# Multi-scale Classification of Moving Objects Trajectories*

Cédric du Mouza
CEDRIC
CNAM
dumouza@cnam.fr

Philippe Rigaux
LRI
Université Paris-Sud
rigaux@lri.fr

## Abstract

*In this paper we propose a classification model for moving objects trajectories. We assume that the classification is based on a multi-scale map, and we simply define a* trajectory pattern *as the sequence of zones an object crosses during its travel. These patterns constitute the basis of classification operators. We also define a pattern-based query language which allows an online and continuous classification of moving objects. Finally a prototype which shows the validity of the approach is briefly described.*

## 1. Introduction

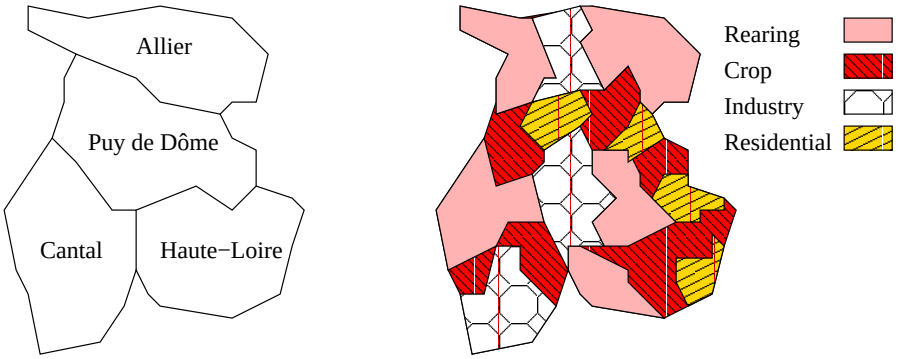
The management of moving objects is nowadays a well-established area of research in the GIS community [12]. The convergence of Internet, location-aware devices, wireless communications and GIS functionalities gives rise to new services and generates large databases of locational and time-stamped information. Because of the peculiar nature of the data involved, location-based services and applications raise many new research challenges, among which data modeling, indexing of continuously evolving locations, uncertainty, notification and routing services, etc. In the present paper we address the issue of classifying, comparing and aggregating moving objects trajectories, i.e., sequences of locations. 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. Next we address tracking applications which receive real-time information provided by GPS-like devices, and describe a query language which allows to express classification queries and to evaluate their result continuously.

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. In essence, we aim at taking a set of apparently erratic trajectories in order to organize and to classify them according to classes of thematically meaningful *patterns*. Each pattern describes the behavior of a typical class of objects, and the analysis process is supported by operators to create new patterns, to map trajectories to existing patterns, and to perform an on-line classification based on a set of pattern-based queries.

Another important requirement, addressed in the present paper, relates to the influence of resolution, or *scale*, on the description of trajectories. Indeed, taking account of a user-specified scale level leads to quite distinct interpretations. The differences are blurred at a small scale, and trajectories will tend to be merged together during the classification process. On the contrary, a large scale will allow to distinguish from one another two trajectories with slightly distinct features. Moreover one might want to describe a trajectory with a high resolution for some parts (e.g., the begining and end), and a low resolution for others. Depending on his needs, the user of the system must be able to tune the level of scale to the appropriate value, making therefore the system much more powerful. Our model copes with these requirements.

The first contribution of the present paper is a model that allows to classify trajectories with respect to a multi-scale representation of the domain area, each scale level being a partition of the domain in zones. We simply define a *trajectory pattern* to be a sequence of zones, and interpret it as the set of trajectories which

* Work partially supported by the French CNRS project MOTIF.




a. Administrative partition b. Space utilization

Fig. 1: Several partitionings of the same area

cross these zones in the required order. Depending on the scale level of interest, patterns will capture more or less similar trajectories. We formalize this intuition, exhibit dependencies between patterns expressed at distinct scale levels, and give classification algorithms.

Next we describe a pattern-based query language which allows an on-line classification of moving objects based on their trajectories. We show that the underlying partition in zones permits to handle the continuous evaluation of these queries as a discrete process. More specifically, we describe a simple and efficient mechanism to test at any instant whether an object belongs to the result of a query, and show how to maintain incrementally the query result. We also describe briefly a prototype which constitutes a practical application of our model to moving object tracking.

In the rest of this paper, we first (Section 2) introduce the multi-scale modeling of space which is used as a basis for pattern descriptions. The classification model is presented in Section 3, and pattern-based queries in Section 4. The Section 5 briefly describes the prototype and Section 6 discusses related work. Section 7 concludes the paper and outlines future work. Proofs are omitted, due to space restrictions. The interested reader is referred to the report [7], available at *http://www.lri.fr/~rigaux/DOC/MR04.pdf*.

## 2. The Reference Map

As mentioned in the introduction, our goal is to determine, for a given population, some representative "patterns" of behavior with respect to the spatial organization of a specific area $A \subseteq \mathbb{R}^2$ in zones. In the following, we define this spatial organization to be a *multi-scale partitioning* of $A$ and formally introduce the concept of scale level which will be used latter on to support trajectory patterns. This space modeling is partially borrowed from [18].

Recall that a partition of $A$ is a finite set of zones $\{z_1, z_2, \ldots, z_n\}$ such that $\bigcup_n z_i = A$ and $z_i \cap z_j = \emptyset, i \neq j$. Figure 1 shows an area (roughly the region of Auvergne, in France) with two distinct partitionings. The first one (left part) corresponds to an administrative partition (in *départements*) whereas the second one (right part) considers space utilization.

