Finding the largest empty rectangle containing only a query point in Large Multidimensional Databases*

Gilberto Gutiérrez[1] and José R. Paramá[2]

[1] Universidad del Bío-Bío, Departamento de Ciencias de la Computación y Tecnologías de la Información, Chillán, Chile
ggutierr@ubiobio.cl
[2] University of A Coruña, Department of Computer Science, España
jose.parama@udc.es

Abstract. Given a two-dimensional space, let S be a set of points stored in an R-tree, let R be the minimum rectangle containing the elements of S, and let q be a query point such that $q \notin S$ and $R \cap q \neq \emptyset$. In this paper, we present an algorithm for finding the empty rectangle with the largest area, sides parallel to the axes of the space, and containing only the query point q. The idea behind algorithm is to use the points that define the minimum bounding rectangles (MBRs) of some internal nodes of the R-tree to avoid reading as many nodes of the R-tree as possible, given that a naive algorithm must access all of them. We present several experiments considering synthetic and real data. The results show that our algorithm uses around 0.71–38% of the time and around 3–4% of the main storage needed by previous computational geometry algorithms. Furthermore, to the best of our knowledge, this is the first work that solves this problem considering that the points are stored in an R-tree.

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)). 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.

The search for empty geometric figures with the largest area, or any other metric, has applications in several fields. Among them, we can cite data mining

* 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); and (for second author) by the Spanish Ministerio de Educación y Ciencia [TIN2010-21246-C02-01].

(a) Largest empty rectangle.

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

Fig. 1. The two variants of the problem.

[EGLM03], geographical information systems (GIS), and very-large-scale integration design [ADM+10a].

In the case of data mining, Edmonds, et al. [EGLM03] propose the search for empty spaces as a complementary technique for the characterization of data patterns. More precisely, it is interesting to discover if there are certain ranges of values that never appear together. For example, suppose a database that stores the amounts and dates of bank deposits. Consider a graph where we have the time in the x axis and the amount in the y axis. An empty space indicates that during a given period of time there were not deposits within a certain range of amounts. For example, if we find that during years 2007 and 2008, there were no deposits of more than one million dollars, this could a symptom of a new economic crisis that is arising. In this scenario, the query point could be defined by a time point and a minimum deposit amount. Apart from the discovered knowledge, the empty spaces have value by their own [EGLM03]. Another example could be a database of a Hospital or Social Security system. Considering data about surgery operations, it is possible to discover that there are not face transplants in the database before 2008. This knowledge indicates that such procedure was not possible before that year, and it can be introduced as a integrity restriction of the database, in order to perform semantic query optimization [Kin81].

As an example of GIS application, suppose that we want to build a park in a region, and that we have a database that stores the buildings/houses, electric towers, and other important facilities of that region. The following queries can be interesting: *which is the largest empty rectangular space*? (this gives the space where it is cheaper to place the park) or *which is the largest empty rectangular space around a certain position* q?, if we have a restriction in the position of the park.

The spatial databases (SDB) represent an important aid for GIS to manage large amounts of data. Yet, SBDs require the development of efficient algorithms

and data structures in order to address several query types, among others, the window query, the intersection query, the nearest neighbor, or the pair of nearest neighbors [GG98,SC03]. 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 SBDs. Following this path, several algorithms have been proposed considering that objects are stored in a multidimensional structure, in most cases an R-tree [Gut84], for example; [HS98,Cor02,CMTV04] present several algorithms that solve the k-pairs ($k \geq 1$) of nearest neighbors between two sets, [RKV95] shows an algorithm to find the nearest neighbor to a given point, and more recently, [BK01] presents an algorithm to obtain the convex hull of a set of points stored in an R-tree.

Given a space with a set of points stored in an R-tree, the main contribution of this paper is an algorithm, called $q-$MER, that finds the largest axis-paralell rectangle that only contains a given query point. The algorithm computes a set of candidate maximum empty rectangles (CERs). At least one of these CERs is either the solution or a higher bound of the solution. Instead of inspecting the complete R-tree, we only inspect the blocks/nodes of the R-tree that intersect with those CERs. Our approach takes advantage of the extensive use of the R-tree in real database management systems (Oracle, PostgreSQL, etc.), extending the usefulness of that data structure.

The results show that our algorithm requires between 0.71% and 38% of the running time and between 3% and 4% of the storage required by the naive approach of reading all the points from disk, storing them in main memory, and finally running a computational geometry algorithm with those points. The improvement of our approach comes, in part, by its filtering capability; in our experiments, $q-$MER only needed to access between 10% and 55% of the blocks of R-tree

The outline of the paper is as follows: Section 2 presents some previous related work. Section 3 presents our algorithm. Section 4 shows the results of our experiments. Finally, Section 5 shows our conclusions and directions for future work.

2 Related work

Given a space with a set of n points, 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, and (ii) information about the position is provided, typically by means of a point.

