Spatio-Temporal Evolution: Querying Patterns of Change in Databases

Nassima Djafri, Alvaro A.A. Fernandes, Norman W. Paton, Tony Griffiths
Department of Computer Science, University of Manchester, Manchester M13 9PL, UK
{ndjafri|alvaro|norm|griffitt}@cs.man.ac.uk

ABSTRACT

This paper contributes a general approach to characterizing patterns of change in a spatio-temporal database. While there is a particular interest in modelling and querying how spatio-temporal entities evolve, the approach contributed by the paper is distinctive in being applicable without modification to aspatial entities as well. The paper uses the Tripod spatio-temporal model to describe and instantiate in detail the contributed approach. After briefly describing a typical application and providing basic knowledge about Tripod, the paper characterizes and classifies evolution queries and describes in detail how they are evaluated.

Categories and Subject Descriptors

H.2.3 [**Information Systems**]: Database Management—*Query Languages*

General Terms

Algorithms

Keywords

Spatio-Temporal Databases, Spatio-Temporal Query

1. INTRODUCTION

Spatio-temporal databases have been an active area of research since, at least, the early 1990s. This surge in interest has resulted in such recent advances as the modelling of moving objects (e.g., [11]), the development of constraint-based formalisms (e.g., [10]) and of spatio-temporal data models (e.g., [8, 15]). These advances suggest that database technologies could come to play as central a role in the development and deployment of spatio-temporal applications as they have done, in the aspatial, non-temporal case for at least thirty years. However, this is unlikely to come about without advanced query capabilities. In particular, spatio-temporal databases will need to support querying of change patterns, i.e., to the evolution of the entities modelled.

While the wealth of proposals in areas such as access methods and indexing strategies as well as (at the other end of the abstraction spectrum) in user-level query syntax is, of course, to be welcomed, more work is needed on how to model advanced spatio-temporal queries at intermediate levels of abstraction. This paper aims to make one such contribution. It describes a general approach to characterizing *evolution queries*, i.e., queries that identify patterns of change in the histories of entities, spatial or not. The approach is distinctive in being applicable not only to spatial but, without modification, aspatial entities as well.

The remainder of the paper is structured as follows. Section 2 provides motivation through a representative application and some examples of the classes of queries that form the focus of the paper's contributions. Section 3 provides a very brief introduction to the Tripod system [7, 8, 9] as a backdrop for the technical development. Section 4 classifies evolution queries and describes in detail a general approach to evaluating them both in the spatial and in the aspatial case. Section 5 discusses related work. Finally, some conclusions are drawn in Section 6.

2. MOTIVATION

The Longitudinal Study (LS) by the UK Office for National Statistics [13] links census and vital event data (e.g., births, deaths, cancer registrations) for a one percent sample of the population of England and Wales (about 500,000 individuals) entered in the 1971, 1981 and 1991 censuses, plus information on other individuals living in the same household at each census. A range of prospective analyses can be carried out with the LS data. Analyses can be census-based (e.g., changes in housing tenure, car ownership, migration behaviour, or occupational mobility between 1971, 1981 and 1991), event-based (e.g., fertility patterns), or both (e.g., how socio-demographic differences in mortality, or fertility, have evolved since 1971). A schema (from [8]), in Tripod's object definition language (ODL), showing a subset of the classes of interest in the LS is presented in Figure 1.

The LS is of particular interest in the context of this paper because the patterns of change that spatial and aspatial entities undergo are a fundamental aspect of what the study was designed to support. The contributions of this paper can, therefore, be seen as instrumental in allowing the data captured in the LS to elicit more information more effectively and efficiently than would otherwise be possible. The following are illustrative examples of queries over change patterns that are suggested by the data in the LS. Note that in order to make concrete examples more informative, this paper assumes the existence of data for more points in time than [13] actually contains.

- **Q1:** In which years, between 1989 and 2002, did the salary of LS member P grow?

GIS'2002, November 8–9, 2002, McLean, Virginia, USA.

```
historical(TimeIntervals, DAY) class LS_member
( extent LS_members )
{       attribute       TimeInstant             date_of_birth;
        attribute       TimeIntervals           periods_of_work;
        historical(TimeIntervals, DAY)
        attribute       enum {S,M,D,W}          marital_status;
        historical(TimeIntervals, YEAR)
        attribute       int                     salary;
        historical(Instants, YEAR)
        attribute       Points                  location;
        historical(TimeIntervals, DAY)
        relationship    Enumeration_district    resides_in
        inverse         Enumeration_district    ::  has_LS_members;
};
historical(TimeIntervals, DAY) class Enumeration_district
( extent Enumeration_districts )
{       attribute       string                  name;
        historical(TimeIntervals, YEAR)
        attribute       Regions                 boundary;
        historical(TimeIntervals, DAY)
        relationship    set<LS_member>          has_LS_members
        inverse         LS_member               ::  resides_in;
};
```

Figure 1: A Tripod Schema for the LS

- **Q2:** In which years, between 1989 and 1991, did the location of LS member P move closer to the boundary of enumeration district E?

- **Q3:** When did the location of LS member P enter enumeration district E?

Q1 requires the query processor to scrutinize consecutive snapshots in a single history. In this paper, this kind of query is referred to as *intra-history cross-timestamp* (IHC). Given two consecutive snapshots v and v' in the history H of the `salary` of LS member P, the goal in evaluating **Q1** is to ascertain that $v < v'$ and to record the outcome for every interval of interest. Note, from **Q1**, that, in this paper, change patterns are not necessarily spatially referenced.

