neighbor to q in each quadrant and computes the dominance areas of those points in each quadrant. Now, the percentage of space (R) covered by these dominance areas are stored in a variable called δ_{heur_2} $(0 \leq \delta_{heur2} \leq 1)$.

In any case, if the percentage of the space of R covered by the dominance areas computed by the heuristics showed above is less than a given threshold, then the $q-$MER algorithm is used, otherwise the dominance-based algorithm is used. When using the combination of $q-\text{MER}+q-\text{MER}_{D1}$, the threshold is 20%, whereas when combining $q-\text{MER}+q-\text{MER}_{D2}$, the threshold is 80%. More formally:

Heuristic 1 *Let S be a set of points in a fixed axis-parallel rectangle $R \subseteq \Re^2$ stored in an R-tree, a query point $q \notin S$ and $q \subseteq R$, and a variable $(0 \leq \delta_{heur1} \leq 1)$. An efficient algorithm to obtain $LQMER$ can be designed as follows:*

1. *if $\delta_{heur1} < 0.2$ then the algorithm $q-MER$ is chosen.*
2. *if $\delta_{heur1} \geq 0.2$ then the algorithm $q-MER_{D1}$ is chosen.*

Heuristic 2 *Let S be a set of points in a fixed axis-parallel rectangle $R \subseteq \Re^2$ stored in an R-tree, a query point $q \notin S$ and $q \subseteq R$, and a variable $(0 \leq \delta_{heur2} \leq 1)$. An efficient algorithm to obtain $LQMER$ can be designed as follows:*

1. *if $\delta_{heur2} < 0.8$ then the algorithm $q-MER$ is chosen.*
2. *if $\delta_{heur2} \geq 0.8$ then the algorithm $q-MER_{D2}$ is chosen.*

The values of the threshold are quite different. The reason is the mechanism used by the two combinations to compute the dominance areas. $q-\text{MER}+q-\text{MER}_{D1}$ uses the MBRs at the root of the R-tree, those MBRs are usually large. This means that it is difficult to find an MBR completely inside one of four quadrants defined by q. Therefore the algorithm that computes δ_{heur1} has problems to create dominance areas. However, the combination $q-\text{MER}+q-\text{MER}_{D2}$ easily obtains the largest possible dominance areas, and thus, the threshold must be higher to balance this ability.

These thresholds were chosen from our experience with different data distributions. Their values depend on where q is located and the distribution of the points. As the points approach to a uniform distribution, the dominance-based algorithm performs better, whereas when the distribution is far from

uniform, the $q-$MER algorithm usually performs better. The basic idea is that $q-$MER works better when there are large empty spaces, this happens mostly in real data distributions. The reason is that in empty spaces there are no points to create dominance relationships, and therefore discarding is low.

The query point also plays an important role. When there are large empty spaces and the query point is in a empty space, $q-$MER quickly obtains a good candidate solution since, the QCERs are processed from largest to smallest. This allows the algorithm to discard many other QCERs that are already smaller than that candidate solution. On the contrary, when the query point is in a zone densely populated, the QCERs are small and therefore they are not capable of discarding most of the QCERs computed by the first step of $q-$MER, and then they should be checked.

However, algorithms $q-\text{MER}_{D1}$ and $q-\text{MER}_{D2}$ lose effectiveness when the query point is in a empty space, as they have less chances to find points to produce dominance relationships, and then the percentage of points that are dominated is smaller, and hence the discarding is lower. On the contrary, if the query point is in a dense populated area, this favors the dominance-based algorithms, as they have good chances to find nearby points that dominate big amounts of points.

The combinations of algorithms take advantage of this behavior, obtaining $q-\text{MER}$ + $q-\text{MER}_{D2}$ the best performance in most cases.

Now the question that might arise is, what happens if the heuristic chooses the wrong algorithm. In Section 5, we will show that in any case, $q-\text{MER}$, $q-\text{MER}_{D1}$, and $q-\text{MER}_{D2}$ perform better than computing the solution with a computational geometry algorithm. Therefore in case of a wrong guess, the chosen algorithm will still get significant improvements.

5 Experimental results

We compared our algorithms against a naive algorithm that retrieves all the points stored in the R-tree by reading all the disk blocks (nodes) and then solving the problem in main memory with the computational geometry algorithm. In any case (naive approach, *ComputeQCER*, and *ComputeER*), we used Orlowski's algorithm [19]. The restriction of the query point allows some improvements in the algorithm that speed up the execution times. Orlowski's algorithm computes several types of MERs. For example, the MERs of Type *bt* are obtained by drawing a vertical line from the top to the bottom of the space passing through each point (see Figure 11). In our case, we only have to compute the MER delimited by the lines that intersect with points W and Z, since that MER is the only one (of this type) that contains q (that is, it is a QMER). To compute this type of MERs, Orlowki's algorithm sorts the points by the X coordinate; we take advantage of this ordering by breaking the process when the QMER is found. Similar improvements were applied to the rest of types of MERs.

