Efficient Management of 2-d Interval Relations

Nikos Lorentzos and Yannis Manolopoulos

Informatics Laboratory, Agricultural University of Athens, 118 55 Athens, Greece.
Department of Informatics, Aristotle University, 54006 Thessaloniki, Greece.

Abstract. We identify a number of problems concerning the management of interval data and propose efficient algorithms in the case of 2-dimensional *interval* relations. The approach is of practical importance and has many applications, one of which is spatiotemporal databases.

1 Introduction

The term *interval* is quite generic. *Time intervals* mark the duration of events (the lifespan of a person). *Alphabetic intervals* have many applications (family names in the range A-C). Given the wide use of intervals, their handling is of major importance. However, there is a number of problems which relate to their management. Such of them were initially identified in research in *temporal databases*. In particular, the necessity to support temporal data led to the formalisation of many distinct temporal extensions to the relational model [1]. In spite however of the major differences between the various modelling approaches, one characteristic, common to almost all of them, is that the ordinary *projection*, *set-union* and *set-difference* operations are adapted appropriately, in all of them, so as to apply appropriately to data incorporating time intervals. Next, it was identified that the same problems arise in the management of certain types of *spatial data* [2, 3], and this gave recently rise in research in *spatiotemporal databases* [4].

The *Interval-Extended Relational Model* (IXRM) was defined to handle them in a uniform way. In this paper we investigate the properties of the IXRM operations and propose efficient algorithms for the above operations. Our work restricts to relations with two *pure* interval attributes. The algorithms have been based on the geometric interpretation of the contents of pure interval attributes and improve substantially the time and space requirements. The remainder of this work is as follows: In section 2 we identify certain problems concerning the management of interval data. In section 3 we present briefly the IXRM and investigate the properties of its operations. In section 4 we make use of these properties and provide efficient algorithms. Conclusions are drawn in the last section.

2 Motivation

In this section we demonstrate the problems concerning the projection, insertion and deletion of interval data. Commercial DBMSs do not support them directly.

Projection: In relation ASSIGNMENT (figure 1) we record the history of employee assignments to projects. The query "list the time intervals during which each employee was assigned to some project" requires to project out the second attribute of ASSIGNMENT. If the standard *projection* operation is used to this end, A1 (figure 1) will be obtained. In contrast, the user would rather obtain A2 (figure 1). We say that A2 is a *normalised* relation, to denote that it does not contain adjacent or overlapping intervals, which data duplication. For example, the fact that John was assigned to some project for each of the dates in [d20,d50), is implicit in A1, from both its first and third tuple.

Assignment

Name	Proj	Time
John	P1	[d10,d50)
John	P1	[d80,d120)
John	P2	[d20,d80)
John	P3	[d80,d100)
Alex	P1	[d30,d50)
Alex	P2	[d70,d150)

A1

Name	Time
John	[d10,d50)
John	[d80,d120)
John	[d20,d80)
John	[d80,d100)
Alex	[d30,d50)
Alex	[d70,d150)

A2

Name	Time
John	[d10,d120)
Alex	[d30,d50)
Alex	[d70,d150)

Fig. 1. Relations with interval data.

LAND

Pno	Depth	Time	pH	
1	[0,20)	[d0,d80)	8.0	r1
1	[0,100)	[d120,d200)	8.0	r2
1	[30,120)	[d0,d80)	8.0	r3
1	[0,30)	[d80,d120)	8.2	
2	[0,60)	[d10,d50)	8.3	
2	[60,100)	[d30,d90)	8.3	

L

Pno	Depth	Time	pH	
1	[70,140)	[d40,d160)	8.0	r2
2	[60,120)	[d10,d40)	8.3	

Fig. 2. Relations with two pure interval atributes.

