Mobility Patterns

Cédric du Mouza and Philippe Rigaux
CEDRIC lab., CNAM, 292 Rue St Martin, F-75141 Paris Cedex 03, France, dumouza@cnam.fr
LAMSADE lab., Univ. Paris-Dauphine, Place du Maréchal de Lattre de Tassigny,F-75775 Paris cedex 16, France, rigaux@lri.fr

December 1, 2004

Abstract. We present a data model for tracking mobile objects and reporting the result of queries. The model relies on a *discrete* view of the spatio-temporal space, where the 2D space and the time axis are respectively partitioned in a finite set of user-defined areas and in constant-size intervals. We define a generic query language to retrieve objects that match *mobility patterns* describing a sequence of moves. We also identify a subset of restrictions to this language in order to express only *deterministic* queries for which we discuss evaluation techniques to maintain incrementally the result of queries. The model is conceptually simple, efficient, and constitutes a practical and effective solution to the problem of continuously tracking moving objects with sequence queries.

Keywords: Mobility patterns, online evaluation, spatio-temporal applications.

1. Introduction

In the database community, several data models have been proposed to enable novel querying facilities over collections of moving objects. A common feature of most of these models is the strong focus on the *geometric* properties of trajectories. Indeed, in most cases, the data representation and the query language are considered as extensions of some existing data model previously designed for (and limited to) 2D geometric data handling. As a result, spatio-temporal data models rely usually on a set of data structures providing support for geometric operations (e.g., geometric intersection).

An assumption commonly adopted is to consider a *dense* embedding space and to model trajectories as *continuous* functions in this space. While this property allows several suitable computations (for instance the position of an object can be obtained at any instant), it is not well adapted to some analysis and classification tasks. In the present paper we investigate an alternative approach, namely the management of queries as a process relying on *events* related to the moves of objects over a discrete representation of the underlying space, called *reference space*. Intuitive examples of events are, for instance, an object *enters* a zone, an object *stays* in a zone, and an object *leaves* a zone. A query in

such a setting is a sequence of primitive events which can be specified either by explicitely referring to the zones of interest ("Give all the objects currently in `a` which arrived 5 minutes ago, coming from `b`"), or by more generic *patterns* of mobility such as, for instance, "Give the objects that moved from `a` to another zone and came back to `a`".

We introduce *mobility patterns* as expressions describing such sequences of events. In the present paper we examine specifically the following aspects of this framework:

- comparison and aggregation of moving objects trajectories,
- on-line classification of trajectories continuously provided by GPS-like devices.

We first consider historical data and the post-acquisition operators that allow to analyse the spatio-temporal behavior of objects belonging to a given population (e.g., taxis, planes, etc.) and to perform clustering and similarity-based analysis and comparisons. Analysis tools that allow to create spatio-temporal "profiles" of objects are certainly of interest to many applications. In the domain of traffic analysis for instance, this permits to better predict and understand the load of a local road network during a typical day. Public services can also be made more efficient when they can be proposed in accordance with the availability of users. The same holds for commercial marketing analysis.

Next we consider the tracking of objects with *continuous queries*, i.e., queries whose result must be maintained during a given (and possibly unbounded) period of time. When asking, for instance, for all the objects that belong to a given rectangle R during the next 3 days, the initial result is subject to vary by considering the objects that leave of enter R. Managing *incrementally* the evolutions of the result (i.e., without recomputing periodically the entire result) is a hard task with a geometric-based query language because the dense-space assumption of the data model often contradicts with the discrete nature of the observation. A trajectory for instance is obtained through sample points provided by the GPS system, and the continuous representation has to be inferred by interpolation between two sample points, or by extrapolation from the last known position (Sistla et al., 1997). Moreover, depending on the geometric operations required by the query, one might have to consult the past trajectory to check whether or not the object belongs to the result. Actually the few works that propose a solution to the problem deal with limited classes of queries (e.g., window and k-NN queries in (Mokbel et al., 2004a; Iwerks et al., 2004)).

We propose in the current paper a data model for representing trajectories as sequences of moves in a discrete spatio-temporal space,

and study the languages to query such sequences of events. Essentially, the languages that we consider rely on mobility patterns to express search operations. We focus specifically on the family of patterns that satisfy the following properties (i) we do not need the past moves of an object o to determine whether o matches or not a given pattern and (ii) the amount of memory required to maintain a query result is small. These properties are essential in the context of continuous queries since they guarantee that a large amount of queries can be evaluated efficiently with limited resources by just considering the last event associated to an object. We define a class of queries which provides an appropriate balance between expressiveness and the fulfillment of these requirements.

In the rest of this paper we first develop an informal presentation of our work (Section 2) with examples of mobility patterns that illustrate the intuition behind the model and its practical interest from the user's point of view. The data model is presented in Section 3. Section 4 surveys related work. Finally Section 5 concludes the paper and discusses future work.

2. Mobility patterns

In our model, the locations of objects are mapped to a set of zones which partition the area of interest. Clearly the considered partition is closely related to a specific thematic interpretation of space which resorts to the user's choice and constitutes a quite classical and common spatial analysis mechanism (Laurini and Thompson, 1992; Rigaux et al., 2001). Defining which partitions are relevant and which are not is beyond the scope of the present paper. Each zone is uniquely identified by a label from a set Σ, and we characterize the trajectory of an object o by the labels of the successive zones crossed by o.

Figure 2 shows a map partitioned in several zones, each one labelled with a symbol (`a, b, c, ...`). Over this map we describe the trajectories of a set of mobile objects. We assume in the following that the trajectory's description has been obtained from a GPS system which gives the sequence of locations, along with their timestamp.

Consider now an application that aims at classifying and analysing the traffic by issuing the following queries:

1. Give all the objects that traveled from `a` to `f`, stayed more than 10 minutes in `f` and then traveled from `f` to `c`.

2. Give all the objects traveling from `f` to `d` or `c` through another, third, zone of the map.

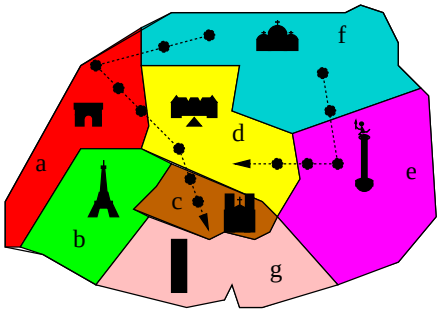

Figure 1. Objects moving over a partitioned map

3. Give all the objects that left a given zone, went to `c` and came back to the first zone.

The common feature of these examples is a specification of the successive zones an object belongs to during its travel, along with temporal constraints. We call *mobility pattern* this specification. The geometric-based approach used in most of the spatio-temporal data models so far is not really adapted for expressing queries based on mobility patterns, because it would require a lot of joins with the reference space. Actually we do no longer need an interpolation or extrapolation mechanism to infer the position of an object at each instant since the discrete succession of events provided by the GPS server is naturally suitable to serve as a support for evaluating these patterns.

Each GPS event provides the position of an object, and this suffices to compute the zone where the object resides when the event is received. It is therefore quite easy to construct a discrete representation of the trajectory of an object as a sequence of the form $l_1\{T_1\}.l_2\{T_2\}.\cdots.l_n\{T_n\}$ featuring the list $l_1, l_2, \ldots, l_n$ of successive zone labels as well as the time spent in each zone. For instance the trajectory of o_1 in Figure 2, assuming that o_1 spent 2 minutes in `f`, 4 minutes in `a`, 3 minutes in `d` and 6 minutes in `c`, will be represented in our model as a sequence `[f{2}.a{4}.d{3}.c{6}]`. Note that each new event either increments the time component of the last label if the object remains in the same zone, or appends a new label to the trajectory's representation.