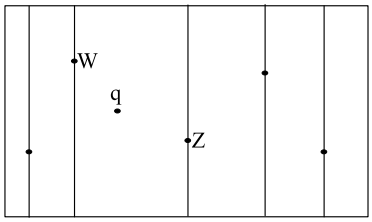


Fig. 11 Orlowki's *bt* MERs.

We suppose that it is possible to store all the points in main memory. This eliminates the effect of the memory over our experiments, since the computational geometry algorithm has all the memory it needs that, as we will see, it is much more than our algorithms.

The algorithms were implemented in Java and the programs were run on an isolated Intel® Xeon® -E5520@2.26GHz with 72 GB DDR3@800MHz RAM with a SATA hard disk model Seagate® ST2000DL003-9VT166. It ran Ubuntu 9.10 (kernel 2.6.31-19-server).

We used Marios Hadjieleftheriou's Java Implementation of an R*-tree.[2]

5.1 Experimental setup

We considered the following sets of points in a two-dimensional space $[0, 1] \times [0, 1]$.

1. Sets of 200K[3], 500K, 1,000K, 2,000K, and 5,000K points with uniform distribution (see Figure 12(a)).
2. Sets of 200K, 500K, 1,000K, 2,000K, and 5,000K points with Gaussian distribution (see Figure 12(b)).
3. Four real datasets denoted by RD1 (see Figure 12(c)), RD2 (see Figure 12(d)), RD3 (see Figure 12(e)), and RD4 (see Figure 12(f)), with 2,249,727, 556,696, 194,971, and 699,900 points, respectively. The real datasets are the California Roads (RD1), Tiger Census Blocks (RD2), and Tiger Streams (RD3) datasets from the web site *rtreeportal*[4] and a dataset (RD4) that was given by a Chilean company provided that the source were not published.

In our experiments, we considered two different disk block sizes, namely 1KB and 4KB. With these disk block sizes (nodes), the maximum capacity of leaf(internal) nodes of the R*-tree were 51(28) and 204(112) points(entries), respectively.

The performance of all algorithms was measured comparing the number of accessed blocks and the response time (it represents the overall execution

[2] `http://libspatialindex.github.com/`

[3] 1K = 1,000 points

[4] `http://rtreeportal.org`

time –elapsed time or wall-clock time– of the algorithms, which is measured in seconds) that each algorithm required to find the solution. The response time includes the time required to read the points from disk. We assume that the R*-tree is already built (that is, the time required to build it is not included in our times) since it is the structure that stores the points. The algorithms use the appropriate structure (a stack for depth-first search or a FIFO queue for breadth-first search) to traverse the R*-tree. The reads of nodes in the stack or the queue are not counted. We do not consider any other read buffer, therefore whenever the algorithms read a node (disk block) that is not in the stack or the queue, that read is counted regardless of whether the node comes from disk or from the operating system buffer cache. Therefore, the measure of accessed blocks ignores the effect of any type of buffer cache (excepting the simple structures commented above). For all measures, we computed the average of 100 random queries.

