Overlapping Linear Quadtrees: a Spatio-temporal Access Method*

Theodoros Tzouramanis *ttzouram@athena.auth.gr*

Michael Vassilakopoulos *mvass@computer.org*

Yannis Manolopoulos *manolopo@eng.auth.gr*

Department of Informatics
Aristotle University
54006 Thessaloniki, Greece

Abstract

Overlapping is a technique used in access methods to combine consecutive structure instances into a single structure by not storing identical sub-structures. This way, space is saved without sacrificing time performance. Here, we present the structure of Overlapping Linear Quadtrees which is used to store consecutive raster images according to transaction time. Experimentation with synthetic region data shows that considerable storage is saved in comparison to independent linear quadtrees, in the case of similar consecutive images. Therefore, this structure can be used in spatio-temporal databases to support query processing of evolving images. Besides, an efficient algorithm that uses the new structure and answers spatio-temporal window queries is presented.

Index terms: Spatio-temporal databases, transaction time, access methods, indexing, B^+trees, quadtrees, linear quadtrees, overlapping, time/space performance.

1 Introduction

Several spatial access methods have been proposed in the literature, for storing multi-dimensional objects (e.g. points, line segments, areas, volumes, and hyper-volumes) without considering the notion of time. These methods are classified in one of the following two categories according to the principle guiding the hierarchical decomposition of data regions in each method.

- Data space hierarchy: a region containing data is split (when, for example, a maximum capacity is exceeded) to sub-regions which depend on these data only (for example, each of two sub-regions contains half of the data)
- Embedding space hierarchy: a region containing data is split (when a certain criterion holds) to sub-regions independently of these data (for example, a square region is always split in four quadrant sub-regions)

The book by Samet [14] and the recent survey by Gaede and Guenther [4] provide excellent information sources for the interest reader.

On the other hand, temporal access methods have been proposed to index data varying over time without considering space at all. The notion of time may be of two types: transaction time (i.e., time when the fact is current in the database and may be retrieved) and valid time (i.e., time when the fact is true in the modeled reality) [6]. A temporal DBMS would support at least one of these two types of time. Thus, we distinguish among three temporal DBMSs: valid-time (in the past called historical), transaction-time (in the past called rollback), and bi-temporal databases. Indexing methods are also classified according to the above taxonomy. A wide range of access methods has been proposed to support multi-version/temporal data by keeping track of data evolution over time. For excellent recent surveys on temporal access methods see [11, 13].

Until recently the field of temporal databases and spatial databases remained two separate worlds. However, modern applications (e.g. geographical information systems, multimedia systems, scientific and statistical databases, such as medical, meteorological, astrophysics oriented databases) involve the efficient manipulation of moving spatial objects, and the relationships among them. For instance, Worboys [21] provides a survey of (mainly GIS oriented) spatio-temporal applications on administrative areas, road networks, and land ownership. Therefore, there is an emerging growing need to study the case of "spatio-temporal databases". According to the first attempt towards a specification and classification scheme for spatio-temporal access methods [17], up until now, only four spatio-temporal indexing methods have appeared in the literature: 3D R-trees [16], MR-trees and RT-trees [22], and HR-trees [12]. These approaches have the following characteristics:

- 3D R-trees treat time as another dimension using a state-of-the-art spatial indexing method, namely the R-tree [5],
- MR-trees and HR-trees use overlapping in R-trees to represent successive states of the database, and
- RT-trees couple time intervals with spatial ranges in each node of the tree structure by adopting ideas from R-trees and TSB-trees [8].

*Research performed under the European Union's TMR Chorochronos project, contract number ERBFMRX-CT96-0056 (DG12-BDCN).

All these methods are extensions of the R-tree, which is based on the "conservative approximation principle", i.e. spatial objects are indexed by considering their minimum bounding rectangle (MBR). These methods are not suitable for representing regional data, in cases where a lot of empty ("dead") space is introduced in the MBRs, since this fact decreases the index ability to prune space and objects during a top-bottom traversal.

In the present paper, we follow a different paradigm, that of quadtrees, by using quadcodes to decompose image data in an exact (i.e. non-rough) manner. More specifically, we present an efficient indexing and retrieval method for regional data and describe the design and implementation of a new structure: Overlapping Linear Quadtrees. Evidently, this structure is based on linear quadtrees which are enhanced by using the overlapping technique in order to avoid storing identical sub-quadrants of successive instances of image data evolving over transaction time.

The rest of the paper is organized as follows. Section 2 describes the building blocks of the implementation of Overlapping Linear Quadtrees. Section 3 describes an efficient algorithm that uses the new structure and answers spatio-temporal window queries. In Section 4 we present the experimental setting and show that the new structure demonstrates a remarkable storage performance. We conclude in Section 5, suggesting also directions for future work.

