Joint Conference of 1996 International Computer Symposium
December 19~21, Kaohsiung, Taiwan, R.O.C.

Two Dominance Searching Based Temporal Join Algorithms

C. Y. Chen and J. F. Hu

C. Y. Chen is with the Department of Electronics,
Feng Chia University, Taichung, Taiwan 40724, R. O. C.

J. F. Hu is with the Institude of Electrical Engineering,
Feng Chia University, Taichung, Taiwan 40724, R. O. C.

Abstract

In this paper, we are concerned with the problem of efficient processing of temporal join operation relations. By mapping time intervals to points in the plane, we first show that the problem of determining the set of all matching tuples of a temporal join is equivalent to a dominance searching problem in the plane. Then, by using an efficient data structure for solving the dominance searching problem as an index for the inner relations, we propose a new nested-loops based temporal join algorithm. For the cases where the index for the inner relations is too large to fit in the primary memory, we propose anther partition_based temporal join algorithm which doesn't need any index for the operand relations. Finally, in order to provide more efficient processing of temporal join, we propose a cluster scheme and an index scheme to support efficient strong of tuples and direct access of matching tuples.

Keyword:Temporal database, temporal join, dominance searching

1. Introduction

Many real database applications intrinsically involve temporal information (or time-varying information). Therefore, in recent years, many efforts have been devoted to the area of efficient processing of temporal data [Snodgrass 1987, Elmasri and Wuu 1990, Leung and Muntz 1990, Gunadhi and Segev 1991, Leung and Muntz 1992, Rana and Fotouchi 1993, Chen et al. 1994, Lu et al. 1994]. These include temporal data modeling [Snodgrass 1987, Elmasri and Wuu 1990] and query optimization [Leung and Muntz 1990, Gunadhi and Segev 1991, Leung and Muntz 1992, Rana and Fotouchi 1993, Chen et al. 1994, Lu et al. 1994].

By a temporal relation we mean a set of temporal data in the relational database model. There are various ways to represent temporal data in the relational model; detail discussion can be found in [Segev and Shoshani 1988]. What we adopt in this paper is a time interval representation [Leung and Muntz 1990, Gunadhi and Segev 1991, Rana and Fotouchi 1993, Lu et al. 1994, Chen et al. 1994]. In this representation model, the time dimension is considered as a sequence of discrete, consecutive, equally-distanced time constants. A time interval is defined as a set of consecutive time instants t_S, t_S+1, t_S+2,......; t_E, and is denoted as $<t_S, t_E>$, where t_S is called the starting time and t_E is called the ending time, respectively, of the time interval. A temporal relation is a set of temporal tuples. A temporal tuple consists of the surrogate of the tuple, some non-time varying attributes, at least one time-varying attribute (or temporal attribute), and two time attributes T_S and T_E. T_S and T_E constitute a time interval$<T_S, T_E>$ which indicates the period of time that the given values of temporal attributes are valid and is called the lifespan of the tuple. The lifespan of a temporal relation is defined as the time interval $<L_S, L_E>$, where L_S is the minimum starting time and L_E is the maximum ending time of all tuples in the relation, respectively. In addition, temporal relations are assumed to be in the first temporal normal form [Segev and Shoshani 1988]; i.e., there are no two intersecting time intervals for a given surrogate instance. Two time intervals $<a, b>$ and $<c, d>$ are said to intersect each other if and only if $a \leq d$ and $c \leq b$.

Temporal join is the most common operation on temporal relations for finding events that happen in the same period of time. Let us call two tuples from two distinct relations respectively matching tuples if their time intervals intersect each other. A temporal join on two temporal relations determines firstly, for each tuple of one relation, the set of all matching tuples from the other relation; then concatenates each pair of matching tuples into a tuple of the resulting relation. Accordingly, the result of a temporal join on two temporal relations is also a temporal relation in which the time interval of each tuple is the intersection of time intervals of the associated matching tuples.

Thus unlike the "snapshot" join in traditionally relational databases which is an equijoin, temporal join is a non-equijoin operation in its nature, and hence is a very expensive operation over temporal relations. Besides, temporal join has great flexibility to specify multiple join predicates over the same pair of temporal relations [Gunadhi and Segev 1991]. Further, temporal join depends heavily on the order of data to be processed [Leung and Muntz 1990, Gunadhi and Segev 1991]. Consequently, a conventional query processor is deficient for the processing of temporal joins. Therefore, in recent years, there has been an increasing interest in the problem of efficient processing of temporal join operation over temporal relations [Leung and Muntz 1990, Gunadhi and Segev 1991, Rana and Fotouchi 1993, Leung and Muntz 1992, Chen et al. 1994, Lu et al. 1994, Soo et al. 1994, Shin and Meltzer 1994].