The first variant has been extensively studied. The first work is [NLH84], 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, in [CDL86], it is presented 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 [AS87], this one using $O(n)$ space. In [Orl90], it is shown an algorithm that takes $O(s \log n)$ time, where s is the number rectangles with the largest area. Moreover, that algorithm has an expected time $O(n \log n)$. A more recent approach [DN11] takes $O(n \log^2 n + s)$ time and $O(\log n)$ space by using a priority search tree.

For the second variant, [ADM$^+$10a], [ADM$^+$10b], and [KS11] present algorithms to find the largest empty rectangle that contains a query point. The algorithm presented in [ADM$^+$10a] and [ADM$^+$10b] 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 maximum empty rectangle (MER). These cells are stored in main memory organized into a data structure for objects in two dimensions called range tree. The preprocessing stage takes time and storage $O(n^2 \log n)$ and, to retrieve the MER corresponding to a query point q, an additional $O(\log n)$ time is needed. The algorithm in [KS11] corresponds to a significant improvement in terms of time and space with respect to those in [ADM$^+$10a] and [ADM$^+$10b]. Specifically, this algorithm requires $O(n\alpha(n) \log^3 n)$ storage to maintain the data structure (a segment tree), $O(n\alpha(n) \log^4 n)$ time to build the structure, and $O(\log^4 n)$ time to find the MER that only contains q, where the term $\alpha(n)$ is the slowly increasing inverse Ackermann function.

All the algorithms commented so far assume that the objects can be fit into main memory. Edmonds, et al. [EGLM03] 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 data set. Yet, this work does not consider the case where the objects are stored in a multidimensional structure.

3 Empty rectangle with largest area that contains only a query point

Given a set of points stored in an R-tree, a naive solution could be to read all of them from the R-tree and then find the largest empty rectangle using some of the algorithms described in Section 2. Instead of reading all nodes (disk blocks) of the R-tree, $q-$MER uses the query point q and the properties of the MBRs of the R-tree to avoid inspecting as many blocks as possible. It requires two steps:

1. First, it computes a set of CERs, where at least one of them is either the solution or a higher bound of the solution. In order to obtain those CERs, another two steps are needed:
   (a) A set of points, called C, is generated from the query point q and the MBRs of the R-tree. Specifically, for each MBR in parent nodes of the leaves of the R-tree, the algorithm might add one or two points to C. Those points correspond to the most distant points to the query point q that could be located in the considered MBR. Therefore, it is likely

that most of the points in C do not exist in reality. Figure 2 displays an example. From the MBRs in parent nodes of leaves (the rectangles drawn with solid lines) and the query point q, $q-$MER produces the set of points $C = \{p_1, p_2, p_3, p_4, p_5, p_6\}$. The MBRs in parent nodes of leaves are processed sequentially. The process of the MBR R_1 produces the point p_1, since it is the farthest point with respect to q that could be located in that MBR, but, as explained, p_1 is probably not part of the set of points actually stored in the R-tree. In the same way, the process of R_2 produces the point p_2, R_3 generates p_5 and p_6, and finally R_4 adds p_3 and p_4.

(b) A computational geometry algorithm is run using the set C as input to finally obtain the CERs. A CER is similar to a MER, which is a rectangle that cannot be enlarged if we want to keep only the query point inside it. The difference between a CER and a MER is that while MERs are computed with real points, CERs are computed using C.
Observe that a MER (CER) is not necessarily the largest empty rectangle containing q. Figure 2 displays two CERs, labeled as A and B, which are the rectangles with dotted lines. There are others, but they are not shown to simplify the illustration. For example, observe that the CER B can not be enlarged in any direction. To the south, the space ends; to the east, the CER can not be enlarged, otherwise B would contain the point p_6; to the north, the CER founds a barrel in points p_3 and p_4; and finally to the west, p_5 represents an obstacle to the growth of B. This does not mean that B is the largest empty rectangle containing q, for instance, A is larger.

2. In the second step, the CERs are processed according to their area, from largest to smallest. For each CER, our algorithm accesses the leaves of the R-tree that contain the real points that intersect with such a CER. 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 CERs). This candidate solution is the real MER (since it is computed using real points) that is equal to or contained into the processed CER. As the process of CERs progresses, the candidate solutions may improve previous ones. For example, when the CER B of Figure 2 is processed, it is necessary to access the children of the entries containing the MBRs R_3 and R_4. Those nodes are leaves of the R-tree, and contain the real points that caused the creation of R_3 and R_4. Then $q-$MER inserts in C' the points that intersect with B and processes C' with the computational geometry algorithm. Finally, if the obtained candidate solution is better than the previous ones, then it passes to be considered the best candidate solution so far.

3.1 Basic definitions

First of all, let us present the problem more formally and some definitions that will be used later.

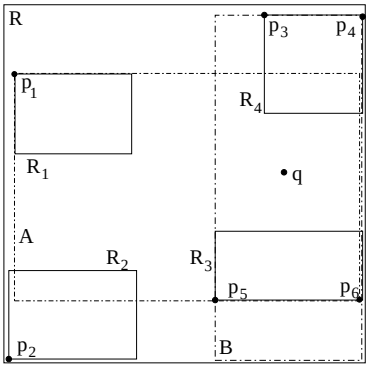


Fig. 2. An example of the elements involved in the first step of $q-$MER.

