Efficiently Querying Vector and Raster Data

Nieves R. Brisaboa[1], Guillermo de Bernardo[1], Gilberto Gutiérrez[2], Miguel R. Luaces[1] and José R. Paramá[1]

[1] *Universidade da Coruña, Departamento de Computación, Facultade de Informática, Campus de Elviña, 15071 A Coruña, Spain*
[2] *Universidad del Bío-Bío, Computer Science and Information Technologies Department, Chillán, Chile*
Email: brisaboa@udc.es; gdebernardo@udc.es; ggutierr@ubiobio.cl; luaces@udc.es; parama@udc.es

Even though the field of spatial databases is more than forty years old, most existing logical data models are highly-focused either on spatial objects (vector data models) or spatial fields (raster data models). Furthermore, spatial index structures and query algorithms are still proposed for one of the approaches and little research work has been dedicated to index structures and query algorithms where both types of information are needed. However, due to the current high availability of different types of data, it is much more common nowadays that applications require querying vector and raster data at the same time.

This paper presents a method to perform a spatial query between a vector dataset represented using an R-tree and a raster dataset represented using a compact and space-efficient data structure called k^2-tree that saves main memory space. Therefore, the method described in this paper solves two problems: first, it can be used to evaluate queries between vector and raster data without having to convert one of the datasets to the other data model; and second, it saves main memory space, thus obtaining a more scalable system.

Keywords: spatial databases, query processing, spatial query, vector data, and raster data

Received 00 January 2009; revised 00 Month 2009

1. INTRODUCTION

The efficient management of spatial information has been a field with much research interest over the last decades that has produced several spatial data models. On the conceptual level, these models describe the space using two different approaches: *object-based spatial models* and *field-based spatial models* [1]. Considering the logical level, spatial data models can be divided into two categories: *vector models* that represent geographic information using finite sequences of points and line segments, and *raster models* that represent geographic information partitioning space into a finite grid of cells and assigning a value to each cell [2]. Even though any logical spatial model can be used to represent any conceptual spatial model, vector models are often used for object-based models and raster models are often used for field-based models.

Current trends like the open data movement, the interoperability programs, and the spatial data infrastructures have greatly increased the availability of different types of spatial data. Most national geographic agencies are using digital cartography and many are publishing cartography using spatial data infrastructures. Public and private organizations offer free and commercial geographic information using airborne and satellite remote sensing technologies. Finally, the quality of user-generated geographic information is increasing and it is becoming an alternative to the information provided by government agencies and private industry. These trends result in high-quality vector and raster information being readily available to practitioners. Furthermore, current application scenarios often require querying vector and raster data at the same time. For instance, public health management services require retrieving populated places (vector data) that will exceed 30 degrees celsius (raster data), or the neighborhoods of a big city (vector data) with levels of nitrogen dioxide above 40 micrograms/m^3 (raster data).

In order to support the increasing complexity and storage requirements of geographic information and the new application scenarios, several efficient data structures, algorithms, and access methods have been proposed for each category of logical spatial models [3, 4]. However, most of the authors focus their work on querying information represented using one of the spatial models, and little research work has been dedicated to index structures and query algorithms where both types of information are used. The appearance of the new NoSQL systems [5, 6], or the column databases [7], which try to face many of the problems of classical database management systems, bring new ways of storing and managing spatial data since many of them have been coupled with spatial information support [8, 9, 10]. However, in many cases, that support is limited [11]. For example, most of them support either only vectorial data or only raster data [12], in the best cases with the data types and operations of the *Simple Features* standard of Open Geospatial Consortium (OGC) [13] and classic index methods such as R-trees [10, 8].

Another field that has also received much attention in the last years is the research on compact data structures [14, 15]. The rationale behind this research line is that compact data structures can be stored in upper levels of the memory hierarchy. Moreover, these structures manage data directly in compressed form and they can access a given datum without decompressing the whole dataset from the beginning. Therefore, it is possible to store large portions or the entire dataset in main memory, and access times are reduced by several orders of magnitude. Therefore, they are suitable for memory-limited scenarios such as applications with huge datasets (e.g., querying remote-sensed and meteorological information, or analyzing the behavior of spatio-temporal objects such as ships, planes, cars or people using their recorded tracks). However, research on spatial access methods and query algorithms focuses on disk-based structures and few research work has considered compact data structures.

In this paper we present a method to perform a spatial query between a vector dataset represented using an R-tree [16, 17] and a raster dataset represented using a compact data structure called k^2-tree [18]. Our method has two key advantages over existing proposals: first, it can be used to evaluate queries between vector and raster data without having to convert one of the datasets to the other data model; and second, raster data is represented using a compact data structure to achieve better access times and a more scalable system. The usual strategy to manage huge raster datasets is to use big hardware resources, instead, the main contribution of this paper is that we use a more intelligent software approach to make a better usage of the limited main memory. We still maintain the classic R-tree for the vector dataset instead of using a compact data structure, yet this is a pragmatic decision since nowadays the R-tree is the de facto standard in the industry to deal with vector information, and it is difficult to improve its features.

Our method is able to answer several queries involving vector and raster data. Among them, we can provide the Minimum Bounding Rectangles (MBRs) stored in an R-tree that overlap regions of a raster having values in a range $[v_b, v_e]$. In our experiments, we compare our proposal against two baselines that compute that query storing the complete input raster in main memory. We have found that our method is specially suited for datasets that have a moderate number of different values (in the order of thousands, i.e., using integer value of the cells), which is a reasonable assumption considering the spatial locality property of spatial data stated by Tobler's law [19]. Our results show that our approach uses up to 18 times less space of main memory and the query time is up to 19 times faster than the baselines.

The rest of the paper is structured as follows. Section 2 presents related work regarding querying vector and raster data and the use of compact structures to represent spatial information. Section 3 presents the k^2-tree data structure in detail. Section 4 shows our algorithms to query raster datasets. Section 5 presents our experimental study. Finally, Section 6 analyses the results of the experiments and presents the conclusions and future work.

2. RELATED WORK

In this section, we review works that deal with querying vector and raster data at the same time. The k^2-tree will be presented in the next section. The R-tree is not described because it is a well-known structure.

It is widely accepted that spatial information is better represented in systems that are capable of managing both raster and vector data, since depending on the type of spatial data, it may be more appropriate to use one of the two models [20]. The models and languages described in [21, 22, 23] include data types and operators to represent and query vector and raster data in traditional database models. However, these models force the user to know and use two different sets of operations, one for vector data and another one for raster data. In [24], a single data model and language is used to represent vector and raster data. However, the model is based on constraints, and therefore it might not be intuitive and friendly for many users. Finally, [25] describes a model based on multidimensional arrays to manage scientific data that could also manage vector and traditional data using arrays. However, these proposals only describe data types and operators to process raster information, and in some cases, to manipulate jointly raster and vector information, yet nothing is said about implementation issues.

Since there is no OGC standard, real systems supporting raster datasets adopt the *de facto* standard called *Map Algebra* [26, 27]. The Map Algebra defines a set of primitive operations which have rasters as input. Examples of real systems including Map Algebra can be the ArcGis of ESRI, QGIS, GDMS-R [28] or GRASS. In some systems, as for example ArcGis, QGis, or GRASS, the zonal statistics operation of Map Algebra admits as input one vector dataset. However, at least in ArcGis and in GRASS, the vector dataset is converted to raster before running the query [29, 30]. GDMS-R is capable of homogeneously integrating different data sources, including vector and raster. This system also provides a homogeneous way of querying those data sources by means of an extension of SQL. However, in [28], there are no implementation details or details on operations involving raster and vector datasets, yet in an example shown in the paper, seems that to operate over raster datasets and vectorial datasets, a previous conversion of the vector dataset to raster is needed.

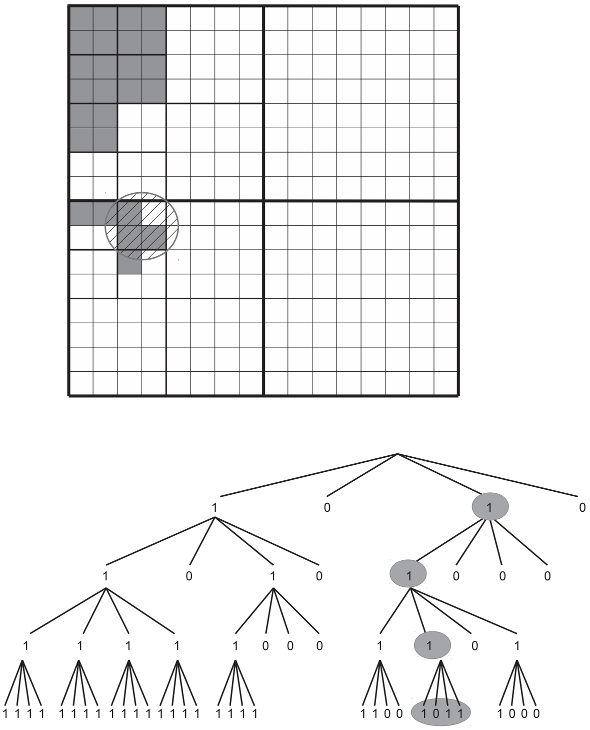
To the best of our knowledge the only works that face the algorithms, data structures, and access methods needed to process jointly raster and vector information are [31, 32]. The first one presents five algorithms for processing joins between an R-tree and a linear region quadtree [33]. As in our case, these works deal with the filtering step of the join and they leave apart the refinement step that requires a computational geometry algorithm. The second paper [32] faces a different problem because it studies the predictive join between regions and moving objects. They use again a linear region quadtree for the raster information, but for the moving objects, they need to use a variation of the R-tree called TPR*-tree. Both works deal with a binary raster. With respect to these proposals, we use a compact data structure to represent any raster data (not only binary data). Using the k^2-tree is conceptually the same as using a region quadtree, yet it is a compact representation designed to allow the management of the data in compressed form in order to save space in main memory.

The use of compact structures to represent spatial information is not a new idea. Until now, the best exponent of this new trend is the wavelet tree, which is succinct-space variant of a classical structure by Chazelle [34]. In [35], the authors propose a new point access method based on the wavelet tree that is competitive in terms of query time efficiency with a K-d-tree [36] while requiring significantly less space. In order to perform several data-analysis queries, [37] designed a structure based on the wavelet tree [38], which represents a set of points in $\mathbb{R}^2$, each one with an associated value given by an integer function. Finally, [39] introduced four new space-efficient structures for general sets of minimum bounding rectangles that offer large space reductions while retaining a time performance that is reasonable for many applications. However, these approaches do not deal with the problem of querying vector and raster data together.

3. THE K^2-TREE

The k^2-tree [18] is a representative of a new family of data-structures and techniques that advocate to take advantage of the higher bandwidth and lower latency of the upper levels of the memory hierarchy [40, 41, 42]. The target is to keep the data in main memory, avoiding disk accesses. For this sake, a new research line proposes to represent the information in a succinct (or compact) way using compact data structures [14, 15], which allow to access the data without decompressing them. Therefore, the k^2-tree has three nice features: it spatially indexes the dataset, it can allocate large datasets in main memory, and it allows its processing without decompressing the data structure.