2 The new structure

The notion of overlapping consecutive instances of access methods has been mentioned in the previous section. Except of the cases of MR-trees and HR-trees, overlapping has been also used in a number of occasions, where successive data snapshots are similar. For example, it has been used as a technique to compress similar text files [2], B-trees and B^+trees [3, 9, 18], as well as main-memory quadtrees [19, 20]. In this section, first we make a short presentation of region quadtrees and the application of overlapping to these trees when they are stored in main memory, and second we describe the application of overlapping to secondary memory quadtree variations.

2.1 Region Quadtrees

The region quadtree is the most popular member in the family of quadtree-based access methods. It is used for the representation of binary images, that is $2^n \times 2^n$ binary arrays (for a positive integer n), where a 1 (0) entry stands for a black (white) picture element. More precisely, it is a degree four tree with height n, at most. Each node corresponds to a square array of pixels (the root corresponds to the whole image). If all of them have the same color (black or white) the node is a leaf of that color. Otherwise, the node is colored gray and has four children. Each of these children corresponds to one of the four square sub-arrays to which the array of that node is partitioned. We assume here, that the first (leftmost) child corresponds to the upper left sub-array, the second to the upper right sub-array, the third to the lower left sub-array and the fourth (rightmost) child to the lower right sub-array. For more details regarding quadtrees see [14]. Figure 1.c shows a quadtree for the 8 × 8 pixel array of Figure 1.a. Note that black (white) squares represent black (white) leaves, whereas circles represent gray nodes.

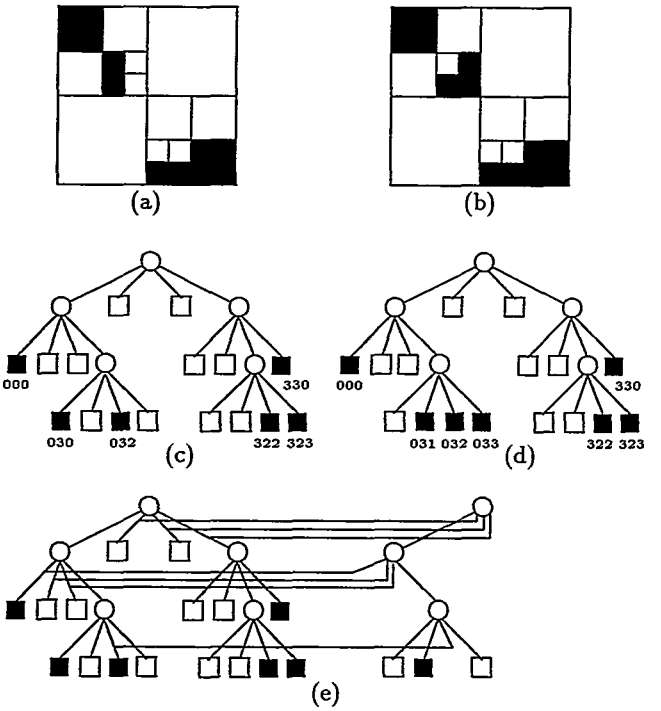

Figure 1: (a),(b): two similar 8×8 images, (c),(d): the corresponding quadtrees and (e): the overlapped structure.

2.2 Overlapped Pointer Quadtrees

Region quadtrees, as presented above, can be implemented as main memory tree structures (each node being represented as a record that points to its children). We can save memory space by overlapping quadtrees, that is by using the same memory area for storing common subtrees of different trees. Consider two quadtrees that represent an image and a variation of that image. If the two images have common sub-images covering the same co-ordinates, the corresponding subtrees of the two trees that represent these images are identical and may be kept only once in memory, while they belong to paths of both trees. In case there are not any common sub-images between the two images we have two distinct trees. In case the two instances are two identical images so are the trees (the pointers to the roots of the trees remain distinct although they point to the same node). If there are not two images only, but a long sequence of images representing for example a gradually changing image, common sub-structures may be shared by more than two consecutive trees. This sharing is transparent to the algorithms accessing the trees. In other words, there is not any access time overhead because of overlapping. Figures 1.a and 1.b demonstrate two similar images and Figures 1.c and 1.d demonstrate the respective quadtrees. The result of applying overlapping to these trees is shown in Figure 1.e. Note, that when there is a different sub-structure (which might even be a different pixel for a certain image position) the total path to this sub-structure is stored. This guarantees that the access time of any node remains the same.

2.3 Overlapping Linear Quadtrees

Variations of region quadtrees have been developed for secondary memory. Linear region quadtrees are the ones used most extensively. A linear quadtree representation consists

of a list of values where there is one value for each black node of the pointer-based quadtree. The value of a node is an address describing the position and size of the corresponding block in the image. These addresses can be stored in an efficient structure for secondary memory (such as a B-tree or one of its variations). There are also variations of this representation where white nodes are stored too, or variations which are suitable for multicolor images. Evidently, this representation is very space efficient, although it is not suited to many useful algorithms that are designed for pointer-based quadtrees. The most popular linear implementations are the FL (Fixed Length), the FD (Fixed length – Depth) and the VL (Variable length) linear implementations [15].