Given a set of points S in a space $R \subseteq \mathcal{R}^2$, which is stored in an R-tree, and a point $q \notin S$ that is in R, find the rectangle, also in R, with the largest area that only includes q.

Let p be a point, v_p is the vertical line that covers R from north to south and passes through p. h_p is the horizontal line that covers R from west to east and passes through p. These two lines define four regions in R: (i) $NW(p)$ is the northwestern region of p, (ii) $NE(p)$ is the northeast region of p, (iii) $SW(p)$ is the southwestern region of p, and (iv) $SE(p)$ is the southeast region of p. Figure 3(a) displays all these elements.

The four corners of a rectangle r are denoted as: (i) $NW(r)$, the northwestern corner, (ii) $NE(r)$, the northeast corner, (iii) $SW(r)$, the southwest corner, and (iv) $SE(r)$ the southeast corner. In addition, a rectangle r defines four lines: (i) $W(r)$ is the line that connects $NW(r)$ with $SW(r)$, (ii) $E(r)$ is the line that connects $NE(r)$ with $SE(r)$, (iii) $N(r)$ is the line that connects $NW(r)$ with $NE(r)$, and (iv) $S(r)$ is the line that connects $SW(r)$ with $SE(r)$. Figure 3(b) shows these elements.

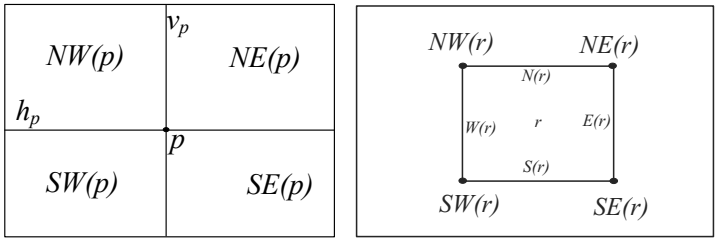
3.2 Obtaining the CERs

As explained, $q-$MER starts by computing a set of CERs. To obtain them, we use a variant of a computational geometry algorithm that obtains the rectangle with the largest empty area. This variant, that we call *ComputeCER*, produces the set of MERs, instead of obtaining only the largest one.

The key idea is to use *ComputeCER* with much fewer points than in the case of using the computational geometry algorithm over the whole set of points.

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.

The first case is when the *if* of line 8 is true. This means that the considered MBR_i is completely inside one of the regions defined by the query point (see Figure 4(a)). In this case, the algorithm produces the point of the farthest corner



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

Fig. 3. Definitions.

of MBR_i with respect to the query point. In Figure 4(a), it is supposed that MBR_i is in $SE(q)$, and therefore the point $SE(MBR_i)$ is added to the set of points C.

Another treated case is when the *if* of line 19 is true. This means that MBR_i intersects with two of the regions defined by the query point (see Figure 4(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 4(b), MBR_i intersects with regions $NE(q)$ and $SE(q)$, and therefore the algorithm adds $NE(MBR_i)$ and $SE(MBR_i)$ to C.

The last case appears when the query point is inside the considered MBR_i. For this situation, we have three options:

1. The first option is to split C in two sets of points $C_1 = C \cup \{NE(MBR_i), SW(MBR_i)\}$ and $C_2 = C \cup \{NW(MBR_i), SE(MBR_i)\}$. 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 CERs should be created, if the number of sets of points grows fast, the same will happen with the number of CERs.
2. To access the leaf node corresponding to MBR_i and add all the real points it contains to C. This increases significatively the number of points in C and thus, the number of CERs 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.
3. No point is added to C.

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

Algorithm 1 Algorithm that processes an R-tree to obtain the CERs.

```
1: step1(q, T)
2: INPUT: q {the query point and T the R − tree}
3: OUTPUT: L_CER {a set of CERs}
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 h_q and v_q do not cross N(MBR_i), S(MBR_i), E(MBR_i), and W(MBR_i) then
9:             if MBR_i is completely inside NW(q) then
10:               add NW(MBR_i) to C
11:            else if MBR_i is completely inside SW(q) then
12:               add SW(MBR_i) to C
13:            else if MBR_i is completely inside NE(q) then
14:               add NE(MBR_i) to C
15:            else if MBR_i is completely inside SE(q) then
16:               add SE(MBR_i) to C
17:            end if
18:         else
19:            if h_q or v_q crosses exactly two of the lines N(MBR_i), S(MBR_i), E(MBR_i), and
               W(MBR_i) then
20:               if MBR_i intersects with NW(q) and SW(q) then
21:                  add NW(MBR_i) and SW(MBR_i) to C
22:               else if MBR_i intersects with SW(q) and SE(q) then
23:                  add SW(MBR_i) and SE(MBR_i) to C
24:               else if MBR_i intersects with NE(q) and NW(q) then
25:                  add NE(MBR_i) and NW(MBR_i) to C
26:               else if MBR_i intersects with SE(q) and NE(q) then
27:                  add SE(MBR_i) and NE(MBR_i) to C
28:               end if
29:            end if
30:         end if
31:      end if
32:   end for
33: end for
34: L_CER = ComputeCER(C, q)
```

As explained, once we have the set of points C, the algorithm runs the *ComputeCER* to obtain the CERs. Next we prove that the solution cannot be larger than, at least, one of these CERs.

