*Instituto de Informatica,*
Universidade Federal do Rio Grande do Sul, Porto Alegre, Brazil

Technical Report TR-357. Pages 22, 2008

# A Spatio-temporal Data Mining Query Language for Moving Object Trajectories

Vania Bogorny[1] , Bart Kuijpers[2] , Luis Otavio Alvares[1]

[1]Instituto de Informatica, UFRGS, Brazil
[2] Hasselt University, Belgium

Mobile devices are becoming very popular in the recent years, and large amounts of trajectory data are generated by these devices. Trajectories left behind cars, humans, birds or other objects are a new kind of data which can be very useful in decision making process in several application domains. These data, however, are normally available as sample points, and therefore have very little or no semantics. The analysis and knowledge extraction from trajectory sample points is very difficult from the user's point of view, and there is an emerging need for new data models, manipulation techniques, and tools to extract meaningful patterns from these data. In this paper we propose a new methodology for knowledge discovery from trajectories. We propose through a semantic trajectory data mining query language several functionalities to select, preprocess, and transform trajectory sample points into semantic trajectories, at higher abstraction levels, in order to allow the user to extract meaningful, understandable, and useful patterns from trajectories. We claim that meaningful patterns can only be extracted from trajectories if the background geographic information is considered, and therefore we build the proposed methodology considering both moving object data and geographic information. The proposed language has been implemented in a toolkit in order to provide a first software prototype for trajectory knowledge discovery.

*Keywords:* Moving Objects, Semantic Data Mining, Spatio-temporal Data Mining, Spatio-temporal Data Mining Query Language, Trajectory Query, Trajectory Knowledge Discovery

## 1 Introduction

Moving object data have become very common in the last few years. Huge amounts of raw trajectory data are available from many sources, and there is an increasing necessity of new methods and tools to extract knowledge from these data in different application domains (Brakatsoulas *et al.* 2004).

Trajectory data are normally available as sample points (x,y,t), which have very little or no semantics. The electronic devices that capture these data do not collect the background geographic information which is relevant for the application. This makes the analysis and knowledge extraction from trajectory sample points very complex, since the integration with semantic geographical information is an essential step towards trajectory knowledge discovery (Alvares *et al.* 2007a) (Alvares *et al.* 2007b) (Alvares *et al.* 2007c).

Recently, several algorithms have been developed for mining trajectory sample points (Tsoukatos and Gunopulos 2001, Laube *et al.* 2005, Verhein and Chawla 2006, Cao *et al.* 2006, Gudmundsson and van Kreveld 2006, Giannotti *et al.* 2007, Lee *et al.* 2007), in particular for discovering similar trajectories or dense regions. These approaches, however, suffer from some general problems that are essentially important for trajectory knowledge discovery: (i) focus on the mining step itself, basically considering the geometric properties of trajectory sample points, without taking into account the semantics of the data; (ii) do not cover the whole trajectory knowledge discovery process, which requires complex data preprocessing and post-processing tasks in order to generate meaningful patterns understandable by humans; (iii) do not consider the geography behind trajectories, which is the basic information to understand patterns in most application domains; and (iv) do not provide preprocessing/transformation mechanisms to manipulate the data at different granularities (e.g. morning/afternoon, rush hours, weekday/weekend), which can be of fundamental importance in the knowledge discovery process (Han 1995).

In the following sections we show the complexity of querying and mining trajectory sample points and how these process can be simplified considering semantics of trajectories.

## 1.1 *Querying Trajectories*

Trajectory queries can be tremendously simplified and optimized in both computational and formulation complexity by integrating trajectories and semantic geographic information a priori, in one single preprocessing step (Alvares *et al.* 2007b). To understand such simplification, let us consider a query example.

Q1: Which are the places that a moving object $A$ has passed during his trajectory, assuming that each object has only one trajectory in the database, given *hotels* and *touristic places* as the relevant geographic places?

Let us consider that the moving object $A$ has the trajectory (1) shown in Figure 1, that has no semantic geographic information, and is represented as a set of $(x,y,t)$. In this scenario, a query to answer question Q1 will contain two spatial joins, and is similar to the following:

```
SELECT h.name
  FROM trajectory t, hotel h
 WHERE t.id='A' AND
       intersects(t.movingpoint.geometry,h.geometry)
UNION
SELECT p.name
  FROM trajectory t, touristicPlace p
 WHERE t.id='A' AND
       intersects(t.movingpoint.geometry,p.geometry)
```

This query is very simple, and considering a trajectory as a set of sample points, the *intersection* operation will be tested until the first point of the trajectory intersects the given geographic places hotel and touristic place. The query can become more complex when a time constraint is specified, like *which are the places that an object* A *has passed and has stayed for more than 3 hours*. In this example both formulation and time complexity of the query increases, because the search cannot stop when the first point of the trajectory intersects a given place. All points in the trajectory that intersect the place have to be tested until either (a) the moving object leaves the place or (b) until the time constraint is satisfied. In this query the search space increases in relation to the previous one.

Now let us consider that object $A$ has the *semantic trajectory* (2), shown in Figure 1, where the aggregation of semantic geographic information with the trajectory has been performed in a preprocessing step, where the user has defined all geographic places that are relevant to the application. In this scenario, the answer to question Q1 will be something like:

```
SELECT place
  FROM semanticTrajectory
 WHERE id='A'
```

A *semantic trajectory*, as shown in Figure 1 (2), aggregates the geographic information that is necessary for the analysis of the trajectory, and this information can be *reused* in any query. For trajectory samples without semantic annotations, as the example shown in Figure 1(1), the relationships/intersection of the trajectory with the relevant geographic object types have to be *recomputed* for each different query, either for the same or different geographic object types.

Considering trajectories with semantic annotations, not only the querying process is simplified, but the data mining and knowledge discovery process is significantly facilitated from the user's perspective, as will be explained in the following section.

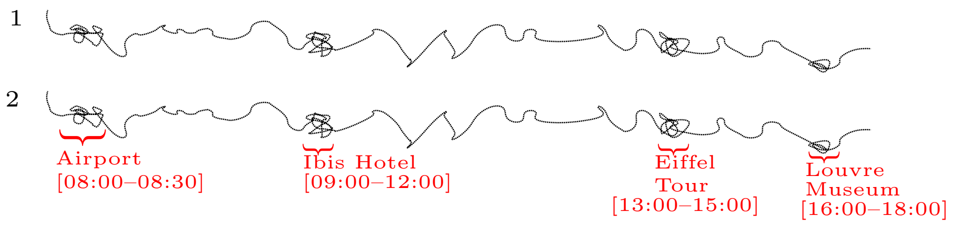

Figure 1. single raw trajectory and single semantic trajectory

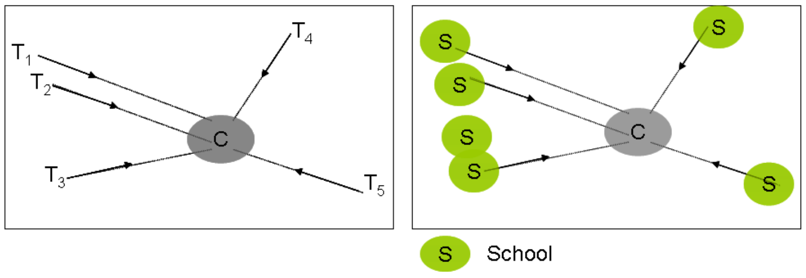

Figure 2. (left) Convergence geometric pattern and (right) semantic trajectories

## 1.2 *Mining Trajectories*

For data mining and knowledge discovery the integration of geographic information to trajectories a priori plays an essential role towards the discovery of semantic trajectory patterns. Because knowledge discovery is an interactive and repetitive process (Fayyad *et al.* 1996), the a priori integration of trajectories and semantic information tremendously simplifies the data mining and post-processing steps.

Existing spatio-temporal data mining algorithms have focused on the geometrical properties of trajectories, and therefore tend to discover geometric patterns. *Geometric patterns* are normally extracted based on the concept of dense regions or trajectory similarity. *Semantic patterns*, however, are independently of $x, y$ coordinates, and can be located in sparse regions and have no similarity. Figure 2 (left) shows an example of a geometric trajectory pattern, called *convergence pattern* (Laube *et al.* 2005), in which the objects move to the same place, at a certain time, which therefore becomes the dense region $C$. In Figure 2 (right), the moving objects go from different Schools (S) to meet at the same place $C$ at a certain time. Let us consider $C$ as a semantically important place, like for instance, a cinema with special price tickets for students on Wednesday afternoon. From these semantic trajectories, the moving pattern *from Schools to Cinema* could be discovered. In this example it is clear that the origin of the trajectories is in *sparse locations*, and a density-based trajectory data mining algorithm would not be able to discover such semantic pattern.

Figure 3 shows another example of a geometrical pattern in which the objects cross the same spatial location $B$. Considering the semantics of the trajectories, such as in Figure 3 (right), two semantic patterns could be discovered: (i) a move from Hotel (H) to Restaurant (R) passing by region $B$; and (ii) a move to Cinema (C), passing by region $B$.

Considering the examples of geometrical and semantic patterns we can clearly notice the important role that semantics plays in the trajectory knowledge discovery process. Even more important is to observe that the a priori integration of trajectories with semantic information is vital for the discovery of semantic patterns. Semantic patterns cannot be obtained in post-processing steps, after the pattens have been generated, because information is lost if the mining algorithm relies only on the geometrical properties of trajectories.