Further, it is seen that the essential issue of the temporal join operation is to determine efficiently the set of all pairs of matching tuples. This is, in fact, equivalent to the following Time Interval Intersection Searching Problem : given a set of time intervals I_1, I_2,......, I_n, store them such that, given a query time interval I, we can efficiently determine those intervals among I_1,I_2,......and I_n, that intersect I.

Since temporal relations are usually very large and are stored on disks, to support temporal join efficiently, another important issue is knowing how to cluster tuples in such a way that matching tuples are stored together. In addition, to support direct access to the matching tuples, an efficient accessing index should be built as well.

In this paper, we are concerned with the problem of efficient processing of temporal join operation on temporal relations. The remainder of the paper is organized as follows. In Section 2, we review the previous works. In Section 3, we first show that the time interval intersection searching problem is equivalent to a dominance searching problem in the plane, and by using an efficient data structure for solving the north-west dominance searching problem as an index for the inner relation, we propose a nested-loops based temporal join algorithm. We also suggest a spatial cluster scheme and a spatial index scheme to support efficient storage of tuples and direct access of matching tuples for the proposed temporal join algorithm. In Section 4, we propose another partition_based temporal join algorithm that doesn't need any index for the operand relations to solve the cases where the index for the inner relation is too large to fit in the primary memory. Finally, conclusions and future research problems are presented in Section 5.

2. A Review of Previous Works

Although temporal join is one of the most frequently used and most expensive operation for temporal relational databases, it was not extensively studied until 1989. Many of the previously suggested methods are in fact extensions of the existing three major methods for the conventional snapshot join; namely, the nested-loops join, the sort-merge join, and the partition_based join [Gunadhi and Segev 1991, Rana and Fotouhi 1993, Lung and Muntz 1994, Lu et al. 1994, Shen et al. 1994].

Segev and Gunadhi [1989] considered strategies to process the temporal join operator. Subsequently, Gunadhi and Segev [1991] analyzed the characteristics and processing requirements of the temporal join operator. Based on different sort-orders on T_S and/or T_E of one or both input relations, they proposed three partial nested-loops temporal join algorithms which,

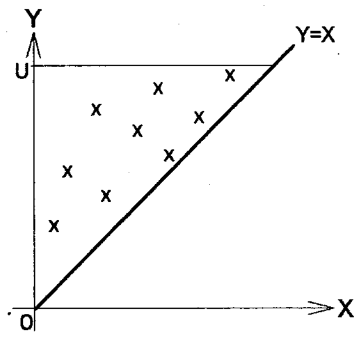


Figure 3.1.1 Spatial rendition of time intervals

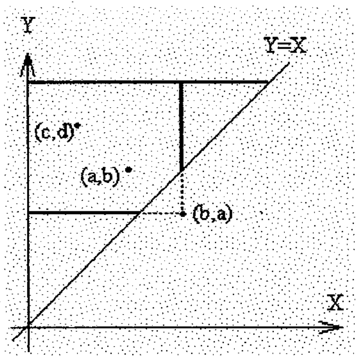


Figure 3.1.2 (c,d) strongly dominates (b,a)

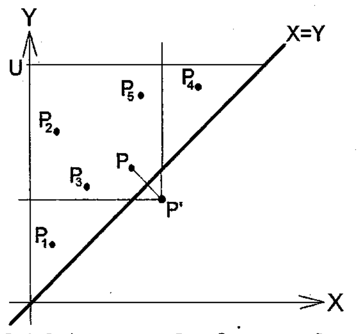


Figure 3.1.3 An example of strong dominance searching problem

unlike the conventional nested-loops join algorithms, requires only partial scan of the inner and/or outer relations. Obviously, performances of their methods depend on the average scan length through the relations. In addition, seven nested-loops like temporal join algorithms were proposed by Rana and Fotouhi [1993] to minimize the number of unnecessary comparisons. Unfortunately, the algorithms assumed that the smaller relation fits in memory. Moreover, no performance analysis was presented. Chen et al. [1994] also proposed a nested-loops like algorithm with an efficient data structure introduced in [Overmas 1985] for solving the line segment intersection searching problem as an index for the inner relation. It was pointed out that their method doesn't need any scan or partial scan on the inner relation. However, their method is not suitable for temporal joins in a dynamic environment for the index structure is a static structure. Further, it must assume that the index structure fits in memory.