In the FL implementation, the address of a black Quadtree node is a code-word that consists of n base 5 digits. Codes 0, 1, 2 and 3 denote directions NW, NE, SW and SE, respectively, while code 4 denotes a do-not-care direction. If the black node resides on level i, where $n \geq i \geq 0$, then the first $n - i$ digits express the directions that constitute the path from the root to this node and the last i digits are all equal to 4. In the FD implementation, the address of a black quadtree node has two parts: the first part is code-word that consists of n base 4 digits. Codes 0, 1, 2 and 3 denote directions NW, NE, SW and SE, respectively. This code-word is formed in a similar way to the code-word of the FL-linear implementation with the difference that the last i digits are all equal to 0. The second part of the address has $\lceil log_2(n+1) \rceil$ bits and denotes the depth of the black node, or in other words, the number of digits of the first part that express the path to this node. In the VL implementation the address of a black quadtree node is a code-word that consists of at most n base 5 digits. Code 0 is not used in addresses, while codes 1, 2, 3 and 4 denote one of the four directions each. If the black node resides on level i, where $n \geq i \geq 0$, then its address consists of $n - i$ digits expressing the directions that constitute the path from the root to this node. The depth of a node can be calculated by finding the smallest value equal to a power of 5 that gives 0 quotient when the address of this node is divided (using integer division) with this value.

Each quadtree, in a sequence of quadtrees modeling time evolving images, can be represented in secondary memory by storing its linear FD codes in a B^+tree. The new structure, called Overlapping Linear Quadtrees, is formed by overlapping consecutive B^+trees. Since in the same quadtree two black nodes that are ancestor and descendant cannot coexist, two FD linear codes that coincide at all the directional digits cannot exist neither. This means that the directional part of the FD codes is sufficient for building B^+trees at all the levels. At the leaf-level, the depth of each black node should also be stored so that images are accurately represented and that overlapping can be correctly applied. In Figures 1.c and 1.d you can see the directional code of each black node of the two depicted trees. The above part of Figure 2 depicts the B^+trees that correspond to the trees of Figures 1.c and 1.d and the below part depicts the resulting overlapped linear structure. Note that in Overlapping Linear Quadtrees there is no extra cost for accesses in a specific linear quadtree.

The choice of FD linear representation, instead of the other two linear representations, is not accidental. The FD linear representation is made of base 4 digits and is thus easily handled using two bits for each digit. Besides, the sorted sequence of FD linear codes is a depth-first traversal of the tree. Since internal and white nodes are omitted, this means that sibling black nodes are stored consecutively in

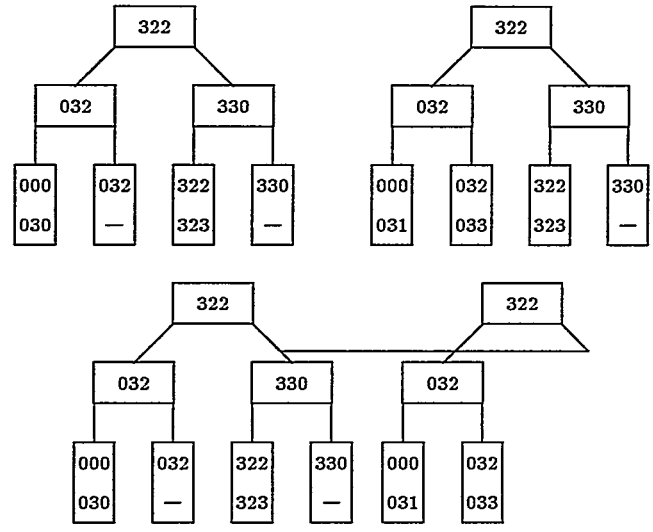

Figure 2: Two B^+trees storing linear quadtree codes and the corresponding linear overlapped structure.

the B^+tree and there is increased probability for the same image part to reside in the same leaf between consecutive B^+trees. This property maximizes the probability that a leaf will not change and will be overlapped between consecutive B^+trees, since consecutive images have large identical parts. In order to make this probability even higher, the B^+tree leaves hold a small number of codes (a few black nodes are very likely to remain unchanged). Note, also, that in our implementation a disk page may host a number of consecutive B^+tree leaves.

During an image update, a number of FD quadcode deletions from, and a number of FD quadcode insertions into the last B^+tree instance has to be performed. However, as usually happens, given a new image, we do not know beforehand which exactly are the quadcode insertions and deletions. Thus, we face image updates in two stages. The first stage is to sort the quadcodes of the new image version and compare this sequence against the set of quadcodes of the last image version, which resides in a binary table (see Section 4 for details). The next stage follows the approach of [7], and builds the new tree instance by performing these insertions/deletions in a batched manner, instead of performing them one at a time.

