현재와 과거 위치 질의를 위한 시공간 색인에 관한 연구

전봉기*

A Study on Spatial-temporal indexing for querying current and past positions

Bong-Gi Jun*

요 약

현재 및 과거 위치 질의를 위해 연속적으로 변경되는 위치의 이동은 저장되고 색인화 되어야 한다. 기존의 R-트리에 시간을 다른 차원으로 추가하여 간단하게 확장한 3차원 R-트리는 현재 위치 질의를 다루지 않고 있으며, 색인 노드들의 많은 중첩으로 인하여 공간 활용도가 낮다는 문제점이 있다. 이 논문에서는 분할된 노드의 공간 활용도를 높이기 위하여 향상된 3차원 R-트리의 동적 분할 정책을 제안한다. 또한, 이동체들의 현재 및 과거 위치를 질의하기 위해 새로운 태그 색인 구조를 소개함으로서 기존의 3차원 R-트리를 확장하였다. 현재 및 과거 위치 질의에서 제안하는 태그 동적 3차원 R-트리는 기존의 3차원 R-트리와 TB-트리 보다 성능이 우수하였다.

ABSTRACT

The movement of continuously changing positions should be stored and indexed for querying current and past positions. A simple extension of the original R-tree to add time as another dimension, called 3D R-tree, does not handle current position queries and does not address the problem of low space utilization due to high overlap of index nodes. In this paper, I propose the dynamic splitting policy for improving the 3D R-tree in order to improve space utilization of split nodes. I also extend the original 3D R-tree by introducing a new tagged index structure for being able to query the current and past positions of moving objects. I found out that my extension of the original R-tree, called the tagged dynamic 3DR-tree, outperforms both the 3D R-tree and TB-tree when querying current and past position.

키워드

Moving objects databases, Spatio-temporal indexing, R-tree, Splitting policies

Ⅰ. Introduction

One of new applications of database technologies is to deal with moving objects whose position changes continuously over time. The movement of moving objects needs to be stored in a moving objects database for answering queries about the object's movements. For speeding up query processing on the object's movements, I need a new method of indexing large amounts of spatial-temporal data in three-dimensional space, where two dimensions correspond to space and one dimension denotes time.

One of the problems not addressed by the

*신라대학교 컴퓨터정보공학부

접수일자 : 2004. 06. 18

3-dimensional R-tree index structures is to process range queries on the current and past positions of moving objects. The current position in this paper represents the lastly sampled location, which is to remain until changed. In this work, I assume the movement of moving objects is modeled as a set of lines, where each line has the two end points (xi, yi, ti) and (xj, yj, tj). The current position (xj, yj, tj) is to remain until changed in the future. For a time point after tj, it is unknown where the object is moving. For the time interval [tj, now), there exists no movement data in the moving objects database. In this paper, the term now is used for describing a time point of issuing queries. Since no movement data exist between the time of the current position and now, it is not possible to answer time interval queries with the interval [tj, now). To my knowledge, range queries on current and past positions cannot be handled by the 3D R-tree[1] and TB-tree[2].

The idea of this paper is to develop a new dynamic index structure based on the R-tree in order to efficiently process range queries on current and past positions. To improve the performance of processing range queries, I devised elaborate splitting policies of the 3D R-tree for inserting the movements of moving objects. This means that the proposed splitting policy provides a new dynamic 3-dimensional index structures for supporting range queries on current and past positions. Through modifications to the original 3-dimensional R-tree, I overcome the problem of low space utilization of index nodes and the problem of processing time interval queries based on now.

From the viewpoint of space utilization, I observe that the old node which is one of two split nodes along time axis does not have additional insertion any more. It is therefore desirable to assign more entries to the old node, which is named as the unbalanced splitting policy along time axis. In order to handle range queries on current and past positions, I modified the original 3D R-tree to keep the now tags which represent whether child index nodes include some of UC(Until Changed) pages or not.

