ELSEVIER

Image and Vision Computing 14 (1996) 159–164

image AND vision COMPUTING

Short communication

On the creation of quadtrees by using a branching process☆

Y. Manolopoulos[a,1], E. Nardelli[b,2], G. Proietti[b,2], M. Vassilakopoulos[c,3]

[a]*Department of Informatics, Aristotelian University of Thessaloniki, 540 06 Thessaloniki, Greece*

[b]*IASI, National Research Council, Viale Manzoni 30, 00185 Roma, Italy and Department of Pure & Applied Mathematics, University of L'Aquila, Via Vetoio, Loc. Coppito, 67100 L'Aquila, Italy*

[c]*Department of Electrical & Computer Engineering, Aristotelian University of Thessaloniki, 540 06 Thessaloniki, Greece*

Received 8 March 1995; revised 21 August 1995

Abstract

We present a branching process that builds region quadtrees which represent random images obeying a general probability distribution. As an example of the usefulness of this process we adapt it to images containing one or more sources of black colour. In such images, which often appear in applications, the probability that a pixel is black depends upon its distance from one or more given centres, the sources of black. The process presented can be used for driving a simulator for such an application: it can create and pass to the simulator quadtrees obeying the above model directly, without creating the respective random image first.

Keywords: Quadtree; Branching; Binary image; Random image

1. Introduction

The region quadtree [1] is a very popular hierarchical data structure for the representation of binary images. We can view such an image as a $2^n \times 2^n$ binary array, for some natural number n, where an entry equal to 0 stands for a white pixel and an entry equal to 1 stands for a black pixel. If every pixel of this image is white (black), its quadtree is made up of a single white (black) node. If, however, the image is not unicolour, its quadtree is made up of a grey root that points to four children (subtrees). Each of these subtrees is a quadtree that represents a quadrant of the image. We assume here that the first (leftmost) child of the root corresponds to the North-West quadrant, the second to the North-East quadrant, the third to the South-West quadrant and the fourth (rightmost) child of the root corresponds to the South-East quadrant of the image.

An example of an 8×8 binary image and its quadtree is shown in Figs 1(a) and 1(c), respectively. Note that black (white) squares represent black (white) leaves, while circles represent grey nodes. The unicolour blocks, to which this image is partitioned by the quadtree external nodes, are depicted in Fig. 1(b).

There are many applications (scientific, industrial, etc.) based on region quadtrees in which images obey a random model based on 'black sources'. More specifically, in such images there are one or more pixels which are called *sources*. Consider one of these images. Then, the probability that a pixel of this image is black depends upon its distance(s) from the source(s) of black: this probability is high when the pixel is close to any of the sources, and gets lower as the distance of the pixel from all the sources increases. For a number of reasons, we are often interested in running a simulator during the development of the respective system for such an application. For example, we may want to test whether the system works correctly, or we may want to estimate the

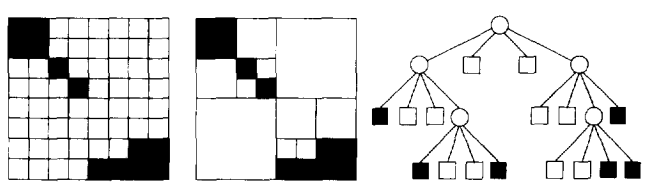

Fig. 1. (a) An 8×8 binary image, (b) the partitioning to unicolour quadrangular blocks, and (c) its quadtree.

☆ Research performed under the 4th Italian-Greek bilateral scientific protocol.

[1] Email: manolopo@eng.auth.gr

[2] Email: {nardelli,guido}@iasi.rm.cnr.it

[3] Email: vasilako@eng.auth.gr

SSDI 0262-8856(95)01059-9

storage requirements and/or the time efficiency of certain algorithms. It is evident that testing often involves a vast number of repetitions. Therefore, the space required for storing a great number of different images would be enormous. On the other hand, real-life or empirical data could be unavailable or unknown. Therefore, one way to run such a simulator would be to create a random image obeying a specific model, build the respective quadtree and pass it to the simulator.