Leung and Muntz [1990], on the other hand, considered stream processing techniques for temporal joins. Several sort-merge based algorithms for various temporal joins and semijoins are proposed and their workspace requirements for various data sort orderings are discussed. Unfortunately, their methods involve additional house-keeping cost.

Partition-based algorithms have also been studied recently. The differences between these algorithms have to do with how the partitions of the relations are determined. In Leung and Muntz's method [1994], operand relations are range-partitioned on the start or end timestamp of the tuple. In Soo, et al.'s method [1994], time line is range-partitioned into n nonoverlapping intervals and a tuple (of both relations) must appear in the i-th partition if its timestamp overlaps the i-th interval. Both the above methods require determining the partition to which a time interval belongs, which incurs severe partitioning overhead. Further, both methods need to replicate some tuples either statistically or dynamically. This introduces both storage and management overhead. Based on a time-space mapping scheme using both timestamp and time interval of the tuple. Lu et al. [1994] proposed another partition_based strategy. In their method, time intervals are mapped to points in a two dimensional space and the space are partitioned into subspaces. Data tuples are clustered based on their mapping in the space. As a result, the overhead of partitioning is avoided and the join performance is improved. Besides, they used a spatial indexing technique introduced in [Shen et al. 1994] to support direct access to the stored partitions.

3. A Nested-Loops Temporal Join Algorithm

3.1 The Equivalence of the Interval Intersection Searching Problem and the Strong Dominance Searching Problem

Consider the following time interval intersection searching problem : given a set V of time intervals I_i=<a_i, b_i>, i=1,2,...,n, and a query time interval I=<a, b>, we want to determine all time intervals in V that intersect I. Suppose each time interval < a_i, b_i >, $1 \leqq i \leqq n$, is mapped to the point (a_i, b_i) in the plane. Since $0 \leqq a_i < b_i \leqq U$, where $U=\max\{b, b_i \mid 1 \leqq i \leqq n\}$, ($a_i$, b_i) lies in the triangular region Ω bounded by the lines x=0, y=U and y=x (see Figure 3.1.1). We call point (a_i, b_i) the spatial rendition of time interval <a_i, b_i> and the region Ω the spatial rendition of V throughtout this paper.

Since, time intervals <c, d> and <a, b> intersect each other if and only if $c \leqq b$ and $a \leqq d$, (c, d) lies in the north-west quadrant from the point (b, a). Accordingly, < c, d > intersects < a, b > if and only if (c, d) lies in the region bounded by the lines : x=0, x=b, y=a, y=U and y=x (see Figure 3.1.2). In this case, we say that (c, d) strongly dominates (b, a).

Consequently, our time interval intersection searching problem is equivalent to the following strong dominance searching problem : given a set of points (a_i,b_i), where a_i, b_i are integers and $0 \leqq a_i < b_i \leqq U$, for $1 \leqq i \leqq n$, store them such that for any query point (a, b), where a, b are integers and $0 \leqq a < b \leqq U$, we can efficiently determine those points among (a_i, b_i), $1 \leqq i \leqq n$, that strongly dominates (b, a). Consider, for instance, five points Pi , $1 \leqq i \leqq 5$, and a query point P as shown in Figure 3.1.3. It is easy to see that among the five points, only P2 ,P3 ,and P5 strongly dominate P'.

3.2 A Dominance Searching Based Nested-Loops Temporal Join Algorithm

Give two point P=(a,b) and $p' = (a', b')$, we say that P dominates P if $a \geq a'$, $b \geq b'$ and $P \neq P'$ Overmars [1988] presented an efficient data structure to solve the following dominance searching problem: store a set of points V such that for any query point P we can efficiently determine those points in V that dominate P. The space required by the structure is O(n) and the total time needed is $O(\log\log u + k)$, where u is the maximum of y-coordinates of all points and k is the total number of reported answers.