The k^2-tree is a data structure for the compact representation of sparse binary matrices. The k^2-tree is based on a recursive partition of a binary matrix. First, the matrix is divided into k^2 submatrices of equal size, where k is a parameter. In the next step, each of those matrices is divided into k^2 submatrices, and so on. We construct a conceptual tree, where each submatrix has a node; the root node has k^2 children, which correspond to a left to right and top to bottom ordering of its submatrices. Recursively, each of these nodes has k^2 children, and so on. Each node in the tree is represented using a single bit: 1 if the submatrix has at least one 1, or 0 otherwise. The method continues the partition recursively adding k^2 children to each node with value 1 until the current submatrix is full of 0s or we reach the cells of the original matrix. Figure 1 shows an example of k^2-tree representation for a binary matrix.



T : 1010 1010 1000 1111 1000 1101

L : 1111 1111 1111 1111 1111 1111 1100 1011 1000

FIGURE 1. A binary matrix and its k^2-tree representation, with $k = 2$.

The tree is conceptual in the sense that it is never stored as such. Bitmaps (binary sequences) are the basis of the k^2-tree. A bitmap $B[1,n]$ stores a sequence of n bits and provides efficient solutions for three basic operations: $rank_a(B,i)$ counts the occurrences of the bit a in $B[1,i]$, $select_a(B,i)$ locates the position for the i^{th} occurrence of a in B, and $access(B,i)$ returns the bit stored in $B[i]$. Several data structures (see [43] for example) allow to solve these operations in constant time and using $n + o(n)$ bits of total space, in practice, around 5% extra space over the original bitmap. The simplest way is to store at regular intervals accumulators storing the number of 1s (or 0s) until its position.

The k^2-tree is implemented in a very compact way using two bitmaps: T stores all levels except the last one, and L stores the last level. The bits in T are placed following a levelwise traversal: first the k^2 binary values of the children of the root node, then the values of the second level, and so on. L stores the last level of the tree, comprising the cells values in the original matrix. Observe in Figure 1, the first four bits of T (1010) correspond to the four children of the root, the first and third bits are set to one, this means that the first and third 8×8 matrices (from left to right and top to bottom) have at least one bit set to one. Recursively, the following four bits (1010) correspond to the decomposition of the first 8×8 matrix, whereas the third set of four bits (1000), corresponds to the third 8×8 matrix. The process continues until we reach the last level of the tree, which is represented in L.

This organization allows efficient navigation over the conceptual tree by performing $rank$ operations on the bitmap T. Given a value 1 at position p in T, its k^2 children will start at position $p_{children} = rank_1(T,p) \cdot k^2$. When the position of the children of a node returns a position $p_{children} > |T|$, we access instead $L[p_{children} - |T|]$ to retrieve the actual value of the cells. As an example, observe in Figure 1 the marked bits in the tree. In the first level, the marked bit is the third one, which is at position 2, since the bitmap starts at position 0. Then the four bits corresponding to its children start at position $rank_1(T,2) \cdot 2^2 = 2 \cdot 4 = 8$. In that position, the set of four bits is 1000, the children of the node represented by the first bit of that sequence start at $rank_1(T,8) \cdot 2^2 = 5 \cdot 4 = 20$. The marked bit at position 21 would have its children at $rank_1(T,21) \cdot 2^2 = 12 \cdot 4 = 48$, that exceeds the 24 values of T, and thus it points to $L[48-24] = L[24]$, which is the set of four bits (1011) corresponding to the 4×4 submatrix shadowed in the matrix.

The k^2-tree supports queries that ask for the value of a single cell, row/column queries or general range reporting queries (i.e., report all the 1s in a range of rows/columns). All the algorithms are similar, and involve a top-down traversal of the conceptual tree. Starting at the root (position 0 in T), we check the children of the current node whose submatrices intersect with the region of interest (a single node if we are asking for a cell, k if we are asking for a row/column, etc.). If the child is a region of zeros ($T[p] = 0$) the current branch is discarded. In other case, we find the children of the current node using the explained rank operation and repeat the process recursively. The space partitioning used allows us to compute easily which of the submatrices should be explored and to keep track of the actual row/column for each result found in L.

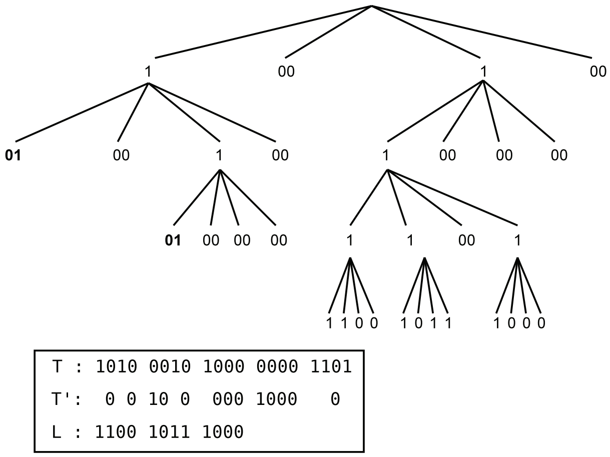
Different enhancements have been proposed to the original k^2-tree. Among them, there is a dynamic version of the k^2-tree called dk^2-tree [44], which supports all the query algorithms of the k^2-tree and allows us to change the contents of the binary matrix efficiently.

3.1. k^2-tree with compression of ones

The k^2-tree was designed for the representation of sparse matrices. It obtains good compression results when the matrix has a relatively small number of ones and they are somehow clustered. However, when representing raster data the binary matrix representing the regions is expected to have large regions full of ones. To better represent matrices with large regions of ones, k^2-tree variants with compression of ones have been proposed [45].

The k^2-tree variants with compression of ones are based on a change in the matrix decomposition: instead of stopping the decomposition process only when the current submatrix is full of 0s, decomposition stops when the current submatrix is *uniform* (it is either full of 0s or full of 1s). This change reduces the number of nodes in the conceptual tree because the subtrees that cover regions full of 1s can be simply replaced by a single node in the new conceptual tree. However, a single bit does not suffice now to encode each node because we must distinguish between internal nodes and two kinds of leaves: those that represent regions of 0s (*white* nodes) and those that represent regions of 1s (*black* nodes).

In this paper we will work with the *2-bits variant* presented in [45] because it obtains the best compression and query times in general. In this representation, each internal node is still encoded with value 1, and leaf nodes are assigned the value 0. However, each leaf node (except those in the last level of decomposition) is assigned a second bit to indicate whether it represents a region of 0s or a region of 1s. In the last level of decomposition each node corresponds to a cell, so a single bit is used to store the actual value of the cell. The actual k^2-tree data structure is stored in 3 bitmaps: T stores the first bit from each node except those in the last level, T' stores the second bit of all the leaves, and L stores the bits of the last level of the conceptual tree. Figure 2 shows a k^2-tree with compression of ones and the bitmaps generated for the matrix of Figure 1. The k^2-tree with compression of ones can be navigated as efficiently as the original k^2-tree, just performing an additional check at T' when we find a 0 in T.



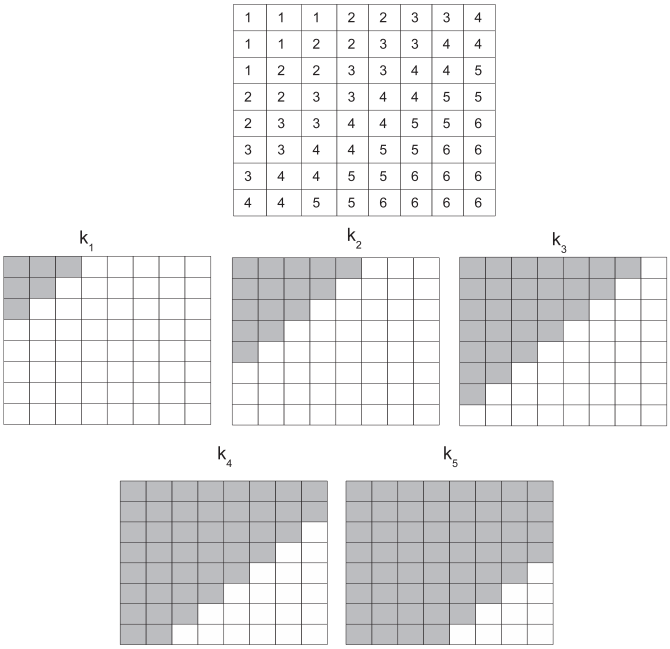
```
T : 1010 0010 1000 0000 1101
T':  0 0 10 0  000 1000   0
L : 1100 1011 1000
```

FIGURE 2. A k^2-tree with compression of ones.

Observe that the k^2-tree with compression of ones uses the same partitioning of the space as the classic *region quadtree* [46, 47], yet the data structures are different. The k^2-tree with compression of ones has been compared with the linear region quadtree [3] to represent binary raster images. In real raster coverages the k^2-tree representation has been proved to require much less space than a classic linear quadtree implementation. Its access times are also several times faster than a linear quadtree, even if the linear quadtree is stored completely in memory [45].

3.2. k^2-trees to represent a non-binary variable

With the k^2-trees shown until now, we can only represent binary rasters, like for example, the area covered by a flood. A structure based on the k^2-tree capable of efficiently storing rasters representing a non-binary variable was presented in [45]. Assume that the raster cells can store m different values and that v_i denotes the i^{th} value in ascending order. To represent a raster dataset, we can use a collection of $m-1$ k^2-trees, $k_{v_1}, k_{v_2}, \ldots, k_{v_m-1}$. In the i^{th} k^2-tree, the cells where the value in the original raster is less than or equal to v_i are set to 1. An example can be seen in Figure 3. In the upper part we show a raster storing values between 1 and 6 in each cell. The five binary matrices for values 1 to 5, that would be represented using a k^2-tree for each matrix, can be seen in the lower part of the figure. Observe that we do not need to store the k^2-tree for value 6, since such a tree would have all its cells set to 1.



1	1	1	2	2	3	3	4
1	1	2	2	3	3	4	4
1	2	2	3	3	4	4	5
2	2	3	3	4	4	5	5
2	3	3	4	4	5	5	6
3	3	4	4	5	5	6	6
3	4	4	5	5	6	6	6
4	4	5	5	6	6	6	6

FIGURE 3. A collection of binary matrices representing a raster.

Using this structure, we can efficiently answer different queries (there is a detailed analysis of space and time efficiency in [45]):

- To obtain the value of a given cell, we can binary search the first k^2-tree that contains the desired cell set to 1. In our example, if we wish to know which is the value of the cell in the eighth row and fourth column, the first k^2-tree with that position set to 1 is the fifth one, so that cell has value 5.
- To obtain the cells with a given value v_j, we have to access two trees, k_{v_j} and $k_{v_{j-1}}$. The result are the cells of k_{v_j} which are set to 1 while in $k_{v_{j-1}}$ were set to 0. If we wish to know which cells have value 3, we take k_3 and we subtract k_2. That is, we set to 0 all the cells of k_3 where k_2 has a 1 (see Figure 4). Efficient set operations over k^2-trees are described in [48].