A consequence of this technique is that a specific B^+ tree leaf may accept a number of quadcode insertions much greater than the number of available free slots. Thus, a specific leaf may split in more than two nodes. In a similar manner, more than two B^+tree consecutive leaves may merge during quadcode deletions.

On top of Overlapping Linear Quadtrees, another tree structure is used to index transaction-time values and to point to the roots of the respective B^+trees. This tree structure has a reasonable size, hence it can be stored in main memory. All nodes of Overlapping Linear Quadtrees have an extra field, called "StartTime", that can be used to detect whether a node is being shared by other trees. We assign a value to StartTime during the creation of the corresponding node and there is no need for future modification of this field. Moreover, leaf-nodes have one more extra field, called "EndTime", that is used to register the transaction time

when a specific leaf changes and becomes historical.

In order to keep track of the image evolution (in other words, the evolution of quadcodes) and efficiently satisfy spatio-temporal queries over the stored raster images, we embed some additional horizontal pointers in the B^+tree leaves. This way there will be no need to top-down traverse consecutive tree instances to search for a specific quadcode, thus avoiding excess page accesses. More specifically, we embed four additional pointers in every B^+tree leaf to support spatio-temporal queries. Names and roles of these pointers are:

B-pointer is used during a temporal query to traverse the structure backwards. When not-null, it points to a historical leaf from the previous tree instance. The node accommodating a not-null B-pointer has been created after a merge/split/update of that historical node.

BC-pointer is also used during a temporal query to traverse the structure backwards. The node accommodating a not-null BC-pointer is always historical. BC-pointer always refers to a historical leaf of the same tree instant. This field is involved in the merge procedure.

F-pointer is used during a temporal query to traverse the structure forwards. The node which accommodates a not-null F-pointer is always historical. In such a case F-pointer points to a successive tree instant leaf, which had been created from this specific historical node after a split/merge/update.

FC-pointer is also used during a temporal query to traverse the structure forwards. When this pointer is not-null, it points to a node of the same tree instant, which had been created after a split.

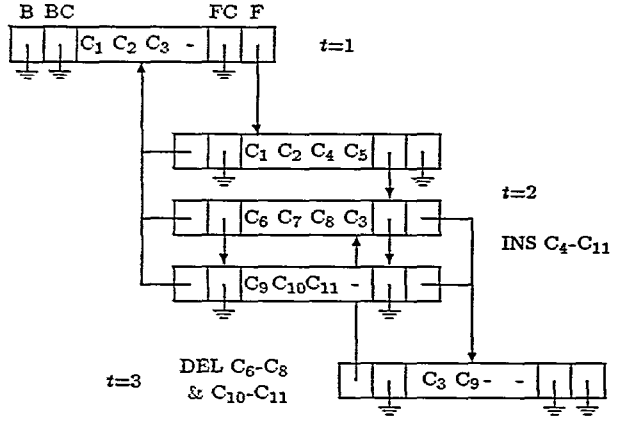

Figure 3: Forward and backward chaining for the support of temporal queries.

Figure 3 shows how the leaves of three successive B^+trees can be forward- and backward-chained to support temporal queries. The leaf on the left-top corner of the figure corresponds to the first time instant, t=1, and contains the FD codes C_1=002/3, C_2=003/3 and C_3=302/3 (the form of the codes is direction/depth). Suppose that during time instant t=2, eight quadcodes with keys C_4=030/3, C_5=031/3, C_6=032/3, C_7=200/2, C_8=211/3, C_9=330/3, C_{10}=331/3, and C_{11}=332/3 are inserted. In such a case, we have a node split. In other words, we have to allocate three new B^+tree leaves at time instant t=2, in order to accommodate eleven quadcodes in total. The leaf in question of time instant t=1 is connected to the first of these three new nodes by using the F-pointer field, whereas the FC-pointer field is used to chain together the three new nodes. During time instant t=3, a set of 5 quadcodes is deleted, namely the quadcodes with key values C_6, C_7, C_8, C_{10} and C_{11}. Thus, two nodes of the tree corresponding to time instant t=2 are merged to produce a new node as depicted in the figure. B-pointer and BC-pointer fields are maintained accordingly.

The structure of Overlapping Linear Quadtrees presents small differences from the structure of Overlapping B^+trees. The former structure has an auxiliary substructure, a binary table that keeps the set of quadcodes of the last image version (see Section 4 for details). Besides, a disk page may host a number of consecutive B^+tree leaves. On the other hand in the latter structure there is a direct correspondence between pages and leaves, assuming that each leaf entry is accompanied with extra data related to it. Nevertheless, Overlapping Linear Quadtrees are used in a completely different context than Overlapping B^+trees: they do not store ordinary numeric data, but they store quadcodes that represent image parts and they are used for answering temporal window queries (see the next section). More details about Overlapping B^+trees and their use can be found in [18].