In this paper, we develop a top-down method (a branching process) that builds a quadtree directly, without creating the respective image first. This process creates quadtrees for random images obeying a general probability distribution. These quadtrees can be produced, used and finally discarded in a one-by-one basis, or they may be stored using an efficient method, such as overlapping [2–4]. In any case, the net saving spacewise, when compared to the alternative scenario of storing a large image database, is considerable. Apart from introducing this general branching process, we show how it can be adapted to images containing one or more sources of black colour. The resulting process is suitable for building quadtrees directly and passing them to a simulator as described above. In this way, we save time by overcoming the raster-to-quadtree conversion (an overview of raster-to-quadtree conversion algorithms appears in Samet [5]). As a consequence, we also save intermediate memory space, since quadtrees often require less memory than the respective images (an analysis of the quadtree storage requirements appears in [6]). To the best of our knowledge, there is lack in the literature with respect to a method able to generate asymmetric images (i.e. with black sources) via the construction of quadtrees.

2. General probability distribution

We consider that our image array is the array $I[0\cdots 2^n-1,\ 0\cdots 2^n-1]$. We also consider that the array element (0, 0) corresponds to the NW corner of the image. This means that the array elements $(2^n-1, 0)$, $(0, 2^n-1)$ and $(2^n-1, 2^n-1)$ correspond to the NE, SW and SE corners of the image, respectively. The probability that the array element (j,k) is black, where $0 \le j,k \le 2^n-1$, independently of any other array element is given by the function:

$$f : \{0\cdots 2^n-1\} \times \{0\cdots 2^n-1\} \rightarrow [0,1]$$

The quadtree for I is of height n, at most. Let us call such a tree a class-n quadtree. There are $2^{(4^n)}$ different image arrays of this size. A node corresponding to a single pixel is at level 0, while the root is at level n. There are at most 4^{n-i} nodes at level i, $0 \le i \le n$, each one representing a subarray of $2^i \times 2^i (= 4^i)$ pixels.

The position of a node is fully specified by a triplet (i,x,y), where i is the level of this node and x,y are the coordinates of the NW corner of the respective subarray (or equivalently, x,y are the bit-strings that, when interleaved, make up the path from the root to this node). A node with position (i,x,y), where $0 \le i \le n$ and $0 \le x,y \le 2^{n-i}-1$, corresponds to the subarray $I[2^i x\cdots 2^i(x+1)-1,\ 2^i y\cdots 2^i(y+1)-1]$. If this node is grey, the positions of its NW, NE, SW and SE children are $(i-1,2x,2y)$, $(i-1,2x+1,2y)$, $(i-1,2x,2y+1)$ and $(i-1,2x+1,2y+1)$, respectively. Since the value of function f for an array element is independent to the value of this function for any other array element, we have:

P(the subarray corresp. to (i,x,y) is all black)

$$= \prod_{\substack{j=2^i x\cdots 2^i(x-1)-1 \\ k=2^i y\cdots 2^i(y+1)-1}} f(j,k)$$

P(the subarray corresp. to (i,x,y) is all white)

$$= \prod_{\substack{j=2^i x\cdots 2^i(x-1)-1 \\ k=2^i y\cdots 2^i(y+1)-1}} 1-f(j,k)$$

P(the subarray corresp. to (i,x,y) is not unicolour)

$$= 1 - P(\text{black}, i, x, y) - P(\text{white}, i, x, y)$$

Note that the above probabilities are probabilities of distinct subimages and of sets of subimages. In the following, we will define sets of trees and subtrees and will look for their probabilities.

3. Set of class-*n* quadtrees

In this section we define the set of all class-n region quadtrees Q_n using a number of symbolic recursive equations. These equations involve the cartesian product and union operators and form a constructive top-down definition. This definition leads us to a branching process for building a class-n quadtree. Note that we can view a region quadtree as a finite ordered tuple that consists of grey, black and white nodes and corresponds to the preorder traversal of the tree. Thus, Q_n is a finite set having as elements variable length tuples. Analogous definitions for other trees and combinatorial structures appear in [7].

In the rest of this paper, the + symbol will represent set union. As *class-i sub-quadtree* we characterize a class-i quadtree that has at least one grey node. We will use:

- the symbol $S_{i,x,y}$ to denote the set of all class-i subquadtrees rooted at (i,x,y), where $1 \le i \le n$ and $0 \le x,y \le 2^{n-i}-1$,
- the symbol $G_{i,x,y}$ to denote the set that contains as its only element the grey node at (i,x,y), where $1 \le i \le n$ and $0 \le x,y \le 2^{n-i}-1$,
- the symbol $W_{i,x,y}$ to denote the set that contains as its

only element the white node at (i, x, y), where $0 \leq i \leq n$ and $0 \leq x, y \leq 2^{n-i} - 1$, and

- the symbol $B_{i,x,y}$ to denote the set that contains as its only element the white node at (i, x, y), where $0 \leq i \leq n$ and $0 \leq x, y \leq 2^{n-i} - 1$.

