STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LI**, Number 2, 2006

INDEXING MOBILE OBJECTS USING BRICKR STRUCTURES

ANDREEA SABĂU

Abstract. A growing number of applications manage mobile objects. The storage and the organization of data within databases is an open challenge. A new indexing method is proposed in this paper. The BrickR structure organizes continuously evolving objects with no extent using two structures: a temporary structure that manages recent data and a permanent structure that stores data belonging to the past. This access method treats identically the spatial and temporal dimensions, thus allowing spatial, temporal and spatio-temporal queries to be answered.

1. Introduction

There is a tendency today to study systems for management of objects with spatial evolution in time, particularly for management of mobile objects. These objects are also known as spatio-temporal objects.

The spatial attributes of which values have evolution in time can represent the shape and / or the location of objects, and these evolutions may be discrete or continuous. For example: land parcels have a discrete evolution in time of the positions and extents; the cars on a road are continuously changing their position, but not the shape. The most significant challenge is to store data about continuous changing.

A new spatio-temporal access method called BrickR is proposed in this paper. The spatial objects that BrickR is organizing are point objects (objects with no extent) with continuous spatial evolution. Such kind of object is a car on the road. This car might move with a variable speed and along the both senses.

Two sub-structures work together so as to manage efficiently the recently received and past data: a permanent physical structure designed as an R*-tree [2] and a temporary structure designed as a grid which evolves in time like a brick wall.

Received by the editors: November 10, 2006.

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* .

The paper is organized as follows: Section 2 is an overview of some spatio-temporal access methods. Characteristics of spatio-temporal data referenced in this paper are presented in Section 3. Section 4 presents the structures of BrickR, the manner in which data received by the system is managed. In Section 5 some experimental results are provided. There are also conclusions and proposed ideas as future work in Section 6.

2. Related Work

A lot of work has been done on organizing spatio-temporal objects in index structures. These structures may be classified as structures which index past data, store data about present and past or index data about present and future (predictions) [8].

Some index structures which organize past information are the following:
STR-tree, TB-tree [10] - are R-tree like structures, with the main characteristic that they attempt to achieve trajectory preservation for the same object by storing trajectory segments of the same object in the same tree node. One advantage is in answering object-oriented queries, like "Which was the object's X trajectory?". On the other hand, spatial window queries are not efficiently solved due to the fact that even if some trajectory segments belonging to two different objects are spatially closed, they are stored in different tree nodes.
SETI [4] - divides the spatial domain into a static partition and the data corresponding to one cell of partition are organized into an R-tree; one segment which intersects two cells is divided and the resulting pieces are stored into the corresponding cells; one major drawback is the lack of efficiency solving temporal queries, because all the cells tree must be searched.

There are spatio-temporal access methods that manage present (and past) data:
2+3 R-tree [9], 2-3 TR-tree [1] - are index structures that contain two R-trees: one tree for points that represent present data and another R-tree for the trajectories from the past; the search is possible to be done in both R-trees.
LUR-tree [7] - indexes only current positions of objects, which means that historical queries are not supported; this is a structure adapted to frequent updates.
Hashing [13] - stores only current data; the space is partitioned into zones that may be overlapped; an object belongs to one zone and its accurate position is acquired using an auxiliary layer.

The following structures belong to the third class of spatio-temporal access methods, which manage present data and data for prediction of future movement.
Duality transformation [6] - transforms a line segment from the time-space domain into a 2D point (an equation x_t = at + b is represented by the point (a, b)); indexing some trajectory segments spatially closed is not a guarantee that the representative points are closely stored in tree nodes.
TPR-tree [12], PR-tree [3], STAR-tree [11], TPR*-tree [14] - are included into a

new category of STAM; these structures index the original time-space domain using parametric bounding rectangles; this means that the rectangles are functions of time and are built to enclose moving objects trajectories; these access methods employ different optimization strategies of the parametric boxes.