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 [14]. Defining which partitions are relevant and which are not is beyond the scope of the present paper. For our concerns, it suffices to note that each object moving in the partitioned area will cross successively a sequence of distinct zones: we shall build our patterns from such sequences. Note that, since the partition constitutes the basis of patterns definition, we will obtain quite distinct classifications of trajectories depending on the thematic focus. A thematically "neutral" partition consists in building patterns on a grid-like partition in equal-sized cells, as discussed in [15].

Let us now turn our attention to the issue of multi-scale representation. We shall here consider more specifically the case where switching from a scale level $S_1$ to a coarser one $S_2$ is done by *aggregating* the zones in a partition $p_1$, forming new, larger zones in a partition $p_2$. This is a quite natural and common situation. Consider again the area of Figure 1 and its two partitions. Switching from the scale level of *départements* to the scale level of *régions* will lead to a single zone representing the whole area. The same concept holds for the partition shown on the right part of Figure 1: we could gather *Residential* areas with *Industrial* areas to form *Built* areas, and *Crop* areas with *Rearing* areas to form *Agricultural* areas. In both cases, this results in *coarser* partitions made of fewer, less differentiated zones. We can therefore introduce the following relation of partitions: if $p_1, p_2$ are two partitions of $A$, we say that $p_1$ is *finer* than $p_2$, denoted

$p_1 \sqsubseteq p_2$, iff $\forall z \in p_1, z' \in p_2, z \cap z' \in \{\emptyset, z\}$.

We simply define a *reference map* to be the zones of a multi-scale partitioning $p_1 \sqsubseteq p_2 \ldots \sqsubseteq p_n$ of the area of interest.

**Definition 1** *Let $A \in \mathbb{R}^2$ be a subset of the plane, and $\{p_1, \ldots, p_n\}$ be a set of partitions of $A$, with $p_i \sqsubseteq p_{i+1}$, for $i < n$. A* reference map $\mathcal{M}_A$ *of $A$ is the set of zones in $p_1 \cup p_2 \ldots \cup p_n$.*

Figure 2 (left part) shows a reference map $\mathcal{M}$ based on the three partitions *cities*, *département* and *région* (for simplicity the figure shows only a sample of the cities, and gives them the pseudo-names $X$, $Y$, etc.). This set of zones is structured by an inclusion relationship. For convenience, we shall represent this structure based on a controlled vocabulary (right part of the figure) which consists of a set of *symbols* $\Sigma$ together with a *partial order* on symbols. Each symbol $a \in \Sigma$ is used as a unique identifier of a zone $z \in \mathcal{M}$.

**Definition 2** *A reference map $\mathcal{M}$ is represented by a pair $(\Sigma, \preceq)$ where $\Sigma$ is a finite and non-empty set of symbols and $\preceq$ is a partial order over $\Sigma$ such that $z_1 \subseteq z_2$ iff $id(z_1) \preceq id(z_2)$.*

We shall consider that the graph of $\preceq$ is a tree $T_\Sigma$, and we shall denote as $root(\Sigma)$ the root of $T_\Sigma$. Hereafter, we shall often confuse the reference map and its representation, i.e., we shall use the terms "reference map" to mean its representation. Moreover we shall say that a location $l$ belongs to $a \in \Sigma$ to mean that $l$ belongs to the zone identified by $a$.

We are now ready to define formally the concept of scale level.

**Definition 3 (Scale level)** *Let $(\Sigma, \preceq)$ be a reference map. The set of scale levels of $\Sigma$, denoted $\mathcal{S}_\Sigma$, is the subset of $2^\Sigma$ recursively defined as follows:*

1. *$root(\Sigma) \in \mathcal{S}_\Sigma$;*
2. *Let $S \in \mathcal{S}_\Sigma$ and $a \in S$ such that $sons(a) \neq \emptyset$, then $S - \{a\} \cup sons(a) \in \mathcal{S}_\Sigma$.*

This constructive definition provides an easy way to obtain the set of scale levels, as shown by the following example.

**Example 1** *The following are the scale levels of the reference map of Figure 2:*

- $S_{root}$ = `{Auvergne}`
- $S_{dept}$ = `{A, C, P, H}`
- $S_1$ = `{U, V, W, C, P, H}`
- $S_2$ = `{A, C, P, X, Y, Z}`
- $S_{leaves}$ = `{U, V, W, C, P, X, Y, Z}`

We shall always consider the classification of objects with respect to a specific scale level since, intuitively, such a level provides the necessary and sufficient set of symbols to describe a trajectory at a specific level of detail. Indeed, one shows easily that if $l$ is any location of the map, then there exists, in each scale level $S$, a unique symbol $a$ such that $l \in a$. In other words, if we choose a scale level $S \in \mathcal{S}_\Sigma$ and consider the location of a moving object, there exists a function $label(S, l)$ that associates a unique symbol in $S$ to $l$.

**Example 2** *Consider the location $l$ in Figure 2, and the scale levels of Example 1. For the scale level $S_{root}$, $l$ is mapped to the zone $label(S_{root}, l)$=`Auvergne`, for $S_{dept}$ and $S_2$, $l$ is mapped to `A`, finally for $S_1$ and $S_{leaves}$, $l$ is mapped to $U$.*

A scale level $S$ can be seen as a "slice" in the tree $T_\Sigma$ which divides $\Sigma$ in two subsets: one below $S$, and the other (strictly) upon $S$. In the following, given a subset $\sigma \subseteq \Sigma$, we shall denote as $cover(\sigma)$ the minimal scale level $S$ which "covers" $\sigma$, i.e., $\forall a \in \sigma, \exists a' \in S | a \preceq a'$.

