Noname manuscript No.
(will be inserted by the editor)

Querying now-relative data

Luca Anselma, Bela Stantic, Paolo Terenziani,
Abdul Sattar

Received: 20/7/2012 / Accepted: date

Abstract *Now-relative* temporal data play an important role in most temporal applications, and their management has been proved to impact in a crucial way the efficiency of temporal databases. Though several *temporal relational* approaches have been developed to deal with now-relative data, none of them has provided a whole *temporal algebra* to query them. In this paper we overcome such a limitation, by proposing a general algebra which is polymorphically adapted to cope with the *MAX* "reference" approach, and with our *POINT* approach (and that is easily adaptable to cope with other relational approaches to now-relative data in the literature, such as the *NULL* and the *MIN* ones). Besides being general enough to provide a query language for several approaches in the literature, our algebra has been designed in such a way to satisfy several theoretical and practical desiderata: *closure* with respect to representation languages, *correctness* with respect to the "consensus" *BCDM* semantics, *reducibility* to the standard non-temporal algebra (which involves *interoperability* with non-temporal relational databases), *implementability* and *efficiency*. Indeed, the *experimental evaluation* we have drawn on our implementation has shown that only a slight overhead is added by our treatment of now-relative data (with respect to an approach in which such data are not present).

L. Anselma
Dipartimento di Informatica, Università di Torino, Italy
E-mail: anselma@di.unito.it

B. Stantic
Institute for Integrated and Intelligent Systems, Griffith University, Brisbane Australia
E-mail: b.stantic@griffith.edu.au

P. Terenziani
Dipartimento di Informatica, Università del Piemonte Orientale "Amedeo Avogadro", Alessandria, Italy
E-mail: paolo.terenziani@mfn.unipmn.it

B. Stantic
Institute for Integrated and Intelligent Systems, Griffith University, Brisbane Australia
National ICT Australia, Brisbane Australia
E-mail: a.sattar@griffith.edu.au

Keywords Temporal relational databases · Now-related data · Querying bitemporal data · Algebraic operators · Experimental evaluation

1 Introduction

Temporal data plays an important role in most domains / applications. In such contexts, to be meaningfully interpreted, data must be paired with the time when they occur (*valid time* henceforth). Additionally, in many applications, also the time when data are inserted/deleted in the database (*transaction time* henceforth) must be maintained (e.g., for legal purposes). As a consequence, starting from the 1980s, there is a long tradition of approaches coping with valid and/or transaction time in relational databases. As a matter of fact, more than twenty years of research in the area of relational databases have widely demonstrated that the treatment of time in the relational approach involves the solution of difficult problems, and the adoption of advanced dedicated techniques. *"Two decades of research into temporal databases have unequivocally shown that a time-varying table, containing certain kinds of DATE columns, is a completely different animal than its cousin, the table without such columns. Effectively designing, querying, and modifying time-varying tables requires a different set of approaches and techniques than the traditional ones taught in database courses and training seminars. Developers are naturally unaware of these research results (and researchers are often clueless as to the realities of real-world application development). As such, developers often reinvent concepts and techniques with little knowledge of the elegant conceptual framework that has evolved and recently consolidated..."* in [22], Section "Preface", Subsection: "A paradigm shift", page XVIII). Practical examples of the problems to be faced are presented in Chapter 1 of the TSQL2 book [21]. Given the pervasive character of time, great efforts in terms of research were made in order to provide once-and-for-all a general solution to such problems (as opposed to ad-hoc solutions to be independently built in each application coping with time). In this spirit, many extensions to the standard relational model were devised, and more than 2000 papers on temporal databases (TDBs) were published over the last two decades (cf., the cumulative bibliography in [32], the section about TDBs in the Springer Encyclopedia of Databases [16], which includes over 90 entries about TDBs, the entry "Temporal Database" in [16], and the surveys in [15, 27, 19, 11]).

Despite such a wide range of approaches, some "consensus" has been found by the TDB community. TSQL2 [21] is a temporal relational approach coping with bitemporal data (i.e., with both valid and transaction time) which has been defined by a significant number of international researchers in the area. *BCDM* (Bitemporal Conceptual Data Model) [12] provides a unifying semantics for TSQL2 and several other approaches in the literature, including [8, 1, 17, 20, 13]. However, several open issues still have to be addressed. One of them is the treatment of *now-relative* data.

Among temporal data, now-relative information play an important role. A temporal piece of data (tuple) is now-relative if one of the following conditions (or both) holds:

– it is part of the current status of the database (i.e., it has been inserted in the past, and has not been deleted yet); in such a case, the ending point of its transaction time is usually set to *now*, to represent the fact that, in the current status, it is part of the database;
– it is currently valid (i.e., the fact it describes holds at the current time and its ending time is unknown); in such a case, the ending point of its valid time is set to *now*, to state that it is currently valid.

An efficient treatment of *now* in temporal databases is very important, since now-relative facts may be very frequent, and are likely to be accessed more frequently. As a matter of fact, it has been shown that the choice of the physical value for *now* significantly influences the efficiency of accessing temporal data [29].

There are two mainstreams in the treatment of now-relative data in TDBs. In the first mainstream, variables (e.g., *now* , $until-changed$, $forever$, ∞, @, and $-$) are introduced, leading to Variable databases [3]. However, Variable databases require a significant departure from the "consensus" relational model (since the domain of SQL1999 values [18] does not support such variables), which is not likely to occur in practice, due to the large commercial investments, both in terms of developed code and expertise (Chapter 1 of [10]). Since we want to adhere to the relational model, we do not follow such a line of research in our approach. In the second mainstream, which we follow in our approach, the relational model has been extended. The literature has concentrated on three basic approaches to denote the value *now* [29]: firstly using $NULL$, secondly using the smallest timestamp (the minimum-value approach *MIN* approach) and thirdly using the largest timestamp supported by the particular RDBMS (*MAX* approach). It has been shown that the *MAX* approach outperforms the $NULL$ and *MIN* approaches [29], so that it will be taken as a reference approach. Recently, we have proposed a novel approach (called the *POINT* approach) to now-relative data [23] which outperforms the *MAX* approach as regards the treatment of range queries.

The problem of querying and updating Variable databases coping with *now* has been widely addressed in the literature (consider, e.g., the survey in [16], and the Concluding section). On the other hand, though the mainstream not using variables has attracted significant attention, in such a mainstream most approaches have focused only on the treatment of range (or slice) queries about now-relative data, i.e., queries selecting tuples holding in a certain period (range queries) or point (slice query) in time. On the other hand, as well as for any other data, it is important to provide an extensive query language also for now-relative data. Specifically, operating at the "semantic" level, it is important to provide a temporal algebra coping also with (bitemporal) now-relative data. To the best of our knowledge, such an extended temporal algebra has not been provided yet by any approach in the literature.

In this paper, we aim to propose a principled approach overcoming such a major limitation of the current literature. In particular, instead of proposing a single temporal algebra, in this paper we propose a polymorphic algebra which, properly instantiated, copes with both the *MAX* and the *POINT* approach (and, possibly, also with $NULL$ and *MIN* approaches). Our polymorphic algebra has been designed in such a way to satisfy several theoretical and practical desiderata:

- **closure**: the algebraic operators must be closed with respect to the (*POINT* or *MAX*) representation language (i.e., given some relations in the *POINT* -or *MAX*– approach, the application of any algebraic operator to them still provides as output a relation that can be expressed in the *POINT* -or *MAX*– representation language);
- **correctness**: the algebraic operators always provide all and only the correct results (i.e., the results that are semantically correct, e.g., considering the *BCDM* semantics);
- **reducibility**: the "reducibility" property is a fundamental one, granting that, if we prune our approach removing the treatment of time (i.e., if we reduce our approach to the treatment of non-temporal attributes only), our temporal relational algebraic operators behave exactly as non-temporal relational algebraic ones. This property grants interoperability with standard non-temporal relational approaches;
- **implementability**: of course, although theoretically grounded, our algebra must be implementable. Notice that the reducibility property is important also to this respect, since it grants that the temporal algebra can be implemented on top of current non-temporal approaches. Additionally, to enforce implementability on top of "standard" non-temporal relational approaches, as discussed above, we do not operate in the field of Variable databases;
- **efficiency**: last, but not least, our approach must be efficient. Indeed, in the paper we also propose an extensive experimental evaluation showing that, both in the cases of the *POINT* and of the *MAX* approach, our temporal algebra does not add any significant overload with respect to the "ideal" (but unrealistic) approach in which now-relative data are not present (since one knows the future end time of all data).

The paper is organized as follows. Section 2 is a preliminary one, in which we briefly describe the *MAX* and the *POINT* approaches to now-relative data, and we discuss the *BCDM* semantic model, which we assume as the reference model to deal with the semantics of our approach, and to prove its correctness. In Section 3 we provide a first new contribution: we provide the formal semantics of the *POINT* and of the *MAX* approaches, on the basis of *BCDM* semantic model. Section 4 describes our algebrae. Specifically, Section 4.1 introduces the basic principles (widely shared by the TDB community) underlying our algebrae, and Section 4.2 briefly sketches *BCDM* reference algebra. Section 4.3 contains the core contributions of our work, namely the description of the polymorphic algebra for the *POINT* and the *MAX* approaches, and the proof of its properties. In particular, the *BCDM* semantic model is used as the basis to prove the semantic correctness of our approach. Finally, in the absence of a "competitor" in the literature, in Section 4.4 we introduce an efficient (but unfeasible) approach to now-relative data (called $NOT-NOW$ approach), which we use as a comparison to study the complexity of our algebrae. Section 5 describes our extensive experimental evaluation, showing that our approach only adds a slight overhead to the optimal (but unfeasible) $NOT-NOW$ approach. Finally, Section 6 contains comparisons and conclusions.

2 Preliminaries

In this section, we set up the stage presenting the background of our approach. The main goal of our approach is that of providing a (algebraic) query language coping with the currently-recognised most efficient representations of *now* in relational databases *without variables*, namely the *MAX* and *POINT* approaches.

First we introduce the MAX and POINT representations. Then, we introduce *BCDM*, a semantic approach which deliberately ignores efficiency issues for the sake of generality and semantic clarity. In Section 3 we will use *BCDM* as a semantic reference for our extensions of the *MAX* and *POINT* approaches, to grant that they are correct, i.e., that (under specific conditions) they provide the same results that would be obtained (much less efficiently) through the unifying and "consensus" *BCDM* approach.

Indeed, the goal of *BCDM* is deliberately semantic clarity (and not computational efficiency). Even the first normal form (1-NF) is not respected, since an arbitrarily large set of bitemporal chronons can be associated with each tuple. Thus, many approaches, including TSQL2 "consensus" approach, have chosen to provide a more compact representation by representing a set of bitemporal chronons with a set of *rectangles* covering them. A single rectangle can be represented through four timestamps (i.e., start and end of transaction and of valid time). 1-NF is then obtained by using as many *value-equivalent* tuples as the number of covering rectangles. This is the approach followed, e.g., by the *MAX* and *POINT* approaches.

2.1 *MAX* and *POINT* representations

The basic idea of the *MAX* representation [30] is very simple: the largest database timestamp (represented by the value *max-value* henceforth) is used in order to denote *now*. Such a value can be used along both (transaction and valid) temporal dimensions.

Example 1 As a trivial example, let us consider a relation *Employee*, recording employee identifiers (*Id* attribute) and departments (*Dept* attribute). Then, the fact that *John* has been employed in the toy department from day 10 to day 12, and that such a piece of information has been inserted into the bitemporal database at transaction time 11, and that at the current time $c_{now} = 14$ is still present in the database (i.e., it has not been deleted) is represented in the *MAX* approach by the tuple

$(John, toy|11, max-value, 10, 13)$

where 11, $max-value$, 10 and 13 are the timestamps denoting the start and end of the transaction and of the valid times, respectively. Notice that, for the sake of homogeneity with the *POINT* approach, we assume that also in the *MAX* approach time periods are represented through time intervals closed to the left and open to the right (so that, in the above example, the end of the valid time is set to 13).

The *POINT* approach is based on the use of the closed-to-the-left, open-to-the-right notation to denote the periods representing the transaction and valid time of tuples [25]. In such a representation, 'degenerate' periods in which the starting point is equal to the ending point are used as a notational device to represent *now* along the transaction-time and/or valid time dimensions.

Example 2 For instance, the piece of information in Example 1 above can be represented in the POINT approach by the tuple

$(John, toy|11, 11, 10, 13)$

where the transaction-time period $[11, 11)$ is the representation used to deal with the period $[11, now]$.

Both in the *MAX* and in the *POINT* approaches, data may be now-relative in both temporal dimensions. Additionally, in line with many approaches in the literature (see [5]), both approaches do not allow *now* to be the value of the starting point of the transaction or the valid time (i.e., a period $[now, 100)$, with 100 greater than the current system time is not allowed either along the transaction- or along the valid time dimensions).

