STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LII**, Number 2, 2007

# INDEXING THE EVOLUTION OF MOVING OBJECTS WITHIN A 2D SPACE USING THE BRICKR STRUCTURES

ANDREEA SABAU

Abstract. A growing number of applications manage mobile objects. The storage and the organization within databases of data describing the evolution of these objects is an open challenge. Data must be managed in efficient structures with respect to both the storage space consumed and the data access through these structures. An indexing method that organizes the evolutions of spatial objects within a 2D space is proposed in this paper. The Dynamic-BrickR access method uses two structures: an underlying permanent R*-Tree structure, and an in-memory dynamic space grid structure, that it used for building the terminal nodes to feed the R*-Tree. Experiments show significant improvements of the Dynamic-BrickR method over the R*-Tree index, regarding the dead space and the overlapping volumes. The Dynamic-BrickR inherits from the R*-Tree the capability to be used in answering spatial, temporal and spatio-temporal queries.

## 1. Introduction

The organization and management of spatio-temporal objects required lately a lot of attention. The growing interest in this area is justified by the proliferation of a large range of applications that can benefit of efficient management methods for spatio-temporal data. Communication and localization in mobile objects networks is one such application domain.

The spatio-temporal data represents the evolution of spatial objects in time. A spatial attribute with discretely or continuously evolving values may represent the shape and / or the location of an object. For example, land parcels positions and extents evolve discretely in time; while the cars on a road are continuously changing their position, but not the shape. The most

Received by the editors: November 1, 2007.

2000 *Mathematics Subject Classification.* 68P05, 68P20.

1998 *CR Categories and Descriptors.* H.2.2 [**Information Systems**]: Database Management – *Physical Design*; H.2.4 [**Information Systems**]: Database Management – *Systems*; H.2.8 [**Information Systems**]: Database Management – *Database Applications* .

significant challenge in the management of spatio-temporal objects is to efficiently organize information about the continuous change in time of their spatial features values

The extended version of the Dynamic-BrickR access method for indexing 3D spatio-temporal data is presented in this paper. The described method is based on the previous work presented in [9, 10]. Dynamic-BrickR organizes the continuous evolutions of spatial objects within a 2D spatial domain. These objects shape is not relevant and they are represented as points (or no extent) objects. A real-world example of such objects is that of the cars on a network of roads that are moving with different speeds, in any way.

Related Work. Many researchers have worked on organizing spatio-temporal objects in index structures. These structures may be classified as structures that index: past data, present and past data or data about present and future.

Several index structures, such as STR-Tree [7] and SETI [4] organize past information. The STR-Tree is an R-Tree like structure that attempts to achieve trajectory preservation for each object by storing its trajectory segments in the same tree node. SETI divides the spatial domain into a static partition and the data corresponding to one cell of the partition are organized into an R-Tree.

There are many spatio-temporal access methods that manage present (and past) data. The 2-3 TR-Tree [1] is an index structure that contains two R-Trees: one tree for points that represent present data and another R-Tree for the trajectories from the past; the search might have to consult one or both R-Trees. The MR-Tree [13] and the HR+-Tree [11] are overlapping R-Trees, where a node may have one or more parents. The LUR-Tree [5] indexes only current positions of objects, so historical queries are not supported.

PR-Tree [3], STAR-Tree [8], and TPR*-Tree [12] belong to a class of spatio-temporal access methods, which manage present data and data for prediction of future movement.

The paper is organized as follows. The Dynamic-BrickR access method, its extended structures and the management of spatio-temporal data are described in Section 2. Section 3 presents the comparative results for Dynamic-BrickR, BrickR and R*-Tree [2] methods in organizing and querying spatio-temporal data. The paper ends with conclusions and future work.

## 2. The 2D Dynamic-BrickR Spatio-Temporal Access Method

This section presents the structures and the involved management operations corresponding to the Dynamic-BrickR access method. These structures represent the upgrade applied on the 1D Dynamic BrickR [9, 10] access method in order to index the continuous evolutions of mobile objects within a 2D space.

The Dynamic-BrickR access method uses a temporary structure and a permanent structure in order to index efficiently the recently received and past data. The permanent physical structure is designed as an R*-Tree and it is called DBR-Tree. The temporary structure, called DBR-Grid, is an in-memory space grid, that it used for building terminal tree nodes to feed the R*-Tree. The dimensionality of the grid corresponds to the space dimensionality of the indexed objects. In the particular studied case, the grid is 2D. The grid has a dynamic evolution, as the strips it is composed of are split or merged to best adjust to the objects evolution in time. The name of the Dynamic-BrickR access method originates in the visual aspect the grid gives to the indexed space, which resembles a brick wall; it also reflects the grid dynamic behavior and the fact it uses an R*-Tree permanent structure.

