Pergamon

Information Systems Vol. 22, No. 8, pp. 465-481, 1997

Printed in Great Britain
PII: S0306-4379(97)00029-X
0306-4379/97 $17.00 + 0.00

MOF-TREE: A SPATIAL ACCESS METHOD TO MANIPULATE MULTIPLE OVERLAPPING FEATURES †

YANNIS MANOLOPOULOS[1], ENRICO NARDELLI[2,3], APOSTOLOS PAPADOPOULOS[1] and GUIDO PROIETTI[2]

[1]Department of Informatics, Aristotle University, Thessaloniki 54006, Greece
[2]IASI, National Research Council, 00185 Roma, Italy
[3]Department of Pure & Applied Mathematics, University of L'Aquila, 67100 L'Aquila, Italy

(Received 14 May 1996; in final revised form 2 December 1997)

Abstract — In this paper we investigate the manipulation of large sets of 2-dimensional data representing multiple overlapping features (e.g. semantically distinct overlays of a given region), and we present a new access method, the MOF-tree. We perform an analysis with respect to the storage requirements and a time analysis with respect to window query operations involving multiple features (e.g. to verify if a constraint defined on multiple overlays holds or not inside a certain region). We examine both the pointer-based as well as the pointerless MOF-tree representations, using as space complexity measure the number of bits used in main memory and the number of disk pages in secondary storage respectively. In particular, we show that the new structure is space competitive in the average case, both in the pointer version and in the linear version, with respect to multiple instances of a region quadtree and a linear quadtree respectively, where each instance represents a single feature. Concerning the time performance of the new structure, we analyze the class of window (range) queries, posed on the secondary memory implementation. We show that the I/O worst-case time complexity for processing a number of window queries in the given image space, is competitive with respect to multiple instances of a linear quadtree, as confirmed by experimental results. Finally, we show that the MOF-tree can efficiently support spatial join processing in a spatial DBMS. © 1997 Elsevier Science Ltd. All rights reserved

Key words: Spatial Databases, Spatial Access Methods, Data/File Structures, Algorithms, Performance Evaluation

1. INTRODUCTION

The efficient handling of large sets of 2-dimensional data is a key issue in many fields (e.g. image processing, computer aided design, scientific visualization, geographical information systems etc.). To obtain efficiency it is necessary to reach the best compromise between storage overhead and cost-effective information retrieval. Many different approaches can be used to represent spatial data, such as: array representation (raster-based), run-length codes, polygons (vector-based), bounding boxes, mapping to higher or lower dimensional spaces and so on. Many access methods have been proposed for specific classes of spatial data (i.e. point, lines, and rectangles); among others we note the family of quadtrees, the family of R-trees, the cell tree, and the grid file. The interested reader can refer to [7, 9, 19] for a survey.

In many cases, multiple pieces of information can be attached to each atomic element (pixel) of a 2-dimensional dataset (termed image in the sequel). For example, in maps used for land planning activities, each point is classified under different views, some related to land constraints (i.e. urban, archaeological, environmental etc.), some to physical aspects (i.e. geological, hydro-geological, climatic, etc.). Another typical example can be found in road maps, where roads, rivers, bridges, railways and other features overlap each other in the image space. Topographic maps also contain multiple data that overlap in space, as cadastral information, drainage and electric systems and so on. Finally, a well established application field where 2-dimensional points have multiple pieces of information attached is remote sensing. Here maps are produced by satellites, which for each point measure values of emission for many different electromagnetic bands. In all these cases, we say that multiple overlapping features are present in the image. We therefore consider the following class of 2-dimensional data: an item of the class is a set of 2-dimensional features

†Recommended by Stavros Christodoulakis

which are allowed to overlap. Each feature, that is an area homogeneous with respect to a certain value, is represented by a 2-dimensional region (not necessarily connected).

From the storage point of view, we can roughly classify access methods either as suitable to main memory or to secondary storage. Main memory data structures to represent multiple overlapping features have been designed: the pyramid [21] with its variants [3, 14], and the DF-expression [10] are the most widely known. All of them belong to the *quadtrees* class [19], since they are hierarchical and developed on the basis of a regular space decomposition. Nevertheless, spatial data have usually a large volume and cannot fit into core memory. Hence, a disk-based representation would be more appropriate. A quadtree based secondary memory access methods for representing multiple features is the HL-quadtree [16], but it cannot manipulate overlapping features. Therefore, the traditional way to store images containing overlapping features in a quadtree-based fashion is based on a collection of linear quadtrees [8], one for each feature. However, such an approach is quite space consuming (since multiple indexes must be maintained) and does not allow to manipulate efficiently operations involving several features simultaneously.

In this work we define and analyze a quadtree-based spatial structure able to efficiently support images with multiple overlapping features. We name this structure *Multiple Overlapping Features Quadtree* (in short, MOF-tree), and we consider both a pointer-based and a pointerless (i.e. linear) implementation. We show that several advantages derive from storing multiple features in a single structure, from the standpoint of saving space and execution time in a large class of operations involving the multiple features as a whole (e.g., to verify if a constraint defined on multiple overlays holds or not inside a certain region).

Concerning the space requirement analysis, the significance of our approach is that the average case analysis carried out (instead of the traditional worst-case, which is of less interest for some applications) is founded on analytic probabilistic models used to describe spatial data. Using as space complexity the required number of bits used to store the compared structures, we first prove that the pointer-based MOF-tree outperforms the traditional approach based on multiple pointer-based region quadtrees. Then, using as space complexity the number of disk pages, we show that the linear MOF-tree version is space competitive with respect to a representation based on multiple linear quadtrees.

On the other hand, for what concerns time requirement analysis, we analyze window queries, which comprise the most important class of queries on 2-dimensional data. Window queries are very important with respect to 2-dimensional data, since they allow the extraction of the interesting image parts. Since we assume to work on large amounts of data, we focus on the worst-case performance of the secondary memory MOF-tree implementation. Time complexity is measured in terms of page accesses, since main memory processing time is negligible compared with latency and seek times of disks. We show that the same performance as for the case of multiple non-overlapping features [16] can be obtained for the *exist* and the *select* query. However, for the *report* query we obtain an improvement which is linear in the number of represented features, with respect to the traditional representation based on multiple linear quadtrees. Such theoretical results are confirmed by experiments on real datasets.

The rest of the paper is organized as follows. In Section 2 we describe both MOF-tree representations, and we give implementation details explaining the differences of MOF-tree in comparison to the HL-quadtree. In Section 3, the space complexity of the two implementations is analyzed and a comparison with the standard approach based on multiple single-feature quadtrees is carried out. In Section 4 we refer to the algorithms performing window operations and we illustrate representative experimental results. In Section 5 we report some considerations for future work on the MOF-tree structure in order to formalize other important user query operations and establish algorithmic alternatives to improve space and time performance. The last section contains concluding remarks and motivates for further research.