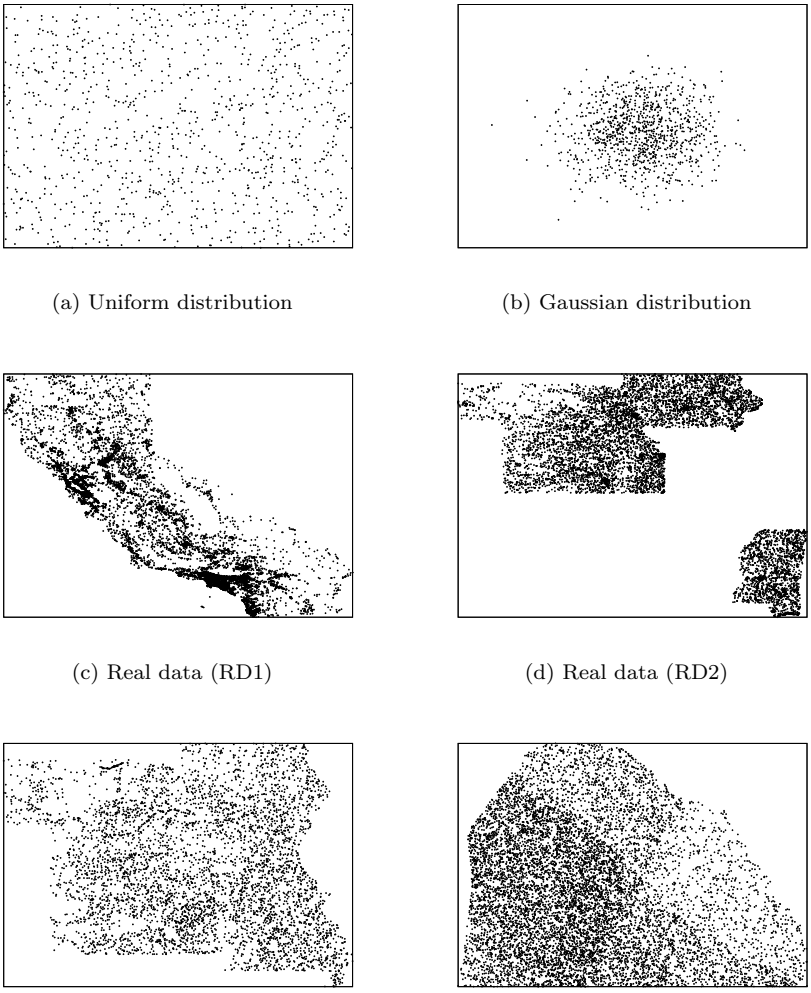
The effect of any read buffer, for example the operating system buffer cache, would benefit only the $q-$MER algorithm, since it might access several times the same R*-tree leaf node when processing different QCERs. However, the naive approach would not improve its performance as it reads from disk each leaf node once, since the repetitive reads of intermediate nodes are solved by the stack or queue. To study this effect, we show the response time required by all algorithms. As we will see, the presence of different buffers (OS buffer or different hardware caches) does not change the values of disk accesses.

5.2 Evaluation

A first test compares all our algorithms against the naive approach using only the synthetic data (uniform and Gaussian distributions).

Figures 13, 14, and 15 show the performance of the algorithms when applied over the datasets with uniform distribution. Figure 13 shows the percentage of disk blocks with respect to the total amount of blocks of the R*-tree that each algorithm needs to access in order to solve the queries. Observe that the naive approach has to access all of them, hence it is not shown in the charts. As it can be seen, $q-\text{MER}_{D1}$ and $q-\text{MER}_{D2}$ present a better behavior than $q-$MER in this experiment. The dominance-based algorithms need only around of the 25% of the disk block accesses required by $q-$MER.

The same magnitude of differences can be observed in Figures 14 and 15, which consider the total amount of accessed blocks and the response time required to solve the queries, respectively. Figures 14 and 15 make clear the difference between the naive approach and our algorithms, observe that those figures use a logarithmic scale in the Y axis. For example, when using a block size of 1 KB, $q-$MER, $q-\text{MER}_{D1}$, and $q-\text{MER}_{D2}$ require only 8.08%, 4.8%, and 4.6%, respectively, of the time required by the naive approach (Figure 15(a)), and the dominance based algorithms around a 40% of that required by $q-$MER. We have not presented the results of the combined algorithms since with these datasets, they achieve marginal improvements.



(a) Uniform distribution

(b) Gaussian distribution

(c) Real data (RD1)

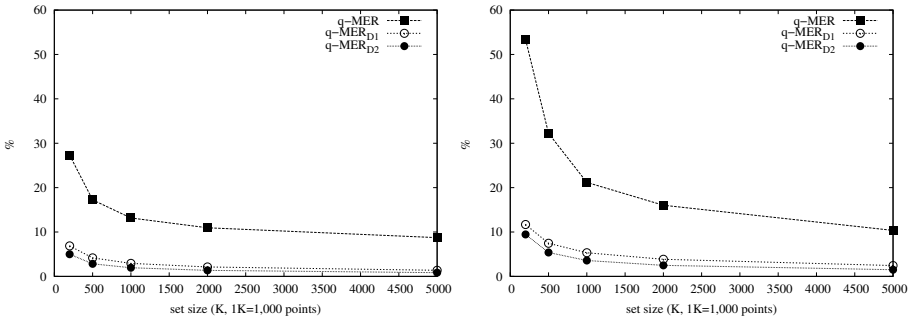
(d) Real data (RD2)

(e) Real data (RD3)

(f) Real data (RD4)