## 3. Trajectories and Patterns

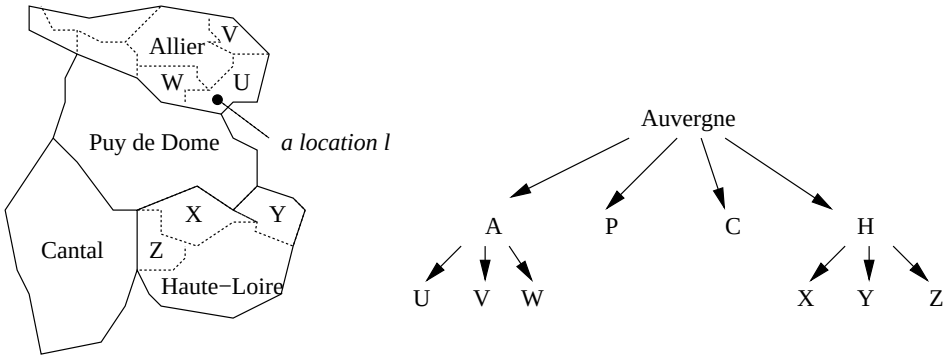
In our model, a moving object is represented by an identifier together with a *trajectory*. A trajectory is simply a sequence of locations. Formally, we assume the existence of a countably finite set *Obj* for identifiers. Locations are points in the Euclidean space $\mathbb{R}^2$.

**Definition 4 (Representation of moving objects)** *A moving object consists of an identifier $o$ together with a list $o.traj =< l_1, l_2, \ldots, l_n >$ of locations.*

Clearly, the description of a trajectory can be obtained from a GPS device which periodically provides the object's location. It is important to note that in our model the continuous aspect of trajectories is ignored because it is not relevant for our purposes. Since we map each location to a zone in a map, we just have to assume that the GPS device will provide at least one location for each zone crossed by an object $o$.

Let us now turn our attention to the classification of objects. As mentioned earlier, this classification relies on a reference map $(\Sigma, \preceq)$ which is structured in scale levels. We simply define a *trajectory pattern* as a sequence of zone symbols from a specific scale level. In the following definition, the composition operator $P.P'$ denotes the operation $reduce(concat(P, P'))$ where *concat* is the string concatenation, and $reduce()$ is a function which removes from a string all the repeated symbols (i.e., $reduce('aabbcccbbdd') =' abcbd'$).

**Definition 5 (Trajectory patterns)** *Let $S$ be a scale level in $(\Sigma, \preceq)$. Then:*




a. The reference space b. Its representation

Fig. 2: A multi-scale reference map and its representation

- *if $a \in S$, then* `[a]` *is a trajectory pattern*
- *if $P_1$ and $P_2$ are trajectory patterns in $S$, then so is $P_1.P_2$*

*We will denote by $\mathcal{P}_\Sigma$ the set of all trajectory patterns in $(\Sigma, \preceq)$.*

Note that we eliminate the repeated symbols from a pattern because such a repetition does not provide any useful information as long as we cannot guarantee that the events related to an object are separated by constant time intervals.

**Example 3** *The following are examples of trajectory patterns:*

$P_1 =$ `[Z.X.P.U]` $\quad P_2 =$ `[C.Z.P.W]`
$P_3 =$ `[H.P.U]` $\quad P_4 =$ `[H.P.A]`

Our motivations for introducing the concept of patterns relate to the clustering of trajectories that present a similar behavior. A pattern describes a class of trajectories. For instance if we consider the Figure 2, the pattern `[H.P.A]` represents all the trajectories that go from `Haute-Loire` to `Allier`, crossing the `Puy de Dôme`. Given a scale level $S$, a trajectory is mapped to a pattern in $S$ as follows:

**Definition 6** *Let $t =< l_1, l_2, \ldots, l_n >$ be the trajectory of a moving object $o$, and let $S$ be a scale level. The* pattern *of $t$ in $S$, denoted pattern$(t, S)$, is:*

$$[reduce(label(S, l_1).label(S, l_2) \cdots label(S, l_n))]$$

*where label() and reduce() are the functions introduced above.*

A trajectory $t$ *matches* a pattern $P$ at scale $S$ if $pattern(t, S) = P$. Depending on the scale level, patterns are matched by more or less trajectories, and thus provide a more or less accurate classification framework. Let us take an example, still based on Figure 2. The pattern `[U.V.P.X.Y]` of the scale level $S_{leaves}$ is matched by all the trajectories that go from cities `U`, then `V` in `Allier`, to cities `X` then `Y` in `Haute-Loire`, crossing the `Puy de Dôme`. Clearly, any trajectory that matches this pattern will also match the pattern `[A.P.H]` at the scale level $S_{dept}$ and, conversely, if a trajectory does *not* match `[A.P.H]`, then it will not match `[U.V.P.X.Y]`. Some patterns are therefore generalizations of others, and this leads to the following relation on $\mathcal{P}_\Sigma$:

**Definition 7 (Refinement relation on patterns)** *The* refinement relation on patterns, $\trianglelefteq$, *is defined by induction as follows.*