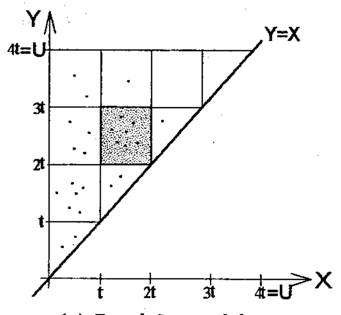
Overmars' data structure can be easily modified to fit our strong dominance searching problem described in the preceding subsection. We call this modified data structure the SDS (strong dominance searching) structure. By exploiting the SDS structure for determining the set of all matching tuples of a temporal join, in this section, we propose a new nested-loops temporal join algorithm which can be formally described as follows.

```
Algorithm DSTJ (Dominance Searching Based Temporal Join)
Input : Two temporal relations R and S
Output : The temporal join of R and S
Preprocessings :
    1. Associate each time interval $< s.T_S, s.T_E >$ of
       the inner relation S, a point $(s.T_S, s.T_E)$ in
       the X-Y plane.
    2. Construct a SDS structure for the set of points
       $(s.T_S, s.T_E)$ for all $s \in S$.
Steps :
    1. Read next tuple r from the outer relation R
       until EOF(r).
    2. Determine, through the use of the SDS
       structure, the set M of all points which
       strongly dominate $(r.T_E, r.T_S)$.
    3. For each tuple s in S with the spatial rendition
           $(s.T_S, s.T_E)$ in M do
           read s
           output the temporal join of r and s
       end for.
    4. Go to Step 1.
```

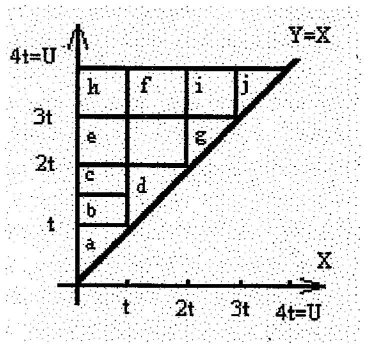
3. 3 A Cluster Scheme and an Index Scheme

In this section, we propose a cluster scheme, called the SP (Spatial Partition) cluster scheme, and an index scheme, called the TP (Time Polygon) index scheme, to support Algorithm DSTJ to efficiently access the required matching tuples.

Our partition_based cluster strategy can be described as follows. Let R be a temporal relation. Based upon mapping each time interval $< r.T_S, r.T_E >$, $r \in R$, to a point $(r.T_S, r.T_E)$, call the spatial rendition of $< r.T_S, r.T_E >$, in the plane; R can be represented as a set of points in the triangular region Ω, called the spatial rendition of R, bounded by x=0, y=U, and y=x, where $U = \max\{r.T_E | r \in R\}$. Then we partition Ω into $m \cdot (m+1)/2$ subregions by a set of vertical lines : x= 0, x= t1, ···, x= tm= U, and a set of horizontal lines : y= 0, y= t1, ···, y= tm= U. For simplicity, we assume that $t_i - t_{i-1} = t$ for $1 \le i \le m$. For instance, Figure 3.3.1(a) depicts a partitioning with m= 4 and 10 partitions. In our SP cluster scheme, tuples of R are clustered according to the time attributes T_S and T_E in such a way that they are stored in the same page if their spatial renditions falling within the same partition. Note that since the spatial renditions of tuples may not be uniformly distributed, a partition in Figure 3.3.1 may consists of a number of pages or one page may cover a number of partitions. For ease of reference, we call the partitions formed by lines $x = it$ and $y = it$, $1 \le i \le m$, logical partitions, while data pages are called physical partitions (see Fig 3.3.1(b)). Further, since it has been shown in Section 3.1 that all points representing matching tuples to a query tuple with time interval <a, b> lie in the north-west quadrant from the point (b, a),