Similar problems also arise in relations with more than one pure interval attribute. For example, LAND (figure 2) has two such *attributes*, Depth, Time, of an *integer, time interval* type, respectively. A non-trivial projection of LAND on a set of attributes which include either Depth or Time will yield a non-normalised relation. Therefore, the projection of a relation with pure interval attributes has to be replaced by some *normalisation* operation, before the result relation is presented to the user.

Data Insertion: Assume that we want to insert into LAND the contents of L (figure 2). Using the standard insertion operation, this will result in relation LAND1 (figure 3). LAND1 is *non*-normalised (for example the soil pH at depth 70 on date d40, is recorded in both r2 and r4). In fact, we would like to obtain LAND2 (figure 3), which is normalised.

Data Deletion: If we use the *standard* deletion operation, to delete from LAND the contents of L, nothing will actually be deleted, whereas we would like to obtain LAND3 (figure 3).

3 The Interval Extended Relational Model

In this section we describe shortly the IXRM, which overcomes the problems identified in the previous section. Its formalisation can be found in [5].

LAND1

Pno	Depth	Time	pH	
1	[0,20)	[d0,d80)	8.0	r1
1	[70,140)	[d40,d160)	8.0	r2
1	[0,100)	[d120,d200)	8.0	r3
1	[30,120)	[d0,d80)	8.0	r4
1	[0,30)	[d80,d120)	8.2	
2	[0,60)	[d10,d50)	8.3	
2	[60,100)	[d30,d90)	8.3	
2	[60,120)	[d10,d40)	8.3	

L

Pno	Depth	Time	pH	
1	[0,20)	[d0,d80)	8.0	t1
1	[30,70)	[d0,d80)	8.0	t2
1	[100,120)	[d0,d160)	8.0	t3
1	[70,100)	[d0,d200)	8.0	t4
1	[120,140)	[d40,d160)	8.0	t5
1	[0,70)	[d120,d200)	8.0	t6
1	[0,30)	[d80,d120)	8.2	
2	[100,120)	[d10,d40)	8.3	
2	[0,60)	[d10,d50)	8.3	
2	[60,100)	[d10,d90)	8.3	

L

Pno	Depth	Time	pH
1	[70,120)	[d0,d40)	8.0
1	[0,20)	[d0,d80)	8.0
1	[30,70)	[d0,d80)	8.0
1	[0,70)	[d120,d200)	8.0
1	[70,100)	[d160,d200)	8.0
1	[0,30)	[d80,d120)	8.2
2	[0,60)	[d10,d50)	8.3
2	[60,100)	[d40,d90)	8.3

A3

Name	Time
John	d10
...	...
John	d119
Alex	d30
...	...
Alex	d49
Alex	d70
...	...
Alex	d149

Fig. 3. Insertion and deletion of interval data. Result of operation unfold.

A *1-dimensional* (1-d) space is a non-empty, finite, totally ordered set D of *points*: $D = d_1, d_2, ..., d_n$. (Without loss of generality, we occasionally start numbering from d_0.) A (1-d) interval over D is defined as

$$[d_m, d_n) = \{d_k | d_m \leq d_k < d_n, d_m, d_n \in D\}.$$

The points d_m, d_n are the boundaries of $[d_m, d_n)$, denoted by $start([d_m, d_n))$, $stop([d_m, d_n))$, respectively. An interval $[d_i, d_{i+1})$, with exactly one point, is called *elementary*. The set of all intervals over D is denoted by I(D). Thus, if DATES=$d_0, d_1, ..., d_{200}$ is a set of consecutive dates then $[d_{10}, d_{21})$ is an interval in I(DATES). If $[d_m, d_n)$ and $[d_p, d_q)$ are two intervals, we define a predicate, *merges*, as

$$[d_m, d_n) \textit{ merges } [d_p, d_q) \text{ if and only if } (d_n \geq d_p \text{ and } d_q \geq d_m)$$