Trajectory data mining algorithms which are based on density or trajectory similarity, in the above examples would discover that several trajectories converge to region $C$ or cross region $B$. From the semantical point of view, these patterns would only be interesting if $C$ and $B$ are important places which are relevant for the application. If semantically $C$ and $B$ are not important for the application, then these patterns will be useless and uninteresting for the user.

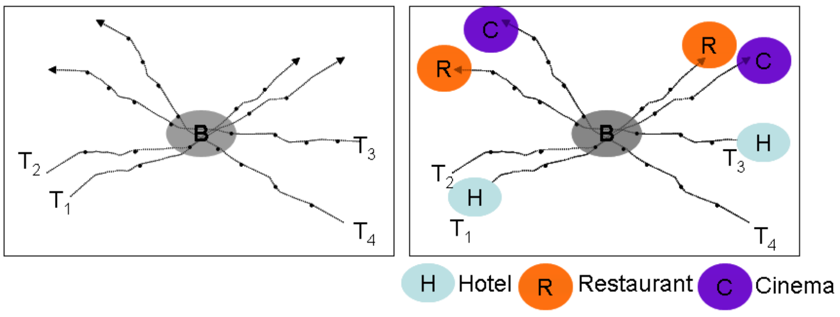

Figure 3. Recurrence geometric pattern X semantic trajectories

So far we have seen the important role of semantics for both querying and mining trajectory data. Complex data preprocessing and data transformation functions are necessary to add semantics to trajectories in order to facilitate their analysis and knowledge extraction from the user's point of view.

Another problem in trajectory knowledge discovery is that as far as we know there are no available toolkits with friendly and graphical user interfaces to help the user to perform the whole knowledge discovery process (Fayyad *et al.* 1996).

Concerning the needs of the user for knowledge discovery from trajectories in real applications, we claim that a new methodology has to be developed to extract meaningful, useful, and understandable patterns from trajectories. Therefore, in this paper we propose a data mining query language for moving object data, focusing on the semantic properties of trajectories. Through a semantic trajectory data mining query language (STDMQL) we introduce a new methodology to cover the whole cycle of the trajectory knowledge discovery process.

### 1.3 *Scope and Outline*

In this paper we propose a STDMQL that is an extension of spatial SQL (Egenhofer 1994). In a data preprocessing step we integrate trajectories with the geographic information that is relevant to the application. For this purpose, we adopt the trajectory data model proposed by (Spaccapietra *et al.* 2007), in which the user has the vision of trajectories as a set of stops and moves.

It is important to emphasize that the novelty and the main objective of this paper relies on the methodology for trajectory knowledge discovery that is defined in terms of a data mining query language. The development of new trajectory data mining algorithms is out of the scope of this work.

This paper is organized as follows: in Section 2 we present the related works. In Section 3 we introduce the basic concepts to enrich trajectories with semantic geographical information. The overall methodology, the primitives, and the prototype of the proposed trajectory data mining query language are presented in Section 4. In Section 5 we conclude the paper and present some directions of future work.

## 2 Related Works

Our work is motivated by (Boulicaut and Masson 2005) and (Han *et al.* 1996), which claim that a data mining query language should provide enough mechanisms to perform the whole knowledge discovery process, specially:

- mechanisms for data selection and data preprocessing where the data miner should be able to specify the portion of data he is interested in. This specification may require a subtask to first retrieve a subset of the data before the mining.
- allow the user to specify the patterns to be mined, like association rules, frequent patterns, sequential patterns, clusters, etc.
- integrate background knowledge, like concept hierarchies, for extracting knowledge at several granularity levels.

- define the constraints of the desired patterns, where the user can define thresholds to extract more interesting patterns or to filter out less interesting ones.
- provide mechanisms for postprocessing the discovered patterns.

According to Boulicaut (Boulicaut and Masson 2005), the first major limitation of existing data mining query languages is the poor support to data preprocessing, because they are essentially designed around the mining step itself, and mainly provide primitives for rule extraction. It has been shown that data preprocessing for knowledge discovery are tedious phases for which the use of integrated tools and operators is needed, and in general existing data mining query languages do not reduce this problem (Boulicaut and Masson 2005). This is specially true for trajectories, which are in general more complex than conventional and spatial data, and require mechanisms to be semantically enriched in order to lead to the discovery of understandable and useful patterns.

Several data mining query languages have been proposed in the last decade for traditional data. Meo has proposed MINE RULE, which basically consists of an extension of SQL for mining association rules (Meo *et al.* 1996). Another approach has been proposed by (Han *et al.* 1996) for mining *characteristic, discriminant, classification*, and *association rules*, named DMQL. This language allows the extraction of rules at different granularity levels, using concept hierarchies (Han 1995) as background knowledge. In (Imieliński and Virmani 1999) a data mining language named MSQL has been proposed for mining association rules, with the main objective to query and filter association patterns after the rule generation. This work points out the importance of storying the patterns for later query and filter these patterns in post-processing. This idea of pattern storage has been recently adopted in several works like (Calders *et al.* 2006a), (Calders *et al.* 2006b), and is also supported by our language for mining and querying both trajectory data and trajectory patterns.

*Atlas* (Wang and Zaniolo 2003) and *LDL* (Arni *et al.* 2003) are languages that provide the ability of user defined aggregates (UDAs). Atlas was build on the principles of relational databases and is basically a programing language based on SQL, which therefore becomes very complex from the user's point of view.

A few data mining query languages have been proposed for spatial data, like GMQL (Han *et al.* 1997) and SDMOQL (Malerba *et al.* 2004). GMQL is an extension of DMQL to support spatial data mining, while SDMOQL is an object data mining query language. Both approaches make use of concept hierarchies for mining geographic data at different granularity levels. In SDMOQL the user needs complex knowledge about the environment in order to be able to use the language. Complex data preprocessing tasks are necessary to transform a spatial database into an inductive database, for then formulate the queries, which is quite complex in real applications.

A temporal data mining query language has been proposed by (Chen and Petrounias 1998), which is an SQL-like approach for mining temporal patterns. Although existing languages may deal with traditional, temporal, or spatial data, to the best of our knowledge no data mining query language has been proposed so far for trajectories, supporting both space and time. Because in general existing data mining query languages do not cover the whole knowledge discovery process and have been designed around the mining task only (Boulicaut and Masson 2005), it is difficult to extend such languages for trajectories. This new domain requires several data preprocessing, data transformation, and data discretization operations that cannot be easily plugged into existing languages. Therefore, we propose an extension of spatial SQL (Egenhofer 1994) without changing the semantics of SQL, in order to be very user friendly and facilitate its usage.

Another problem which is well known is that data mining techniques like frequent patterns and association rules tend to generate hundreds or thousands of patterns that the user has to analyze in order to find novel and useful knowledge (Bogorny *et al.* 2007). Therefore, we store the discovered patterns to apply constraints (Bonchi and Lucchese 2007). Besides, by storing the discovered patterns in the database we allow the user not only to constraint the patterns in post-processing, but also to join data with patterns for further analysis.

## 3 The Semantic Trajectory Data Model

In order to provide data preprocessing functions and the extraction of more meaningful patterns, the first step toward a STDMQL is to aggregate semantic geographic information to trajectories, according to the needs of the application. We aggregate geographic information using the concept of stops and moves introduced by (Spaccapietra *et al.* 2007), and that has been adopted in several works like (Alvares *et al.* 2007b) (Alvares *et al.* 2007a) (Alvares *et al.* 2007c). *Stops* are important places of the trajectory from the application point of view, where the moving object has stayed for a minimal amount of time. *Moves* are subtrajectories between two consecutive stops.

The aggregation of trajectory sample points into stops and moves allows more sophisticated analysis over trajectories, and without sacrificing the user with this task. This aggregation is performed a priori and only once, and therefore can be reused as many times as necessary in the discovery process. As a consequence, the output of the data mining step will be semantic patterns, which will facilitate their interpretation by the user in the post-processing phase.

To better understand what stops and moves are, in this section we present one formal model where geographic object types are defined a priori by the user as the important places of the trajectory. This model has been introduced in (Alvares *et al.* 2007b) for querying trajectories, but it is not the only way to define stops and moves, because stops are application dependent.

### 3.1 *Trajectory Samples and Candidate Stops*

Trajectory data are normally available as sample points.

**Definition 3.1** A *sample trajectory* is a list of space-time points $\langle(x_0, y_0, t_0), (x_1, y_1, t_1), \ldots, (x_N, y_N, t_N)\rangle$, where $x_i, y_i, t_i \in \mathbf{R}$ for $i = 0, \ldots, N$ and $t_0 < t_1 < \cdots < t_N$.

To transform trajectory sample points in *stops* and *moves*, it is necessary to provide the important places of the trajectory which are relevant for the application. The important places correspond to different spatial feature types (OGC 1999), specified in a geographic database. For each relevant spatial feature type that is important for the application, a minimal amount of time is specified by the user, such that a trajectory should continuously intersect this feature in order to be considered a stop. This pair is called candidate stop.