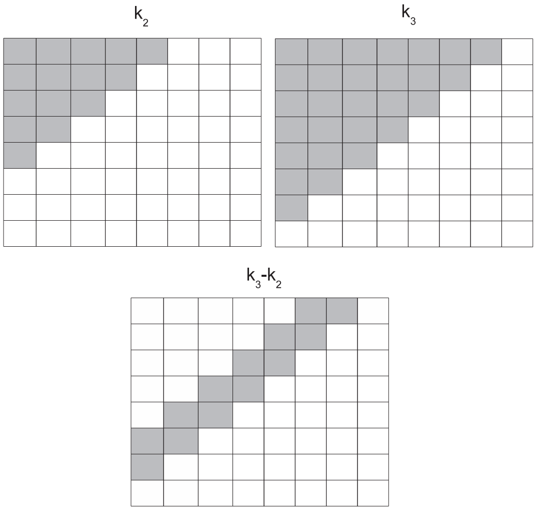


FIGURE 4. Cells with value 3.

- The process to get the cells with values within a given range $[v_b, v_e]$ is equal to the previous case, but we subtract k_{v_b-1} to k_{v_e}. Observe that by simply accessing two k^2-trees we can obtain the cells with a given value or with values in a given range.
- We can also obtain the cells with a value higher or lower than a given value v_t. The cells with a value higher than v_t are those set to 0 in k_{v_t}, whereas the cells with values lower than v_t are those set to 1 in $k_{v_{t-1}}$.

4. QUERYING VECTOR AND RASTER DATA

In this work we tackle a type of query that we called *range query*. By a range query, we mean that we want to obtain the set of polygons of a vector dataset that intersect a region of the raster dataset where the cells have values in a given range $[v_b, v_e]$. This query is a variation of the query *Regions Of Interest* (ROI) [49, 50]. ROI only deals with raster datasets and returns the regions of the dataset that have values in a query range. Therefore, we add to the ROI another restriction, we return those ROI that intersect an MBR stored in an R-tree.

We deal with two variants, the first one returns the polygons that intersect a region where *some* of the cells have values within the given range, also known as *weak semantics*, whereas the second one returns the polygons that intersect a region where *all* cells have values within the given range, also known as *strong semantics*. We also deal with a variation of the previous query, where one of the boundaries of the range is not limited. This transforms the query in: obtain the polygons such that *all* the cells they intersect have values higher (lower) than a given value (strong semantics) or obtain the polygons such that *some* of the cells they intersect have values higher (lower) than a given value (weak semantics).

The input of the algorithm is an R-tree that indexes all the minimum bounding rectangles (MBRs) from the vector dataset and the set of k^2-trees that represents the raster dataset. R-trees are classical indexes in the sense that they serve to speed up queries over a dataset, but the actual data must be stored apart, usually in disk. R-trees only contain the MBRs of the spatial objects because their actual representation requires a much larger space and more complex computations, rendering the index useless. Therefore, the rest of the section considers the vector dataset as a collection of MBRs. Regarding the raster dataset, observe that the k^2-trees store the actual data, and therefore it can recover the original raster. Therefore, in our scenario, the raster information does not require any other additional data either in memory or disk, hence saving even more space and processing time.

The result of the algorithm is composed of two lists: a list of *definitive results* and a list of *probable results*. The difference is that MBRs in the list of definitive results fully intersect a region in the raster with values in the given range, whereas MBRs in the second list intersect a region in the raster with *some* values in the range. Therefore, the elements in the second list have to go through an additional refinement task that uses the complete geometric value of the spatial object to check whether the intersection holds or not. This task is not considered in the algorithm description and the experimental evaluation because it is a common task when spatial indexes are used to solve queries.

4.1. Range queries

4.1.1. Preliminaries

To solve range queries, we use the R-tree and two of the k^2-trees storing the raster dataset, since as explained in Section 3.2, checking the extremes of a range is enough to obtain the regions of the raster having values within that range, and we do not need to process the k^2-trees corresponding to the values *inside* the range. More precisely, if our range is $[v_b, v_e]$, we only need to work with k_{v_b-1} and k_{v_e}. This is one of the key features of our method. The k^2-trees are compact data structures, they are designed to occupy little space combining several strategies. In any case, the two k^2-trees will occupy much less space than the original dataset, and thus our technique will save big amounts of main memory space.

Figure 5 shows our running example, it displays two rasters corresponding to two k^2-trees $k_{v_{b-1}}$ and k_{v_e}, used to solve the range query $[v_b, v_e]$. We have also represented over $k_{v_{b-1}}$ six polygons and the MBRs of the R-tree indexing them. The rectangles with dotted lines are the first level MBRs, whereas the rectangles with solid lines are the second level MBRs. Finally, we have also represented over k_{v_e} the same objects with their second level MBRs and the divisions of the first and the second level of the k^2-tree. We do not include the first level MBRs and the third level division of the k^2-tree in order to avoid cluttering the figure. In this example, the divisions of the space in $k_{v_{b-1}}$ are the same as those shown for k_{v_e}, and thus we do not include them to simplify the figure.

Given that to know whether there are cells within an MBR with values in the query range, we have to subtract the cell values that overlap the MBR in $k_{v_{b-1}}$ to those in k_{v_e}, many operations of the algorithm are based on a check of an MBR in both k^2-trees to compute whether the raster cells overlapped by the MBR are all 0s, all 1s, or a mixture of 0s and 1s. Therefore, we define the concept of *colour* to describe the result of checking the values of all the cells overlapped by an MBR r in a raster represented by a k^2-tree. The *colour* of an MBR r can be:

- *white* if all the cells overlapped by r are 0s.
- *black* if all the cells overlapped by r are 1s.
- *gray* if the cells overlapped by r are a mixture of 0s and 1s.

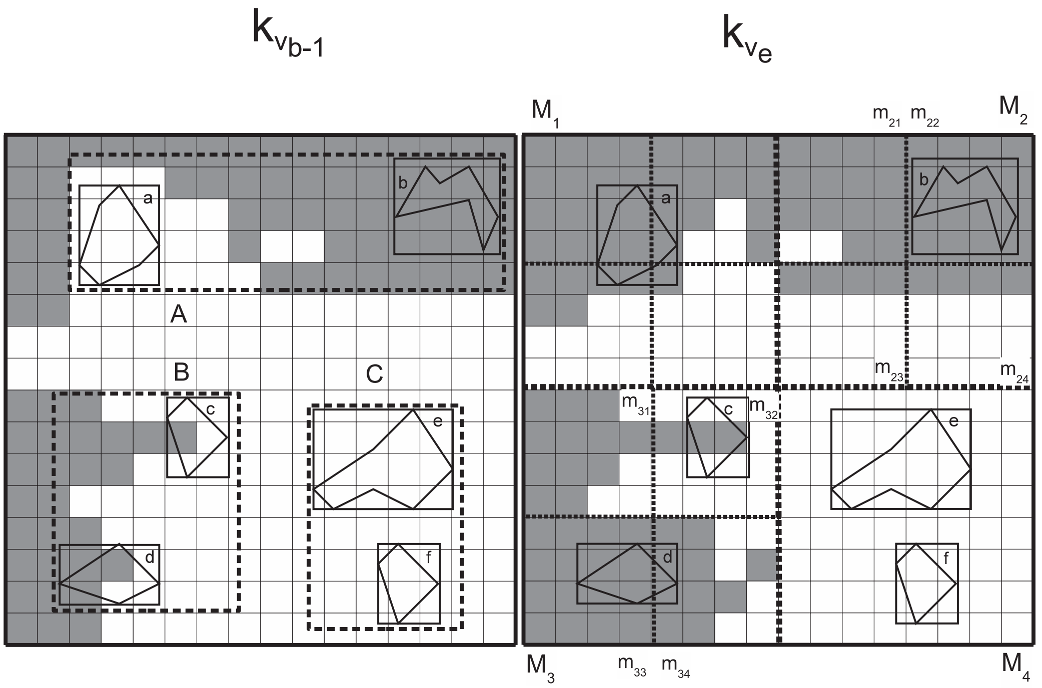


FIGURE 5. The rasters stored at two k^2-trees corresponding to the values v_{b-1} and v_e and four spatial objects with their MBRs.

$k_{v_{b-1}}$ colour	k_{v_e} colour	Result
white	white	the region does not have values in the given range
white	gray	the region has *some* values in the given range
white	black	the region has *all* values in the given range
gray	gray	we do not know
gray	black	the region has *some* values in the given range
black	black	the region does not have values in the given range

TABLE 1. Possible results based on the *colour* values.

Comparing the *colour* of an MBR r in two k^2-trees we can quickly determine the result of subtracting all the cells in both k^2-trees as shown in Table 1. The algorithm can use this result to decide whether it is necessary to continue exploring downwards the k^2-tree or whether it can stop now and return a result.

Observe that when the two *colours* are gray, we cannot conclude anything. We have to go through a more detailed analysis that, in a extreme case, can reach the cells of the raster in order to count the number of ones in the regions overlapped by r, and if there is an increment, then r has *some* values in the given range.

To compute the *colour* value, the algorithm uses two functions: *getMinMatrix* and *rangeSearch*. The function *getMinMatrix* takes an MBR and a reference of a node in the k^2-tree and returns a pair ⟨k,colour⟩, where k is the deepest node (descendant of the node of the k^2-tree provided as input) that completely contains the given MBR, whereas none of its children do. The operation to determine the *colour* of that node is simple because if the node is a leaf (*colour* = black or *colour* = white) the bit representing such a node in T' indicates whether it represents a region of 0s or a region of 1s. On the other hand, if the node is not a leaf, then *colour* is always equal to gray (see Section 3.1 and Figure 2 for further details).

As an example, if we check in Figure 5 the MBR d in k_{v_e}, *getMinMatrix* returns M_3, given that none of its children completely contain d, and the *colour* is *gray*, since M_3 has cells set to 1 and set to 0. If we check the MBR e, *getMinMatrix* returns the submatrix M_4 and the *colour* is white. Finally, if we check b, the result is m_{22} with *colour* black.

The function *rangeSearch* takes an MBR and a reference of a node in the k^2-tree as well. The function returns whether the *exact* cells of the raster that intersect the MBR are all 0s, all 1s, or a mixture. This operation is more complex than *getMinMatrix* because it must navigate down all the k^2-tree branches that intersect with the MBR and retrieve all the cells in the k^2-tree that intersect the MBR in order to compute the *colour* that is returned.

Using the MBR d from the previous example, *rangeSearch* inspects the cells of the raster that are overlapped by the MBR. Therefore, the *colour* is gray in the raster $k_{v_{b-1}}$, whereas in k_{v_e}, the *colour* is black.