2. THE MOF-TREE STRUCTURE

2.1. Description of the MOF-Tree

Assume that an image of size $T \times T$ containing k overlapping features is given, where $T = 2^m$ and m is an integer. Similarly to the region quadtree [19] and the HL-quadtree [16], the MOF-tree is based on a recursive image decomposition into four quadrants of equal size. Since features are overlapped, the decomposition stops only when a quadrant is fully covered by all the features contained in it. In such a case, we say that the quadrant is *homogeneous*. For the sake of uniformity, the image background will be considered as a feature.

The decomposition can be represented as a tree of outdegree 4, where the root (at level m) corresponds to the whole image and each node (at level m–d) to a quadrant of side length $T/2^d$. Internal nodes are associated to non-homogeneous quadrants, while homogeneous quadrants correspond to leaves. The structure generated using this decomposition scheme is the *Multiple Overlapping Features Quadtree* (MOF-tree).

In Figure 1, an example of a MOF-tree in a $2^2 \times 2^2$ image space representing two overlapping features on a white background is given. Note that internal nodes can be fully covered by some feature and partially covered by others (see for example the SE son of the root, which is fully covered by the vertical feature and partially covered by the horizontal feature). Features partially covering a quadrant are depicted inside a circle, while features totally covering a quadrant are depicted inside a square.

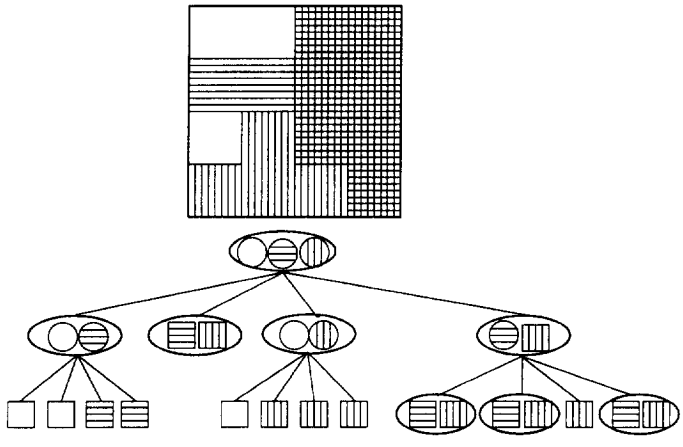

Fig. 1: Two Overlapping Features on a White Background and the Corresponding MOF Representation

2.2. MOF-Tree Implementations

The MOF-tree can be represented as a tree (pointer-based version) or as a list (pointerless version). In the former, random access is privileged, but a substantial amount of space overhead is introduced. In the latter case, the storage medium can be a disk, absolutely needed for large amounts of 2-dimensional data required by modern applications. In this subsection we introduce two possible implementations of the MOF structure.

The Pointer-Based MOF-Tree. In the following, we will use indifferently the term MOF-tree to refer to the data structure and to its pointer-based representation. For a large class of operations to be executed on the MOF-tree, we need to know when accessing a node, if each contained feature fully covers or not the considered quadrant. To this aim, to internal nodes we associate a record of type *nonleaf* containing, apart from the four pointers to the sons and the one to the father, all the information concerning the contained features. This information is stored in a bit-vector *FEATURES* of size k, in which the i-th bit is set to 1 if and only if the i-th feature is contained in the associated quadrant, and in a bit-vector *COVER* of size

k, in which the i-th bit is set to 1 if and only if the i-th feature fully covers the associated quadrant. Concerning leaf nodes, since they represent quadrants fully covered by the features contained, we associate to them a record of type *leaf* containing a pointer to the father and a bit-vector *FEATURES* of size k.

The Linear MOF-Tree. A linear MOF-tree version can be obtained by coding all the nodes by means of a base 5 locational key of length m (the image resolution) [8]. This allows the use of an indexing scheme as the B^+-tree [1] to efficiently support random access to every MOF-tree node. As in the case of the pointer-based MOF-tree, we distinguish among records associated to internal and to leaf nodes. More precisely, an internal node is represented by means of a record containing a k-bit vector *FEATURES* and a k-bit vector *COVER*, being used in the same way as in the pointer-based MOF-tree. Concerning records associated to leaf nodes, they lack of the *COVER* vector in comparison to an internal record. To distinguish between the two kinds of records, we also associate to each of them a *LEAF* bit whose value is 1 if and only if the corresponding node is a leaf.

Figure 2 shows the structure of the records both for pointer (above) and pointerless (below) representation.

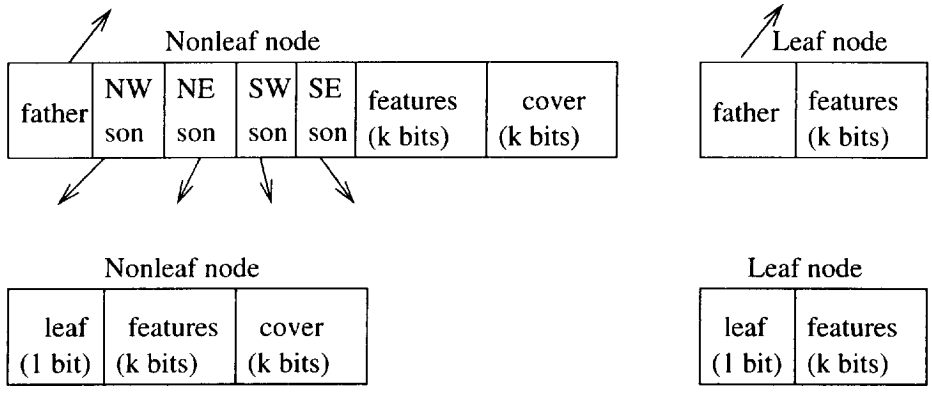

Fig. 2: Pointer (Top) and Pointerless (Bottom) Records' Structure

The most important structural difference of MOF-tree in comparison to the HL-quadtree is that, in order to manipulate overlapping features, we introduce the *COVER* vector. The *COVER* vector is not needed in HL-quadtrees since an internal node is always partially covered by the features contained in it. Also, note that in the case of non-overlapping features, since a leaf node represents a quadrant fully covered by only one features, it is enough to store the identifier of the feature itself, and then $\lceil \log k \rceil$ bits suffice to fully describe a leaf record.

The obtained list of records can be sorted according to ascending values of the l-keys. For example, for the MOF-tree in Figure 1, we have the following sequence:

(0,00,111,000), (0,10,110,000), *(1,11,100)*, *(1,12,100)*, *(1,13,010)*, *(1,14,010)*, *(1,20,011)*, (0,30,101,000), *(1,31,100)*, *(1,32,001)*, *(1,33,001)*, *(1,34,001)*, (0,40,011,001), *(1,41,011)*, *(1,42,011)*, *(1,43,001)*, *(1,44,011)*.