1. *if $P$ and $P'$ are two patterns of the form $P = [b_1.b_2.\cdots.b_n]$ and $P' =$* `[a]`*, with $b_i \preceq a$ for $i = 1, \ldots n$, then $P \trianglelefteq P'$.*
2. *if $P$ and $P'$ are two patterns of the form $P_1.P_2$ and $P'_1.P'_2$, with $P_1 \trianglelefteq P'_1$ and $P_2 \trianglelefteq P'_2$, then $P \trianglelefteq P'$.*

*If $P \trianglelefteq P'$, we will say that $P$ is* finer *than $P'$.*

**Example 4** *The pattern* `[Z.Y.P.U]` *is finer than* `[H.P.A]`*, because* `[Z.Y]` $\trianglelefteq$ `[H]`*,* `[P]` $\trianglelefteq$ `[P]` *and* `[U]` $\trianglelefteq$ `[A]`*.*

The following is easily verified:

**Proposition 1** *The relation $\trianglelefteq$ is a partial order on $\mathcal{P}_\Sigma$.*

Further more, it can be shown that two trajectory patterns have a unique minimal common ancestor with respect to relation $\trianglelefteq$. This ancestor represents the minimal level of description of the trajectories at which they can no longer be distinguished. Consider for instance $P_1$ =`[X.Y.P.A]` and $P_2$ =`[H.P.U.V]`. Both patterns represent a travel from département `Haute-Loire` to département `Allier` through the `Puy-de-Dôme` département. It turns out that $P_1$ features a more detailed representation for the part which belongs to `Haute-Loire`, whereas $P_2$ is more detailed for the part that belongs to `Allier`. Neither $P_1 \trianglelefteq P_2$ nor $P_2 \trianglelefteq P_1$

hold, but both are less than $P_3$ =[H.P.A] with respect to $\trianglelefteq$.

**Proposition 2** *Let $\mathcal{D} = \{P_1, .., P_n\}$ be any set of trajectory patterns. Let $\mathcal{U}$ be the set of all patterns $P$ such that $P_i \trianglelefteq P, i = 1, 2, ..., n$. Then $\mathcal{U}$ has a unique minimal element, that we shall denote as $lub(\mathcal{D}, \trianglelefteq)$.*

The existence of a least upper bound (lub) is important, both for our classification purposes and for optimization during on-line query processing. From a classification point of view, the semi-lattice structure of our patterns ensures that we can always obtain, for a set of trajectories, a scale level at which these trajectories are represented by a single pattern.

The optimization aspect is less common, although important as well. Basically, we know that if a trajectory does *not* match the lub of a set of patterns, then it will not match *any* of these patterns. This property allows to avoid many useless computations in large-scale notification systems (see for instance [25, 3]), by creating clusters of pattern-based queries and by filtering out the evaluation of these queries thanks to their lub. We shall develop this aspect in the next section.

We need an algorithm to compute the least upper bound of a set of patterns. Furthermore, if we take into account the previous motivation (using a lub as a filter over a set of patterns), we need an *incremental* algorithm because we must be able to add a new pattern in an existing cluster without having to recompute everything. The technique proposed hereafter relies on the following observation: finding the lub of a set of patterns is equivalent to finding the scale level $S$ at which all the patterns become similar. Once $S$ is found, it remains to find the (unique) ancestor of the patterns in $S$.

The LUB algorithm takes a pair of patterns $(P_1, P_2)$ as input and delivers the lub of $\{P_1, P_2\}$. It relies on the functions $lca(a, b)$ which returns the least common ancestor of the pair of symbols $(a, b) \in \Sigma^2$ with respect to $\preceq$ and $ancestor(S, a)$ which returns the ancestor of the symbol $a$ at the scale level $S$. Recall that $cover(\sigma)$ computes the minimal scale level $S$ which "covers" $\sigma \subseteq \Sigma$.

The LUB algorithm can be used to compute incrementally the lub of a set $\{P_1, .., P_n\}$, with $n > 2$: first compute $\text{LUB}(P_1, P_2)$, then $\text{LUB}(\text{LUB}(P_1, P_2), P_3)$, etc. We end this section by working out an example illustrating how this algorithm works, still referring to the reference map of Figure 2.

**Example 5** *Consider the patterns $P_1$ =[X.Y.P.A] and $P_2$ =[X.Z.P.U.V]. In order to compute $\text{LUB}(\{P_1, P_2\})$, we first initialize $S_{lub}$ with $S_{leaves}$ = {U, V, W, C, P, X, Y, Z}. Each item in the following enumeration represents a step in the* **while** *loop.*

1. *First $l$ is assigned to $lca(P_1[1], P_2[1]) = lca(X, X) = X$. We skip the part covered by X, both for $P_1$ and $P_2$: $i$ and $j$ are set to 2. $S_{lub}$ remains unchanged.*
2. *Next $l$ is assigned to $lca(P_1[2], P_2[2]) = lca(Y, Z) = H$. We skip the part covered by H: both $i$ and $j$ are set to 3. The scale level is $S_{lub} = cover(S_{lub} \cup \{H\})$ = {U, V, W, C, P, H}.*
3. *$P_1[3]$ and $P_2[3]$ are compared. Since both are P which belongs to $S_{lub}$, we just advance one symbol on each pattern: $i = j = 4$*
4. *Finally $l$ is assigned to $lca(P_1[4], P_2[4]) = lca(U, A) = A$. We skip the end of both patterns, i.e., $i = 5$ and $j = 6$. $S_{lub}$ is $cover(\{U, V, W, C, P, H\} \cup \{A\})$ = {A, C, P, H}.*