Algorithm 1 R-tree × Raster with a range query

```
 1: Let R be an R-tree
 2: Let Ra be a $k^2$-tree based representation of a raster $Ra = \langle k_{v_1}, k_{v_2}, \ldots, k_{v_{m-1}} \rangle$
 3: Let $v_b$ and $v_e$ the values delimiting the range
 4: Let D and P be lists with pointers ⟨MBR, oid⟩
 5: Let S be a stack with entries ⟨⟨MBR, ref, oid⟩,⟨k,x,y,level,offset⟩,⟨k,x,y,level,offset⟩⟩
 6: kCur$_{b-1}$ ← ⟨$k_{v_b-1}$,0,0,0,0⟩
 7: kCur$_e$ ← ⟨$k_{v_e}$,0,0,0,0⟩
 8: for all rRoot ∈ R.ref do
 9:     push(S, ⟨rRoot,kCur$_{b-1}$,kCur$_e$ ⟩) {Insert in the stack all entries at the root node}
10: while S ≠ empty do
11:     ⟨rCur,kCur$_{b-1}$,kCur$_e$⟩ ← pop(S)
12:     ⟨kNew$_{b-1}$,colour$_{b-1}$⟩ ← getMinMatrix(rCur.MBR, kCur$_{b-1}$)
13:     ⟨kNew$_e$,colour$_e$⟩ ← getMinMatrix(rCur.MBR, kCur$_e$)
14:     if colour$_{b-1}$ = white and colour$_e$ = black then
15:         if isLeafNode(rCur) then
16:             add(rCur.oid, D) {Adds the objects in that node as definitive}
17:         else
18:             addDescendants (rCur.ref, D) {Adds all objects in descendant leaves as definitive}
19:     else if colour$_{b-1}$ = white and colour$_e$ = gray then
20:         ⟨S,D,P⟩ ← continueSearch(rCur, S, kNew$_{b-1}$, kNew$_e$, D, P )
21:     else if colour$_{b-1}$ = gray and colour$_e$ = black then
22:         ⟨S,D,P⟩ ← continueSearch(rCur, S, kNew$_{b-1}$, kNew$_e$, D, P )
23:     else if colour$_{b-1}$ = gray and colour$_e$ = gray then
24:         ⟨S,D,P⟩ ← continueSearch(rCur, S, kNew$_{b-1}$, kNew$_e$, D, P )
25: return ⟨D,P⟩
```

4.1.2. The algorithm

Algorithm 1 shows the main body of the algorithm. The two versions of the query can be obtained by using two variants of the function *continueSearch*: the one in Algorithm 2 for weak semantics, and the one in Algorithm 3 for strong semantics. The algorithm uses a stack (S in the pseudo-code) containing the branches of the R-tree that have to be checked. Each entry has a pointer to an entry of an R-tree node and two pointers to nodes of the two considered k^2-trees, with the following format:

- Pointers to R-tree nodes: ⟨MBR, ref, oid⟩. *MBR* is the MBR of the entry, *ref* is a list of references to children nodes (*null* in leaf nodes), and *oid* is a list of object ids that are included in the MBR (null in internal nodes).
- Pointers to k^2-tree nodes: ⟨k,x,y,level,offset⟩. k is the k^2-tree, x, y are the coordinates of the top-left cell of the submatrix corresponding to the k^2-tree node, *level* is the level of the node, and *offset* is the position in T of the bit representing such node.

The algorithm stores the result in two lists: D is the list of *definitive* results, whereas P is the list of *probable* results. The entries in those lists have the format ⟨MBR, oid⟩, where MBR and oid have the same meaning as in the pointers to the nodes of the R-tree of the stack. The function $add(rCur.oid, D)$ adds the objects pointed by $rCur.oid$ to the list D, whereas $add(rCur.oid, P)$ adds the objects to the list P. The function $addDescendants(rCur.ref, D)$ accepts as a parameter a list of pointers to nodes of the R-tree ($rCur.ref$), traverses the subtrees rooted at those nodes until finding all the leaves of those subtrees, and adds them to the list D.

The algorithm starts inserting the entries of the root node into the stack (lines 8-9). Then the main loop (lines 10-24) proceeds extracting the top of the stack. Using the extracted MBR, it applies the function *getMinMatrix* twice, one for each of the two k^2-trees. Therefore we have two pointers to nodes of k_{v_b-1} and k_{v_e}, respectively, and two values of colour: $colour_{b-1}$ and $colour_e$. These values are managed according Table 1. If one or both are gray, all the possible cases are managed by the function *continueSearch* that receives the pointer to the processed node of the R-tree ($rCur$), the stack S, the two pointers to nodes of the two k^2-trees ($kNew_{b-1}$ and $kNew_e$), and the two lists D and P.

The *continueSearch* for the weak semantics is shown in Algorithm 2. First, if the pointer *rCur* points to an internal node of the R-tree, then we introduce in the stack its descendants (with the same pointers to k^2-trees), in order to be processed later (lines 3-4), and the flow returns to the main loop. If the pointer to the R-tree is to a leaf node, then the algorithm uses the *rangeSearch* function to inspect the nature of the cells intersecting the processed MBR in both k^2-trees (lines 6-7). Again, we have two values of colour ($colour_{b-1}$, $colour_e$) managed according Table 1. This finishes the process of the processed R-tree node, and then the flow returns to the main loop. The strong semantics version of *continueSearch* is shown in Algorithm 3. It is a simplified version of the Algorithm 2 since we only add a result when the pair white/black is obtained.

In order to illustrate the algorithm, we are going to follow the algorithm with *weak semantics* using Figure 5. The processing of the root of the R-tree (lines 8-9) inserts in the stack the entries corresponding to the MBRs A, B, and C. In the case of A, *getMinMatrix* returns the whole 16×16 matrices in both $k_{v_{b-1}}$ and k_{v_e}, and the *colour* is gray in both trees. According to Table 1, nothing can be concluded, and then the algorithm inserts in the stack the descendant MBRs of A (a and b). In the case of B, *getMinMatrix* returns the submatrix M_3 in both trees, with *colour* gray in both cases, so again, the descendants of B are inserted in the stack. When processing C, *getMinMatrix* returns M_4 and *colour* white in both trees. Therefore, according to Table 1, it is sure that none of the descendants of C will overlap cells with values in the given range, and thus we can prune the R-tree branch corresponding to C. Observe that here is the biggest gain, since already in the first level of the R-tree and in the second level of the k^2-trees, we find that there is no need to inspect the subtree rooted at C. Here, we can see the index effect of the k^2-tree since we stop the search without checking the details of the raster.

Now, we analyze the MBRs that remain in the stack (a, b, c, and d). When b is processed, *getMinMatrix* returns m_{22} and *colour* black in both trees, so it can be discarded without further inspect.

The case of d is a bit more complex, *getMinMatrix* returns M_3 and *colour* gray in both trees, therefore *getMinMatrix* did not help, and since d is in a leaf of the R-tree, we have to use rangeSearch, which examines the exact cells of the raster that overlap d. The *colour* is gray in $k_{v_{b-1}}$ and black in k_{v_e}, then we can add it as a *probable* result.

Something similar occurs with a, *getMinMatrix* returns M_1 and *colour* gray in both trees. rangeSearch gives white in $k_{v_{b-1}}$ and black in k_{v_e}, then this a *definitive* result.

The case of c is the worst one. Even using *rangeSearch*, the result is gray in both trees, then in this case we have to count the number of black cells in both rasters intersecting it. Given that in this case the number of ones increases from 1 to 2, we have a *probable* case.

Observe that when we add a descendant MBR to the stack, we add the three pointers indicating the point where we can resume the search in the three trees. Following our example, after the first application of *getMinMatrix*, we concluded that B should be substituted by c and d. Then for each of those MBRs, we add a pointer to that MBR and two pointers to M_3 in $k_{v_{b-1}}$ and k_{v_e}. The pointers to the k^2-trees nodes include, in addition to the offset of the node in the T bitmap, the coordinates of the top-left cell of the submatrix and the level of the node in order to be able to resume the search in that node.

4.2. Range queries with only one boundary

Algorithm 4 shows the main body of the algorithm. To solve this query we only have to check the k^2-tree corresponding to the query value (v_t), thus saving even more main memory. If the query is the version *higher*, we use k_{v_t}, whereas in the case of *lower*, we use k_{v_t-1}. The rest of the algorithm is very similar to the previous one, but we only deal with one k^2-tree and thus only one colour value.

Algorithm 2 ContinueSearch for weak semantics

```
continueSearch(rCur, S, kNew$_{b-1}$, kNew$_e$, D, P )
if isInternalNode(rCur) then
    for all rNew $\in$ rCur.ref do
        push(S, $\langle$rNew, kNew$_{b-1}$, kNew$_e$ $\rangle$)
else
    colour$_{b-1}$ $\leftarrow$ rangeSearch(rCur.MBR, kNew$_{b-1}$)
    colour$_e$ $\leftarrow$ rangeSearch(rCur.MBR, kNew$_e$)
    if colour$_{b-1}$ = white and colour$_e$ = black then
        add(rCur.oid, D) {Definitive}
    else if colour$_{b-1}$ = white and colour$_e$ = gray then
        add(rCur.oid, P) {Probable}
    else if colour$_{b-1}$ = gray and colour$_e$ = black then
        add(rCur.oid, P) {Probable}
    else if colour$_{b-1}$ = gray and colour$_e$ = gray then
        if nOfOnes(rCur.MBR,kNew$_{b-1}$) < nOfOnes(rCur.MBR,kNew$_e$) then
            add(rCur.oid, P) {Probable}
return $\langle$S,C,P$\rangle$
```

Algorithm 3 ContinueSearch for strong semantics

```
continueSearch(rCur, S, kNew$_{b-1}$, kNew$_e$, D, P )
if isInternalNode(rCur) then
    for all rNew $\in$ rCur.ref do
        push(S, $\langle$rNew, kNew$_{b-1}$, kNew$_e$ $\rangle$)
else
    colour$_{b-1}$ $\leftarrow$ rangeSearch(rCur.MBR, kNew$_{b-1}$)
    colour$_e$ $\leftarrow$ rangeSearch(rCur.MBR, kNew$_e$)
    if colour$_{b-1}$ = white and colour$_e$ = black then
        add(rCur.oid, D) {Definitive}
return $\langle$S,D,P$\rangle$
```

When the output of *getMinMatrix* is gray, we have to call again the function *continueSearch*, which again has two versions: Algorithm 5 shows the weak semantics version, whereas Algorithm 6 shows the strong semantics version. The search is solved according to Table 2.

4.3. Discussion

In this article we focus on the description of the algorithm for a classic single-processor machine. However, we give a few pointers on an parallel version of our algorithm that could be used a multiprocessor machine. Different proposals exist to work with R-trees in this environment [51, 52], focusing mainly on improving times in range queries. Join operations between multiple R-trees have also been studied [53]. Distributed query processing using k^2-trees has also

colour $k_{v_t}/k_{v_{t-1}}$	higher	lower
white	Definitive	No
black	No	Definitive
gray	Probable	Probable

TABLE 2. Values of the different operations depending on the *colour* obtained for a given MBR.

Algorithm 4 R-tree × Raster with a range query with only one limit

```
Let R be an R-tree
Let Ra be a $k^2$-tree based representation of a raster $Ra = \langle k_{v_1}, k_{v_2}, \ldots, k_{v_m} \rangle$
Let $v_t$ the value delimiting the range
Let Hi/Lo be a parameter indicating 'higher' or 'lower'
Let D and P be lists with pointers ⟨MBR, oid⟩
Let S be a stack with entries ⟨⟨MBR, ref, oid⟩,⟨x,y,level,offset⟩⟩
if Hi/Lo='>' then
    kCur ← ⟨$k_{v_t}$,0,0,0,0⟩
else
    kCur ← ⟨$k_{v_t - 1}$,0,0,0,0⟩
for all rRoot ∈ R.ref do
    push(S, ⟨rRoot,kCur ⟩) {Insert in the stack all entries at the root node}
while S ≠ empty do
    ⟨rCur,kCur ⟩ ← pop(S)
    ⟨kNew,colour⟩ ← getMinMatrix(rCur.MBR, kCur)
    if (colour = white and Hi/Lo='>') or (colour = black and Hi/Lo='<') then
        if isLeafNode(rCur) then
            add(rCur.oid, D) {Adds the objects in that node as definitive}
        else
            addDescendants (rCur.ref, D) {Adds all objects in descendant leaves as definitive}
    else if colour = gray then
        ⟨S,D,P⟩ ← continueSearch(rCur, S, kNew, D, P, Hi/Lo)
return ⟨D,P⟩
```

Algorithm 5 ContinueSearch for the weak semantics (ranges of only one limit)

```
continueSearch(rCur, S, kNew, D, P, Hi/Lo )
if isInternalNode(rCur) then
    for all rNew ∈ rCur.ref do
        push(S, ⟨rNew, kNew, ⟩)
else
    colour ← rangeSearch(rCur.MBR, kNew)
    if (colour = white and Hi/Lo='>') or (colour = black and Hi/Lo='<') then
        add(rCur.oid, D) {Definitive}
    else if colour = gray then
        add(rCur.oid, P) {Probable}
return ⟨S,C,P⟩
```

been studied for classic k^2-trees [54]. This analysis is designed for Web graph or social networks, but the techniques to distribute data among the nodes and the general steps in the algorithms can be directly applied to our case.

