MOVING OBJECT JOIN ALGORITHMS USING TB-TREE

Jai-Ho Lee, Seong-Ho Lee, Ju-Wan Kim

LBS Research Team, Telematics Research Group, ETRI
{snoopy, sholee, juwan}@etri.re.kr

ABSTRACT:

The need for LBS (Location Based Services) is increasing due to the widespread of mobile computing devices and positioning technologies. In LBS, there are many applications that need to manage moving objects (e.g. taxies, persons).

The moving object join operation is to make pairs with spatio-temporal attribute for two sets in the moving object database system. It is import and complicated operation. And processing time increases by geometric progression with numbers of moving objects. Therefore efficient methods of spatio-temporal join is essential to moving object database system.

In this paper, we apply spatial join methods to moving objects join. We propose two kind of join methods with TB-Tree that preserves trajectories of moving objects. One is depth first traversal spatio-temporal join and another is breadth-first traversal spatio-temporal join. We show results of performance test with sample data sets which are created by moving object generator tool.

KEY WORDS: Moving Objects, join, TB-tree

1. INTRODUCTION

Many researchers have been studying moving object database system which manages moving objects such as vehicle on road or soldier on the battlefield. Technologies of moving object database are applied various field for example LBS(Location Baseed Service) or Telematics. The technology of Moving objects database which manages effectively moving objects getting more and more important.

We divided Research for moving objects into two parts. One is moving objects data model and query language. The other is index for moving objects. In this paper, we considered the moving point(MPoint)[4] among moving objects, because our research oriented base engine for LBS applications. We focus on query and join algorithm for past trajectory.

The spatio-temporal join operation is to make pairs with spatio-temporal attribute for two sets. It is an important type of query for multiple scanning in moving databases. It is a sample query of spatio-temporal join : "Find all pairs of friends which distance of between two friends had been smaller than ten meter in yesterday."

Because the spatio-temporal join requires multiple scanning, it needs large disk I/O and CPU time. Until now studying of join for moving object was in defect.

We can process join operation more effective, if two sets which are target of join operation have spatio-temporal indexes. In the paper, we consider that all two sets have spatio-temporal index. We use the TB-tree among past spatio-temporal indexes. First we overview spatial join algorithm developed formerly, and describe how to extend it to spatio-temporal join.

2. RELATED WORKS

We classify spatial join algorithm using two spatial indexes into depth-first traversal R-tree spatial join, breadth-first traversal R-tree spatial join and transform-space view based spatial join.

The depth-first spatial join algorithm finds pairs of leaf node overlapping each other through depth-first traversing. The basic idea is to use the property that directory rectangles from minimum bounding box of ther data rectangles in the corresponding subtrees. Thus, if the rectangles of two directory entries Er and Es do not have a common intersection, there will be no pair (rectr, rects) of intersecting data rectangles where rectr is in the subtree of Er and rects is in the subtree of Es. Otherwise, there might be a pair of intersecting data rectangles in the corresponding sub trees. And local plane sweeping and pinning methods are used in order to preventing several reading to same nodes. But because heuristic methods do not consider access order of whole nodes participated in join operation and interest only access order of subtrees whose one pair of non-leaf nodes has common intersection, it's defect is that it can't find global optimum.

The breadth-first spatial join algorithm store pair of nodes have a common intersection in IJI(intermediate join index) through breadth-first searching. In IJI, since access order of whole node is considered, global optimal join sequences can be created. But if numbers of node pairs

have a common intersection store IJI, size of IJI increase rapidly and there is a remarkable drop in performance in contrast to depth-first traversal R-tree spatial join.

The transform-space view based spatial join order leaf nodes of one R-tree with spatial adjacency, and process region query one by one on other R-tree using ordered leaf nodes. Nodes accessed in previous query processing can be used to the highest degree in next query processing. That improves usage of buffer. Ordering leaf nodes of R-tree is achieved by space filling curve. Cost of ordering can be minimized if index is used when it is ordered. This method is proposed recent and become generally known that it shows best performance among join algorithms. But we need extension because it can only be applied in spatial index.

3. DEPTH-FIRST TB-TREE JOINS

The depth-first TB-Tree join algorithms is extended from R-tree join algorithm.