**Definition 3.2** A *candidate stop* $C$ is a tuple $(R_C, \Delta_C)$, where $R_C$ is a (topologically closed) polygon in $\mathbf{R}^2$ and $\Delta_C$ is a strictly positive real number. The set $R_C$ is called the *geometry* of the candidate stop and $\Delta_C$ is called its *minimum time duration.*

An *application* $\mathcal{A}$ is a finite set $\{C_1 = (R_{C_1}, \Delta_{C_1}), \ldots, C_N = (R_{C_N}, \Delta_{C_N})\}$ of candidate stops with mutually non-overlapping geometries $R_{C_1}, \ldots, R_{C_N}$.

In case that a candidate stop is a point or a polyline, a polygonal buffer is generated around this object, and thus represent it as a polygon in the application, in order to overcome spatial uncertainty.

### 3.2 *Stops and Moves*

**Definition 3.3** Let $T$ be a trajectory and let

$$\mathcal{A} = (\{C_1 = (R_{C_1}, \Delta_{C_1}), \ldots, C_N = (R_{C_N}, \Delta_{C_N})\})$$

be an application.

Suppose we have a subtrajectory $\langle(x_i, y_i, t_i), (x_{i+1}, y_{i+1}, t_{i+1}), \ldots, (x_{i+\ell}, y_{i+\ell}, t_{i+\ell})\rangle$ of $T$, where there is a $(R_{C_k}, \Delta_{C_k})$ in $\mathcal{A}$ such that $\forall j \in [i, i+\ell] : (x_j, y_j) \in R_{C_k}$ and $|t_{i+\ell} - t_i| \geq \Delta_{C_k}$, and this subtrajectory is maximal (with respect to these two conditions), then we define the tuple $(R_{C_k}, t_i, t_{i+\ell})$ as a *stop of* $T$ *with respect to* $\mathcal{A}$.

A *move of* $T$ *with respect to* $\mathcal{A}$ is one of the following cases:

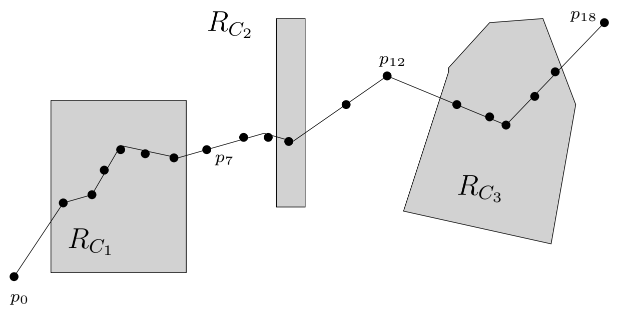

Figure 4. Example of application with three candidate stops

- a maximal contiguous subtrajectory of $T$ in between two temporally consecutive stops of $T$;
- a maximal contiguous subtrajectory of $T$ in between the initial point of $T$ and the first stop of $T$;
- a maximal contiguous subtrajectory of $T$ in between the last stop of $T$ and the last point of $T$;
- the trajectory $T$ itself, if $T$ has no stops.

When a move starts in a stop, it starts in the last point of the subtrajectory that intersects the stop. Analogously, if a move ends in a stop, it ends in the first point of the subtrajectory that intersects the stop.

Figure 4 illustrates these concepts. In this example, there are three candidate stops with geometries $R_{C_1}, R_{C_2}$, and $R_{C_3}$. Let us imagine that the spatial projection of the trajectory $T$ is run through from left to right and $t_0, \ldots, t_{18}$ are the time points of $T$. First, $T$ is outside any candidate stop, so we start with a move. Then $T$ enters $R_{C_1}$ at time point $t_1$. Since the duration of staying inside $R_{C_1}$ is long enough, $(R_{C_1}, t_1, t_6)$ is the first stop of $T$. Next, $T$ enters $R_{C_2}$, but for a time interval shorter than $\Delta_{C_2}$, so this is not a stop. We therefore have a move until $T$ enters $R_{C_3}$, which fulfills the requests to be a stop, and so $(R_{C_3}, t_{13}, t_{17})$ is the second stop of $T$. The trajectory $T$ ends with a move.

It is important to notice that the place where a stop occurs is a spatial feature (or relevant geographic object type) which is intersected by a trajectory for the minimal amount of time. This spatial feature will enrich the trajectory with its spatial and non-spatial information. For instance, if a hotel is a stop, its geometry and the non-spatial attributes (e.g. name, stars, price) is information that can be further used for both querying and mining trajectories.

Based on the concept of stops and moves, in the following section we present the STDMQL.

## 4 A Spatio-Temporal Data Mining Query Language

In Figure 5 we present a general overview of the framework for the STDMQL. In this framework the user can do both querying and mining. The STDMQL will provide preprocessing operators to easily integrate trajectories with geographic information in order to generate semantic trajectories. These data can then be mined and the extracted patterns will be stored in order to be queried and filtered in post-processing steps.

To provide support for the whole knowledge discovery process we propose a semantic trajectory data mining query language with support to three main functionalities: data preprocessing, data mining, and pattern storage for post-processing. The data preprocessing functions integrate trajectory samples with the relevant geographic information, in order to generate trajectories in a higher abstraction level. These functions generate stops and moves, which are stored as two relations in the database, as described in Section 4.1. Additionally, discretization functions are presented to transform the granularity of the space and time dimensions.

The data mining tasks supported by the STDMQL are frequent trajectory patterns, association rules, and sequential patterns. These tasks are explained in Section 4.2.

The patterns generated by the data mining operations are automatically stored in the database in a pattern relation, in order to allow both querying and filtering the discovered patterns in post-processing

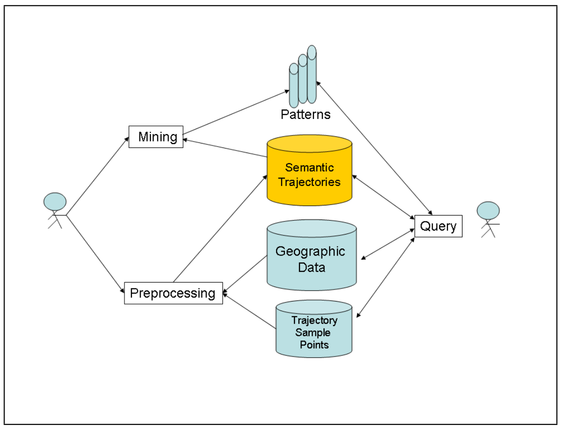

Figure 5. STDMQL framework

steps. These facility is explained in Section 4.3.

## 4.1 *Preprocessing and Data Transformation Functions*

Several data preprocessing and data transformation functions are common to the different data mining tasks supported by the STDMQL.

**4.1.1** ***Generating Stops and Moves.*** The first general function to obtain semantic trajectories is called *generateSM*. Through this function the user can select a subset of data he is interested in, and provide the geographic information that is relevant for the discovery process.

The operation *generateSM* has as input the following parameters: the *method* to compute stops and moves, the *candidate stops*, and the *buffer* area around candidate stops represented by points or lines. The output of this function is a relation of *stops* and a relation of *moves*.

The general query structure to extract stops and moves from trajectory samples is given as follows:

```
SELECT generateSM (method, candidateStops, buffer)
  FROM trajectory
```

The parameter *method* specifies how stops and moves are computed. So far, two methods have been developed: SMoT (Alvares *et al.* 2007b) and CB-SMoT (Palma *et al.* 2008). The former is an intersection-based method, and the later is a clustering approach based on speed, as will be explained further in this section.

The *candidateStops* are the set of relevant spatial feature types with respective minimal stop duration. Different relevant feature types may be a stop with large differences on the minimal stop duration. For instance, in a specific application a gas station can be considered as a stop if the object has stayed for 5 minutes, and a shopping center may be a stop if the object has stayed for at least 30 minutes. Therefore, a minimal stop duration has to be provided for each candidate stop. The parameters of *candidateStops* are:

- *relevantFeatureType*: each relevant spatial feature type is a place of interest for the application. It is a geographic object type that has spatial and non-spatial properties. By adding a relevant spatial feature as a stop in a trajectory, all semantic properties of this object will characterize one part of this trajectory.
- *minimalStopDuration*: is the minimal time interval that a trajectory must intersect the geometry of a relevant feature for this feature be considered a stop. This value is an integer and is defined in the same time unit as the time dimension in the trajectory dataset.

The parameter *buffer* is the radius (size) of the zone around relevant features represented by points or lines, to overcome problems of geometrical accuracy.

We define the syntax of the function to extract stops and moves from trajectory sample points using BNF. In general words, according to the method specified to compute stops and moves, different parameters are defined. For the method *SMoT* the user provides all candidate stops and the buffer area. For the *CB-SMoT* method the user provides the same parameters and one extra value, the *Eps* parameter (Palma *et al.* 2008), which is necessary to find clusters in single trajectories.

```
      <generateSM> ::= generateSM <left_par> <GS_parameters> <right_par>

          <left_par> ::= (
     <GS_parameters> ::= <SMoT_parameters> | <CBSMoT_parameters>
         <right_par> ::= )

   <SMoT_parameters> ::= <SMoT> <coma> <left_bracket> <candidate_stops_set>
                         <right_bracket> <coma> <buffer>
              <SMoT> ::= I-SMOT
              <coma> ::= ,
      <left_bracket> ::= [
<candidate_stops_set> ::= <candidate_stop> [{candidate_stop}...]
    <candidate_stop> ::= <relevant_feature_type> <coma> <min_stop_duration>
<relevant_feature_type> ::= string
 <min_stop_duration> ::= integer
     <right_bracket> ::= ]
            <buffer> ::= real

  CBSMoT_parameters> ::= <CBSMoT> <coma> <left_bracket> <candidate_stops_set>
                         <right_bracket> <coma> <buffer><coma><Eps_parameter>
            <CBSMoT> ::= CB-SMOT
     <Eps_parameter> ::= real
```

To better understand the syntax of the function to extract stops and moves let us consider a few query examples:

```
SELECT generateSM (I-SMOT,[Hotel,60,TouristicPlace,15,ShoppingCenter,30], 5)
  FROM trajectory
```

In this query the candidate stops are the feature types Hotel, TouristicPlace, and ShoppingCenter, and the respective minimum stop duration is 60, 15, and 30. The last parameter indicates that 5 meters will be the radius of the buffer around point and line candidate stops. In this query the method *I-SMOT* is used to generate stops and moves.