We consider a multiprocessor architecture and a packing of the data among the nodes following a grid-like pattern in the k^2-trees, that was studied in [54]. This partition of the space is similar to the Hilbert curve used in different distributed R-tree representations. A simple distribution of the vector data, either in multiple R-trees or specific data structures, could be matched by an equivalent distribution of the raster data generating the corresponding subtrees for each region of the space. This would keep locality of the data and also keep in each node the relevant fragment of the raster and vector datasets. Our algorithm can then use a master node to distribute the subqueries according to the MBRs in the R-tree among the processors, running in each processor appropriate range subqueries to locate the appropriate paths and child nodes in the k^2-tree and R-tree. The returned values would be treated as candidate results, where MBRs that span across multiple nodes can be joined as shown in Table 1. Assuming sufficiently large contiguous regions of the data are assigned to each processor, the size of the k^2-trees should not increase significantly, and even if we use partial or full replication of the k^2-tree data in all the processors the memory size of the k^2-trees should still be small enough in each processor.

Algorithm 6 ContinueSearch for strong semantics (ranges of only one limit)

```
continueSearch(rCur, S, kNew, C, P, Hi/Lo )
if isInternalNode(rCur) then
    for all rNew ∈ rCur.ref do
        push(S, ⟨rNew, kNew ⟩)
else
    colour ← rangeSearch(rCur.MBR, kNew)
    if (colour = white and Hi/Lo='>') or (colour = black and Hi/Lo='<') then
        add(rCur.oid, C) {Definitve}
return ⟨S,C,P⟩
```

A note on complexity

In general, worst-case analysis for window queries (spatial join among others) processed using spatial data structures such as R-tree are very far from what actually happens in practice because the performance depends on the size (in the case of window query) and on the intersection level of the areas or zones in each level of the spatial structure [55]. Note that our problem can be modeled as a spatial join between an R-tree and a raster whose spatial predicate is the intersection of two areas or zones. In order to get closer to the actual performance of the structure, cost models have been proposed in the literature to estimate their efficiency, proving that they are more useful. For example, [56] proposes a cost model to estimate the cost of a window query using a R-tree and in [57] a cost model to estimate the time consumed by a spatial join considering two R-trees. The development of a cost model for our algorithm is beyond the scope of this paper, because it corresponds to a result in itself. However, in order to establish a preliminary analysis, we next study the performance of our algorithm in both favorable and unfavorable scenarios.

Let M be the number of MBRs in the last level of the R-tree and let n be the number of rows/columns of the raster. A simple baseline approach to solve the query should obtain the MBRs in the leaves of the R-tree, and then, for each one, inspect all cells of the raster that overlap that MBR. The number of cells overlapping an MBR is bounded by the size of the raster, that is, $O(n^2)$. Therefore the time complexity of that baseline approach is $\Theta(M) + \Theta(M \cdot n^2)$.[3]

Now, let us analyze our algorithm. Instead of considering the two k^2-trees needed to solve the query in our algorithm, let us consider only one k^2-tree having a 1 in the cells that match the query and a 0 in the rest. Inspecting that k^2-tree is conceptually the same as inspecting the two k^2-trees. Let l_i, $0 \leq i < \lceil log_k n \rceil$, be the levels of that k^2-tree. The $(k^2)^i$ nodes at level l_i represent regions of size $(n/(k \cdot i)) \cdot (n/(k \cdot i))$. Let r_i be the percentage of the regions represented by l_i having all 0s or all 1s. For those $r_i \cdot (k^2)^i$ regions of l_i, the search ends there, while the rest should continue. Observe that when the search ends in a level before the last one, the rest of the k^2-tree is not inspected downwards for that MBR, but in addition, that MBR is probably in a non-leaf node of the R-tree and if the MBR overlaps a region of 0s, then the traversal of the R-tree downwards also stops in that node without reaching the descendants of that MBR. That is, in our approach, it is possible that our algorithm does not process all leaf nodes of the R-tree.

Let us consider the MBRs in the last level of the R-tree such that in some step of the tree traversal, the search ended before reaching them, let us denote $R = r_1 + r_2 + \ldots + r_{\lceil log_k n \rceil - 1}$ the percentage of those MBRs. For such MBRs, the search is bounded by the height of the tree, that is $O(log_k n)$. For the rest of MBRs, the search must reach the last level and inspect the cells (the leaves of the k^2-tree) overlapping that MBR and thus, the cost is $O(log_k n) + O(n^2) = O(n^2)$.

Therefore, the overall cost of our algorithm is $O(M) + O(M \cdot n^2)$, that is, the worst case is the same as the baseline, but for those MBRs that overlap uniform regions, we remove the quadratic cost, and instead, we have a logarithmic search. This is shown by the above and below bounds in the cost of the baseline, whereas in our cost, both terms have an *upper* bound.

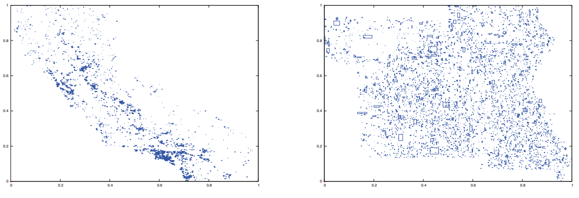
Therefore, the question is the size of R, that is, the percentage of regions of the space that at some level of the tree have an uniform region of 0s or 1s. By Tobler's law, we expect that in real examples, the size of R will be big enough to obtain an average complexity better than that of the baseline approach.

5. EXPERIMENTS

In order to evaluate the performance of the algorithms described in Section 4, we have performed several experiments using real data. The raster datasets are digital terrain models obtained from the Spanish Geographic Institute[4] that represent the spatial elevation of the terrain. The vector datasets are the California Roads (vecca) and Tiger Streams (vects) datasets from the ChoroChronos.org[5] web site. Figure 6 shows the distribution of the MBRs in the two datasets using a representative fraction of the MBRs. The raster and vector datasets were scaled and translated in such a way that they cover the same space.

Dataset	# MBRs	R-tree size (nodes)
vects	194,971	3,130
vecca	2,249,727	33,754

TABLE 3. MBR sets used in the experiments.



(a) vecca

(b) vects

FIGURE 6. MBR distributions of the vector datasets.

Table 3 describes the two sets of minimum bounding rectangles that represent the vector spatial objects. For each MBR dataset, the table describes the number of MBRs in the set, and the number of nodes of the R-tree that indexes the set.

As baselines, we compare our algorithms against two approaches that process a raster dataset which is completely stored in main memory. The first one uses 16-bit integers to represent the value of each cell in the raster, enough to represent all rasters of our experiment. The second baseline uses $\lceil \log(v) \rceil$ bits to represent the cells of the raster, where v is the number of different values in the raster. In both cases, the layout of the cells in main memory is a simple fill curve traversing the original raster row by row.

We used two different sets of raster coverages to test the performance of our technique in two scenarios, which we call *scenario 1* and *scenario 2*. The scenario 1 tries to replicate a real world example: the vector dataset represents spatial objects (e.g., populated places) and we try to find those objects in a given range of elevations. The raster data (i.e., the elevation) has been discretized removing the decimal part of the numbers (i.e., each cell of the raster stores the elevation of that geographic location in meters). Furthermore, we have used five different raster datasets of increasing size. Table 4 describes the five raster coverages of scenario 1. For each raster, the table describes its size in pixels (or cells), the size of the representations that use 16-bit integers and $\lceil \log(v) \rceil$ bits (in parenthesis the number of bits used to represent each cell), the size of the set of k^2-trees representing that dataset, the average size of individual k^2-trees, and the number of different values in the dataset. The most important value for our

[3] Although $\Theta(M) + \Theta(M \cdot n^2) = \Theta(M \cdot n^2)$, we keep the term $\Theta(M)$ with the purpose of giving more precision in the analysis.

[4] http://www.ign.es

[5] http://chorochronos.datastories.org/?q=node/17

method is the average size of a k^2-tree, since to solve our queries, we only need, at most, two of them. Therefore, to solve a query over the dataset of size 16384×16384, we only need $2 * 88.6 = 177.2$ KB on average.

To obtain compression, all methods take advantage of some characteristic of the data. For example, text compressors use the skewed distribution of frequencies of words in human languages [58], or video compression uses the gradual change of colors in adjacent pixels [59]. The k^2-tree follows the same strategy as quadtrees to obtain compression, which requires regions having the same value. Therefore, the discretization of the raster data reduces the number of different values, it increases the chances of having regions with the same value, and thus, it reduces the size of the k^2-tree and it improves the query times. However, it also changes the original dataset. The scenario 2 analyzes the way in which the number of different values affects the performance of our technique. We take a raster file of size 4096×4096 from the same source and we vary the number of different values (by considering different precisions of the spatial elevation values). The second set of rasters is shown in Table 5. The first column gives the size of the representation that uses 16-bit integers, the second column presents the size of the representation that uses $\lceil \log(v) \rceil$ bits (in parenthesis the number of bits used to represent each cell), the third one gives the size of the collection of k^2-trees, the fourth one shows the average size of the k^2-trees, and the last one the number of different values.

From the tables, we can observe that the number of different values has a big impact in the size of the collection of k^2-trees representing the raster. The set of rasters of scenario 1 has a low number of different values, and then the quadtree strategy achieves a good compression, around 10% of the original size. However, observe that in the scenario 2, as the number of different values increases, the size of the k^2-tree representation grows rapidly, reaching a point where the k^2-tree representation starts to be larger than the original one. Nevertheless this is a problem for disk space, where the collection is located, whereas the size of an individual k^2-tree is small in all cases. Observe that even when we have a large number of different values, the average size of a tree is much less than the size of the dataset, and thus, since we only need to load into main memory two k^2-trees to solve a query, we still have big main memory savings.

The tests 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 12.04.5. All algorithms were implemented using Java and Marios Hadjieleftheriou's Java implementation of an R^*-tree. [6] The R^*-tree was configured to use 4KB nodes and 100 entries on each node, and it is the result of a bulk loading.

We measure the values (memory consumption and running time) needed to check whether an MBR is a solution, but in the case of probable solutions, as it is common when presenting spatial indexes, we do not compare the exact geometry to give a final solution neither in the baselines nor in our algorithm. For the memory consumption, we measured the values provided by `java.lang.Runtime.getRuntime()` during the computation, and we report the maximum value. All structures reside on disk at the beginning of the experiment, this means that the running times include the load of the R-tree and the raster representation. However, the result of running the queries (the output) is not written to disk.