(a) Logicl partitions



(b) Physical partitions

Fifure 3.3.1 An example of the proposed cluster scheme

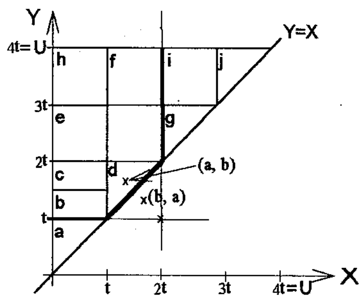


Figure 3.3.2 The determination of qualified data pages of a query tuple

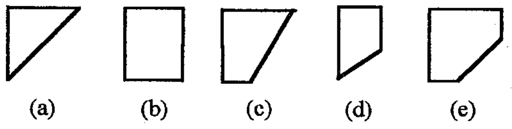


Figure 3.3.3 Five well-formed shapes of TP polygons

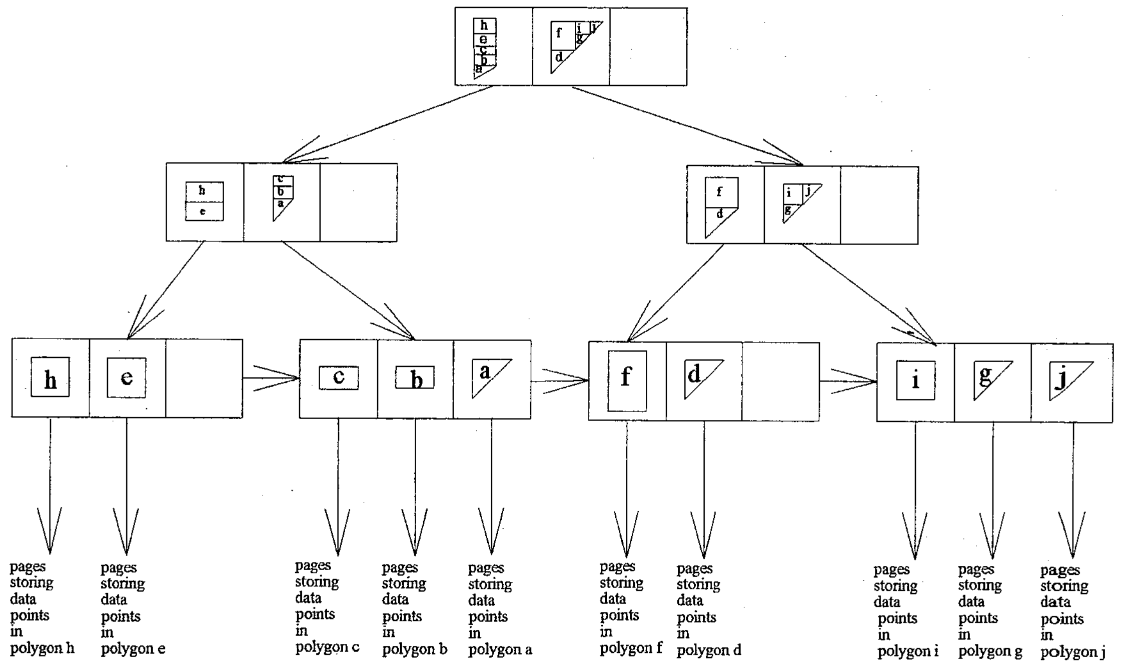

pages storing data points in polygon h
pages storing data points in polygon e
pages storing data points in polygon c
pages storing data points in polygon b
pages storing data points in polygon a
pages storing data points in polygon f
pages storing data points in polygon d
pages storing data points in polygon i
pages storing data points in polygon g
pages storing data points in polygon j

Figure 3.3.4 A demonstration of TP-tree

together in our cluster scheme.

When a temporal relation is SP-clustered, it is easy to determine the data pages qualified by a query tuple, i.e., the data pages containing at least one matching tuple of the query tuple. Suppose the query time interval is <a, b> We first compute $\left\lceil \frac{b}{t} \right\rceil t$ and $\left\lfloor \frac{a}{t} \right\rfloor t$, respectively. Then the qualified data pages are those contained in the region G bounded by the line x=0, $x = \left\lceil \frac{b}{t} \right\rceil t$, $y = \left\lfloor \frac{a}{t} \right\rfloor t$, y=U, and y=x. An illustrated example is given in Figure 3.3.2.

To support direct access of matching pages from a SP-clustered temporal relation, we propose an indexing technique. Our index scheme is similar to that of Lu et al. [1994] and is based upon the TP-tree suggested by Shen et al. [1994]. The TP-tree is a B^+-tree like index structure, the leaf nodes contain pointers pointing to the data pages. However, unlike the conventional B^+-tree whose index nodes contain numeric or alphabetical key values, an index node of TP-tree represents the subspace comprised by the data pages in the subtree rooted at the node. In detail, an internal node of TP-tree has entries of the following format (child-pointer, polygon) where child-pointer points to a child node and polygon describes the entire data space of the child node, while the leaf entry is of the form (page-pointer, polygon, leaf-pointer) where page-pointer points to a data page storing points in the polygon and leaf-pointer points to a succeeding leaf node. The polygon of the root entry is the spatial rendition of a temporal relation, the polygon stored in other nodes are constrained to the five well-formed shapes illustrated in Figure 3.3.3. When a polygon contains more data points than a page can accommodate, it has to be partitioned by a horizontal line or a vertical line into two polygons of well-formed shapes, called buddies of the polygon; and only buddies of a polygon can be merged together so that the resultant polygon can still have a well-form shape. Figure 3.3.4 demonstrates a TP-tree of a temporal relation with 10 data pages where we suppose each node can accommodate three entries.