*Next the step (b) is carried out: taking $P_1$, one replaces each symbol by its ancestor in $S_{lub}$ and one obtains $P_{lub}$ = [H.H.P.A] which, once reduced, yields the final result $lub(P_1, P_2)$ = [H.P.A]. Note that one would obtain at this step the same result by taking $P_2$: $P_{lub} = reduce($[H.H.P.A.A]$) =$ [H.P.A].*

In summary, the computation of the least upper bound of a set of patterns, using the LUB algorithm, provides the finest pattern common to the trajectories of interest and, incidently, delivers the finest scale level at which the trajectories will be considered as similar.

## 4. Pattern-based queries

We now turn our attention to the application of our pattern-based framework to mobile object tracking [22, 24, 10]. Using a pattern representation for trajectories, as advocated in the previous section, leads to a new approach for querying information related to the motion of objects, based on discrete *events*. Indeed, the result set of a pattern-based query changes only when an object enters or leaves a zone, and this allows an easy maintenance of this result, compared to the complex techniques which are required when continuous motion is considered.

### 4.1. Pattern-based query language

The simplest application of our classification approach is to define a query as a *trajectory pattern*, expressed at a given scale level, and to report all the tra-

**Algorithm** LUB
**Input**: A pair of patterns $P_1, P_2$
**Output**: $lub(\{P_1, P_2\}, \trianglelefteq)$
**begin**
– Step (a): one computes the scale level $S_{lub}$ of the least upper bound –
// Initialize $S_{lub}$ to be the finest possible scale level, $S_{leaves}$
$S_{lub} := S_{leaves}$
// Scan $P_1$ and $P_2$, and determine at each step if a change of scale is required
$i := 1, j := 1$ // Current position in $P_1$ and $P_2$.
**while** ($i \leq length(P_1)$ **and** $j \leq length(P_2)$) **do**
// Take the least common ancestor of $P_1[i]$ and $P_2[j]$ in $(\Sigma, \preceq)$
$l := lca(P_1[i], P_2[j])$
// Skip the part "covered" by this lca in the two patterns
**while** $P_1[i] \preceq l$ **do** $i := i + 1$; **while** $P_2[j] \preceq l$ **do** $j := j + 1$
// Compute the minimal scale level which contains $l$
$S_{lub} := cover(S_{lub} \cup \{l\})$
**end while**
// $P_1$ or $P_2$ has not been entirely scanned? Then $S_{lub}$ is the coarser scale level, $S_{root}$
**if** ($i \leq length(P_1)$ **or** $j \leq length(P_2)$) **then** $S_{lub} := S_{root}$
– Step (b): replace each symbol in $P_1$ by its ancestor in $S_{lub}$ –
**for each** ($i \in [1, length(P_1)]$) $P_{lub}[i] := ancestor(S_{lub}, P_1[i])$
– Step (c): compute the reduction of $P_{lub}$: this is the result –
**return** $reduce(P_{lub})$
**end**

Fig. 3: The LUB algorithm

jectories that match this pattern. However, in an online classification context, we need to extend the power of a pattern-based query language by searching the occurrences of a pattern in a *stream* of locations.

This extension is motivated by practical considerations. The goal, in a traffic monitoring application for instance, is to observe a sequential stream of GPS events and to detect whether a pattern is matched by the most *recent* events which have been observed for a given object. In other words we need to match query patterns with the *suffix* of the sequence of locations obtained so far. If, for instance, a query is of the form `[P.C.Z]`, the answer will include all the objects whose (currently known) trajectory can be mapped to a word in the language $\Sigma^* PCZ$.

The query language and its semantics are defined as follows:

**Definition 8 (Pattern-based query language)** *Let $\mathcal{M}$ be a reference map $(\Sigma, \preceq)$. A query $q$ over $\mathcal{M}$ is a pair $(S, P_q)$ where $S$ is a scale level in $\mathcal{M}$ and $P_q$ is a trajectory pattern in $S$.*

The answer to a query $q(S, P_q)$ is the set of objects whose trajectory $traj$ is such that $P_q$ is a suffix of $pattern(traj, S)$.

**Definition 9 (Query answer)** *The* answer *to a query $q(S, P_q)$ over a set of objects $O$, denoted $ans_O(q)$, is the subset of $O$ defined as follows: $o \in ans_O(q)$ iff $P_q$ is a suffix of $pattern(o.traj, S)$.*

## 4.2. Query evaluation

The evaluation of queries relies on a simple application of standard pattern matching techniques. The problem is that of finding the occurrences of $P_q$ (the pattern) in a text $t \in \Sigma^*$ (the sequence of locations). We need to find all the prefix of $t$ that end with $P_q$. This can be done with an automaton $\mathcal{A}(P_q)$ which recognizes the language $\Sigma^* P_q$. The construction of $\mathcal{A}(P_q)$ is standard and can be found in any textbook devoted to pattern-matching algorithms (see, for instance, [5]). Let $h$ be the function that, given a trajectory pattern $t \in S^*$, returns the longuest suffix of $t$ which is also a prefix of $P_q$. The automata $\mathcal{A}(P_q)$ is then defined as follows:

- its states are the set of prefix of $P_q$ (including the empty pattern);
- its initial state is the empty pattern, and its final/accepting state is $P_q$.
- the transitions are of the form $s \xrightarrow{a} h(sa)$

An automaton $\mathcal{A}(P_q)$ is illustrated in Figure 4 for the query pattern $P_q =$ `[Z.X.P.W.U]`. The representation of the transition function, $\delta$, is simplified. We show

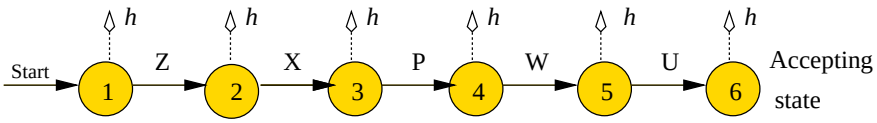


Fig. 4: An automaton for the query $P_q$ = [Z.X.P.W.U]

explicitly the transitions from one state to another for each symbol in $P_q$ (black arrows). Clearly the transitions explicitly shown on Figure 4 will lead to the accepting state if a matching trajectory is read.

The other transitions correspond to a failure to match the query pattern with a trajectory, and are summarized by the function $h$ (arrows with dotted line) on each state. The transitions denoted by $h$ reduce, for this particular automaton, to two cases:

1. on input Z, $h$ leads to state 2 because Z is in that case the longest suffix of the trajectory pattern which is also a prefix of $P_q$.
2. on any input which is *not* Z, $h$ leads to state 1, because in that case there is no suffix of the trajectory which is also a prefix of $P_q$.

In general, the function $h$ leads to the state representing the longest prefix of $P_q$ which is also a suffix of the trajectory pattern. The computation of $h$ was first described in [11] and can be found in specialized textbooks [5].

An important point is that the automaton is deterministic. This ensures that we can locate a query pattern on an input stream by just maintaining the current state in the automaton of the object that generates the stream. The query evaluation algorithm takes as input the query definition, an object $o$, its current location, and returns `true` when $o$ belongs to the answer of $q$, or `false` otherwise.

**Algorithm** EVAL
**Input**: A query $q(S, P_q)$, an object $o$, a location $loc$
**Output**: A boolean which states whether $o$ belongs or not to the query answer
**begin**
  // Get the label of *pos* in the scale level $S$
  $a = label(S, loc)$
  // Check that $o$ entered a new zone
  **if** ($a$ is distinct from the previous label) **then**
    // Apply the transition function on $a$,
    // and compute the new current state of $o$
    $o.s_{cur} = \delta_q(o.s_{cur}, a)$
  **endif**
  // Check whether $o$ belongs to the answer of $q$
  **if** ($o.s_{cur}$ is the accepting state) **return** true
  **else return** false
**end**

The algorithm EVAL must be called each time a new event (e.g., a GPS message) is received for an object, and its result can be used for the maintenance of the query result set. It is worth underlying that we do *not* need to record the history of an object $o$ during this process. Clearly, it suffices to record the current status of $o$ with respect to the automaton $\mathcal{A}(P_q)$. This minimizes memory requirements, and avoids to backtrack on a trajectory pattern during query evaluation.

### 4.3. Optimization issues

Our approach offers several opportunities for optimization. We consider in particular systems that aim at classifying in real time a set of objects whose locations are periodically updated. When the system processes a very large number of pattern queries over a very large number of objects, we can reduce the workload by the following means:

1. *event filtering*: as mentioned in the previous section, we can use the semi-lattice structure of a set of patterns to filter out unnecessary computation;
2. *resource sharing*: we can group together the automata of a set of queries.

Both techniques can be combined. The first one is quite specific to our multi-scale framework. It consists in grouping queries in *clusters* and to filter out, for each cluster, the GPS events which cannot affect the result of the cluster's queries. We know that a set of queries $\{q_1, q_2, \ldots, q_n\}$ in a given cluster can be represented by its lub $q_{lub}$, and that a trajectory cannot match a query $q_i$ if it does not match also $q_{lub}$. In some sense we can "index" the set of queries registered by the system [10, 13] and save a lot of irrelevant computations. Note that this index is explicitly based on sequences of locations, and therefore behaves quite differently than classical spatial indexes (e.g., R-trees).

Second we can, in each cluster, "merge" together the automata associated to the queries in the cluster into *one* global automaton shared by all the queries, thereby saving computational resources. The technique is well-known in the pattern-matching area ("dictionary automata"), and catches up the multiple optimization research conducted for a long time in the database community [20, 3], which is nowadays intensively investigated in the context of data streams management systems [1, 9].

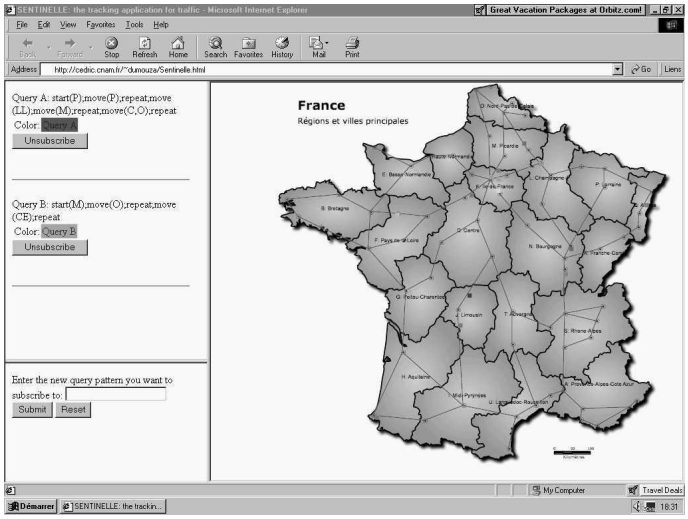



Fig. 5: The graphical interface of the system

## 5. Implementation

We are currently implementing a system which extends a first prototype presented in [6]. The goal is first to show the effective applicability of our model, second to enable a platform for testing our optimization techniques, and third to experiment visualization and animation coupled with the server's functionalities. We present here briefly both the functionalities of the system and our implementation choices.