3 Temporal Window Query Processing

In Spatial Databases and Geographical Information Systems there exists the need for processing a significant number of different spatial queries. For example, nearest neighbor finding, similarity queries, spatial joins of various kinds, window queries, etc. In this section we provide a solution for the *temporal window strict containment* query and the *temporal window partially overlap* query for evolving regional data. Given a window belonging in the area covered by our images and a time interval the following spatio-temporal query may be expressed:

- Find the black regions of the window at all the time points within the interval.

Although this kind of query is very likely to be made by users, no efficient method for its processing has been presented so far. In this subsection, we present such a method as a sequence of steps.

1. The window is broken into maximal sub-quadrants, as if it were a black region represented by a region quadtree.

2. For each of these sub-windows, the smallest and largest directional codes that may appear in the sub-window are computed. The range of these codes includes all the codes of the black sub-quadrants that are strictly included within the sub-window. A respective range search is performed in the B^+tree of the first time point and the leaves that either contain such codes or would contain them if they had been inserted are discovered. The codes that fall within the above range and appear in these nodes are the black sub-quadrants that are strictly contained within the window for the specific time point.

3. For each leaf discovered, following the F-pointer at first step and the chain of FC-pointers at second step, the leaves that *evolve* from this leaf at the next time point are discovered. The leaves that do not intersect with the range specified in step 2 are discarded from further consideration. The codes that fall within this range and appear in the remaining leaves are the black sub-quadrants that are strictly contained within the sub-window for the specific time point.

We proceed to the tree for the next time point by repeating step 3 for each remaining leaf. We stop when we reach the last time point of the time interval.

An analogous procedure could be followed by starting from the end of the time interval and by using the B-pointer and BC-pointers.

Note that the above procedure finds all the black sub-quadrants that fall strictly within the query window. In order to be able to find black regions that partially overlap with the window, the directional codes for all the ancestors of the sub-windows specified in step 1 must be calculated. Since an ancestor may be common to two or more sub-windows, duplicate entries must be eliminated. Then, the leaves of the B^+tree for the first time point that either contain such codes or would contain them if they had been inserted are discovered. These leaves are treated in a similar manner as the leaves that concern strictly contained black sub-quadrants (step 3).

An alternative algorithm for answering this spatio-temporal query would be to perform a suitable range search for all the trees that correspond to the given time interval. By intuition, we claim that the algorithm presented, when applied to a sequence of similar images, will demonstrate an O(# of leaves needed for answering the query + range search at the first tree) time performance on the average. The alternative algorithm will perform in time that is O(range search at the 1st tree + range search at the 2nd tree +...+ range search at the last tree). Thus, the algorithm presented is very promising in terms of expected execution time. Its theoretical performance analysis is among the issues currently studied by the authors.

4 Experiments

We implemented the structure of Overlapping Linear Quadtrees in C++. The maximum and minimum capacity of internal nodes (leaves) were 250 and 125 keys (10 and 5 FD codes), respectively. Therefore, with a page size of 2.048 bytes, the size of each internal node (leaf) was one (1/20 of a) page. The size of our images was 256 × 256 pixels and we used the algorithm OPTIMAL_BUILD described in [15] for converting the images from raster to linear FD representation. Every experiment was repeated 50 times using a pair of similar images. At the start, the first image was created and its FD codes were inserted in an empty B^+tree. The codes were inserted one at a time, as they were produced by OPTIMAL_BUILD. Thus, we obtained the result of a typical B^+tree with average storage utilization equal to ln 2. Next, the second image was created as an alternation of the first image and its FD codes were inserted in the second B^+tree, so that the identical subtrees between the two trees overlapped. The codes were first sorted in increasing order, so that an algorithm which performs batch modifications (i.e. insertions, deletions and updates), along the lines of [7], could be used. There was no I/O cost for black quadrants that were identical between the two consecutive images, since, by keeping the quadcodes of the last inserted image in a main memory compacted binary array (256 × 256/8 bytes), we were able to sort out the respective identical FD codes.

The main goal of our experiments was to count the average storage gain measured as a ratio of the number of common B^+tree nodes between the two images over the number of the B^+tree nodes of the second image. The parameters that varied in our experiments were:

- The black/white analogy, i.e. the percentage of the black pixels (from 50% up to 95%) for the creation of the first random image. Note that a random image is not considered very realistic when the black/white analogy does not differ significantly from 50%, since the image created includes very small unicolor regions and corresponds to an almost full quadtree.
- The aggregation coefficient $agg(I)$ of an image I. This quantity has been defined and studied in [10] and expresses the coherence of unicolor regions of the image. Starting from a random image and using the algorithm presented in [10], an image with the same black/white analogy and higher aggregation (more realistic) can be created.
- The image difference, the percentage of pixels changing value from the first image to the consecutive one (from 2% to 10%). Note that the random changing of single pixels is an extreme method of producing evolving images and the results produced by this policy should be seen as very pessimistic. In practice, much higher storage gains are expected.