2.1. Queries

Mobility patterns are used to query this database and retrieve the objects that match a pattern. Basically, they consist of sequences of expressions built from a small set of operators and featuring either labels from Σ or *variables* from a set $\mathcal{V}$. Variables can be instantiated

to any of the labels of the map, and allow to denote complex classes of trajectories within a compact and expressive language. The presentation that follows is mostly based on examples, intended to illustrate its main features.

As a first example, assume that we want to retrieve all the objects that started from `b`, moved to `e`, crossing one of the cities `c`, `d`, or `f` (see Figure 2), and finally went from `e` to `a` via the *same* city. This is expressed as follows in our language:

```
start_at b, follow @x, follow e repeat, follow @x.a
```

This mobility pattern is built with two operators: `start_at` and `follow`, the latter being optionally qualified with `repeat`. Two other operators, `now_at` and `roam` will be presented in further examples.

A zone is represented by its label (here `a`, `b`, `e`) or by a variable (here `@x`) if it is left undetermined by the user. A variable is here necessary to represent the city where an object moved after leaving `b`, and expressing that the object must come back to `a` via the *same* city. Labels or variables can be concatenated (for instance `@x.a` in our example) to describe a path, and labels (but not variables) can be grouped in sets (for instance {`a, b`}) to describe a union of zones.

Users can specify a `start_at` operator to indicate where a trajectory is supposed to begin. Another operator, `now_at`, indicates where an object is required to be at the present moment. Without such statements, the mobility pattern is only required to match any subpart of the trajectory.

A `follow` operator describes a move which can be repeated with the `repeat` statement either for a fixed number of times, or many times but at least one. In our example the `follow e repeat` means that an object that matches the pattern moves to `e` and stays there for an unbounded period of time before coming back to `a`.

Intuitively, an object *o matches* the pattern P above if the following conditions hold:

1. there exists a valuation of the variables in P such that the valued pattern is a substring of the object's trajectory;
2. if the pattern begins with `start_at` (resp. ends with `now_at`), the valued pattern is a prefix (resp. a suffix) of the trajectory;
3. the time spent in each zone complies with the temporal constraint expressed in the pattern.

For instance an object whose trajectory is represented by the sequence of zone labels **[b.d.e.d.a**`.c.f]` (we omit the temporal information for simplicity) matches the pattern above where the value of the

variable `@x` is set to the label `d`. The string in boldface is a valuation of P, prefix of the discrete trajectory representation.

More generally, mobility patterns denote classes of trajectories. An object o matches a pattern P if a substring of its discrete trajectory representation belongs to the class denoted by P. We shall provide in the following precise definitions of the syntax and meaning of patterns. Let us first give some other intuitive examples that will be used throughout the rest of the paper to illustrate the model.

– **Q1.** Give all the objects travelling from `a` to `f` and then from `f` to `c` in 10 minutes.

```
start_at a, follow f, roam 10, follow c
```

If the trajectory of an object o matches the pattern, then: `start_at a` means that o starts from `a`; `follow f` means that o leaves `a` for `f`; `roam 10` denotes that the object moves in any zones of the map during 10 minutes. The `roam` operator is neutral and never restricts in any way the trajectory of an object: leaving `f`, it is possible to wander in different zones before reaching `c`, yet being qualified to the result of the query.

– **Q2.** Give all the objects that stayed in `a` or `b` all the time except for one minute when they were in another, third, zone.

```
start_at {a,b}, follow {a,b} repeat,
          follow @x, follow {a,b} repeat;
@x != a, @x!= b
```

This example requires a variable `@x` which expresses a move from `a` or `b` to any other zone of the map. It is possible to express additional constraints on the possible instantiations of a variable, using equalities or inequalities. The user requires in this example the object to leave `a` or `b` for a third area.

– **Q3.** Give all the objects that went through `f` to another zone then went to `d` or `c`, and came back to `f` using the *same* zone.

```
follow f.@x, follow {d,c}, follow @x.f;
@x != f
```

Recall that a variable sticks to its value once instantiated.

This language brings new querying facilities which would be quite difficult, if not impossible, to express or compute with a geometric based spatio-temporal query language, because of the many spatial joins necessary to determine the zone where an object resides at a given instant, and more importantly because of the sequential aspect of the queries. We provide now the grammar of the user language. First we define the following types:

ZONE $\rightarrow x$, with $x \in \Sigma$
ZONESET $\rightarrow \{x_1, \ldots, x_k\}$ with, $k \geq 1$, and $\forall i, x_i \in \Sigma$
VAR $\rightarrow v$, with $v \in \mathcal{V}$
ZONESTRING $\rightarrow x_1 \ldots x_k$, with $k \geq 1$, and $\forall i, x_i \in \Sigma$
VARSTRING $\rightarrow x_1 \ldots x_k$, with $k \geq 1$, and $\forall i, x_i \in \Sigma \cup \mathcal{V}$

A mobility pattern is built with the following grammar, where INT denotes a positive integer token.

```
START_BLOC  → start_at ZONESET | start_at VAR | ε
NOW_BLOC    → now_at ZONESET | now_at VAR | ε
BLOC         → follow ZONESET REPEAT_BLOC BLOC
                          | follow VARSTRING REPEAT_BLOC BLOC
                          | roam INT | roam | ε
REPEAT_BLOC → repeat INT | repeat | ε

QUERY        → START_BLOC BLOC NOW_BLOC
```

2.2. Continuous queries

The query language defined above can be used to analyse and classify objects trajectories stored in a database. It provides also a support for *continuous* queries. However the peculiarity of such queries entails some restrictions. Indeed, an object can be added or removed from the result set during the query lifetime, depending on its most recent moves. Under our modeling perspective, this means that the mobility patterns relevant for a continuous evaluation are those that end with `now_at`: only those objects whose trajectory's suffix, at the current instant, match the pattern, are included in the result.

As a first example, assume that we want to retrieve all the objects that went from `a` or `b`, moved to `e`, crossing one of the zones `c`, `d`, or `f` (see Figure 2), and finally went back from `e` to `a` via the *same* zone where it actually lies. This query is expressed with our language by:

```
follow {a,b}, follow @x repeat,
            follow e repeat, follow @x repeat, now_at a;
@x != f
```

The continuous evaluation aims at matching the pattern with a suffix of the objects' trajectory. The suffix represents here the most recent part of the continuous stream of GPS events. Since the trajectory representation evolves as new events are received, the matching must be evaluated periodically – almost continuously. Our goal is to perform this evaluation with minimal space and time consumption. We consider two essential criteria for measuring the easiness and efficiency of this evaluation:

1. Do we need to consider the past moves of an object to evaluate a query?
2. What is the amount of memory required to maintain a query result?

Consider first the case of patterns without variables. Evaluating a pattern P is then a standard operation which simply requires to build the Finite State Automaton (FA) that recognizes the language $\Sigma^*.L_P$ (Hopcroft and Ullman, 1979), where L_P is the regular language denoted by P and Σ is the set of labels of the map.

In the general case, the FA associated to a regular expression is non-deterministic. Then an object o might be associated to several states at a given time instant, and we must record the list of current states for o. This list can be represented as a mask of bits, one bit for each state of the FA. The value 1 (resp. 0) for a bit means that o is (resp. is not) in the associated state. This gives a rather compact structure: for a pattern with 8 symbols, a mask of 8 bits (one byte) must be recorded for each object. One can track a database of one million objects with only one megabyte in main memory.

The pseudo-code of the procedure *HandleEvent(q, id, x, y)* summarizes how to update the result of a query q when a GPS event is received, giving a new location (x, y) for the object o. The reference map is a set of zones denoted by M.