The rest of the paper is organized as follows. Section 2 reviews the spatial access methods related to my work. Section 3 presents a tagged dynamic 3D R-tree for storing the current and past positions of moving objects. Section 4 describes the experimental comparison of the performance of the tagged dynamic 3D R-tree with the original 3DR-tree and TB-tree. Finally I summarize my contributions in the section 5.

II. Related Works

Many researches on spatial-temporal indexing methods have been done on the basis of data-driven approaches like variants of the original R-tree[3]. The continuous growth of time domain in three-dimensional space makes it difficult to use space-driven index structures like the KDB-tree[4], the quad-tree[5], etc.

Since the TB-tree[2] aims only for trajectory preservation, this allows that a leaf node only contains the line segments belonging to the same trajectory. While the TB-Tree are suited for trajectory-based querying, the performance is not adequate to process range queries on current and past positions.

The spatio-temporal indexing methods, namely the 3D R-tree[1] and HR-tree[6] are based on the R-tree where the third dimension corresponds to time. The 3D R-tree is the 3-dimensional version of the original R-tree. The 3D R-tree does not address the problem of high overlap of adjacent index nodes and does not describe how to process range queries on current and past positions. The HR-tree maintains an R-tree for each timestamp. The HR-tree performs well for moving objects not moved frequently. Because leaf nodes and non-leaf nodes should be newly created for frequently moved objects, the performance will be poor in range queries.

There exist a small number of proposals addressing the problem of handling now or until changed. The GR-tree[7], an extension of the R*-tree[8], was proposed for indexing general bitemporal data. But the GR-tree does not consider the overlap problem that caused by

inserting closed line segments.

III. MODELING AND STRUCTURES

In this section, I present the structure and algorithms of the tagged dynamic 3DR-tree for indexing both current and historical positions. To represent current and past positions of moving objects, I define UC lines that keep the lastly reported positions of moving objects until changed. In the following, I use the notion of the UC MBB, which denotes a time growing bounding box of which size depends on now.

3.1 Modeling positions of moving objects

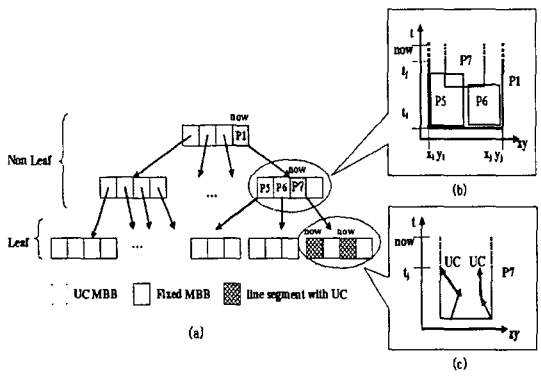

Fig. 1 Structure of the tagged dynamic 3DR-tree. (a) a hierarchy of nodes, (b) an example of non-leaf nodes (c) an example of leaf nodes.

Moving objects continuously move after the time point of the lastly sampled position. Let the previous sampled position be (xi, yi, ti) for a moving object MOi. If the new sampled position is (xj, yj, tj), a new line segment <(xi, yi, ti), (xj, yj, tj)> should be inserted. I define the lastly sampled position as the UC point, which means it should be to remain until changed. I need to consider two different types of line. First, the closed line, like <(xi, yi, ti), (xj, yj, tj)>, should be represented and stored in the databases. Second, the open-ended line (e.g. <(xj, yj, tj), (xj, yj, now)>) is a virtual line segment, where now is a variable representing the (ever increasing) current time. If the moving object reports another sampled position (xk, yk, tk) after tj where tk > tj, the open-ended line should be changed as follows : <(xk, yk, tk), (xk, yk, now)>. Whenever the recently sampled position is reported, the open-ended line needs to be redefined.