Note that in the previous list, leaf records (being in italics in order to improve the readability) have a structure (*LEAF*, l-key, *FEATURES*), whereas the structure of internal records is (*LEAF*, l-key, *FEATURES, COVER*).

Symbol	Definition
k	number of features
T	side of the image space
m	image resolution ($T = 2^m$)
n	side of the query window
M	number of maximal blocks inside the query window
p	probability for a pixel to be black
r	disk page capacity in records
P	disk page size in bits
N_M	number of nodes in the pointer MOF-tree
N_m	number of nodes in a class-m pointer Quadtree
B_m	number of black leaves in a class-m pointer Quadtree
L_M	number of leaf pages in the Linear MOF-tree
L_m	number of leaf pages in a class-m Linear Quadtree
I_M	number of index pages in the Linear MOF-tree
I_m	number of index pages in a class-m Linear Quadtree
h_M	height of the index part of the Linear MOF-tree
h_m	height of the index part of a class-m Linear Quadtree
S_M	space occupancy of the pointer MOF-tree
S_m	space occupancy of a class-m pointer Quadtree
SL_M	space occupancy of the Linear MOF-tree
SL_m	space occupancy of a class-m Linear Quadtree

Table 1: List of symbols and definitions.

3. SPACE ANALYSIS

In this section we first prove that a single MOF-tree representing k overlapping features is space competitive with respect to a representation based on k single-feature quadtrees (MQ in the following); then, a comparison between the linear MOF-tree version against a multiple linear quadtrees representation (MLQ in the following) is done. Space complexity will be examined in terms of the number of bits and the number of disk pages used to represent the structures respectively. We analyze space performance with respect to the most significant probabilistic models of binary images. The following table contains the symbols and their corresponding definitions used throughout the analysis.

Model for Random Images: According to this model each image pixel is assumed to be statistically independent to any other pixel. If p is the probability for a generic pixel to contain the feature, then a level-i node is fully covered by the feature with probability $b_i = p^{4^i}$, fully uncovered with probability $w_i = (1-p)^{4^i}$, and partially covered (and thus internal to the quadtree) with probability $g_i = 1 - p^{4^i} - (1-p)^{4^i}$. Therefore, by varying the p value we reach a range of statistical distributions. A large class of images (for example pebble maps or point data maps) are truly described from such a kind of distribution.

Model for Random Trees: This model has been proposed by Puech and Yahia [12, 17]. Let Q_m be the set of all class-m quadtrees, i.e. quadtrees of height less than or equal to m. Let b_i be a non increasing sequence of $m+1$ reals, where $0 < b_i < 1/2$ and $b_0 = 1/2$. A random tree of Q_m is built by using a branching process, such that the quantity $2b_i$ is the probability for a level i node to be external and then $g_i = 1 - 2b_i$ is the probability for a level i node to be internal. Therefore, by definition $2b_0 = 1$, i.e. at level 0 (pixel level) all the nodes are external with probability 1 as expected. Once a node is known to be external, we do not make assumption on its color, i.e. the node can be indifferently white or black, because if we do, we may lead to a non-condensed quadtree (a quadtree where the four children of an

internal node can be all white or all black). This model has been further refined in [11, 23] by defining a more complicated branching process that produces condensed (ordinary) region quadtrees, only.

Under a different perspective, Samet introduced a similar model [18], where an external node containing a fixed pixel has equal probability to be of any size, or, in other words, a leaf node is equally likely to appear in any position and level in the quadtree. A large class of geographical images, such as floodplain, topographical and landuse maps, seems to be modeled by this definition of random trees, and then we will use this model to perform theoretical comparisons. From an analytical point of view, this model is an instance of the models of references [15, 23] obtainable by setting the probability that a black node exists at any level i position equal to $\frac{1}{2(i+1)}$.

3.1. MOF-Tree vs. MQ

Let S_M and S_m denote the space occupancy of the MOF-tree and a class-m quadtree, respectively. Also, let N_M be the number of nodes in the MOF-tree. As in the case of a pointer quadtree, we distinguish the nodes of MOF-tree in two types of nodes: internal and leaf. In the former, we observe that each pointer has to address among N_M nodes and then needs $\lceil \log N_M \rceil + 1$ bits (the extra bit distinguishes an internal from a leaf node). Regarding the *FEATURE* and *COVER* node fields, each one needs exactly k bits. Therefore, each internal MOF-tree node needs $5(\lceil \log N_M \rceil + 1) + 2k$ bits. Regarding the leaf nodes, they only require the pointer to the father and a binary string to store the contained features. Thus, the required storage is limited to $(\lceil \log N_M \rceil + 1) + k$. Being approximately equal to 3 the ratio between leaf and internal nodes, it follows that the space complexity (in bits) for the pointer MOF-tree is:

$$S_M \approx \frac{N_M}{4}(5(\lceil \log N_M \rceil + 1) + 2k) + \frac{3N_M}{4}(\lceil \log N_M \rceil + 1 + k)$$

Since the number of nodes of a complete quadtree is $N_M = \frac{4^{m+1}-1}{3}$, it follows that $\lceil \log N_M \rceil \leq 2m + 1$ (worst case). Therefore, the previous expression becomes:

$$S_M \approx N_M(4m + \frac{5}{4}k + 4) \tag{1}$$

The total number of bits needed to store a class-m pointer quadtree with N_m nodes is:

$$S_m \approx \frac{N_m}{4}(5(\lceil \log N_m \rceil + 1)) + \frac{3N_m}{4}(\lceil \log N_n \rceil + 1) = N_m(4m + 4)$$

Then we obtain the following ratio:

$$\frac{S_M}{S_1 + S_2 + ... + S_k} \approx \frac{N_M\,(4m + \frac{5}{4}k + 4)}{(N_1 + N_2 + ... + N_k)(4m + 4)} = \frac{N_M}{N_1 + N_2 + ... + N_k} \times \frac{m + 1 + \frac{5}{16}k}{m + 1}$$

For the sake of simplicity we assume that each independent feature is described by the same probability law. Therefore, it follows that $N_1 = N_2 = ... = N_k$; setting N_m as the expected number of nodes in a class-m quadtree, the above formula becomes:

$$\frac{S_M}{S_1 + S_2 + ... + S_k} \approx \frac{N_M}{N_m} \times \frac{m + 1 + \frac{5}{16}k}{(m + 1)k} \tag{2}$$

To compare the space performance of the two approaches, we have then to estimate the number of nodes contained in the MOF-tree and in each quadtree with respect to the two proposed models.