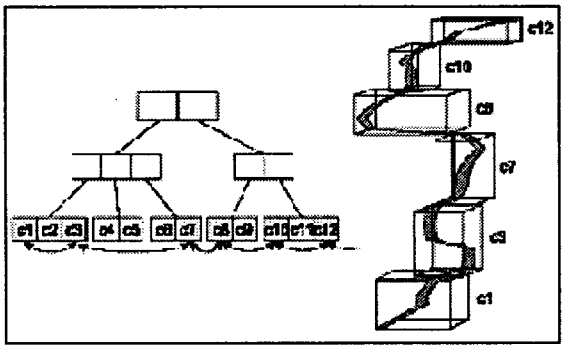

Figure 1. Structure of TB-Tree

Figure 1 is a structure of TB-tree. The TB-tree discretely extracts moving object location, represents two continuous locations of extracted moving objects as line segment, and represents set of connected line segments as trajectory. It is an access method that strictly preserves trajectories such that a leaf node only contains segments belonging to the same trajectory.

The basic algorithm of spatio-temporal join for TB-tree uses a characteristic of TB-tree strurecture that entries in nodes are sorted by time and MBB like cube. Axises of cube consist of X axis, Y axis and T(time) axis. If the MBBs of two node entrires Er and Es do not have a common intersection, there will be no pair (MBBr, MBBs). Otherwise there might be a pair of intersecting data MBBs in the corresponding subtrees. In the following subsections we assume that both trees are of the same height.

```
SpatioTemporalJoin1( R, S : TB_Node)        (* height
of R is equal height of S *)
    SortedIntersectionTest( R,S, Seq );
    For I = 1 To Seq.legth Do
        (Er, Es) = Seq[I];
        if( Both R and S is a leaf page )
            output(Er,Es)
        Else
            ReadPage( Er.rrn );ReadPage( Es.rrn )
            SpatioTemporalJoin1( Er.rrn, Es,rrn )
        End
    End
End
SortedIntersectionTest( Rseq, Sseq: Sequence of
mbb;
                var output : sequence of pair of
MBB );
    output = null;
    I = 1; j= 1;
    While( I <= Rseq.Length  and j <= Sseq.Length )
Do
        If Rseq[I].StartTime < Sseq[I].StartTime Then
            InternalLoop( Rseq[I], j, Sseq, output )
            I = I + 1;
        Else
            InternalLoop( Sseq[I], I, Rseq, output )
            J= J + 1;
        End
    End
End
```

STJ1(SpatioTemporalJoin1) presents first approach based depth-first traversal approach. The TB-tree has the structual charateristic that all entries are sorted by time. The SortedIntersectionTest function is applied plance-sweep method for intersection operation using this characteristic. It is unnecessary to sort by one axix, because nodes were already sorted. This can reduce cost.

Additionally, the local plane-sweep order with pinning method is a optimal method which reduce CPU-time and IO-time.

We restrict the search space of the join. In algorithm SpatioTemporalJoin1, each entry of the one node is checked for the join condition using the plane sweep against all entries of the other node. But before all entries of R is compared all entries of S, each entries of R is checked having a common intersection with S. each entries of S also is checked having a common intersection with R. This process can reduce considerably comparison number.

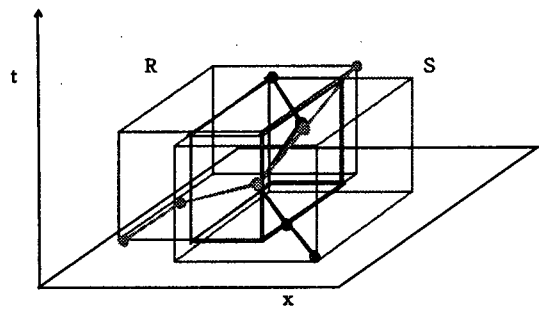

Figure 2. Restriction of search area.