HandleEvent (q, o, x, y)
begin
  // Compute the current zone, z
  z = *PointInPolygon(M, x, y)*
  // Get the label of z
  l = *label(z)*
  // For each bit set to 1 in the status of o,
  // compute the transition l
  for each bit i with value 1 in $status_o$
    Compute $s_j = \delta(FA_q, s_i, l)$
    Set the bit j to 1 and the bit i to 0 in $status_o$

end for
end

The result set of q can then be updated according to the new status of object o. Essentially, if at least one of the new states is an accepting one, o will be *in* the result set, else it will be *out* of this result set. In this simple case we obtain a direct answer to the two questions above:

1. It is *not* required to maintain historical information on a trajectory, since, it suffices to know the current state(s) of the FA, reached by taking account of the events received so far.

2. The space required to maintain a query result is, in the worst case, the set of all states in the FA (which might be non-deterministic) and is therefore proportional to the size of the query.

If we consider now patterns with variables, the language is much more expressive, but some care is required for executing queries. Take for instance the example Q3 above. Each time an object leaves the zone `f` for another one, a new label is bound to the variable `@x`. One must then store this value in order to check for the consistency of any further occurrence of `@x`.

The next section is devoted to the data model, and focuses on the evaluation of queries with variables. We show that we can still avoid to rely on historical information on trajectories, and study more specifically the memory requirements for several classes of queries.

3. The model

We consider an embedding space partitioned in a finite set of zones, each zone being uniquely labeled with a symbol from a finite alphabet Σ. The time axis is divided in constant size units. For concreteness we still assume in the following that the time unit is 1 minute. We also assume a set $\mathcal{V}$ of variables with $\Sigma \cap \mathcal{V} = \emptyset$ and denote as Γ the union $\Sigma \cup \mathcal{V}$. In the following, letters `a`, `b`, `c`, ... will denote symbols from Σ, and `@x`, `@y`, `@z`, ... variables. We assume the reader familiar with the basic notions of regular expressions and regular languages, as found in (Hopcroft and Ullman, 1979).

3.1. Data representation and query language

We adopt a standard extended relational framework for the database, with $\mathcal{O}$ denoting the relation of moving objects, and $o.traj$ the trajectory of an object o. The representation of trajectories is then defined as follows:

DEFINITION 1 (Representation of trajectories). *A* trajectory *is represented by an expression of the form*

$$s_1\{T_1\}.s_2\{T_2\}.\cdots.s_n\{T_n\}$$

where $s_i, i = 1, \ldots, n$ *are symbols from* Σ *and* T_i *represents the number of time units spent in the zone* s_i*.*

Hereafter, we shall use the term "trajectory" to mean its representation. For convenience, we shall often omit the temporal components and use a simplified representation of a trajectory as a word $[s_1.s_2.\cdots.s_n]$ in Σ^*.

A natural choice is to build mobility patterns as regular expressions on $\Gamma = \Sigma \cup \mathcal{V}$, and to search for the substring of trajectories that match the expression for some value of the variables. Consider for example the regular expression $E =$`a.@x+.b+.@x`. The trajectory $t =$`f.d.a.c.b.c.b` matches E because we can find a word $w =$ `a.@x.b.@x` in the language denoted by E (w is called a *witness* in the following) and a valuation $\nu : \{$`@x` $:= c\}$ such that $\nu(w)$ is a substring of t. However this approach raises some ambiguities regarding the role of variables. Consider the following examples:

1. Let E be the regular expression `b.(a|@x)+.c`. Then the trajectory `b.a.c` has two witnesses in the regular language denoted by E: `b.@x.c` and `b.a.c`. In the first case `@x` must be valued to `a`, but in the second case any value of `@x` is acceptable.

2. Let E be the regular expression `a.(@x|@y).b.(@x|@y)`. The variables `@x` and `@y` can be used interchangeably, which makes the role of variables ambiguous.

As shown by the previous examples, if we build mobility patterns with unrestricted regular expressions over Γ, the assignment of variables is non deterministic, and sometimes meaningless. For safety reasons, when reading a word w and checking whether w matches a mobility pattern P, we require each variable in P to be explicitely bound to one of the symbols in w. We thus adopt a more rigorous definition of the language by considering only *unambiguous* regular expressions on Γ such that each variable always plays a role in the evaluation of the query. We need first to introduce *marked* regular expressions.

DEFINITION 2 (Marked expressions (Book et al., 1971)). *Let* E *be a regular expression over the alphabet* Γ*. We define the* marking *of* E *as the regular expression* E' *where each symbol of* Γ *is marked by a subscript over* $\mathbb{N}$*, representing the position of the symbol in the expression.*

The marking of the regular expression $\texttt{a}^*\texttt{.@x.((b.a)|(c.b)).c.@x}^*\texttt{.a}$ is for instance the expression $\texttt{a}_1^*\texttt{.x}_2\texttt{.((b}_3\texttt{.a}_4\texttt{)|(c}_5\texttt{.b}_6\texttt{)).c}_7\texttt{.@x}_8\texttt{.a}_9$. We can now define *mobility patterns* as the class of regular expressions that satisfy the following property:

DEFINITION 3 (Mobility patterns). *A* mobility pattern *is a regular expression P over Γ such that each variable of P' appears in each word of the language $\mathcal{L}(P')$.*

This property ensures that each variable in any pattern is always assigned to a relevant label during query evaluation. The expression P `= (a|b)+.@x.(a|b)+` is for instance a mobility pattern because `@x` appears in all the words of the language $\mathcal{L}(P)$. Any successful matching of P with a trajectory `t` results therefore in an assignment of `@x` to one of the symbols of `t`. It can be tested whether a regular expression matches the required condition, and thus can be used as a mobility pattern.

PROPOSITION 1. *There exists an algorithm to check whether a regular expression is a mobility pattern.*

Proof (sketch): Let E be a regular expression. Then $\mathcal{L}(E)$ and $\mathcal{L}(E')$ are regular languages. We define the language $\mathcal{L}_m = \{\Gamma^*.\texttt{@x}_1.\Gamma^*.\texttt{@x}_2.\cdots.\texttt{@x}_k.\Gamma^*\}$, where Γ stands for $\Sigma \cup \mathcal{V}$, and $\texttt{@x}_1\texttt{,..,@x}_k$ are the variables of E'. $\mathcal{L}_m$ is regular by construction, and so are $\overline{\mathcal{L}_m}$ and $\mathcal{L}(E') \cap \overline{\mathcal{L}_m}$. Therefore the fact that $\mathcal{L}(E') \cap \overline{\mathcal{L}_m}$ is empty is decidable. And if $\mathcal{L}(E') \cap \overline{\mathcal{L}_m} = \emptyset$, then all the marked variables appear in all the words of $\mathcal{L}(E')$. □

EXAMPLE 1. *The following regular expressions represent the mobility patterns of the sample queries **Q1**, **Q2** and **Q3** given in Section 2.*

1. $P_1 = \texttt{a.f\{2,\}.c}$
2. $P_2 = \texttt{(a|b)}^+\texttt{.@x.(a|b)}^+$
3. $P_3 = \texttt{f.@x}^+\texttt{.(c|d)}^+\texttt{.@x}^+\texttt{.f}$

The user query language proposed in section 2.1 allows to construct expression that can be transformed in mobility patterns via the following interpretation function [.] :

1. $[x] = x, x \in \Sigma$
   $[v] = v, v \in \mathcal{V}$
   $[x_1 \ldots x_k] = x_1 \ldots x_k, x_i \in \Sigma \cup \mathcal{V}$
   $[\{x_1, \ldots, x_k\}] = (x_1 | \ldots | x_k)$