As it was mentioned before, the structures of the Dynamic-BrickR indexing method organizes data corresponding to the spatial evolutions of objects in time. It is considered that the objects shape is not relevant (it is not changing in time and it is not significant). Therefore, the objects are modeled as points within the 2D spatial domain.

The spatial domain is considered to be (relatively) constant in time, without affecting the organization rules and the generality of the indexing method. If a set of objects moved beyond the initially spatial domain borders, the working space would be extended in a straightforward fashion. In order to facilitate the generation and the visualization of data, the spatial domain is considered to be $[0, L] \times [0, L]$. The temporal domain is considered to be isomorphic to the set of real numbers. Therefore, the temporal domain is treated as an auxiliary domain to the spatial one.

Regarding the indexed spatio-temporal data, usually the mobile objects are moving within the spatial domain with a variable speed. In order to facilitate the representation of their trajectories, the speed of an object is considered to be constant during a certain time interval. Therefore, the trajectory on that time interval is approximated by a linear function of time and it is represented geometrically as a 3D line segment (also called trajectory segment). The set of trajectory segments corresponding to a mobile object represents the trajectory of that object, or its spatial evolution in time.

Theoretically, there is no restriction on the manner the mobile objects are sending the data to the system. The mobile objects can be equipped with some GPS devices and can send the positioning information with regularity or when the value of some parameter of movement (the direction and / or the speed) is changed. Furthermore, the received information can be a spatio-temporal point (the new position at a certain time instant) or a spatio-temporal line segment (the trajectory segment corresponding to a certain time interval). However, at physical level, the received information is stored as linear functions of time (as 3D spatio-temporal line segments).

The Dynamic-BrickR access method contains two sub-structures [9, 10]:

- A temporary structure,
- A persistent structure.

The permanent structure in the Dynamic-BrickR method, called the DBR-Tree, is essentially an R*-Tree structure used to index spatio-temporal data (3D trajectory segments) from the remote past. It is assumed to be stored in secondary memory; therefore the paginated storage of the nodes is facilitated.

The temporary structure, called the DBR-Grid, is a grid structure that indexes the newest spatio-temporal data received by the system. This structure is stored in the main memory, having the advantage of a short data access time and efficient operations.

Regarding the receiving of data, it is natural to consider that data about an object trajectory arrives in ascending order of the timestamps of the trajectory segments end points.

As it was described in [9, 10], the temporary structure is designed to create an extra "thinking" moment of how to group trajectory segments into MBRs. The main idea is not to insert a segment into the main R*-Tree structure as soon as it is received by the system. The most recent segments are kept in memory and clustered in in-memory tree nodes, which will be entirely inserted in the tree afterwards. This way, there are almost void chances to get overlapping areas between the resulting leaf nodes MBRs and this conducts to smaller overlapping at superior levels.

As in the 1D Dynamic-BrickR structures [9, 10], the DBR-Grid is obtained by partitioning the 3D spatio-temporal domain using two sets of hyper-planes parallel with the temporal axis (Ot). One set's hyper-planes are also parallel to the Oy axis, and perpendicular on the Ox axis, and the hyper-planes of the other set are parallel with the Ox axis, and perpendicular on the Oy axis. The initial number of hyper-planes and their positions are set at the beginning of the grid's construction. The partitioning of the spatial domain is accomplished by using (initially) equidistant hyper-planes on each of the two spatial axes. The pieces of the DBR-Grid are 3D orthogonal polyedra and, as in the case of 1D Dynamic-BrickR structures, these elements are called strips.

It is also initially considered that the number of hyper-planes on the Ox axis (*nbx*) is equal to the number of divisions on the Oy axis (*nby*). This number is noted here *nb*. Therefore, the initial number of grid strips is $nb^2$. Let the DBR-Grid obtained in this manner be built by the strips $B_{ij}$, i, j:=1..*nb*. The length of the spatial projection of a strip on the Ox and Oy axis is initially

$\Delta b = lS/nb$. The coordinates of the partitioning hyper-planes are given by the two arrays XD = ($xd_0$, $xd_1$, ..., $xd_{nb}$), and YD = ($yd_0$, $yd_1$, ..., $yd_{nb}$), respectively, where $xd_k = k * \Delta b$ and $yd_k = k * \Delta b$, k:=0..$nb$. Let *nSegm_ij* be the number of trajectory segments contained within a grid strip $B_{ij}$, and $SB_k^{ij}$ the segments included in $B_{ij}$, i, j:=1..$nb$, k:=1..*nSegm_ij*. The initial configuration of the DBR-Grid is considered in Fig. 1.