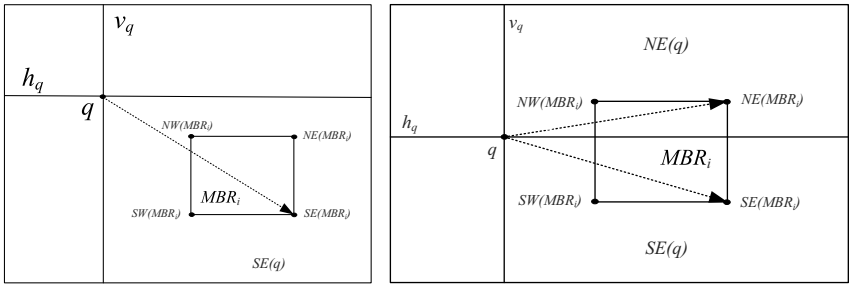
Theorem 1. *Let L_{CER} the list of CERs obtained by the first step of q-MER. The solution or solutions can not be larger than one of the CERs in L_{CER}.*

Proof:

It is clear that the solution should be one (or more) of the real MERs. For each real MER, we are going to prove that the first step of $q-$MER produces at least one CER that is either equal or a higher bound of that MER. We prove this by showing that the points computed by the first step of $q-$MER using one MBR can not shorten a CER with respect to the corresponding real MER. Once we prove this for the points obtained from one MBR, the proof trivially extends for any number of MBRs.

Let MBR_p a MBR in an entry of a node of the R-tree, which is parent of leaves:

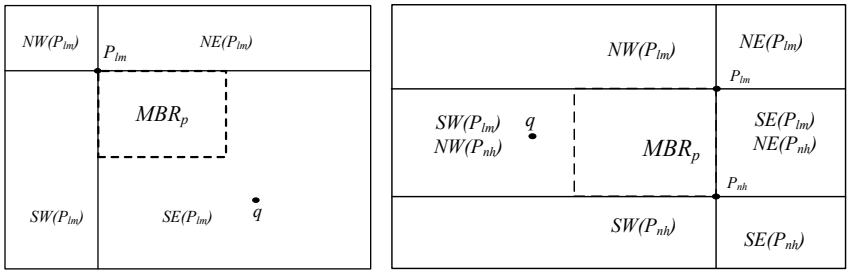
Case 1 Assume that MBR_p is fully inside one of the regions defined by q, then the first step of $q-$MER adds only one point to the set C. Let P_{lm} be that point



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

Fig. 4. The two cases tackled by the first step of $q-$MER.

and let us suppose without loss of generality that P_{lm} is in $NW(q)$. Figure 5(a) shows an example of the four regions defined by P_{lm} and the MBR responsible of its creation.



(a) Example of Case 1. (b) Example of Case 2.

Fig. 5. The two cases of the proof.

Assuming no other MBR, the algorithm *ComputeCER* creates two CERs: (1) the union of $SW(P_{lm})$ and $SE(P_{lm})$, and (2) the union of $NE(P_{lm})$ and $SE(P_{lm})$. The region $NW(P_{lm})$ can not be part of a CER that contains q and does not contain P_{lm}, by construction.

If by chance, there is a point in S (the set of real points) with the same coordinates as P_{lm}, we are going to show that the CERs (1) and (2) are larger than corresponding the real MERs. It is clear that the existence of MBR_p requires the presence of more points than P_{lm}, at least one more. One of these two cases should occur. (i) At least the point $SE(MBR_p)$ exists: this point would not allow the existence of MERs with the same size as the CERs (1) and (2), given that $SE(MBR_p)$ is closer to q than P_{lm}, since it is more to the right and in a lower position with respect to P_{lm}. Thus, it

would represent an obstacle that would produce MERs shorter than (1) and (2). (ii) The existence of, at least, another two points, one that intersects with $N(MBR_p)$ and another one that intersects with $W(MBR_p)$. These points would be placed with respect P_{lm} more to the right (that lying on $N(MBR_p)$) and in a lower position (that lying on $W(MBR_p)$). Therefore, again the real MERs would be shorter than the CERs (1) and (2).
If P_{lm} does not exist in reality, by construction, MBR_p requires the existence of at least two points; one intersecting $N(MBR_p)$ and another one intersecting $W(MBR_p)$. Those points, once again, must be placed more to the right and in a lower position with respect to P_{lm}, respectively. Therefore, the real MERs would be shorter than the created CERs, as well.
Thus, we can conclude that the CERs (1) and (2) are higher bounds of the real MERs.

