UNIVERSITÀ DEGLI STUDI DI MILANO

GRADUATE SCHOOL IN COMPUTER SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

PhD in Computer Science
Cycle XXVIII


# Spatio-textual trajectories: models and applications

Supervisor: Prof. Maria Luisa Damiani

PhD Thesis of:
Hamze Issa

Academic year 2015 - 2016

# Abstract

The Thesis concerns the management and analysis of *mobility data*. The pervasiveness of geo-positioning technologies, sensors and communication networks has led to the collection of large amounts of data on the movement of objects. A major issue is thus how to effectively organize and access such a data. An important category of mobility data is that of *spatial trajectories*. A spatial trajectory describes the continuous movement of an object in a reference space, e.g. the Euclidean plane, through a set of temporally annotated and ordered sample points. Spatial trajectories can represent the movement of vehicles, people and animals, for example equipped with a GPS receiver.

Yet, spatial trajectories can represent the movement exclusively in terms of locations, thus the evolution of the context in which the movement takes place is ignored. In general, the *contextual data* can be acquired directly from the environment, for example through the use of sensors, or be the result of an analytical process. In this sense, spatial trajectories lack expressivity. An important step towards the specification of richer and more expressive data models, is the *symbolic trajectories* data model. This model allows for the representation of sequences of activities (or labels), each annotated with a time period. A major novelty of the model is the query language that is based on pattern matching. Nevertheless also this solution presents important limitations because the notion of symbolic trajectory is orthogonal to that of spatial trajectory and thus does not include any location information.

The Thesis addresses the problem of integrating the spatial dimension with the symbolic one, providing as well a mechanism enabling the efficient access to a database of *spatio-textual trajectories*. The major contribution of this research is the proposal of a framework for the indexing of spatio-textual trajectories. The goal of the index is to support the processing of queries taking the form of *Sequenced queries*, that is complex queries expressed as sequences of ordered simple spatio-textual queries. The index is called *IRWI*. The system is hybrid in that it combines an *R-tree* for the indexing of spatial trajectory segments with *inverted files* for the indexing of the textual part. A Sequenced query is next processed in parallel, evaluating first every single query of the sequence,

and finally analyzing and recomposing the sequence. A related though different topic of the Thesis regards the study of techniques for mobility data analysis. The objective is to extract behavioral patterns from spatial trajectories and next represent them in terms of spatio-textual trajectories. The major contribution is the definition of an algorithm for the segmentation of the trajectories based on clustering and relying on a novel model of noise. Finally, a case study illustrates and summarizes the methodology proposed for the analysis and representation of mobility data. Specifically the above segmentation technique is used for the extraction of the migratory behavior of a group of animals equipped with GPS collars. Next such a knowledge is encoded in terms of spatio-textual trajectories.

The results of this research, spanning data representation and analysis, have been presented in conferences and journals.

# Acknowledgments

It is a humbling experience to acknowledge those people who have, mostly out of kindness, helped along the journey of my PhD. I am indebted to so many for encouragement and support.

First and foremost, I want to thank my advisor Prof. Maria Luisa Damiani for being a tremendous mentor for me. Over my PhD journey, she continuously offered her kindness, guidance and support. She was always there to encourage, motivate and guide me in the right direction. The joy and enthusiasm she has for research were contagious and motivational for me, especially during the tough times. Her efforts, support and patience will never be forgotten.

I would like to thank the reviewers of my thesis, Prof. Ralf Hartmut Güting, Prof. Goce Trajcevski and Prof. Dino Pedreschi for their excellent and detailed reviews. I truly appreciate the time they spent and thank them for their valuable comments and corrections.

I am very grateful to the collaborators for lending me their expertise. Specially, I greatly appreciate the group of Prof. Ralf Guting at Hagen University and the group of Prof. Francesca Cagnacci at Fondazione Edmund Mach for the fruitful collaboration we had throughout my thesis. Working with them was both enjoyable and productive.

I gratefully acknowledge the scholarship received towards my PhD from Università degli Studi di Milano.

I am thankful to all my friends and colleagues for their friendship, support, and for creating a cordial working environment. Special thanks go to Prof. Rima Kilany, Prof. Roula Karam, Fatima Hachem and Zaffar Haider Janjua for their support and encouragement.

My deepest heartfelt gratitude goes to my partner, Malak Safa, for sharing with me every moment of this journey with tremendous love and support. She unselfishly encouraged me to pursue my ambitions and made my dreams hers. I thank her for enduring my absence, for always being there for me, for reading my papers and thesis nearly as many times as I did, for sheering me up, for being my best friend and sweetest companion and for giving me joy, hope and reason.

Finally, my deep gratitude goes to my precious family. I thank my brothers

and sister for always supporting me in every walk of life and encouraging me to overcome challenges. I am forever indebted to my parents for their immense sacrifices. I thank my Mom for her infinite love, and for lovingly dedicating her life so we can enjoy ours. I thank my Dad for being a great role model in strength, character and commitment. They have always believed in me and provided unconditional love and support throughout my entire life. I never would have made it here without them, and I hope I have made them proud. This doctoral dissertation is dedicated to them.

For all of you, Thank you.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Motivation

### 1.1.1 Mobility Data

Mobility is one of the major contexts that characterize the current trends of developments of any modern society. Due to the wide spread of GPS-embedded devices and geo-location services and technologies, recent years have witnessed a proliferation of applications in different disciplines that collect and record location and possibly additional contextual information about moving entities, including people, animals, objects. That motivates the increased availability of large *mobility data sets* [51]. Mobility data often take the form of location histories, commonly referred to as *spatial trajectories*. Spatial trajectories, especially those reporting the traces of a large population of individuals over a significant period, represent a valuable source of information and an enabling factor in a variety of applications. For example, intelligent transportation, and more generally smart cities applications, can leverage citizens' mobility patterns detected in trajectories (e.g., traffic distribution in space and time), to plan more efficient public transportation and social services, in this way contributing to a better quality of life [53]. Similarly, in the field of animal ecology, modern animal telemetry and sensor networks (e.g., GPS receivers and other sensors mounted on devices deployed on animals, such as collars) make it possible to collect large amounts of mobility data at a unprecedented level of accuracy, opening up new exciting perspectives in the study of animal behavior [22].