According to these observations, the partitioning hyper-planes initially determine on the spatial domain (xOy) a 2D rectangular regular grid (see the initial partitioning of the spatial domain xOy in Fig. 1). Because of the grid's dynamicity, the spatial partitioning may become non-regular by performing some division and / or merge operations (see the Examples 1 and 2). Furthermore, the numbers of partitioning hyper-planes on the two spatial axes may differ in time ($nbx \neq nby$).

As in the 1D case, the trajectory segments are first inserted in the DBR-Grid. One segment may be clipped according to the grid strips it intersects, and then the resulted sub-segments are inserted in the corresponding strips of the DBR-Grid. In other words, if a segment intersects two or more strips, it is divided into pieces, each piece being totally enclosed within a strip. Fig. 1 shows the result of a fragmentation operation performed on a trajectory segment, having as result three sub-segments.

The MBR of a 3D strip $B_{ij}$ is given by the points ($x_1^{MBR}$, $y_1^{MBR}$, $t_1^{MBR}$) and ($x_2^{MBR}$, $y_2^{MBR}$, $t_2^{MBR}$), where $x_1^{MBR} = xd_{i-1}$, $x_2^{MBR} = xd_i$, $y_1^{MBR} = yd_{j-1}$, $y_2^{MBR} = yd_j$, $t_1^{MBR} = \min\{SB_k^{ij}.t_1|$ k:=1..*nSegm_ij*$\}$, and $t_2^{MBR} = \max\{SB_k^{ij}.t_2|$ k:=1..*nSegm_ij*$\}$.

The modification of the `Insert_segment` and `Cut_MBR` algorithms corresponding to the 2D DBR-Grid so that to manage 3D trajectory segments is straightforward. The major difference between the 2D DBR-Grid and the 3D DBR-Grid is found at operational level (the management of the partitioning hyper-planes, the division and merge operations). Furthermore, an object $B$ of type *Strip* [10] is characterized by its spatial borders, given by *B.xmin*, *B.xmax*, *B.ymin*, *B.ymax*, and an object $g$ of type DBR-Grid is a 2D array with elements of type *Strip*. The structure of $g$ is determined by the XD and YD arrays: $nbx$ is the number of rows of $g$, $nby$ gives the number of columns of $g$, and $g[i][j]$ represents the strip that contains a set of SO segments, so that $SO.x_1$, $SO.x_2 \in [xd_{i-1}, xd_i)$ and $SO.y_1$, $SO.y_2 \in [yd_{j-1}, yd_j)$, i:=0..$nbx$, j:=0..$nby$.

It can be observed that the partioning determined by the object $g$ on the spatial domain does not concur with the physical delimitation of the strips. Therefore, it have to be mentioned that $g$ determines a *virtual partitioning* of the spatial domain, and the borders of the strips (the hyper-planes of XD and YD of which coordinates are found within the strips member data *xmin*, *xmax*, *ymin*, *ymax*) determine the *physical partitioning* of the spatial domain.

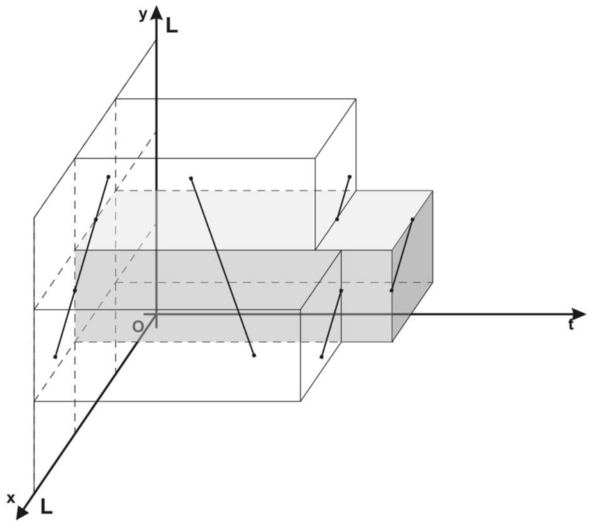

FIGURE 1. The fragmentation of the trajectory segments within the strips of the DBR-Grid. The graphical representation shows the inserted segment, its projection on the xOy plane, and the spatial projections of the obtained three subsegments.