2.2 The *BCDM* semantic model

In this section, we describe *BCDM* (Bitemporal Conceptual Data Model) [12, 21], a unifying *semantic* data model, which has been developed in order to isolate the *core* notions underlying many temporal relational approaches, including the "consensus" TSQL2 one [21].

BCDM does not face issues such as data representation and storage optimization, aiming at a "semantic" approach, in the sense discussed in the below citation, quoted from [21]: *"It is our contention that focusing on data presentation (how temporal data is displayed to the user), on data storage with its requisite demands of regular structure, and on efficient query evaluation, has complicated the central task of capturing the time-varying semantics of data. [...] We therefore advocate a separation of concerns. Time-varying semantics is obscured in the representational schemes by other considerations of presentation and implementation. We feel that the conceptual data model to be discussed shortly [i.e., BCDM] is the most appropriate basis for expressing this semantics."*

In *BCDM*, tuples are associated with valid time and transaction time. For both domains, a limited precision is assumed (the chronon is the basic time unit). Both time domains are totally ordered and isomorphic to the subsets of the domain of natural numbers. The domain of valid times D_{VT} is given as a set $D_{VT} = \{t_1, t_2, \ldots, t_k\}$ of chronons, and the domain of transaction times as $D_{TT} = \{t'_1, t'_2, \ldots, t'_j\} \cup \{UC\}$ (where *UC* -Until Changed- is a distinguished value). In general, the schema of a bitemporal conceptual relation $R = (A_1, \ldots, A_n|T)$ consists of an arbitrary number of non-timestamp attributes $A_1, \ldots, A_n$, encoding some fact, and of a timestamp attribute T, with domain $D_{TT} \times D_{VT}$. Thus, a tuple $x = (a_1, \ldots, a_n \mid t_b)$ in a bitemporal relation $r(R)$ on the schema R consists of a number of attribute values associated with a set of bitemporal chronons $t_{bi} = (c_{ti}, c_{vi})$, with $c_{ti} \in D_{TT}$ and $c_{vi} \in D_{VT}$. The intended meaning of a bitemporal *BCDM* tuple is that the recorded fact is true in the modeled reality during each valid-time chronon in the set, and is current in the relation during each transaction-time chronon in the set. Valid time, transaction-time and atemporal tuples are special cases, in which either the transaction time, or the valid time, or none of them are present.

The *BCDM* model explicitly requires that no two tuples with the same data part (i.e., value-equivalent tuples [21]) are allowed in the same relation. As a consequence, in *BCDM* the full history of a fact is recorded in a single tuple. This choice enhances the semantic clarity of the model. The special value *UC* is used to model *now* along the transaction-time dimension. A special routine makes explicit the semantics of the special value *UC*: as time passes, at each clock tick for each bitemporal chronon (UC, c_v), a new bitemporal chronon (c_t, c_v) is added to the set of chronons, where c_t is the new transaction-time value.

Example 3 For instance, the piece of information in Example 1 above can be represented in *BCDM* as follows:
$(John,\ toy \mid \{(11,\ 10),\ (11,\ 11),\ (11,\ 12),\ (12,\ 10),\ (12,\ 11),\ (12,\ 12),\ (13,\ 10),$
$(13,\ 11),\ (13,\ 12),\ (14,\ 10),\ (14,\ 11),\ (14,\ 12),\ (UC,\ 10),\ (UC,\ 11),\ (UC,\ 12)\}$

An extension of standard relational algebraic operators has been provided to operate on the *BCDM* model, and proper insertion and deletion manipulation operations have been also defined.

In [21] it has been proven that *BCDM* represents the semantic level underlying the TSQL2 approach as well as the approaches of Snodgrass, Jensen, Gadia (more precisely, Gadia-3), McKenzie, and Ben-Zvi [20, 13, 8, 17, 1]. Additionally, it is also shown that *BCDM* is *reducible* and *equivalent* to the standard relational model. These two properties are very important: intuitively, they conjunctively grant that *BCDM* is a consistent extension of the standard (non-temporal) relational model, and that it operates in the same way as the standard model when time is disregarded (inter-operability with the standard relational model is a desirable side effect of these properties).

It is worth stressing that, in the *BCDM* model, *now* is only coped with along the transaction-time dimension, through the introduction of the *UC* special symbol. Moreover, *BCDM* treatment of *now* is not feasible in practical implementations: of course, no approach can update the temporal component of all *current* tuples (i.e., tuples such that the end of the transaction time is set to *UC*) at each tick of the system's clock!

3 Semantics of the *MAX* and *POINT* approaches

In this section we introduce a set of functions which relate the *MAX* and the *POINT* representations to the reference *BCDM* one. Such functions make the underlying semantics of the *MAX* and the *POINT* representations explicit, and will be used in the following sections in order to prove the correctness of our approach. Such functions are a first original contribution of the approach in this paper.

We aim at providing a polymorphic temporal relational algebra (coping with possibly now-relative data), which, in particular, can be properly instantiated to cope with the *MAX* and with the *POINT* approach. To obtain such a goal, we have to abstract from the specific way that the different approaches (*MAX* and *POINT*) use to *represent* now-relative data. The functions *isNowRelative* and *setNow* in the following are used exactly in order to enable us to abstract from the specific implementations. Specifically, the former function takes in input a period (represented by its starting and ending timestamps) and returns true if the period is now-relative; the latter takes in input a (now-relative) period (represented by its starting and ending timestamps) and returns the value used in order represent the value *now* in the specific approach. For the *MAX* approach, the function *isNowRelative* simply tests whether the ending point of the time period corresponds to the largest timestamp $max-value$. Analogously, the function *setNow* sets $max-value$ as the ending point of the time period. For the *POINT* approach, the function *isNowRelative* tests whether the ending point of the time period equals its starting point. Analogously, the function *setNow* sets as the ending point of the time period its starting point.

Function 1 $isNowRelative(Start, End)$

```
case of: $POINT$ approach:
if $Start = End$ then
  return true
else
  return false
end if
case of: $MAX$ approach:
if $End = max - value$ then
  return true
else
  return false
end if
```

Function 2 $setNow(Start)$

```
case of: $POINT$ approach:
return $Start$
case of: $MAX$ approach:
return $max - value$
```

In this paper, we prove some properties of our temporal algebrae for *MAX* and *POINT* representations relating them with a reference approach, i.e., $BCDM$. We define the function $to\_BCDM^{c_{NOW}}(r)$, that converts a relation from either the *MAX* or the *POINT* representation to a $BCDM$ representation; it assumes the value c_{NOW} for *now* (i.e., c_{NOW} is the timestamp that represents the current time).

In the rest of the paper, we adopt the following notation.

Notation 1 *Given a tuple x defined on the schema $R = (A_1, \ldots, A_n \mid T)$, we denote with A the set of attributes $(A_1, \ldots, A_n)$. Then $x[A]$ denotes the values in x of the attributes in A, $x[T]$ denotes the values of the temporal attributes of x. In particular, $x[TT_S]$, $x[TT_E]$, $x[VT_S]$, $x[VT_E]$ denote the starting and ending points of the transaction and valid times of tuples.*

The $to\_BCDM^{c_{NOW}}(r)$ function is very similar to *snap_to_conceptual* in TSQL2 [21] and basically associates with each tuple in the ($POINT$ or MAX) representation the set of all the bi-temporal chronons corresponding to the rectangle represented by its temporal attributes (i.e., start and end of transaction and valid times). To achieve such a result, it adopts the $EXT^{c_{NOW}}$ function, which only operates on the temporal component of a tuple. In turn, $EXT^{c_{NOW}}$ is defined on the basis of two functions: $EXT\_TT^{c_{NOW}}(Start, End)$ and $EXT\_VT^{c_{NOW}}(Start, End)$.

$EXT\_TT^{c_{NOW}}(Start, End)$ operates on transaction times only. It takes in input the starting and the ending timestamp of a transaction-time period and it returns the set of transaction-time chronons included in the period. Notice that, since we assume a representation closed to the left and open to the right, the ending timestamp is not included. If the period is now-relative, the transaction-time chronons also include the chronon c_{NOW}. Moreover, since *BCDM* supports now-relative transaction time, the proper chronon UC is added to the set of transaction-time chronons. $EXT\_VT^{c_{NOW}}(Start, End)$, operating on valid times, is analogous to $EXT\_TT^{c_{NOW}}(Start, End)$, and it is not reported. Since *BCDM* does not support now-relative valid times, $EXT\_VT^{c_{NOW}}(Start, End)$ simply "instantiates" *now* at c_{NOW}.

Function 3 $EXT\_TT^{c_{NOW}}(Start, End)$

```
$chronons \leftarrow \emptyset$
**if** $isNowRelative(Start, End)$ **then**
  $endChronon \leftarrow c\_NOW + 1$
**else**
  $endChronon \leftarrow End$
**end if**
**for all** chronons $chron$ such that $Start \leq chron < endChronon$ **do**
  $chronons \leftarrow chronons \cup \{chron\}$
**end for**
**if** $isNowRelative(Start, End)$ **then** $chronons \leftarrow chronons \cup \{UC\}$
**return** $chronons$
```

The function $EXT^{c_{NOW}}(TT_S, TT_E, VT_S, VT_E)$ takes in input the starting and ending timestamps of a bitemporal period, and returns the corresponding set of bitemporal chronons, which is obtained by performing the cartesian product of the chronons representing the transaction and the valid times (as obtained through the application of the $EXT\_TT^{c_{NOW}}$ and $EXT\_VT^{c_{NOW}}$ functions).

Finally, $to\_BCDM^{c_{NOW}}(r)$ iterates the $EXT^{c_{NOW}}$ functions on (the temporal components of) all the tuples in the input relation r. It is worth noting that, since *BCDM*

Function 4 $EXT^{c_{NOW}}(TT_S,\ TT_E,\ VT_S,\ VT_E)$

```
  return $EXT\_TT^{c_{NOW}}(TT_S,\ TT_E)\ \times\ EXT\_VT^{c_{NOW}}(VT_S,\ VT_E)$
```

does not admit value-equivalent tuples, the *to_BCDM*$^{c_{NOW}}$ function must merge together (i.e., coalesce [2]) all value-equivalent tuples in the *MAX* or *POINT* representations, in order to produce one *BCDM* tuple from each set of value-equivalent representational tuples.

Function 5 *to_BCDM*$^{c_{NOW}}(r)$

```
  $s \leftarrow \emptyset$
  for all $z \in r$ do
    $x[A] \leftarrow z[A]$
    $x[T] \leftarrow EXT^{c_{NOW}}(z[TT_S],\ z[TT_E],\ z[VT_S],\ z[VT_E])$
    for all $y \in r$ do
      if $z[A] = y[A]$ then
        $r \leftarrow r - \{y\}$ {Remove $y$ from $r$ so that it will not be considered in the
        next iterations.}
        $x[T] \leftarrow x[T] \cup EXT^{c_{NOW}}(y[TT_S],\ y[TT_E],\ y[VT_S],\ y[VT_E])$
      end if
    end for
    $s \leftarrow s \cup \{x\}$
  end for
  return $s$
```

Example 4 For instance, applying the *to_BCDM* function to the piece of information in Example 1 gives as a result the tuple in Example 3.

Example 5 As in Example 1, let us consider a relation *Employee*. The fact that *Alice* is employed in the toy department since day 11 and is currently employed at the current time $c_{NOW} = 14$, that such a piece of information has been inserted into the bitemporal database at transaction time 12, and that at the current time it is still present in the database (i.e., it has not been deleted) is represented in the *POINT* approach by the tuple

$(Alice,\ toy \mid 11,\ 11,\ 12,\ 12)$.

In the *MAX* approach it is represented by the tuple

$(Alice,\ toy \mid 11,\ max-value,\ 12,\ max-value)$.

Applying the *to_BCDM*$^{c_{NOW}}(r)$ function with $c_{NOW} = 14$ to a relation consisting of the above tuple (expressed either with the *POINT* or *MAX* approach), gives as a result a relation consisting of the following tuple

$(Alice, toy \mid \{(12,\ 11),\ (12,\ 12),\ (12,\ 13),\ (12,\ 14),\ (13,\ 11),\ (13,\ 12),\ (13,\ 13),$
$(13,\ 14),\ (14,\ 11),\ (14,\ 12),\ (14,\ 13),\ (14,\ 14),\ (UC,\ 11),\ (UC,\ 12),\ (UC,\ 13),\ (UC,\ 14)\})$.

4 Algebra

In this section, we introduce temporal query languages operating on (possibly now-relative) bitemporal data. For the sake of compactness, generality and clarity, we operate at the algebraic level. We first sketch the general principles underlying our extended temporal algebrae, and sketch the algebra of our reference approach (i.e., of the *BCDM* semantic approach). Then, the subsection 4.3 is the core of our contribution, introducing temporal algebraic operators to query data in the *POINT* and *MAX* approaches, and discussing their properties.