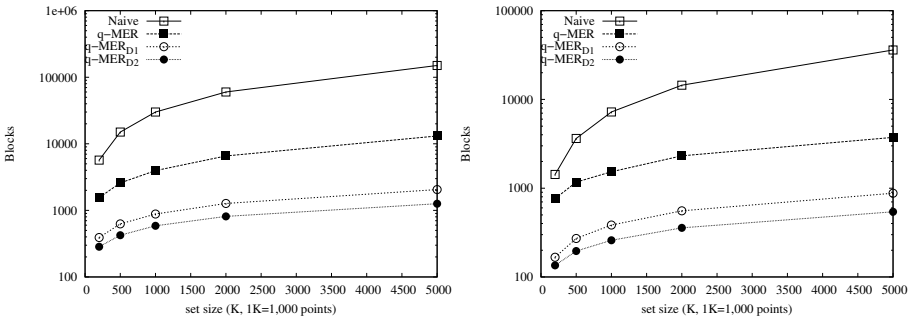
Fig. 12 The datasets used in the experiments (to avoid cluttering the graphs, only some of the points were drawn).

Figures 16, 17, and 18 show the results obtained when using the Gaussian data distribution as input. When the values of the combination of algorithm are not shown in Figures 16 and 17, this means that those values are practically the same as those obtained by the basic algorithms. As in the case of the uniform distribution, all our algorithms significantly overcome the naive approach. For example, observe that in Figure 18(a), $q-$MER, $q-\text{MER}_{D1}$, $q-\text{MER}_{D2}$,



(a) size of block 1 KB (b) size of block 4 KB

Fig. 13 Percentage accessed blocks (data with a uniform distribution).



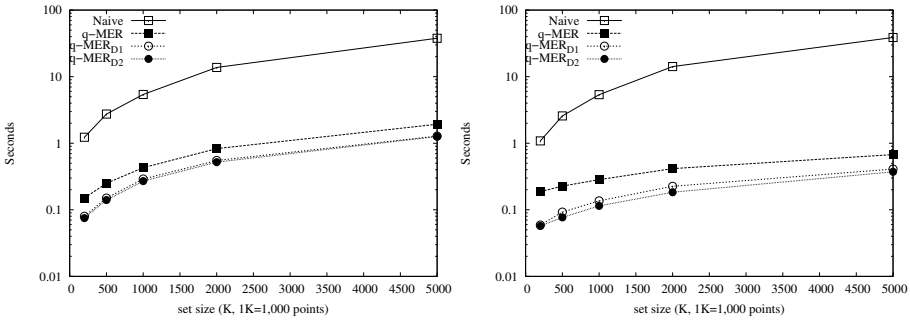
(a) size of block 1 KB (b) size of block 4 KB

Fig. 14 Amount of accessed disk blocks (data with uniform distribution). Observe that the Y axis uses a logarithmic scale.

$q-$MER + $q-$MER$_{D1}$, and $q-$MER + $q-$MER$_{D2}$ require only 7.1%, 5.7%, 6.2%, 5.5%, and 5.5%, respectively, of the time required by the naive approach.

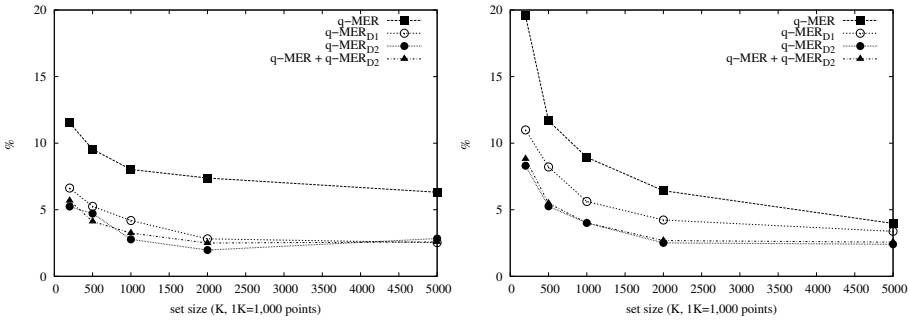
In order to avoid any distortion due to the arrangement of the data in an R*-tree that might increase the disk seek times, we stored all the points in a sequential file, in such a way that the naive approach can read the points sequentially. The results of this experiment can be seen in Figure 19. Observe that, our algorithms also overcome the naive approach in this scenario. The disk blocks were of 1 KB, but the results with 4KB were similar.

Tables 5 and 6 show the performance of our algorithms and the naive approach over the real datasets. In this experiment, we used only one size of disk block of 1KB, and the results for 4KB followed the same trend. Table 5 summarizes the disk block accesses, whereas Table 6 shows the response time consumed to solve the queries. In Table 5, the five rows under the title *#Accessed blocks* show the total amount of blocks accessed, while the last five



(a) size of block 1 KB (b) size of block 4 KB

Fig. 15 Response time required by the algorithms (data with uniform distribution). Observe that the Y axis uses a logarithmic scale.