Q2 requires the query processor to scrutinize consecutive snapshots in a pair of (aligned) histories. In this paper, this kind of query is referred to as *cross-history cross-timestamp* (CHC). Given the history H of the `location` in space of P and the history H' of the `boundary` of enumeration district E, the goal in evaluating **Q2** is to record when the distance between the location of P and the boundary of E in one snapshot was greater than in the consecutive one.

Finally, **Q3** requires the query processor to scrutinize each snapshot in a pair of (aligned) histories to ascertain whether, at each timestamp, a particular relationship holds or not. In this paper, this kind of query is referred to as *cross-history intra-timestamp* (CHI). Given the history H of the location of P and the history H' of the boundary of E, the goal in evaluating **Q3** is to record when the location of P was outside the boundary of E in one snapshot and then inside it, in the consecutive snapshot.

Note that IHI (i.e., intra-history intra-timestamp) queries are not evolution queries: they do not compare states of affairs, neither across histories nor across snapshots.

3. BACKGROUND: THE TRIPOD SYSTEM

The Tripod project [7, 8, 9] is developing a complete spatio-historical database system by extending the ODMG database standard for object databases [2] with two new kinds of primitive type, viz., spatial types and timestamp types and a new collection type, called *history*. The spatial types are based on the ROSE (RObust Spatial Extensions) algebraic types described in [12], viz., `Points`, `Lines` and `Regions`. The timestamp types, viz., `Instants` and `TimeIntervals`, are one-dimensional versions of the ROSE-algebraic types `Points` and `Lines` respectively. For each spatial and timestamp type, there exist operations, referred to as `assemble` and a `disassemble` that map from a set of singular values (e.g., of type `Point`) to the corresponding collection value (i.e, `Points`, in the example)

A *history* is a quadruple $H = \langle V, \theta^*, \gamma, \Sigma \rangle$, where V denotes the domain of values whose changes H records, $\theta^* \in \Theta^*$ and Θ^* is the set of all Tripod timestamp types, γ is the granularity of θ^*, and Σ is a set of pairs, called *states*, of the form $\langle \tau, \sigma \rangle$, where τ is a timestamp and σ is a snapshot. Σ is referred to as the *state set* of H.

Notational conventions regarding histories are as follows. Let $\mathbb{T}$ denote the set of all timestamps; $\mathbb{V}$, the set of all snapshots; $\mathbb{S}$, the set of all states; and $\mathbb{H}$, the set of all histories. A `TimeIntervals` value is written as $[t_1^s : t_1^e, \ldots, t_n^s : t_n^e]$, where each element is a *half-open (at the end)* interval. An element in a state set is written as exemplified by $\langle [01/01/1983 : 03/05/1987, 12/04/1992 : 19/12/2000], v \rangle$, where v denotes, e.g., a snapshot value from the domain of `boundary`. Dot notation is used to denote the individual elements of a particular state. For example, the timestamp of a particular state s is denoted by s.timestamp, and the corresponding snapshot by s.snapshot. The state set of a history H is denoted by H.state. Each Tripod type that extends the ODMG standard has a rich set of operations (see [7, 9]).

Tripod extends the monoid comprehension approach [6] to provide a semantics for Tripod queries. A *monoid* is a triple $(T, \oplus, Z_\oplus)$ consisting of a set T together with a binary associative operation $\oplus : T \times T \rightarrow T$, called the *merge* function for the monoid, and an identity element $Z_\oplus$, called the *zero element* of the monoid. In database contexts, T is taken to be a database type. Examples of monoids are $(\texttt{int}, +, 0)$ and $(\texttt{Bool}, \wedge, \texttt{true})$. A *comprehension over the monoid* $\oplus$ is an expression of the form $\oplus\{e|\overrightarrow{r}\}$, where the expression e is called the *head* of the comprehension and $\overrightarrow{r} = r_1, \ldots, r_n$, $n \geq 0$, is a sequence of qualifiers each one of which is either a *generator* of the form $v \leftarrow e'$ or a *filter* in the form of a predicate p over the terms in the comprehension. Collection monoids can be defined over $\mathbb{H}$. For example, one where the zero element is the empty history, denoted by $\{\langle\ \rangle\}$, the unit function is $\lambda x.\{\langle x \rangle\}$ for any $x \in \mathbb{S}$, and the merge function is $\Cup$. This history monoid allows monoid comprehensions to be defined over $\mathbb{H}$ in which the lexicon for filters stems from the set of Boolean-valued operations over histories, and the set of domain generators is extended by history-denoting expressions (including monoid comprehensions other than $(\Cup, \{\langle\ \rangle\})$).

For reasons of space, some definitions and examples are omitted here. Readers are referred to [7, 8, 9] for full details.

4. EVOLUTION QUERIES

In this paper, a *change pattern* is a chronologically ordered sequence of observations describing how, within a history, the relationship of one state to its predecessor has evolved, or how a relationship between two entities has evolved across their joint (i.e., aligned) histories.

An example of the first kind might be **Q1** in Section 2, i.e., an IHC (for intra-history cross-timestamp) query on the evolution of a person's salary to establish that it has never reduced. Another example might be to query whether

or when the boundary of a district has grown (or shrunk). An example of the second kind might be **Q2** in Section 2, i.e., a CHC (for cross-history cross-timestamp) query as to when a person's location moved closer to the boundary of a district. Another example might be **Q3** in Section 2, i.e., a CHI (for cross-history intra-timestamp) query as to when a point entered a region.