4.1 General Principles

Codd designated as complete any query language that was as expressive as his set of five relational algebraic operators, relational union ($\cup$), relational difference ($-$), selection (σ_P), projection (π), and Cartesian product ($\times$) [4]. In this section we propose an extension of Codd's algebraic operators in order to query the data models introduced in Section 2.1.

Several temporal extensions have been provided to Codd's operators in the TDB literature [21], [15]. In many cases, such extensions behave as standard non-temporal operators on the non-temporal attributes, and involve the application of set operators on the temporal parts. This approach ensures that the temporal algebrae are a consistent extensions of Codd's operators and are reducible to them when the temporal dimension is removed. For instance, in *BCDM*, temporal Cartesian product involves pairwise concatenation of the values for non-temporal attributes of tuples and pairwise intersection of their temporal values (see Subsection 4.2).

We ground our approach on such a "consensus" background, extending it in order to cope with the *POINT* and *MAX* representations of now-relative data.

Before proceeding to the definitions, we need to introduce a brief digression about the treatment of value-equivalent tuples. In the temporal relational literature, two tuples are said to be $value-equivalent$ if they have exactly the same values as regards their non-temporal attributes. In the *BCDM* model, it is agreed that (from the abstract -semantic- point of view) each tuple should be equipped with all its temporal information, so that no value-equivalent tuple can coexist in the same relation. While this is a major source of clarity for the abstract model, the different practical logical representations adopted in the literature have used different strategies to cope with value-equivalent tuples (see, e.g., the discussion in the TSQL2 book [21]). For instance, the "consensus" TSQL2 approach admits value-equivalent tuples *at the logical (representation) level*, still retaining the underlying semantics dictated by the *BCDM* model. This choice has a strong impact on the definition of relational algebraic operators. For instance, in *BCDM*, and in the logical representations in which no value-equivalent tuples are admitted, relational union needs to *coalesce* [2] the times of value-equivalent tuples deriving from the relations being united; on the other hand, if value-equivalent tuples are admitted *at the representation level*, temporal relational union can simply put all the input tuples in the result. However, in order to maintain the underlying *BCDM semantics*, in such approaches (such as TSQL2), temporal relational difference needs to consider the fact that value-equivalent tuples may be present in both input relations, so that all their times must be collected before performing the temporal difference between the temporal components of the tuples.

In the following, we have chosen to follow the line of TSQL2 "consensus" approach, thus admitting value-equivalent tuples in our models. The main advantage of such a choice is the fact that temporal projection and union are more efficient (since no operation needs to be performed on the temporal component of tuples), while most of the complexity of the treatment of time is demanded to the relational difference operator.

4.2 *BCDM* algebra

The temporal algebraic operators in *BCDM* follow the "consensus" principles discussed above. Temporal Cartesian product involves pairwise concatenation of the values for non-temporal attributes of tuples and pairwise intersection of their temporal values. Relational union, projection and difference behave in a standard way on non-temporal attributes, and perform union (for relational union and projection) and difference (for relational difference) on the temporal part of value-equivalent tuples. (In *BCDM*, nontemporal selection operates in the standard way on the non-temporal part, neglecting the temporal part). For instance, temporal Cartesian product is reported in the following. In the definition below, we denote by $t[X_1, \ldots, X_k]$ the value of the attributes $X_1, \ldots, X_k$ in the tuple t.

Definition 1 (Temporal Cartesian product $\times^B$) Given two bitemporal relations r and s defined over the schemas $R_1^B = (A_1, \ldots, A_n \mid T)$ and $R_2^B = (B_1, \ldots, B_k \mid T)$ respectively, the temporal Cartesian product $r \times^B s$ is a temporal relation q defined over the schema $R_3^B = (A_1, \ldots, A_n, B_1, \ldots, B_k \mid T)$ and is defined as follows:
$r \times {}^B s = \{t \mid \exists t_r \in r, \exists t_s \in s,$
$t[A_1, \ldots, A_n] = t_r[A_1, \ldots, A_n] \wedge$
$t[B_1, \ldots, B_k] = t_s[B_1, \ldots, B_k] \wedge$
$t[T] = t_r[T] \cap t_s[T] \wedge t[T] \neq \emptyset \}.$

It is worth stressing that, in *BCDM*, it is highlighted that similar operations can be defined, operating on temporal relations in which only the transaction time (or only the valid time) is present. The same consideration also holds as concerns the temporal algebrae we propose in the following. Therefore, for the sake of brevity, in the following we only focus on bitemporal relations, with no loss of generality.

4.3 Temporal algebrae

In this section, we describe a temporal extension of Codd's non-temporal algebraic operators to cope with time, considering both the *MAX* and *POINT* representation of now-relative data. Instead of proposing two different algebrae (one for the *MAX* and one for the *POINT* approach), we capture the generalities between the two, and "hide" the differences through the adoption of the *isNowRelative* and *setNow* functions described above. The specific temporal algebra for the *MAX* (*POINT*) approach can be simply obtained by properly instantiating *isNowRelative* and *setNow*.

Our definition of temporal Cartesian product is reported in the following.

Definition 2 (Temporal Cartesian product $\times^N$) Given two bitemporal relations r and s in the *POINT* or *MAX* approaches, defined over the schemas $R_1^N = (A_1, \ldots, A_n \mid TT_S, TT_E, VT_S, VT_E)$ and $R_2^N = (B_1, \ldots, B_k \mid TT_S, TT_E, VT_S, VT_E)$ respectively, the temporal Cartesian product $r \times^N s$ is a temporal relation q defined over the schema $R_3^N = (A_1, \ldots, A_n, B_1, \ldots, B_k \mid TT_S, TT_E, VT_S, VT_E)$ and is defined as follows:

```
$r \times^N s = \{t \mid \exists t_r \in r, \exists t_s \in s,$
$t[A_1, \ldots, A_n] = t_r[A_1, \ldots, A_n] \wedge$
$t[B_1, \ldots, B_k] = t_s[B_1, \ldots, B_k] \wedge$
if $(isNowRelative(t_r[TT_S], t_r[TT_E]) \wedge isNowRelative(t_s[TT_S], t_s[TT_E]))$
    then $t[TT_S] = max(t_r[TT_S], t_s[TT_S]) \wedge t[TT_E] = setNow(t[TT_S])$
else if $(isNowRelative(t_r[TT_S], t_r[TT_E]) \wedge \neg\, isNowRelative(t_s[TT_S], t_s[TT_E]))$
    then $t[TT_S] = max(t_r[TT_S], t_s[TT_S]) \wedge t[TT_E] = t_s[TT_E] \wedge t[TT_S] < t[TT_E]$
else if $(\neg\, isNowRelative(t_r[TT_S], t_r[TT_E]) \wedge isNowRelative(t_s[TT_S], t_s[TT_E]))$
    then $t[TT_S] = max(t_r[TT_S], t_s[TT_S]) \wedge t[TT_E] = t_r[TT_E] \wedge t[TT_S] < t[TT_E]$
else if $(\neg\, isNowRelative(t_r[TT_S], t_r[TT_E]) \wedge \neg isNowRelative(t_s[TT_S], t_s[TT_E]))$
    then $t[TT_S] = max(t_r[TT_S], t_s[TT_S]) \wedge t[TT_E] = min(t_r[TT_E], t_s[TT_E]) \wedge t[TT_S] < t[TT_E]$
if $(isNowRelative(t_r[VT_S], t_r[VT_E]) \wedge isNowRelative(t_s[VT_S], t_s[VT_E]))$
    then $t[VT_S] = max(t_r[VT_S], t_s[VT_S]) \wedge t[VT_E] = setNow(t[VT_S])$
else if $(isNowRelative(t_r[VT_S], t_r[VT_E]) \wedge \neg\, isNowRelative(t_s[VT_S], t_s[VT_E]))$
    then $t[VT_S] = max(t_r[VT_S], t_s[VT_S]) \wedge t[VT_E] = min(c_{NOW} + 1, t_s[VT_E]) \wedge t[VT_S] < t[VT_E]$
else if $(\neg\, isNowRelative(t_r[VT_S], t_r[VT_E]) \wedge isNowRelative(t_s[VT_S], t_s[VT_E]))$
    then $t[VT_S] = max(t_r[VT_S], t_s[VT_S]) \wedge t[VT_E] = min(c_{NOW} + 1, t_r[VT_E]) \wedge t[VT_S] < t[VT_E]$
else if $(\neg\, isNowRelative(t_r[VT_S], t_r[VT_E]) \wedge \neg\, isNowRelative(t_s[VT_S], t_s[VT_E]))$
    then $t[VT_S] = max(t_r[VT_S], t_s[VT_S]) \wedge t[VT_E] = min(t_r[VT_E], t_s[VT_E]) \wedge t[VT_S] < t[VT_E]\}$.
```

As can be seen, the result of the Cartesian product is a relation whose schema contains both the explicit attributes of r and of s. The timestamps of tuples in q correspond to the intersection of timestamps of the corresponding tuples in r and s, possibly now-relative.

In the definition, the transaction and the valid times of the output tuples are independently defined by cases. For instance, the first case states that, in case the transaction time of both t_r and t_s is now-relative, then the output transaction time is still now-relative along the transaction-time dimension. It is worth stressing that, in all the cases in which the tests $t[TT_S] < t[TT_E]$ or $t[VT_S] < t[VT_E]$ fail, the corresponding tuple is not part of the output. For instance, the second case in the definition states that if the transaction time of t_r is now-relative and the transaction time of t_s is not now-relative, then the resulting transaction time is not now-relative (along the transaction-time dimension) and can be determined as follows: its starting point is the maximum between the two starting times and its ending point is the minimum between the two ending times: since the transaction time of t_r is now-relative, for the semantics of transaction time, certainly the transaction time of t_s ends before *now* and it is the minimum. However, if the result of such an evaluation is a degenerate period whose ending point is not after the starting point (which happens just in case there is no intersection between the two transaction times), then the tuple is not part of the output.

Temporal relational union takes in input the tuples of two bitemporal relations r and s, and gives them in output unchanged both in the non-temporal and temporal part.

Definition 3 (Temporal union $\cup^N$) Given two bitemporal relations r and s in the $POINT$ or MAX approaches, defined over the same schema $R^N = (A_1,\ \ldots,A_n \mid TT_S,\ TT_E,\ VT_S,\ VT_E)$, the temporal union $r \cup^N s$ is a bitemporal relation q defined over the schema R^N, and is defined as follows:

$r \cup^N s = \{t \mid \exists t_r \in r,\ t[A_1,\ \ldots,A_n] = t_r[A_1,\ \ldots,A_n] \wedge t[TT_S] = t_r[TT_S] \wedge t[TT_E] =$

$t_r[TT_E] \wedge t[VT_S] = t_r[VT_S] \wedge t[VT_E] = t_r[VT_E] \vee \exists t_s \in s,\ t[A_1,\ \ldots,A_n] = t_s[A_1,\ \ldots,A_n] \wedge t[TT_S] =$

$t_s[TT_S] \wedge t[TT_E] = t_s[TT_E] \wedge t[VT_S] = t_s[VT_S] \wedge t[VT_E] = t_s[VT_E]\}$.

Temporal relational projection simply operates on the non-temporal part of the input tuples, retaining only the values of the input attributes. The temporal component of the tuples is left unchanged.

Definition 4 (Temporal projection $\pi^N_{A_i,\ \ldots,\ A_j}$) Given a bitemporal relation r in the $POINT$ or MAX approaches, defined over the schema $R^N = (A_1,\ \ldots,A_n \mid TT_S,\ TT_E,\ VT_S,\ VT_E)$, and given a subset $\{A_i,\ \ldots,A_j\}$ of the set $\{A_1,\ \ldots,A_n\}$, temporal projection $\pi^N_{A_i,\ \ldots,\ A_j}(r)$ is a bitemporal relation q defined over the schema $R'^N = (A_i,\ \ldots,A_j \mid TT_S,\ TT_E,\ VT_S,\ VT_E)$, and is defined as follows:

$\pi^N_{A_i,\ \ldots,\ A_j}(r) = \{t \mid \exists t \in r,\ t[A_i,\ \ldots,A_j] = t'[A_i,\ \ldots,A_j] \wedge$
$t[VT_S] = t'[VT_S] \wedge t[VT_E] = t'[VT_E] \wedge t[TT_S] = t'[TT_S] \wedge t[TT_E] = t'[TT_E]\}$.

The definition of selection on non-temporal attributes is trivial: only the input tuples whose non-temporal component satisfy the selection predicate ϕ are reported in output, unchanged (both in their temporal and nontemporal parts). Notice that ϕ is a predicate regarding non-temporal attributes only.