Two aspects have to be mentioned in order to understand the division algorithm that is performed on a 3D strip:

(1) The division is performed on a single spatial dimension.
(2) The division is physically affecting a single strip.

The second observation is related to the fact that choosing a new partitioning hyper-plane *h_div* for a strip *B'* virtually affects all the strips intersected by it, unlike the division of a 2D strip. But it is unlikely that another strip than *B'* needs to be divided in that moment. Therefore, only the strip *B'* is physically affected by the division by *h_div*. If another strip *B"* needs later to be divided and *h_div* intersects *B"*, then the hyper-plan *h_div* is considered as candidate in performing the division of *B"*. The algorithm by which the partitioning hyper-plan is chosen is described next (`ChooseDivision`).

**ChooseDivision(B, D, h_div)**

```
// Input:
//   B - the strip of the DBR-grid which has to be divided
// Output:
//   D - the spatial dimension on which the division will
//     be performed
//   h_div - the partitioning hyper-plane on D dimension

// The ChooseDivision routine determines the dimension D
// on which the division will be performed and the
// position of the partitioning hyper-plane $h\_div$

Let R be the projection of B on the spatial domain
Let $dx$ and $dy$ the projections of R on the Ox, and Oy,
respectively
If $dx \leq dy$ then D := Ox
Else D := Oy
End if
If D = Ox then
   If $\exists xd_i \in XD$ such as $B.xmin < xd_i < B.xmax$ then
      $h\_div$ := $xd_i^*$, where $xd_i^* \in XD$, $B.xmin < xd_i^* < B.xmax$,
                  $|xd_i^* - (B.xmin + B.xmax)/2| =$
                  $min\{xd_i - (B.xmin + B.xmax)/2 | xd_i \in XD\}$
      // If there exists at least one hyper-plan in XD
      // that intersects B, then $h\_div$ is chosen so that
      // to be the closest to the median of R on Ox
   Else
      Choose_x_division(b, $h\_div$)
      // The routine is similar to the Choose_x_division
      // algorithm presented in [10]
   End if
Else // D = Oy
   Choose $h\_div$ on the Oy axis as it was chosen in the
   first case
End if
End ChooseDivision
```

The following two examples show the updates on the *DBR_Grid* object ($g$) during the division and merge operations. The notation $B = (g[i][j] |$ i:=1..$nbx$, j:=1..$nby$) is used in order to specify all the g's elements that refer the strip $B$.

Example 1. Let L = 100, $nbx$ = $nby$ = 4, XD = YD = (0, 25, 50, 75, 100) be the initial configuration data of the DBR-Grid $g$. The initial virtual partitioning concure with the physical partitioning (see Fig. 2(a)). Let $x\_div$ = 35 be the position of the division hyper-plane used to divide the strip $B = (g[2][2])$ (the marked cell).

Fig. 2(b) shows the configuration of the spatial projections of the strips, the physical and the virtual partitioning, after the strip has been divided. It can be noticed that the strips that are intersected by $x\_div$ are not physically divided; they preserve the position and the content, but they are referred by two elements of the virtual grid. For example, the strip $B' = (g[2][1])$ (see Fig. 2(a)) is given next by $B' = (g[2][1], g[3][1])$. The virtual partitioning lines that are not part of the physical partitioning are represented by dotted lines.

Later, the coordinate of a new partitioning hyper-plan $y\_div = 60$ on Oy axis is determined in order to divide the strip $B = (g[4][3])$. After updating the object $g$, it can be observed that all the strips referred by elements on the $4^{th}$ row (with the exception of $g[4][3]$) are now referred by one more element (see Fig. 2(c)). For example, the strip $B' = (g[2][3], g[3][3])$ from Fig. 2(b) is now referred as $B' = (g[2][3], g[3][3], g[2][4], g[3][4])$. Fig. 2(d) represents the configurations of the physical and virtual partitions after a division by $x\_div = 65$ have been performed. It can be observed that the shape of the spatial projections of the grid's strips does not depend on the position, dimension or the order in which the divisions are performed; their shape is continuously rectangular.

Corresponding to the definition of neighbor strips given in [10], two 3D strips, $B_1$ and $B_2$, are considered to be neighbors if

$$(B_1.xmax = B_2.xmin \text{ or } B_1.xmin = B_2.xmax) \text{ or}$$
$$(B_1.xmax = B_2.xmin \text{ or } B_1.ymin = B_2.ymax).$$

Also similar to the measure of density defined in [10], the density of segments within an object R (strip or MBR of a tree node) is given by

$$C(R) = nSegm * dx * dy/dt,$$

where M denotes the maximum capacity of a tree node.