Such queries on change patterns are referred to in this paper as *evolution queries*. In the aspatial case, they are related to *sequence queries* [17]. In the spatial case, they are related to the notion of *developments* [3]. Evolution queries generalize both of these proposals in the sense that they cover both aspatial and spatial entities uniformly.

In the approach contributed by this paper, given a history (or a pair thereof), the kind of evolution that has taken place is characterized by firstly comparing successive snapshots (or by testing whether some predicate holds between two values at the same point in time). This comparison (e.g., for one snapshot being smaller than its successor) establishes what kind of change (if any) has taken place (e.g., that the value was growing). Note that, in this paper, given two timestamps t_1 and t_2, "between t_1 and t_2" is taken to mean "at every granule contained in the (half-open) interval denoted by $[t_1 : t_2]$".

In some cases, before the comparison is carried out, a mapping is applied to each snapshot value. For example, to consecutive snapshots of type `Lines` one might apply the mapping `length` : `Lines` $\rightarrow$ `float` and compare the outcomes with a view to concluding whether or when that length was growing. In this paper, a mapping is always a Tripod primitive operation (or the identity mapping), and a comparator is always a Tripod predicate (e.g., $>$: `float` $\times$ `float` $\rightarrow$ `Bool`, or `on_border_of` : `Points` $\times$ `Regions` $\rightarrow$ `Bool`) [7, 9].

There is an initial stage in which an intermediate representation is built that plays the role of interpretation structure for a query. Individual elements in this structure map one-to-one onto the instants contained in the history under analysis. Each element either denotes whether change has taken place, or that this fact cannot be determined. The interpretation structure takes the form of a string over the alphabet $\{\mathtt{t}, \mathtt{f}, \mathtt{u}\}$, where, as expected, `t` denotes truth (e.g., yes, the corresponding, possibly mapped, value is greater than its predecessor), `f` denotes falsity (e.g., no, it is not greater) and `u` denotes the impossibility to assign a truth value (e.g., there is a gap where a predecessor state would be expected, making it impossible to conclude that the snapshot is greater or not). To query for a change pattern, one then specifies this pattern as a regular expression over the interpretation structure. In a second and final stage, the answer to the query is computed. For that purpose, regular expression pattern matching is used over the string. Given what each character denotes and the bijective property with respect to the history being queried, the result can be used to construct the answer, as described in detail in Sections 4.1 to 4.6. In this way one can express queries regarding continuity, alternation, etc. Likewise, one can express queries regarding evolution over intervals whose length is not known.

To conclude this overview, a taxonomy of evolution queries in Tripod is partially depicted in Figure 2. Since, in Tripod, every construct that can be assigned a value can be construed as historical, and since evolution queries can be posed over any history, it is clear that Figure 2 merely exemplifies, rather than exhaustively covers, the kinds of evolution queries in Tripod. The purpose of Figure 2 is to underpin the detailed description of how evolution queries can be posed and answered. The question as to what other kinds of evolution queries there exist that cannot be depicted in a drawing such as Figure 2 is addressed formally at the end of this section.

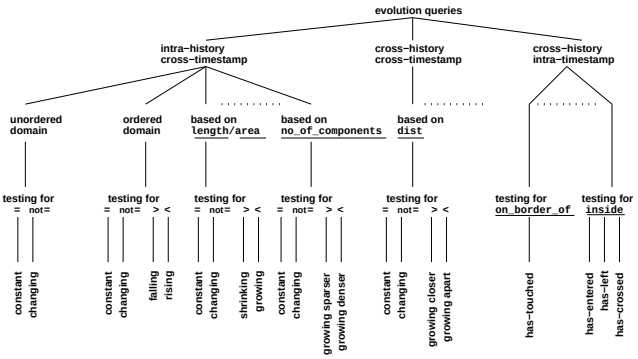

Figure 2: Kinds of Evolution Queries in Tripod

The left child of the root comprises IHC queries. The leftmost path in the IHC sub-tree captures the simplest case, viz., if the history is of a nominal attribute (e.g., an `enum` attribute such as `marital_status`), one can only query based on equality (e.g., if one were to use equality as a comparator, then a regular expression such as $[\mathtt{t}]*$ could be used to identify the periods in which the marital status remained constant, and possibly equal to some specific value, e.g., `W`, for widowed). For an ordered attribute, using comparators such as $>$ and $<$ opens the way to characterizing periods in the history in which the snapshot values are falling and rising, respectively. The remaining branches of the IHC subtree contain examples of how a mapping can be used to characterize certain phenomena. For example, if one takes the history of a `Lines` value, say, of (the geometrical representation of) a road and applies the mapping `length`, one can characterize that the road is, e.g., growing, by using the comparator $<$. Likewise, one could query whether the number of components in, say, a `Regions` value is growing. In all the examples above, consecutive snapshots (hence the cross-timestamp qualifier) from the same history (hence the intra-history qualifier) are being compared.

The middle child of the root comprises CHC queries. For example, the histories of a `Points` and a `Regions` value can be first aligned (hence the cross-history qualifier) so as to compute the `dist` between the paired snapshots and then identify whether it is, e.g., growing. The comparison is of distances in consecutive snapshots (hence the cross-timestamp qualifier) in the aligned history.

Finally, the right child of the root comprises CHI queries. It illustrates how predicates other than the classical ones can be used as comparators. Using a predicate such as `on_border_of` allows one to characterize change patterns relating to whether two spatial entities came into contact (e.g., if on the predecessor state it was false that one was `on_border_of` the other, and, now, on the current state, it is true). Similarly, patterns based on the predicate `inside` give rise to a characterization as to whether an object entered another (e.g., a person moved to an enumeration district, assuming the former to be spatially referenced). Because two histories are aligned, these are cross-history queries, but because the comparator is applied at each snapshot (rather than consecutive ones), they qualify as intra-timestamp.