Definition 5 (Nontemporal selection σ^N_ϕ) Given a bitemporal relation r in the $POINT$ or MAX approaches, defined over the schema $R^N = (A_1,\ \ldots,A_n \mid TT_S,\ TT_E,\ VT_S,\ VT_E)$, and a predicate ϕ regarding the non-temporal attributes only, $\sigma^N_\phi(r)$ is a bitemporal relation q defined over the schema R^N, and is defined as follows:

$\sigma^N_\phi(r) = \{t \mid \exists t \in r,\ \phi(t[A_1,\ \ldots,A_n])\}$.

While the choice of admitting value-equivalent tuples makes the definition of the union, projection (and non-temporal selection) operators quite easy (and efficient, since no manipulation on the temporal components is needed), the definition of temporal difference necessarily results to be quite complex, since an unpredictable number of value-equivalent tuples may be present in the input relations. Intuitively speaking, in the temporal difference $r -^N s$, each tuple $t' \in r$ which has no value-equivalent tuple in s is simply reported unchanged in output. Otherwise, let $\{t_1, \ldots, t_k\}$ the set of all and only the tuples in s that are value-equivalent to t' (the quantifier $\exists!$ is used in the definition to denote that they are "all and only"). The time of the resulting tuple is obtained by removing from the time of t' the (union of the) times of $t_1, \ldots, t_k$. Of course, if the result of such a removal is empty, the tuple is not part of the output. Although the basic idea is simple, its realization is technically complex, since:

- as in many temporal representations (including e.g., TSQL2), in the *POINT* and *MAX* representations, times are represented through pairs of periods (start-end of the valid time, start-end of the transaction time), i.e., as rectangles in the bi-dimensional space [21], [12];
- additionally, in the *POINT* approach, point and lines in the bitemporal space are used to model now-relative data.

In the definition below, which, notably, is quite similar to the definition of difference in *BCDM*, the function $EXT^{c_{NOW}}$ (see the definition in Section 3) is used in order to generate all the bitemporal chronons corresponding to the *POINT* or *MAX* representations. The (union of the) bitemporal chronons of $t_1, \ldots, t_k$ are subtracted from the bitemporal chronons of t'. Finally, the function *cover* reconverts the resulting set of bitemporal chronons into the *POINT* or *MAX* representations. Specifically, given a set S of bitemporal chronons, $cover(S)$ provides in output a set of rectangles (i.e., (TT_S, TT_E, VT_S, VT_E) quadruples) in the *POINT* or *MAX* representations, covering all and only the chronons in S. For each one of such rectangles, a tuple value-equivalent to t' (having such a rectangle as its temporal component) is reported in output. Obviously, if the difference of bitemporal chronons is empty, *cover* is applied to the empty set, and no tuple corresponding to t' is reported.

Definition 6 (Temporal difference $-^N$) Given two bitemporal relations r and s in the *POINT* or *MAX* approaches, defined over the same schema $R^N = (A_1, \ldots, A_n \mid TT_S, TT_E, VT_S, VT_E)$, and given any value c_{NOW} for *now*, the temporal difference $r -^N s$ is a bitemporal relation q defined over the schema R^N defined as follows:
$r -^N s = \{t \mid (\exists\, t' \in r,\ t[A_1, \ldots, A_n] = t'[A_1, \ldots, A_n] \wedge t[TT_S] = t'[TT_S] \wedge t[TT_E] = t'[TT_E] \wedge t[VT_S] = t'[VT_S] \wedge t[VT_E] = t'[VT_E] \wedge \neg\ \exists\ t' \in s,\ t[A_1, \ldots, A_n] = t'[A_1, \ldots, A_n]) \vee (\exists t' \in r,\ \exists!\, t_1, \ldots, t_k \in s,\ t[A_1, \ldots, A_n] = t'[A_1, \ldots, A_n] = t_1[A_1, \ldots, A_n] = \ldots = t_k[A_1, \ldots, A_n] \wedge t[TT_S, TT_E, VT_S, VT_E] \in cover(EXT^{c_{NOW}}(t') - (EXT^{c_{NOW}}(t_1) \cup \ldots \cup EXT^{c_{NOW}}(t_k)))))\}$.

Of course, many different implementations of the $cover(S)$ function are possible, depending on the policy used in order to generate the covering rectangles.

Actually, the above definition of difference has been provided to enhance clarity, and to stress our adherence to the reference *BCDM* model. On the other hand, for the sake of computational efficiency, the translation from *POINT* or *MAX* representations to bitemporal chronons (through the $EXT^{c_{NOW}}$ function) and back (through the *cover* function), as well the application of unions and difference to bitemporal chronons, are not strictly necessary. As a matter of fact, one can more efficiently devise an algorithm that, taken in input a rectangle r (in the *POINT* or *MAX* representations) and a set of rectangles $\{r_1, \ldots, r_k\}$ (in the *POINT* or *MAX* representations), directly provides in output a set of rectangles (in the *POINT* or *MAX* representations) covering the whole difference (or the empty set if the difference is empty). In other words, one may take the above definition of difference as an abstract specification, and then implement a function to compute $cover(EXT^{c_{NOW}}(t'[TT_S, TT_E, VT_S, VT_E]) - (EXT^{c_{NOW}}(t_1[TT_S, TT_E, VT_S, VT_E]) \cup \ldots \cup EXT^{c_{NOW}}(t_k[TT_S, TT_E, VT_S, VT_E])))$ in a more efficient way. This is the strategy we adopt in this paper, proposing the function $difference$ as a sample implementation of the above computation.

4.3.1 A covering difference algorithm for the POINT and MAX approaches

The algorithm introduced in this section is an extension of the generic algorithm for computing the difference between sets of rectangles described in [7] to consider also rectangles bounded by *now*.

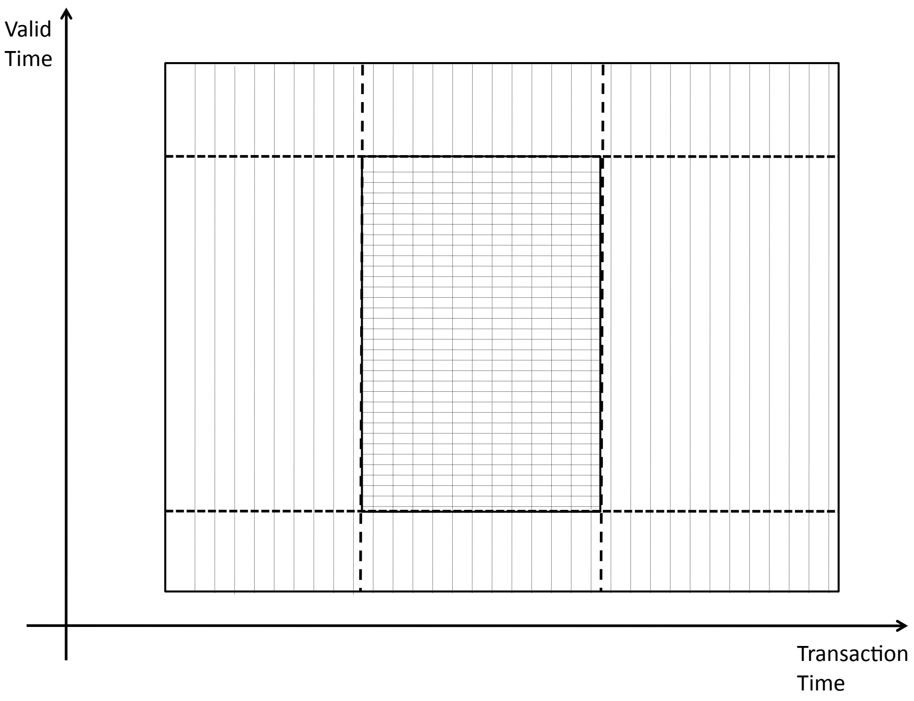

Fig. 1 Graphical representation of the difference operation between rectangles r (the biggest rectangle) and s (the smallest rectangle) with the possible four resulting rectangles (left, right, top, bottom). Some of such rectangles could be missing, depending on the relative location of r and s.

In the algorithm, the set r contains the minuends, the set s the subtrahends and the set *result* the difference. All three sets contain the timestamps of the tuples participating in the operation. In each iteration of the outer loop, a rectangle of set r is chosen. Then, for each rectangle in s that intersects with the chosen rectangle, we compute

Function 6 $difference((TT_S, TT_E, VT_S, VT_E), \{(TT^1_S, TT^1_E, VT^1_S, VT^1_E), \ldots, (TT^k_S, TT^k_E, VT^k_S, VT^k_E)\})$

```
1: $r \leftarrow \{(TT_S, TT_E, VT_S, VT_E)\}$
2: $s \leftarrow \{(TT^1_S, TT^1_E, VT^1_S, VT^1_E), \ldots, (TT^k_S, TT^k_E, VT^k_S, VT^k_E)\}$
3: $result \leftarrow \emptyset$
4: while $r \neq \emptyset$ do
5:   choose a $(TT^r_S, TT^r_E, VT^r_S, VT^r_E) \in r$
6:   $changed \leftarrow$ false
7:   for all $(TT^s_S, TT^s_E, VT^s_S, \mathrm{VT}^s_E) \in$ s do
8:     if $intersects((TT^r_S, TT^r_E, VT^r_S, VT^r_E), (TT^s_S, TT^s_E, VT^s_S, VT^s_E))$ then
9:       $r \leftarrow r - \{(TT^r_S, TT^r_E, VT^r_S, VT^r_E)\}$
10:      if $TT^r_S < TT^s_S$ then $r \leftarrow r \cup (TT^r_S, TT^s_S, VT^r_S, VT^r_E)$ {left rectangle}
11:      if $\neg$ $isNowRelative(TT^r_S, TT^r_E)$ and $\neg$ $isNowRelative(TT^s_S, TT^s_E)$ and $TT^r_E > TT^s_E$ then
             $r \leftarrow r \cup \{(TT^s_E, TT^r_E, VT^r_S, VT^r_E)\}$ {right rectangle}
12:      if $isNowRelative(TT^r_S, TT^r_E)$ and $\neg$ $isNowRelative(TT^s_S, TT^s_E)$ then $r \leftarrow$
         r $\cup$ $(TT^s_E, setNow(TT^s_E), VT^r_S, VT^r_E)$
13:      if $VT^r_S < VT^s_S$ then $r \leftarrow r \cup \{(TT^r_S, TT^r_E, VT^r_S, VT^s_S)\}$ {bottom rect-
         angle (with possible overlaps with left/right rectangles)}
14:      if $\neg$ $isNowRelative(VT^r_S, VT^r_E)$ and $\neg$ $isNowRelative(VT^s_S, VT^s_E)$ and
         $VT^r_E > VT^s_E$ then $r \leftarrow r \cup \{(TT^r_S, TT^r_E, VT^s_E, VT^r_E)\}$ {top rectangle
         (with possible overlaps with left/right rectangles)}
15:      if $isNowRelative(VT^r_S, VT^r_E)$ and $\neg$ $isNowRelative(VT^s_S, VT^s_E)$ and
         $VT^s_E \leq c_{NOW}$ then $r \leftarrow r \cup \{(TT^r_S, TT^r_E, VT^s_E, setNow(VT^s_E))\}$
16:      if $\neg$ $isNowRelative(VT^r_S, VT^r_E)$ and $isNowRelative(VT^s_S, VT^s_E)$ and
         $VT^r_E > c_{NOW} + 1$ then $r \leftarrow r \cup \{(TT^r_S, TT^r_E, c_{NOW} + 1, VT^r_E)\}$
17:      $changed \leftarrow$ true
18:    end if
19:  end for
20:  if $changed$ = false then $r \leftarrow r - (TT^r_S, TT^r_E, VT^r_S, VT^r_E)$;
21:  $result \leftarrow result \cup (TT^r_S, TT^r_E, VT^r_S, VT^r_E)$
22: end while
23: return $result$
```

Function 7 $intersects((TT^r_S, TT^r_E, VT^r_S, VT^r_E), (TT^s_S, TT^s_E, VT^s_S, VT^s_E))$

```
1: if $isNowRelative(TT^r_S, TT^r_E)$ then $tt^r_E \leftarrow c_{NOW} + 1$ else $tt^r_E \leftarrow TT^r_E$
2: if $isNowRelative(TT^s_S, TT^s_E)$ then $tt^s_E \leftarrow c_{NOW} + 1$ else $tt^s_E \leftarrow TT^s_E$
3: if $isNowRelative(VT^r_S, VT^r_E)$ then $vt^r_E \leftarrow c_{NOW} + 1$ else $vt^r_E \leftarrow VT^r_E$
4: if $isNowRelative(VT^s_S, VT^s_E)$ then $vt^s_E \leftarrow c_{NOW} + 1$ else $vt^s_E \leftarrow VT^s_E$
5: if $TT^r_S \geq tt^s_E$ or $TT^s_S \geq tt^r_E$ or $VT^r_S \geq vt^s_E$ or $VT^s_S \geq vt^r_E$ then
6:   return false
7: else
8:   return true
9: end if
```

the difference, determining the left, right, bottom and top rectangles resulting from the operation, if they exist (see Figure 1).