2. Interpretation of a query:
   [START_BLOCK BLOCK END_BLOCK]=[START_BLOCK].[BLOCK].[END_BLOCK]

3. Interpretation of the START_BLOCK:
   [start_at ZONESET]=[ZONESET]
   [start_at VAR]=[VAR]
   $[\varepsilon]=\Sigma^*$

4. Interpretation of the NOW_BLOCK:
   [now_at ZONESET]=[ZONESET]
   [now_at VAR]=[VAR]
   $[\varepsilon]=\Sigma^*$

5. Interpretation of the BLOCK:
   [follow ZONESET repeat INT BLOCK]=$[\text{ZONESET}]^{[INT]}$.[BLOCK]
   [follow ZONESET repeat BLOCK]=$[\text{ZONESET}]^{+}$.[BLOCK]
   [follow VARSTRING repeat INT BLOCK]=$[\text{VARSTRING}]^{[INT]}$.[BLOCK]
   [follow VARSTRING repeat BLOCK]=$[\text{VARSTRING}]^{+}$.[BLOCK]
   [follow ZONESET BLOCK]=[ZONESET].[BLOCK]
   [follow VARSTRING BLOCK]=[VARSTRING].[BLOCK]
   [roam INT]=$\Sigma^{[INT]}$
   [roam]=Σ^*

The interpretation of a query is a mobility pattern. However the user query language does not capture all the mobility patterns, as illustrated by the following example. Let E =b.(a|c$^+$).@x. E is a mobility pattern since the variable @x appears in all the words of $\mathcal{L}(E)$. However our language does not allow to use the + operator in a *or* expression.

The pattern @x.a.b.c.@x denotes the family of regular languages that consists of words in Σ^* with exactly 5 letters, the first one being equal to the last one, separated by a.b.c. A mobility pattern P denotes a regular language $\mathcal{L}(P) \subseteq \Gamma^*$. More generally a mobility pattern P with k variables is equivalent to the union of $|\Sigma|^k$ regular expressions enumerating the $|\Sigma|^k$ possible combinations of variables values. Variables give an exponentially concise way of expressing such languages.

In the following we shall denote as $var(P)$ the set of variables in a pattern P. The query language and its semantics are now defined as follows.

DEFINITION 4 (Syntax of queries). *A* query *is a pair* $(\mathcal{P}, \mathcal{C})$ *where* $\mathcal{P}$ *is a mobility pattern and* $\mathcal{C}$ *is a set of constraints of the form* $s_1 \neq s_2$, *with* $s_1, s_2 \in \Sigma \cup var(P)$

Let q be a query of the form (P, LC) where LC is a list of constraints $\{C_1, \ldots C_l\}$. The result of q, denoted $q(\mathcal{O})$, is a subset of $\mathcal{O}$ defined as follows:

DEFINITION 5 (Semantics of queries). *An object o belongs to $q(O)$ if there exists a mapping $\nu : \mathcal{V} \rightarrow \Sigma$, called a* valuation*, with the following properties:*

1. *ν satisfies all the constraints $C, \forall C \in LC$.*
2. *$o.traj \in \nu(\mathcal{L}(P))$*

The constraints in a query can be used to forbid explicitely a variable to take a value (e.g., `@x` $\neq$ `a`). The *domain* of a variable `@x` for a given query q, denoted $dom_q($`@x`$)$, represents the set of possible values for `@x` given the constraints of q.

EXAMPLE 2. *The following queries correspond to the examples given in Section 2.*

1. q_1 = `({a.f{2,}.c}, ∅)`
2. q_2 = `({(a|b)`$^+$`.@x.(a|b)`$^+$`}, {@x ≠ a, @x ≠ b })`
3. q_3 = `({f.@x`$^+$`.(c|d)`$^+$`.@x`$^+$`.f}, {@x ≠ f})`

3.2. Query evaluation

We describe now an algorithm for evaluating a query q. First we show how to obtain an automaton which, given a mobility pattern P, accepts the trajectories that match P. This automaton also provides the valuation of variables in P. In a second step we explain how the automaton can be used at runtime, and discuss the size of the memory used to store the relevant information. For simplicity, we consider the automata that accept the language $\mathcal{L}(P)$: their extension to automata that accept $\Sigma^*.\mathcal{L}(P)$ is trivial and can be found in any specialized textbook.

Since a mobility pattern P is a regular expression over the alphabet Γ, we can build a non-deterministic finite state automaton (NFA) N_Γ that accepts the language of Γ^* denoted by P. Starting from N_Γ we can build a new automaton, N_Σ, which checks whether a trajectory t of Σ^* belongs to $\nu(\mathcal{L}(P))$, and delivers the valuation ν.

Essentially, N_Σ is N_Γ with a management of variable bindings based on the following extensions: (i) a transition labeled with a variable `@x` on a symbol α sets the value of `@x` to α if `@x` was not yet bound and (ii) with each state one maintains the bindings of the variables met so far. Transitions from s_i to s_j, labeled with a variable `@x`, are then interpreted as follows:

1. If `@x` is bound to α in s_i, and the current symbol of the input word is α, then s_j can be reached and the binding of s_j is identical to the binding of s_i.

2. If `@x` is not bound in s_i, and the current symbol of the input word is α, then s_j can be reached and the binding of s_j is the binding of s_i augmented with $@x \leftarrow \alpha$.

The definition of N_Σ is given below.

- The set of states of N_Σ, $states(N_\Sigma)$, is $states(N_\Gamma) \times \Sigma^{|var(P)|}$, i.e., all the possible associations of a state of N_Γ with a valuation ν of the variables in P. A state of N_Σ is denoted $< S, \nu >$.

- The set of accepting states of N_Σ, $accept(N_\Sigma)$ is $accept(N_\Gamma) \times \Sigma^{|var(P)|}$.

- The transition function of N_Σ, δ_Σ, is drawn from the transition function of N_Γ, δ_Γ, as follows:

  - if $\delta_\Gamma(S_i, \alpha) = S_j$ is a transition of N_Γ with $\alpha \in \Sigma$, then $\delta_\Sigma(< S_i, \nu >, \alpha) =< S_j, \nu >$. In other words the transition has no effect on variable bindings.
  - if $\delta_\Gamma(S_i,$`@x`$) = S_j$ is a transition of N_Σ , then $\delta_\Sigma(< S_i, \nu >, \alpha) =$
    $\begin{cases} < S_j, \nu +$`@x`$:= \alpha >$ if $\nu($`@x`$)$ is undetermined and the binding of `@x` with α is allowed by the constraints. \\ < S_j, \nu >$ if $\nu($`@x`$) = \alpha$. \\ is undefined otherwise. \end{cases}$

Whenever an accepting state $< S, \nu >$ of N_Σ is reached, the input trajectory is accepted and the valuation ν defines the valuations of all the variables (recall that, by definition, any word in a language defined by a mobility pattern contains all the variables).

3.3. Evaluation of continuous queries

In order to check at run time whether an object o matches a mobility pattern, we do not need to fully construct the automaton described above. Instead, we start with a minimal representation, and build in a progressive way, according to the symbols appended to the trajectory of o, the valuation of the variables which potentially leads to an accepting state. The initial representation of N_Σ consists only of the set of states of N_Γ, each associated with the empty valuation. By keeping all the current states of N_Σ associated with o, the following operations can

be performed whenever a new move m is appended to $o.traj$ to test whether o enters, stays or quits the query result:

1. If the transitions labeled with m lead o to at least one accepting state, then o becomes part of the result of the query.
2. If the transitions labeled with m are such that o is no longer in at least one accepting state, then it must be removed from the result of the query.

This yields a first, convenient, property for the evaluation of continuous queries: the last move suffices to deliver the information needed to maintain a query result. Here is an example that illustrates the process (more details can be found in the long version).