Note that evolution queries are, in their most general formulation, type-independent, i.e., one can query how values evolve irrespective of their types. In particular, because Tripod models spatial values as primitive values, it is largely immaterial whether the change pertains to a spatial attribute or an aspatial one (unlike, e.g., [5], where the focus is specifically on changes with reference to space, or [17], where changes in space are not captured at all).

4.1 Interpretation Structure for IHC Queries

The algorithm in Figure 3 constructs the interpretation structure corresponding to an IHC query, i.e., an evolution query over consecutive snapshots within a single (non-empty) history. The signature of the BUILD-IHC-STRUCTURE function defined in Figure 3 is History $< \tau, T_1 > \times (T_1 \rightarrow T_2) \times ((T_2 \times T_2) \rightarrow \texttt{Bool}) \rightarrow (\texttt{Instant} \times \texttt{String}^{\{\texttt{t,f,u}\}})$ where History $< \tau, T_1 >$ ranges over history values with timestamp type τ and snapshot type T_1; T_2 is a Tripod type (possibly the same as T_1); $(T_1 \rightarrow T_2)$ is a unary mapping from T_1 to T_2 (possibly the identity); $((T_2 \times T_2) \rightarrow \texttt{Bool})$ is a binary predicate on T_2; and $(\texttt{Instant} \times \texttt{String}^{\{\texttt{t,f,u}\}})$ is a pair, with the left element an instant value (denoting the earliest granule in the period covered by the argument history), and the right element a string over the alphabet $\{\texttt{t}, \texttt{f}, \texttt{u}\}$.

```
BUILD-IHC-STRUCTURE(H, μ, φ)
 1  H' ← ⊎{(t, μ(v)) | (t, v) ← H}
 2  ES ← ϵ
 3  if H' has timestamp type Instants
 4     then e ← first(EarliestState(H'). timestamp )
 5     else e ← start(first(EarliestState(H'). timestamp ))
 6  it ← new INSTANT-STATE-ITERATOR(H')
 7  s ← it. get_element()
 8  while ¬(it. at_end() )
 9  do
10     it. next()
11     s' ← it. get_element()
12     d ← dist(s. timestamp , s'. timestamp )
13     if d ≠ 1
14        then ES ← ES^(d * u)
15        else if φ(s. snapshot , s'. snapshot )
16                then ES ← ES^t
17                else ES ← ES^f
18     s ← s'
19  return ⟨e, ES⟩
```

Figure 3: Interpretation Structure for IHC Queries

In Figure 3, INSTANT-STATE-ITERATOR is one of the iterators defined in the Tripod language bindings. It traverses the state set of a history in ascending chronological order, ensuring that if the timestamp value is not of the type `Instant`, then it emits, in order, as many states (with `Instant` timestamp type) as there are instants in the original timestamp by replicating the original snapshot value. For example, on a history $H = \{\langle [t_1 : t_3], 30\rangle, \langle [t_4 : t_5, t_6 : t_8], 12\rangle, \langle [t_9 : t_{10}], 14\rangle\}$ with timestamp type `TimeIntervals`, an INSTANT-STATE-ITERATOR would return the following states, in the stated order, ($\langle t_1, 30\rangle$, $\langle t_2, 30\rangle$, $\langle t_4, 12\rangle$, $\langle t_6, 12\rangle$, $\langle t_7, 12\rangle$, $\langle t_9, 14\rangle$). An INSTANT-STATE-ITERATOR responds to an invocation of *at_end()* with `true` if it has emitted the last state. It responds to *get_element()* by emitting the state corresponding to its current index. It responds to *next()* by pointing to the state that lies next (in ascending chronological order) from its current index.

The first step in the algorithm in Figure 3 assigns to H' the output of a monoid comprehension in which the input mapping μ (possibly the identity) is applied to each snapshot value in the (non-empty) input history H. Then the *evolution string* ES is initialized to the empty string ϵ before iteration starts. The earliest instant in H' is recorded as e, taking into account the two possible timestamp types when invoking the Tripod operations to obtain that instant. Next, H' is traversed instant by instant in ascending chronological order. At each pass, the comparator ϕ is applied to the snapshots of two consecutive states s and s'. String concatenation (denoted by '^') is used to record the outcome in the evolution string ES. True is recorded as `t` and false as `f`. If between the timestamps of s and s' there is a gap (measurable by the Tripod operation `dist`) of d granules, then the character `u` (for undefined) is concatenated d times to ES. This guarantees that, for each instant contained in the timespan of H, there is a corresponding character in ES. When the traversal ends, the algorithm pairs e and ES as the output interpretation structure I for an IHC query.

Values for the unary mapping μ (other than the identity function) include those in Figure 2, i.e., `area`, `length` and `no_of_components`. Values for the binary comparator ϕ include those in Figure 2, i.e., $=, \neq, >, <$.

For example, consider again query **Q1** in Section 2. To see how the Tripod query processor constructs the interpretation structure needed to answer **Q1**, assume H_S to denote the history of the salary of LS member P, then the call BUILD-IHC-STRUCTURE($H_S, \iota, <$) builds the required structure. This call returns a pair where the left element is that `Instant` value corresponding to the earliest granule in the history of the salary of P and the right element is a string, referred to as an `evolution string`, where a `t` denotes that, at that granule, the salary of P was smaller than in the succeeding granule, an `f` denotes that it was not, and an `u` denotes that the succeeding granule is not one granule away (i.e., there is a gap of size, at least, one).