As for the Cartesian product, the operation is defined by cases. For instance, regarding the right rectangle, the first case (line 11) states that, if the transaction times of the element of r and of the element of s are not now-relative, the right rectangle exists if the transaction time of the minuend r ends after the transaction time of the subtrahend s (i.e., $TT_E^r > TT_E^s$). The transaction time of the right rectangle starts at TT_E^s and ends at TT_E^r and its valid time starts at VT_S^r and ends at VT_E^r.

On the other hand, if the transaction time of r is now-relative and the transaction time of s is not (line 12), because of the semantics of transaction time, the end of transaction time of r (i.e., *now*) is certainly after the end of the transaction time of s and the transaction time of the resulting rectangle ends at *now*.

The cases where r is not now-relative and s is now-relative and where both r and s are now-relative give no right rectangle because the end of the transaction time of s is certainly not before the end of the transaction time of r.

The bottom and top rectangles are computed in a similar way with a notable difference: the end of the valid time can be after *now*. This fact impacts line 16, where the valid time of r is not now-relative and the valid time of s is now-relative: differently from the analogous case in line 12, we must test whether the end of valid time of r is not after *now* (c_{NOW} is incremented because the valid time period is open to the right). Moreover, if the valid time of r is now-relative and the valid time of s is not, the top rectangle exists if the end of valid time of s is not after *now*.

It is worth noting that in the cases in lines 12 and 16 the result is also now-relative. However, neither the *MAX* approach nor the *POINT* approach admit now-relative time periods where *now* is the starting point of the period. Therefore, in the case where we would obtain *now* as the starting point (see line 16), we rather set, as starting time, the current chronon, i.e., c_{NOW}.

The resulting rectangle is then added to r in such a way that in subsequent iterations it will be checked against the other elements of s.

Please notice that, for the sake of easiness of definition, we have chosen to give as a result possibly overlapping rectangles, but it is possible to define the $difference$ function in such a way to give as a result non-overlapping rectangles.

4.3.2 Temporal selection operators

Until now, we have taken into account extensions of Codd's basic operators. However, Codd's operators were not originally intended to operate on time, so that no specific operator was provided in order to query the temporal component of temporal tuples.

Many of such operators have been provided in the literature (consider, e.g., the TSQL2 "consensus" approach [21]). For the sake of brevity, in this paper we just focus on *temporal selection*.

Definition 7 (Temporal selection $\sigma^N_{\phi_t}$) Given a bitemporal relation r in the *POINT* or *MAX* approaches, defined over the schema $R^N = (A_1, \ldots, A_n \mid TT_S, TT_E, VT_S, VT_E)$ and a temporal predicate ϕ_t regarding the temporal attributes only, $\sigma^N_{\phi_t}(r)$ is a bitemporal relation q defined over the schema R^N, and is defined as follows:

$$\sigma^N_{\phi_t}(r) \ = \ \{t \mid \exists\, t \ \in \ r, \ \ \phi_t \ (t[TT_S, \ TT_E, \ VT_S, \ VT_E])\}$$

In other words, all and only the tuples in r^N whose temporal component satisfy the selection predicate ϕ_t are reported in output. Such a general definition can be further specified, in order to identify specific temporal selection predicates. In particular, *range queries* have been proven to play a major role within TDBs [14]. Range queries have been already defined for the *POINT* approach (the interested reader is referred to [23]).

4.3.3 Properties of the POINT and MAX algebrae

In this subsection, we explore some of the main properties of the temporal relational algebra for the *POINT* and *MAX* approaches.

By definition, the input relations of the *POINT* (and *MAX*) algebraic operators are relations in the *POINT* (and *MAX*) approach. The outputs are *POINT* (and *MAX*) relations by construction. This can be easily proved by cases. For instance, the output of temporal Cartesian product is by definition a relation whose temporal component consists of the TT_S, TT_E, VT_S, VT_E attributes (as requested by the *POINT* –and *MAX*– models). In turn, the temporal component of each output tuple has, by construction, values that conform to the *POINT* (and *MAX*) models. Therefore the following property holds:

Property 1 (Closure) The temporal relational algebra is closed, in the sense that the algebraic operators, taking in input relations in the *POINT* (or *MAX*) model, give in output relations in the *POINT* (or *MAX*) model.

A fundamental property is the correctness of the *POINT* and *MAX* algebrae with respect to the *BCDM* approach. This means that, although the *POINT* and *MAX* approaches adopt an efficient *representation* to cope with now-relative data, the *BCDM* underlying semantics is still preserved, in that the output of the *POINT* and *MAX* operations is exactly a *representation* (in the *POINT* and *MAX* approaches) of the output of the corresponding operations in the *BCDM* semantic approach.

Property 2 (Correctness of the POINT and MAX algebrae) The *POINT* and *MAX* algebrae are correct, since they provide (in the *POINT* and *MAX* representations) all and only the results that would be obtained by the underlying *BCDM* semantic approach.

Proof The correctness of the *POINT* and *MAX* algebrae is proved by showing that, given any relations r^N and s^N and any binary operator Op^N in the *POINT* or *MAX* approaches (the proof is analogous when considering unary operators), the application of the operator in *POINT* or *MAX* algebrae gives a result equivalent to the application of the corresponding operator in the *BCDM* algebra (indicated by Op^B):

$to\_BCDM^{cNOW}(r^N \ Op^N \ s^N) \ = \ to\_BCDM^{cNOW}(r^N) \ Op^B \ to\_BCDM^{cNOW}(s^N)$.

We prove the result for the operation of Cartesian product by proving the two inclusions, that is we prove that:

$to\_BCDM^{cNOW}(r^N \ \times^N \ s^N) \ \subseteq \ to\_BCDM^{cNOW}(r^N) \ \times^B \ to\_BCDM^{cNOW}(s^N)$

and
$to\_BCDM^{c_{NOW}}(r^N \times^N s^N) \supseteq to\_BCDM^{c_{NOW}}(r^N) \times^B to\_BCDM^{c_{NOW}}(s^N)$.

Let x be a *BCDM* tuple in $to\_BCDM^{c_{NOW}}(r^N \times^N s^N)$ and let $R = (A_1, \dots, A_a, B_1, \dots, B_b \mid T)$ be the schema of $to\_BCDM^{c_{NOW}}(r^N \times^N s^N)$. We denote as A the attributes $A_1, \dots, A_a$ and as B the attributes $B_1, \dots, B_b$. Then, by the definition of the $to\_BCDM^{c_{NOW}}$ function, there exists a maximal set of (possibly now-relative) tuples $\{y_0, y_1, \dots, y_k\}$ in $r^N \times^N s^N$ $(k \geq 0)$ such that $x[AB] = y_0[AB] = \dots = y_k[AB]$ and that, considering the rectangles resulting from the extension of their timestamps (with $now = c_{NOW}$), their union corresponds to $x[T]$, i.e., $x[T] = EXT^{c_{NOW}}(y_0[T]) \cup \dots \cup EXT^{c_{NOW}}(y_k[T])$. If such tuples $\{y_0, y_1, \dots, y_k\}$ belongs to $r^N \times s^N$, by the definition of $\times^N$, for each $y_i \in \{y_0, y_1, \dots, y_k\}$ there must be a tuple $w'_{i'} \in r^N$ and a tuple $w''_{i''} \in s^N$ such that $x[A] = y_i[A] = w'_{i'}[A]$, $x[B] = y_i[B] = w''_{i''}[B]$ and the bitemporal chronons of y_i correspond to the intersection of the bitemporal chronons of $w'_{i'}$ and $w''_{i''}$, i.e., $EXT^{c_{NOW}}(y_i[T]) = EXT^{c_{NOW}}(w'_{i'}[T]) \cap EXT^{c_{NOW}}(w''_{i''}[T])$. In the following, let $\{w'_1, \dots, w'_l\}$ and $\{w''_1, \dots, w''_m\}$ the sets of all and only the tuples in r^N and in s^N respectively that generate the above tuples $\{y_0, y_1, \dots, y_k\}$.

Let us consider the tuples $w'_1, \dots, w'_l$ in r^N. Applying the *to_BCDM* function, since these tuples have the same values for the explicit attributes, by definition of *to_BCDM*, *to_BCDM* returns one *BCDM* tuple $y \in to\_BCDM(r^N)$ such that $y[A] = w'_1[A] = \dots = w'_l[A]$ and $y[T] = EXT(w'_1[T]) \cup \dots \cup EXT(w'_l[T])$. If we consider the tuples $w''_1, \dots, w''_m$ in s^N, the function *to_BCDM* returns one *BCDM* tuple $z \in to\_BCDM(s^N)$ such that $z[A] = w''_1[A] = \dots = w''_m[A]$ and $z[T] = EXT(w''_1[T]) \cup \dots \cup EXT(w''_m[T])$.

Applying the $\times^B$ operator to y and z, by definition of the operator, we obtain a relation on schema $R(AB \mid T)$ containing a tuple $x' \in to\_BCDM^{c_{NOW}}(r^N) \times^B to\_BCDM^{c_{NOW}}(s^N)$ such that $x'[A] = y[A] = x[A]$ and $x'[B] = z[B] = x[B]$ and $x'[T] = y[T] \cap z[T] = (EXT^{c_{NOW}}(w'_1[T]) \cup \dots \cup EXT^{c_{NOW}}(w'_l[T])) \cap (EXT^{c_{NOW}}(w''_1[T]) \cup \dots \cup EXT^{c_{NOW}}(w''_m[T]))$. For the distributive property of intersection over union, $x'[T] = (EXT^{c_{NOW}}(w'_1[T]) \cap EXT^{c_{NOW}}(w''_1[T])) \cup \dots \cup (EXT^{c_{NOW}}(w'_1[T]) \cap EXT^{c_{NOW}}(w''_m[T])) \cup \dots \cup (EXT^{c_{NOW}}(w'_l[T]) \cap EXT^{c_{NOW}}(w''_1[T])) \cup \dots \cup (EXT^{c_{NOW}}(w'_l[T]) \cap EXT^{c_{NOW}}(w''_m[T]))$. Since by construction $\{w'_1, \dots, w'_l\}$ and $\{w''_1, \dots, w''_m\}$ are all and only the tuples that generates $\{y_0, y_1, \dots, y_k\}$, i.e., such that for each i, $0 \leq i \leq k$, $x[A] = y_i[A] = w'_{i'}[A]$, $x[B] = y_i[B] = w''_{i''}[B]$ and $EXT^{c_{NOW}}(w'_{i'}) \cap EXT^{c_{NOW}}(w''_{i''}) = EXT^{c_{NOW}}(y_i[T])$, we have that $x'[T] = EXT^{c_{NOW}}(y_1[T]) \cup \dots \cup EXT^{c_{NOW}}(y_k[T]) = x[T]$. Therefore, $x = x'$.

Now we prove the other direction of the inclusion.

Let x' be a *BCDM* tuple in $to\_BCDM^{c_{NOW}}(r^N) \times^B to\_BCDM^{c_{NOW}}(s^N)$. Then, by definition of the $\times^B$ operator, there exist a tuple $y \in to\_BCDM^{c_{NOW}}(r^N)$ and a tuple $z \in to\_BCDM^{c_{NOW}}(s^N)$ such that $x'[A] = y[A]$, $x'[B] = z[B]$ and $x'[T] = y[T] \cap z[T] \neq \emptyset$.

By definition of *to_BCDM* function, if $y \in to\_BCDM^{c_{NOW}}(r^N)$ then there exists a (maximal) set of (possibly now-relative) tuples $\{w'_0, \dots, w'_l\}$ that are in r^N such that $y[A] = w'_0[A] = \dots = w'_l[A]$ and $y[T] = EXT^{c_{NOW}}(w'_0[T]) \cup \dots \cup EXT^{c_{NOW}}(w'_l[T])$. The same holds for z: there exists a (maximal) set of (possibly now-relative) tuples $\{w''_0, \ldots, w''_m\}$ that are in s^N such that $z[A] = w''_0[A] = \ldots = w''_m[A]$ and $z[T] = EXT^{cNOW}(w''_0[T]) \cup \ldots \cup EXT^{cNOW}(w''_m[T])$.