To access data pages qualified by a query region G, we first search the TP-tree for the leaves whose polygons are subregions of G. Then the page-pointers in the leaf entries are followed to retrieve the required data pages. The search algorithm of a TP-tree can be described as follows.

Algorithm STP (Search on a TP-tree)

Input : A TP-tree T of a data space and a query region G.

Output : Pointers of data pages qualified by G.

Steps :

```
1. Let P be a pointer points to a node of T
     P <- root
2. Let e be a entry in the node pointed by P
     If e is a leaf entry
           then return the page-pointer stored
           in e
3. If the polygon in e intersects G
           then P<- child-pointer of e and
                go to Step 2.
```

When two temporal relations are SP-clustered and TP-indexed, Algorithm DSTJ can be elaborately redescribed as follows.

Algorithm EDSTJ (Elaborate Dominance Searching Based Temporal Join)

Input : Two SP-clustered and TP-indexed temporal relation R and S with the same partitioning time interval.

Output : The temporal join of R and S.

Preprocessings :

```
1. Construct the SDS structure for the inner relation S.
2. Built a TP-tree of spatial rendition of the inner relation S.
```

Steps :

```
1. Read next tuple r from the outer relation R until EOF(r).
2. Call STP(G), where G is the region bound by
   x=0, $x=\left\lceil \frac{r.T_E}{t} \right\rceil t$, $y=\left\lfloor \frac{r.T_S}{t} \right\rfloor t$, y=U, and
   y=x, to obtain a set D of page-pointers.
3. For each page P pointed by a pointer of D
       for each tuple s in P
           if r and s match each other
               then output the temporal join of r and s
       end for
   end for.
4. Go to Step 1.
```

4. A Spatially Partition_Based Temporal Join Algorithm

One disadvantage of Algorithm EDSTJ is that the SDS structure of the inner relation must fit in primary memory. In addition, since SDS structure is a static structure, Algorithm EDSTJ is not suitable for dynamic temporal databases for which tuples may frequently added into or delete from them. In order to efficiently process temporal joins of large temporal relations in a dynamic environment, in this section we propose a spatially partition_based temporal join algorithm without using any index for the operand relations.

4.1 A Spatially Partition_Based Temporal Join Algorithm

Our algorithm is based upon the SP-cluster structure and TP-index structure introduced in the preceding chapter. Each operand relation is first partitioned and clustered on its time attributes according to the SP-cluster scheme. Each logical partition under

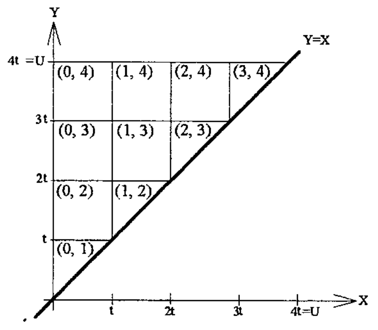


Figure 4.1.1 Logical partitions and their partition_id's of a temporal relation using a SP-cluster scheme with partitioning time interval length t

the SP-cluster scheme is uniquely identified by an ordered pair (i, j) if it is bounded left by the line x= it and above by the line y= jt, where t is the length of partitioning time interval. Accordingly, given a tuple with a time interval <a, b>, its partition_id can be easily determined as the ordered pair $(\left\lfloor \frac{a}{t} \right\rfloor, \left\lceil \frac{b}{t} \right\rceil)$. Figure 4.1.1 illustrates logical partitions and their partition_id's of a sample relation using a SP-cluster scheme with partitioning time interval length t.

Furthermore, suppose both operand relations are clustered with the same partitioning time interval. (It is pointed out in [Lu et al. 1994] that partitioning two relations using the same time interval performs better than using different intervals.) Then for a given partition of one relation, it is easy to determine all the partitions of the other relation that are joinable (i.e., containing at least one tuple that matches some tuples of the given partition) with this partition by applying the following algorithm.

Algorithm CJP (Compute Joinable Partition_id's)

Input : Two SP-clutered and TP_indexed relations R and S; a partition P^* of R with partition_id (c,d).

Output : Id's of partitions of S that are joinable with P^*.