4.2 Identifying Patterns in IHC Queries

In this paper, an evolution query specifies the changes of interest as a pattern. The pattern is matched against the evolution string in the output pair computed by the algorithm in Figure 3. The earliest instant (i.e., the other element in the output pair) is then used to compute the intervals in which the desired changes were observed (if any).

The algorithm in Figure 4 constructs a `TimeIntervals` value that represents those intervals in the history being queried in which the change pattern occurs. The signature of the BUILD-IHC-INTERVALS function defined in Figure 4 is $\text{RegExp}^{\{\texttt{t,f,u}\}} \times (\texttt{Instant} \times \texttt{String}^{\{\texttt{t,f,u}\}}) \rightarrow \texttt{TimeIntervals}$

```
BUILD-IHC-INTERVALS(r, ⟨e, ES⟩)
1  TS ← {}
2  M ← MATCH-REGULAR-EXPRESSION(r, ES)
3  for (i, m) ∈ M
4  do TS ← TS ∪ {MakeInterval(e + i, e + i + m)}
5  return assemble(TS)
```

Figure 4: Change Patterns in IHC Queries

In Figure 4, MATCH-REGULAR-EXPRESSION is a string matching algorithm that takes a regular expression r and an interpretation structure $I = \langle e, ES\rangle$ returned by BUILD-IHC-STRUCTURE in Figure 3 and returns the index i and the length m of each match of r in ES. Two Tripod operations are used in Figure 4, viz., `MakeInterval`, which constructs a `TimeInterval` value from two `Instant` values,

and assemble which constructs a TimeIntervals value out of a set of TimeInterval values, with coalescing taking place by default, if necessary.

The algorithm in Figure 4 begins by setting up an accumulator for the return set TS of TimeInterval values. It then obtains the set M of all matches of the regular expression r in the evaluation string ES. Then, for each such match $(i, m) \in M$, it adds to TS a TimeInterval defined as follows. The start of the interval is determined by taking the index i of the match as an offset on the earliest instant e returned by BUILD-IHC-STRUCTURE in Figure 3. The end of the interval is determined by adding to the start of the interval the length m of the match.

For example, assume a history H of an attribute salary of type int, with TimeIntervals timestamps and granularity YEAR. Assume the state set of H to be $\{\langle[1989 : 1994, 1997 : 1999], 20\rangle, \langle[1999 : 2000], 22\rangle, \langle[2000 : 2002], 25\rangle\}$. Recall that an interval is interpreted as half-open (at the end). Then, BUILD-IHC-STRUCTURE$(H, \iota, <)$ (with ι denoting the identity mapping) returns the pair $\langle 1989, \texttt{ffffuuuufttf}\rangle$, i.e, taking the intervals to be open at the end, the two points in which it can be said that the salary grew was between 1998 and 1999, and then between 1999 and 2000. A call to BUILD-IHC-INTERVALS([t]+, BUILD-IHC-STRUCTURE$(H, \iota, <)$) identifies (non-empty) periods of indefinite length when the salary rose. In the above example, the single match is denoted by $(9, 2)$. Given that the earliest instant is $e = 1989$, the interval set returned is $\{[(1989 + 9) : (1989 + 9 + 2)]\} = \{[1998 : 2000]\}$. This is the answer to query **Q1** if the history of the salary of LS member P is H given above.

4.3 Interpretation Structure for CHC Queries

Interpretation structures for CHC queries, i.e., evolution queries over consecutive snapshots in a paired (non-empty) history are built by a variant of the algorithm in Figure 3. In this case (and that of CHI queries discussed later), the focus is on the evolving relationship between two objects, rather than on the evolution of a property of an object.

In order to determine the evolution of a relationship between two historical objects it is necessary to compare their snapshot values for all timestamp values they have in common in their respective histories. To address this requirement, the two histories need to be *aligned*, a notion that closely resembles the process called *splitting* in [1]. In Tripod, the binary Align operation on two histories H_1 and H_2 returns a new history H_3 such that, for each pair of timestamps t_1 and t_2 (from H_1 and H_2, respectively) that have instants in common, it generates one state in H_3 having as timestamp the intersection of t_1 and t_2 and as snapshot a pair (v_1, v_2) where v_1 and v_2 are the snapshots associated with t_1 and t_2, respectively. More formally, Align can be defined using a monoid comprehension as follows:

$$\begin{aligned}\texttt{Align}(H_1, H_2) = \ & \Cup\{(\underline{\texttt{intersection}}(t_1, t_2), (v_1, v_2)) \mid \\ & (t_1, v_1) \leftarrow H_1, \\ & (t_2, v_2) \leftarrow H_2 \\ & \underline{\texttt{common\_instants}}\_\exists(t_1, t_2)\}\end{aligned}$$