Any constraint to filter the trajectory dataset can be specified on the *where* clause. This clause is used to select a subset of trajectories to be considered in the mining process. For example, suppose that the user is interested only in trajectories that intersect the Bela Vista district. Such query can be expressed as follows:

```
SELECT generateSM (CB-SMOT,[Hotel,60,TouristicPlace,15,ShoppingCenter,30], 5, 20)
  FROM trajectory t, district d
 WHERE d.name='Bela Vista' and intersects(t.movingpoint.geometry, d.geometry)
```

The output of the query is a relation of *stops* and a relation of *moves*. The schema of the stop table has the following attributes:

```
STOP (Tid integer, Sid integer, SFTname varchar, SFid integer,
      startT timestamp, endT timestamp)
```

where:

- *Tid*: is the trajectory identifier.
- Sid: is the stop identifier. It is an integer value starting from 1, in the same order as the stops occur in the trajectory. This attribute represents the sequence as stops occur in the trajectory.
- *SFTname*: is the name of the relevant spatial feature type (geographic database relation) where the moving object has stayed for the minimal amount of time.
- *SFid*: is the identification of the instance (e.g. gid - Ibis) of the spatial feature type (e.g. Hotel) in which

the moving object has stopped.
- *startT*: is the time in which the stop has started, i.e., the time that the object enters in a stop.
- *endT*: is the time in which the moving object leaves the stop.

In a relational model the attributes *SFTname* and *SFid* is a foreign key to a geographic relation. Therefore, the stop relation significantly facilitates querying trajectories from a semantic point of view. Queries can be performed considering both spatial and non-spatial attributes of any spatial object that represents a stop, as the example shown bellow.

```
SELECT distinct s.Tid
  FROM stops s, hotel h, trainStation r
 WHERE s.SFTname='hotel' and s.SFid=h.gid and h.stars=2 and
       distance(h.the_geom,r.the_geom) < 200
```

This query retrieves all trajectories that have a stop at two-stars hotels which are less than 200 meters from a train station.

A move in a trajectory is characterized by the stops where it respectively starts and finishes, the start and end time, and the moving points that correspond to the move. The relation of moves has the following schema:

```
MOVE (Tid integer, Mid integer, SFT1name varchar, SF1id integer, SFT2name varchar,
      SF2id integer, startT timestamp, endT timestamp, the_move movingpoint )
```

where:

- *Tid*: is the trajectory identifier.
- *Mid*: is the identifier of the move in the trajectory. It starts with 1, in the same order as the moves occur in the trajectory.
- *SFT1name* : is the name of the spatial feature type in which the move starts.
- *SF1id*: is the identifier (feature instance) where the move starts.
- *SFT2name* : is the name of the spatial feature type in which the move finishes.
- *SF2id*: is the identifier (feature instance) of the of the stop where the move ends.
- *startT*: is the timestamp of the begining of the move.
- *endT*: is the timestamp of the end of the move.
- *the_move*: is the subtrajectory between the start and end of the move.

The attributes *SFT1name* and *SF1id* correspond to the stop in which a move starts and the attributes *SFT2name* and *SF2id* represent the stop where the move finishes. We adopted this structure instead of the sequential identifier of the stop (Sid) in order to allow the mining task directly over the move relation. Besides, following this representation the moves can be directly queried without the necessity of a join operation with the stops every time the user executes a query over the moves. Additionally, the user can easily perform either non-spatial, spatial, or spatio-temporal queries because we keep the spatio-temporal properties (the_move) of the move.

**4.1.2** ***The Methods SMoT and CB-SMoT.*** Two algorithms to generate stops and moves have been developed so far. The first one, called SMoT, is described in (Alvares *et al.* 2007b). It considers the intersection of a trajectory with the user-specified relevant feature types for a minimal time duration (candidate stops).

The second algorithm, called CB-SMoT and described in (Palma *et al.* 2008), is a clustering method based on the variation of the speed of the trajectory. The intuition of this method is that the parts of a trajectory in which the speed is lower than in other parts of the same trajectory, correspond to interesting places. CB-SMoT is a two-step algorithm. In the first step the slower parts of a trajectory that form clusters are identified using a variation of the DBSCAN (Ester *et al.* 1996) algorithm that considers one dimensional line (trajectories) and speed. In the second step the algorithm identifies where these potential stops (clusters) found in the previous step are located, considering the candidate stops. In case that a potential stop does not intersect any of the existing geographic objects, it still can be an interesting place. Then, in order to provide this information to the user, the algorithm labels such places as *unknown stops.*

Unknown stops are interesting because although they may not intersect any relevant spatial feature type given by the user, a pattern can be generated for unknown stops if several trajectories stay for a minimal amount of time at the same unknown stop. In this case the user may need further investigation about such pattern.

The two methods cover a relevant set of applications. SMoT is interesting in applications where the speed is not important, like tourism and urban planning. For traffic management, for instance, CB-SMoT, which is based on speed, would be more appropriate.

Stops are always extracted at the lowest granularity level (feature instance), by both methods. The stop duration is recorded when the moving object enters and leaves the stop. If a trajectory stops at the Eiffel tower, for instance, this will be the name of the stop. Starting from this low granularity level, the mining task can then transform the stop into a higher granularity, like touristic place. Analogously for the time dimension.

The granularity transformation of both *stop* (place) and *time* is specified by the operations *timeG* and *stopG*, explained in the following sections.

**4.1.3** ***Time Granularity.*** The time granularity is defined by the function *timeG*. This function generalizes a time stamp into different granularities, according to the user's interest. More specifically, it converts a time stamp into a semantic discretized *label*. For example, an interval between [07:00-09:00] can be labeled as "rushHour".

The function *timeG* is very powerful because it allows the user to discretize the time dimension in several granularities, and this is necessary in any trajectory data mining task.

The syntax of the function *timeG* is given as follows:

```
          <timeG> ::= <pre_defined_TG> | <user_defined_TG>
 <pre_defined_TG> ::= WEEKEND-WEEKDAY | YEAR | MONTH | SEASON | DAY-OF-THE-WEEK
<user-defined_TG> ::= <left-bracket><startTime><hifen><endTime> [AS <label>]
                      [{<startTime><hifen><endTime> [AS <label>]}...]
                      [OTHER]<right-bracket>
      <startTime> ::= <Time>
        <endTime> ::= <Time>
           <Time> ::= <hour> :<minute> [ :<second>]
           <hour> ::= 01 | 02 | 03 | ... | 23 | 24
         <minute> ::= 01 | 02 | 03 | ... | 59 | 60
         <second> ::= 01 | 02 | 03 | ... | 59 | 60
          <label> ::= string
          <hifen> ::= -
```

We provide some pre-defined time granularities in our language, which are: WEEKEND-WEEKDAY, YEAR, MONTH, SEASON, and DAY-OF-THE-WEEK. For these granularities the user can use the parameter *pre_defined_TG* for the function *timeG*. This will automatically transform the time dimension of either stops or moves in the specified granularity.

If the user is interested in specific time intervals, then he can decide for the parameter *user_defined_TG*, where he can specify a given time interval and choose the label, like for instance, *14:00-18:00* as *afternoon*.

**4.1.4** ***Stop Granularity.*** It is well known that for data mining and knowledge discovery the user may need to perform the discovery process several times, and therefore to transform the data at different granularity levels (Han 1995). The granularity of a stop is managed by the function *stopG*. Two granularity levels can be automatically generated, without the need of any specification of background knowledge like concept hierarchies or ontologies. These granularities are called *feature instance* (e.g. Centrum Hotel) and *feature type* (e.g. Hotel).

It is important to notice that the granularity of the stops also defines the granularity of the moves, because a move is formed by two consecutive stops (e.g. hotel-touristicPlace, IbisHotel-LouvreMuseum).