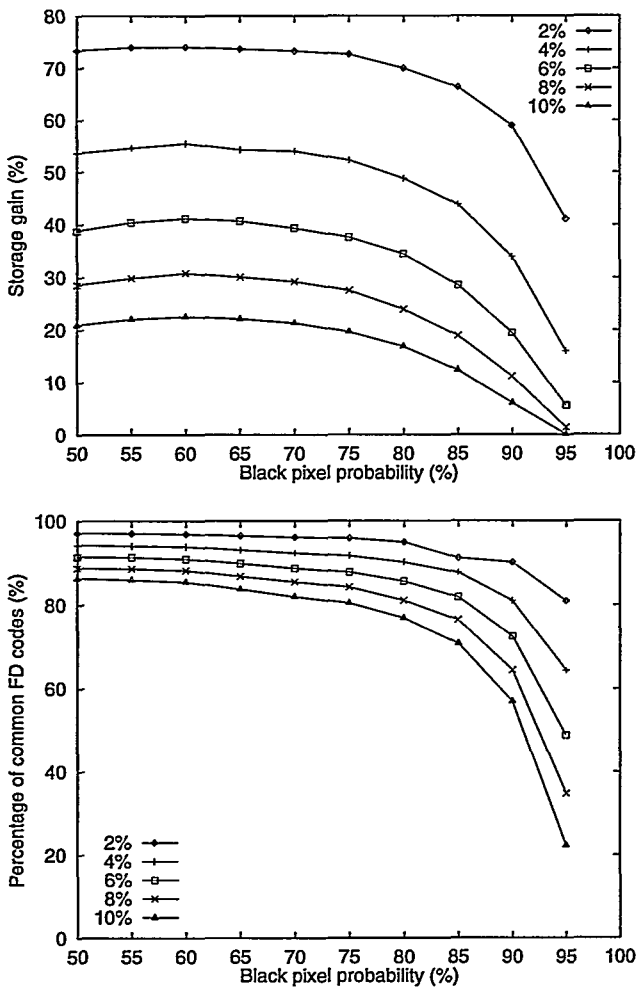

Figure 4: The storage gain and the percentage of common FD codes (obtained during the first experiment) as a function of black probability, for various change percentages.

First experiment. This experiment concerned random images of various black/white analogies, the aggregation of which remained unchanged. Each image was randomly changed and overlapped with its changed version. The upper part of Figure 4 depicts the average storage gain as a function of black/white analogy for the first image of each pair, for various change percentages. The lower part of this figure depicts the average percentage of common FD codes between the images of each pair. In most cases, the storage gain is significant and varies between 20% and over 70%, according to the change probability. The abrupt decrease of the common disk space after the 85% black percentage is explained by the fact that images with 85% black pixels and higher, form many large and solid black spatial regions ("islands"). Thus, when we change the color of the 2% (for instance) of the pixels of this image in order to produce the consecutive image, many of these islands are fragmented, producing different quadrants. It is evident that all the curves of Figure 4 converge to point (100,0).