EXAMPLE 3. *Consider the mobility pattern* $P = (a|b)^+.@x.(a|b)^+$. *Figure 3 shows an NFA automaton* N_Γ *which recognizes the words of* $\mathcal{L}(P)$, S_0 *being the initial state and* S_4, S_5 *the final states.*

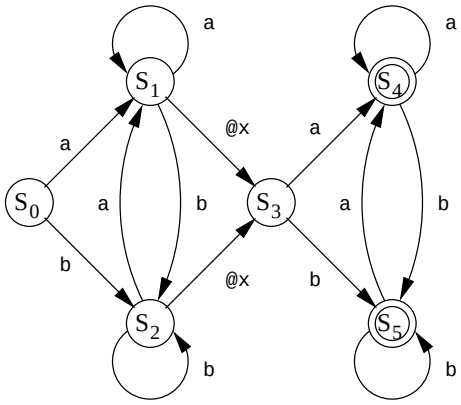

Figure 2. An automaton for the mobility pattern `(a|b)`$^+$`.@x.(a|b)`$^+$

Assume that one receives successively the following events for an object o: `a`, `a`, `b`, `b`, `c` *and* `a`. *Each row in the table of the figure 3 shows the states of the NFA* N_Σ *after reading a symbol, as well as the possible valuations of variable* `@x`. *The accepting states are in bold font and mean that the trajectory belongs to the query result set.*

Example 3 shows that we might have to maintain, during the analysis of an input trajectory, several valuations associated to a same state. In the worst case one might have $|states(N_\Gamma)| \times |\Sigma^k|$ simultaneous states to maintain, representing all the possible valuations of variables that lead to an accepting state. Consider the following pattern:

Input	Reached states in N_Σ
a	$< S_1$,@x=$\perp >$
a[2]	$< S_1$,@x=$\perp >, < S_3$,@x=a$>$
a[2].**b**	$< S_2$,@x=$\perp >, < S_3$,@x=b$>$, $<$ **S$_5$**,**@x=a**$>$
a[2].**b**[2]	$< S_2$,@x=$\perp >, < S_3$,@x=b$>, <$ **S$_5$**,**@x=a**$>, <$ **S$_5$**,**@x=b**$>$
a[2].**b**[2].**c**	$< S_3$,@x=c$>$
a[2].**b**[2].**c**.**a**	$<$ **S$_4$**,**@x=c**$>$

Figure 3. Evaluation of a query

$$\texttt{@x}^+.\texttt{@y}^+.\texttt{@z}^+$$

It is no difficult to find a word such that @x, @y and @z take all their possible valuations.

Depending on the application, the size of the database and the number of queries, maintaining a large amount of informations to continuously evaluate a query might become costly. In some cases we might therefore want to restrict the expressive power of the language to obtain low memory needs. Consider for instance a web server providing a subscribe/publish mechanism over a (possibly large) set of moving objects. In such a system, web users can register queries, waiting for notification of the results. The performance of the system, and in particular its ability to serve a lot of queries under an intensive incoming of events, depends on the efficiency of the query result maintenance, and therefore on the size of the data required to perform this maintenance. We define below a fragment of the query language which meets the requirement of this kind of application.

3.4. Deterministic queries

The class of *deterministic* queries is such that, at any instant, there is only one possible valuation for each variable of the mobility patterns. Deterministic queries are defined by the following property:

DEFINITION 6 (Deterministic queries). *A query* $q(P,\mathcal{C})$ *is* deterministic *iff* $\forall u, v \in (\Sigma \cup \mathcal{V})^*, \forall \texttt{@x} \in \mathcal{V}, u.\texttt{@x}.v \in \mathcal{L}(P) \Rightarrow \nexists \alpha \in dom_q(\texttt{@x}),$ / $\exists w \in (\Sigma \cup \mathcal{V})^*, u.\alpha.w \in \mathcal{L}(P)$.

The intuition is that when it becomes possible to instantiate a variable during the analysis of a trajectory, then this transition is the only

possible choice. This makes the binding of variables deterministic, and ensures that, for a given word, there is only one (if any) possibility to instantiate a variable.

EXAMPLE 4. *The following examples illustrate deterministic queries.*

- *The query* $q(\texttt{f.@x.(c|d).@x.f}, \emptyset)$ *is deterministic. Whenever a* `f` *symbol has been read, the only possible choice is to bind* `@x` *to the symbol that follows immediately* `f`.
- *The query* $q(\texttt{(a|b)}^+\texttt{.@x.(a|b)}^+, \emptyset)$ *is non-deterministic since the words* `a.@x.a.b` *and* `a.b.@x.b` *both belong to* $\mathcal{L}(P)$. *However* $q'(\texttt{(a|b)}^+\texttt{.@x.(a|b)}^+, \{\texttt{@x}\neq\texttt{a},\texttt{@x}\neq\texttt{b}\})$ *is deterministic.*

PROPOSITION 2. *There exists an algorithm to check whether a query is deterministic.*

Proof (sketch): Let q be a query, P be a mobility pattern in q and N_Γ a deterministic automaton which recognizes $\mathcal{L}(P)$. Since N_Γ is deterministic, for any input string we reach at most one state s of N_Γ. If one can find a state s with two transitions: $\delta(s,\texttt{@x}) = s'$ and $\delta(s,\alpha) = s''$, with $\alpha \in dom_q(\texttt{@x})$, then it suffices to check whether there exists two words $@x.u$ and $\alpha.v$ which both permit to reach a final state from s. If this is the case, then q is not deterministic. □

PROPOSITION 3. *Let* $q(P,\mathcal{C})$ *be a deterministic query. Then, for each word* w *of* Σ^*, *there is at most one witness of* w *in* $\mathcal{L}(P)$.

Consider again the queries of Example 4. In the first example an accepted word can only have one single witness, either `f.@x.d.@x.f` or `f.@x.c.@x.f`. In the second example, with constraints {`@x`≠`a`,`@x`≠`b`}, any witness consists of two words of {`a`,`b`}$^+$, separated by a symbol distinct from `a` or `b`.

It follows that if $q(P,\mathcal{C})$ is a deterministic query, the memory space required to check whether a word matches q is $|P| + |var(P)|$, where $|P|$ represents the number of symbols in P. Essentially, we need one FA for q, plus a storage for each variable, and we can build a FA with a number of states equal to the number of symbols in the expression.

When evaluating a continuous query, we need to maintain for each object o the set of its current states, as well as the binding of variables and this suffices to determine, at each GPS event, whether o enters, stays or quits the query result.

The proofs of the properties for mobility patterns rely on the *Glushkov automaton* of regular expressions (Book et al., 1971; Bruggemann-Klein and Wood, 1998). Given a regular expression E, we first introduce the following definitions:

- $first(E) = \{\alpha \mid$ there is a word w such that $\alpha.w \in \mathcal{L}(E)\}$
- $last(E) = \{\alpha \mid$ there is a word w such that $w.\alpha \in \mathcal{L}(E)\}$
- $follow(E, \omega) = \{\alpha \mid$ there are words v and w such that $v.\omega.\alpha.w \in \mathcal{L}(E)$ $\}$ for each symbol ω.

Basically, a Glushkov automaton possesses as many states as marked symbols of the expression it stands for. Each incoming transition to a state s_i is labelled by the unmarked symbol s. The definition presented by Book *et al.*, ((Book et al., 1971)) is the following:

DEFINITION 7. *The Glushkov automaton of a regular expression E is the automaton $G_E = (Q', \Sigma, \delta', q_I, F')$ with:*

1. *q_I is the initial state*
2. *$Q' = sym(E') \cup \{q_I\}$*
3. *For $\alpha \in \Gamma, \delta'(q_I, \alpha) = \{x | x \in first(E'), x^\natural = \alpha\}$*
4. *For $\sigma_i \in sym(E')$ and $\alpha \in \Gamma, \delta'(\sigma_i, \alpha) = \{x | x \in follow(E', x)$ and $x^\natural = \alpha\}$*
5. *$F' = last(E')$*

The Glushkov automaton G_E recognizes the langage denoted by E (Book et al., 1971). The number of states in G_E is equal to the number of symbols in E.