The function *stopG* is even more powerful than *timeG*, because it allows the user to specify stops at different granularities in the same mining process. The user can, for instance, set as default the granularity

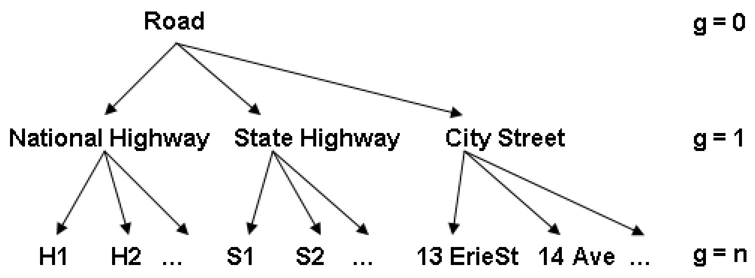

Figure 6. Example of concept hierarchy

*type* or *instance*, and specify intermediate granularity levels defined in a concept hierarchy for some specific feature types. To better understand the *stopG* function let us observe the BNF syntax specified for *stopG*.

```
          <stopG> ::= <single_level> | <multiple_level>
   <single_level> ::= TYPE | INSTANCE
 <multiple_level> ::= <left_bracket> <default_level> <coma> <specific_level>
                      [{<coma> <specific_level>}... ]<right_bracket>
  <default_level> ::= TYPE | INSTANCE
 <specific_level> ::= <relevant_feature_type_hierarchy> <equal> <hierarchy_level>
<hierarchy_level> ::= integer
```

In the syntax the granularity of a stop may be at *single level* or *multiple level*. In the former case, the granularity of the stop can be either *type* or *instance*. In the case of *multiple level* the user can define a default granularity for most stops, and for some exceptions specify the granularity level using a concept hierarchy. This is specifically practical when only a few types of stops have their granularity based on concept hierarchies.

We consider that in a concept hierarchy the highest level starts with 0 and ends with $n$, top-down, where $n$ is the lowest level in the hierarchy. In Figure 6, the level 0 corresponds to the feature type granularity (road), and $n$ always represents the lowest level, i.e., the feature instance (e.g. 13 ErieST). For illustration we show the feature instance level ($n$) in the concept hierarchy, which in reality corresponds to the instances of a relation in the database.

Let us consider some examples. The function *stopG = instance* expresses that all stops will be represented at the feature instance granularity level. *stopG = [type, roadH=1, hotelH=2]* expresses that the feature types *road* and *hotel* will respectively use the associated concept hierarchy *roadH* at level 1 and *hotelH* at level 2. All other stops will be considered at feature type granularity level.

## 4.2 *Data Mining Functions*

The data preprocessing functions explained in the previous section basically add semantic information to trajectory samples, generating trajectories as an ordered set of *stops* and *moves*. In this sense we can notice that this model generates trajectories at a higher level of abstraction, in which the mining task will be applied over semantic trajectories and not over trajectory samples.

The proposed STDMQL supports three data mining functions to extract knowledge from semantic trajectories, which in some preliminary analysis have demonstrated to be the most appropriate for mining trajectories represented as stops and moves. These mining tasks are frequent patterns (Agrawal *et al.* 1993), association rules (Agrawal and Srikant 1994), and sequential patterns (Agrawal and Srikant 1995), and we assume that the reader is familiar with these concepts. These data mining tasks will generate the following patterns from trajectories: *frequent stops* and *frequent moves*; *sequential stops* and *sequential moves*; and *association stops* and *association moves*.

We define the mining operations in a general way, such that *any/several* of the algorithms proposed in the literature can be implemented to perform these mining tasks. In other words, the operations are not limited to one specific algorithm, and therefore different input parameters can be provided.

A BNF to specify the data mining tasks is given as follows:

```
    <mining_function> ::= <pattern_type> <left_par> <mining_parameters> <right_par>
       <pattern_type> ::= FREQUENTSTOPS | FREQUENTMOVES | ASSOCIATESTOPS | ASSOCIATEMOVES |
                          SEQUENTIALSTOPS | SEQUENTIALMOVES
           <left_par> ::= (
  <mining_parameters> ::= <item> [<coma> <timeG>] <coma> <stopG> <coma> <minSup>
                          [{<coma> <other_parameter>}...]
               <coma> ::= ,
               <item> ::= ITEM <equal> <item_def>
           <item_def> ::= NAME | NAMESTART | NAMEEND | NAMESTARTEND
             <minSup> ::= MINSUP <equal> real
    <other_parameter> ::= <parameter_name> <equal> <parameter_value>
     <parameter_name> ::= string
    <parameter_value> ::= string | real | integer
          <right_par> ::= )
```

Two basic parameters for the three mining tasks are necessary: *item* and *minsup*. The parameter *item* is the attribute that will be considered in the mining step, i.e., is an element that will compose the pattern. We call this attribute *item* because this term is well known in the data mining literature. In the context of trajectories an *item* represents a set of information. On the contrary to transactional data mining in which an item is a single attribute like milk or bread, a *trajectory item* contains information about space and time. In summary, a trajectory *item* is build according to one of the following types, which can be defined at different granularities:

- *Name*: is the name of the stop or the move, according to the relation to be mined. For mining stops, it is the name of the spatial feature type (e.g. Hotel) or instance (e.g IbisHotel) in which a stop has occurred. For moves, it will be the name of the two stops which define the move (e.g. Airport-Hotel, SchipholAirport-IbisHotel).
- *NameStart*: is the name of the stop/move with the time period in which a stop or move has started (e.g. Hotel[morning], LouvreMuseum[weekend], Airport[10:00-12:00]).
- *NameEnd*: is the name of a stop or move with the time period in which a stop/move finishes (e.g. Hotel[afternoon], Museum[10:00-12:00], IbisHotel[afternoon], Hotel-Museum[afternoon]).
- *NameStartEnd*: is the name of a stop or move with the time period in which the stop/move has respectively stated and finished (e.g. Hotel[08:00-12:00][12:01-18:00], Museum[morning][afternoon], Airport-Hotel[afternoon][afternoon]).

The granularity in which both the name of the stop/move and the time is represented will be specified by *stopG* and *timeG*, respectivelly. In the following sections we provide query examples to illustrate the extraction of different kinds of patterns.

The parameter *minsup* in the mining operation specifies the minimal support of a set of items in order to be considered a pattern. Concerning trajectories, the *support* of an item is defined as the fraction of trajectories that contain this item. *minSup* is a real number between 0 and 1, inclusive.

**4.2.1** ***Mining Frequent Trajectory Patterns.*** Frequent trajectory patterns can be extracted from both stops and moves. Frequent stops are combinations of stops that have support higher than *minSup*. To understand frequent trajectory patterns let us consider a few query examples in the context of a tourism application. Let us suppose that the user wants to answer the following question: which are the types of places most frequently visited by tourists on weekdays and weekend? This question can be answered by the following simple query:

```
SELECT frequentStops (item=NameStart, timeG=WEEKEND-WEEKDAY,
                      stopG=[type,touristicPlaceH=1], minsup=0.2)
  FROM stop
```

In this query, the item that composes the pattern will contain the name of the stop and the time in which the stop started. This is expressed by the parameter *item=NameStart*. The time granularity (timeG) is weekday and weekend, while the granularity of the stop will be at the feature type level for all stops, except for those that occur at touristic places. For touristic places a concept hierarchy *touristicPlaceH* is used at the granularity level 1, and therefore touristic stops will be aggregated by types like, for instance, religious, museum, and monument. The query will generate patterns over stops that appear in at least 20% of the trajectories, and the output of this query will be frequent patterns as shown bellow.

{*Hotel[weekday]*} (s=0.45)
{*Museum[weekday]*} (s=0.15)
{*Religious[weekend], Restaurant[weekend]*} (s=0.07)

The first pattern expresses that 45% of the trajectories have a stop at hotels during the week. In the second pattern, 15% of the trajectories stop at museums during the week, and in the last example 7% of the trajectories stop at both religious places and restaurants on the weekends.

The function *frequentMoves* generates combinations of moves that have support higher than minimum support in the move dataset. For example, one single move, in which the time is not relevant, can be a pattern like {*Hotel-Airport* (s=0.30)}, which expresses a single move from Hotel to Airport, in this order, in 30% of the trajectories. A longer moving pattern like, for instance, {*Hotel-Museum, ShoppingCenter-Hotel* (s=0.25)}, expresses that 25% of the trajectories have the movements from Hotel to Museum and from Shopping Center to Hotel. In a frequent moving pattern with more than one move, the order as the moves appear in the pattern does not represent the order as the moves occur in the trajectories. In the second example, for instance, the move from Shopping Center to Hotel can occur before the move from Hotel to Museum. When the order as the moves occur in the trajectories is important, then the sequential pattern mining method should be used, as will be explained later in this section.

To extract frequent moves the user can also specify the granularity of the item with the functions *timeG* and *stopG*. For example, suppose one is interested in movements that occur with a frequency of at least 5%, aggregating these movements in morning and afternoon, and considering the exact places in which the moves occur. By exact places we mean the instances of the stops given by the funcion *stopG=instance*. Such query can be expressed as follows:

```
SELECT frequentMoves (item=NameStart, timeG=[08:00-12:00,14:00-18:00,other],
                      stopG=instance, minsup=0.05)
  FROM move
```

The item in the pattern will contain the name of the stops with the time period in which the moves started. This is defined by the parameter *item=NameStart*. The granularity of the item is specified by *stopG*. In this query example, three time intervals have been defined: from 8AM to 12AM, from 2PM to 6PM, and any other starting time is represented by 'other'. From this query, patterns like the following will be extracted.

{*Lido-ParisienHotel[other]*} (s=0.06)
{*TriumphArc-LouvreMuseum[08:00-12:00],IbisHotel-TriumphArc[08:00-12:00]*} (s=0.07)

In the first example the move goes from Lido to ParisienHotel at any time different from the intervals [8:00 to 12:00] and [14:00 and 18:00]. This move is frequent in 6% of the trajectories. In the second example, a frequent move goes from TriumphArc to LouvreMuseum, starting in between 8 and 12 in the morning, and another move in the same trajectories goes from IbisHotel to TriumphArc, also starting in between 8:00 and 12:00. This moving pattern is frequent in 7% of the trajectories.