If we apply the $\times^N$ operator to r^N and s^N, by definition of the operator, because $\{w'_0, \ldots, w'_l\} \subseteq r^N$ and $\{w''_0, \ldots, w''_m\} \subseteq s^N$, we obtain, among the other tuples, a set of (possibly now-relative) tuples $\{y_1, \ldots, y_k\}$ such that $y_i[A] = w'_{i'}[A]$, $y_i[B] = w''_{i''}[B]$ and $EXT^{cNOW}(y_i[T]) = EXT^{cNOW}(w'_{i'}[T]) \cap EXT^{cNOW}(w''_{i''}[T])$. Because $w'_{i'}[A] = y[A]$ and $w''_{i''}[B] = z[B]$, the *to_BCDM* function coalesces these tuples in one *BCDM* tuple x such that $x[A] = y_0[A] = \ldots = y_k[A] = y[A], x[B] = y_0[B] = \ldots = y_k[B] = z[B]$ and $x[T] = EXT^{cNOW}(y_0[T]) \cup \ldots \cup EXT^{cNOW}(y_k[T])$. Since for each i, $0 \leq i \leq k$, $EXT^{cNOW}(y_i[T]) = EXT^{cNOW}(w'_{i'}[T]) \cap EXT^{cNOW}(w''_{i''}[T])$, $x[T] = EXT^{cNOW}(y_0[T]) \cup \ldots \cup EXT^{cNOW}(y_k[T]) = (EXT^{cNOW}(w'_{i'_0}[T]) \cap EXT^{cNOW}(w''_{i''_0}[T])) \cup \ldots \cup (EXT^{cNOW}(w'_{i'_k}[T]) \cap EXT^{cNOW}(w''_{i''_k}[T]))$.

For the distributive property of union of sets over intersection of sets, $x[T] = (EXT^{cNOW}(w'_{i'_0}[T]) \cap EXT^{cNOW}(w''_{i''_0}[T])) \cup \ldots \cup (EXT^{cNOW}(w'_{i'_k}[T]) \cap EXT^{cNOW}(w''_{i''_k}[T])) = (EXT^{cNOW}(w'_1[T]) \cup \ldots \cup EXT^{cNOW}(w'_l[T])) \cap (EXT^{cNOW}(w''_1[T]) \cup \ldots \cup EXT^{cNOW}(w''_m[T]))$. Since $y[T] = EXT^{cNOW}(w'_0[T]) \cup \ldots \cup EXT^{cNOW}(w'_l[T])$ and $z[T] = EXT^{cNOW}(w''_0[T]) \cup \ldots \cup EXT^{cNOW}(w''_m[T])$, we have that $x[T] = y[T] \cap z[T] = x'[T]$. Therefore, $x = x'$.

The proof for the other operators is similar.

4.3.4 Reducibility to the snapshot algebra

An important issue concerning a temporal algebra lies in its compatibility and reducibility to the standard (non-temporal) relational model, in order to grant that, if time is disregarded, the extended temporal model behaves like the standard model. These properties are important for temporal approaches, since they grant interoperability with pre-existent non-temporal approaches (and, as a matter of facts, such properties have been proved for many temporal approaches, such as TSQL2).

Therefore, we prove the reducibility and consistent-extension properties for our approach. For defining these properties, we first have to introduce transaction- and valid timeslice operators, which take in input a bitemporal relation and a time value and give as output, respectively, a valid time and a transaction-time relation, by "slicing" a temporal dimension at the given time value. For instance, the transaction-timeslice operator takes in input a bitemporal relation r and a time t and gives as output a valid-time relation r', containing all and only the tuples in r whose transaction time contains t. Only the valid time of such tuples is retained in r'. The valid timeslice operator is analogous: if any tuple of the bitemporal relation has a valid time which includes the time value given in input, the relation given in output contains the explicit values of the tuple and its transaction time.

The transaction-timeslice operator for transaction-time relations and valid timeslice operator for valid time relations are straightforward special cases.

Definition 8 (Transaction timeslice) Let r a bitemporal relation in the *POINT* or *MAX* approaches, defined over the schema $R^N = (A_1, \ldots, A_n \mid TT_S, TT_E, VT_S, VT_E)$

and c_T an arbitrary time value not exceeding current time:
$\rho^N_{c_T}(r) \ = \ \{t \mid \exists\, t' \in r,\, t[A] = t'[A] \wedge t[VT_S] \ = \ t'[VT_S] \wedge t[VT_E] \ = \ t'[VT_E] \wedge c_t \in EXT^{c_{NOW}}(t'[TT_S],\, t'[TT_E])\}$.

The result of transaction-timeslice operator is a valid time relation, defined over the schema $R^N_V \ = \ (A_1,\ \ldots,\ A_n \mid VT_S,\ VT_E)$.

Definition 9 (Valid timeslice) Let r a bitemporal relation in the *POINT* or *MAX* approaches, defined over the schema $R^N \ = \ (A_1,\ \ldots,\ A_n \mid TT_S,\ TT_E,\ VT_S,\ VT_E)$ and c_V an arbitrary time value:
$\tau^N_{c_V}(r) \ = \ \{t \mid \exists\, t' \in r,\, t[A] \ = \ t'[A] \wedge t[TT_S] \ = \ t'[TT_S] \wedge t[TT_E] \ = \ t'[TT_E] \wedge c_v \in EXT^{c_{NOW}}(t'[VT_S],\, t'[VT_E])\}$.

The result of valid timeslice operator is a transaction-time relation, defined over the schema $R^N_T \ = \ (A_1,\ \ldots,\ A_n \mid TT_S,\ TT_E)$.

Notice that the combined application to a bitemporal relation r of a transaction and a valid timeslice operators at times t_{TT} and t_{VT} respectively provides as output a standard (non-temporal) relation r', consisting of (the non-temporal part of) all and only those tuples in r which hold at the bi-temporal chronon $(t_{TT},\ t_{VT})$. Informally, r' is the standard relation capturing the single *snapshot* $(t_{TT},\ t_{VT})$ of the bitemporal relation r.

Property 3 (Reducibility of POINT and MAX algebrae to the standard (non-temporal) algebra) The *MAX* and *POINT* algebrae reduce to the standard algebra, i.e., the non-temporal relation obtained by applying extended temporal operators to temporal relations and then taking a snapshot is equivalent to the standard (non-temporal) relation obtained by first taking a snapshot of the temporal relations and then applying a standard (non-temporal) operator.

More formally, let r^N and s^N be bitemporal relations defined over a schema $R^N \ = \ (A_1,\ \ldots,\ A_n \mid TT_S,\ TT_E, VT_S,\ VT_E)$, c_T an arbitrary time value not exceeding current time and c_V an arbitrary time value,
$\rho_{c_T}(\tau_{c_V}(r^N\ Op^N\ s^N)) \ = \ \rho_{c_T}(\tau_{c_V}(r^N))\ Op^S\ \rho_{c_T}(\tau_{c_V}(s^N))$,
where Op^N is a temporal operator and Op^S is a standard (non-temporal) operator. The statement is analogous for unary operators.

Proof We prove the property for the operator of Cartesian product. The proofs for the other operators are analogous.

The equivalence is proved by proving the two inclusions.

First we prove that $\rho_{c_T}(\tau_{c_V}(r^N \times^N s^N)) \subseteq \rho_{c_T}(\tau_{c_V}(r^N)) \times^S \rho_{c_T}(\tau_{c_V}(s^N))$.

Let $R \ = \ (A_1,\ \ldots,\ A_a,\ B_1,\ \ldots,\ B_b \mid TT_S,\ TT_E,\ VT_S,\ VT_E)$ be the schema of $r^N \times^N s^N$. We denote as A the attributes $A_1,\ \ldots,\ A_a$ and as B the attributes $B_1,\ \ldots,\ B_b$. Let $x \in \rho_{c_T}(\tau_{c_V}(r^N \times^N s^N))$. Then, by definition of slice operators, there is a tuple $x' \in r^N \times s^N$ such that $x'[AB] = x[AB]$ and $(c_T,\ c_V) \in EXT^{c_{NOW}}((r^N \times^N s^N)[T])$. Therefore, by definition of the $\times^N$, there exist a tuple $y \in r^N$ and a tuple $z \in s^N$ such that $y[A] \ = \ x'[A], z[B] \ = \ x'[B],\ (c_T,\ c_V) \in EXT^{c_{NOW}}(y[T])$ and $(c_T,\ c_V) \in EXT^{c_{NOW}}(z[T])$. Therefore, by definition of slice operators, there exist a tuple $w' \in \rho_{c_T}(\tau_{c_V}(r^N))$ and a tuple $w'' \in \rho_{c_T}(\tau_{c_V}(s^N))$ such that $w' \ = \ y[A] \ = \ x'[A]$ and $w'' \ = \ z[B] \ = \ x'[B]$.

By definition of standard Cartesian product, there exists a tuple $x'' \in \rho_{c_T}(\tau_{c_V}(r^N)) \times^S \rho_{c_T}(\tau_{c_V}(s^N))$ such that $x''[A] = w' = y[A] = x'[A]$ and $x''[B] = w'' = z[B] = x'[B]$. Therefore, $x = x''$.

Now we prove that $\rho_{c_T}(\tau_{c_V}(r^N \times^N s^N)) \supseteq \rho_{c_T}(\tau_{c_V}(r^N)) \times^S \rho_{c_T}(\tau_{c_V}(s^N))$.

Let us assume that $x'' \in \rho_{c_T}(\tau_{c_V}(r^N)) \times^S \rho_{c_T}(\tau_{c_V}(s^N))$. By definition of $\times^S$ operator, there exist a tuple $w' \in \rho_{c_T}(\tau_{c_V}(r^N))$ and a tuple $w'' \in \rho_{c_T}(\tau_{c_V}(s^N))$ such that $x''[A] = w'$ and $x''[B] = w''$. By definition of the slice operators, there exist a tuple $y \in r^N$ and a tuple $z \in s^N$ such that $y[A] = w'$, $z[B] = w''$, $(c_V, c_T) \in EXT^{c_{NOW}}(y[T])$ and $(c_V, c_T) \in EXT^{c_{NOW}}(z[T])$.

By definition of $\times^N$ operator, there exists a tuple $x' \in r^N \times^N s^N$ such that $x'[A] = y[A]$, $x'[B] = z[B]$ and $(c_V, c_T) \in EXT^{c_{NOW}}(x'[T])$. By definition of slice operators, there exists a tuple $x \in \rho_{c_T}(\tau_{c_V}(r^N\ Op^N\ s^N))$ such that $x = x'[AB]$; therefore, since $x[A] = x'[A] = y[A] = w' = x''[A]$ and $x[B] = x'[B] = z = w'' = x''[B]$, we have that $x = x''$.

Definition 10 (Temporal transform) Let r be a standard (non-temporal) relation, defined over the schema $R_1 = (A_1, \ldots, A_n)$ and tt_start, tt_end, vt_start, vt_end timestamps,
$transform(r, tt\_start, tt\_end, vt\_start, vt\_end) = \{t \mid t[A] \in r \wedge t[T] = (tt\_start, tt\_end, vt\_start, vt\_end)\}$.

The result of temporal-transform operator is a bitemporal relation, defined over the schema $R_2 = (A_1, \ldots, A_n \mid TT_S, TT_E, VT_S, VT_E)$.

The following property grants that queries that are possible when time is not represented are possible when time is added.

Property 4 (Consistent extension) The temporal algebrae are consistent extensions of the standard (non-temporal) algebra, i.e., the standard relational operators have a counterpart in the temporal relational operators.

Proof We have to prove that, let r^N and s^N be standard (non-temporal) relations defined over a schema $R = (A_1, \ldots, A_n)$ and tt_start, tt_end, vt_start, vt_end timestamps, $transform(r\ Op\ s, tt\_start, tt\_end, vt\_start, vt\_end) = transform(r, tt\_start, tt\_end, vt\_start, vt\_end)\ Op^N\ transform(s, tt\_start, tt\_end, vt\_start, vt\_end)$, where Op is a standard (non-temporal) operator and Op^N a temporal operator. The statement is analogous for unary operators.

We prove the two inclusions separately for the Cartesian product. The proofs for the other operators are analogous.

Let $x \in transform(r \times s, tt\_start, tt\_end, vt\_start, vt\_end)$. Then, by definition of $transform$ function, $x[AB] \in r \times s$ and $x[T] = (tt\_start, tt\_end, vt\_start, vt\_end)$. By definition of $\times$, there exist a tuple $y \in r$ and a tuple $z \in s$ such that $x[A] = y$ and $x[B] = z$. Applying the $transform$ function to y, we obtain a bitemporal tuple y' such that $y'[A] = y$ and $y'[T] = (tt\_start, tt\_end, vt\_start, vt\_end)$. The same holds for z: applying the $transform$ function to z, we obtain a bitemporal tuple z' such that $z'[B] = z$ and $z'[T] = (tt\_start, tt\_end, vt\_start, vt\_end)$. By definition of the $\times^N$ operator, there exists a bitemporal tuple x' such that $x'[A] = y'[A]$, $x'[B] = z'[B]$ and $x'[T] = (tt\_start, tt\_end, vt\_start, vt\_end)$. Then, $x = x'$.

