Spatio-Temporal Join for Trajectory of Moving Objects in the Moving Object Database

Jai-Ho Lee
ETRI Telematics Research Division LBS Team
161 Gajeong-dong, Yuseong-gu, Daejeon, 305-350, Korea
snoopy@etri.re.kr

Kwang-Woo Nam
Kusan National Univ
san 68, Miryong-dong, Kunsan, Jeollabuk-do, 573-701 Korea
kwnam@kunsan.ac.kr

Kwang-Soo Kim
ETRI Telematics Research Division LBS Team
161 Gajeong-dong, Yuseong-gu, Daejeon, 305-350, Korea
enoch@etri.re.kr

Abstract: In the moving object database system, spatio-temporal join is very import operation when we process join moving objects. Processing time of spatio-temporal join operation 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 propose spatio-temporal join algorithm with TB-Tree that preserves trajectories of moving objects, and show result of test. We first present basic algorithm, and propose cpu-time tunning algorithm and IO-time tunning algorithm. We show result of test with data set created by moving object generator tool.

Keywords: moving object, join.

1. Introduction

Global Positioning System(GPS) has been widely accepted as the technology for location objects such as vehicles on a highway or soldiers in a battlefield. Wireless communications technology has also gained popularity. With these technologies in place, finding and managing the current location of moving objects have become possible. The Moving Object Database appeared in order to deal with objects changing location with time.

In this paper, we focus movong point among moving objects because our research oriented base engine for LBS applications. A Moving Point is a point object that moves along linear moving trajectory and occupies a part of Euclidian space over varying time. We need spatio-temporal acess methods that support efficiently spatio-temporal query such as "find some object within specific area durion specific period" and high performance. The spatio-temporal acess methods can be divided one reseach managing future location based current location from the other reseach managing past location.

We can separate query into two types in moving object database. One is time-series query and the other is continuous moving object query. The continuous moving object query has slice, snapshot, project and join query. We can process time-serises query and slice, snapshot and project query through one data aceess. But join query is required multiple data access. In addition, processing time of join query increases by geometric progression with object's numbers. Generally, join operation need data acess of several times. The spatio-temporal join operation is to make pairs with spatio-temporal attribute for two sets. And the import spatio-temporal join query is intersection of trajectory. A sample query of spatio-temporal join is such as :

```
Select intersection( r.location, s.location )
From R r, S s;
```

The above example query means that search pairs of data intersecting their trajectory for two moving object data sets.

It is various and sophisticated to process query about moving object. So we need query process method considered the characteritic of moving objects. Above all, we focus on design and implement algorithm of spatio-temporal join with spatio-temporal index. We will suggest a effective spatio-temporal join algorithm with spatio-temporal join index designed previous. A reason of using spatio-tmeporal index to procesing spatio-temporal join operation has a strong resemblance to using sptial index to processing spatial join operation. So we take into consinderation the TB-Tree index among diverse spatio-temporal indexes.

The structural characteristic of TB-tree is that entries stored always are sorted by time. We can simplify and improve spatio-temporal join algorithms. And using extra information stored at the leaf level, we can process filter and refinement step only access index, in order to process join query without acessing data. Those are the reason why we chose TB-tree index as spatio-temporal join index.

2. Related Works

Until now, many research focus on spatial join than spatio-temporal join.

The research of spatial join with R-tree started from [1]. In the [1], they presented join algorithm with R*-tree and used LRU buffer so as to reduce I/O time. They suggested two method reducing CPU time in the processing spatial join. One is resticting the search space and the other is sptaial sorting and plane sweep approach. In order to improve disk I/O time, they decide order of page demanded to read in buffer, in join processing. They improved I/O time using methods such as local plane-sweep order, local place-sweep order with pinning and local Z-order. We try appling these methodes to spatio-temporal join processing.

The TB-tree [2] discretely extracts moving object location, represents two continuous locations of extracted moving objects as line segment, and represents set of connected line segements as trajectory. It is an access method that strictyly preserves trajectories such that a leaf node only contains segements belonging to the same trajectroy. And the TB-tree has doubly linked list that connects leaf nodes including parts of the same trajectory in a way that preserves trajectory evolution. So the TB-tree is supports trajectroy-based queries much more efficiently. Because the TB-tree use method that last entry is inserted new node, unlike the R-tree use split method, it has high space utilization.

3. Spatio-Temporal Join

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 in [1], 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 comparions.

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. Comparions 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 propertis of TB-tree R and S with diverse page sizes.

1) Basic Algorithm of a spatio-temporal join for TB-tree

Table 1. Properties of TB-tree R and S.

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

We first describe the basic version of the algorithm, and then introduce extensions to it as well as ways to improve its performance

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.

The following algorithm presents first approach.

```
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 )
            // Refinement
            if( RealIntersectionTest(Er,Es) )
                InsertResultHashTable(R.id,S.id)
        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 );
    (*Rseq and Sseq are sorted *)
    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
```

In *SpatioTemporalJoin1* algorithm, TB_Node is node type in TB-tree and each node contains a set of entry.

One Entry E consist of RRN(Relavtive Record Number) and MBB. The RRN is information of sub node.

The TB-tree has a 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. The SortedIntersectionTest function performs plance sweep operation by time axis. Because a trajectory of one object can be stored in some nodes, a pair of object may be already finded. Therefore, we remove some replication as result sets store hash table.

The **Table** 2 shows results of the algorithm *SpatioTemporalJoin1* using TB-tree R and S as input. The spatio-temporal join is performed with various page sizes.