3. Spatio-Temporal Data

As it was mentioned earlier in this paper, a new indexing method for one-dimensional objects with continuous evolution in time is proposed. A mobile object of which extent is not for interest is represented as point. In order to facilitate the representation and implementation of the system the movement of these objects in one-dimensional space is considered. The extension to the movement in R^n space, $n \geq 2$, is straightforward, and the indexing of objects with shape (for example polygonal shape) is proposed as future work.

The motion of a spatio-temporal object in 1D space is continuous in time: the objects cannot disappear or be teleported. It is natural for that object to travel with variable speed during different time intervals. Therefore the movement on a time interval is represented as a function of time defined and continuous on that interval, and the trajectory of a mobile object is built using these functions of time. Graphically representing these functions, we obtain a set of connected segments (the end of a line is the start of the next trajectory line), except the situation when, during a time interval, the information about the object's motion is unknown. The trajectory segments of an object do not intersect (an object cannot travel back in time and cannot exist simultaneous in two different locations). The only common points are the end points of the two temporal consecutive segments.

Therefore, an essential feature of a spatio-temporal object is the arbitrary movement in spatial domain, but only the chronological travel in the temporal domain. In this paper we consider that the object's speed stays invariably on a certain time interval, so that the time function representing a trajectory segment is linear.

It is not imposed a certain unique manner of receiving the spatio-temporal data sent by the mobile objects. The objects may possess GPS equipment and may send positioning data regularly or after the change of the motion parameters (speed, direction), or they may have a local implementation of a process unit of data and send to the server the trajectory segments. In any of these situations, the input data for BrickR structure are linear functions of time, and each of them is defined on a certain time interval and represented as line segment. The temporal feature of motion is the foundation of the design of BrickR structure (see section 4.2).

4. BrickR Structure

The permanent structure of the BrickR-tree is based on the R-tree design, therefore in the next section the R-tree family access methods are presented.

4.1. **R-Trees.** The R-trees [5, 2] are index structures for point objects and spatial objects with extent in R^d, d=2. These spatial access methods are based on the division of space depending on data adapting to the position of objects in space. The inclusion relation is used to establish the hierarchy between tree nodes [5, 2].

The objects stored in the R-tree nodes are called Minimum Bounding Rectangles (MBR). An MBR is the smallest d-dimensional rectangle including one or more given objects. The leaf nodes contain pointers to objects stored in the database, but represented inside the index by its MBR. The internal nodes of the tree contain pointers to child nodes and the smallest rectangles that include all the entries of those child nodes. Generally, a record stored within an R-tree node has the structure (P, MBR). If the node is a leaf node, P is a pointer to an object and MBR is the MBR of that object. Otherwise, P points to a child node and MBR is the minimum rectangle that encloses the child node's entries.

Two parameters of an R-tree are the maximum and the minimum number of entries within a node. These are noted here by M and m respectively. Their values are chosen by taking into account the size of the page that stores a node, and on the request of minimizing the number of nodes accessed within a query. Usually $m = \left\lceil \frac{M}{2} \right\rceil$, but it can be even smaller. The only exception is the root node, which can contain a minimum of one entry.

The well-known structures of the R-tree family are the R-tree, R*-tree and R+-tree. The main difference between the R-tree, R*-tree and R+-tree is the insertion algorithm of a new object into the tree. In the case of R-tree and R*-tree, the first step is the selection of the leaf node into which the new object will be inserted, based on some heuristic. If necessary, the MBR of that node will be enlarged, to enclose the whole object. The insertion into R+-tree is done using the clipping method: an object is divided intro pieces regarding the leaf nodes that intersect the object to be inserted. Therefore, comparing to R-tree, the R+-tree has the advantage that between leaf nodes are no overlapping areas, but the main disadvantage is that there are indexed more objects because of the clipping method (there may be more pieces for a single object).

In the case of R-tree and R*-tree, the nodes of the same level can overlap, so that the division of space cannot be necessary a disjunctive one. An MBR can intersect or be included within the MBR of more nodes, but it is associated with only one node. On the other side, on the same level in an R+-tree, the MBRs of nodes define disjunctive partitions of space.

The R*-tree is actually an optimized R-tree, mainly by the insertion and deletion algorithms [2], therefore the structure used for BrickR-tree is the R*-tree.