Once the two histories are aligned, a binary mapping into a single value (e.g., one that computes the distance between two spatial objects), and a binary comparator (e.g., $<$) are used to build the interpretation structure. The signature of the BUILD-CHC-STRUCTURE function is History $< \tau, T_1 >$ $\times$ History $< \tau, T_2 > \times ((T_1 \times T_2) \rightarrow T_3) \times ((T_3 \times T_3) \rightarrow$ Bool$) \rightarrow ($Instant $\times$ String$^{\{\texttt{t,f,u}\}})$ where History $< \tau, T_1 >$ and History $< \tau, T_2 >$ range over history values with timestamp type τ and snapshot types T_1 and T_2, respectively; T_3 is a Tripod type (with all of T_1 to T_3 possibly the same); $((T_1 \times T_2) \rightarrow T_3)$ is a binary mapping (possibly the identity); $((T_3 \times T_3) \rightarrow$ Bool$)$ is a binary predicate on T_3; and (Instant $\times$ String$^{\{\texttt{t,f,u}\}}$) is as for BUILD-IHC-STRUCTURE above. The BUILD-CHC-STRUCTURE function differs from the algorithm in Figure 3 only insofar as line 1 is replaced by the following two lines:

$$\begin{aligned} A &\leftarrow \texttt{Align}(H_1, H_2) \\ H' &\leftarrow \Cup\{(t, \mu(v_1, v_2)) \mid (t, (v_1, v_2)) \leftarrow A\}\end{aligned}$$

where H_1 and H_2 are the two input histories and μ is binary, rather than unary as in Figure 3. The rest of the algorithm behaves in the way described for IHC queries.

4.4 Identifying Patterns in CHC Queries

Because the interpretation structure for CHC queries is syntactically and semantically equivalent to that returned for IHC queries, it follows that

$$\text{BUILD-CHC-INTERVALS} \equiv \text{BUILD-IHC-INTERVALS}$$

i.e., the function defined in Figure 4 is applicable without change to CHC queries.

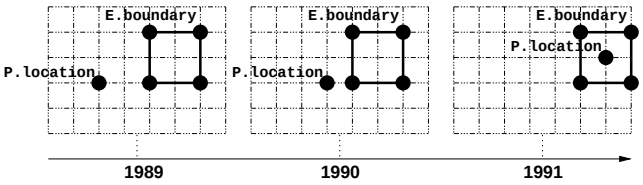

Figure 5: Aligned Histories of Spatial Objects

For example, assume two histories H_1 and H_2 of a Points and a Regions object, respectively (e.g., the geographical location of an LS member P and the boundary of an enumeration district E), both with Instants timestamp type and granularity YEAR. Assume the state sets of H_1 and H_2 to be such as can be induced from Figure 5, where the timestamps are marked in the horizontal axis and the underlying (contrivedly small) realm grid is shown at each timestamp.

A call such as BUILD-CHC-STRUCTURE$(H_1, H_2, \underline{\texttt{dist}}, >)$ would return the interpretation structure $\langle 1989, \texttt{tt}\rangle$, i.e, the two spatial objects grew closer between 1989 and 1990, and between 1990 and 1991 (when it became zero, according to the semantics of dist in the ROSE algebra [12]). This result relies on the alignment of the two histories which is already (graphically) reflected in Figure 5. Hence, the outcome is t between 1989 and 1990, and t between 1990 and 1991, and ES is tt . A call to BUILD-CHC-INTERVALS([t]+, BUILD-CHC-STRUCTURE$(H_1, H_2, \underline{\texttt{dist}}, >)$) identifies (non-empty) periods of indefinite length when the two objects grew closer at every granule. In the above example, the single match is denoted by $(0, 2)$. Given that the earliest instant is $e = 1989$, the interval set returned is $\{[(1989 + 0) : (1989 + 0 + 2)]\} = \{[1989 : 1991]\}$. This is the answer to query **Q2** if the histories of the location of LS member P and of the boundary of enumeration district E are, after alignment, as graphically depicted in Figure 5.

4.5 Interpretation Structure for CHI Queries

As is the case with CHC queries, the interpretation structure for CHI queries, i.e., evolution queries over the snapshots in a paired (non-empty) history at each timestamp, also requires aligning the two input histories.

The signature of the BUILD-CHI-STRUCTURE function defined in Figure 6 is History $< \tau, T_1 >$ ×History $< \tau, T_2 >$ $\times((T_1 \times T_2) \rightarrow (T_3 \times T_3)) \times ((T_3 \times T_3) \rightarrow \texttt{Bool}) \rightarrow (\texttt{Instant} \times \texttt{String}^{\{\texttt{t},\texttt{f},\texttt{u}\}})$

```
BUILD-CHI-STRUCTURE(H1, H2, μ, φ)
 1  H3 ← Align(H1, H2)
 2  H4 ← ⊎{(t, μ(v1, v2)) | (t, (v1, v2)) ← H3}
 3  H5 ← ⊎{(t, φ(v, v')) | (t, (v, v')) ← H4}
 4  ES ← ϵ
 5  if H5 has timestamp type Instants
 6     then e ← first(EarliestState(H5). timestamp )
 7     else  e ← start(first(EarliestState(H5). timestamp ))
 8  it ← new INSTANT-STATE-ITERATOR(H5)
 9  s ← it. get_element()
10  while ¬(it. at_end() )
11  do
12      it. next()
13      s' ← it. get_element()
14      d ← dist(s. timestamp , s'. timestamp )
15      if d ≠ 1
16         then ES ← ES^((d − 1) ∗ u)
17         else  if s. snapshot
18                  then ES ← ES^t
19                  else  ES ← ES^f
20      s ← s'
21  return ⟨e, ES⟩
```

Figure 6: Interpretation Structure for CHI Queries

The algorithm in Figure 6 resembles, but is significantly different from, the one in Figure 3. Besides the alignment step and the application of the binary mapping, which are also present in BUILD-CHC-STRUCTURE, there is an additional, crucial difference (with respect to BUILD-IHC-STRUCTURE): the comparator ϕ is applied not to snapshots in consecutive instants, but rather to the snapshots of two objects in the aligned history. This is captured in line 3 in Figure 6. Contrast this with the application of the comparator in line 15 in Figure 3. Because of this crucial difference, the iteration is not over the aligned (possibly μ-mapped) snapshots but rather over the history that results from applying the comparator to two snapshots at the same instant. In other words, the snapshot type of H_5 in Figure 6 is Boolean, and hence is tested as such in line 17 so as to concatenate to ES the character that properly represents the truth value resulting from the application of the comparator to the corresponding aligned snapshots. Another difference is the computation of the size of a gap in line 16, which also stems from the fact that H_5 already represents the outcome of applying the comparator ϕ.