The time interval query with the range [tj, now) is required to compute with the open-ended lines. To find open-ended lines which intersect with the time interval [tj, now),first of all, it is necessary to find index nodes whose MBB overlaps with now-related time intervals. If leaf nodes include one or more UC line, the MBB of the leaf nodes should be logically and dynamically extended for supporting now-related time interval queries. I need to differentiate the UC MBB including UC lines from usual MBBs not including UC lines. Let UCLinei be a UC line segment of a moving object MOi. MBB(Minimum Bounding Box) including one or more UCLinei is denoted as the UC MBB. The fixed MBB means the MBB that does not contain any UCLinei.

3.2 Index structures

To indicate UC lines and UC MBBs, I add a tag now to the node structure of R-tree. Figure 2(a) illustrates the structure of the tagged dynamic 3DR-tree. In leaf nodes, the tag now indicates the corresponding entry has the UC point. In non-leaf nodes, the tag now indicates the corresponding entry has UC_MBB. Figure 1(b)(c) shows that the UC MBB is denoted as the time growing bounded box. The upper time bound of UC MBB is growing over time, and its value is now. In Figure 1(b), The bounding box of stored MBB are specified by <(xi, yi, ti), (xj, yj, tj)>, but the time-growing bounding box of UC MBB is specified by <(xi, yi, ti), (xj, yj, now)>. To process range queries, I substitute the time now of UC MBB for a time tq of issuing queries.

3.3 Search algorithms

Since the 3D R-tree handles only closed lines, it cannot answer now-related time interval queries. The tagged dynamic 3DR-tree is possible

to process range queries on current positions and trajectories of moving objects. Figure 2 shows the algorithms of processing range queries of the tagged dynamic 3DR-tree. The tagged dynamic 3DR-tree allows us to issue current position queries as well as range queries. For the current position queries, the SearchCurrentPosition algorithm checks that UC MBBs intersect with a given query region. Since the query of searching current positions is the time slice query with a specific time now, UC MBBs only are checked in the non-leaf nodes, and UC points only are returned in the leaf nodes.

```
Algorithm SearchCurrentPostions(node N, rectangle W)
IF N is a non-leaf node
    Invoke SearchCurrentPostions(N',W) for UC_MBBs
    that their MBBs intersect with the query window W.
ELSE
    Report oids and UC points of UC lines that their MBBs
    intersect with the query window W.

Algorithm SearchRangeTrajectories(node N, mbb W)
IF N is a non-leaf node
    Invoke SearchRangeTrajectories(N',W) for every entry
    intersects with the query window w.
ELSE {
    FOR EACH UC points    {
            // check open-ended lines
        Let UC_line be Line(the UC point, now)
        Report oid and UC_line that intersect with the
        query window w.
    }
    FOR EACH closed lines
            // check closed lines
        Report oid and the closed line that intersect the
        query window w.
}
```

Fig. 2 Search query algorithms for current and past positions.

The SearchRangeTrajectories algorithm is to process range queries related now, and it is easy to extend of the classical range query processing using the R-tree. For the tagged non-leaf node, the upper bound of UC MBB is changed to [xj, yj, now). A time-growing bounding box specified by <(xi, yi, ti), (xj, yj, now)> satisfies a range query <(x1, y1, t1), (x2, y2, t2)>, if and only if <(xi, yi), (xj, yj)> intersects <(x1, y1), (x2, y2)> AND ti $\leq$ t2 $\wedge$ now $\geq$ t1. For the tagged leaf node, it first needs to check open-ended lines with UC points. To find UC point, it needs to check the tag now of entries. Each UC line is checked the intersection with the query window. Second, it needs to check closed lines that intersect with the query window.