3.1.1. Random Image Model

It has been proved that under the random image model, the average number of nodes N_m of a class-m quadtree obeys the equation [22]:

$$N_m = 1 + \sum_{i=0}^{m-1} 4^{m-i}(1 - w_{i+1} - b_{i+1})$$

where w_i and b_i denote the probability that a level-i node is white and black respectively. Assuming p as the probability for a pixel to be black and $1-p$ as the probability for a pixel to be white. It easily follows that:

$$b_i = p^{4^i} \qquad w_i = (1-p)^{4^i}$$

and then the average number of expected nodes for a class-m quadtree becomes:

$$N_m = 1 + \sum_{i=0}^{m-1} 4^{m-i} \left(1 - p^{4^{i+1}} - (1-p)^{4^{i+1}}\right) \tag{3}$$

Let us now estimate the number of MOF-tree nodes. The probability for a node to exist at level i in one of the given quadtrees is:

$$p_1 = 1 - p^{4^{i+1}} - (1-p)^{4^{i+1}}$$

and therefore the probability that the node does not exist is:

$$p_2 = 1 - p_1 = p^{4^{i+1}} + (1-p)^{4^{i+1}}$$

Given the independence among the features, it follows that the probability that a node is absent from all independent quadtrees is:

$$p_3 = p_2^k = \left(p^{4^{i+1}} + (1-p)^{4^{i+1}}\right)^k$$

and finally the probability that the node is present at level i to at least one quadtree (and then present at level i in the MOF-tree) is:

$$E_i = 1 - p_3 = 1 - \left(p^{4^{i+1}} + (1-p)^{4^{i+1}}\right)^k$$

Therefore, since at level i there are at most 4^{m-i} nodes, it follows that the expected number of nodes in the MOF-tree is:

$$N_M = 1 + \sum_{i=0}^{m-1} 4^{m-i} E_i = 1 + \sum_{i=0}^{m-1} 4^{m-i} \left(1 - \left(p^{4^{i+1}} + (1-p)^{4^{i+1}}\right)^k\right) \tag{4}$$

Finally, by combining Equations (2), (3) and (4), we obtain:

$$\frac{S_M}{S_1 + S_2 + \ldots + S_k} \approx \frac{1 + \sum_{i=0}^{m-1} 4^{m-i} \left(1 - \left(p^{4^{i+1}} + (1-p)^{4^{i+1}}\right)^k\right)}{1 + \sum_{i=0}^{m-1} 4^{m-i} \left(1 - p^{4^{i+1}} - (1-p)^{4^{i+1}}\right)} \times \frac{m + 1 + \frac{5}{16}k}{(m+1)k}$$

which, for standard values of p, m and k, produces the ratios depicted in Table 2. From this table, it immediately follows that in the random image model, the MOF-tree outperforms the MQ representation for any p, k and m.

3.1.2. Random Tree Model

It has been proved that the average number of nodes N_m of a class-m random quadtree is [23]:

$$N_m = \frac{4^{m+2} - 3m - 7}{9(m+1)} \tag{5}$$

Let us now estimate the number of MOF-tree nodes. Since b_i is equal to $\frac{1}{2(i+1)}$, the probability for a node to exist at level i of an independent quadtree is [15, 23]:

$$p_1 = \frac{i+1}{m+1}$$

	p=0.1				p=0.3				p=0.5			
	m				m				m			
k	8	10	12	14	8	10	12	14	8	10	12	14
2	0.76	0.75	0.74	0.74	0.62	0.62	0.61	0.61	0.58	0.57	0.57	0.57
4	0.52	0.51	0.50	0.50	0.35	0.34	0.33	0.33	0.31	0.31	0.30	0.30
8	0.33	0.32	0.31	0.30	0.20	0.19	0.18	0.18	0.18	0.17	0.16	0.16
16	0.21	0.19	0.18	0.18	0.12	0.11	0.11	0.10	0.11	0.10	0.09	0.09
32	0.14	0.13	0.12	0.11	0.08	0.07	0.07	0.06	0.07	0.06	0.06	0.06

Table 2: Space Ratio MOF-Tree/MQ for p=0.1, 0.3 and 0.5 (or p=0.9, 0.7 and 0.5 Respectively).

and therefore the probability of the node to be absent is:

$$p_2 = 1 - p_1 = 1 - \frac{i+1}{m+1} = \frac{m-i}{m+1}$$

Given the independence among the features, it follows that the probability that a node is absent at level i from all independent quadtrees is:

$$p_3 = p_2^k = \left(\frac{m-i}{m+1}\right)^k$$

and finally the probability that the node is present at level i to at least one quadtree (and then present at level i in the MOF-tree) is:

$$E_i = 1 - p_3 = 1 - \left(\frac{m-i}{m+1}\right)^k$$

Since we know that at level i there are at most 4^{m-i} nodes, it follows that the expected number of nodes in the MOF-tree is:

$$N_M = 1 + \sum_{i=0}^{m-1} 4^{m-i} E_i = 1 + \sum_{i=0}^{m-1} 4^{m-i} \left(1 - \left(\frac{m-i}{m+1}\right)^k\right) \quad (6)$$

Therefore, by combining Equations (2), (5) and (6), we obtain:

$$\frac{S_M}{S_1 + S_2 + \ldots + S_k} = \frac{1 + \sum_{i=0}^{m-1} 4^{m-i} \left(1 - \left(\frac{m-i}{m+1}\right)^k\right)}{\frac{4^{m+2}-3m-7}{9(m+1)}} \times \frac{m+1+\frac{5}{16}k}{(m+1)k}$$

The following table depicts a sample of results obtained by inserting some standard values of m and k in the above expression. From this table, it immediately follows that in the random tree model, the MOF-tree outperforms the MQ representation for every practical value of k and m.

3.2. Linear MOF-Tree vs. MLQ

Maintaining the same notation, we now confront space complexity of the linear MOF-tree against MLQ. This comparison is of great importance whenever a secondary memory implementation is required.

Let SL_M and SL_m denote the space occupancy of respectively the linear MOF-tree and the MLQ, measured in disk pages. A B^+-tree comprises of two parts: a) the leaf part where all useful information is stored and b) the index part, used to guide the search in the tree. Therefore, in order to compare the linear MOF-tree and the MLQ we must estimate both numbers of pages in the leaf and index parts.

	m			
k	8	10	12	14
2	0.97	0.97	0.98	0.98
4	0.87	0.89	0.90	0.92
8	0.73	0.77	0.80	0.82
16	0.58	0.62	0.66	0.68
32	0.44	0.47	0.51	0.54

Table 3: Space ratio MOF-tree/MQ in the random tree model.