An example is illustrated in Figure. 2. When algorithm SpatioTemporalJoin1 performs the plane sweep, 4 segments in R area and 4 segments in S area are participated. But new algorithm SpatioTemporalJoin2 are required 3 segments in R area and 3 segments in S area. Therefore, number of comparison required in order to compute the join condition is reduced. The modified algorithm is specified as follows.

```
SpatioTemporalJoin2( R, S : Node,
                     mbb : cube_Parallelepiped )
   R = { Ei | (Ei ∈ R) and ( Ei.mbb ^ mbb <> null )
   S = { Ei | (Ei ∈ S) and ( Ei.mbb ^ mbb <> null )
   SortedIntersectionTest( R,S, Seq );
   For I = 1 To Seq.legth Do
       (Er, Es) = Seq[i];
       if( Both R and S is a leaf page )
         output (Er,Es)
       Else
         ReadPage( Er.ref );ReadPage( Es.ref )
          SpatioTemporalJoin2( Er.rrn, Es.rrn, Er.mbb
^ Es.mbb)
       End
   End
End
```

We apply "local plane-sweep order with pinning" method to TB-tree joins. We determine a pair (Er,Es) of entries with respect to the local plane sweep order. After the corresponding subtree Er.rrn and Es.rrn, we compute degree of the cubes of both entries. The degree of an cube of entry E, short degree(E.cube) is given by the number of intersection between cube E.cube and the cubes which belong to entries of the other tree not processed util now. we pin the page in the buffer whose corresponding cube has maximal degree. The spatio-temporal join is performed on the pinned page with all other pages. Then we determine the next pair of entries using the local plane-sweep order again. In example of Figure 3, we obtain degree(R1)=0 and degree(S2)=2. Thus, the read schedule is <R1, S2, R4, R3, S1, S2>.

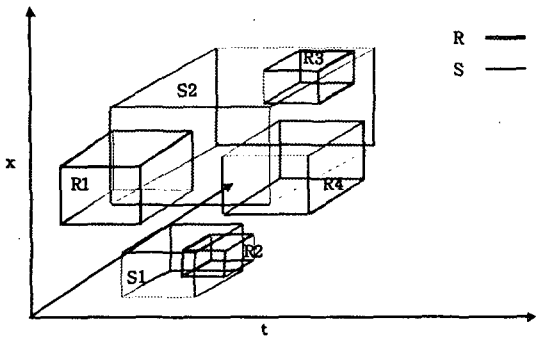

Figure 3. Example of spatio-temporal join.

We call the corresponding join algorihm SpatioTemoralJoin3 that reduce to repeat accesses for same page.

```
SpatioTemporalJoin3( R, S : Node, mbb :
    Cube_Parallelepiped )
R = { Ei | (Ei ∈ R) and ( Ei.mbb ^ mbb <> null ) }
S = { Ei | (Ei ∈ S) and ( Ei.mbb ^ mbb <> null ) }
SortedIntersectionTest( R,S, Seq );

Do
    (Er, Es) = Seq[0];
    if( Both R and S is a leaf page )
      output(Er,Es)
    Else
      ReadPage( Er.rrn ); ReadPage( Es.rrn )
        SpatioTemporalJoin( Er.rrn, Es.rrn, Er.mbb ^
Es.mbb)
    End
    Seq.Remove(0);
    DegR = Degree( Er ); DegS = Degree( Es );

    If( DegR = 0 and DegS = 0 )
          // go to first of loop
    Else If( DegR >= DegS )
        SeqIntersectR = { (Ex,Ey) | (Ex,Ey) ∈ Seq
and ( Er = Ex ) or (Er = Ey) }
        For I = 1 To SeqIntersectR.length Do
         (Ex, Ey) = SeqIntersectR[I];
         SpatioTemporalJoin3( Ex.rrn, Ey.rrn, Ex.mbb
^ Ey.mbb)
         Seq.Remove( Ex,Ey );
        End
    Else
        SeqIntersectS = { (Ex,Ey) | (Ex,Ey) ∈ Seq
and ( Es = Ex ) or (Es = Ey) }
        For I = 1 To SeqIntersectS.length Do
         (Ex, Ey) = SeqIntersectS[i];
         SpatioTemporalJoin3( Ex.rrn, Ey.rrn, Ex.mbb
^ Ey.mbb)
         Seq.Remove( Ex,Ey );
        End
    End
While( Seq is not Empty )
End
```

4. BREADTH-FIRST TB-TREE JOINS

Breadth-first R-tree join algorithms also extend to breath-first TB-tree joins. This method was proposed in order to optimize global. Since depth-first traversal algorithm is based on heuristic, it only optimize local.

STJ4(SpatioTemporalJoin4) is a bread-first TB-tree join algorithm. It uses data structure which called a intermediate join index.