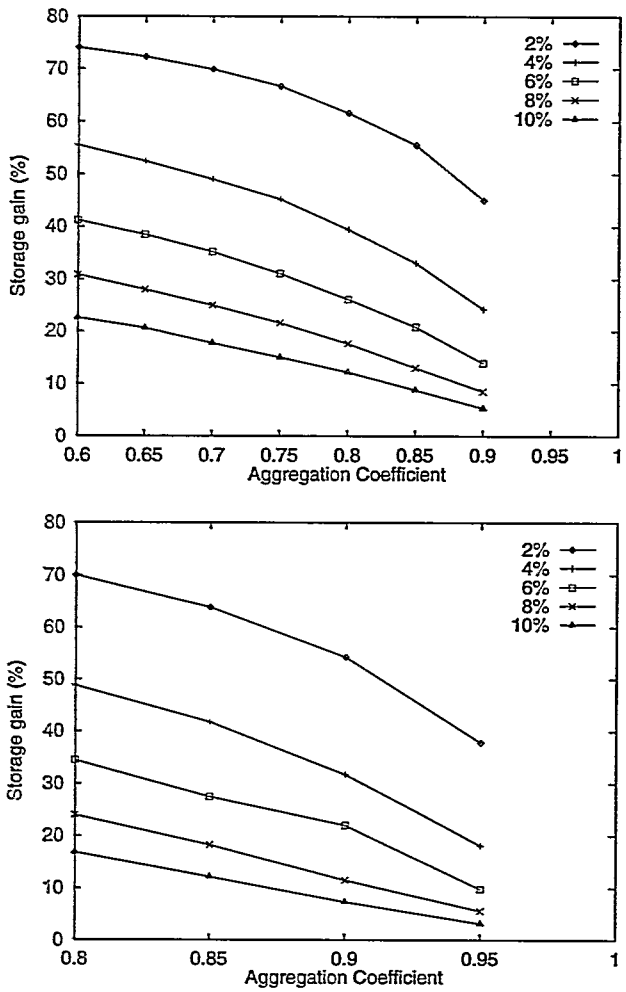

Figure 5: The storage gain as a function of aggregation (obtained during the second experiment), for various change percentages. The images were 60% black (above) and 80% black (below).

Second experiment. This experiment concerned random images of various black/white analogies, the aggregation of which was increased at various amounts. Then, each image was randomly changed and overlapped with its changed version. Since a large amount of results was produced, Figure 5 depicts for two cases only (for images being 60% black, above and 80% black, below) the storage gain as a function of aggregation for various change percentages.

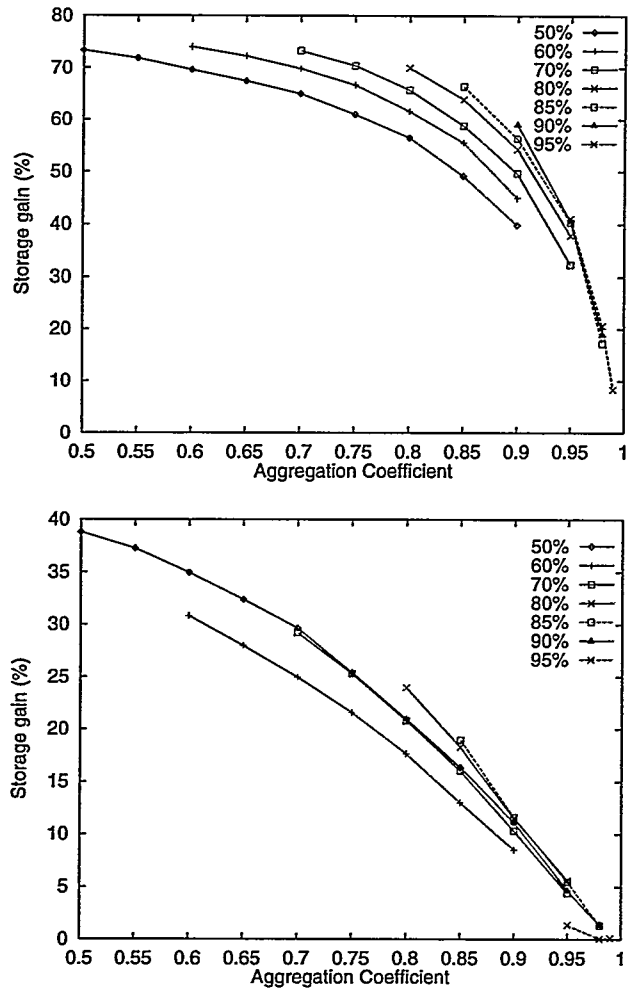

Figure 6: The storage gain as a function of aggregation (obtained during the second experiment), for various black probabilities. The change percentage was 2% (above) and 8% (below).

Figure 6 was based on the data produced by the same experiment. The difference in this figure is that in each plotted line the black/white analogy was stable (while in Figure 5 the change percentage was stable). Again, for two cases only (for change percentage equal to 2%, above and to 8%, below) the storage gain as a function of aggregation for various black probabilities is depicted. In this experiment, the storage gain is significant also and, in most cases, varies between 15% and 70%, according to the change probability. The decrease of the percentage of the common disk-space inversely to the increase of the coefficient of the aggregation took place, for the same reason that was described in the first experiment.

5 Conclusions

In the present report, we presented a new structure: Overlapping Linear Quadtrees, which may be used as an access mechanism for consecutive transaction-time raster images. Experimentation with synthetic region data revealed that considerable storage is saved with respect to the case of independent linear quadtrees when used to store similar consecutive images. An efficient algorithm that takes advantage of the new structure and answers spatio-temporal window queries was also presented. Therefore, this structure can be used in spatio-temporal databases to support query processing of evolving images.

In the future, we plan to examine the structure of multiversion B-Trees [1] by accommodating quadcodes instead of ordinary numeric data, and pursue an experimental performance comparison for these compatible methods (compatible in the sense that they do not follow the "conservative approximation principle", but use instead quadcodes to decompose and store image data). Also, important is to examine the performance of Overlapping Linear Quadtrees in the context of various spatio-temporal operations, such as spatio-temporal joins, as well as spatio-temporal nearest neighbor queries [17].