If $D_1, ..., D_n$ are spaces then every subset R of the Cartesian product $I(D_1)$ x ... x $I(D_n)$ is an *n-d interval relation*. Each tuple (element) of R represents an *n-d interval*. An n-d interval $([d_{i1}, d_{i1+1}), [d_{i2}, d_{i2+1}), ..., [d_{in}, d_{in+1}))$ with exactly one *n-d point* $(d_{i1}, ..., d_{in})$, is called *n-d elementary interval*. We further notice that a point d_i can be seen as an interval $[d_i, d_{i+1})$. Therefore, if we use the notation X(D) to denote exclusively either I(D) or D, then every subset of $X(D_1)$ x ... x

$X(D_n)$ is, up to an isomorphism, an interval relation. Hence, all the relations in figures 1-3 are interval relations.

Two operations have been formalised in the IXRM: If R(A,B=I(D)) is a relation then for each tuple $(a,[d_m, d_{n+1}))$ in R, S=*unfold*[B](R) consists of the set of tuples: (a, d_m), (a, d_{m+1}), (a, d_{m+2}), ..., (a, d_n). Conversely, if S consists of the above tuples, but it does not contain any of (a,d_{m-1}) and (a,d_{n+1}), then *fold*[B](S) produces from them the single tuple $(a,[d_m,d_{n+1}))$. More generally, *fold coalesces* two or more overlapping or adjacent intervals into one. Examples of these operations, based on the relations in figures 1-3, are

A3=*unfold*[Time](A1), A3=*unfold*[Time](A2), A2=*fold*[Time](A3),
A2=*fold*[Time](A1)

If $[B_1, ..., B_m]$ is any list of the attributes of a relation $R(A_1, ..., A_n)$, the definitions of *unfold* and *fold* are now extended as follows:

$$unfold[B_1, B_2, ..., B_m](R) = unfold[B_m](\ ... \ (unfold[B_2](unfold[B_1](R))))$$
$$fold[B_1, B_2, ..., B_m](R) = fold[B_m](\ ... \ (fold[B_2](fold[B_1](R))))$$

Another operation, *normalise*, can also be defined in terms of *unfold* and *fold*:

$$normalise[B_1, B_2, ..., B_m](R) = fold[B_1, B_2, ..., B_m](unfold[B_1, B_2, ..., B_m](R))$$

By the definition of *normalise*, *unfold* initially eliminates duplicate tuples. Next *fold* yields a relation which does not contain adjacent or overlapping intervals.

Finally, if R and S be two union-compatible relations, operations *unique points set-union (punion)* and *unique points set-difference (pdiff)* are defined as

$$punion[B_1, ..., B_m](R, S) = fold[B_1, ..., B_m](unfold[B_1, ..., B_m](R) \cup unfold[B_1, ..., B_m](S))$$
$$pdiff[B_1, ..., B_m](R,S) = fold[B_1, ..., B_m](unfold[B_1, ..., B_m](R) - unfold[B_1, ..., B_m](S))$$

By their definition, these operations return a normalised relation. It is easy to show that

$$punion[B_1, .., B_m](R,S) = fold[B_1, .., B_m](unfold[B_1, .., B_m](R \cup S)).$$

Using the above operations, the problems identified in the previous section are faced as follows:

Normalisation: A2 = *normalise*[Time](A1),
LAND2 = *normalise*[Time,Depth](LAND1)
Insertion of Interval Data: LAND2 = *punion*[Time,Depth](LAND, L)
Deletion of Interval Data: LAND3 = *pdiff*[Time,Depth](LAND,L)

The above three operations can be used to handle relations with arbitrarily many pure interval attributes. However, their definition in terms of *unfold*, makes them prohibitively costly. In the sequel we investigate the properties of *unfold* and *fold* and provide efficient algorithms in the next section.