As explained in Section 3, in k^2-trees, to obtain the child node of a given node we have to perform a rank operation and a multiplication. When the algorithm is processing a part of the tree, it is likely that it will compute the child of the same node several times. As an enhancement, to avoid performing repetitively the same multiplications and rank operations, our algorithms used a hash table of constant size, which contains pointers to the last 1,000 computed children.

The k parameter of the k^2-trees used in the experiment is 2. The ranges of the queries cover 5% of the possible values. The size of the query range is not important, since as we have already shown, a query is always solved accessing the R-tree and two k^2-trees. What really matters is whether one or both boundaries are represented by k^2-trees that are not sparse. In a non-sparse k^2-tree, when the algorithm applies *getMinMatrix* over an MBR, in many cases the *colour* will be gray, which does not allow to prune parts of the R-tree, and thus the search should continue. The problem can be attenuated if one of the boundaries is sparse, but if both suffer this problem, the times can be affected. The sparser k^2-trees are those at the beginning of the range of possible values, as the 0s will dominate the raster, and those at the end of the range of possible values, as the 1s will dominate the raster (see Figure 3), whereas the values in the middle display a mixture of 0s and 1s that will harm the search times. Another way of seeing the same effect is that, since the k^2-trees follow a quadtree strategy, the sparser trees will be smaller, and thus they will allow faster searches, whereas, the trees corresponding to values in the middle of the range will have the opposite behaviour. Therefore, having queries with random ranges covering 5% of the possible values will produce faster and smaller queries, but the displayed times are the average value of 100 queries, where the boundaries of the range are randomly chosen.

5.1. Main memory consumption

Figure 7 shows the main memory consumption during the computation of range queries using the set of rasters

[6] The author changed his implementation to C++ `http://libspatialindex.github.com/`, however, a slightly modified version of the original Java implementation can be found at `https://github.com/felixr/java-spatialindex`.

	baselines size (MB)		k^2-tree	avg size of	# diff.
Cells	16-bit int	$\lceil\log(v)\rceil$ bits	size(MB)	a tree(KB)	values
1024 × 1024	2	1.0 (8)	0.3	1.5	208
2048 × 2048	8	4.5 (9)	1.5	3.7	413
4096 × 4096	32	20.0 (10)	5.8	8.4	704
8192 × 8192	128	80.1 (10)	25.6	26.9	974
16384 × 16384	512	352.0 (11)	148.5	88.6	1,715

TABLE 4. A set of rasters of increasing size (scenario 1).

baselines size (MB)		k^2-tree	avg size of	# diff.
16-bit int	$\lceil\log(v)\rceil$ bits	size (MB)	a tree (KB)	values
32	16 (8)	3.2	14.6	224
32	20 (10)	51.0	58.7	890
32	24 (12)	666.7	192.2	3,552
32	28 (14)	3,636.7	264.5	14,197

TABLE 5. A set of rasters of size 4096 × 4096 and increasing number of different values (scenario 2).

of scenario 1, strong semantics, and query ranges (that is, with two boundaries). The values of weak semantics and only one boundary do not differ at all. As expected, our technique uses much less main memory space, up to 18 times less. This memory savings yield a much more scalable system, or if we have to process a raster by pieces, it is likely that we will obtain better running times. However, if we have a high number of different values in the dataset, regions with the same value will be rare, and thus the quadtree strategy to compress of the k^2-tree will fail, in other words, the branches of the k^2-tree cannot stop the decomposition, and they will reach the individual cells. This implies that the trees will be taller, and thus they will occupy more space. This can be seen in Table 5, where the average size of the k^2-trees grows as the number of different values increases. The effect over the main memory consumption can seen in Figure 8, that displays the values of the set of rasters of scenario 2. At the beginning, the memory consumption of our approach grows rapidly, but as the number of different values continues to grow, the curve stabilizes at values below the memory consumption of both baselines. This is expected, as the height of the trees for a given raster size is bounded by $\lceil\log_k n\rceil$, where k is the parameter used by the k^2-trees and n the number of rows/columns of the dataset.

The increment in the main memory consumption is not caused by the increment of different values. It is caused by the increment in the average size of the k^2-trees. To show this fact, the next experiment uses a range covering the 99% of the range, this forces to use two trees in the extremes of the range. Since at the beginning of the range the k^2-trees are dominated by 0s and at the end of the range the k^2-trees are dominated by 1s, then it is likely that they will have a similar height (and this height will be small). The results are shown in Figure 9. Clearly, we can see that the number of different values (and thus, the number of k^2-trees in the dataset) does not affect to the memory consumption. Instead, the size of the k^2-trees does affect, and this size increases with the number of different values as it was shown in Table 5.

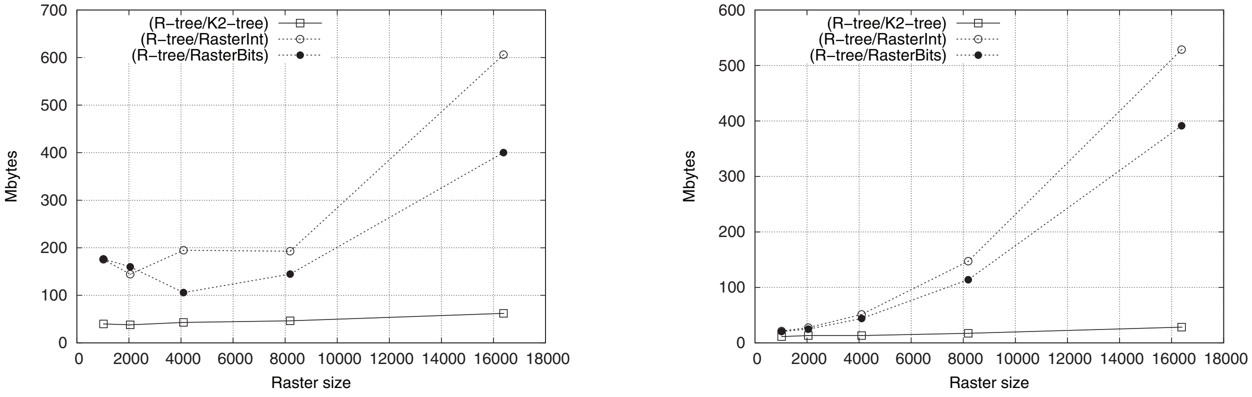
5.2. Running times

Considering that the main goal of the experiments is to measure the memory savings of our technique, we have ignored that the computation times of the baseline are favored by the fact that accessing a plain raster in main memory is really fast, and it is difficult to overcome. Furthermore, the baseline algorithm is provided with all the memory it needs. However, with the following experiments we will show that our method is not penalized by the fact that we are using compact data structures. Instead, we show that query times are better than the baseline with our method most of the times. Furthermore, our proposal is able to fit larger raster datasets in memory that could not be processed by the baseline algorithm.

5.2.1. Ranges with two boundaries

Figure 10 shows the processing time for range queries with strong semantics in the scenario 1. The values with weak semantics are practically equal and thus they are not shown. With the *vecca* dataset our approach is between 2.5 to 5.5 times faster. In the case of the *vects* dataset the improvements are much smaller, between 6% and 34% in the case of the integer-based baseline, and between 61% and 11 times faster in the case of the bit-based baseline. The differences between the two vector datasets is due to the distribution of the MBRs. If the R-tree indexes a distribution that covers most of the space, like in the case of the *vects* dataset, our algorithm has to traverse the two k^2-trees completely and therefore, it has to perform several top-down traversals. A similar phenomenon occurs in common indexes, like in B^+-trees, where searching a value that it is present in most of the records can worsen the search times because the index will not help in the search.

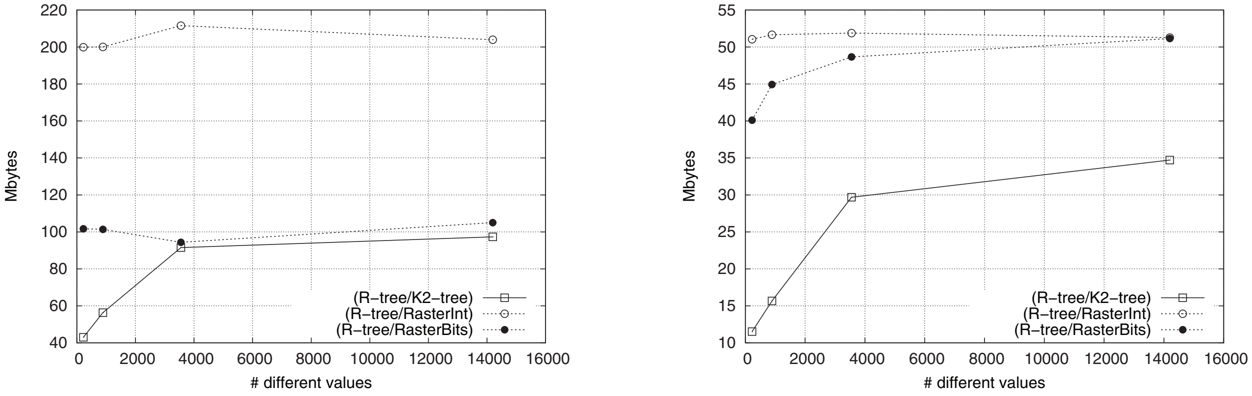
Figure 11 shows the results of the experiment using the rasters of the scenario 2. As explained, the average size of the k^2-trees increases with the number



(a) *vecca* dataset

(b) *vects* dataset

FIGURE 7. Memory consumption with rasters of scenario 1.



(a) *vecca* dataset

(b) *vects* dataset

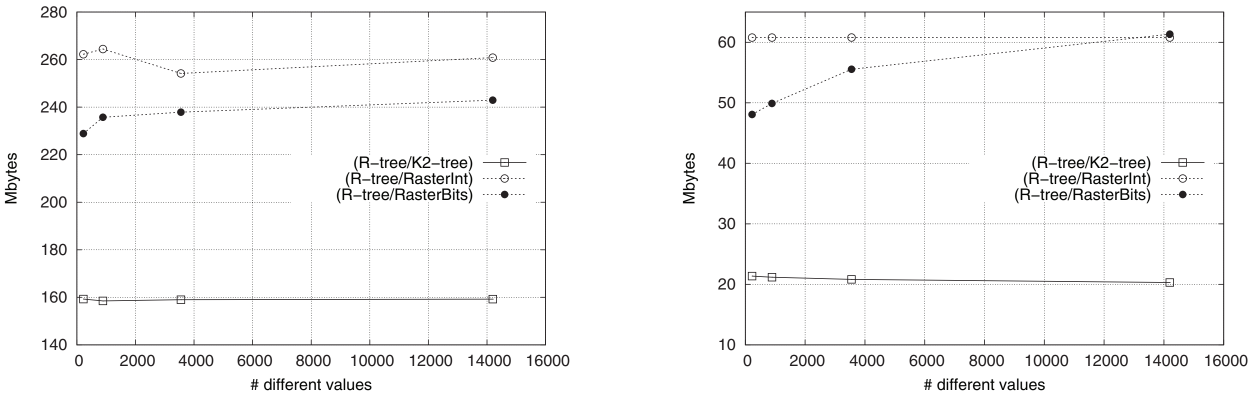
FIGURE 8. Memory consumption with rasters of scenario 2.

of different values. If the original raster has more different values, then the quadtree strategy is much less successful, as it is more difficult to find regions having the same value, and thus this implies we obtain taller trees, which yield larger top-down traversals during the application of our algorithm. Therefore, the times are harmed by this parameter. However, this is a worst-case scenario that shows that the number of different values in the raster dataset affects the performance of our proposal as can be seen in Table 5 and Figure 11.