4.2. **Structures of BrickR.** One of the drawbacks of using an R-tree as a STAM [1, 9] is the possible extension of an MBR because of a trajectory segment too long. It is known that a MBR having large margin and area may cause large overlapping areas among tree nodes and dead space within nodes, which decreases

the performances solving queries.

The BrickR structures are mainly designed to minimize the overlapping in the tree structure. More then overlapping area, the experimental results (see Section 5) prove a better value of the sum of perimeters and areas of the tree nodes MBRs.

The design of the BrickR structures had as start ideas the following facts:

(1) the forced enlargement of an MBR because of a too long trajectory segment;
(2) the objects random movement within the spatial domain;
(3) the strictly chronological evolution on the temporal domain.

BrickR is an index structure and is composed of two structures:

- the permanent structure: an R*-tree type structure used to index spatio-temporal data 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: a grid structure which indexes the newest ST data received by the system; this structure is stored in main memory, having the advantage of a short time accessing data and performing operations.

The Temporary Structure

The temporary structure is designed to offer an extra "thinking" moment of how to group objects into MBRs. The main idea is not to insert an object into the main structure as soon as the data is received by the system. Storing more recent objects allows a better clustering of these, having almost void chances to get overlapping areas between the resulting MBRs.

The grouping of objects is accomplished with the help of one grid-type layer on the spatial domain. This grid is obtained by dividing the n-dimensional space using (n-1)-dimensional hyper-planes parallel with the Ot axis. As it was earlier mentioned, this paper discus the indexing of 1D data, but the extrapolation in a space of a greater dimension is straightforward. Therefore, in order to index one-dimensional ST objects, the 2-dimensional space is divided using a set of lines parallel with the temporal axis.

In this case, this grid is composed by a set of strips. The setting of the dividing lines (the margins of strips) may take into account the spatial distribution and the density of mobile objects or these lines may be equally distanced.

As data is arriving into BrickR system, each segment data is inserted into intersecting grid strips, using the clipping method. That means that if a segment intersects two or more strips, it is divided into pieces, each piece being totally enclosed within a strip. These segments are later grouped into rectangular bounding boxes and then such a MBR is sent to the permanent structure to be inserted.

The construction operation of a new MBR occurs when a strip satisfies the

adequacy criterion, meaning the number of segments stored in that strip. The adequacy criterion can vary, in accordance with the density of objects or their trajectories. In most of the tests, it was checked whether the strip contains at least fan-out-T * stripOccupancy segments, where fan-out-T is defined as the maximum capacity of a node of BrickR-T structure and stripOccupancy is equal to 3. It has been considered that a greater value than 3 may have as effect the management of a too big number of segments within a strip. On the other side, based on test results, it has been found out that the value 3 is large enough not to get overlapping areas between the MBRs cut out from the grid strips.

he value of the stripOccupancy parameter and the division of grid can be set according to the known or predicted movement of objects. This stripOccupancy's value is seen as a compromise between the number of trajectory segments stored within a strip and the chance that some overlapping areas may appear.

It can be noticed in figure 1(a) that, when stripOccupancy = 1, overlapping areas are obtained even within the temporary structure (the shadowed areas). In the situation when a greater value for stripOccupancy parameter is used (eg. 2), the segments are clipped and no overlapping area is obtained (see figure 1(b)).

The algorithm for the MBR shaping operation is based on the clipping method, as well as the insert operation. To get a new MBR from a strip, the algorithm tries to find a 1D line perpendicular on Ot axis, or a (d-1)-dimensional hyper-plane in R^d space, which cuts the strip so that the new rectangle to contain at most fan-out-T segments. Each line segment from the current strip is clipped if the cutting line intersects it: one piece is stored in the new MBR and the rest of it remains in the strip.

An observation has to be made about the flexibility of the grid structure. Let consider tStart(S) to be the smallest timestamp of a start point segment by the strip S. It may happen that the number of the alive objects at time tStart(S) to be greater than fan-out-T. This means that a new MBR cannot be obtained by cutting the strip with a line perpendicular on Ot axis. In such a case, the strip is divided using a hyper-plane parallel to Ot axis, resulting two narrower strips (see figure 1(b)). The division operation may continue until a new MBR can be built. The reverse operation of joining two neighbor strips may be included in the management algorithms of the grid structure, depending on implementation or input data.