4.6 Identifying Patterns in CHI Queries

Again, because the interpretation structure for CHI queries is syntactically and semantically equivalent to that returned for IHC (and CHC) queries, it follows that

$$\text{BUILD-CHI-INTERVALS} \equiv \text{BUILD-IHC-INTERVALS}$$

i.e., the function defined in Figure 4 that builds the intervals of interest for IHC queries is applicable without change to CHI queries. Consider again H_1 and H_2, the histories of the `location` of LS member P and of the `boundary` of enumeration district E in Section 4.4. A call such as BUILD-CHI-STRUCTURE$(H_1, H_2, \iota, \underline{\texttt{inside}})$ would return the interpretation structure $\langle 1989, \texttt{fft} \rangle$, i.e, the `Points` object was only `inside` the `Regions` object in 1991. This result relies on the alignment of the H_1 and H_2 which is already (graphically) reflected in Figure 5, on applying the identity mapping on the resulting paired snapshots, and on applying the comparator `inside` to the latter. Therefore, the history denoted by H_5 in Figure 6 has the following value in this case: $\{\langle [1989 : 1990], \texttt{false} \rangle, \langle [1990 : 1991], \texttt{false} \rangle, \langle [1991 : \texttt{now}], \texttt{true} \rangle\}$, where `now` denotes the fact that the snapshot for 1991 holds at the moment in which the query is evaluated (assumed here to be in the year 1992). Hence, the resulting value for ES is `fft`. A call to BUILD-CHI-INTERVALS(`[ft]+`, BUILD-CHI-STRUCTURE$(H_1, H_2, \iota, \underline{\texttt{inside}})$) is meant to identify paired instants such that the first object was not inside the second in the first instant, and then, in the second instant, it was. In other words, occasions where it can be said that the first object has entered the second. In the above example, the single match is denoted by $(1, 2)$. Given that the earliest instant is $e = 1989$, the interval set returned is $\{[(1989 + 1) : (1989 + 1 + 2)]\} = \{[1990 : 1992]\}$. This is the answer to query **Q3** if the histories of the `location` of LS member P and of the `boundary` of enumeration district E are, after alignment, as graphically depicted in Figure 5.

Note that Figure 2 only constitutes an exhaustive taxonomy at depth 1. As hinted by the dotted lines at depth 2, the kinds of evolution queries that can be formulated is far too large to be captured in a drawing. Nevertheless, the set of all evolution queries is precisely defined if the data model is. To see this, note that an evolution query is either an IHC, a CHC or a CHI query. Now, consider Figures 3 to 6. Note that, apart from the histories themselves, all the above queries are parameterized by a mapping μ and a predicate ϕ. Thus, the set of all evolution queries is well-defined given a set T of types, a set $M \ni \mu$ of mappings over T and a set $\Phi \ni \phi$ of comparators over T. In Tripod, all of T, M and Φ have such cardinalities that the set of all possible combinations of their elements is quite large. Now, taking a subset of such combinations characterizes one kind of Tripod evolution query that a taxonomy might attach an identifier to and depict as a node in Figure 2. Other settings than Tripod will have different extensions for T, M and Φ, but, again, given those, the set of evolution queries for each such setting is well-defined.

5. RELATED WORK

Erwig and Schneider present a formal model of spatio-temporal predicates in [5] that can be used to characterize how a topological relationship between two spatial objects unfolds in a temporal sequence. This notion, referred to by Erwig and Schneider as *developments* [3], has inspired the approach contributed in this paper. Erwig and Schneider have also devised a two-dimensional visual language [4] for specifying these temporally changing topological relationships, and translating them into a sequence of spatio-temporal predicates. However, their approach seems to focus on the representation of topological relationships, whereas the approach described in this paper can be used to capture a wider range of queries, both metric and topological, on spatial objects while also ranging over aspatial objects.

The *behavioural time sequences* model [19] for managing highly variable data (e.g., spatial objects that evolve over time) bears some similarity to the approach contributed in this paper. In [19] data is represented using time sequences, where each element of the sequence contains a geometric value, a date, and a behavioural function. The latter describes how the data evolves between two consecutive elements of the sequence. Evaluating spatio-temporal queries on highly variable data is then achieved by computational-

geometric techniques on a geometric representation of the sequence. While some of the ideas in [19] are also present in different forms in this paper, the question as to which behavioural functions are legitimate to apply in each particular system is not discussed in [19]. In contrast, this paper indicates precisely how the underlying type system is related to (and ultimately determines) the class of all evolution queries that an effective system supports.

In [17] an SQL-based query language for time series is proposed, called SQL-TS, which can be used to specify complex sequential patterns. The authors also describe an optimized pattern search algorithm for querying complex sequential patterns of aspatial data based on the text searching algorithms by Knuth, Morris and Pratt [14]. There are similarities with the algorithms in Section 4 for identifying patterns. However, because this paper expresses patterns as regular expressions, it is possible to answer a much larger class of queries than can be done in SQL-TS. Moreover, the approach presented here makes no distinction between spatial and aspatial data other than those that stem from the semantics captured by their respective algebras.