Steps :

```
For y= (c+1)t to U
    for x= 0 to (d-1)t and x<y
        output (x, y)
    end for
end for.
```

For instance, suppose relations R and S are partitioned and clustered with the same partitioning time interval as shown in Figure 4.1.2. Then the id's of S-partitions that are joinable with the R-partition having id (1, 3) are (0,2), (1, 2), (0, 3), (1, 3), (2, 3), (0, 4), (1, 4) and (2, 4).

After clustering of tuples, we build a TP-tree for indexing data pages of each relation. Now, suppose the available memory capacity is m data pages. To

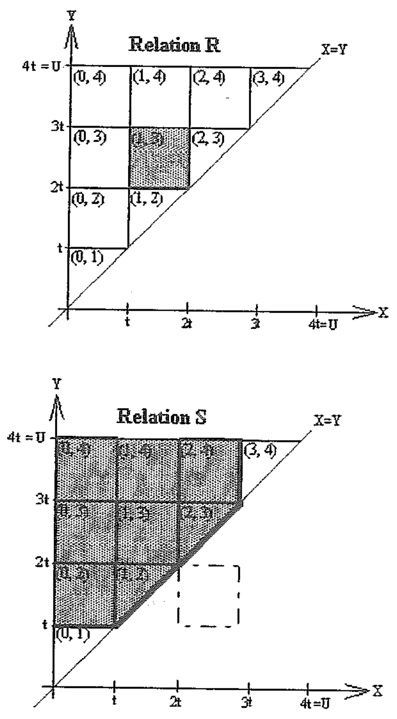


Figure 4.1.2 Id's of S-partitions that are joinable with the R-partition having id (1,3)

temporally join two SP-clustered and TP-indexed relations, m data pages of the outer relation are first read, in batch, into memory according to the order they stored in leaves of the TP-tree. For each data page read in, we compute the pages from the inner relation that are joinable with this outer page by applying Algorithm CJP. Finally, the TP-tree of the inner relation is used to retrieve the desired pages into memory to perform the join. Our approach can be formally expressed as the following algorithm.

```
Algorithm SPTJ (Spatially Partition_Based
    Temporal Join)
Input : Two SP-clutered relations R and S, and their
        TP-trees TP_R
     and TP_S.
Output : The temporal join of R and S.
Steps :
    1. Read in m pages of relation R in batch.
    2. For each logical partition P* covered by the m
       R-pages in Step 1 do
          compute JP, the set of S-partitions that are
          joinable with P*, by calling Algorithm
          CJP(P*)
             for each logical partition P** in JP
             do
                read in P**
                   for each tuple pair in P* and
                   P** do
                   if two time intervals
                   intersect
                         then output the join of
                         the tuple pair
                   end for
             end for
       end for.
```

4.2 Some Discussions

It should be emphasized that Algorithm SPTJ can be used to process temporal join in a dynamic temporal database environment. To insert a new tuple into a relation when applying the algorithm, we first determine the partition to which the spatial rendition of the tuple belongs, then search the TP-tree of the relation for the data page where the partition resides, and finally added the tuple into the page. If a partition overflows after inserting a new tuple, it splits into two partitions of well-formed shapes, and a new page is acquired to accommodate the new partition. On the other hand, a tuple can be deleted from a relation by first determining the data page where this tuple resides and remove it from the page. If the page remains nonempty, the deletion is finished. Otherwise, the page is released and the partition containing the spatial rendition of the deleted tuple is merged with its buddy partition into a new well-formed partition.

Among other things, it was pointed out in [Lu et al. 1994] that since temporal join is a non-equijoin operation, a partitions from one relation must be compared with several partitions from the other relation in a partition_based temporal join; hence, the partition_based temporal join doesn't perform very well compared to the nested-loops join, the saving in reducing the number of comparisons may not be enough to offset the overhead incurred by partitioning. In other words, the partition_based temporal join will not be very attractive unless the overhead of partitioning can be reduced or avoided. When applying Algorithm SPTJ, tuples, when they are generated, are clustered into partitions based on their spatial renditions. Accordingly, the partition phase is avoided and no overhead of partitioning is incurred. Therefore, the join performance is significantly improved. Lu et al. [1994] also proposed a spatially partition_based temporal join algorithm in which each time interval is mapped to the point (a, b-a) in the plane. Their method doesn't involve partitioning overhead either. However, comparing with their method, our method is simpler, less computation and easier to implement.

5.Conclusions and Future Research Problems

In this paper, based upon a dominance searching technique proposed by Overmars [1988],we have proposed a static nested-loops temporal join algorithm. An efficient cluster scheme, called the SP-cluster scheme, and an efficient indexing scheme, called the TP-index scheme have been presented as well to promote the performance of the proposed algorithm.