Case 2 Now we consider that MBR_p is between two regions of those defined by q. Let P_{lm} and P_{nh} be the two points produced by the first step of $q-$MER. Without loss of generality, let us suppose that those points are to the east of q (see an example in Figure 5(b)). As it can be seen in the figure, there are only two CERs that only contain q: (1) The area resulting from $(SW(P_{lm}) \cap NW(P_{nh})) \cup (SE(P_{lm}) \cap NE(P_{nh}))$, and (2) the area resulting from $NW(P_{lm}) \cup SW(P_{lm})$, or which is the same, $NW(P_{nh}) \cup SW(P_{nh})$.
For the CER (1), if the real points in MBR_p are placed only intersecting with $N(MBR_p)$ or $S(MBR_p)$, the obtained CER is equal to the real MER. In any other case, the MER that contains q would be, at least, a rectangle less high, and therefore it would have a shorter area.
In the case of CER (2), the existence of MBR_p requires the existence of, at least, one point lying on $W(MBR_p)$. That point would be more to the west than P_{lm} and P_{nh} and, at the same time, more to the east of q (otherwise MBR_p would include q), therefore that point would shorten the real MER with respect to the CER (2).

Observe in Figure 2 that, for example, any point that do not lie on the east and south lines of the MBR R_3 would shorten the CER A. Indeed, those points should exist, otherwise R_3 would not be created. Therefore A is a higher bound for the real MER.

Figure 6 displays a possible arrangement of the real points. As seen, the real MER A' is shortened with respect to the CER A, due to, among other points, the existence of a point intersecting with $N(R_3)$. The case of the CER B is even worse, as its corresponding MER B' is not a MER anymore, as now B' is included in the MER A'.

3.3 Computing the rectangle with the largest area containing q

Algorithm 2 shows the second step of $q-$MER, which obtains the largest MER containing q. The set of CERs obtained from the first step are stored in a heap where the largest CER is at the top.

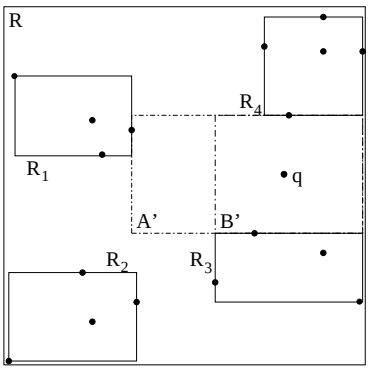


Fig. 6. An example of a real MER corresponding to Figure 2.

The algorithm starts by checking the largest CER. The function *Get* extracts the top of the heap. Now the corresponding real MER is computed by accessing the real points stored in the leaves of the R-tree. We run the computational geometry algorithm *computeER*[3] with the real points that intersect the considered CER. To obtain them, we check all the MBRs of parent nodes of leaves that intersect with the current CER. Moreover, from the points inside those MBRs, we only consider those that actually intersect with the considered CER.

The real MER is stored in a temporary object (TMPMER). The function *area* computes the area of that MER, and if its area is greater than that of the current MER_{max}, then TMPMER becomes MER_{max}. The process ends when the heap becomes empty or the area of the CER at the top of the heap is shorter than that of the current MER_{max}.

4 Experimental results

We compared $q-$MER with a naive algorithm that retrieves all the points stored in the R-tree by reading all the blocks and then solving the problem in main memory with the computational geometry algorithm (*computeER*). We used the algorithm of Naamad, et al. [NLH84] modified to meet our requirements, that is, the computation of the largest rectangle containing only q. *computeCER* is also a modification of the same algorithm, which obtains all the possible MERs containing q.

The restriction of the query point allows some improvements in the algorithm that speed up its execution times. Naamad's algorithm compute three types of MERs (called A, B, and C). The MERs of Type A are obtained by drawing a vertical line from the top to the bottom of the space passing through each point (see Figure 7). In our case, we only have to compute the MER delimited by the

[3] *computeER* is similar to *computeCER*, with the difference that *computeER* only returns the largest MER.

Algorithm 2 Algorithm that computes the rectangle with the largest area containing a query point

```
1: Step2(H_CER, q, T)
2: INPUT: {H_CER a heap with the CERs obtained by ComputeCER, q the query point, and T
   the R − tree}
3: OUTPUT:MER_max {the largest rectangle containing only q}
4: Let a = 0 {The area of the MER currently stored at MER_max}
5: repeat
6:    Let C' = ∅ {A set of points}
7:    Let CER = Get(H_CER) {Extracts the first CER of the heap H_CER}
8:    for each node n parent of leaves with MBRs intersecting with CER do
9:       Let C' = C' ∪ all the points stored in the children of n that intersect with CER
10:   end for
11:   Let TMPMER = computeER(C', q) {The computational geometry algorithm computes the
      largest rectangle only containing q considering the points in C'}
12:   if area(TMPMER) > a then
13:      Let MER_max = TMPMER
14:      Let a=area(TMPMER)
15:   end if
16: until (H_CER is empty) OR (area(CER)< a)
```

lines that intersect with points W and Z, since that MER is the only one (of this type) that contains q. To compute this type of MERs, Naamad's algorithm sorts the points by the x coordinate, we take advantage of this ordering to obtain the target MER by performing a binary search that obtains the nearest points to q, in our example, the points W and Z. This reduces the cost from linear to logarithmic. Similar improvements were applied to MERs of Type B and C.