The Permanent Structure

The permanent structure has the architecture of an R*-tree [2], but the nodes are organized into two different sub-structures. The terminal nodes containing the segment data are stored into BrickR-T and are organized into linked lists. A linked list corresponds to a strip from the temporary structure. The other structure called BrickR-I stores the internal nodes and follow the organization of an

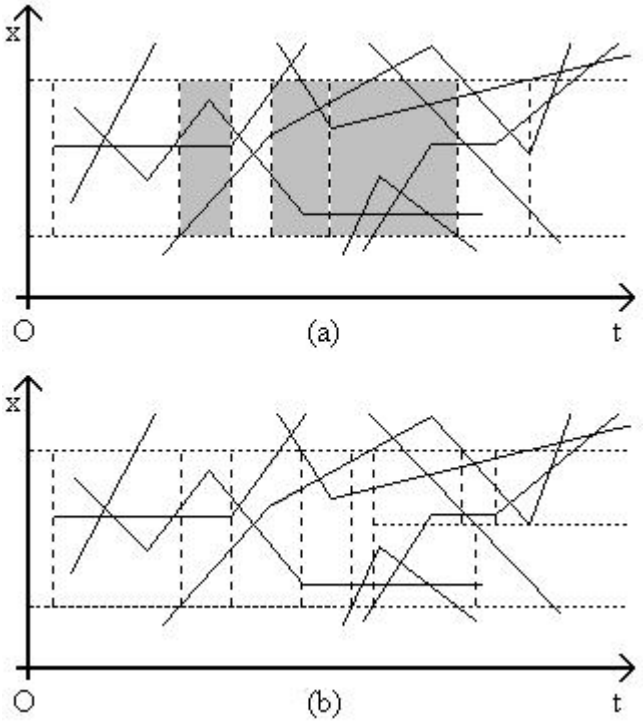

FIGURE 1. The construction of MBRs cut out from a grid strip, if fan-out-T = 4; the segment data is arriving in chronological order. (a) stripOccupancy = 1: the shadowed surface represents overlapping areas between MBRs (b) stripOccupancy = 2: no overlapping area is obtained.

R*-tree. Therefore, the insert operation into the permanent structure has two steps: the addition of the node containing segment data at the end of a linked list and the insertion of the node's MBR into the R*-tree like structure. In this way, the insertion of the terminal node is simplified.

The main advantages of the two structures of BrickR are emphasized:

- it does not have to insert separately each segment into the physical structure (many I/O operations are skipped), but the already grouped segments are sent as a node to the BrickR-T structure; the next step is the insertion of the new node's MBR into the upper structure (the BrickR-I)

- the odds to result overlapping areas between the MBRs sent to the permanent structure are minimized.

5. Experimental Results

A Delphi 2005 application using tables and stored procedures on a MS-SQL server has been developed to manage spatio-temporal data using 2D R*-tree, respectively BrickR. Tests has been run on two sets of data, containing different number of objects and using a fan-out of nodes of value 10 and a minimum occupied space of 40%. A record does not use a lot of space, therefore a paginated node stores usually more records than the fan-out considered. Some trial tests using a greater fan-out have showed an improvement of the BrickR efficiency and this demonstrates that the structure presented in this paper works well for realistic values of fan-out.