An interval $[d_m, d_n)$ can geometrically be seen as a line segment which is closed to the left and open to the right. Hence, every tuple of an n-ary relation can be seen as an n-d cuboid. Thus, the single tuple of relation R (figure 4(a)) is represented by the orthogonal rectangle WXYZ (figure 4(b)). The 2-d points on sides XY and YZ are not points of the interval. In contrast, the 2-d points in the

rectangle and also those on sides WX and WZ (excluding X and Z) are points of it. If R1=*unfold*[B](R) is now issued, then R1 will consist of four tuples, ([1, 4), i), $i = 1, 2, 3, 4$. Since point i is isomorphic with the interval $[i, i+1)$, each of these tuples can geometrically be interpreted by one of the four adjacent rectangles in figure 4(c). This figure shows that *unfold*[B](R) *splits* each tuple (interval) of a relation R into intervals whose values for attribute B are elementary intervals. Similarly, if we issue next R2=*unfold*[A](R1) we obtain a relation consisting of 2-d elementary points whose geometric interpretation is shown in figure 4(d).

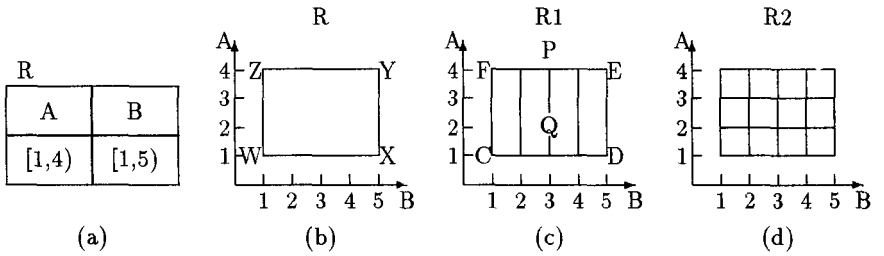

Fig. 4. Geometric interpretation of a 2-d interval and of operations unfold and fold.

It can easily be seen that the relations, whose representation is given in figure 4, satisfy R1=*fold*[A](R2), R=*fold*[B](R1) and, therefore, R=*fold*[A,B](R2). The following properties can be deduced from the definition of *fold* (the reader can found their formal proofs in [6]).

Property 1: *unfold*[B,A](R) = *unfold*[A,B](R).

Property 2: If a relation R(A,B) is sorted with respect to A and B then *fold*[B](R) can be accomplished in one pass, by scanning sequentially the tuples of R.

Property 3: If $R_1, R_2, ..., R_k$ is a partition of R(A,B) such that all the tuples in the same R_i have identical A values whereas tuples in distinct R_i and R_j have different A values, then

$$fold[B](R) = fold[B](R_1) \cup fold[B](R_2) \cup ... \cup fold[B](R_k).$$

Property 4: Let R(A,B=I(D)) be a relation and BP a subset of points in D. If S is the relation obtained from R if all the tuples of R are split with respect to the points in BP, then

$$fold[B](R) = fold[B](S).$$

Property 5: $fold[B](R) = fold[B](unfold[B](R))$.

Property 6: Let R(A,B=I(D)) be a relation and BP be a subset of points in D. If S is the relation obtained from R if all the tuples of R are split with respect to the points in BP, then

$$fold[B](unfold[B](R)) = fold[B](S).$$

Property 7: For any relation R,

$$fold[A,B](unfold[A,B](R)) = fold[A,B](unfold[B](R)).$$

Because of the above properties *normalise* and *punion* are equivalent to

$$normalise[B_1, B_2, ..., B_m](R) = fold[B_1, B_2, ..., B_m](unfold[B_2, B_3, ..., B_m](R))$$
$$punion[B_1, ..., B_m](R,S) = fold[B_1, ..., B_m](unfold[B_2, B_3, ..., B_m](R \cup S)).$$

4 Efficient Algorithms

Using the properties of the previous section, we now provide efficient algorithms for normalise, *punion* and *pdiff*.