5.2.2. Ranges with only one boundary
Figure 12 shows the performance of our algorithms and the baselines, considering strong semantics and rasters of the scenario 1. Again, the values of the query threshold were completely random. We can see that with *vecca* dataset, our approach is between 50% and 2.2 times faster than the integer-based baseline, and between 70% and 4.2 times in the bit-based baseline. The strange shape of the curve in the *vecca* dataset is probably due to the size of the output (list of MBRs which overlap regions with the query range) which is larger in the smaller rasters. In the case of the *vects* dataset, times range from being on a par to 2.25 times faster in the case of the integer-based baseline, and from 28% to 19 times faster in the bit-based baseline.

6. CONCLUSIONS

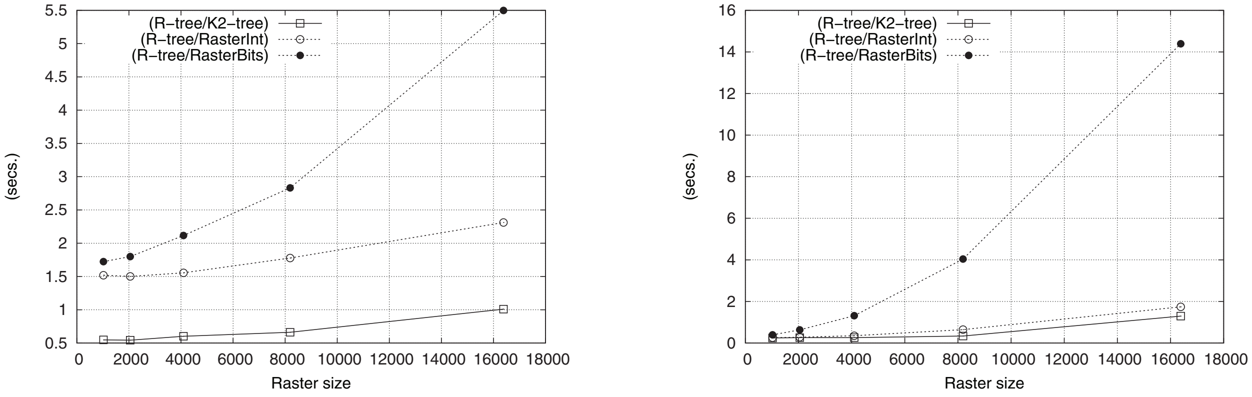
We have presented in this paper a method to perform a spatial query between a vector dataset represented using an R-tree and a raster dataset represented using k^2-trees. The method solves two problems: first, it can be used to evaluate efficiently queries between vector and raster data without having to convert one of the datasets to the other data model; and second, it



(a) *vecca* dataset

(b) *vects* dataset

FIGURE 9. Memory consumption with with rasters of scenario 2 and a query range covering 99% of the possible values.



(a) *vecca* dataset

(b) *vects* dataset

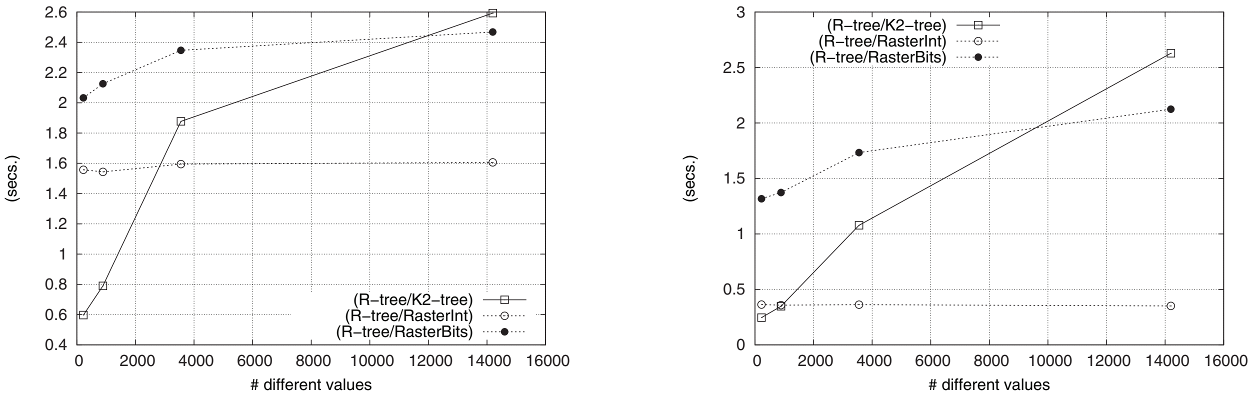
FIGURE 10. Processing time with strong semantics with rasters of scenario 1.

is space-efficient and can be used for memory-limited scenarios like applications with huge datasets.

Few conceptual data models support both the object-based and the field-based views of space. Even international standards for geographic information separate both views [60, 61]. The same happens at the logical level with vector and raster spatial data models. International standards [62, 63] separate both views and do not provide languages, data structures and algorithms to perform queries that use information from both approaches. GIS tools, with few exceptions, force users to convert data form one data model to the other to operate over them.

However, with the current high availability of data, it has become more common to have application scenarios that require queries over vector and raster data. We have mentioned two in this paper, disaster management and weather analysis, but many other scenarios arise where natural information captured by sensors (which is best represented using raster models) is used together with man-made structures (which is best represented using vector models). Our proposal helps to bridge this gap by defining an algorithm that solves efficiently this type of queries using two spatial access structures: an R-tree and a k^2-tree and without having to convert one of the datasets to the other data model.

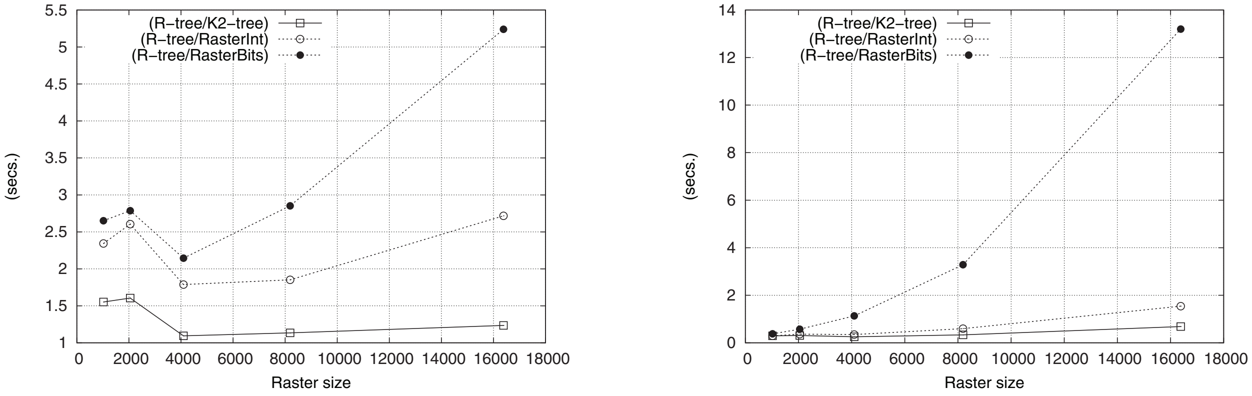
We have shown that our technique obtains important main memory savings compared with two baselines, which store the raster in plain form in main memory. Even more, if the number of different values in the raster is small, our method is even faster than those powerful baselines, a typical phenomenon when using compact data structures. The main drawback of our method is the performance when the dataset has a large number of different values, which probably implies a big disk space consumption and a degradation in the running



(a) *vecca* dataset

(b) *vects* dataset

FIGURE 11. Processing time of range queries with rasters of scenario 2.



(a) *vecca* dataset

(b) *vects* dataset

FIGURE 12. Processing time of queries with strong semantics and only one boundary.

times.

There are also compact structures for vector data (e.g., [39]). We plan to explore algorithms with the vector data represented using compact structures and with both datasets represented using compact structures. We also plan to explore the implementation of other spatial query algorithms using compact data structures. Finally, we plan to explore other compact data structures to improve the disk space consumption and the query performance when processing datasets with a large number of different values.

ACKNOWLEDGEMENTS

The work of the third author was partially supported by the research projects DIUBB [140515 3/R; 142719 3/R]s and MECESUP UBB0704. For the rest of authors, this work was supported by Ministerio de Economía y Competitividad (PGE and FEDER) under grants [TIN2016-78011-C4-1-R, TIN2016-77158-C4-3-R, TIN2015-69951-R, TIN2013-46238-C4-3-R, TIN2013-46801-C4-3-R]; Centro para el desarrollo Tecnológico e Industrial under grants [IDI-20141259, ITC-20151305, ITC-20151247] and European Unions Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No 690941 (Project BIRDS).

REFERENCES

[1] Worboys, M. F. and Duckham, M. (2004) *GIS: a computing perspective.* CRC press, Boca Raton, FL.

[2] Longley, P. A., Goodchild, M. F., Maguire, D. J., and Rhind, D. W. (2005) *Geographic Information Systems and Science.* Wiley, Chichester, UK.

[3] Samet, H. (1990) *The Design and Analysis of Spatial Data Structures.* Addison-Wesley, Reading, MA, USA.

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

[5] Cattell, R. (2011) Scalable sql and nosql data stores. *SIGMOD Record*, **39**, 12–27.

[6] Han, J., Haihong, E., Le, G., and Du, J. (2011) Survey on nosql database. *Proceedings of ICPCA 2011*, Port Elizabeth, South Africa, 26-28 October, pp. 363–366. IEEE, Los Alamitos, CA.

[7] Abadi, D. J., Boncz, P. A., and Harizopoulos, S. (2009) Column-oriented database systems. *Proceedings of the VLDB Endowment*, **2**, 1664–1665.

[8] Alsubaiee, S. and et al. (2014) Asterixdb: A scalable, open source bdms. *Proceedings of the VLDB Endowment*, **7**, 1905–1916.

[9] MonetDB team. Monetdb sql/gis module. https://monetdb.org/Documentation/Extensions/GIS. 24 nov 2016.

[10] Thompson, M. (2011) *Getting Started with GEO, CouchDB, and Node. js.* O 'Reilly Media, Sebastopol, CA.

[11] Baas, B. (2012) NoSQL spatial: Neo4j versus PostGIS. PhD thesis Delft University of Technology Delft, Netherlands.

[12] Wikipedia. Spatial database. https://en.wikipedia.org/wiki/Spatial_database. 24 nov 2016.

[13] ISO 19125:2004 (2004) *Geographic information – Simple feature access.* International Organization for Standardization (TC 211). Geneva, Switzerland.

[14] Jacobson, G. (1989) Succinct Static Data Structures. PhD thesis Carnegie-Mellon University Pittsburgh, PA. Tech Rep CMU-CS-89-112.