Other proposals exist for data models that capture objects whose properties (spatial and aspatial) are continuously changing. These models are typified by the *moving object* approach adopted in [11] and [16]. Such models allow the state of each spatial and aspatial property to be expressed as a continuous function of time. Queries about the position of spatial data can then be inferred by the interpolation of spatial values between known bounds [18]. This provides an expressive mechanism for the representation of moving points and polygons ([16] only considers points). It should be noted, however, that such models do not provide comprehensive support for temporally changing aspatial data and object model constructs such as relationships, which are supported in a uniform way by the Tripod data model. In contrast, the Tripod data model and calculus do not model continuous change, as they explicitly target applications in which objects change in discrete steps; for example cadastral, cartographic, and demographic ones.

6. CONCLUSIONS

Support for spatio-temporal queries is one of the primary functions of geographical information systems and spatio-temporal database systems. Simple spatio-temporal queries tend to be limited to producing information from stored data by simple retrieval (e.g., based on selection, projection, join, etc). In contrast, advanced spatio-temporal queries require mechanisms to capture and represent derived data in order to produce added-value information, e.g., about how a property or a relationship between two (possibly spatial) objects has evolved over time. This paper has characterized a class of spatio-temporal queries in which change patterns can be identified and has shown how to compute answers for such queries in the setting of Tripod, a complete spatio-temporal object database system under development by a team that includes the authors [7, 8, 9]. It has shown how the underlying type system determines, for a particular platform, the class of all evolution queries that it is legitimate to pose. The paper has thus contributed a general, detailed and well-founded account of how a class of advanced spatio-temporal query capabilities can be supported in applications.

Acknowledgements: This work is partly supported by a grant from the UK Engineering and Physical Sciences Research Council for which the authors are grateful.

7. REFERENCES

[1] M. Agesen, M. H. Böhlen, L. Poulsen, and K. Torp. A split operator for now-relative bitemporal databases. In *ICDE*, pages 41–50, 2001.

[2] R. G. G. Cattell, editor. *The Object Database Standard: ODMG 3.0*. Morgan Kaufmann, 2000.

[3] M. Erwig and M. Schneider. Developments in Spatio-Temporal Query Languages. In *IEEE Int. Workshop on Spatio-Temporal Data Models and Languages*, pages 441–449, 1999.

[4] M. Erwig and M. Schneider. Visual specifications of spatio-temporal developments. In *15th IEEE Symp. on Visual Languages*, pages 187–188, 1999.

[5] M. Erwig and M. Schneider. Spatio-temporal predicates. *TKDE*, 2002. to appear.

[6] L. Fegaras and D. Maier. Optimizing Object Queries Using an Effective Calculus. *TODS*, 25(4):457–516, 2000.

[7] T. Griffiths, A. A. A. Fernandes, N. Djafri, and N. W. Paton. A Query Calculus for Spatio-Temporal Object Databases. In *Proc. TIME*, pages 101–110, 2001.

[8] T. Griffiths, A. A. A. Fernandes, N. W. Paton, K. T. Mason, B. Huang, and M. Worboys. Tripod: A Comprehensive Model for Spatial and Aspatial Historical Objects. In *Proc. ER*, pages 84–102, 2001.

[9] T. Griffiths, A. A. A. Fernandes, N. W. Paton, K. T. Mason, B. Huang, M. Worboys, and C. Johnson. Tripod: A Comprehensive System for the Management of Spatial and Aspatial Historical Objects. In *Proc. ACM-GIS*, pages 118–123, 2001.

[10] S. Grumbach, P. Rigaux, M. Scholl, and L. Segoufin. The DEDALE Prototype. In G. Kuper, L. Libkin, and J. Paredaens, editors, *Constraint Databases*, pages 365–382. Springer, 2000.

[11] R. H. Güting, M. H. Böhlen, M. Erwig, C. S. Jensen, N. A. Lorentzos, M. Schneider, and M. Vazirgiannis. A Foundation for Representing and Querying Moving Objects. *TODS*, 25(1):1–42, 2000.

[12] R. H. Güting and M. Schneider. Realm-Based Spatial Data Types: The ROSE Algebra. *VLDB Journal*, 4(2):243–286, 1995.

[13] L. Hattersley and R. Creeser. *Longitudinal Study 1971-1991: History, Organization and Quality of Data*. Number 7 in ONS Series LS. The Stationery Office., London, 1995.

[14] D. E. Knuth, J. H. Morris, and V. R. Pratt. Fast Pattern Matching in Strings. *SIAM Journal of Computing*, 6(2):323–350, 1977.

[15] C. Parent, S. Spaccapietra, and E. Zimányi. Spatio-temporal conceptual models: Data structures + space + time. In C. B. Medeiros, editor, *Proc. ACM-GIS*, pages 26–33, 1999.

[16] D. Pfoser, C. S. Jensen, and Y. Theodoridis. Novel Approaches in Query Processing for Moving Object Trajectories. In *Proc. VLDB*, pages 395–406, 2000.

[17] R. Sadri, C. Zaniolo, A. Zarkesh, and J. Adibi. A Sequential Pattern Query Language for Supporting Instant Data Minining for e-Services. In *Proc. VLDB*, pages 653–656, 2001.

[18] E. Tøssebro and R. H. Güting. Creating Representations for Continuously Moving Regions from Observations. In *Proc. SSTD*, volume 2121 of *LNCS*, pages 321–344, 2001.

[19] T. S. Yeh and B. de Cambray. Modeling Highly Variable Spatio-Temporal Data. In *Proc. Australasian Database Conf.*, pages 221–230, 1995.