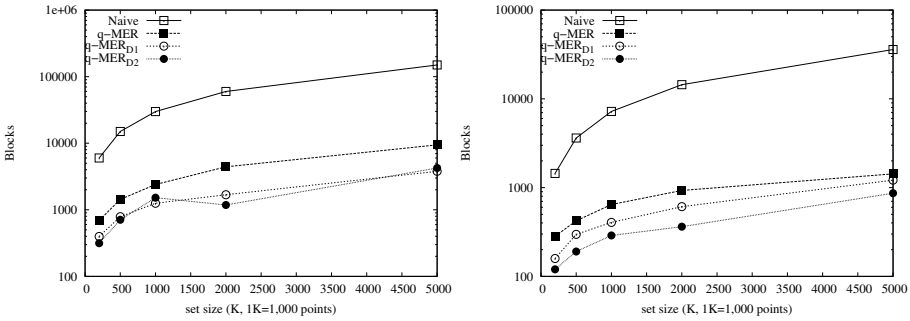


(a) size of block 1 KB (b) size of block 4 KB

Fig. 16 Percentage accessed blocks (data with a Gaussian distribution).

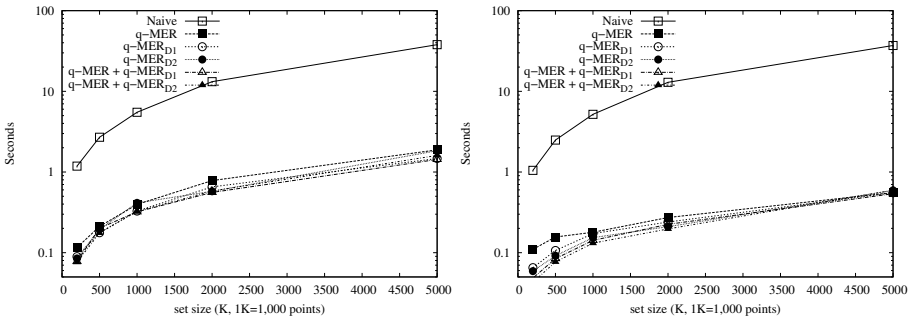
rows present the percentage of blocks accessed by each algorithm with respect to the total amount of blocks. In the case of Table 6, the upper data rows show the response time required by each algorithm, whereas the last five data rows show the percentage of time required by our algorithms with respect to the naive approach.

As in previous data distributions, our algorithms overcome the naive approach. For example, in the case of the RD1 dataset, $q-$MER needs only 6.48% of the time required by the naive approach. Table 5 shows an excellent performance of $q-$MER in the RD1 and RD2 datasets, where it outperforms $q-\text{MER}_{D1}$ and $q-\text{MER}_{D2}$. Yet, datasets RD3 and RD4 show the opposite behavior. If we analyze the distribution of the four real datasets, as we can see in Figure 12, RD1 and RD2 have cluttered zones, whereas others are completely empty. Instead, in RD3 and RD4, the objects occupy almost all the space, having a distribution close to uniform. Therefore, as explained



(a) size of block 1 KB (b) size of block 4 KB

Fig. 17 Accessed disk blocks with Gaussian data distribution. The Y axis is a logarithmic scale.

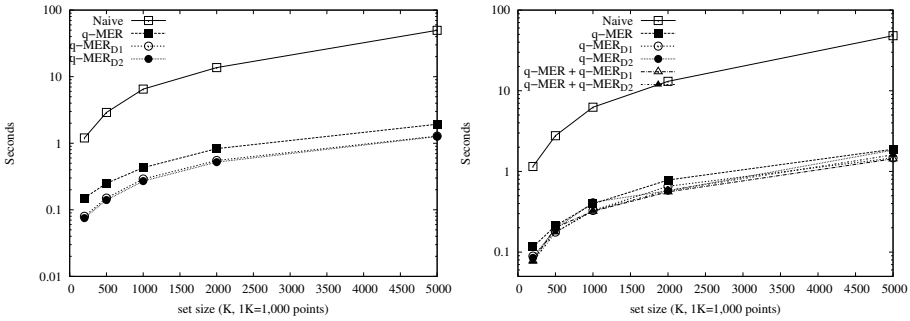


(a) size of block 1 KB (b) size of block 4 KB

Fig. 18 Response time required by the algorithms (Gaussian distribution). The Y axis uses a logarithmic scale.

above, $q-$MER has a better behavior with datasets with large empty spaces, and worse when the distribution gets closer to uniform. On the contrary, the dominance-based algorithms show a better performance with uniform distributions.