3.4 Insert algorithms

When an object reports a new position at time t, this requires two types of operations. One is the insertion of a new closed line segment that means the movement of the object. The other is to redefine UC point of the moving object. To choose the best leaf node for inserting the closed line segment, I use the algorithm ChooseSubTree of the original R-tree. The procedure InsertLSto3DRtree will handle the overflow of node, which is based on the dynamic merging and splitting policies of the tagged dynamic 3DR-tree described in the next section. Since a new inserted line segment has always a new UC point, I set the tag now of the inserted line segment. I try to find it's previous UC point, first, in the same leaf node, and next, in other leaf nodes. Searching of the previous sampled position can be achieved by point search queries. The update of the tag of the UC point does not lead to the structural change, but if a given leaf node does not have any UC points by the tag update, its changing information should be propagated to the parents of the leaf node.

```
Algorithm Insert(OID Id, Point PrevPos, Point NewPos)
LS ← Line(PrevPoint, NewPos);
        // make a closed line
PageNum ← ChooseSubTree(LS);
    // choose the best leaf node for inserting line segments
InsertLSto3DRtree(PageNum, id, LS);
        // insert the closed line into the 3DR-tree
ChangeUCPoint(id, LS.endpoint, PageNum, TRUE);
        // set the tag now of the new UC point
IF FindPrevUCinLeaf(id, PrevPos, PageNum);
        // find the previous UC point in the leaf node.
  ChangeUCPoint(id, PrePos, PageNum, FALSE)
        // reset the tag now of the previous UC point
ELSE  {
  PageNum ← FindPrevUCinTree(ROOT, PrePos);
        // find the previous UC point in the tree.
  IF PageNum > 0
        // success to find the previous UC point
    ChangeUCPoint(id, PrePos, PageNum, FALSE)
        // reset the tag now of the previous UC point
}
IF PageNum does not have UC lines
  AdjustTreeUC(PageNum, False);
        // propagate to the parents of this leaf node.
```

Fig. 3 Insertion algorithm.

3.5 The splitting policy along time axis

I propose a new splitting policy to improve space utilization of split nodes. The continuous growth of the movement requires enlargement of MBB along time axis. This results in splitting of overfull nodes along time axis. The old node of

two balanced split nodes shows low space utilization because of no insertion of the old node.

Figure 4 shows an example of unbalanced splitting along time axis. The results of unbalanced splitting are two split nodes, a UC MBB and a fixed MBB, where a UC MBB contains closed lines with UC points, but a fixed MBB does not include any UC point. In order to increase space utilization, I assign the maximum number of lines to the fixed MBB, if possible. The fixed MBB is allowed to contain more than a half capacity of a leaf node within the upper bound of Cnode.

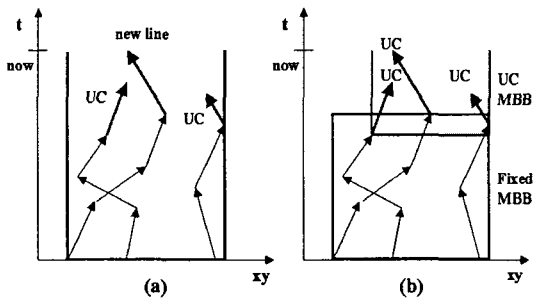

Fig. 4 The unbalanced splitting along time axis. (a) the insertion of a new line segment. (b) the unbalanced split along time axis.

IV. EXPERIMENT

In this section, in order to evaluate the performance of the tagged dynamic 3DR-tree proposed in this paper, I compared it with the original 3DR-tree and TB-tree.

Since there is no the real dataset related with moving objects, I utilize the GSTD[9] generator to generate a dataset being used for this experiment. I used two groups of datasets with varying number of objects and varying frequency of sampling position. VNO(Varying Number Objects) consists of four different datasets with objects per snapshot ranging from 500 to 2500, and report frequency per objects is 2000. VRF(Varying Report Frequency) consists of four different datasets with report frequency per objects ranging from 500 to 5000, and the number of moving objects is 1000. The performance evaluation vectors are summarized in Table 1.