The merging technique is related to the manner in which the virtual partitioning was managed during the division operations. Another condition that must be fulfilled by two neighbor strips, $B_1$ and $B_2$, is that their spatial projections on the dimension complementary to the dimension on which the strips are neighbors to concur (for example, if the dimension on which the two strips are neighbors is Oy, then $B_1.xmin = B_2.xmin$ and $B_1.xmax = B_2.xmax$). Then, among all the candidate neighbor strips, it is selected the pair of strips which minimizes the sum of their densities of segments. For example, it is considered the configuration represented in Fig. 2(d): let $B_1 = (g[5][4])$ and $B_2 = (g[6][3], g[6][4])$ be two neighbor strips; these two strips cannot be merged because their spatial projections on the Oy axis does not concur ($[60, 75) \neq [50, 75)$).

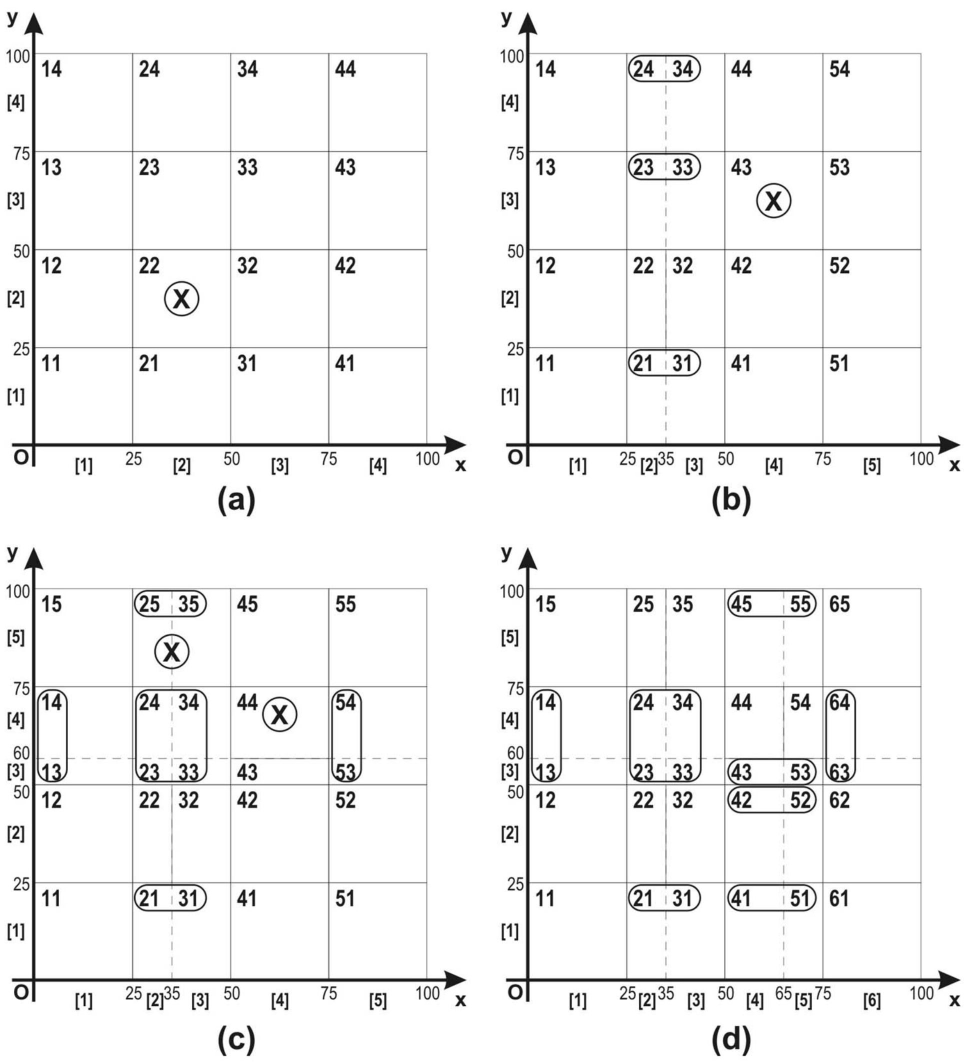

FIGURE 2. The updates performed on the DBR-Grid during the division operations presented in Example 1.

The manner in which the merge of two neighbor strips is performed is presented by Example 2.