Figure 5 shows the graphical interface of the system. It consists of a few frame boxes displayed in a web browser. The main frame displays the reference map (here the french territory with the scale level of *régions*) and any other convenient geographic layer. For this particular example, the reference map is combined with a network of roads joining the main french cities.

The lower-left frame is used to register queries to the system. We rely on a very simple user-friendly language to express query patterns, based on a few intuitive primitives: `start`, `move`, etc. A query is assigned to a set of specific graphical attributes which allow to distinguish its result set from the result sets of other queries. In the current implementation, a query result is simply associated to a specific color, but more sophisticated graphical characterization should be used if many queries are registered concurrently.

Figure 5 shows a state where two queries have been registered by the client. These queries are listed in the upper-left frame, together with a button to un-register the query and the color assigned to the query result set. The result itself is graphically represented as a set of animated objects moving over the map, each one characterized by the color of its associated query(ies).

Figure 6 sums up the whole architecture of the prototype. Clients are web browsers equipped with the *SVG browser* from Adobe [1]. A map is an SVG document, represented at the client's side as a DOM tree. The document is updated by the applet thanks to DOM functions whenever an update message is received. The SVG browser modifies then automatically the display (including animated objects) presented to the user.

The server is a java Servlet which gets the events from the simulator. The server sends initially to each new client a SVG document representing the geographic data (map and network). Then for each query a java thread is created on the server's side. The thread manages the main memory structure that maintains the query result, namely

1. the automaton associated to the query pattern;
2. a list $L$, with one entry per object recording its current state.

The states of the automaton are represented as a mask of bits. The value 1 (resp. 0) for a bit means that $o_i$ is (resp. is not) in the associated state. This gives a compact structure: for a pattern with 4 symbols, a mask of 4 bits must be recorded for each object. One can therefore track a database of one million objects with only 500K in main memory.

When it receives a GPS event related to an object $o$, the thread calls the EVAL algorithm presented in Section 4. Depending on the result, $o$ will enter or leave the result set. The server sends then a message to the browser which updates the graphical representation at the client's side.

The current implementation does not include the optimization techniques presented at the end of Section 4. Extending the system to support a large number of queries is part of future work.

1 Adobe. The SVG Viewer. http://www.adobe.com/svg

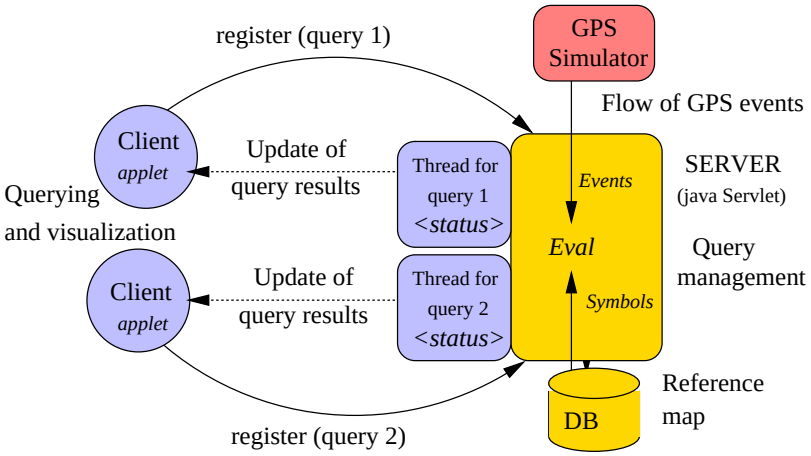



Fig. 6: The system's architecture

## 6. Related work

The work presented here can be related to several areas, including multi-scale representation in GIS applications, time series, and moving objects tracking.

While several multi-scale representation frameworks for spatial databases have been recently presented [18, 23, 27], they do not cover the trajectories of moving objects. [27] implements multi-scale data access schemes using scale-priority values for the vertices of the geometric objects. While this model supports queries on arbitrary scale values, it does not aim at representing and querying trajectories.

In time-series databases several approaches ([4, 17, 26]) for shape-based retrieval have been proposed. They define a similarity distance to compare sequences of real numbers, along with indexing techniques and efficient evaluation algorithms. In [26] the authors present a new similarity measure to compare 2D-space trajectories using the longest common subsequence (LCSS) model. The authors focus on the noise that results from incorrect data measurement and demonstrate through analysis and experimental evaluation the robustness of their technique. Although the approach of [26] relies on numerical computation and is therefore essentially different from our thematic classification of the underlying space, the potential impact of errors is indeed something to consider. We plan to exploit and adapt approximate string matching techniques [16] to our string-based pattern representation in order to cope with this challenge.