Table 2. The number of disk access and comparion for SpatiotemporalJoin1

Page size	# of comparions	# of disk access
1KB	195,573,208	2,126,230
2KB	102,352,608	839,466
4KB	70,735,540	719,142
8KB	72,216,669	684,342

There are two parts in the algorihm SpatioTemporalJoin1 which are worth to be improved. First, CPU-time consumption is rather high since each entry of the one node is checked against all entries of the other node. Second, pages are selected for the next call of *SpatioTemporalJoin1* without taking into account the I/O cost for reading these pages. A better apporach would be to compute the best sequence of pairs of pages required for computing the join on the next level of the TB-tree.

2) CPU-time Tunning

The consumption of CPU-time is proportional to the number of comparions required for computing the join condition (i.e the test whether two cubes or trajectories intersect). So our aim is to reduce the number of comparions for each call of SpatioTemporalJoin1, i.e. for each pair of qualifying nodes.

In order to reduce CPU-time, we examine this approach. For a given pair of node, we restrict the search space of the join such that in SpatioTemporalJoin1 only a small number of entries has to be considered. 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.

Let us consider two directory entries Er and Es which fulfil the join condition, i.e. Er.mbb $\cap$ Es.mbb $\neq \varnothing$. Then, *SpatioTemporalJoin1* is invoked using Er.rrn and Es.rrn as input. The following property is very important with respect to reducing CPU-time. Only the entries of Er.rrn and Es.rrn which intersect the intersection cube Er.cube $\cap$ Es.cube may have a common intersection. This property can be used for improving the join.

An example is illustrated in Fig. 1. When algorithm SpatioTemporalJoin1 performs the plane sweep, 4 segments in R area and 4 segements in S area are participated. But new algorithm SpatioTemporalJoin2 are required 3 segments in R area and 3 segements in S area. Therefore, the number of comparions required for computing the join condition reduces. The modified algorithm is specified as follows.

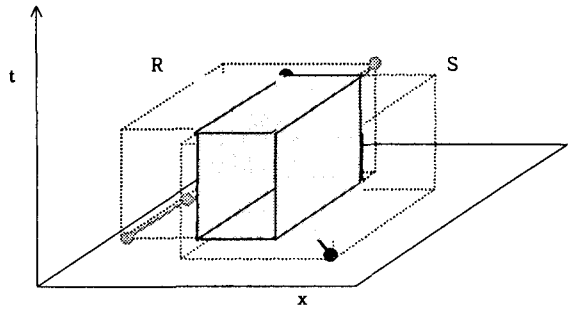

Fig. 1. Restricting the search space of a spatio-temporal join.

```
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 )
            // Refinement
            if( RealIntersectionTest(Er,Es) )
                InsertResultHashTable(Er,Es)
        Else
            ReadPage( Er.rrn );
            ReadPage( Es.rrn )
            SpatioTemporalJoin2( Er.rrn, Es.rrn,
Er.mbb ^ Es.mbb)
        End
    End
End
```

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

Table 3. Comparions of the with/without restricting the search space.

Page size	STJ1	STJ2	performance 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

3) IO-time Tunning

In order to reduce I/O-cost, we must computer a read schedule of pages required for spatio-temporal join of two TB-trees such that the number of disk access is

minimal. To solve this problem, we apply "local plane-sweep order with pinning" method that was already presented in [1].

We determine a pair (Er,Es) of entries with respect to the local plane sweep order. After the correspongding subtree Er.rrn and Es.rrn, we compute degree of the cubes of both enteries. 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 perfomed on the pinned page with all other pages. Then we determin the next pair of entries using the local plane-sweep order again. In example of "Fig. 2", We obtain degree(R1)=0 and degree(S2)=2. Thus, the read schedule is

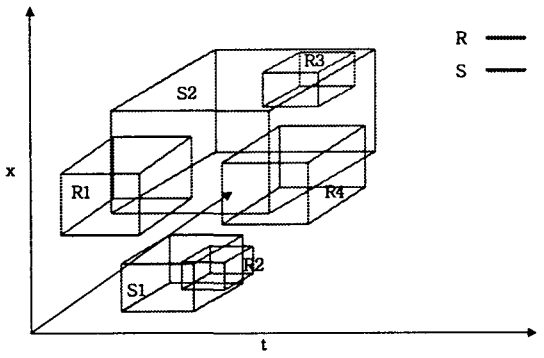

Fig. 2. Example for spatio-temporal join situations

<R1, S2, R4, R3, S1, S2>.

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

The **Table 2** 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 2. Number of disk access of algorithms.

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

4. Conclusions

In order to perform join query in moving object database that store trajectroies of moving point, we have presented study of spatio-temporal join processing using TB-tree.

First, we have presented a straightforwd join algorithm For TB-tree. We have shown that this apprach requires optimization with respect to both CPU- and I/O-time. In order to reduce CPU- and I/O-time, we suggest techniques which make use of restricting the search space and local plane-sweep order with pinning.

```
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 )
        // Refinement
        if( RealIntersectionTest(Er,Es) )
            InsertResultHashTable(Er,Es)
    Else
        ReadPage( Er.rrn );
        ReadPage( Es.rrn )
        SpatioTemporalJoin3( 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
```

In our future work, we are interested in a more study of spatio-temporal joins.

References

[1] 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

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

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

[4] Ming-Ling Lo, Chinya V. Ravishankar: Spatial Joins Using Seeded Trees. SIGMOD Conference 1994: 209-220.

[5] Cotelo L., J. A. Forlizzi,, L, Guting, R. H. , Nardelli, E., and Schneider, M., "Algorithms for Moving Objects Databases", FernUniversitat Hagen, Informatik-Report 289, October 2001.