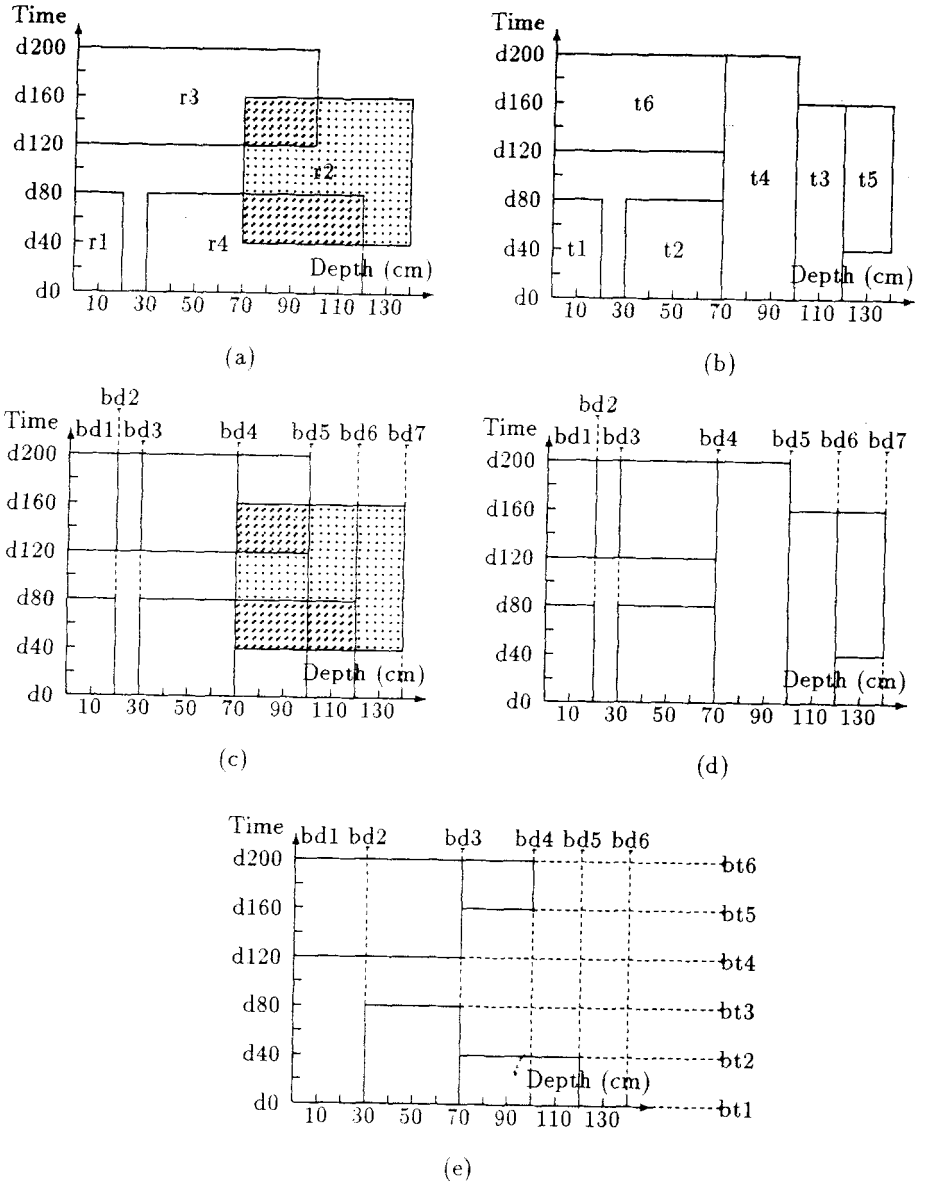

Fig. 5. Geometric interpretation of efficient operations.

4.1 Efficient Normalization of 2-d Interval data

LAND1 (figure 3) is non-normalised, therefore *normalise/fold* has to be applied. From property 3 it suffices to normalise each distinct set in the partition of LAND1. Indeed, LAND1 is already sorted with respect to Pno and pH and the partition is {P,Q,R}, where P consists of the first four tuples, Q consists of the fifth and R contains consists of the last three tuples. The interpretation of P is shown in figure 5(a). Tuple r2, in particular, is dark. The rectangles in r2 which are darker, including their bottom and left sides, denote duplicate points.