Example 2. It is considered the configuration represented in Fig. 2(d). The first merge operation is performed on the strips $B_1 = (g[1][1])$ and $B_2 = (g[1][2])$. Because the hyper-plan $x\_div = 25$ physically delimits other strips, the reunion affects only the $B_1$ and $B_2$ strips, and does not affect the virtual partitioning. A single physical strip $B = (g[1][1], g[1][2])$ results after performing the merge operation (see Fig. 3(a)).

Next, the reunion of strips $B_1 = (g[1][3], g[1][4])$ and $B_2 = (g[2][3], g[3][3], g[2][4], g[3][4])$ is considered. The result of the reunion is given by the strip $B = (g[1][3], g[2][3], g[3][3], g[1][4], g[2][4], g[3][4])$ (see Fig. 3(b)). Fig. 3(c) depicts the spatial configuration obtained by merging the strips $B_1 = (g[2][2])$ and $B_2 = (g[3][2])$. Because the hyper-plane $x\_div = 35$ does no delimit two physical strips, the last merge operation also affects the structure of the virtual grid by eliminating this hyper-plane.

## 3. Experimental Results

This section presents the comparative results obtained for three access methods: R*-Tree method, BrickR method - similar to Dynamic-BrickR, does not perform strips merging and the Dynamic-BrickR method. These three methods are evaluated in respect to the quality of data organization and the efficiency of answering queries using the corresponding built indexes.

Tests have been run on three synthetic data sets of 3D trajectory segments, which were constructed using sets of points associated to the mobile objects. Each test data set numbers 10000 trajectory segments, recorded for 100 mobile objects. The coordinates (x, y, t) of the data points belong to a well-determined spatio-temporal working interval: the temporal domain was [1, 50000], and the spatial domain was $[1..1000] \times [1..1000]$. The three test data sets were constructed from: points randomly generated (with uniform distribution), points following a Gaussian distribution and points following a Poisson distribution. For each of the three methods, the capacity of a tree node was set to 25 - considering the size of a disk page equal to 512B. The minimum occupancy of a tree node was set to 40

Regarding the data organization, the measures evaluated on the permanent structures built by the three compared access methods are: the degree of tree nodes occupation, the sum of nodes MBRs areas, the sum of nodes MBRs volumes and the total value of the overlapping volumes between the nodes on the same tree level.

Even if the number of indexed segments by the Dynamic-BrickR permanent structure is greater than the initial number of segments, due to the clipping method, the tests show a better node occupation than in the case of R*-Tree: on the average, the node occupancy in the R*-Tree is 54.15% and in the BrickR and Dynamic-BrickR is 95.27% and 92.76% respectively.

Fig. 4(a) and Fig. 4(b) comparatively present for each sort of data sets the sums of the MBRs area and the sums of the MBRs volumes, and Fig.