[15] Navarro, G. (2016) *Compact Data Structures – A practical approach.* Cambridge University Press, New York, NY.

[16] Guttman, A. (1984) R-trees: A dynamic index structure for spatial searching. *Proceedings of SIGMOD 84*, Boston, MA, 18-21 June, pp. 47–57. ACM, New York, NY.

[17] Manolopoulos, Y., Nanopoulos, A., Papadopoulos, A. N., and Theodoridis, Y. (2005) *R-trees: Theory and Applications (Advanced Information and Knowledge Processing).* Springer-Verlag New York, Secaucus, NJ.

[18] Brisaboa, N. R., Ladra, S., and Navarro, G. (2014) Compact representation of web graphs with extended functionality. *Information Systems*, **39**, 152–174.

[19] Tobler, W. B. (1970) A computer model simulation of urban growth in the detroit region. *Economic Geography*, **46**, 234–240.

[20] Couclelis, H. (1992) People manipulate objects (but cultivate fields): Beyond the raster-vector debate in GIS. *Proceedings of GIS: from space to territory - theories and methods of spatio-temporal reasoning*, Pisa, Italy, 21-23 September, pp. 65–77. Springer-Verlag Berlin, Heidelberg, Germany.

[21] Svensson, P. and Zhexue, H. (1991) Geo–SAL: A query language for spatial data analysis. *Proceedings of SSD 91*, Zurich, Switzerland, 28-30 August, pp. 119–142. Springer-Verlag Berlin, Heidelberg, Germany.

[22] Baumann, P., Dehmel, A., Furtado, P., Ritsch, R., and Widmann, N. (1998) The multidimensional database system rasdaman. *Proceedings of SIGMOD 98*, Seattle, WA, 2-4 June, pp. 575–577. ACM press, New York, NY.

[23] Vaisman, A. and Zimányi, E. (2009) A multidimensional model representing continuous fields in spatial data warehouses. *Proceedings of SIGSPATIAL GIS 2009*, Seattle, WA, 4-6 November, pp. 168–177. ACM press, New York, NY.

[24] Grumbach, S., Rigaux, P., and Segoufin, L. (2000) Manipulating interpolated data is easier than you thought. *Proceedings of VLDB 2000*, Cairo, Egypt, 10-14 September, pp. 156–165. Morgan Kaufmann Publishers, San Fransisco, CA.

[25] Brown, P. G. (2010) Overview of scidb: large scale array storage, processing and analysis. *Proceedings of SIGMOD 2010*, Indianapolis, IN, 6-11 June, pp. 963–968. ACM press, New York, NY.

[26] Tomlin, C. and Berry, J. (1979) mathematical structure for cartographic modeling in environmental analysis. *Proceedings of the American Congress on Surveying and Mapping annual meeting*, Washington, DC, 18-24 March, pp. 269–283. American Congress on Surveying and Mapping, Falls Church, VA.

[27] Tomlin, D. C. (1990) *Geographic Information Systems and Cartographic Modeling.* Prentice-Hall, Englewood Cliffs, NJ.

[28] Leduc, T., Bocher, E., Cortés, F. G., and Moreau, G. (2009) Gdms-r: A mixed sql to manage raster and vector data. *Proceedings of GIS Ostrava 2009*, Ostrava, Czech Republic, 25-28 January. Tanger Ltd.

[29] Zonal statistics help—arcgis for desktop. http://webhelp.esri.com/arcgisdesktop/9.3/index.cfm?TopicName=Zonal_Statistics. 1 dec 2016.

[30] Grass gis manual:v.rast.stats. https://grass.osgeo.org/grass72/manuals/v.rast.stats.html. 6 dec 2016.

[31] Corral, A., Vassilakopoulos, M., and Manolopoulos, Y. (1999) Algorithms for joining r-trees and linear region quadtrees. *Proceedings of SSD 99*, Hong Kong, China, 20-23 July, pp. 251–269. Springer-Verlag Berlin, Heidelberg, Germany.

[32] Corral, A., Torres, M., Vassilakopoulos, M., and Manolopoulos, Y. (2008) Predictive join processing between regions and moving objects. *Proceedings of ADBIS 2008*, Pori, Finland, 5-7 September, pp. 46–61. Springer-Verlag Berlin, Heidelberg, Germany.

[33] Gargantini, I. (1982) An effective way to represent quadtrees. *Communications of the ACM*, **25**, 905–910.

[34] Chazelle, B. (1988) A functional approach to data structures and its use in multidimensional searching. *SIAM Journal on Computing*, **17**, 427–462.

[35] Brisaboa, N. R., Luaces, M. R., Navarro, G., and Seco, D. (2009) A new point access method based on wavelet trees. *Proceedings of ER 2009 Workshops CoMoL, ETheCoM, FP-UML, MOST-ONISW, QoIS, RIGiM, SeCoGIS*, Gramado, Brazil, 9-12 November, pp. 297–306. Springer-Verlag Berlin, Heidelberg, Germany.

[36] Bentley, J. L. (1975) Multidimensional binary search trees used for associative searching. *Communications of the ACM*, **18**, 509–517.

[37] Navarro, G. and Russo, L. M. S. (2011) Space-efficient data-analysis queries on grids. *Proceedings of ISAAC 2011*, Yokohama, Japan, 5-8 December, pp. 323–332. Springer-Verlag Berlin, Heidelberg, Germany.

[38] Grossi, R., Gupta, A., and Vitter, J. S. (2003) High-order entropy-compressed text indexes. *Proceedings of SODA 2003*, Baltimore, Maryland, 12-14 January, pp. 841–850. Society for Industrial and Applied Mathematics, Philadelphia, PA.

[39] Brisaboa, N. R., Luaces, M. R., Navarro, G., and Seco, D. (2013) Space-efficient representations of rectangle datasets supporting orthogonal range querying. *Information Systems*, **38**, 635–655.

[40] DeWitt, D. J., Katz, R. H., Olken, F., Shapiro, L. D., Stonebraker, M. R., and Wood, D. A. (1984) Implementation techniques for main memory database systems. *Proceedings of SIGMOD 84*, Boston, MA, 18-21 June, pp. 1–8. ACM, New York, NY.

[41] Plattner, H. (2013) *A Course in In-Memory Data Management: The Inner Mechanics of In-Memory Databases*. Springer-Verlag Berlin, Heidelberg, Germany.

[42] Plattner, H. and Zeier, A. (2012) *In-Memory Data Management: Technology and Applications*. Springer-Verlag Berlin, Heidelberg, Germany.

[43] Munro, J. I. (1996) Tables. *Proceedings of FSTTCS 96*, Hyderabad, India, 18-20 December, pp. 37–42. Springer-Verlag Berlin, Heidelberg, Germany.

[44] Brisaboa, N. R., de Bernardo, G., and Navarro, G. (2012) Compressed dynamic binary relations. *Proceedings of DCC 2012*, Snowbird, UT, 10-12 April, pp. 52–61. IEEE Computer Society, Los Alamitos, CA.

[45] de Bernardo, G., Álvarez-García, S., Brisaboa, N. R., Navarro, G., and Pedreira, O. (2013) Compact querieable representations of raster data. *Proceedings of SPIRE 2013*, Jerusalem, Israel, 7-9 October, pp. 96–108. Springer-Verlag Berlin, Heidelberg, Germany.

[46] Klinger, A. (1971) Patterns and search statistics. *Proceedings of a Symposium Held at The Center for Tomorrow The Ohio State University*, Columbus, OH, 14-16 June, pp. 303 – 337. Academic Press, New York, NY.

[47] Klinger, A. and Dyer, C. R. (1976) Experiments on picture representation using regular decomposition. *Computer Graphics Image Processing*, **5**, 68–105.

[48] Brisaboa, N. R., de Bernardo, G., Gutiérrez, G., Ladra, S., Penabad, M. R., and Troncoso, B. A. (2015) Efficient set operations over k2-trees. *Proceedings of DCC 2015*, Snowbird, UT, 7-9 April, pp. 373–382. IEEE Computer Society, Los Alamitos, CA.

[49] Sinha, R. R., Winslett, M., and Wu, K. (2009) Finding regions of interest in large scientific datasets. *Proceedings SSDBM 2009*, New Orleans, LA, 2-4 June, pp. 130–147. Springer-Verlag Berlin, Heidelberg, Germany.

[50] Zhang, J. and You, S. (2010) Supporting web-based visual exploration of large-scale raster geospatial data using binned min-max quadtree. *Proceedings of SSDBM 2010*, Heidelberg, Germany, 2 July- 30 June, pp. 379–396. Springer-Verlag Berlin, Heidelberg, Germany.

[51] Koudas, N., Faloutsos, C., and Kamel, I. (1996) Declustering spatial databases on a multi-computer architecture. *Proceedings of EDBT '96*, Avignon, France, 2529 March, pp. 592–614. Springer-Verlag Berlin, Heidelberg, Germany.

[52] Schnitzer, B. and Leutenegger, S. T. (1999) Master-client r-trees: A new parallel r-tree architecture. *Proceedings of SSDBM '99*, Cleveland, OH, 28-30 July, pp. 68–77. IEEE, Los Alamitos, CA.

[53] Mutenda, L. and Kitsuregawa, M. (1999) Parallel r-tree spatial join for a shared-nothing architecture. *Proceedings of DANTE '99*, Kyoto, Japan, 28 - 30 November, pp. 423–. IEEE, Los Alamitos, CA.

[54] Álvarez-Garca, S., Brisaboa, N. R., Gómez-Pantoja, C., and Marín, M. (2013) Distributed query processing on compressed graphs using k2-trees. *Proceedings of SPIRE 2013*, Jerusalem, Israel, 7-9 October, pp. 298–310. Springer-Verlag Berlin, Heidelberg, Germany.

[55] Brinkhoff, T., Kriegel, H.-P., and Seeger, B. (1993) Efficient processing of spatial joins using r-trees. *SIGMOD Record*, **22**, 237–246.

[56] Theodoridis, Y., Stefanakis, E., and Sellis, T. K. (2000) Efficient cost models for spatial queries using r-trees. *IEEE Transactions on Knowledge and Data Engineering*, **12**, 19–32.

[57] Corral, A., Manolopoulos, Y., Theodoridis, Y., and Vassilakopoulos, M. (2006) Cost models for distance joins queries using r-trees. *Data and Knowledge Engineering*, **57**, 1–36.

[58] Huffman, D. A. (1952) A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, **40**, 1098–1101.

[59] Liebchen, T. and Reznik, Y. A. (2004) MPEG-4 ALS: an emerging standard for lossless audio coding. *Proceedings of DCC 2004*, Snowbird, UT, 23-25 March, pp. 439–448. IEEE Computer Society, Los Alamitos, CA.

[60] ISO 19107:2003 (2003) *Geographic information – Spatial schema*. International Organization for Standardization (TC 211). Geneva, Switzerland.

[61] ISO 19123:2005 (2005) *Geographic information – Schema for coverage geometry and functions*. International Organization for Standardization (TC 211). Geneva, Switzerland.

[62] OGC 09-025r2 (2010) *OpenGIS Web Feature Service 2.0 Interface Standard*. Open Geospatial Consortium, Inc. Wayland, MA.

[63] OGC 09-110r4 (2012) *OpenGIS Web Coverage Service 2.0 Interface Standard - Core: Corrigendum*. Open Geospatial Consortium, Inc. Wayland, MA.