In the area of spatio-temporal databases, the problem of tracking continuously moving objects is explicitely addressed in several works [2, 10, 24]. [2] for instance describes a web-based architecture for reducing the volume and frequency of data transmissions between the client and the server. [10] presents a system that indexes queries in order to recompute periodically the whole result of each query. This is in constrast with the incremental computation advocated in the current paper.

Finally a relevant work is the aggregation and comparison approach proposed by Meratnia and de By in [15]. They present an interesting method which computes the similarity between trajectories, based on a raster representation. However they ignore multi-scale issues, and the fixed space partition can not adapt to the thematic partition of space that we consider.

## 7. Conclusion

We presented in this paper a novel approach for classifying, clustering and continuously querying moving objects. Computations are based on a thematic multi-scale partition of the underlying space and can be supported by simple and efficient tools (i.e., pattern matching on strings). This is an original work which constitutes a convenient way of managing continuous queries over a set of objects continuously moving in the 2D space. To our knowledge this approach has not been explored so far in the spatio-temporal literature, although somewhat similar ideas can be found in other domains, e.g., sequences databases [21, 19]. The technique can augment the power of practical moving objects monitoring systems [9] and more generally of manipulation languages [8] for such applications.

We believe that 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 optimization techniques for the evaluation of our pattern-based queries. The goal, in the latter case, is to meet the requirements of very large monitoring systems that handle millions of objects and queries. In such a context, an important issue is the ability to *share* the resources between

queries [1].

We believe that combining the exploitation of the refinement relation on patterns and the extension of pattern-matching techniques offers fruitful opportunities. We are currently designing a clustering algorithm that puts "close" patterns together in a same cluster, and thereby allows to characterize each cluster by a precise lub which can then be used for filtering purposes.

## References


[1] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and Issues in Data Stream Systems. In *Proc. ACM Symp. on Principles of Database Systems*, pages 1–16, 2002.

[2] T. Brinkhoff and J. Weitkämper. Continuous Queries within an Architecture for Querying XML-Represented Moving Objects. In *Proc. Intl. Conf. on Large Spatial Databases (SSD)*, 2001.

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

[4] K.K.W. Chu and M.H. Wong. Fast Time-Series Searching with Scaling and Shifting. In *Proc. ACM Symp. on Principles of Database Systems*, 1999.

[5] M. Crochemore and W. Rytter. *Text algorithms*. Oxford University Press, 1994.

[6] C. du Mouza and P. Rigaux. Web architectures for scalable moving object servers. In *Proc. Intl. Symp. on Geographic Information Systems*, pages 17–22, 2002.

[7] C. du Mouza and P. Rigaux. Multi-scale Classification of Moving Objects Trajectories. Technical Report 1371, Laboratoire de Recherche en Informatique, 2004.

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

[9] M.A. Hammad, W.G. Aref, and A.K. Elmagarmid. Stream Window Join: Tracking Moving Objects in Sensor-Network Databases. In *Proc. Intl. Conf. on Scientific and Statistical Databases (SSDBM)*, 2003.

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

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

[12] M. Koubarakis, T. Sellis, and *al.*, editors. *Spatiotemporal Databases: The* Chorochronos *Approach*. Springer Verlag, 2003.

[13] L.V.S. Lakshmanan and P. Sailaja. On Efficient Matching of Streaming XML Documents and Queries. In *Proc. Intl. Conf. on Extending Data Base Technology*, 2002.

[14] R. Laurini and D. Thompson. *Fundamentals of Spatial Information Systems*. Number 37 in The A.P.I.C. Series. Academic Press, New York, 1992.

[15] N. Meratnia and R.A. de By. Aggregation and Comparison of Trajectories. In *Proc. Intl. Symp. on Geographic Information Systems*, 2002.

[16] G. Navarro. A Guided Tour to Approximate String Matching. *ACM Computing Surveys*, 33(1):31–88, 1997.

[17] D. Rafiei and A. Mendelzon. Querying Time Series Data Based on Similarity. *IEEE Transactions on Knowledge and Data Engineering*, 12(5), 2000.

[18] P. Rigaux and M. Scholl. Multi-scale partitions: Application to Spatial and Statistical Databases. In *Proc. Intl. Conf. on Large Spatial Databases (SSD)*, 1995.

[19] Reza Sadri, Carlo Zaniolo, Amir M. Zarkesh, and Jafar Adibi. A Sequential Pattern Query Language for Supporting Instant Data Mining for e-Services. In *Proc. Intl. Conf. on Very Large Data Bases (VLDB)*, 2001.

[20] T. Sellis. Multiple-Query Optimization. *ACM Trans. on Database Systems*, 13(1):23–52, 1988.

[21] Praveen Seshadri, Miron Livny, and Raghu Ramakrishnan. SEQ: A Model for Sequence Databases. In *Proc. IEEE Intl. Conf. on Data Engineering (ICDE)*, pages 232–239, 1995.

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

[23] J. G. Stell and M. F. Worboys. Generalizing Graphs Using Amalgamation and Selection. In *Proc. Intl. Conf. on Large Spatial Databases (SSD)*, volume 1651 of *LNCS*. Springer, 1999.

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

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

[26] M. Vlachos, G. Kollios, and D. Gunopulos. Discovering Similar Multidimensional Trajectories. In *Proc. IEEE Intl. Conf. on Data Engineering (ICDE)*, 2002.

[27] S. Zhou and C.B. Jones. Design and Implementation of Multi-Scale Databases. In *Proc. Intl. Symp. on Spatial and Temporal Databases (SSTD)*, 2001.