Next we focus our study in the storage required for each algorithm. The second step of $q-$MER obtains several sets of points C', one for each QCER computed by the first step. Each set is provided as input to the algorithm *ComputeER* (line 14 of Algorithm 2), which obtains the $LQMER$. We denote each run of *ComputeER* with a set of points as a *case*. The dominance-based algorithms only run *ComputeER* once, and therefore they always tackle just one case. In this experiment, we are not interested in the number of cases, as this parameter is captured by the response time needed to solve the queries. We are interested in the *size* of the cases, which measure the memory needed



(a) Uniform distribution.

(b) Gaussian distribution (the Y axis is in logarithmic scale).

Fig. 19 Experiment where the naive approach reads the points from a sequential file.

	RD1	RD2	RD3	RD4
Size of set(points)	2,249,727	556,696	194,971	699,900
Size of R*-tree(# blocks)	71,720	18,509	6,245	21,066
Algorithm	#Accessed blocks			
$q-$MER	4,799	1,614	1,199	2,416
$q-$MER$_{D1}$	9,478	2,414	448	971
$q-$MER$_{D2}$	11,627	2,354	350	825
$q-$MER + $q-$MER$_{D1}$	8,866	1,611	448	2,346
$q-$MER + $q-$MER$_{D2}$	3,093	938	340	659
Algorithm	Percentage accessed blocks with respect the total			
$q-$MER	6.7	8.7	19.2	11.5
$q-$MER$_{D1}$	13.2	13.0	7.2	4.6
$q-$MER$_{D2}$	16.2	12.7	5.6	3.9
$q-$MER + $q-$MER$_{D1}$	12.4	8.7	7.2	11.1
$q-$MER + $q-$MER$_{D2}$	4.3	5.1	5.4	3.1

Table 5 Blocks accessed with real data, using a disk block size of 1 KB.

by each algorithm. Table 7 describes each of those cases considering several datasets and using a size of disk block of 1KB, and the results for 4KB followed the same trend.

Observe that $q-$MER needs to process many cases to solve a query. However, the average size of those cases is small compared to the total number of points. Moreover, the maximum size of a case is still very small; 3% approximately of the total amount of points. These values remain invariant regardless of the distribution of the considered dataset. The maximum size of a case indicates the amount of space needed to process a query. In our experiments, assuming that each point occupies 20 bytes, 4 bytes for an integer that stores an identifier, and 16 bytes to store two coordinates (two double precision real numbers), $q-$MER needs around 1.1 MB with a dataset of size 2000K points (around 38.1 MB) and uniform or Gaussian distribution.

	RD1	RD2	RD3	RD4
Algorithm	time (seconds)			
Naive	15.60	3.22	1.27	5.10
$q-$MER	1.01	0.29	0.15	0.37
$q-\text{MER}_{D1}$	1.88	0.44	0.09	0.28
$q-\text{MER}_{D2}$	2.33	0.44	0.09	0.27
$q-$MER + $q-\text{MER}_{D1}$	1.77	0.31	0.11	0.28
$q-$MER + $q-\text{MER}_{D2}$	0.89	0.21	0.08	0.25
Algorithm	Percentage of time required with respect to the naive approach			
$q-$MER	6.48	9.00	11.81	7.25
$q-\text{MER}_{D1}$	12.05	13.66	7.08	5.49
$q-\text{MER}_{D2}$	14.93	13.66	7.08	5.29
$q-$MER + $q-\text{MER}_{D1}$	11.34	9.62	8.66	5.49
$q-$MER + $q-\text{MER}_{D2}$	5.70	6.52	8.12	4.90

Table 6 Performance with real datasets and disk block size 1KB.

$q-\text{MER}_{D1}$ and $q-\text{MER}_{D2}$ only need to solve one case for each query, yet the maximum size of a tackled case can reach around the 70% of the total amount of points, when the distribution is far from uniform. However, the space consumed gets lower as the distribution approaches to the uniform distribution, until they reach a 0.3% when it is completely uniform.

This confirms even more the adequacy of $q-$MER to real data with empty spaces and the dominance-based algorithms to the uniform distributions. Once again, the combination of algorithms ($q-$MER + $q-\text{MER}_{D1}$ and $q-$MER + $q-\text{MER}_{D2}$) take advantage of each approach, that is, they reduce the size of the cases (the main advantage of $q-$MER) and the number of cases (the main advantage of the dominance-based algorithms). In Table 7, it can be seen that the algorithm $q-$MER + $q-\text{MER}_{D2}$ achieves a good balance between the size of the cases and the number of cases; this added to the low number of blocks that this algorithm needs (see Table 5), yielding a good average performance.