If we now issue *normalise*[Time,Depth](P), we shall obtain the first six tuples of LAND2 (figure 3). The interpretation of these tuples is shown in figure 5(b). By using property 7, the number of attributes on which LAND1 has to be unfolded may by reducing by one, that is
normalise[Time,Depth](LAND1) = *fold*[Time,Depth](*unfold*[Depth](LAND1)).
If we issue *unfold*[Depth](LAND1), each of the rectangles in figure 5(a) will be *split* into a large number of rectangles with Depth components of exactly one point each. The same result can be obtained if the rectangles in figure 5(a) are *split* only into sub-rectangles with respect to the boundary points of their Depth values. This *split* is shown in figure 5(c). The set of these boundary points with respect to which the *split* will take place, is *DepthBoundaries*={bd1,...,bd7}. Hence, rather than using *unfold*, we can use this *split*, to finally achieve an equivalent result. Thus, if P1 is the relation having as tuples the rectangles in figure 5(c), then the interpretation of P2=*fold*[Time](P1) is shown in figure 5(d). Finally, P3=*fold*[Depth](P2) shown in figure 5(b), is the interpretation of the normalised relation LAND2. The above procedure has to be repeated for each set in the partition {P,Q,R} of LAND1.

Input : Relation R(NI, A, B), Index(NI) of relation R.
Output: Relation S=*normalise*[A,B](R).
begin
  for each distinct ni value stored in Index(NI) **do**
  begin
    retrieve the subset Ri of R with Ri[NI]=ni
    for each tuple r(ni,a,b) of Ri **do**
    begin
      store the boundaries of b in BBoundaries(B)
      store (a,b) in Temp(A,B)
    end
    sort BBoundaries(B) **and** eliminate any duplicate rows
    split each tuple in Temp(A,B) wrt the points in BBoundaries(B)
    fold(Temp)
    for each tuple r(a,b) in Temp(A,B), write S(ni,a,b)
  end
end

Table 1: Algorithm for *Efficient-normalise* on two attributes.

Let NI be a set of zero or more attribute and let R(NI,A=I(D1),B=I(D2)) be a relation. We assume that two temporary files, BBoundaries(B=D2) and

Temp(A=I(D1),B=I(D2)) can fit in main memory. For simplicity, we assume also that the result of a *split* or *fold* in a main memory structure, is stored in this same structure. The algorithm *normalise*, shown in table 1, calls algorithm *fold*, given in table 2.

Input : Relation Temp(A,B).
Output: Relation *fold*[A,B](Temp).
begin
    sort Temp(A,B) on B, A
    fold Temp(A,B) on A
    sort Temp(A,B) on A, B
    fold Temp(A,B) on B
end

Table 2: Algorithm to fold on two attributes.

4.2 Efficient Insertion of 2-d Interval Data

Consider LAND (figure 2), indexed on NI=[Pno, pH] and assume that we want to insert into L (figure 2). By using property 3, we initially partition L with respect to its distinct NI values and for each set in this partition we consider only the respective partition in LAND. More precisely, L can be partitioned into two sets, of one tuple each. The Pno and pH values of the tuples in the first set (tuple r2) are 1 and 8.0, respectively. For this set, it suffices to consider from LAND only those tuples with the same respective values. However, tuple r1 (figure 5(a)) can in fact be excluded because neither its Time nor its Depth component merges with the respective component of r2. In contrast, tuples r3 and r4 of LAND have to be involved in *punion* because: (i) Their Pno and pH values are identical with the respective values of r2 (also satisfied by r1) and (ii) both their Depth and Time values merge with the respective values of r2 (not satisfied by r1).