In order to efficiently process temporal joins of large or very large temporal relations in a dynamic environment, we have proposed another partition_based temporal join algorithm in which tuples are partitioned and clustered by a SP-cluster scheme where they are inserted into the relation. It has been shown that the proposed method can easily manipulate various operations in a dynamic temporal database environment, such as search a relation for a tuple, insert or delete a tuple into a relation and so on. Moreover, since tuples, when they are inserted into a relation, are clustered into partitions based on their spatial renditions, the partitioning overhead is avoided. Therefore, the join performance is significantly improved.

Though it is seen that our algorithms have a number of merits over other methods suggested previously , a performance study is still very conductive. We are now conducting some performance studies of our methods.

In addition, our future research problems include (1) designing more powerful data structures to speed up the determination of matching tuples, (2) designing more efficient clustering and indexing techniques to help direct access of matching tuples, and (3) designing new efficient join strategies other than the nested-loops strategy, the sort-merge strategy, and the partition_based strategy.

REFERENCES

Chen, C. Y. ,Chang, C. C. and Lee, R. C. T. (1994) : "A Line Segment Intersection Based Temporal Join," Proceedings International Symposium on Advanced Database Technologies and Their Integration, Nara, Japan, October 1994, pp.183-187.

Elmasri, R. and Wuu, G. (1990) : "A temporal model and query language for temporal databases," Proceedings of the Sixth International Conference on Data Engineering, Los Angeles, CA, April 1990, pp.76-83.

Gunadhi, H. and Segev, A. (1991) : "Query Processing Algorithms for Temporal Intersection Joins," Proceedings of the Seventh International Conference on Data Engineering, Kobe, Japan, April 1991, pp.336-344.

Leung, T. and Muntz, R. (1990) : "Query Processing for temporal Databases," Proceedings of the Sixth International Conference on Data Engineering, Los Angeles, CA, April 1990, pp.200-208.

Leung, T. and Muntz, R. (1992) : "Temporal Query Processing and Optimization in Multiprocessor Database Machines," Proceedings of the 18th International Conference on Very Large Data Bases, Vancouver, Canada, August 1992, pp.383-394.

Lu, H. , Ooi, B. and Tan, K. (1994) : "On Spatially Partitioned Temporal Join," Proceedings of 20th International Conference on Very Large Databases, Santiago, Chile, September 1994, pp.546-557.

McCreight, E. (1985) : "Priority Search Trees," SIAM Journal on Computing , Vol.14, No.2, 1985, pp.257-276.

Overmars, M. H. (1988) : "Efficient Data Structures for Range Searching on a Grid," Journal of Algorithms, Vol.9, No.2, 1988, pp.254-275.

Rana, S. and Fotouchi, F. (1993) : "Efficient Processing of Time-join in Temporal Data Bases," Proceedings of the 3rd International Symposium on Database Systems for Advanced Applications, Taejon, Korea, April 1993, pp.427-432.

Segev, A. and Gunadhi, H. (1989) : "Event-join Optimization in Temporal Reational Data bases," Proceedings of the 15th International Conference on Very Large Data Bases, August 1989, pp.205-215.

Segev, A. and Shoshani, A. (1988) : "The Representation of a Temporal Data Model in the Relational Environment," Lecture Notes in Computer Science, Vol.339, M. Rafanelli, Klensin, J. C. and Svensson, P. (eds.), Springer-Verlag, 1988, pp.39-61.

Shen, H. , Ooi, B. and Tan, K. (1994) : "The *TP*-Index: A Dynamic and Efficient Indexing Mechanism for Temporal Databases," Proceedings of Tenth International Conference on Data Engineering, April 1994, pp.274-281.

Shin, D. K. and Meltzer, A. C. (1994) : " A New Join algorithm," SIGMOD RECORD, Vol.23, No.4, December 1994, pp.13-18.

Snodgrass, R. (1987) : "The Temporal Query Language TQuel," ACM Transaction on Database Systems, Vol.12, No.2, June 1987, pp.247-298.

Soo, M. , Snodgrass, R. and Jenson, C. (1994) : "Efficient Evaluation of the Valid-time Natural Join," Proceedings of the Tenth International Conference on Data Engineering, April 1994, pp.282-290.

Willard, D. (1983) : "Long-logarithmic worst-case range queries are possible in space $\theta(n)$," Information Processing Letters, Vol.17, No.2, 1983, pp.81-84.