**4.2.2** ***Mining Sequential Patterns from Trajectories.*** The sequential pattern mining technique differs from the frequent pattern mining in the order as either stops or moves occur in a trajectory. In sequential pattern mining the order as the items occur plays the essential role. It corresponds to the relative order between items, but not necessarily the absolute order of the items. Let us consider the following example: {*Hotel, Museum, ShoppingCenter*} *(s=0.35%)*. This sequential pattern expresses that 35% of the trajectories stop at Hotel, Museum, and Shopping Center, in this relative order.

Sequential moving patterns are sequences of movements that have support higher than minimum support. For instance, the pattern {*Hotel-Museum, ShoppingCenter-Hotel*} *(s=0.25%)* expresses that 25% of the trajectories have the moves from Hotel to Museum and after from Shopping Center to Hotel.

Now let us suppose that the user has the following question: which are the sequences of moves that occur most frequently in the morning and in the evening? This question can be answered by the following query:

```
SELECT seqentialMoves (item=NameEnd,timeG=[8:00-12:00 AS morning, 18:00-23:00 AS evening],
                       stopG=instance, minsup=0.03)
  FROM move
```

In this query a move has to occur in at least 3% of the trajectories in order to generate a pattern. The pattern will contain the name of the move and the time that the move finishes, and time will be aggregated in morning and evening. In this data mining query there is no aggregation on the name of the stop, i.e., the patterns will contain the places where the moves happened. A sequential pattern that will be generated by this query may be similar to the following:

{*IbisHotel-NotreDame[morning], EiffelTower-IbisHotel[evening]*} (s=0.04)

In this example, the moves from Ibis Hotel to Notre Dame in the morning, and from Eiffel Tower to Ibis Hotel in the evening, occur in 4% of the trajectories, in this relative order.

**4.2.3** ***Mining Trajectory Association Rules.*** An association rule consists of an implication of the form $X \Rightarrow Y$ (s) (c), where $X$ and $Y$ are disjoint sets of items, which in the context of our approach correspond to sets of stops or moves. The support $s$ of the rule $X \Rightarrow Y$ is given as $s(X \cup Y)$. The confidence $c$ is given as $s(X \cup Y)/s(X)$.

Trajectory association rules represent associations among either stops or moves. A query to extract this kind of pattern is similar to a query for computing frequent patterns, with at least one additional parameter, the *minimum confidence*. Different parameters can be provided to filter the rules, according to the algorithm implemented in the operations *associateStops* and *associateMoves*. An association rule extracted from stops could be for instance,

*Airport[morning]* $\rightarrow$ *Hotel[morning]* (s=0.20) (c=0.70)

This rule is generated from a combination of two stops, and expresses that in 20% of the trajectories the moving object has stoped at Airport and Hotel in the morning. Among the trajectories that stop at Airport in the morning, 70% also stop at Hotel in the morning.

Association rules over moves contain at least one pair of stops in the antecedent and another pair in the consequent of the rule, where each pair characterizes a move. For example,

*Hotel-Museum[morning]* $\rightarrow$ *Monument-Restaurant[noon]* (s= 0.15) (c=0.30)

This rule expresses that 15% of the trajectories that go from Hotel to Museum in the morning do also go from Monument to Restaurant at lunch time.

The parameters *minSup, item, timeG,* and *stopG* have the same syntax and meaning as in the trajectory frequent patterns. The parameter *minConf* is needed to filter the rules. It is a real number in the interval [0..1].

Trajectory association rules can be interesting for finding relationships among either stops or moves. Let us now consider a traffic management application where in general the user may be interested in the discovery of patterns related to traffic jams. For this application, the first task would be to compute stops with the method *CB-SMoT* (Palma *et al.* 2008), which is based on speed, and therefore would find traffic jams and very low speed regions as stops. The important places for this application (candidate stops) are the streets. The granularity of the time dimension is defined in a lower resolution, like for instance, every hour in the period of rush hours. Having computed stops and moves taking the above characteristics into account, the user may ask a question like: which are the relationships among low speed regions in a city at different time periods during rush hours? This question may be answered by the following query:

```
SELECT associateStops (item=NameStart, stopG=instance,
                       timeG=[07:00-08:00, 08:01-09:00, 16:00-17:00,
                       17:01-18:00, 18:01-19:00], minsup=0.01, minconf=0.40)
  FROM stop
```

This query will generate association rules that have at least 1% support and at least 40% confidence. Each *item* in the rule, either in the antecedent or the consequent, will contain the name of the stop and the time period in which the stop starts. The stop granularity will be at the feature instance level for all stops, and the time are specific hours within the rush hours. From this query the user may obtain association rules like, for instance:

*AvGrandArmee[07:00-08:00]* → *AvChampsElisees[08:01-09:00]* (s=0.06) (c=0.40)

This rule expresses that if there is a traffic jam at Av. Grand Armee between 7 and 8 in the morning, there is a probability of 40% to also be a traffic jam between 8 and 9 in the morning at Av. Champs Elisees.

In this section we have seen that different kinds of patterns can be easily extracted from trajectories using the proposed STDMQL, which provides powerful functionalities that transform raw trajectories into a higher level of abstraction, to meet the context of the application according to the user's needs. From this higher abstraction level (stops and moves) the user can transform the granularities of space and time, and therefore extract multiple-level semantic trajectory patterns.

In the following section we show how the semantic patterns are stored and can be filtered in post-processing steps.

## 4.3 *Storing and Querying Patterns*

Since the objective of the proposed STDMQL is to extend spatial SQL with the minimal changes, we opted to apply pattern constraints in post-processing steps. Therefore, each data mining function will automatically store the extracted patterns in a database pattern relation. Having the patterns stored in the database, the user does not need to analyze all the extracted patterns in order to find interesting ones, but SQL queries can be executed to get specific patterns, as we will explain in the remaining of this section.

**4.3.1** ***Storing Patterns.*** Frequent patterns and sequential patterns are stored in the same format. The difference relies on the meaning of the attribute *pattern*. While in sequential patterns the order of the items is crucial, in frequent patterns it is not. The relation to store frequent patterns and sequential patterns has the following structure:

```
frequentPattern/sequentialPattern (Pid integer, size integer,
                                   pattern itemSetType, support real)
```

The attributes are respectively the identifier of the pattern, which is a sequential number, the number of elements in each pattern (e.g. 2-itemset, 3-itemset), the pattern itself, which we represent and store as a *nested relation* (Abiteboul *et al.* 1989), and the support of the pattern.

Because the pattern attribute is a complex object, or in other words, a set of items, it has to be represented in a way to be easily queried. Representing a pattern as a nested relation we store the pattern as data, what allows the user to do simple queries and join patterns with data. The type *itemsettype*, which represents a nested relation, is defined by the following structure:

```
itemsettype (SFT1name varchar, SF1id integer, SFT2name varchar,
             SF2id integer, startT varchar, endT varchar)
```

The pattern schema is similar to the structure of the move relation, which is used to store patterns extracted from either stops or moves. Although the pattern structure is similar to the move relation, the instances of a pattern represent a *summarization* of data, i.e., it is an aggregation of data transformed into higher granularities. This is specially true for the time dimension, which will rarely generate a pattern if the original time stamp of either stops or moves has not been aggregated. The attributes of the relation *itemsettype* have the following semantics:

- SFT1name: is the name of the spatial feature type for a stop pattern or the first stop in a move pattern.
- SF1id: is the identifier of the instance of the spatial feature in a stop pattern or the instance of the first stop in a move pattern. This value will be NULL if the granularity level is different from feature instance.
- SFT2name: corresponds to the name of the spatial feature type of the second stop in a move pattern. The value of this attribute will be NULL if it represents a stop pattern;
- SF2id: is the identifier of the spatial feature of the second stop in a move pattern when the instance granularity level was considered, or NULL otherwise.
- startT: represents the time period in which stops or moves have started, but only if *NameStart* or *NameStartEnd* have been used to specify the *item* in the mining query. In case no time has been considered, this attribute will be NULL.
- endT: is the time period in which stops or moves have ended, and when *NameEnd* or *NameStartEnd* were specified in the mining task. In case no time has been considered, the value of this attribute will be set as NULL.

Concerning trajectory association rules, the association pattern relation has the following structure:

```
associationPattern (Pid integer, antecedent itemsettype,
                    consequent itemsettype, support real,
                    confidence real)
```

The attributes are respectively the identifier of the rule, the antecedent of the rule, the consequent, the support, and the confidence. In this type of pattern both antecedent and consequent are stored as nested relations, and therefore both have the structure of the *itemsettype*.