Every notation of the form $\langle T1, T2, T3, T4\rangle_{i,x,y}$ (where each of the symbols $T1$, $T2$, $T3$ and $T4$ represent one of the symbols W, B or S) will denote the subset of $S_{i,x,y}$ defined as the cartesian product:

$$G_{i,x,y} \times T1_{i-1,2x,2y} \times T2_{i-1,2x+1,2y} \times T3_{i-1,2x,2y+1} \times T4_{i-1,2x+1,2y+1}$$

Thus, the set of all class-n quadtrees is:

$$Q_n = W_{n,0,0} + B_{n,0,0} + S_{n,0,0}$$

$$\begin{aligned}
S_{i,x,y} = {} & \langle WWWB\rangle_{i,x,y} + \langle WWBW\rangle_{i,x,y} + \langle WBWW\rangle_{i,x,y} \\
& + \langle BWWW\rangle_{i,x,y} + \langle BBBW\rangle_{i,x,y} + \langle BBWB\rangle_{i,x,y} \\
& + \langle BWBB\rangle_{i,x,y}\langle WBBB\rangle_{i,x,y} + \langle BBWW\rangle_{i,x,y} \\
& + \langle BWWB\rangle_{i,x,y} + \langle WWBB\rangle_{i,x,y} \\
& + \langle WBWB\rangle_{i,x,y} + \langle WBBW\rangle_{i,x,y} + \langle BWBW\rangle_{i,x,y} \\
& + \langle BBBS\rangle_{i,x,y} + \langle BBSB\rangle_{i,x,y} + \langle BSBB\rangle_{i,x,y} \\
& + \langle SBBB\rangle_{i,x,y} + \langle WWWS\rangle_{i,x,y} \\
& + \langle WWSW\rangle_{i,x,y} + \langle WSWW\rangle_{i,x,y} \\
& + \langle SWWW\rangle_{i,x,y} + \langle WWBS\rangle_{i,x,y} \\
& + \langle WBWS\rangle_{i,x,y} + \langle BWWS\rangle_{i,x,y} + \langle WWSB\rangle_{i,x,y} \\
& + \langle WBSW\rangle_{i,x,y} + \langle BWSW\rangle_{i,x,y} + \langle WSWB\rangle_{i,x,y} \\
& + \langle WSBW\rangle_{i,x,y} + \langle BSWW\rangle_{i,x,y} + \langle SWWB\rangle_{i,x,y} \\
& + \langle SWBW\rangle_{i,x,y} + \langle SBWW\rangle_{i,x,y} + \langle BBWS\rangle_{i,x,y} \\
& + \langle BWBS\rangle_{i,x,y} + \langle WBBS\rangle_{i,x,y} + \langle BBSW\rangle_{i,x,y} \\
& + \langle BWSB\rangle_{i,x,y} + \langle WBSB\rangle_{i,x,y} + \langle BSBW\rangle_{i,x,y} \\
& + \langle BSWB\rangle_{i,x,y} + \langle WSBB\rangle_{i,x,y} + \langle SBBW\rangle_{i,x,y} \\
& + \langle SBWB\rangle_{i,x,y} + \langle SWBB\rangle_{i,x,y} + \langle WWSS\rangle_{i,x,y} \\
& + \langle WSWS\rangle_{i,x,y} + \langle SWWS\rangle_{i,x,y} + \langle SWSW\rangle_{i,x,y} \\
& + \langle SWSW\rangle_{i,x,y} + \langle SSWW\rangle_{i,x,y} + \langle BBSS\rangle_{i,x,y} \\
& + \langle BSBS\rangle_{i,x,y} + \langle SBBS\rangle_{i,x,y} + \langle SBSB\rangle_{i,x,y} \\
& + \langle SBSB\rangle_{i,x,y} + \langle SSBB\rangle_{i,x,y} + \langle WBSS\rangle_{i,x,y} \\
& + \langle WSBS\rangle_{i,x,y} + \langle SWBS\rangle_{i,x,y} + \langle SWSB\rangle_{i,x,y} \\
& + \langle SWSB\rangle_{i,x,y} + \langle SSWB\rangle_{i,x,y} + \langle BWSW\rangle_{i,x,y} \\
& + \langle BSWS\rangle_{i,x,y} + \langle SBWS\rangle_{i,x,y} + \langle SBSW\rangle_{i,x,y} \\
& + \langle SBSW\rangle_{i,x,y} + \langle SSBW\rangle_{i,x,y} + \langle SSSW\rangle_{i,x,y} \\
& + \langle SSWS\rangle_{i,x,y} + \langle SWSS\rangle_{i,x,y} + \langle WSSS\rangle_{i,x,y} \\
& + \langle SSSB\rangle_{i,x,y} + \langle SSBS\rangle_{i,x,y} + \langle SBSS\rangle_{i,x,y} \\
& + \langle BSSS\rangle_{i,x,y} + \langle SSSS\rangle \quad \forall\, i > 1
\end{aligned}$$