First, for two root nodes, pairs of node which satisfy join condition are stored IJI[0] by calling SortedIntersectionTest function. Next as it goes down to child nodes, it find next pairs using pairs stored in IJI[i]. Therefore it considers globally pair of nodes.

```
SpatioTemporalJoin4( R, S : Node)
// R,S are two TB-trees, hR= hS
// IJI[i] intermediate join index at level i

    IJI = null;
    IJI[0] = SortedIntersectionTest( R,S );
    Int i=0;
    While( i < hR-1 do )
        foreach <r_node,s_node> in IJI[i] Do
         IJI[i+1] = IJI[i+1]U
            SortedIntersectionTest( r_node,s_node )
      End
      i=i+1
    End
    output IJI[i]
End
```

5. COMPARIONS OF PERPORMANCE

We take two sets of moving object traejctory for spatio-temporal join with TB-tree. One is TB-tree R, the other is TB-tree S. We will experiment with CPU prcoessing time and I/O processing time by spatio-temporal join between R and S. Like performance test for spatial join, to check the join condition in case of spatio-temporal join is far more expensive. Therefore, a good measure for performance consists of both, the number of disk accesses and the number of comparison.

The I/O-time is measured in the number of disk accesses required for perfoming the join. The CPU-time of a spatio-temporal join is measured in the number of comparions. Comparis on is to check the join condition, i.e. whether two MBB(minimum Bounding Box)s intersect or two trajectories intersects.

It is very difficult to get the real data for test of spatio-temporal join. So we use two moving object data set that generated by City Simulator among moving objects data genertors. Two data sets have different options when those are generating. But those are same attribute that a thousand of moving objects report a thousand of location. The Table 1 shows properties of TB-tree R and S with diverse page sizes.

Table 1. Properties of TB-Tree R and S

Page size	M	TB-Tree R, S		
		높이	노드수	데이터수
1KB	16	5	68,273	1,009,000
2KB	33	3	31,970	1,009,000
4KB	67	3	16,245	1,009,000
8KB	136	2	8,060	1,009,000

Table 2 reports the result for the number of comparison for both SpatioTemporalJoin1 and SpatioTemporalJoin2. The results show improvement in the number of comparison.

Table 2. Properties of TB-Tree R and S

Page size	STJ1	STJ2	Performanc e gain
1KB	195,573,208	155,644,663	1.3
2KB	102,352,608	88,991,603	1.2
4KB	70,735,540	57,201,341	1.2
8KB	72,216,669	47,681,498	1.5

The Table 3 reports the result for the number of disk access for SpatioTemporalJoin1, SpatioTemporalJoin2 and SpatioTemporalJoin3. The results show a huge improvement in the number of disk access.

Table 3. Properties of TB-Tree R and S

Page size	STJ1, STJ2	STJ3	Performance gain
1KB	2,126,230	1,219,160	1.7
2KB	839,466	512,915	1.6
4KB	719,142	427,169	1.6
8KB	684,342	387,041	1.7

6. CONCLUSION

The moving object database need a efficient join operation because it requires a large number of disk I/O and CPU time. But, until now, research about spatio-temporal join is insufficient.

We studied spatio-temporal join using TB-tree. We applied techniques of spatial join to spatio-temporal join. We showed depth-first traversal TB-tree joins and breadth-first traversal TB-tree joins and result of performance comparison.

Since volume of moving object data is generally huge, a research about optimal join algorithms is required.

References

Dieter Pfoser, Christian S. Jensen, Yannis Theodoridis: Novel Approaches in Query Processing for Moving Object Trajectories. VLDB 2000: 395-406

Brinkhoff, Hans-Peter Kriegel, Bernhard Seeger, Efficient processing of spatial joins using R-trees, Proceedings of the 1993 ACM SIGMOD international conference on Management of data, p.237-246, May 25-28, 1993, Washington, D.C., United States

Farshad Fotouhi, Sakti Pramanik: Optimal Secondary Storage Access Sequence for Performing Relational Join. IEEE Trans. Knowl. Data Eng. 1(3): 318-328 (1989)

Martin Erwig, Ralf Hartmut Güting, Markus Schneider, Michalis Vazirgiannis: Spatio-Temporal Data Types: An Approach to Modeling and Querying Moving Objects in Databases. GeoInformatica 3(3): 269-296 (1999)