EXAMPLE 5. *Figure 5 shows the Glushkov automaton G_E for the regular expression $E =$* `a((xa)|(bx))*b`*, whose marking is* $\mathtt{a}_1(\mathtt{(x}_2\mathtt{a}_3\mathtt{)|(b}_4\mathtt{x}_5\mathtt{))}^*\mathtt{b}_6$.

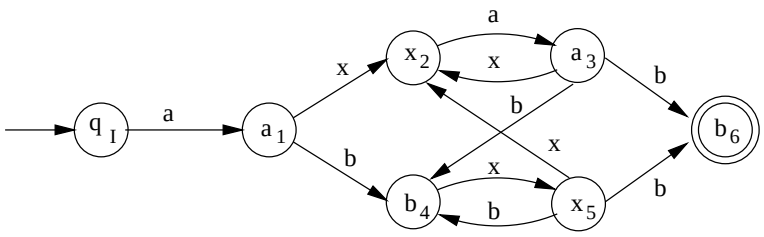

Figure 4. Glushkov automaton of `a((xa)|(bx))*b`

The following can now be obtained from the definition of the Glushkov automaton.

PROPOSITION 4. *Let P be a mobility pattern. Then for any two words* $\mathtt{w}_1$,$\mathtt{w}_2$ *of* $\mathcal{L}(P)$*, the marked variables appear in the same order in* $\mathtt{w}_1$ *and* $\mathtt{w}_2$*.*

Proof (sketch): Assume there exists two accepted words $\mathtt{w}_1$ and $\mathtt{w}_2$ such that $\mathtt{w}_1$=$\mathtt{p}_1\mathtt{.@x}_1\mathtt{.q}_1\mathtt{.@x}_2\mathtt{.r}_1$ and $\mathtt{w}_2$=$\mathtt{p}_2\mathtt{.@x}_2\mathtt{.q}_2\mathtt{.@x}_1\mathtt{.r}_2$ and $\mathtt{@x}_1$ and $\mathtt{@x}_2$ do not appear in $\mathtt{p}_1$ and $\mathtt{p}_2$. If $\mathtt{@x}_1$ does not appear in $\mathtt{r}_1$, since $\mathtt{r}_1$ is a path from the state $\mathtt{@x}_2$ of the Glushkov automaton to a final state, then $\mathtt{p}_2\mathtt{.@x}_2\mathtt{.r}_1$ is an accepted word. This raises a contradiction because this word does not contain $\mathtt{@x}_1$. A similar reasonning shows that we obtain the same contradition by assuming that $\mathtt{@x}_1$ appears in $\mathtt{r}_1$. □

It follows that, if P is a mobility pattern in a deterministic query, for any input string in Σ^*, there is only one possible valuation for each variable of P. The memory space required to check whether a word matches P is therefore $|states(N_\Gamma)| + |var(P)|$, where N_Γ is the Glushkov automaton of P.

The N_Γ automaton is non-deterministic, so in the worst case all the states of the automaton can be reached simultaneously. In addition we need to store the instanciation of variables. Since variables are instanciated in a known order, a list of $|var(P)|$ memory units is sufficient.

The following example illustrates this property. Indeed, it shows that for a given input, only one state can be reached, and that we just have to store the unique valuation of `x`.

EXAMPLE 6. *Let us consider again the query $q(P,\mathcal{C})$, with* P = `(a|b)`$^+$`.@x.(a|b)`$^+$ *and* $\mathcal{C}$ = {`@x`$\neq$`a`,`@y`$\neq$`b`}*. The automaton remains identical (see Figure 3) but the evaluation on input* `a.a.b.b.c.a` *is now given in Table I.*

The properties of deterministic queries ensure that the required amount of memory is independent from the size of Σ, and thus of the underlying partition of space used to describe the trajectories of moving objects. This property might be quite convenient if the space of interest is very large, or if the number of queries to maintain is such that the memory usage becomes a problem.

Expressions of deterministic queries are simple and can therefore easily be introduced in a SQL-based query language such as SQL, extended with a `matches` boolean operator, as illustrated by the following examples.

– **Q1.** Give all the objects that traveled from `a` to `f`, stayed at least 2 minutes in `f` and then traveled from `f` to `c`.

Table I. Evaluation of a deterministic query

Input	Reached states in N_Σ	Transitions not allowed
a	$< S_1$,@x=$\perp >$	
a[2]	$< S_1$,@x=$\perp >$	$< S_3$,@x=a> since a $\notin dom(@x)$
a[2].**b**	$< S_2$,@x=$\perp >$	$< S_3$,@x=b> since b $\notin dom$(@x)
a[2].**b**[2]	$< S_2$,@x=$\perp >$	$< S_3$,@x=b> since b $\notin dom$(@x)
a[2].b[2].**c**	$< S_3$,@x=c>	
a[2].b[2].c.**a**	$< \mathbf{S_4}$,**@x=c**>	

```
SELECT *
FROM Mob
WHERE traj matches(a.f{2,}.c)
```

The *matches* function checks whether a suffix of the spatio-temporal attribute *traj* matches the mobility pattern `a.f.c`. An additional temporal constraint states that the object must spend at least 2 time units (e.g., 2 minutes) in `f`.

– **Q2.** Give all the objects that stay in `a` or `b` all the time except for one minute when they were in another, third, zone.

```
SELECT *
FROM Mob
WHERE traj matches('(a|b)+.@x.(a|b)+')
AND    @x != 'a' AND @x != 'b'
```

This example requires a variable `@x` which expresses a move not assigned to a specific label but instantiated to the choice of a moving object when it leaves `a` or `b`. It is possible to express additional constraints on the instantiations allowed for a variable, using equalities or inequalities. The user requires in this example the object to leave `a` or `b` for a third, distinct, area.

– **Q3.** Give all the objects that went through `f` to another zone then went to `d` or `c`, and came back to `f` using the *same* zone.

```
SELECT *
FROM   Mob
WHERE  traj matches('f.@x+.(d|c)+.@x+.f')
AND    @x != 'f'
```

4. Related work

The modeling of moving objects has been strongly influenced by the existing spatial models. Representative examples are (Sistla et al., 1997; Güting et al., 2000; Forlizzi et al., 2000; Grumbach et al., 2000; Su et al., 2001; Vazirgiannis and Wolfson, 2001; Güting et al., 2003; Gupta et al., 2004; Ding and Güting, 2004; Sun et al., 2004). The fundamental work of (Forlizzi et al., 2000) and (Güting et al., 2000) is based on spatiotemporal abstract types and extends SQL to query spatiotemporal data at an abstract level. The model is powerful to represent and query the past and present position of an object. The *MOST* model ((Sistla et al., 1997), (Wolfson et al., 1999)) supplies a query language named *FTL* that enables the specification of queries that refer to future states of the database. Other works ((Chomicki and Revesz, 1997), (Grumbach et al., 2000)) use the constraint databases framework and are convenient for representing and manipulating trajectories as infinite sets of positions. Some relevant models ((Gupta et al., 2004; Ding and Güting, 2004)) have also been proposed to manage the peculiar problem of moving objects on a constraint network. For instance (Gupta et al., 2004) describes an evaluation technique based on hypercube graphs to compute the shortest paths. (Sun et al., 2004) presents approximation techniques for query processing to interrogate the past, the present and the future of objects trajectories. The authors propose a multi-dimensional histogram incrementally updated for queries at present time, histories for queries concerning the past, and a predictive technique for queries pertaining to future positions.