Let us first examine the case of the linear MOF-tree. In the case of the MOF-tree, all nodes of the pointer version must be represented in the linear version. Each internal node needs $3m$ bits to store the m digits (in base 5) of the l-key, $2k$ bits to store the *FEATURES* and the *COVER* vectors and one bit to store the *LEAF* bit, while each external node needs $3m$ bits to store the l-key, k bits to store the *FEATURES* vector and one bit to store the *LEAF* bit. This information is stored in the leaf level of the B^+-tree. Let L_M denote the number of leaf pages in the linear MOF-tree. From Equation (1) it follows that:

$$L_M = \left\lceil \frac{S_M}{P} \right\rceil = \left\lceil \frac{N_M(16m + 5k + 16)}{4P} \right\rceil \tag{7}$$

where P is the disk page size (in bits).

Each page in the index part, contains locational codes and pointers to the lower level. If there are r locational codes present in an index page (which is assumed to be fully utilized), there must be $r + 1$ pointers to the lower level. Assuming that each pointer takes 4 bytes (=32 bits), then r can be estimated as follows:

$$P = 32(r + 1) + 3mr \Rightarrow r = \left\lfloor \frac{P - 32}{32 + 3m} \right\rfloor$$

The height h_M of the index part is therefore:

$$h_M = \lceil log_r(L_M) \rceil \tag{8}$$

Let I_M denote the number of index pages in the linear MOF-tree. It is clear that:

$$I_M = \sum_{j=1}^{h_M} \left\lceil \frac{L_M}{r^j} \right\rceil \approx \sum_{j=1}^{h_M} \left(\frac{L_M}{r^j} + 1 \right) = h_M + L_M \frac{1 - r^{-h_M}}{r - 1}$$

Therefore, from the above expressions, we get that the total space requirements of the linear MOF-tree are:

$$SL_M = L_M + I_M \approx L_M \frac{r - r^{-h_M}}{r - 1} + h_M$$

On the other hand, in the case of the MLQ, we know that in each linear quadtree, only the black nodes are taken in consideration, and each one needs only $3m$ bits to store the l-key. Therefore, the number of leaf pages in the linear representation of a class-m quadtree equals:

$$L_m = \left\lceil \frac{3mB_m}{P} \right\rceil \tag{9}$$

where B_m is the number of black nodes in a class-m quadtree.

As in the case of the linear MOF-tree, the index pages contain locational codes and pointers to the lower levels. Therefore, the height of the index part and the number of index pages are respectively:

$$h_m = \lceil log_r(L_m) \rceil \tag{10}$$

and

$$I_m = \sum_{j=1}^{h_m} \left\lceil \frac{L_m}{r^j} \right\rceil \approx \sum_{j=1}^{h_m} \left(\frac{L_m}{r^j} + 1 \right) = L_m \frac{1 - r^{-h_m}}{r - 1} + h_m$$

The total space requirements of each class-m linear quadtree is:

$$SL_m = L_m + I_m \approx L_m \frac{r - r^{-h_m}}{r - 1} + h_m$$

Since we assume for simplicity that all the features are described by the same probability law, it follows that $SL_1 = SL_2 = ... = SL_k$. Using the above derived formulae and substituting the appropriate values we get:

$$\frac{SL_M}{SL_1 + SL_2 + ... + SL_k} \approx \frac{L_M \frac{r - r^{-h_M}}{r-1} + h_M}{k(L_m \frac{r - r^{-h_m}}{r-1} + h_m)} \tag{11}$$

In order to compare the space performance under the two linear approaches, we have then to estimate the number of black nodes contained in a class-m quadtree with respect to the two proposed models.

3.2.1. Random Image Model

It is easily derived that the expected number of black nodes in a class-m quadtree is:

$$B_m = p^{4^m} + \sum_{i=0}^{m-1} 4^{m-i} \left(p^{4^i} - p^{4^{i+1}} \right) \tag{12}$$

By substituting the parameters L_M, L_m, h_M, h_m, N_M, and B_m from Equations (7,9,8,10,4,12) respectively in Equation (11) we get the ratio of space requirements for the random image model. The following Table 4 depicts the ratio of space occupancy of the linear MOF-tree over MLQ for some values of p, m and k. From this table, it immediately follows that under the random image model the linear MOF-tree outperforms the MLQ for every m as soon as $k > 4$.

	p=0.1				p=0.3				p=0.5			
	m				m				m			
k	8	10	12	14	8	10	12	14	8	10	12	14
2	1.91	1.85	1.81	1.78	1.96	1.90	1.85	1.82	2.32	2.25	2.20	2.16
4	1.30	1.25	1.22	1.19	1.09	1.04	1.01	0.99	1.25	1.20	1.16	1.13
8	0.82	0.78	0.75	0.72	0.61	0.58	0.55	0.53	0.70	0.66	0.63	0.61
16	0.51	0.47	0.45	0.43	0.37	0.34	0.32	0.31	0.42	0.39	0.36	0.35
32	0.35	0.31	0.28	0.27	0.25	0.22	0.21	0.19	0.29	0.26	0.23	0.22

Table 4: Space Ratio Linear MOF-Tree/MLQ for p=0.1, 0.3 and 0.5 (or p=0.9, 0.7 and 0.5 Respectively).

3.2.2. Random Tree Model

Since according to this probabilistic model the expected number of black nodes is half of the expected number of the leaves (which are 3/4 of all the nodes), we have that:

$$B_m = \frac{3N_m}{8} \tag{13}$$

where N_m is given by Equation (5). By substituting the parameters L_M, L_m, h_M, h_m, N_M, and B_m from Equations (7,9,8,10,4,13) respectively in Equation (11) we get the ratio of space requirements for the random image model. The following table depicts the space occupancy ratio of the linear MOF-tree vs. the MLQ under the random tree model for some values of m and k.

	m			
k	8	10	12	14
2	3.50	3.80	3.77	3.74
4	3.14	3.47	3.50	3.50
8	2.64	3.00	3.07	3.12
16	2.08	2.41	2.53	2.61
32	1.57	1.84	1.95	2.04

Table 5: Space Ratio Linear MOF-Tree/MLQ under the Random Tree Model.

From the later table, it emerges that a small overhead in the space complexity of the linear MOF-tree with respect to MLQ is introduced. In fact, this model produces very compact quadtrees, and consequently the number of black leaves to store is very low; on the contrary, the expected number of nodes in the MOF-tree is quite high (since being the features independent, they tend to occupy the whole image space, so producing an almost complete MOF-tree), and therefore the space occupancy of the linear MOF-tree is high also. However, this is a little price to pay for a big improvement in terms of spatial queries processing, as we shall see in the next section.

4. PERFORMANCE EVALUATION

Large sets of 2-dimensional overlapping data can be queried in many possible ways. In particular, for multiple overlapping features window queries are formulated as follows:

- *exist* (f, w): Determine whether or not the feature f exists inside window w.
- *report* (w): Report the identity of all the features that exist inside window w.
- *select* (f, w): Determine the locations of all occurrences of the feature f inside window w. This means to output all the blocks fully covered by feature f.