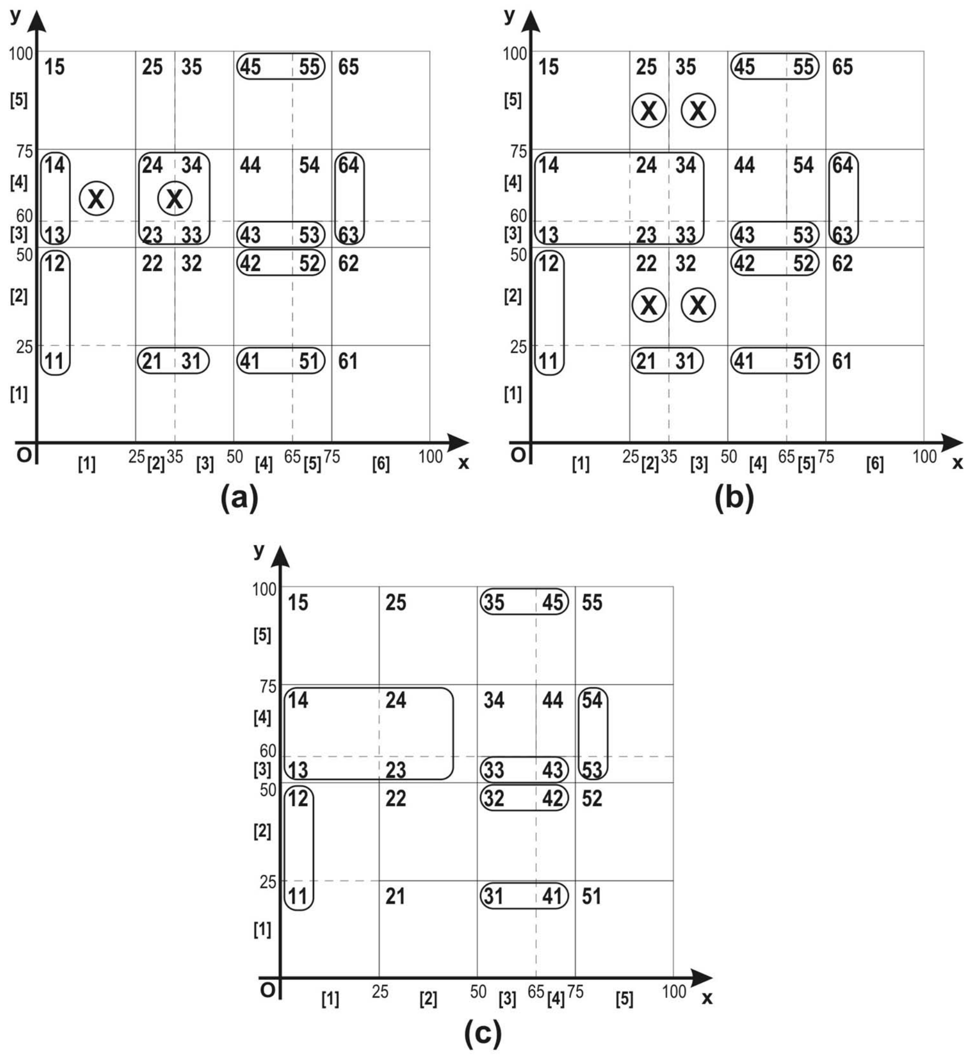

FIGURE 3. The updates performed on the DBR-Grid during the merge operations presented in Example 2.

5 shows the obtained results regarding the overlapping volumes between the nodes on the same tree level. It can be noticed that with only one exception in the case of data sets with Gaussian distribution in evaluating the MBRs areas, the obtained results were better (smaller) for the BrickR methods than for the R*-Tree method.

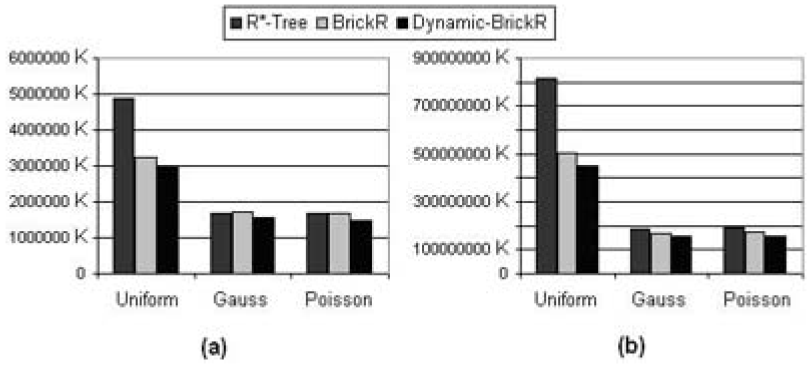

FIGURE 4. (a) The sum of MBRs areas and (b) the sum of MBRs volumes, for the R* Tree, BrickR and Dynamic BrickR methods

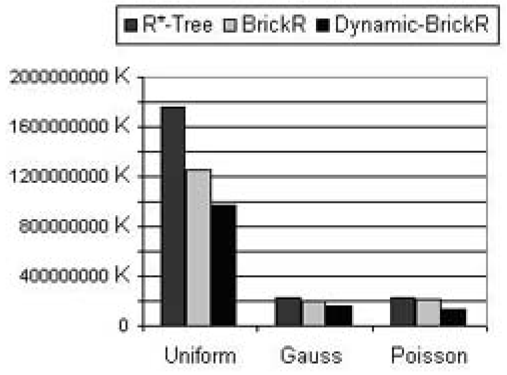

FIGURE 5. Overlapping volumes between the MBRs of nodes on the same tree levels, for all tree nodes, for the R* Tree, BrickR and Dynamic BrickR methods

The queries performed on spatio-temporal data may be classified as: spatial queries, temporal queries and spatio-temporal queries. The types of queries for which tests have been performed consist of combinations of spatial and / or temporal, point and / or window queries, denoted by:

- S-P T-P query: spatial-point temporal-point query;
- S-W query: spatial-window query;
- T-W query: temporal-window query;
- S-W T-W query: spatial-window temporal-window query;
- S-W T-P query: spatial-window temporal-point query;

- S-P T-W query spatial-point temporal-window query.

It can be observed that the S-P T-P, S-W T-P, and S-P T-W types of queries are particular cases of the S-W T-W queries.