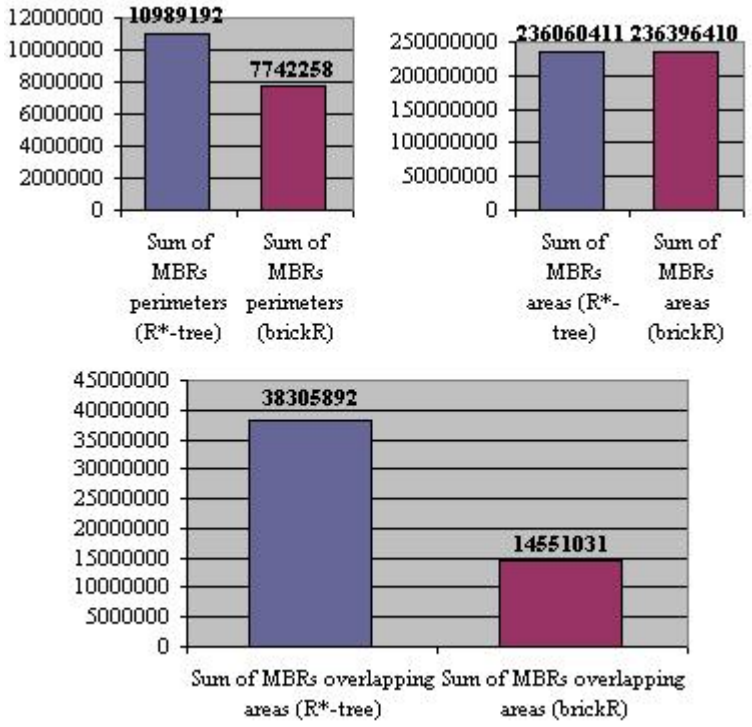

Figure 2. The comparisons between the sums of MBRs margins, the sums of the total areas, and the sums of the overlapping areas of the R*-tree and BrickR structures.

The point data signifying the end points of the trajectory segments have been randomly generated. Their coordinates (x, t) belong to a well-determined interval and the lengths of the resulting segments are not greater than some threshold (4%, 8.5%, 13% and 17% by the working space).

The parameters evaluated for the two tested access methods are the number of indexed objects, the number of nodes included into permanent structure, the sum of nodes MBR margin, the sum of nodes MBR area and the total value of the overlapping area between the nodes on the same tree level.

It was mentioned earlier that the clipping method has as consequence the growth of indexed objects number. This fact is pointed out by the tests: the amount of indexed objects during all tests within BrickR-I and BrickR-T structures is greater than the number of objects indexed using the R*-tree by approximately 45%. Nevertheless, the number of nodes occupied within BrickR structure is less than the number of nodes within R*-tree. This proves a much better usage of space occupied by BrickR structure: on the average, the node occupancy in the R*-tree is 58.61% and in the BrickR is 90.1%.

Figure 2 shows a lower value for the sums of MBRs margins and the overlapping area between the MBRs of BrickR structure's nodes, than the values obtained for R*-tree.

Another result noticed during tests is a substantial improvement of insertion operations running time: BrickR structure works 9 times faster than the R*-tree.

6. Conclusions and Future Work

The arguments and the experimental results show that the presented spatio-temporal indexing structure outperforms other ST index structures. Some improvements can be made for the insertion algorithm in BrickR. The optimization of temporal queries and the adjustment of the BrickR structure to allow mobile objects with shape to be indexed are proposed as future work.

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] A. Guttman, *R-Trees: A Dynamic Index Structure for Spatial Searching*, In Proc. of the ACM Intl. Conf. on Management of Data, SIGMOD, 47-57, 1984.

[6] G. Kollios, D. Gunopoulos, V. J. Tsotras, *On Indexing Mobile Objects*, In Proc. of the ACM Symp. on Principles of Database Systems, PODS, 261-272, 1999.
[7] 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.
[8] M. F. Mokbel, T. M. Ghanem, W. G. Aref, *Spatio-temporal Access Methods*, IEEE Data Eng. Bull., 26(2), 40-49, 2003.
[9] 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.
[10] 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.
[11] 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.
[12] S. Saltenis, C. S. Jensen, S. T. Leutenegger, M. A. Lopez, *Indexing the Positions of Continuously Moving Objects*, In Proc. of the ACM Intl. Conf. on Management of Data, SIGMOD, 331, 342, 2000.
[13] Z. Song, N. Roussopoulos, *Hashing Moving Objects*, In Mobile Data Management, 161-172, 2001.
[14] 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.

Faculty of Mathematics and Computer Science, Babes-Bolyai University, Cluj-Napoca, Romania
E-mail address: deiush@cs.ubbcluj.ro