$$\begin{aligned}
S_{1,x,y} = {} & \langle WWWB\rangle_{1,x,y} + \langle WWBW\rangle_{1,x,y} \\
& + \langle WBWW\rangle_{1,x,y} + \langle BWWW\rangle_{1,x,y} \\
& + \langle BBBW\rangle_{1,x,y} + \langle BBWB\rangle_{1,x,y} + \langle BWBB\rangle_{1,x,y} \\
& + \langle WBBB\rangle_{1,x,y} + \langle BBWW\rangle_{1,x,y} \\
& + \langle BWWB\rangle_{1,x,y} + \langle WWBB\rangle_{1,x,y} \\
& + \langle WBWB\rangle_{1,x,y} + \langle WBBW\rangle_{1,x,y} \\
& + \langle BWBW\rangle_{1,x,y}
\end{aligned}$$

It is not difficult to see that Q_n includes all the possible class-n quadtrees, since its definition is a top-down constructive one, where all the possible configurations for the children of a grey node appear (excluding the two illegal configurations where all four children are unicolour leaves).

Note that the above symbolic equations (when fully expanded) define the set of all class-n quadtrees as a large expression made of union and cartesian product operators and a number of initial operands, the sets $G_{i,x,y}$, $W_{i,x,y}$ and $B_{i,x,y}$.

4. Branching process

These symbolic equations describe a branching process by which we can construct any legal class-n region quadtree. More specifically:

- At the start, we perform the *initial branching*: we choose between the root being a black node, a white node or a grey node (the tree being a member of the set of class-n sub-quadtrees).
- At any level $i, n \geq i > 1$, for any grey node at this level, we perform a *level-i branching*: we choose between 79 different sub-quadtree subsets so that the subtree rooted at this node belongs in the chosen set.
- At level 1, for any grey node at this level, we perform a *level-1 branching*: we choose between 14 different subquadtree subsets so that the subtree rooted at this node belongs in the chosen set.

A branching process for creating quadtrees was first introduced in [8], where at each node (starting at the root), we always choose between white, black and grey colours; if we colour a node grey, we continue the process recursively for each one of its children choosing always between all these three colours. Note that according to the approach of [8] a grey node may have four children that are all black or white (the branching process is not restricted so as not to produce four sibling leaves of the same colour). The resulting set of trees is a superset of the

usual quadtrees. The trees that belong in this set and are not usual quadtrees loose (part of) the memory savings introduced by creating maximal white and black blocks, and are not used in practice. A branching process that builds only usual region quadtrees is described in [9]. This process is similar to the process of this paper, however it is simpler since all level-i sub-quadtrees are treated equivalently (the resulting random model is symmetrical in regard to every grey node). Note that in the branching process of this paper and in the branching process of [9] except for the initial branching (where we choose between different colours) at all other branchings we choose between different sub-quadtree configurations (we choose the sub-quadtree subset defined by a $\langle\,\rangle$ notation in which the subtree rooted at the grey node of each branching belongs). This approach is different to the approach of [8], where we always choose between three colours.

We can create a probabilistic model for our branching process by assigning probabilities to the different choices for every branching. This process must be legal under the fundamental probability axioms. Thus, for every specific branching the probabilities of all the different choices must sum to 1. We want an assignment of probabilities that makes the branching process create trees which correspond to images obeying the random model defined in Section 2. In other words, the probability of a specific class-n quadtree (class-i sub- quadtree) is equal to the probability of the image (subimage) it represents. By adding the probabilities of the distinct trees (subtrees) present in the related set of trees (subtrees), we get that:

- $P(B_{i,x,y}) = P$(the subarray corresp. to (i,x,y) is all black), where $0 \le i \le n$ and $0 \le x, y \le 2^{n-i} - 1$,
- $P(W_{i,x,y}) = P$(the subarray corresp. to (i,x,y) is all white), where $0 \le i \le n$ and $0 \le x, y \le 2^{n-i} - 1$,
- $P(S_{i,x,y}) = P$(the subarray corresp. to (i,x,y) is not unicolour), where $1 \le i \le n$ and $0 \le x, y \le 2^{n-i} - 1$ and
- for $T1, T2, T3, T4 \in \{\text{'}W\text{'}, \text{'}B\text{'}, \text{'}S\text{'}\}$:

$$P(\langle T1, T2, T3, T4\rangle_{i,x,y}) = P(T1_{i-1,2x,2y}) \cdot P(T2_{i-1,2x+1,2y}) \times P(T3_{i-1,2x,2y-1}) \cdot P(T4_{i-1,2x+1,2y+1})$$

where $1 \le i \le n$ and $0 \le x, y \le 2^{n-i} - 1$.

Our assignment of probabilities is:

- At the initial branching, the probability that our tree is a white, black or grey node equals $P(W_{n,0,0})$, $P(B_{n,0,0})$ and $P(S_{n,0,0})$, respectively. Obviously, these probabilities sum to 1.
- At any level-i branching, $n \ge i > 1$, the probability of each of the 79 different choices of the form $P(\langle T1, T2, T3, T4\rangle_{i,x,y})$ equals:

$$\frac{P(\langle T1, T2, T3, T4\rangle_{i,x,y})}{P(S_{i,x,y})}$$

We had to divide by the probability that the subarray corresponding to (i,x,y) is not unicolour in order for the sum of all the 79 probabilities to be 1. This is because the set of all 79 choices does not include the two illegal configurations where all four children are unicolour leaves.

- At any level-1 branching the probability of each of the 14 different choices of the form $P(\langle T1, T2, T3, T4\rangle_{1,x,y})$ equals:

$$\frac{P(\langle T1, T2, T3, T4\rangle_{1,x,y})}{P(S_{1,x,y})}$$

For the same reason as above, we had to divide by the probability that the subarray corresponding to $(1,x,y)$ is not unicolour.

5. Models of black sources

In this section, we explain how the above branching process can be adapted to images containing one or more sources of black. To make the presentation simpler, we consider that there is only one source of black in our images, that is an element of I with coordinates x_s and y_s, $2^n - 1 \ge x_s, y_s \ge 0$. Function f should be replaced with a function:

$$d_{x_s,y_s} : \{0 \cdots 2^n - 1\} \times \{0 \cdots 2^n - 1\} \to [0,1]$$

The value of this function for an element (x,y) of I should depend on the distance of (x,y) from (x_s,y_s), i.e. the smaller this distance, larger is the value of the function, and vice versa. If we calculate the above probabilities using function d_{x_s,y_s}, our branching process is fully specified and can be used to produce quadtrees for images obeying the source-of-black random model. Two popular distance functions are the *euclidean* distance and the *manhattan* distance. These functions should be normalized with respect to the largest possible value of each function in an image. This means that the euclidean definition of function d_{x_s,y_s} could be:

$$d_{x_s,y_s}(x,y) = 1 - \frac{\sqrt{(x_s - x)^2 + (y_s - y)^2}}{g_e(2^n)}$$

where $g_e(2^n)$ is a function which depends on the image resolution and determines the degree of image coherence (the aggregation of black pixels) around the sources of black. This function should never give values equal to zero, and in any case, it should be larger or equal to the greatest distance for a specific resolution. An example of this function is: $g_e(\,) = \sqrt{2}(2^n - 1)$. In a similar manner,

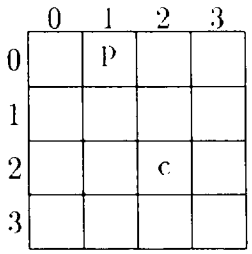

Fig. 2. A 4 × 4 image containing a black centre c and a black pixel p.

the manhattan definition of function d_{x_s,y_s} could be:

$$d_{x_s,y_s}(x,y) = 1 - \frac{(x_s - x) + (y_s - y)}{g_m(2^n)}$$

In this case, the function $g_m(2^n)$ could be equal to $2(2^n - 1)$. For example, consider the $2^2 \times 2^2$ image of Fig. 2. The value of the euclidean function d_{x_s,y_s} of pixel p and black centre c is calculated as:

$$d_{2,2}(1,0) = 1 - \frac{\sqrt{(2-1)^2 + (2-0)^2}}{\sqrt{23}} = 1 - \frac{\sqrt{5}}{\sqrt{23}} = 0.47$$

whereas the value of the manhattan function d_{x_s,y_s} of pixel p and black centre c is calculated as:

$$d_{2,2}(1,0) = 1 - \frac{2 - 1 + 2 - 0}{2 \cdot 3} = 0.5$$

It is not difficult to calculate (by using these definitions of d_{x_s,y_s}) the probabilities for the different choices of each branching. The basis of these calculations for the euclidean d_{x_s,y_s} are the probabilities:

P(the subarray corresp. to (i,x,y) is all black)

$$= \frac{1}{g_e(2^n)} \prod_{\substack{j=2^i x \cdots 2^i(x+1)-1 \\ k=2^i y \cdots 2^i(y+1)-1}} \sqrt{(x_s - j)^2 + (y_s - k)^2}$$

P(the subarray corresp. to (i,x,y) is all white)

$$= \frac{1}{g_e(2^n)} \prod_{\substack{j=2^i x \cdots 2^i(x+1)-1 \\ k=2^i y \cdots 2^i(y+1)-1}} \sqrt{2}(2^n - 1) - \sqrt{(x_s - j)^2 + (y_s - k)^2}$$

and for the manhattan d_{x_s,y_s} are the probabilities:

P(the subarray corresp. to (i,x,y) is all black)

$$= \frac{1}{g_m(2^n)} \prod_{\substack{j=2^i x \cdots 2^i(x+1)-1 \\ k=2^i y \cdots 2^i(y+1)-1}} (x_s - j) + (y_s - k)$$

P(the subarray corresp. to (i,x,y) is all white)

$$= \frac{1}{g_m(2^n)} \prod_{\substack{j=2^i x \cdots 2^i(x+1)-1 \\ k=2^i y \cdots 2^i(y+1)-1}} \times 2(2^n - 1) - (x_s - j) - (y_s - k)$$

In an analogous manner, similar formulae can be produced for images with two or more black sources.

6. Conclusion

In this paper, we present a branching process that can build region quadtrees for images obeying a general probabilistic model. This model depends upon the definition of a probability distribution $f : \{0 \cdots 2^n - 1\} \times \{0 \cdots 2^n - 1\} \rightarrow [0,1]$. By defining f appropriately, this branching process can build trees for images obeying the source-of-black random model. It is not difficult to extend the presented method to include images with more than one sources of black. The process presented looks complicated, since in most steps it distinguishes between 79 choices. However, we had to consider that many cases to avoid creation of illegal region quadtrees.

This branching process can be used for driving a simulator for applications that involve region quadtrees and images obeying the source-of-black random model or another random model, according to the definition of the general probability distribution f. Our method saves time and memory space, since it does not create temporary random images which then have to be transformed to quadtrees. Moreover, we note that our method saves space for an additional reason. During the branching process, as presented above, probability values are generated for each and every node of a full quadtree. These probability values can be either ignored and recalculated later on-the-fly at the cost of CPU overhead, or otherwise they have to be stored. In the former case, only a limited and constant number of floating point parameters has to be stored, such as the black source coordinates and the code to calculate the distance function. In the latter case, the necessary space is considerable, but again, it is negligible when compared to the storage requirements of a large set of quadtrees representing an image database. Hence, for any image of a non-trivial size, the saving with respect to storing the values of all pixels is very significant, even if each pixel requires just one bit.

References

[1] H. Samet, The quadtree and related hierarchical data structures, *ACM Computing Surveys, 16* (2) (1984) 187–260.

[2] E. Nardelli and G. Proietti, Managing overlapping features in spatial database applications, in *Proc. International Computer Symposium (ICS94), Taiwan, China, December 1994.*

[3] G. Proietti, A space efficient representation of image containing multiple overlapping features, *Rivista di Informatica of the Italian Association for Automated Computing (AICA)*, May 1995.

[4] M. Vassilakopoulos, Y. Manolopoulos and K. Economou, Overlapping quadtrees for the representation of similar images, *Image and Vision Computing, 11* (5) (1993) 257–262.

[5] H. Samet, *Applications of Spatial Data Structures*, Addison-Wesley, Reading, MA, 1990.

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

[7] J.S. Vitter and Ph. Flajolet, Average-case analysis of algorithms and data structures, in *Handbook of Theoretical Computer Science, Vol. A, Algorithms and Complexity*, Chapter 9, Elsevier, The Netherlands, 1990.

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

[9] M. Vassilakopoulos and Y. Manolopoulos, On random models for analyzing region quadtrees, *Pattern Recognition Letters*, accepted.