Expressing sequences of moves as proposed in the present paper is close in spirit to the area of sequence databases (Seshadri et al., 1995; Mecca and Bonner, 1995; Sadri et al., 2001a; Sistla et al., 2002). The SQL-TS language of (Sadri et al., 2001a) and (Sadri et al., 2001b) allows to express sequences of conditions. The paper describes an efficient algorithm for query evaluation. The idea of representing temporal sequences as strings, relying on pattern-matching algorithms for query evaluation, is also present in (Dumas et al., 1998) and (Djafri et al., 2002). The system explores the consecutive snapshots contained in the spatio-historical database, creates an evolution string and looks for a

matching of this string with the histories of the objects. In (Ramakrishnan et al., 1998) sequences are considered as sorted relations, and each tuple gets a number that represents its position in the sequence. A shift operator using this number is defined in order to join tuples of the same sequence. Aggregation operators on sequences for a given range of sequencing numbers is also presented. Some other works (Kim et al., 2002; Goldin and Kanellakis, 1995; Guralnik and Srivastava, 1999; Chen and Ng, 2004; Law et al., 2004) present algorithms for querying and mining similar subsequences, as well as event detection from time series data (i.e., sequences of real numbers). (Law et al., 2004) for instance describes the necessary restrictions of the SQL language when dealing with streams, focusing on the problem of the aggregation. After defining the "nonblocking" queries, the paper defines aggregates (*UDA*) that can be applied on stream data. Nonetheless all these approaches are significantly different from ours. In particular there is nothing similar to the concept of mobility pattern, featuring variables, proposed in our data model.

The notion of continuous queries, described as queries that are issued once and run continuously, is first proposed in (Terry et al., 1992). The approach considers append-only databases and relies on an incremental evaluation on delta relations. Availability of massive amounts of data on the Internet has considerably increased the interest in systems providing event notification across the network. Some representative works are the *Active Views* system (Abiteboul et al., 1999), the NiagaraCQ system (Chen et al., 2000), and the prototypes described in (Liu et al., 1999; Fabret et al., 2001). In the area of spatio-temporal databases, the problem is explicitely addressed in several works (Brinkhoff and Weitkämper, 2001; Kalashnikov et al., 2002; Tao et al., 2002; Mokbel et al., 2004b; Mokbel et al., 2004a; Xiong et al., 2004; Iwerks et al., 2004; Jensen et al., 2004). (Brinkhoff and Weitkämper, 2001) for instance describes a web-based architecture for reducing the volume and frequency of data transmissions between the client and the server. (Kalashnikov et al., 2002) presents a system that indexes queries in order to recompute periodically the whole result of each query. This is in contrast with the incremental computation advocated in the current paper. (Iwerks et al., 2004) describes an evaluation method to maintain the result of a kNN-query by retrieving all the objects within a given range that may affect in a close future the result of the query. (Mokbel et al., 2004b; Mokbel et al., 2004a; Xiong et al., 2004) discuss algorithms to incrementally evaluate a set of continuous queries that consist of three steps: (i) join with the cache for new incoming tuples (positive updates), (ii) invalidate some results (negative updates) after a timeout, (iii) join with the stored data (positive and negative updates). (Jensen

et al., 2004) presents an index based on the B^+-tree, called the B_x-tree, that allows an efficient evaluation of continuous, range and k-NN queries. They reconsider the concept of conservative approximation by taking account of the enlargement of the queries' regions.

5. Conclusion and further work

We described in this paper a new approach for querying and tracking a moving object database by means of mobility patterns. Our proposal is based on a data model which allows to retrieve objects whose trajectory matches a parameterized sequence of moves expressed with respect to a set of labeled zones. We investigated the applicability of the model to continuous query evaluation, showed how to maintain incrementally the result of a query, and identify a fragment of the query language such that the amount of space required to maintain this result is very low.

A simplified version of the language can easily be introduced as complement of a geometric-based extension of SQL, as shown by the query samples proposed in Section 3. The properties of the language make it a convenient candidate for mobile object tracking based on sequences patterns, and its simplicity leads to an easy implementation.

A prototype is being developed in order to assess the relevancy of this approach in a web-based context where a lot of clients can register queries, receive an initial result set, and wait for notification of updates to this result set. In particular we are currently working on optimization techniques for patterns expressed as words in Γ^*. We believe that standard pattern matching techniques (Knuth et al., 1977; Crochemore and Rytter, 1994) can be extended to such parameterized strings, thereby allowing an evaluation by a simple scan of the input trajectory, without additional memory requirements.

This framework raises several interesting research issues that we plan to investigate in a near future: approximate similarities between trajectories represented as strings, extension of the query language to express temporal and topological constraints on the zones of a pattern, and finally multi-resolution sequences. Indeed, introducing multi-scale patterns is likely to raise the power of the classification and analysis aspects of our model.

Acknowledgments: we are very grateful to D. Vodislav and D. Gross-Amblard for early discussions on this model.

References

Abiteboul, S., B. Amann, S. Cluet, A. Eyal, L. Mignet, and T. Milo: 1999, 'Active Views for Electronic Commerce'. In: *Proc. Intl. Conf. on Very Large Data Bases (VLDB)*.

Book, R., S. Even, S. Greibach, and G. Ott: 1971, 'Ambiguity in Graphs and Expressions'. *IEEE Transactions on Computers* **20**(2), 149–153.

Brinkhoff, T. and J. Weitkämper: 2001, 'Continuous Queries within an Architecture for Querying XML-Represented Moving Objects'. In: *Proc. Intl. Conf. on Large Spatial Databases (SSD)*.

Bruggemann-Klein, A. and D. Wood: 1998, 'One-Unambiguous Regular Languages'.

Chen, J., D. DeWitt, F. Tian, and Y. Wang: 2000, 'NiagaraCQ: A Scalable Continuous Query System for Internet Databases'. In: *Proc. ACM SIGMOD Symp. on the Management of Data*.

Chen, L. and R. T. Ng: 2004, 'On The Marriage of Lp-norms and Edit Distance'. In: *Proc. Intl. Conf. on Very Large Data Bases (VLDB)*. pp. 792–803.

Chomicki, J. and P. Z. Revesz: 1997, 'Constraint-Based Interoperability of Spatiotemporal Databases'. In: *Proc. Intl. Conf. on Large Spatial Databases (SSD)*, Vol. 1262 of *Lecture Notes in Computer Science*. pp. 142–161, Springer.

Crochemore, M. and W. Rytter: 1994, *Text algorithms*. Oxford University Press.

Ding, Z. and R. H. Güting: 2004, 'Managing Moving Objects on Dynamic Transportation Networks'. In: *Proc. Intl. Conf. on Scientific and Statistical Databases (SSDBM)*. pp. 287–296.

Djafri, N., A. Fernandes, N. W. Paton, and T. Griffiths: 2002, 'Spatio-Temporal Evolution: Querying Patterns of Change in Spatio-Temporal Databases'. In: *Proc. Intl. Symp. on Geographic Information Systems*. pp. 35–41.

Dumas, M., M.-C. Fauvet, and P.-C. Scholl: 1998, 'Handling Temporal Grouping and Pattern-Matching Queries in a Temporal Object Model'. In: *Proc. Intl. Conf. on Information and Knowledge Management*. pp. 424–431.

Fabret, F., H. Jacobsen, F. Llirba, K. Ross, and D. Shasha: 2001, 'Filtering Algorithms and Implementations for Very Fast Publish/Subscrib Systems'. In: *Proc. ACM SIGMOD Symp. on the Management of Data*.