The basic approach to process a window query is to decompose it from an $n \times n$ window into a sequence of smaller queries according to the $M{=}O(n)$ maximal blocks contained inside the given window [3]. Since the linear MOF-tree stores all the nodes of the corresponding pointer tree, to answer the window queries it is possible to apply the same algorithms as those proposed for the HL-quadtree [16]. Then, it follows that the exist and the report query can be satisfied with $O(M \log_r T)$ disk accesses, while the select query can be answered with $O(M \log_r T + n^2/r)$ disk accesses. It is worth noting that for the exist and select queries, the same theoretical upper bounds are obtained as in the case of an MLQ representation [13]. On the other hand, for the report query there is an improvement which is linear in the number of represented features, since an MLQ representation needs $O(kM \log_r T)$ disk accesses to satisfy such a query. Table 6 summarizes I/O performance of the linear MOF-tree and the MLQ when executing window queries.

Detailed experiments comparing the linear MOF-tree and the MLQ have been performed. We implemented all the structures in C programming language under UNIX and run the experiments on a SUN SPARC workstation. We executed the window queries on 1024×1024 images containing 16 overlapping features. Images have been built by overlapping meteorological satellite views of European and Asian regions. For each of the window sizes, we perform 100 queries, and sum the number of disk accesses. Some representative results are illustrated in Figures 3 to 7.

By inspecting the Figures 3 to 7, in the case of window queries operations, it is evident that the MOF-tree while it shows practically the same performance in comparison to the traditional

Query	Linear MOF-tree	MLQ
exist	$O(M \log_r T)$	$O(M \log_r T)$
report	$O(M \log_r T)$	$O(kM \log_r T)$
select	$O(M \log_r T + n^2/r)$	$O(M \log_r T + n^2/r)$

Table 6: Time Performance of the Linear MOF-Tree and the MLQ Representation for Window Queries.

MLQ approach for the exist and the select query, it demonstrates considerable improvement for the report query. More specifically, we observe that the performance gain of multiple linear quadtrees over the MOF-tree is around 10% on the average for the exist and select queries, as it is illustrated in Figures 3 to 6. On the other hand, in the case of the report query, the MOF-tree outperforms by factors the multiple linear quadtree representation, as it is demonstrated in Figure 7. Therefore, the use of the MOF structure is recommended, since queries referring to more than one features are posed very frequently in applications that manipulate spatial data.

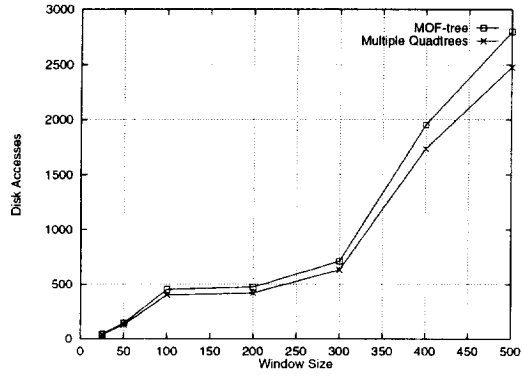

Fig. 3: Exist Query for Features Covering 71% of the Image.

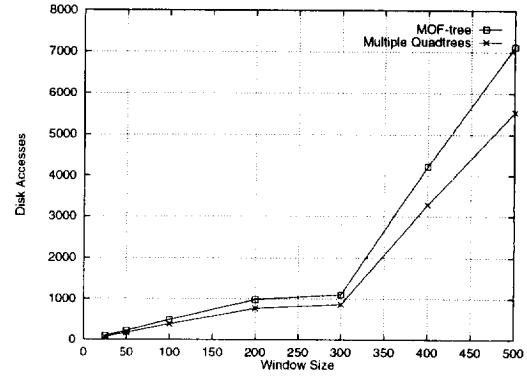

Fig. 4: Exist Query for Features Covering 43% of the Image.

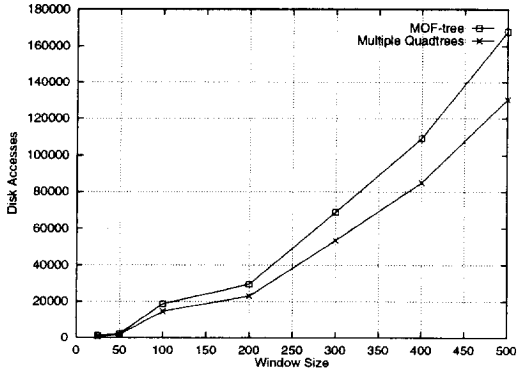

Fig. 5: Select Query for Features Covering 43% of the Image.

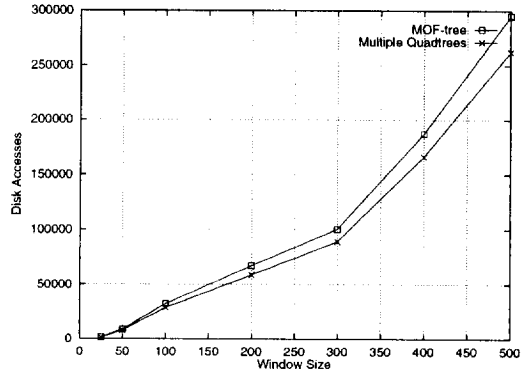

Fig. 6: Select Query for Features Covering 71% of the Image.

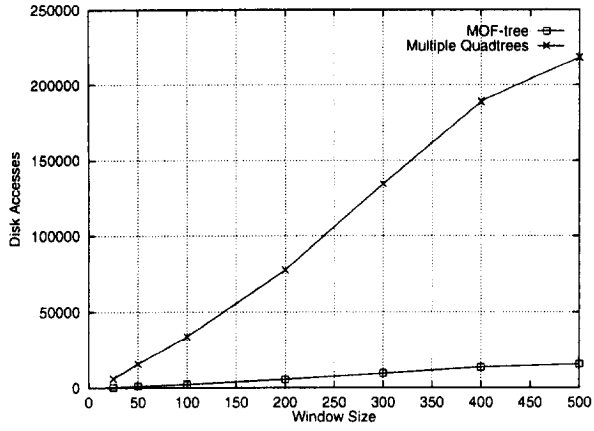

Fig. 7: Report Query.