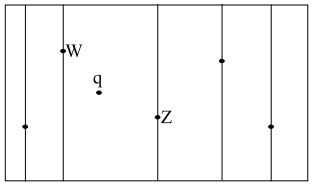


Fig. 7. Naamad's Type A MERs.

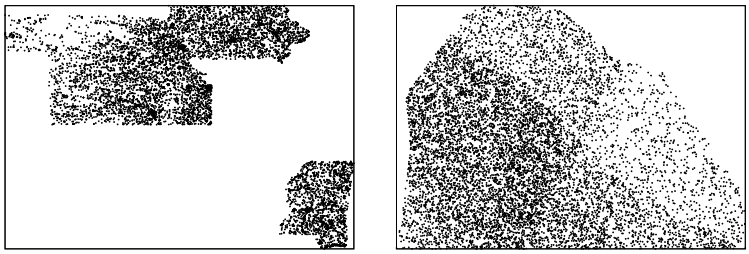
We suppose that it is possible to store all the points in main memory. This eliminates the effect of the memory over our experiments, since we provide the computational geometry with all the memory it needs that, as we will see, it is much more than $q-$MER.

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).

The performance of both algorithms was measured comparing the number of accessed blocks and the real time that each algorithm required to find the solution. The time includes the time required to read the points from disk. We

recall that 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. Both $q-$MER and the naive approach use a stack to traverse the R-tree. This stack is used to avoid the repetitive read of internal nodes of the R-tree from disk. As a search traverses the R-tree downwards, the nodes are stored in the stack, if we need to go back upwards, we already have the parent node at the top of the stack, therefore the maximum size of the stack is the height of the R-tree. For measurement purposes, these reads of nodes in the stack are not counted. We do not use any other read buffer, therefore whenever the algorithms read a node (disk block) that is not in the stack, that read is counted regardless of whether the node comes from disk or from the operating system cache. Therefore, the measure of accessed blocks eliminates the effect of any type of buffer cache (excepting the simple stack).

We considered real and synthetic set of points in a two-dimensional space $[0,1]\times[0,1]$. The synthetic data sets have uniform distribution. The real data sets are the Tiger Census Blocks data set (RD1) from the web site `rtreeportal.org` and a data set (RD2) that was given by a Chilean company provided that the source were not published. Figure 8 shows some of the points of these data sets (we only plot some of them in order to simplify the graphs). The sizes of the sets were as follows: for the synthetic data, we used sets with size 200K[4], 500K, 1,000K, 2,000K, and 5,000K; and the real data sets have size 556K (RD1) and 700K (RD2). In the case of the synthetic data sets, we considered block sizes of 1KB[5] and 4KB. Furthermore, for all measures, we computed the average of 100 queries.



(a) RD1. (b) RD2.

Fig. 8. Real data sets.

The results of the experiments using synthetic data are summarized in Figures 9–11 and Table 1. In Figures 9 and 10, we can observe the filtering capability of the $q-$MER algorithm. For example, $q-$MER only needs to access around be-

[4] 1K = 1,000 points
[5] 1KB = 1,024 Bytes

tween 10% and 55% (Figure 9) of the blocks of the R-tree, whereas the naive algorithm needs to access all of them. Furthermore, we can see that when the size of the set increases, the percentage of the blocks accessed by $q-$MER decreases.

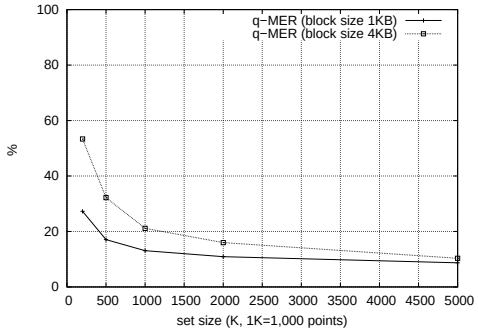
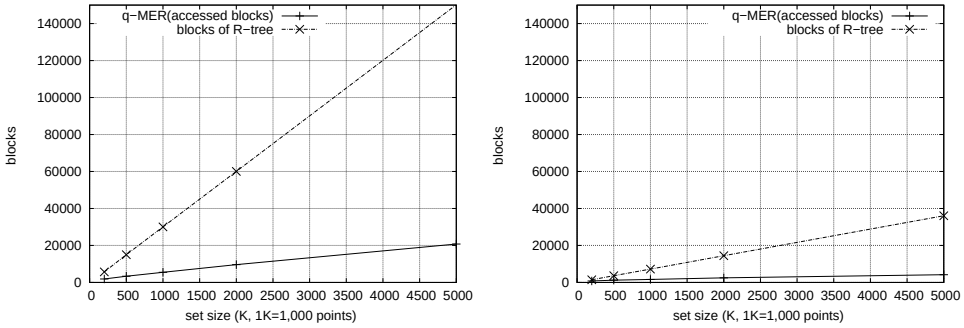


Fig. 9. Percentage accessed blocks.