**4.3.2 *Querying Patterns.*** Since the patterns are stored as database relations, any query which can be performed over the data can also be executed over the patterns. Any filter or constraint may be applied directly over the patterns. For instance, let us suppose that the user is interested only in association rules that have *Hotel* in the consequent. This patterns can be retrieved by the following query:

```
SELECT *
  FROM associationPattern
 WHERE consequent.SFT1name='Hotel'
```

Now let us suppose that the user is interested in association patterns which have *weekend* as the time dimension in the antecedent of the rule. A query like the following can be performed to obtain the constrained rules:

```
SELECT *
  FROM associationPattern
 WHERE antecedent.startT='weekend' or antecedent.endT='weekend'
```

Another kind of constraint that is common and useful to filter patterns is through the number of items that compose the pattern. For instance, the user may want to know which are the frequent patterns that contain *touristic place* and have more than 3 items. This pattern constraint can be specified by the following query.

```
SELECT *
  FROM frequentPattern
 WHERE pattern.SFT1name='touristic place' AND size > 3
```

When mining data at the feature instance granularity level (e.g. IbisHotel, OakStreet) more sophisticated queries can be performed. The user can join patterns and data. The items in a pattern correspond to a geographic object, and therefore any pattern can be constrained by either the spatial or non-spatial attributes of the geographic object. For example, suppose the user is interested only in sequential patterns over stops which pass at 2-star hotels in the morning. This pattern constraint can be expressed by the following query:

```
SELECT *
  FROM frequentPattern f, Hotel h
 WHERE f.pattern.SFT1name='Hotel' AND
       f.pattern.SF1id=h.id AND  h.stars=2 AND
       (f.pattern.startT='morning' OR f.pattern.endT='morning')
```

The power of storing the discovered patterns for further analysis allows even more complex queries, where the geometry of either stops or moves can be used to filter the patterns. For instance, suppose the user is interested in how many moves that cross the bridge Pont Neuf are contained in sequential patterns. This query will join three relations: sequential patterns, moves, and bridge, as shown bellow.

```
SELECT count(m.*)
  FROM sequentialPattern s, bridge b, move m
 WHERE s.pattern.SFT1name=m.SFT1name AND
       s.pattern.SF1id=m.SF1id AND
       s.pattern.SFT2name=m.SFT2name AND
       s.pattern.SF2id=m.SF2id AND
       b.name='Pont Neuf' AND
       intersects(m.the_move.geometry,b.geometry)
```

With a few query examples, we have seen in this section the usefulness of storing patterns for post-processing. By storing the patterns as nested relations we provide to the user a simple way to execute queries over the patterns, since patterns are stored as data. In the following section we give an overview of the implementation of the proposed STDMQL.

## 4.4 *A prototype for the STDMQL*

The proposed data mining query language for trajectories can be implemented in several ways, like for instance an extension of a spatial or spatio-temporal DBMS (database management system) or in a toolkit with a friendly and graphical user interface. As a first prototype of our approach we have decided to develop a tool for mining and querying trajectories, in order to not restrict the user to one specific database management system. The proposed methodology involves spatio-temporal data (trajectories) and geographic information, and to transform and export/import this kind of data among different DBMSs is not a trivial task.

The prototype is an evolution of Weka-GDPM (Bogorny *et al.* 2006), which was specifically developed to support automatic geographic data preprocessing for spatial data mining. Within Weka-GDPM the user is able to connect to several DBMS through JDBC, like for instance Oracle, Postgres/PostGIS, and MySQL. Weka-GDPM follows OGC standards, and therefore becomes interoperable with several spatial and spatio-temporal DBMS, such as Hermes (Pelekis *et al.* 2006), which is a spatio-temporal DBMS developed on the top of Oracle Spatial.

Figure 7 presents an overview of the framework T-Miner, which we have specially developed for mining trajectories with the proposed STDMQL. On the bottom of the framework are the data, the patterns, and the background knowledge. In the center are the data preprocessing and data mining modules, which contain the components of the STDMQL. On the top of these modules we developed a graphical GUI to provide to the user several facilities for querying and mining both data and patterns.

Most functionalities of the proposed STDMQL have been implemented in the prototype in order to evaluate the methodology for trajectory knowledge discovery. The system connects to the user specified spatial or spatio-temporal DBMS that should contain trajectories and all geographic object types (database relations) that are relevant for the application. From the set of objects the user can choose the trajectory dataset and all geographic object types that will be considered as the candidate stops. Having selected the trajectory dataset and the candidate stops, the module *generateSM* will generate *semantic trajectories*, and create and instantiate in the database, a relation of stops and a relation of moves. According to the application, the user can choose among the methods I-SMoT and CB-SMoT to compute stops and moves, which are respectively based on intersection and speed.

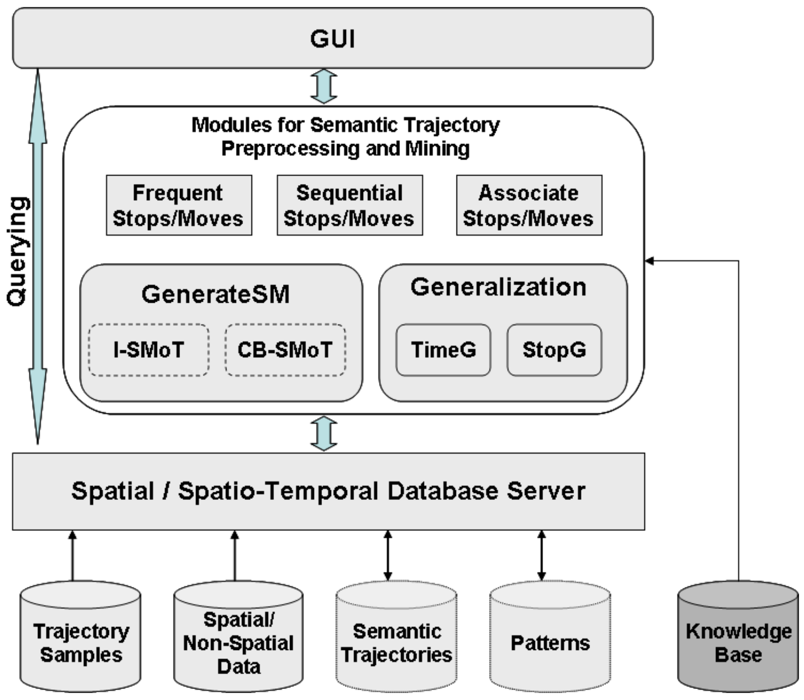

Figure 7. T-Miner - a Framework for Trajectory Querying and Mining

Stops and moves are stored in the database at the lowest granularity level, i.e., at the feature instance granularity. Before mining the user can transform the granularity of the space and time dimension, using the respective *stopG* and *timeG* functions. For the granularity transformation these functions may use background knowledge as concept hierarchies and pre-defined time granularities, which are stored in the *knowledge base*.

After the generalization, the user can apply the data mining algorithms to compute frequent stops/ moves, sequential stops/moves and association stops/moves. The output of these modules are the *patterns*, which are stored in the database.

At any time, from the graphical GUI the user can directly query the raw trajectories, the spatial and non-spatial data, the semantic trajectories and the patterns.

So far we have tested the prototype with real data stored in a Postgresql/Postgis database, using the sequential pattern mining method.

## 5 Conclusions and Future Work

Trajectory data are normally available as sample points, what makes their analysis in different application domains expensive from a computational point of view and quite complex from a user's perspective. Although several works have been developed so far for both querying and mining trajectories of moving objects, these works have focused on the geometric properties of trajectories and, therefore, deal with trajectory sample points. Indeed, most of these works focus on trajectory similarity and dense regions.

In this paper we addressed the problem of querying and mining trajectories from a semantic point of view. We have considered trajectories from a higher abstraction level, and have focused on the semantic properties of trajectories, therefore providing mechanisms to extract more meaningful and human understandable patterns. More specifically, we have presented through a data mining query language for trajectories a new methodology for extracting multiple-level *patterns* from moving object data. We introduced powerful data preprocessing operations which allow the user to transform trajectory sample points into higher levels of abstraction, in order to extract semantic and comprehensive patterns from trajectories. To accomplish this goal we adopted the model of stops and moves, where the user can specify the important parts of trajectories that are specially relevant for the application.

In summary, we make the following contributions:

- We extend spatial SQL with operations for semantic trajectory knowledge discovery. The aim is to facilitate the process to the user, which may use spatial SQL without having to learn a new language;
- We provide a data preprocessing strategy to add semantic geographic information to trajectories, before applying data mining techniques. This strategy tremendously simplifies and optimizes the data mining step. More specifically, we provide two different methods to automatically integrate trajectory samples and geographic information into a higher abstraction level, where the user can define the important parts of trajectories. This aggregation is of fundamental importance for the discovery of patterns in real application of different domains;
- We provide powerful data discretization functions for space and time, which allows the user to extract patterns at several granularity levels;
- The output of the proposed language are semantic trajectory patterns, which are stored in the database for further querying. By generating semantic patterns, which are at a higher level of abstraction, we significantly reduce the complexity of pattern interpretation in post-processing, which is still a human-dependent task in the discovery process;
- The language is simple and can be implemented in several spatial or spatio-temporal database management systems, or in a toolkit with friendly and graphical user interfaces. We have implemented a first prototype of the language in a tool, in order to not restrict the user to one specific DBMS.

By adopting the model of stops and moves of trajectories we provide to the user aggregated information about space and time. Therefore, we facilitate not only trajectory mining, but also trajectory querying. Any information about trajectories can be directly obtained with queries over stops and moves. The generation of stops and moves in a preprocessing step tremendously simplifies and optimizes trajectory queries. It reduces the query complexity in both formulation and computational time (Alvares *et al.* 2007b).

As future work we will extend the proposed language to support trajectory classification, and investigate the use of ontologies to infer new knowledge in the trajectory knowledge discovery process. We will also evaluate the possibility to implement the proposed STDMQL as an extension of a spatio-temporal DBMS.