Input : Relations R(NI,A,B), Index(NI) of relation R, T(NI,A,B)
(* R is normalised with respect to A, B *)
Output: Relation R=*punion*[A,B](R,T)
begin
    sort T on NI
    for each subset Ti(NI,A,B) of T with identical NI values **do**
    begin
        for each tuple r(ni,a2,b2) of Ti **do**
        begin
            store the boundaries of b2 in BBoundaries(B)
            store (a2,b2) in Temp(A,B)
            for each tuple r(ni,a1,b1) in R such that
                (a1 *merges* a2) and (b1 *merges* b2) **do**
            begin
                store the boundaries of b1 in BBoundaries(B)
                store (a1,b1) in Temp(A, B)

```
            delete r from R
        end
    end
    sort BBoundaries(B) and eliminate any duplicate rows
    split each tuple in Temp(A,B) wrt the points in BBoundaries(B)
    fold(Temp)
    for each tuple r(a,b) in Temp(A,B), write R(ni,a,b)
  end
end
```

Table 3: Algorithm for efficient punion on two attributes.

It is now clear from sub-section 4.1, that if *punion*[Time,Depth](LAND,L) is issued, the resulting relation will contain the tuples t2-t6 (figure 5(b)), in place of tuples r3 and r4 (figure 5(a)). This means that tuples r3 and r4 have to be deleted from LAND and tuples t2-t6 will next have to be inserted into LAND. Furthermore, since only the tuples r2-r4 need be split, the set of the Depth boundary points with respect to which the split will take place, is $DepthBoundaries$ = {bd1,bd3,...,bd7} (figure 5(c)).

Finally, the tuples derived by this split have to be folded on Time and Depth and be inserted into LAND. The same procedure has to be repeated with each set into which L is partitioned. The above imply the algorithm for *punion* which is given in table 3.

4.3 Efficient Deletion of 2-d Interval Data

Now assume that we want to delete from LAND the contents of L (figure 2). Since *pdiff* involves *fold*, similarly to *punion*, L has to be partitioned with respect to the values of its tuples on attributes NI. Then *pdiff*[Time,Depth](LAND,L) can be implemented by considering separately each distinct set in this partition, with the respective partition in LAND. The first set in the partition of L consists of tuple r2, only. The respective set of LAND consists again of the tuples r1, r3, r4 from which only r3 and r4 have again to be considered because (i) their Pno and pH values match with the respective values of r2 and (ii) both their Depth and Time values merge with the respective values of r2. As can be seen in figure 5(a), only the portion in dark of tuples r3 and r4 has to be eliminated. Figure 5(e) shows the tuples which LAND will contain in place of r3 and r4, after the execution of *pdiff*. This implies that (i) once r3 and r4 have been identified, they have to be deleted from LAND and (ii) once the tuples in figure 5(e) have been computed, they have to be inserted into LAND. The tuples in figure 5(e) can be computed as follows:

```
Input : Relations R(NI,A,B), Index(NI) of relation R, T(NI,A,B)
(* R is normalised with respect to A, B *).
Output: Relation R=pdiff[A,B](R,T).
begin
    sort T on NI
    for each subset Ti(NI,A,B) of T with identical NI values do
```

```
begin
    for each tuple r(ni,a2,b2) of Ti do
    begin
        store the boundaries of a2 in ABoundaries(A)
        store the boundaries of b2 in BBoundaries(B)
        store (a2,b2) in Temp2(A,B)
        for each tuple r(ni,a1,b1) in R such that
            (a1 merges a2) and (b1 merges b2) do
        begin
            store the boundaries of a1 in ABoundaries(A)
            store the boundaries of b1 in BBoundaries(B)
            store (a1,b1) in Temp1(A,B)
            delete r from R
        end
    end
    sort ABoundaries(A) and eliminate any duplicate rows
    sort BBoundaries(B) and eliminate any duplicate rows
    split each tuple in Temp1(A,B) wrt the points in ABoundaries(A)
    split each tuple in Temp1(A,B) wrt the points in BBoundaries(B)
    split each tuple in Temp2(A,B) wrt the points in ABoundaries(A)
    split each tuple in Temp2(A,B) wrt the points in BBoundaries(B)
    delete from Temp1(A,B) the tuples in Temp2(A,B)
    fold(Temp1)
    for each tuple r(a,b) in Temp1(A,B), write R(ni,a,b)
end
end
```

Table 4: Algorithm for efficient pdiff on two attributes.