5. ALGORITHMIC AND STRUCTURAL ALTERNATIVES

5.1. Extended Window Queries

In an MLQ representation, exist and select queries require the traversal of the quadtree representing the specified feature f. This implies that an aggregating representation as offered by the MOF-tree does not induct advantages other than space savings. However, in real applications, when dealing with k multiple overlapping features, extended versions of the window queries are often needed [2], including also set theoretical operations among the features. The exist and the select query can then be extended to a subset of h $(1 < h \leq k)$ features in the following way:

- *extended_exist*$(f_{i_1}, f_{i_2}, ..., f_{i_h}, w)$: Determine whether or not all of the features $f_{i_1}, f_{i_2}, ..., f_{i_h}$ exist inside window w;

- *extended_select*$(f_{i_1}, f_{i_2}, ..., f_{i_h}, w)$: Determine the locations of all occurrences of all the features $f_{i_1}, f_{i_2}, ..., f_{i_h}$ inside window w. This means to output all blocks homogeneous for at least one of the features $f_{i_1}, f_{i_2}, ..., f_{i_h}, w$;

Operating on the linear MOF-tree, a simple way to execute the extended_exist query is to repeat the same procedure as for the exist query, being careful to verify the existence or not of the given features inside each maximal block contained in the window (or in the respective direct ancestor) and returning an affirmative answer only when all the given features have been found. By looking at the I/O complexity of the exist query as obtained in the Section 5, it follows that such a process has a worst-case bound on the number of secondary storage accesses of $O(M \log_r T)$. Concerning the extended_select query, we have to focus exclusively on the complete nodes of the linear MOF-tree inside the given window, operating respectively on each of them an OR-ing or an AND-ing on the features bits concerned in the query. With a similar reasoning, it follows that such a process has a worst-case bound on the number of secondary storage accesses of $O(M \log_r T + n^2/r)$. Conversely, using an MLQ representation, although the essence of operations remains the same, we have to load each involved quadtree independently, which implies an I/O time overhead linear in the number of concerned features.

Operations of such a kind have a common property, that is the need to linearly scan the occurrences of all the features involved in the operation itself. And this is the reason why a physical aggregation of the data can lead to significant savings in terms of I/O execution time. In fact, the above sketched algorithms for the given queries require only one scanning of the MOF-tree, while h distinct scans in the MLQ representation are requested: the advantage is then linear in h.

These operations are not exhaustive, in the sense that they do not cover completely all application needs. In such an environment it is important to support an additional operator, known as spatial_join that is able to formally represent the whole class of operations combining in some way a subset of the given features. In such a way a large class of spatial predicates (such as overlap and containment) among features can be supported. An immediate example derives directly from various applications in the geographic field, where a widely used type of query is: "Determine all the locations where there are forests or meadows but not cities inside the query window w". Such nice zones can be located simply combining known queries as follows:

$$(\text{select(forest,}w\text{)}\ \textbf{OR}\ \text{select(meadow,}w\text{))}\ \textbf{AND}\ (\textbf{NOT}(\text{select(city,}w\text{))})$$

Another typical query could be: "Determine all the factories contained in cities inside the query window w". To locate such dirty zones, we simply combine known queries as follows:

$$\text{select(factory,}w\text{)}\ \textbf{AND}\ \text{select(city,}w\text{)}$$

If we indicate with $P(f_{i_1}, f_{i_2}, ..., f_{i_h})$ a logical predicate on a subset of h $(1 < h \leq k)$ features, then we can formally define a spatial_join as an operation involving all the h features in the following way:

spatial_join$(f_{i_1}, f_{i_2}, ..., f_{i_h}, w, P(f_{i_1}, f_{i_2}, ..., f_{i_h}))$: Determine the locations where occurrences of the features $f_{i_1}, f_{i_2}, ..., f_{i_h}$ satisfy a logical predicate $P(f_{i_1}, f_{i_2}, ..., f_{i_h})$.

A special case of the spatial join operation is the spatial intersection, where the spatial predicate is a feature intersection. This operation is very frequent in spatial applications. It has the following from:

spatial_intersect$(f_{i_1}, f_{i_2}, ..., f_{i_h}, w)$: Determine the locations where occurrences of all the features $f_{i_1}, f_{i_2}, ..., f_{i_h}$ intersect inside window w. This means to output all blocks homogeneous for all of the features $f_{i_1}, f_{i_2}, ..., f_{i_h}$.

Operating on the linear MOF-tree version, to execute the spatial_join query, we have to focus exclusively on the complete nodes of the linear MOF-tree inside the given window, verifying on each of them if the logical clause describing the given predicate is true or not. It follows then, also looking at the extended_select query studied above, that such a process has a worst-case bound on the number of secondary storage accesses of $O(M \log_r T)$. Conversely, using a MLQ representation, once again we have to load each involved quadtree independently, and this implies, as for the above defined extended queries, an I/O time overhead linear in the number of relevant features. Therefore, it is clear that when a certain number of features is involved in a query as a whole, then the MOF-tree introduces an improvement that is linear in the number of features into consideration.

5.2. *Algorithmic Enhancements*

As it has been described earlier, the basic approach to process all the window queries is to decompose the query over an $n \times n$ window into a sequence of smaller queries onto the $O(n)$ maximal blocks contained inside the window [3]. This approach (which will be called *vertical* approach in the sequel) is then to descent the B^+-tree $O(n)$ times, once for each maximal block. However, there is another possible execution strategy that may reduce the number of disk accesses. Instead of descending the tree $O(n)$ times, we can descent it only once with respect to the maximal block which has the smallest locational code value. Then, using the horizontal pointers, present at the leaf level of the B^+-tree, we seek for all the other maximal blocks scanning the pages up to the maximal block with the biggest locational code value. We will call such a strategy the *horizontal* approach. Therefore, it is necessary to define a criterion specifying when to use each approach. Let M_{min} and M_{max} be the two maximal blocks having respectively the smallest and the greatest l-key among those belonging to the window w (see Figure 8).

Let M be the number of maximal blocks inside the window and let N be the number of elements in the linear MOF-tree. We descent the tree two times, one for the M_{min} and one for the M_{max}

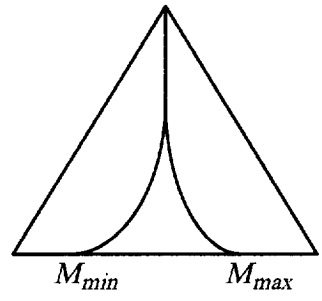

Fig. 8: A B$^+$-tree with the Two Bounds M_{min} and M_{max}.

locational codes. Then we determine the number of pages W that separate the pages of M_{min} and M_{max}; this will be called the page width of the window.