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

Fig. 10. Performance (accessed blocks) against total blocks of R-tree.

Figure 10 shows the amount of blocks accessed by $q-$MER and the total number of blocks that the R-tree uses to store the points. When the size of the collection of points increases, we can see that the slope of the line that shows the blocks accessed by $q-$MER to solve the problem is less pronounced than that showing the size of the R-tree.

Figure 10 also shows the effects of the size of the blocks over the amount of accessed blocks, which, as expected, decreases inversely proportional to the size of the block.

size of set (thousand)	size of block(KB)	# problems	# points average	# points minimum	# points maximum
200	1	54	928	57	5,893
	4	48	2,206	217	8,464
500	1	60	1,307	90	14,520
	4	54	3,002	306	12,941
1,000	1	63	1,799	81	28,893
	4	58	3,601	416	17,508
2,000	1	69	2,599	89	57,555
	4	66	4,774	344	32,559
5,000	1	77	4,345	82	143,402
	4	74	6,782	365	53,101

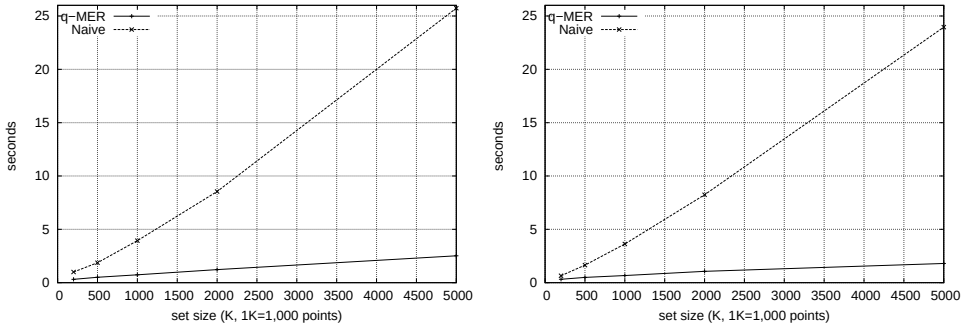
Table 1. Description of the problems.

As explained, we do not use any read buffer (except the stack). 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 CERs. 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.

The second step of $q-$MER obtains several sets of points C', one for each CER computed by the first step. Each set is provided as input to the algorithm *ComputeER*, which solves the problem of finding the largest empty rectangle with those points in main memory. In Table 1, we denote each run of *ComputeER* with a set of points C' (line 11 of Algorithm 2) a *problem*. Table 1 describes each of those problems considering different sizes of original sets of points and sizes of blocks. The third column shows the amount of problems solved by $q-$MER to obtain the final solution. This amount also includes the run of the algorithm *ComputeCER* of the first step (line 33 of Algorithm 1). Note that each CER obtained by the first step of $q-$MER represents a *problem*, that is, the total number of problems is the number of CERs provided by the first step of $q-$MER, plus one. Observe that the naive algorithm only solves one *problem*, but with much more points.

Table 1 shows that the bigger problem treated by *ComputeER* in the $q-$MER algorithm includes a very short percentage of the total number of points, namely around 3%–4%. This means that $q-$MER only needs around the 3%–4% of the main memory needed by the naive algorithm to solve the same problem.

Figure 11 compares the average execution time of a query using $q-$MER and the naive algorithm when the points are retrieved from the R-tree. In all cases, $q-$MER overcomes the naive approach, since $q-$MER uses between 7%-38% of the time required by the naive approach. The differences get bigger as the size of the problem size increases. In order to avoid any distortion due to the arrangement of the data in a 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 12. As it can be seen, $q-$MER also overcomes the naive approach in this



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

Fig. 11. Performance (real time) of algorithm $q-$MER.

scenario. Figure 12 displays the values for disk blocks of 1 KB, since the results for 4KB block size were similar as the time required to read the points is a very small portion of the time required to solve the problem.

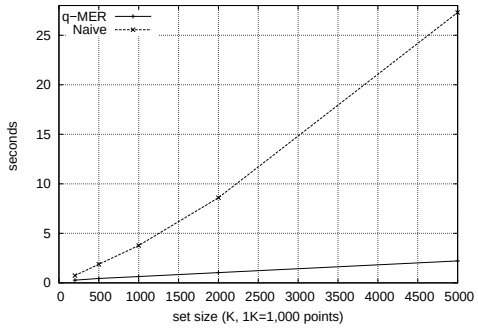


Fig. 12. Experiment where the naive approach reads the points from a sequential file.

With regard to the main memory consumption, each *problem* (as explained each run of *ComputeER* or *ComputeCER*) needs to store in main memory the points involved in that computation. In the worst case, a problem solved by $q-$MER needs only 3%–4% of the points (or main memory), which needs the naive approach. If the available memory is not enough for the naive approach, this implies a important increment of the time required to run the algorithm, since external sorts would be required.

Tables 2 and 3 show the performance of our algorithm and the naive approach over the real datasets. In this experiment, we used only one size of disk block

	RD1	RD2
Size of R-tree(# blocks)	18,509	21,066
#Accessed blocks	1,614	2,416
% accessed blocks with respect the total	8.7	11.5

Table 2. Blocks accessed by $q-$MER with real data.