6 Conclusions

In this work, we have presented three basic algorithms to solve the problem of finding the largest axis-parallel empty rectangle containing only a query point q in a rectangular space that contains a set of points stored in an R*-tree. We also presented two heuristics that choose the basic algorithm that better adapts to the input data distribution in order to obtain the best performance.

We performed a set of experiments to measure the performance of our algorithms considering several synthetic and real datasets. The results show that our algorithms only need to access 3%-55% of the nodes of the R*-tree, that is, they are able to discard a big amount of the points stored in the R-tree. Obviously, this implies that our algorithms require much less response time than the naive approach; in the range 0.8%-15.0% of the time required by the naive algorithm. Regarding space, our algorithms require 0.3%-73% of the space required by the naive algorithm. Moreover, the algorithm with best

Algorithm		RD1	RD2	RD3	RD4	Uniform	Gauss
	Size	2,249K	556K	194K	699K	2,000K	2,000K
$q-$MER	#p	29	20	40	47	70	34
	Avg	2,489	1,040	328	617	997	1,904
	Max	68,214	17,805	6,090	20,371	57,555	57,642
	PMax	3.0	3.2	3.1	2.9	2.9	2.9
$q-\text{MER}_{D1}$	#p	1	1	1	1	1	1
	Avg	258,814	55,946	3,177	13,229	3,584	14,464
	Max	1,472,361	266,253	34,140	126,629	6,887	151,746
	PMax	65.4	47.8	17.5	18.1	0.3	7.6
$q-\text{MER}_{D2}$	#p	1	1	1	1	1	1
	Avg	349,679	61,367	3,484	15,402	3,624	21,377
	Max	1,634,021	268,257	37,465	145,641	6,992	393,380
	PMax	72.6	48.2	19.2	20.8	0.3	19.7
$q-$MER	#p	3	4	1	45	3	2
+	Avg	97,908	9,458	3,177	618	2,146	10,212
$q-\text{MER}_{D1}$	Max	1,472,361	148,993	34,140	20,371	57,555	151,746
	PMax	65.4	26.8	17.5	2.9	2.9	7.6
$q-$MER	#p	7	2	2	3	1	3
+	Avg	8,683	9,200	2,960	3,933	3,624	11,212
$q-\text{MER}_{D2}$	Max	417,966	122,151	25,921	80,726	6992	393,380
	PMax	18.6	21.9	13.3	11.5	0.3	19.7

Table 7 Description of the cases solved by each algorithm. $\#p$ indicates the number of cases; *Avg* the average size of the cases; *Max* the maximum size of the tackled cases; and *PMax* the size of the largest case solved with respect to the total amount of points (in percentage).

average behavior requires 0.8%-8% of the time and 0.3%-21.9% of the space required by the naive approach.

To the best of our knowledge, this is the first work that solves this problem considering that the points are stored in a spatial data structure as the R*-tree. As future work, we want to extend our proposal to objects with more dimensions and to rectangles with sides that are not necessarily parallel to the axes of the original space. We plan also to work in developing a cost model to predict the time and space consumed by our approach.

Acknowledgements This work was supported in part by the project MECESUP UBB0704 (Chile) in the context of a postdoctoral stay of the first author at the University of A Coruña (Spain). For the second and third authors by Ministerio de Educación y Ciencia [TIN2009-14560-C03-02] and [TIN2010-21246-C02-01], and Xunta de Galicia [grant 2010/17]. Finally, for the last author, his work has been supported by the Ministerio de Educación y Ciencia [TIN2008-003063], and the Junta de Andalucía research project [TIC-06114].

References

1. G. Gutiérrez, J. Paramá, Finding the largest empty rectangle containing only a query point in large multidimensional databases, in: Proceedings of SSDBM 2012, Springer, 2012.
2. J. Augustine, S. Das, A. Maheshwari, S. C. Nandy, S. Roy, S. Sarvattomananda, Localized geometric query problems, Computational Geometry 46 (3) (2013) 340 – 357.
3. G. T. Toussaint, Computing largest empty circles with location constraints, International Journal of Computer and Information Sciences 12 (5) (1983) 347 – 358.
4. J. Edmonds, J. Gryz, D. Liang, R. J. Miller, Mining for empty spaces in large data sets, Theoretical Computer Science 296 (2003) 435–452.