Let us now present the way that the page width is calculated. Assume that each page contains on the average r pointers (the fanout) to the lower and that leaves appear at level 0 and the root at level h (tree height). The key idea is to calculate W incrementally during the B$^+$-tree traversal. Assume that P_a denotes the number of pages between the left-most leaf and M_{min} and P_b the number of pages between the left-most leaf and M_{max}. Also, let j_i and k_i denote the subtree we choose in each level i during the lookup for M_{min} and M_{max} respectively. Clearly, we have:

$$P_a = \sum_{i=1}^{h} (j_i - 1) * r^{i-1}$$

and

$$P_b = \sum_{i=1}^{h} (k_i - 1) * r^{i-1}$$

Therefore, the number of leaf pages separating the pages holding M_{min} and M_{max} is determined by the formula:

$$W = P_b - P_a - 1, \quad \text{if } P_a \neq P_b$$
$$W = 0, \quad \text{if } P_a = P_b$$

Therefore, under the assumption that the retrieval of the B$^+$-tree is not buffered, the horizontal approach is better than the vertical approach if and only if:

$$h + W \leq (M-2)h \iff W \leq (M-3)\left(1 + \log_r \frac{N+1}{2}\right) \tag{14}$$

It is noted that instead of using a very rough approximate for M in this formula (e.g. $M = n$), we could use its expected number, which equals [5, 6]:

$$M = 2^{j+2} - 3j - 4 + 4i + \frac{i^2}{4^j} - \frac{4i}{2^j}$$

where i, j are uniquely determined by the equations $n = 2^j - 1 + i$ and $i \leq 2^j$.

Clearly, Inequality (14) can be used as a criterion. If it is true, then we use the horizontal approach; if it is false, we follow the vertical approach. We anticipate that in general, there will be a reduction in the number of page accesses and therefore the structure can be more competitive in all query types.

6. CONCLUDING REMARKS AND FUTURE WORK

In this paper we have proposed and analyzed space and time complexity of a new access method, namely the MOF-tree, for efficiently storing and processing large 2-dimensional datasets representing multiple overlapping features. We have used as space complexity measure the number of bits used to store the structure in main memory and the number of disk pages to store it in secondary memory, and as performance measure the number of secondary storage accesses for processing window queries. We have analyzed our structure with respect to multiple instances of structures each representing a single feature and we have given theoretical as well as experimental evidence that the number of accesses for answering classical window queries is always competitive with respect to known algorithms using an MLQ representation. Future work could involve:

- the analysis of space complexity under other models of image randomness and time complexity for other widely used operations and for comparisons between MOF-tree and other largely used structures for 2-dimensional data,
- the study of the structure's behavior into a full dynamic environment, where features could be added or deleted arbitrarily,
- investigation of new techniques to reduce the space requirements of the structures, and
- a new MOF-tree implementation along the lines of the Paged Quadtree [20] that may produce a powerful structure for secondary storage.

Acknowledgements — The authors would like to thank Dr. Michael Vassilakopoulos for his valuable comments and suggestions on previous versions of this paper. Research supported by the 4th Italian-Greek bilateral scientific protocol, by the European Union's TMR program ("CHOROCHRONOS" project, contract number ERBFMRX-CT96-0056), by PENED 95 program of the General Secretariat of Research and Technology of Greece and by the "Algoritmi, Modelli di Calcolo e Strutture Informative", 40%-Project of the Italian Ministry for University and Scientific & Technological Research (MURST).

REFERENCES

[1] D.J. Abel. A B^+-tree structure for large quadtrees. *Computer Vision, Graphics and Image Processing*, **27**(1):19-31 (1984).

[2] F. Arcieri and P. Dell'Olmo. A data structure for the efficient treatment of topological join. *Proceedings of the 4th International Symposium on Computer and Information Sciences*, Turkey (1989).

[3] W.G. Aref and H. Samet. Efficient processing of window queries in the pyramid data structure. *Proceedings of the 9th ACM Symposium on Principles of Database Systems (PODS)*, pp.265-272, Nashville, TN (1990).

[4] W.G. Aref and H. Samet. Decomposing a window into maximal quadtree blocks. *Acta Informatica*, **30**:425-439 (1993).

[5] C. Faloutsos. Analytical results on the quadtree decomposition of arbitrary rectangles. *Pattern Recognition Letters*, **13**(1):31-40 (1992).

[6] C. Faloutsos, H.V. Jagadish and Y. Manolopoulos. Analysis of n-dimensional quadtree decomposition of arbitrary rectangles. *IEEE Transactions on Knowledge and Data Engineering*, **9**(3):373-383 (1997).

[7] V. Gaede. Multidimensional access methods. *ACM Computing Surveys*, to appear.

[8] I. Gargantini. An effective way to represent quadtrees. *Communications of the ACM*, **25**(12):905-910 (1982).

[9] O. Guenther. *Efficient Structures for Geometric Data Management*. Lectures Notes in Computer Science No.337, Springer Verlag (1988).

[10] E. Kawaguchi, T. Endo and M. Yokota. Depth-first expression viewed from digital picture processing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **5**:373-384 (1983).

[11] Y. Manolopoulos, E. Nardelli, G. Proietti and M. Vassilakopoulos. On the creation of quadtrees by using a branching process. *Image Vision and Computing*, **14**(2):159-164 (1996).

[12] C. Mathieu, C. Puech and H. Yahia. Average efficiency of data structures for binary image processing. *Information Processing Letters*, **26**(2):89-93 (1987/88).

[13] E. Nardelli and G. Proietti. Efficient secondary memory processing of window queries on spatial data. *Information Sciences*, **80**:1-17 (1994).

[14] E. Nardelli and G. Proietti. An optimal resolution sensitive pyramid representation for hierarchical memory models. *Journal of Computing and Information*, **1**:385-402 (1994).

[15] E. Nardelli and G. Proietti. Probabilistic models for images and quadtrees: Differences and equivalences, Technical Report No.402, Institute for Systems Analysis and Informatics, C.N.R., Roma, January 1995, submitted for publication.

[16] E. Nardelli and G. Proietti. Time and space efficient secondary memory representation of quadtrees, *Information Systems*, **22**(1):25-37 (1997).

[17] C. Puech and H. Yahia. Quadtrees, Octrees, Hyperoctrees: a unified analytical approach to tree data structures used in graphics, geometric modeling and image processing. *Proceedings of the Symposium on Computational Geometry*, pp.272-280, Baltimore, MD (1985),

[18] H. Samet. Computing perimeters of images represented by quadtrees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **3**(6):683-687 (1981).

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

[20] C.A. Shaffer and P.R. Brown. A paging scheme for pointer-based quadtrees, *Proceedings of the 3rd International Symposium on Spatial Databases (SSD)*, pp.89-104, Lecture Notes in Computer Science No.692, Springer-Verlag, Singapore (1993).

[21] S. Tanimoto and T. Pavlidis. A hierarchical data structure for picture processing, *Computer Graphics and Image Processing*, **4**(2):104-119 (1975).

[22] M. Vassilakopoulos and Y. Manolopoulos. Analytical comparison of two spatial data structures. *Information Systems*, **19**(7):269-282 (1994).

[23] M. Vassilakopoulos and Y. Manolopoulos. A random model for analyzing region quadtrees, *Pattern Recognition Letters*, **16**:1132-1145 (1995).