(1 KB). Table 2 summarizes the disk block accesses, whereas Table 3 shows the time consumed to solve the queries.

As in previous data distributions, our algorithm overcome the naive approach. In this case, differences are even bigger; $q-$MER needs only 0.71% and 1.5% (for the data sets RD1 and RD2, respectively) of the time required by the naive approach (see Table 3).

5 Conclusions

Given a space with a set points stored in spatial index R-tree, we described the algorithm $q-$MER that obtains the largest rectangle containing just a query point q. The results of the experiments with synthetic and real data show that $q-$MER requires between 0.71% and 38% of the running time and between 3% and 4% of the storage required by the naive algorithm. In part, the performance of $q-$MER can be explained by its filtering capability, since it requires to access only around between 10% and 55% of the total blocks of the R-tree.

The experiments also show the scalability of our algorithm, which obtains better improvements as the size of the collection grows. In addition, $q-$MER would benefit from any improvement in the computational geometry algorithm used by *computeCER* and *computeER*.

To the best of our knowledge, this is the first work that solves this problem considering that the points are stored in a multidimensional data structure 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. We would like to thank Juan Ramón López Rodríguez for his comments and suggestions.

	RD1	RD2
Algorithm	time (seconds)	
Naive (read from R-tree)	62.155	32.939
Naive (read from sequential file)	62.085	32.807
$q-$MER	0.441	0.494

Table 3. Performance with real datasets and disk block size 1KB.

References

[ADM+10a] J. Augustine, S. Das, A. Maheshwari, S. C. Nandy, S. Roy, and S. Sarvattomananda. Recognizing the largest empty circle and axis-parallel rectangle in a desired location. *CoRR*, abs/1004.0558, 2010.

[ADM+10b] John Augustine, Sandip Das, Anil Maheshwari, Subhas C. Nandy, Sasanka Roy, and Swami Sarvattomananda. Querying for the largest empty geometric object in a desired location. *CoRR*, abs/1004.0558v2, 2010.

[AS87] A. Aggarwal and S. Suri. Fast algorithms for computing the largest empty rectangle. In *Proceedings of the third annual symposium on Computational geometry*, SCG '87, pages 278–290, New York, NY, USA, 1987. ACM.

[BK01] C. Böhm and H-P. Kriegel. Determining the convex hull in large multidimensional databases. In *Proceedings of the Third International Conference on Data Warehousing and Knowledge Discovery*, DaWaK '01, pages 294–306, London, UK, 2001. Springer-Verlag.

[CDL86] B. Chazelle, R. L. Drysdalet, and D. T. Lee. Computing the largest empty rectangle. *SIAM Journal Computing*, 15:300–315, 1986.

[CMTV04] A. Corral, Y. Manolopoulos, Y. Theodoridis, and M. Vassilakopoulos. Algorithms for processing k-closest-pair queries in spatial databases. *Data & Knowledge Engineering*, 49(1):67–104, 2004.

[Cor02] A. Corral. *Algoritmos para el Procesamiento de Consultas Espaciales utilizando R-trees. La Consulta de los Pares Más Cercanos y su Aplicación en Bases de Datos Espaciales.* PhD thesis, Universidad de Almería, Escuela Politécnica Superior, España, Enero 2002.

[DN11] M. De and S. C. Nandy. Inplace algorithm for priority search tree and its use in computing largest empty axis-parallel rectangle. *CoRR*, abs/1104.3076, 2011.

[EGLM03] J. Edmonds, J. Gryz, D. Liang, and R. J. Miller. Mining for empty spaces in large data sets. *Theoretical Computer Science*, 296:435–452, March 2003.

[GG98] V. Gaede and O. Günther. Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–231, 1998.

[Gut84] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *ACM SIGMOD Conference on Management of Data*, pages 47–57. ACM, 1984.

[HS98] G. R. Hjaltason and H. Samet. Incremental distance join algorithms for spatial databases. In *ACM SIGMOD Conference on Management of Data*, pages 237–248, Seattle, WA, 1998.

[Kin81] J. J. King. *Query optimization by semantic reasoning.* PhD thesis, Stanford University, CA, USA, 1981.

[KS11] H. Kaplan and M. Sharir. Finding the maximal empty rectangle containing a query point. *CoRR*, abs/1106.3628, 2011.

[NLH84] A. Naamad, D. T. Lee, and W.-L. Hsu. On the maximum empty rectangle problem. *Discrete Applied Mathematics*, 8:267–277, 1984.

[Orl90] M. Orlowski. A new algorithm for the largest empty rectangle problem. *Algorithmica*, 5:65–73, 1990.

[RKV95] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *SIGMOD '95: Proceedings of the 1995 ACM SIGMOD international conference on Management of data*, pages 71–79, New York, NY, USA, 1995. ACM Press.

[SC03] S. Shekhar and S. Chawla. *Spatial databases - a tour.* Prentice Hall, 2003.