## 6 Acknowledgments

This research has been funded by the Brazilian Agency CAPES and the European Union (FP6-IST-FET Programme Project n. FP6-14915, GeoPKDD: Geographic Privacy-Aware Knowledge Discovery and Delivery).

## References

Abiteboul, S., Fischer, P. and Schek, H.J. (Eds) , 1989, *Nested Relations and Complex Objects in Databases*, Lecture Notes in Computer Science Vol. 361 (Springer-Verlag).

Agrawal, R., Imielinski, T. and Swami, A.N., 1993, Mining Association Rules between Sets of Items in Large Databases.. In *Proceedings of the SIGMOD Conference*, P. Buneman and S. Jajodia (Eds) (ACM Press), pp. 207–216.

Agrawal, R. and Srikant, R., 1994, Fast Algorithms for Mining Association Rules in Large Databases.. In *Proceedings of the VLDB*, pp. 487–499.

Agrawal, R. and Srikant, R., 1995, Mining Sequential Patterns. In *Proceedings of the ICDE*, P.S. Yu and A.L.P. Chen (Eds) (IEEE Computer Society), pp. 3–14.

Alvares, L.O., Bogorny, V., de Macedo, J.F., Moelans, B. and Spaccapietra, S., 2007a, Dynamic Modeling of Trajectory Patterns using Data Mining and Reverse Engineering. In *Proceedings of the Twenty-Sixth International Conference on Conceptual Modeling - ER2007 - Tutorials, Posters, Panels and Industrial Contributions*, 83, Auckland, New Zealand, November (CRPIT).

Alvares, L.O., Bogorny, V., Kuijpers, B., de Macedo, J.A.F., Moelans, B. and Vaisman, A., 2007b, A Model for Enriching Trajectories with Semantic Geographical Information. In *Proceedings of the ACM-GIS*, Seattle, USA (New York, NY, USA: ACM Press).

Alvares, L.O., Bogorny, V., Kuijpers, B., de Macelo, J.A.F., Moelans, B. and Palma, A.T., 2007c, Towards Semantic Trajectory Knowledge Discovery. Technical report, Hasselt University.

Arni, F., Ong, K., Tsur, S., Wang, H. and Zaniolo, C., 2003, The Deductive Database System LDL++. *TPLP - Theory and Practice of Logic Programming*, **3**, 61–94.

Bogorny, V., Kuijpers, B. and Alvares, L.O., 2007, Reducing Uninteresting Spatial Association Rules in Geographic Databases using Background Knowledge: a Summary of Results. *International Journal of Geographical Information Science.*

Bogorny, V., Palma, A.T., Engel, P. and Alvares, L.O., 2006, Weka-GDPM: Integrating Classical Data Mining Toolkit to Geographic Information Systems. In *Proceedings of the WAAMD*, Florianopolis, Brazil (SBC), pp. 9–16.

Bonchi, F. and Lucchese, C., 2007, Extending the state-of-the-art of constraint-based pattern discovery. *Data Knowl. Eng.*, **60**, 377–399.

Boulicaut, J.F. and Masson, C., 2005, Data Mining Query Languages.. In *The Data Mining and Knowledge Discovery Handbook*, O. Maimon and L. Rokach (Eds) (Springer), pp. 715–727.

Brakatsoulas, S., Pfoser, D. and Tryfona, N., 2004, Modeling, Storing, and Mining Moving Object Databases. In *Proceedings of the IDEAS* (IEEE Computer Society), pp. 68–77.

Calders, T., Goethals, B. and Prado, A., 2006a, Integrating Pattern Mining in Relational Databases. In *Proceedings of the PKDD*, J. Fürnkranz, T. Scheffer and M. Spiliopoulou (Eds), 4213 of *Lecture Notes in Computer Science* (Springer), pp. 454–461.

Calders, T., Lakshmanan, L.V.S., Ng, R.T. and Paredaens, J., 2006b, Expressive power of an algebra for data mining. *ACM Trans. Database Syst.*, **31**, 1169–1214.

Cao, H., Mamoulis, N. and Cheung, D.W., 2006, Discovery of Collocation Episodes in Spatiotemporal Data. In *Proceedings of the ICDM* (IEEE Computer Society), pp. 823–827.

Chen, X. and Petrounias, I., 1998, Language Support for Temporal Data Mining. In *Proceedings of the Proceedings of the Second European Symposium on Principles of Data Mining and Knowledge Discovery* (London, UK: Springer-Verlag), pp. 282–290.

Egenhofer, M., 1994, Spatial SQL: A query and presentation language. *IEEE Transactions on Knowledge and Data Engineering*, **6**, 86–95.

Ester, M., Kriegel, H.P., Sander, J. and Xu, X., 1996, A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, E. Simoudis, J. Han and U.M. Fayyad (Eds) (AAAI Press), pp. 226–231.

Fayyad, U.M., Piatetsky-Shapiro, G. and Smyth, P., 1996, From Data Mining to Knowledge Discovery in Databases. *AI Magazine*, **17**, 37–54.

Giannotti, F., Nanni, M., Pinelli, F. and Pedreschi, D., 2007, Trajectory pattern mining. In *Proceedings of the KDD*, P. Berkhin, R. Caruana and X. Wu (Eds) (ACM), pp. 330–339.

Gudmundsson, J. and van Kreveld, M.J., 2006, Computing longest duration flocks in trajectory data. In *Proceedings of the GIS*, R.A. de By and S. Nittel (Eds) (ACM), pp. 35–42.

Han, J., 1995, Mining Knowledge at Multiple Concept Levels. In *Proceedings of the CIKM* (ACM), pp. 19–24.

Han, J., Fu, Y., Wang, W., Koperski, K. and Zaiane, O., 1996, Dmql: A data mining query language for relational databases. In *Proceedings of the SIGMOD'96 Workshop on Research Issues in Data Mining and Knowledge Discovery*, Montreal, Canada, pp. 27–33.

Han, J., Koperski, K. and Stefanovic, N., 1997, GeoMiner: A System Prototype for Spatial Data Mining. In *Proceedings of the SIGMOD Conference*, J. Peckham (Ed.) (ACM Press), pp. 553–556.

Imieliński, T. and Virmani, A., 1999, MSQL: A Query Language for Database Mining. *Data Mining and Knowledge Discovery*, **3**, 373–408.

Laube, P., Imfeld, S. and Weibel, R., 2005, Discovering relative motion patterns in groups of moving point objects. *International Journal of Geographical Information Science*, **19**, 639–668.

Lee, J.G., Han, J. and Whang, K.Y., 2007, Trajectory clustering: a partition-and-group framework.

In *Proceedings of the SIGMOD Conference*, C.Y. Chan, B.C. Ooi and A. Zhou (Eds) (ACM), pp. 593–604.

Malerba, D., Appice, A. and Ceci, M., 2004, A Data Mining Query Language for Knowledge Discovery in a Geographical Information System.. In *Proceedings of the Database Support for Data Mining Applications*, pp. 95–116.

Meo, R., Psaila, G. and Ceri, S., 1996, A New SQL-like Operator for Mining Association Rules. In *Proceedings of the VLDB*, T.M. Vijayaraman, A.P. Buchmann, C. Mohan and N.L. Sarda (Eds) (Morgan Kaufmann), pp. 122–133.

OGC, "Topic 5, OpenGIS abstract specification - Features (Version 4) (1999)", Available at: http://www.OpenGIS.org/techno/specs.htm. Accessed in August (2005) 1999.

Palma, A.T., Bogorny, V. and Alvares, L.O., 2008, A Clustering-based approach for discovering interesting places in trajectories. In *Proceedings of the ACMSAC*, Fortaleza, Brazil (New York, NY, USA: ACM Press).

Pelekis, N., Theodoridis, Y., Vosinakis, S. and Panayiotopoulos, T., 2006, Hermes - A Framework for Location-Based Data Management. In *Proceedings of the EDBT*, Y.E. Ioannidis, M.H. Scholl, J.W. Schmidt, F. Matthes, M. Hatzopoulos, K. Böhm, A. Kemper, T. Grust and C. Böhm (Eds), 3896 of *Lecture Notes in Computer Science* (Springer), pp. 1130–1134.

Spaccapietra, S., Damiani, M.L., de Macedo, J.A.F., Parent, C., Porto, F. and Vangenot, C., 2007, A Conceptual View on Trajectories. Technical report, Ecole Polytechnique Federal de Lausanne.

Tsoukatos, I. and Gunopulos, D., 2001, Efficient Mining of Spatiotemporal Patterns. In *Proceedings of the SSTD*, C.S. Jensen, M. Schneider, B. Seeger and V.J. Tsotras (Eds), 2121 of *Lecture Notes in Computer Science* (Springer), pp. 425–442.

Verhein, F. and Chawla, S., 2006, Mining Spatio-temporal Association Rules, Sources, Sinks, Stationary Regions and Thoroughfares in Object Mobility Databases. In *Proceedings of the DASFAA*, M.L. Lee, K.L. Tan and V. Wuwongse (Eds), 3882 of *Lecture Notes in Computer Science* (Springer), pp. 187–201.

Wang, H. and Zaniolo, C., 2003, ATLaS: A Native Extension of SQL for Data Mining.. In *Proceedings of the SDM*, D. Barbará and C. Kamath (Eds) (SIAM).