5. H. Kaplan, S. Mozes, Y. Nussbaum, M. Sharir, Submatrix maximum queries in monge matrices and monge partial matrices, and their applications, in: Proceedings of SODA 2012, SIAM, 2012, pp. 338–355.
6. V. Gaede, O. Günther, Multidimensional access methods, ACM Computing Surveys 30 (2) (1998) 170–231.
7. S. Shekhar, S. Chawla, Spatial databases - a tour, Prentice Hall, 2003.
8. A. Guttman, R-trees: A dynamic index structure for spatial searching, in: Proceedings of SIGMOD '84, ACM, 1984, pp. 47–57.
9. G. R. Hjaltason, H. Samet, Incremental distance join algorithms for spatial databases, in: Proceedings of SIGMOD '98, ACM, 1998, pp. 237–248.
10. A. Corral, Y. Manolopoulos, Y. Theodoridis, M. Vassilakopoulos, Algorithms for processing k-closest-pair queries in spatial databases, Data & Knowledge Engineering 49 (1) (2004) 67–104.
11. N. Roussopoulos, S. Kelley, F. Vincent, Nearest neighbor queries, SIGMOD Rec. 24 (2) (1995) 71–79.
12. C. Böhm, H.-P. Kriegel, Determining the convex hull in large multidimensional databases, in: Proceedings of DaWaK '01, Springer, 2001, pp. 294–306.
13. Y. Manolopoulos, A. Nanopoulos, A. N. Papadopoulos, Y. Theodoridis, R-Trees: Theory and Applications (Advanced Information and Knowledge Processing), Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.
14. Oracle spatial user's guide and reference (Mar. 2012).
URL `http://docs.oracle.com/html/A88805_01/sdo_intr.htm`
15. Postgis 1.5.3 manual (Mar. 2012).
URL `http://postgis.refractions.net/documentation/manual-1.5/`
16. A. Naamad, D. T. Lee, W.-L. Hsu, On the maximum empty rectangle problem, Discrete Applied Mathematics 8 (1984) 267–277.
17. B. Chazelle, R. L. Drysdalet, D. T. Lee, Computing the largest empty rectangle, SIAM Journal Computing 15 (1986) 300–315.
18. A. Aggarwal, S. Suri, Fast algorithms for computing the largest empty rectangle, in: Proceedings of SCG '87, ACM, 1987, pp. 278–290.
19. M. Orlowski, A new algorithm for the largest empty rectangle problem, Algorithmica 5 (1990) 65–73.
20. M. De , S. C. Nandy, Inplace algorithm for priority search tree and its use in computing largest empty axis-parallel rectangle, CoRR abs/1104.3076.
21. S. Nandy, B. Bhattacharya, Maximal empty cuboids among points and blocks, Computers and Mathematics with Applications 36 (3) (1998) 11 – 20.
22. D. Minati, S. Nandy, Space-efficient algorithms for empty space recognition among a point set in 2d and 3d, in: Proceedings of the 23rd Annual Canadian Conference on Computational Geometry, 2011, pp. 347–353.
23. J. Augustine, S. Das, A. Maheshwari, S. C. Nandy, S. Roy, S. Sarvattomananda, Recognizing the largest empty circle and axis-parallel rectangle in a desired location, CoRR abs/1004.0558.
24. J. Augustine, S. Das, A. Maheshwari, S. C. Nandy, S. Roy, S. Sarvattomananda, Querying for the largest empty geometric object in a desired location, CoRR abs/1004.0558v2.
25. H. Kaplan, M. Sharir, Finding the maximal empty disk containing a query point, in: Proceedings of SCG 2012, SoCG '12, ACM, New York, NY, USA, 2012, pp. 287–292.
26. J. Augustine, B. Putnam, S. Roy, Largest empty circle centered on a query line, Journal of Discrete Algorithms 8 (2) (2010) 143–153.
27. L. P. Chew, R. L. S. Drysdale, Finding largest empty circles with location constraints, Tech. Rep. PCS-TR86-130, Dartmouth College, Computer Science, Hanover, NH (1986).
28. S. Börzsönyi, D. Kossmann, K. Stocker, The skyline operator, in: Proccedings of ICDE '01, 2001, pp. 421–430.
29. D. Papadias, Y. Tao, G. Fu, B. Seeger, Progressive skyline computation in database systems, ACM Transactions on Database Systems 30 (1) (2005) 41–82.
30. E. Dellis, B. Seeger, Efficient computation of reverse skyline queries, in: Proceedings of VLDB '07, ACM, 2007, pp. 291–302.
31. N. Beckmann, H. Kriegel, R. Schneider, B. Seeger, The R*-tree: An efficient and robust access method for points and rectangles, in: H. Garcia-Molina, H. V. Jagadish (Eds.), Proceedings of SIGMOD '09, ACM Press, 1990, pp. 322–331.