The data within query sets follow a uniform distribution on the spatio-temporal working space, and cover the whole working space. On the other hand, two sorts of query sets have been used in the case of window or point-window queries, in accordance with the maximum length of generated intervals: the spatial and temporal length of the query windows represent maximum 25

In all experiments, the average number of tree nodes visited for answering the query was measured. BrickR and Dynamic-BrickR methods obtained better results on uniformly distributed data sets, except in the case of T-W queries. The execution of the T-W queries showed a better performance for the R*-Tree. In the other cases the obtained results for the three compared access methods were relatively closed.

## 4. Conclusions and Future Work

The Dynamic-BrickR access method that indexes 3D spatio-temporal data preserves the advantages of the previous version of it:

- The trajectory segments are grouped within the grid structure and sent as a node (completely built) in the tree structure. Thus, the number of performed I/O operations is reduced.
- The chances to obtain overlapping volumes between the MBRs of DBR-Tree's terminal nodes are minimized.;
- The inserted terminal nodes are almost fully occupied.

The proposed future work includes the development of a BrickR-like structure for organizing the continuous evolutions of objects with shape. On the other side, the re-organization of the BrickR permanent structure is proposed as future work, so that to limit the number of tree levels by temporal fragmenting of the node set. This way, the enlargement of the indexed data set does not affect in a major way the performance of the structure.

## References

[1] M. Abdelguerfi, J. Givaudan, K. Shaw, R. Ladner, *The 2-3 TR-tree, A Trajectory-Oriented Index Structure for Fully Evolving Valid-time Spatio-temporal Datasets*, In Proc. of the ACM Workshop on Adv. in Geographic Information Systems, ACM GIS, 29-34, 2002.

[2] N. Beckmann, H. P. Kriegel, R. Schneider, B. Seeger, *The R*-tree: An Efficient and Robust Access Method for Points and Rectangles*, In Proc. of the Intl. Conf. on Management of Data, SIGMOD, 322-331, 1990.

[3] M. Cai, P. Revesz, *Parametric R-Tree: An Index Structure for Moving Objects*, In Proc. of the Intl. Conf. on Management of Data, COMAD, 57-64, 2000.

[4] V. P. Chakka, A. Everspaugh, J. M. Patel, *Indexing Large Trajectory Data with SETI*, In Proc. of the Conf. on Innovative Data Systems Research, CIDR, 164-175, 2003.

[5] D. Kwon, Sj. Lee, S. Lee, *Indexing the Current Positions of Moving Objects Using the Lazy Update R-tree*, In Mobile Data Management, MDM, 113-120, 2002.

[6] M. A. Nascimento, J. R. O. Silva, Y. Theodoridis, *Evaluation of Access Structures for Discretely Moving Points*, In Proc. of the Intl. Workshop on Spatio-Temporal Database Management, STDBM, 171-188, 1999.

[7] D. Pfoser, C. S. Jensen, Y. Theodoridis, *Novel Approaches in Query Processing for Moving Object Trajectories*, In Proc. of the Intl. Conf. on Very Large Data Bases, VLDB, 395-406, 2000.

[8] C. M. Procopiuc, P. K. Agarwal, S. Har-Peled, *STAR-Tree: An Efficient Self-Adjusting Index for Moving Objects*, In Proc. of the Workshop on Alg. Eng. and Experimentation, ALENEX, 178-193, 2002.

[9] A. Sabau, *Indexing Mobile Objects Using BrickR Structures*, In Studia Universitatis Babes-Bolyai, Informatica, Vol. LI(2), 71-80, 2006.

[10] A. Sabau, A. Campan, *BrickR: The Dynamic-BrickR Access Method for Mobile Objects*, Proc. of the 22nd International Symposium on Computer and Information Sciences, Turkey, 2007.

[11] Y. Tao, D. Papadias, *Efficient Historical R-trees*, In Proc. of the Intl. Conf. On Scientific and Statistical Database Management, SSDBM, 223-232, 2001.

[12] Y. Tao, D. Papadias, J. Sun, *The TPR*-Tree: An Optimized Spatio-temporal Access Method for Predictive Queries*, In Proc. of the Intl. Conf. on Very Large Data Bases, VLDB, 790-801, 2003.

[13] X. Xu, J. Han, W. Lu, *RT-Tree: An Improved R-Tree Indexing Structure for Temporal Spatial Databases*, In Proc. of the Intl. Symp. on Spatial Data Handling, SDH, 1040-1049, 1990.

Faculty of Mathematics and Computer Science, Babes-Bolyai University, Cluj-Napoca, Romania

*E-mail address*: `deiush@cs.ubbcluj.ro`