Firstly, by comparing figures 5(a) and 5(e), it is clear that, initially, tuples r2-r4 have to be split twice: The first split will be with respect to their Depth boundary points, *DepthBoundaries*={bd1,...,bd6}. The tuples obtained this way, will next have to be split with respect to their Time boundary points, *TimeBoundaries*={bt1,...,bt6}. Thus, two distinct main storage structures have now to be maintained, *DepthBoundaries* and *TimeBoundaries*. The result obtained after the split of r3 and r4 is maintained in a main storage structure, Temp1(A,B). Similarly, the result obtained by the split of r2 is maintained in a main storage structure Temp2(A,B).

Secondly, in order that the dark rectangles of figure 5(c) are eliminated, we have to issue

$$\text{Temp1(A,B)} = \text{Temp1(A,B)} - \text{Temp2(A,B)}.$$

Finally, the result of *fold*[Time,Depth](Temp1), shown in figure 5(e), has to be inserted into LAND. Hence, the algorithm for an efficient *pdiff*[A,B] is the one given in table 4.

5 Conclusions

We considered relations with two interval attributes and examined the problems arising when certain operations are applied to them. At the logical level, it was shown how the relational algebra should be extended to overcome these problems, by using two additional operations, *fold* and *unfold*. We investigated the properties of these operations and presented efficient algorithms for the management of 2-d interval data. Work concerning their implementation can be found in [6]. In our approach we considered indices on non-interval attributes but spatiotemporal indexing approaches have also been proposed (see for example [7, 8, 9] for surveys). However, the indexing of interval data is still an open research problem. Thus, for the problems examined in this work, even more efficient algorithms have to be investigated.

Acknowledgment

This work has been funded by the ESPRIT III Project ORES, 7224.

References

1. L.E. McKenzie and R.T. Snodgrass: *Evaluation of Relational Algebras Incorporating the Time Dimension in Databases*, ACM Computing Surveys 23(4), 501-543, 1991.
2. C.S. Jensen and R.T. Snodgrass: *Extending Normal Forms to Temporal Relations*, TR 92-17, Computer Science Dept., University of Arizona, 1992.
3. S.K. Gadia: *Parametric Databases: Seamless Integration of Spatial, Temporal, Belief and Ordinary Data*, ACM SIGMOD RECORD 22(1) 15-20, 1993.
4. K.K. Al-Taha, R.T. Snodgrass and M.D. Soo: *Bibliography on Spatiotemporal Databases*, ACM SIGMOD Record 22(1), 59-67, 1993.
5. N.A. Lorentzos: *The Interval Extended Relational Model and its Application to Valid Time Databasesr*, in *Theory, Design and Implementation* (ed. A. Tansel et.al.), Benjamin/Cummings, 67-91, 1993.
6. N.A. Lorentzos, A. Poulovassilis and C. Small: *Implementation of Update Operations for Interval Relations*, The Computer Journal, 37(3), 164-176, 1994.
7. Y. Manolopoulos and G. Kapetanakis: *Overlapping B+trees for Temporal Data*, Proc. 5th JCIT Conference, 491-498, 1990.
8. H. Samet: *The Design and Analysis of Spatial Data Structures*, Addison-Wesley, Reading MA, 1990.
9. C. Kolovson: *Indexing for Historical Databases*, in *Theory, Design and Implementation* (ed. A. Tansel et.al.), Benjamin/Cummings, 418-432, 1993.