Now we assume that $x' \in transform(r, tt\_start, tt\_end, vt\_start, vt\_end) \times^N transform(s, tt\_start, tt\_end, vt\_start, vt\_end)$. Then, by definition of $\times^N$, there exist a tuple $y' \in transform(r, tt\_start, tt\_end, vt\_start, vt\_end)$ and a tuple $z' \in transform(s, tt\_start, tt\_end, vt\_start, vt\_end)$ such that $x'[A] = y'[A]$ and $x'[B] = z'[B]$, and $z'[T] \cap y'[T] \neq \emptyset$. By definition of the *transform* function, there exist a tuple $y \in r$ and a tuple $z \in s$ such that $y = y'[A]$ and $z = z'[A]$, and $y[T] = (tt\_start, tt\_end, vt\_start, vt\_end)$ and $z[T] = (tt\_start, tt\_end, vt\_start, vt\_end)$. By definition of the $\times$ operator, there exists a tuple $x'' \in r \times s$ such that $x''[A] = y, x''[B] = z$. By definition of the *transform* function, there exists a tuple $x \in transform(r \times s, tt\_start, tt\_end, vt\_start, vt\_end)$ such that $x[A] = x''[A] = y'[A] = y = x'[A]$, $x[B] = x''[B] = z'[B] = z = x[B]$ and $x[T] = z'[T] = y'[T] = x[T] = (tt\_start, tt\_end, vt\_start, vt\_end)$.

4.4 $NOT-NOW$ algebra

In the next section, we show some experimental evaluations for the temporal relational algebrae we have defined. To the best of our knowledge, there are no temporal relational algebrae in the literature supporting the treatment of now-relative data without resorting to Variable databases. Therefore, in order to provide a "reference" comparison term for our MAX and $POINT$ algebrae, in this section we provide a third approach, that we term "$NOT-NOW$". The $NOT-NOW$ approach is the simplest approach to cope with now-relative data. It assumes that the future is known, so that the ending times (of both valid and transaction times) of all tuples are known timestamps. Of course, in such a database there is no way to state, e.g., Example 1. One has to state, e.g., that the transaction time of the tuple starts at 11 and ends at a specific date. $NOT-NOW$ is an "ideal" approach and it is not feasible in practice because we cannot know the future. However, such a less-expressive approach is useful for the sake of our experiments: by comparing the MAX and $POINT$ approaches to the $NOT-NOW$ one, we can show what is the additional cost of coping with now-relative data in such approaches. Of course, comparisons will be drawn by considering corresponding datasets in the three different approaches, in the sense that now-relative data are represented as discussed in Section 2.1 as regards the MAX and $POINT$ approaches, and the corresponding data in the $NOT-NOW$ approach use specific timestamps to cope with the end of now-relative transaction and/or valid time.

As an example, we show the MAX, $POINT$ and $NOT-NOW$ representation of a relation $EMPLOYEE$ containing the following information (and assuming the value 25 for *now*):

- John worked in the toy department from 10 to 20 (inserted at $TT = 11$, deleted at $TT = 15$)
- John worked in the travel department from 10 to 20 (inserted at $TT = 16$, still present in the DB)
- John worked in the toy department from 21 to *now* (inserted at $TT = 21$, still present in the DB)

ID	*Dept*	TT_S	TT_E	VT_S	VT_E
John	*toy*	11	16	10	21
John	*travel*	16	$max-value$	10	21
John	*toy*	21	$max-value$	21	$max-value$

Table 1 *EMPLOYEE* relation in the *MAX* approach.

ID	*Dept*	TT_S	TT_E	VT_S	VT_E
John	*toy*	11	16	10	21
John	*travel*	16	16	10	21
John	*toy*	21	21	21	21

Table 2 *EMPLOYEE* relation in the *POINT* approach.

ID	*Dept*	TT_S	TT_E	VT_S	VT_E
John	*toy*	11	16	10	21
John	*travel*	16	41	10	21
John	*toy*	21	1001	21	31

Table 3 *EMPLOYEE* relation in the $NOT-NOW$ approach.

Considering the $NOT-NOW$ approach, we also assume to know that the second tuple will be deleted at time 40 in the future, that John will stop working in the toy department at time 30, and that such a tuple will be deleted at $TT = 1000$.

These information are represented in Tables 1, 2 and 3. Notice that, for the sake of homogeneity, also in the $NOT-NOW$ approach we have followed the convention that periods are closed to the left and open to the right.

Interestingly, given the generality of the family of algebrae we have defined in Section 4.3, also the $NOT-NOW$ algebra can be obtained as a specific instance of the family (as it is the case for the *MAX* and for the *POINT* algebrae). In particular, the $NOT-NOW$ algebra is a trivial simplification of the above.

For the sake of readability, we report here the Cartesian product operator for the $NOT-NOW$ approach:

Definition 11 ($NOT-NOW$ Temporal Cartesian product $\times^{NN}$) Given two bitemporal relations r and s in the $NOT-NOW$ approach, defined over the schemas $R_1 = (A_1,\ldots,A_n \mid TT_S,TT_E,VT_S,VT_E)$ and $R_2 = (B_1,\ldots,B_k \mid TT_S,TT_E,VT_S,VT_E)$ respectively, the $NOT-NOW$ temporal Cartesian product $r \times^{NN} s$ is a temporal relation q defined over the schema $R_3 = (A_1,\ldots,A_n,B_1,\ldots,B_k \mid TT_S,TT_E,VT_S,VT_E)$ and is defined as follows:
$r \times^{NN} s = \{t \mid \exists\, t_r \in r, \exists\, t_s \in s, t[A_1, \ldots, A_n] = t_r[A_1, \ldots, A_n] \wedge$
$t[B_1, \ldots, B_k] = t_s[B_1, \ldots, B_k] \wedge t[TT_S] = max(t_r[TT_S], t_s[TT_S]) \wedge$
$t[TT_E] = min(t_r[TT_E], t_s[TT_E]) \wedge t[TT_S] < t[TT_E] \wedge$
$t[VT_S] = max(t_r[VT_S], t_s[VT_S]) \wedge t[VT_E] = min(t_r[VT_E], t_s[VT_E]) \wedge t[VT_S] < t[VT_E]\}$

5 Experimental evaluation

In this section, we empirically compare the performances of the *MAX*, the *POINT* and the *NOT-NOW* approaches. We have not considered the approaches modeling *now* using variables [3], since we consider approaches which do not require any modification of the kernel so that, e.g., off-the-shelf indexing techniques can be used. We generated the same data set for each approach and then on each relation we performed Temporal Cartesian Product, Temporal Difference, and Temporal Selection (e.g., range queries).

5.1 Data sets

In absence of real data we have randomly generated data sets with different data distributions to simulate a real-world scenario. For each approach we generated three tables differing in the percentage of now-relative data. As suggested in the literature [28], [25] we considered 10%, 20% and 40% of now-relative data. The structure of all tables are identical to the sample data shown in Tables 1, 2, and 3, where, besides the *ID* and *Dept* attributes, there are four temporal attributes VT_s, VT_e, TT_s, and TT_e representing the start and end of the valid and transaction time. Each table contains one million tuples. However, for Temporal Cartesian Product and Temporal Difference, due to the answer size, we considered only one thousand rows in each table. For the *NOT-NOW* approach we have replaced now-relative valid time and/or transaction time ends with randomly generated timestamps in the future.

The starting time of the periods were always uniformly distributed on the time domain, while the duration and percentage of now-relative data was varied. While we have chosen a uniform distribution of period starts, we adopted an exponential distribution of the durations because it reflects most real-world applications where short periods are more likely to occur than long periods [6].

5.2 Environment

Our implementation has been carried out on a 8 x UltraSparc III @ 900Hz with 8GB of RAM memory, running Oracle 11 RDBMS, with a database block size of 8K and size of SGA of 1000MB. The SGA was locked into memory to ensure that paging does not affect results. To ensure that the logical read of data already in SGA does not influence the results we flushed the the database buffer cache in SGA before every particular test. At the time of testing the database server did not have any other significant load. We used Oracle built-in methods for statistics collection, analytic SQL functions and the PL/SQL procedural runtime environment. For range queries we utilised the TD-tree indexing method [24], as it has been shown that the TD-tree has the best performance, considering the physical disk I/O and the query response time and at the same time can be employed within the commercial RDBMS [26].

Approach	Now-relative data ratio	Disk Accesses	CPU usage (10s of milliseconds)	Response time (min:sec)	Logic.Reads	Answer size (number of tuples)
MAX	10%	16	10,521	1:46.325	1,132,103	104,501
POINT	10%	16	10,338	1:44.393	1,128,668	104,501
NOT − NOW	10%	16	9,113	1:31.672	1,121,456	101,968
MAX	20%	16	12,679	2:13.714	1,240,591	228,466
POINT	20%	16	12,377	2:10.687	1,236,454	228,466
NOT − NOW	20%	16	11,352	2:01.404	1,228,571	226,481
MAX	40%	16	15,990	2:55.695	1,439,416	408,485
POINT	40%	16	15,547	2:51.133	1,429,377	408,485
NOT − NOW	20%	16	14,801	2:37.906	1,418,882	400,766

Table 4 Temporal Cartesian Product

5.3 Results and analysis

In the following, we first present the results concerning the three different types of queries separately. In all cases, we consider a database containing different ratios of now-relative data. This is important in order to show to what extent our results depend on the presence of such data in the database. Both physical disk I/Os and CPU usage are taken into account in our analysis. It worth stressing that, according to the analysis in [9], physical disk I/Os is the most important parameter. As a matter of fact, physical disk accesses are considered to be the bottleneck because CPU time might be reduced through the introduction of more powerful CPUs, and/or with an extension of the number of CPUs. However, considering the relatively small number of records in tables used for evaluations of Temporal Cartesian Product and Temporal Difference and, since in such cases a full table scan needs to be performed, CPU usage and response time are useful indicators for such operations.

5.3.1 Temporal Cartesian Product

As can be seen in Table 4, since Cartesian Product requires a full table scan, disk accesses are the same in all approaches. Also as regards CPU usage and response time, the *POINT*, *MAX* and *NOT-NOW* approaches provide similar results. Actually, the *NOT-NOW* approach, which is the optimal one (since it indeed does not require any extension to cope with *now*) but is not practically feasible (since it assumes to know the future), requires slightly less CPU usage for any percentage of now-relative data and therefore has the fastest response time. However, it is also important to notice that this is also due to the fact that, in the *NOT-NOW* approach, the answer size is slightly smaller than in the other approaches. In fact, in the *NOT-NOW* approach, we have replaced the now-relative data with the random future timestamps, and therefore less tuples satisfy the query criteria (i.e., intersection of bitemporal times). Also, the *POINT* approach is slightly better than the *MAX* approach with regard to CPU usage and has a faster response time (due to the less expensive algorithm of temporal Cartesian product). This difference is increasing as the percentage of now-relative data increases, as a result of the increase of the answer size.

Approach	Now-relative data ratio	Disk Accesses	CPU usage (10s of milliseconds)	Response time (min:sec)	Logic.Reads	Answer size (number of tuples)
MAX	10%	16	88	0:01.591	4,744	1,007
POINT	10%	16	87	0:01.523	4,742	1,007
$NOT-NOW$	10%	16	84	0:01.482	4,740	1,007
MAX	20%	16	87	0:01.972	4,736	1,007
POINT	20%	16	86	0:01.875	4,734	1,007
$NOT-NOW$	20%	16	85	0:01.775	4,730	1,007
MAX	40%	16	89	0:01.749	4,734	1,007
POINT	40%	16	87	0:01.716	4,734	1,007
$NOT-NOW$	20%	16	84	0:01.664	4,732	1,007

Table 5 Temporal Difference Product

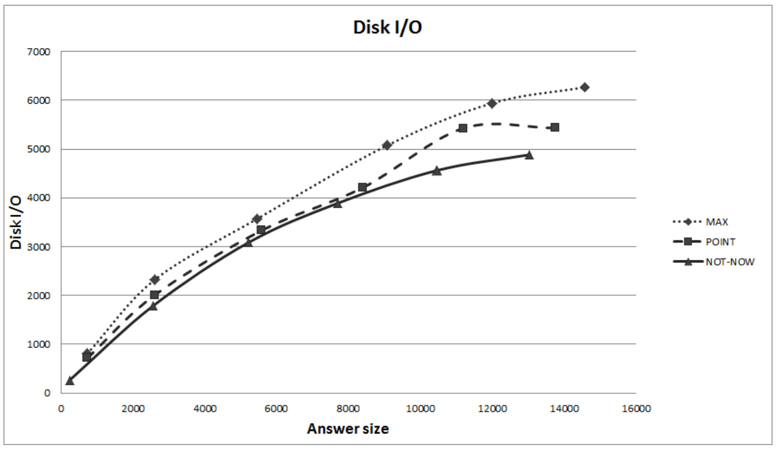

Fig. 2 Range Query - Physical Disk I/O

5.3.2 Temporal Difference

Table 5 shows the results of the *POINT*, the *MAX*, and *NOT-NOW* approaches considering the physical disk I/Os, CPU usage, and response time for temporal difference. Notably, there are only slight differences in CPU usage and response time between the evaluated approaches. Due to the nature of temporal difference (and the possible dimension of the answer size), we had to consider temporal difference between tables with only one thousand rows. Since difference requires a full table scan to consider all the records, physical disk I/O is constant. As it can be seen, both *MAX* and *POINT* approaches are only slightly worse with regard to the CPU usage and response time than the optimal *NOT-NOW* approach.