Table. 1 Performance Evaluation Vectors

Symbols	Definitions	Evaluation Example
N_{MO}	The number of moving objects	500, 1000, 1500,2500
Ts	The frequency of sampling positions for each object	500, 1000, 2000, 5000
RQS	The Size of Range Query	5%, 10%, 20%

4.1 Comparison of space utilization

The high space utilization can reduce the cost of query processing because the height of the tree decreases. The space utilization of the R-tree is usually about 57%. The R*-tree[8] increases the space utilization up to 63% with the reinsert algorithm. The average space utilization of the R*-tree is higher than the R-tree, but the performance of range query processing is similar to each other.

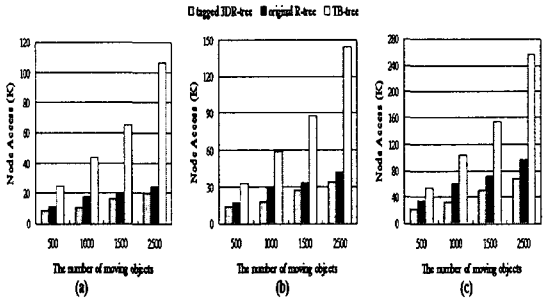

Fig. 6 Comparison for current position queries with VNO datasets : various range, (a) 5%, (b) 10% and (c) 20% in each dimension.

The TB-tree does not allow different segments from different trajectories to be stored in the same leaf node. A new line segment is inserted to the leaf node that contains its predecessor in the trajectory. The space utilization of the TB-tree is over than 90%. The disadvantage of the TB-tree is that leaf nodes of TB-tree are deeply

overlapped. My tagged dynamic 3DR-tree using the unbalanced splitting policy for the time domain shows that the space utilization is between 68% and 77%.

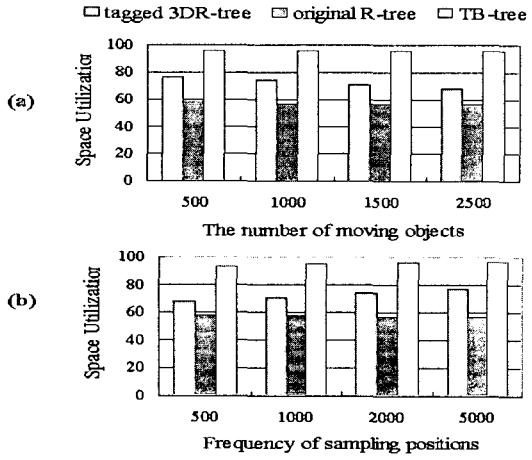

Fig. 5 Comparison of space utilization. (a) versus the number of moving objects. (b) versus the frequency of sampling positions.

Figure 5 shows space utilization for varying the number of moving objects and the frequency of sampling of an object's positions. As the experiments show, if the number of moving objects increases, the space utilization of the tagged dynamic 3DR-tree is worse. If the report frequency increases, the space utilization of the tagged dynamic 3DR-tree is better.

4.3 Performance evaluation of current position queries

The original R-tree and TB-tree does not support for querying the lastly reported positions of moving objects. Current position queries for the original R-tree and TB-tree are used range queries with one sampling interval of moving objects. I ignored the accuracy of query results for the original R-tree and TB-tree, but Figure 6 shows that the tagged dynamic 3DR-tree performs better than the original R-tree and TB-tree. Since time intervals of moving objects differ from each other, the performance of the original R-tree is worse than the tagged 3DR-tree. The performance of the TB-tree is not good than others, because the TB-tree is highly overlapped between nodes.