Forlizzi, L., R. Güting, E. Nardelli, and M. Schneider: 2000, 'A Data Model and Data Structures for Moving Objects Databases'. In: *Proc. ACM SIGMOD Symp. on the Management of Data*.

Goldin, D. Q. and P. C. Kanellakis: 1995, 'On Similarity Queries for Time-Series Data: Constraint Specification and Implementation'. In: *Proceedings of the 1st International Conference on Principles and Practice of Constraint Programming (CP'95)*.

Grumbach, S., P. Rigaux, and L. Segoufin: 2000, 'Manipulating Interpolated Data is Easier than you Thought'. In: *Proc. Intl. Conf. on Very Large Data Bases (VLDB)*.

Gupta, S., S. Kopparty, and C. V. Ravishankar: 2004, 'Roads, Codes and Spatiotemporal Queries'. In: *Proc. ACM Symp. on Principles of Database Systems*. pp. 115–124.

Guralnik, V. and J. Srivastava: 1999, 'Event detection from time series data'. In: *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*. pp. 33–42.

Güting, R. H., M. H. Böhlen, M. Erwig, C. S. Jensen, N. A. Lorentzos, E. Nardelli, M. Schneider, and J. R. R. Viqueira (eds.): 2003, *Spatio-temporal Models and*

Languages: An Approach Based on Data Types. Spatio-Temporal Databases. The CHOROCHRONOS Approach.

Güting, R. H., M. H. Böhlen, M. Erwig, C. S. Jensen, N. A. Lorentzos, M. Schneider, and M. Vazirgiannis: 2000, 'A Foundation for Representing and Quering Moving Objects'. *ACM Trans. on Database Systems* **25**(1), 1–42.

Hopcroft, J. and J. Ullman: 1979, *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley.

Iwerks, G. S., H. Samet, and K. Smith: 2004, 'Maintenance of Spatial Semijoin Queries on Moving Points'. In: *Proc. Intl. Conf. on Very Large Data Bases (VLDB)*. pp. 828–839.

Jensen, C. S., D. Lin, and B. C. Ooi: 2004, 'Query and Update Efficient B^+-Tree Based Indexing of Moving Objects'. In: *Proc. Intl. Conf. on Very Large Data Bases (VLDB)*. pp. 768–779.

Kalashnikov, D., S. Prabhakar, W. Aref, and S. Hambrusch: 2002, 'Efficient Evaluation of Continuous Range Queries on Moving Objects'. In: *Proc. Intl. Conf. on Databases and Expert System Applications (DEXA)*. pp. 731–740.

Kim, S.-W., J. Yoon, S. Park, and T.-H. Kim: 2002, 'Shape-based retrieval of similar subsequences in time-series databases'. In: *Proceedings of the 17th symposium on Proceedings of the 2002 ACM symposium on applied computing*. pp. 438–445.

Knuth, D., J. Morris, and V. Pratt: 1977, 'Fast Pattern Matching in Strings'. *SIAM J. Computing* **6**(2), 323–350.

Laurini, R. and D. Thompson: 1992, *Fundamentals of Spatial Information Systems*, No. 37 in The A.P.I.C. Series. New York: Academic Press.

Law, Y.-N., H. Wang, and C. Zaniolo: 2004, 'Query Languages and Data Models for Database Sequences and Data Streams'. In: *Proc. Intl. Conf. on Very Large Data Bases (VLDB)*. pp. 492–503.

Liu, L., C. Pu, and W. Tang: 1999, 'Continual Queries for Internet Scale Event-Driven Information Delivery'. *IEEE Transactions on Knowledge and Data Engineering* **11**(4), 610–628.

Mecca, G. and A. J. Bonner: 1995, 'Finite Query Languages for Sequence Databases'. In: *Proc. Intl. Workshop on Database Programming Languages*.

Mokbel, M. F., X. Xiong, and W. G. Aref: 2004a, 'SINA: Scalable Incremental Processing of Continuous Queries in Spatio-temporal Databases'. In: *Proc. ACM SIGMOD Symp. on the Management of Data*.

Mokbel, M. F., X. Xiong, W. G. Aref, S. E. Hambrusch, S. Prabhakar, and M. A. Hammad: 2004b, 'PLACE: A Query Processor for Handling Real-time Spatio-temporal Data Streams.'. In: *Proc. Intl. Conf. on Very Large Data Bases (VLDB)*. pp. 1377–1380.

Ramakrishnan, R., D. Donjerkovic, A. Ranganathan, K. S. Beyer, and M. Krishnaprasad: 1998, 'SRQL: Sorted Relational Query Language'. In: *Proc. Intl. Conf. on Scientific and Statistical Databases (SSDBM)*. pp. 84–95, IEEE Computer Society.

Rigaux, P., M. Scholl, and A. Voisard: 2001, *Spatial Databases*. Morgan Kaufmann.

Sadri, R., C. Zaniolo, A. M. Zarkesh, and J. Adibi: 2001a, 'Optimization of Sequence Queries in Database Systems'. In: *Proc. ACM Symp. on Principles of Database Systems*.

Sadri, R., C. Zaniolo, A. M. Zarkesh, and J. Adibi: 2001b, 'A Sequential Pattern Query Language for Supporting Instant Data Mining for e-Services'. In: *Proc. Intl. Conf. on Very Large Data Bases (VLDB)*.

Seshadri, P., M. Livny, and R. Ramakrishnan: 1995, 'SEQ: A Model for Sequence Databases'. In: *Proc. IEEE Intl. Conf. on Data Engineering (ICDE)*. pp. 232–239.

Sistla, A., O. Wolfson, S. Chamberlain, and S. Dao: 1997, 'Modeling and Querying Moving Objects'. In: *Proc. IEEE Intl. Conf. on Data Engineering (ICDE)*. pp. 422–433.

Sistla, A. P., T. Hu, and V. Chowdhry: 2002, 'Similarity Based Retrieval from Sequence Databases using Automata as Queries'. In: *Proc. Intl. Conf. on Information and Knowledge Management*. pp. 237–244.

Su, J., H. Xu, and O. Ibarra: 2001, 'Moving Objects: Logical Relationships and Queries'. In: *Proc. Intl. Conf. on Large Spatial Databases (SSD)*. pp. 3–19.

Sun, J., D. Papadias, Y. Tao, and B. Liu: 2004, 'Querying about the Past, the Present, and the Future in Spatio-Temporal'. In: *Proc. IEEE Intl. Conf. on Data Engineering (ICDE)*. pp. 202–213.

Tao, Y., D. Papadias, and Q. Shen: 2002, 'Continuous Nearest Neighbor Search'. In: *Proc. Intl. Conf. on Very Large Data Bases (VLDB)*. pp. 287–298.

Terry, D., D. Goldberg, D. Nichols, and B. Oki: 1992, 'Continuous Queries over Append-Only Databases'. In: *Proc. ACM SIGMOD Symp. on the Management of Data*.

Vazirgiannis, M. and O. Wolfson: 2001, 'A Spatiotemporal Model and Language for Moving Objects on Road Networks'. In: *Proc. Intl. Conf. on Large Spatial Databases (SSD)*, Vol. 2121 of *LNCS*. pp. 20–35.

Wolfson, O., A. P. Sistla, B. Xu, J. Zhou, and S. Chamberlain: 1999, 'DOMINO: Databases fOr MovINg Objects tracking'. In: *Proc. ACM SIGMOD Symp. on the Management of Data*. pp. 547–549. (Demo sessions).

Xiong, X., M. F. Mokbel, W. G. Aref, S. E. Hambrusch, and S. Prabhakar: 2004, 'Scalable Spatio-temporal Continuous Query Processing for Location-aware Services.'. In: *Proc. Intl. Conf. on Scientific and Statistical Databases (SSDBM)*. pp. 317–326.