5.3.3 Range Queries

In Figures 2, 3, and 4 we show the results for physical disk I/O, CPU usage, and query response time for range queries as a factor of the answer size. In these experi-

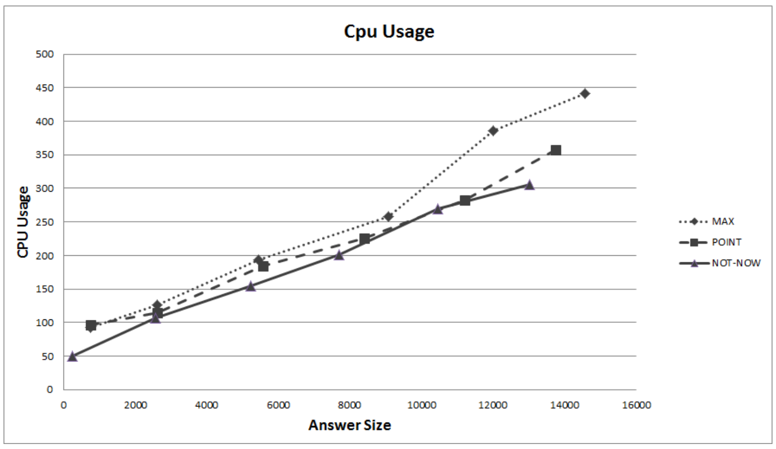

Fig. 3 Range Query - CPU Usage

ments, we have considered 20% of now-relative data. However, the results obtained with 10% and 40% now-relative data are similar. Once again, although the "ideal" *NOT-NOW* approach performs slightly better that the *POINT* and *MAX* approaches, differences are minimal.

Thus, experiments clearly show that the performances of the *POINT* and *MAX* algebrae are very close to that of the *NOT-NOW* one, which is indeed the optimal one (but which is not practically feasible), since, by assuming that the future end-times of now-relative data are known, does not actually need to deal with *now* at all.

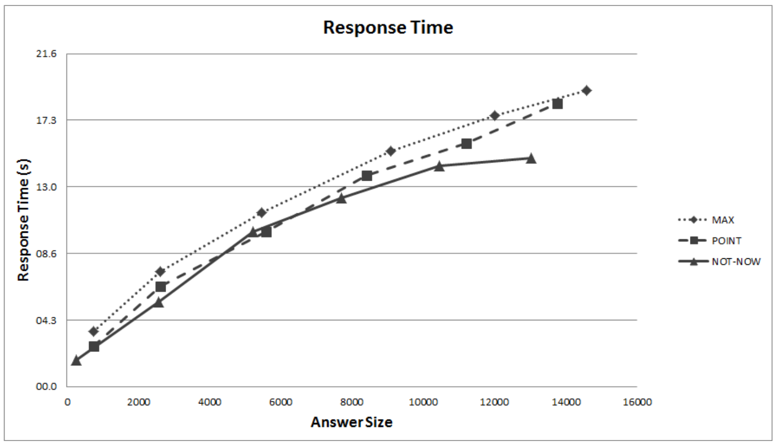

Fig. 4 Range Query - Response Time

6 Comparisons and conclusions

Now-relative temporal data play an important role in most temporal applications. As a consequence, the treatment of such data has attracted a significant amount of attention in the (temporal) database literature. In the recent Database Encyclopedia by Springer, Dyreson, Jensen and Snodgrass [16] have pointed out that there are three different uses of *now* in databases. The first use of *now* is as a function within queries, views, assertions. The second use is as a database variable used as a special timestamp value associated with tuples or attribute values in TDB instances. The third use of *now* is as a database variable with a specified offset. In this paper, we focus on the second use, i.e., *now* used as a "special" timestamp in the valid and/or transaction time of temporal data. Early approaches to now-relative data have been mostly based on the use of variables (called "variable databases" [16]). The semantics of *now* variables have been widely studied [16]. For instance, in [3], the semantics of *now* variables has been formalized through the introduction of *extensionalization* functions, that map from a database containing variables into an extensional database level which is fully ground and constitutes its sematics, at a given *reference time*. A significant amount of work has also been devoted to the problem of querying (see, e.g., [3]) and updating (consider, e.g., [31]) *variable databases*. Considering queries, which is the main focus of our approach, most researchers have tried to simplify the treatment of now-relative data by simply augmenting a "standard" temporal query language with the addition of a *binding* operator. This operator is used when user-level queries are mapped to the internal representation. It operates on tuples with variables, and substitues variables with a ground value (the *reference time* in [3]), thus setting the perspective of the user raising the query. Existing query languages usually assume that, for the sake of simplicity, the reference time is the time when the user starts to raise the query. More recently, as already discussed in the introduction, a different mainstream of approaches have tried to deal with *now* in relational databases without resorting to the use of variables, with the advantage of adhering to the standard relational model. These approaches include the *MIN*, *MAX*, *NULL* [29] and *POINT* [23] approaches. In such approaches, *now* is not represented by a variable, but by a "special" ground value (e.g., the maximum possible time in the *MAX* approach), or by using degenerate representations of intervals (in the *POINT* approach). The idea is that the specific value is used to represent *now* is not an "actual" value, but just a "marker" for a variable. Thus, while the representation is still a conventional one (in which no variable is used), the semantics is similar to the one proposed for "variable databases". However, such an intended semantics must be supported also by query languages. Indeed, to the best of our knowledge, current approaches to *now* not using variables have only focused on the treatment of a restricted type of queries: *range* (or *slice*) queries (i.e., queries asking for all data holding at a given time interval or time point). Indeed, besides supporting such queries, it is important to provide a full query language, so that also "standard" queries can be asked on data in the *MIN*, *MAX*, *NULL*, and/or *POINT* approaches.

In this paper we overcome such a limitation of the current literature. We focus on the purely relational approaches (thus neglecting variable databases), and provide a general temporal relational algebra which can be polymorphically adapted to act as the query language not only for the *MAX* and *POINT* approaches (as shown in this paper), but also to the other "non-variables" approaches in the literature (i.e., the *NULL* and the *MIN* ones). Besides generality, our algebra meets also several other theoretical and practical desiderata: *closure* with respect to representation languages, *correctness* with respect to the "consensus" *BCDM* semantics, *reducibility* to the standard non-temporal algebra (which involves interoperability with non-temporal relational databases), *implementability* and *efficiency*. Indeed, the experimental evaluation we have drawn on our implementation has shown that only a slight overhead is added by our treatment of now-relative data (with respect to an "ideal" but unfeasible approach in which such data are not present since future is known).

References

1. Ben-Zvi. "the time relational model.". *PhD. Dissertation. Computer Science Department, UCLA*, 1982.
2. M. H. Bohlen, R. T. Snodgrass, and M. D. Soo. Coalescing in Temporal Databases. *Proc. of the 22nd VLDB Conf*, pages 180–190, 1996.
3. J. Clifford, C. Dyreson, T. Isakowitz, C. S. Jensen, and R. T. Snodgrass. On the semantics of "Now" in databases. *ACM Transactions on Database Systems (TODS)*, 22(2):171–214, 1997.
4. E. F. Codd. Relational completeness of data base sublanguages. *In: R. Rustin (ed.): Database Systems: 65-98, Prentice Hall and IBM Research Report RJ 987, San Jose, California*, 1972.
5. C. E. Dyreson, C. S. Jensen, and R. T. Snodgrass. Now in temporal databases. In L. LIU and M. ZSU, editors, *Encyclopedia of Database Systems*, pages 1920–1924. Springer US, 2009.
6. R. Fenk, V. Markl, and R. Bayer. Interval Processing with the UB-Tree. In *Proceedings of the 2002 International Symposium on Database Engineering and Applications*, pages 12–22, 2002.
7. D. S. Franzblau and G. Xenakis. An algorithm for the difference between set covers. *Discrete Appl. Math.*, 156(10):1623–1632, May 2008.
8. S. K. Gadia. A seamless generic extension of sql for querying temporal data. *Technical Report TR-92-02. Computer Science Department, Iowa State University. May*, 1992.
9. J. Hellerstein, E. Koutsupias, and C. Papadimitriou. On the Analysis of Indexing Schemes. *16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 1997.
10. C. S. Jensen. *Temporal Database Management*. PhD thesis, Department of Computer Science, Aalborg University, 2000.
11. C. S. Jensen and R. Snodgrass. Temporal Data Management. *IEEE Transactions on Knowledge and Data Engineering*, 11(1):36–44, 1999.
12. C. S. Jensen and R. T. Snodgrass. Semantics of Time-Varying Information. *Information Systems*, 21(4):311–352, 1996.

13. L. M. Jensen, C. S. and N. Roussopoulos. ”incremental implementation model for relational databases with transaction time.” ieee transactions. *IEEE Transactions on Knowledge and Data Engineering*, 3(4):461–473, 1991.
14. H. Kriegel, M. Ptke, and T. Seidl. Managing intervals efficiently in object-relational databases. *Proceedings of the 26th International Conference on Very Large Databases*, pages 407–418, 2000.
15. J. L. Edwin McKenzie and R. T. Snodgrass. Evaluation of Relational Algebras Incorporating the Time Dimension in Databases. *ACM Computing Surveys (CSUR)*, 23(4):501–543, 1991.
16. L. Liu and M. T. Özsu, editors. *Encyclopedia of Database Systems*. Springer US, 2009.
17. L. E. Mckenzie, Jr. *An algebraic language for query and update of temporal databases*. PhD thesis, The University of North Carolina at Chapel Hill, 1988. AAI8914449.
18. J. Melton and A. R. Simon. *SQL:1999 - Understanding Relational Language Components*. Morgan Kaufman, 2002.
19. G. Ozsoyoglu and R. Snodgrass. Temporal and real-time databases: A survey. *IEEE Trans. On Knowledge and Data Engineering*, 7(4):513–532, 1995.
20. R. Snodgrass. The temporal query language tquel. *ACM Trans. Database Syst.*, 12(2):247–298, June 1987.
21. R. T. Snodgrass. *The TSQL2 Temporal Query Language*. Kluwer Academic, 1995.
22. R. T. Snodgrass. *Developing Time-Oriented Database Applications in SQL*. Morgan Kaufmann, 2000.
23. B. Stantic, A. Sattar, and P. Terenziani. The point approach to represent it now in bitemporal databases. *J. Intell. Inf. Syst.*, 32(3):297–323, 2009.
24. B. Stantic, J. Terry, R. W. Topor, and A. Sattar. Indexing Temporal Data with Virtual Structure. In *Advances in Databases and Information Systems - ADBIS*, pages 591–594, 2010.
25. B. Stantic, J. Thornton, and A. Sattar. A Novel Approach to Model NOW in Temporal Databases. *In Proceeding of the 10th International Symposium on Temporal Representation and Reasoning (TIME-ICTL 2003), Cairns*, pages 174–181, 2003.
26. B. Stantic, R. W. Topor, J. Terry, and A. Sattar. Advanced indexing technique for temporal data. *Comput. Sci. Inf. Syst.*, 7(4):679–703, 2010.
27. A. U. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. Snodgrass, editors. *Temporal databases: theory, design, and implementation*. Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA, 1993.
28. K. Torp, C. S. Jensen, and M. Bohlen. Layered implementation of temporal DBMS concepts and techniques. *A TimeCenter Technical Report TR-2*, 1999.
29. K. Torp, C. S. Jensen, and M. H. Böhlen. Layered temporal dbms: Concepts and techniques. In *Proceedings of the Fifth International Conference on Database Systems for Advanced Applications (DASFAA)*, pages 371–380. World Scientific Press, 1997.
30. K. Torp, C. S. Jensen, and R. T. Snodgrass. Effective Timestamping in Databases. *VLDB Journal: Very Large Data Bases*, 8(3–4):267–288, 2000.

31. K. Torp, C. S. Jensen, and R. T. Snodgrass. Modification semantics in now-relative databases. *Inf. Syst.*, 29(8):653–683, Dec. 2004.
32. V. Tsotras and A. Kumar. Temporal database bibliography update. *ACM Sigmod Record*, 25(1):41–51, 1996.