4.4 Performance evaluation for varying the number of moving objects

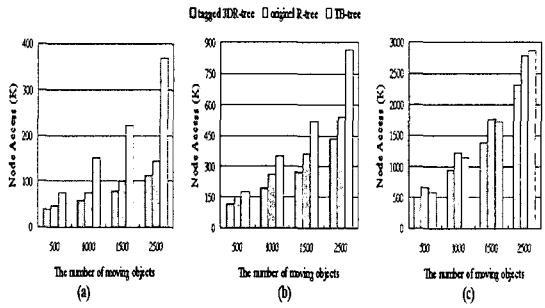

Fig. 7 The number of node accesses for processing range queries versus the number of moving objects : various range, (a) 5%, (b) 10% and (c) 20% in each dimension

The number of disk accesses for processing range queries depends on the number of moving objects. As Figure 7 shows, the tagged dynamic 3DR-tree performs better than the original R-tree and TB-tree for all range queries. The experiments show that the tagged dynamic 3DR-tree considerably can reduce the overlap between nodes by using the forced merging policy and the clipping big line segment policy. For a large range query, the performance of the original R-tree is similar to the TB-tree. Since the space utilization of the TB-tree is better than the others, the TB-tree is the best method for lowering the height of the tree.

V. Conclusion

I propose the modified R-tree based index structure, called the tagged dynamic 3DR-tree, to handle the problems of range queries on current and past positions. To improve the space utilization of index nodes, I proposed the unbalanced splitting policy for dividing an overfull node along time axis. By virtue of the unbalanced splitting along time axis, the tagged dynamic 3DR-tree has better space utilization

than the original R-tree.

The TB-tree is more efficient than the tagged dynamic 3DR-tree in the space utilization, but the performance of range queries is not good compared with the original R-tree and the tagged dynamic 3DR-tree because of high overlap of index nodes. In all experiments, the tagged dynamic 3DR-tree's space utilization is better than the original R-tree, and the performance of current position queries and range queries are also better. Because the choosing split axis algorithm is simple, the average insertion cost of the tagged dynamic 3DR-tree is lower than for the original R-tree and TB-tree.

References

[1] Y. Theodoridis, M. Vazirgiannis, and T. K. Sellis, "Spatio-temporal indexing for large multimedia applications," IEEE Int'l Conf. on Multimedia Computing and Systems, pp. 441-448, 1996.

[2] D. Pfoser, C. S. Jensen, and Y. Theodoridis, "Novel approaches in query processing for moving objects," Proc. of Int'l Conf. on Very Large Data Bases, pp. 395-406, 2000.

[3] A. Guttman, "R-trees: A dynamic index structure for spatial searching," Proc. of the ACM SIGMOD Conf. , pp. 47-54, 1984.

[4] J. T. Robinson, "The K-D-B-tree: A search structure for large multidimensional dynamic indexes," Proc. of the ACM SIGMOD Conf. , pp. 10-18, 1981.

[5] H. Samet, The Design and Analysis of Spatial Data Structures, Addision-Wesley, Reading, MA, 1990.

[6] M. A. Nascimento and J. R. O. Silva, "Towards historical R-trees," ACM symposium on Applied Computing, pp. 235-240, 1998.

[7] R. Bliujute, C. S. Jensen, S. Saltenis, and G. Slivinskas, "R-tree based indexing of now-relative bitemporal data," Proc. of Int'l Conf. on Very Large Data Bases, pp. 345-356, 1998.

[8] N. Beckmann and H. P. Kriegel, "The R*-tree: An efficient and robust access method for points and rectangles," Proc. of the ACM SIGMOD Int'l Conf. , pp. 332-331, 1990.

[9] Y. Theodoridis, J. R. O Silva, and M.A Nascimento, "On the generation of spatiotemporal datasets," Proc. of Int'l Symposium on Spatial Databases, pp. 147-164, 1999.

저자소개

Bong-Gi Jun

Received his M.S. and the Ph.D. degrees in Department of Computer Engineering from Pusan University, Pusan, Korea in 1993 and 2003, respectively.

From 1993 to 1998, he joined at Korea Telecom Research Center, where he worked as a manager of technical staff.

In 2003, he joined the Division of Computer and Information Engineering, Silla University, Korea, where he is presently a full time instructor.

His research interests include Geographic Information Systems, Moving Object Databases and Mobile Information Systems, etc.