Acknowledgments

The authors would like to thank Alex Nanopoulos for his useful comments during the development of this work. The second author, who is a Post-doctoral Scholar of the State Scholarship Foundation of Greece, would like to thank this foundation for its financial assistance.

References

[1] B. Becker, S. Gschwind, T. Ohler, B. Seeger and P. Widmayer: "An Asymptotically Optimal Multiversion B-tree", *The VLDB Journal*, Vol.5, No.4, pp.264-275, 1996.

[2] F.W. Burton, M.W. Huntbach and J. Kollias: "Multiple Generation Text Files Using Overlapping Tree Structures", *The Computer Journal*, Vol.28, No.4, pp.414-416, 1985.

[3] F.W. Burton, J.G. Kollias, V.G. Kollias and D.G. Matsakis: "Implementation of Overlapping B-trees for Time and Space Efficient Representation of Collection of Similar Files", *The Computer Journal*, Vol.33, No.3, pp.279-280, 1990.

[4] V. Gaede and O. Guenther: "Multidimensional Access Methods", *ACM Computer Surveys*, to appear. Address for downloading: http://www.wiwi.hu-berlin.de/~gaede/survey.rev.ps.Z.

[5] A. Guttman: "R-trees - a Dynamic Index Structure for Spatial Searching", *Proceedings of the ACM SIGMOD Conference*, pp.47-57, Boston MA, 1984.

[6] C.S. Jensen, J. Clifford, R. Elmasri, S.K. Gadia, P. Hayes and S. Jajodia (ed.): "A Consensus Glossary of Temporal Database Concepts", *ACM SIGMOD Record*, Vol.23, No.1, pp.52-64, 1994.

[7] S.D. Lang and J.R. Driscoll: "Improving the Differential File Technique via Batch Operations for Tree Structured File Organizations", *Proceedings of the IEEE International Conference on Data Engineering*, 1986.

[8] D. Lomet and B. Saltzberg: "Access Methods for Multiversion Data", *Proceedings of ACM SIGMOD Conference*, pp.315-324, Portland OR, 1989.

[9] Y. Manolopoulos and G. Kapetanakis: "Overlapping B+-trees for Temporal Data", *Proceedings of the 5th Jerusalem Conference on Information Technology (JCIT)*, pp.491-498, Jerusalem, Israel, 1990.

[10] Y. Manolopoulos, E. Nardelli, G. Proietti and M. Vassilakopoulos: "On the Generation of Aggregated Random Spatial Regions", *Proceedings of the 4th International Conference on Information and Knowledge Management (CIKM)*, pp.318-325, Washington DC, 1995.

[11] M. Nascimento and M. Eich: "An Introductory Survey to Indexing Techniques for Temporal Databases", Southern Methodist University, Technical Report, 1995.

[12] M.A. Nascimento and J.R.O. Silva: "Towards Historical R-trees", *Proceedings of ACM Symposium on Applied Computing (ACM-SAC)*, 1998.

[13] B. Saltzberg and V. Tsotras: "A Comparison of Access Methods for Time Evolving Data", *ACM Computing Surveys*, to appear. Address for downloading: ftp://ftp.ccs.neu.edu/pub/people/salzberg/tempsurvey.ps.gz.

[14] H. Samet: "The Design and Analysis of Spatial Data Structures", *Addison-Wesley*, Reading MA, 1990.

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

[16] Y. Theodoridis, M. Vazirgiannis and T. Sellis: "Spatio-Temporal Indexing for Large Multimedia Applications", *Proceedings of the 3rd IEEE Conference on Multimedia Computing and Systems (ICMCS)*, 1996.

[17] Y. Theodoridis, T. Sellis, A. Papadopoulos and Y. Manolopoulos: "Specifications for Efficient Indexing in Spatiotemporal Databases", *Proceedings of the 7th Conference on Statistical and Scientific Database Management Systems (SSDBM)*, pp.123-132, Capri, Italy, 1998.

[18] T. Tzouramanis, Y. Manolopoulos and N. Lorentzos: "Overlapping B+trees - an Implementation of a Transaction Time Access Method", submitted.

[19] M. Vassilakopoulos, Y. Manolopoulos and K. Economou: "Overlapping for the Representation of Similar Images", *Image and Vision Computing*, Vol.11, No.5, pp.257-262, 1993.

[20] M. Vassilakopoulos, Y. Manolopoulos and B. Kroell: "Efficiency Analysis of Overlapped Quadtrees", *Nordic Journal of Computing*, Vol.2, pp.70-84, 1995.

[21] M.F. Worboys: "A Unified Model for Spatial and Temporal Information", *The Computer Journal*, Vol.37, No.1, pp.26-34, 1994.

[22] X. Xu, J. Han, and W. Lu: "RT-tree - an Improved R-tree Index Structure for Spatiotemporal Databases", *Proceedings of the 4th International Symposium on Spatial Data Handling (SDH)*, 1990.