Generally speaking, *trajectories* are complex structured data encompassing time and data. Spatial trajectories, for example, are sequences of temporally annotated points $(p_1, t_1)...(p_n, t_n)$ sampling the physical movement of the entity in a geographical space [68]. The sampling rate depends on both the application

needs and the geo-location technologies employed for data collection. Whenever the sampling rate is sufficiently high, the velocity of the object is likely not to change significantly between two consecutive samples, thus the missing locations can be estimated using (linear) interpolation. The result is an approximate representation of the continuous movement as shown in Figure 1.1. However, historical mobility data is not limited to spatial trajectories. Following current trends, time-varying data other than location can be acquired for example from mobile sensors or result from some sophisticated analytical task, for example, the Geolife dataset[67]. As a consequence, trajectories may have a complex structure.

Mobility datasets tend also to have a huge size. In this sense, mobility data represent a prominent example of "big data". For example the dataset released by the New York City Taxi & Limousine Commission is described as a *staggeringly detailed historical dataset covering over 1.1 billion individual taxi trips in the city from January 2009 through June 2015. Taken as a whole, the detailed trip-level data is more than just a vast list of taxi pickup and drop off coordinates: it's a story of New York. How bad is the rush hour traffic from Midtown to JFK? Where does the Bridge and Tunnel crowd hang out on Saturday nights?*[1]. Note that in this example, the taxi trips take the form of spatial trajectories while the reported questions exemplify the kind of analysis that could be performed over such trajectories.

In the light of these considerations, it is clear that the efficient handling of large amounts of complex structured trajectories becomes a prime demand for organizations willing to capitalize on mobility data. That raises important challenges because of the intrinsic limitations of the available technologies, still anchored to a *GPS-driven view* of the movement. Such challenges cannot be adequately tackled unless devising, developing and experimenting novel techniques for trajectory data management and analysis, which is the topic of this thesis.

[1] http://toddwschneider.com/posts/analyzing-1-1-billion-nyc-taxi-and-uber-trips-with-a-vengeance/

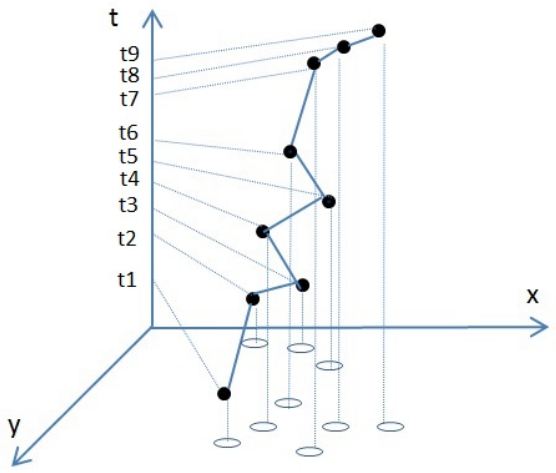



Figure 1.1: Spatio-temporal representation of a spatial trajectory

### 1.1.2 Trajectory Data Management

As we have seen, a trajectory often takes the form of a spatial trajectory representing the continuous movement of an object. A straightforward approach to trajectory data management is to store spatial trajectories in a standard *spatial database*, e.g., PostGIS [18]. Indeed, this is the naive approach entailing a separate representation of the two components of the object, i.e., space and time, as well as the ad-hoc management of the interrelationship between the two components. This solution is typically very inefficient and thus not suitable for large trajectories datasets. A much more effective solution is to store spatial trajectories in a *moving object database* [30]. In essence, a moving object database provides a set of data types for the representation of time-varying objects, such as geometric objects, e.g., *moving point*, and objects of simple type, e.g., *moving real*. In addition, a set of operations support the manipulation of the spatial and temporal components, either taken separately or simultaneously. For example, one can query when two moving vehicles are at a distance of at most 100 meters or determine the percentage of people that move with an average speed greater than 10 km/h in a given spatial region. While dedicated indexing frameworks have been developed to speed up the processing of spatio-temporal queries [13, 55, 47], the development of effective access methods remains a priority.

In the recent years, especially following the spread of sensors and analytical techniques, there has been an increasing concern for the notion of *semantic trajectory* [45]. Basically, semantic trajectories are trajectories that, besides the location history, incorporate supplementary contextual information, also called *annotations*, such as the weather conditions during a travel, the places of in-

terest visited by a tourist (hotels, entertainment spots), the sequence of home ranges traversed by an animal, the activities (shopping, driving), the transportation means used during a trip, etc. The ultimate goal of semantic trajectories is to facilitate the understanding of the moving object behavior. For instance, biologists can leverage the information on the sequences of land types traversed by the observed animals to analyze the interaction of those animals with the environment. Presently, however, the notion of semantic trajectory is exclusively defined at conceptual level, namely in an abstract and substantially informal way. This means that no operational artifact has been developed such as a 'semantic trajectory database'.

The notion of trajectory annotation has been given a precise meaning in the *symbolic trajectory* data model recently presented in [31]. Accordingly, an annotation is simply a short character string typically representing the value of a categorical attribute. For example the annotation can regard the transportation means used by the individual moving in the city. A symbolic trajectory can be represented by a stepwise function, as shown in Figure 1.2.

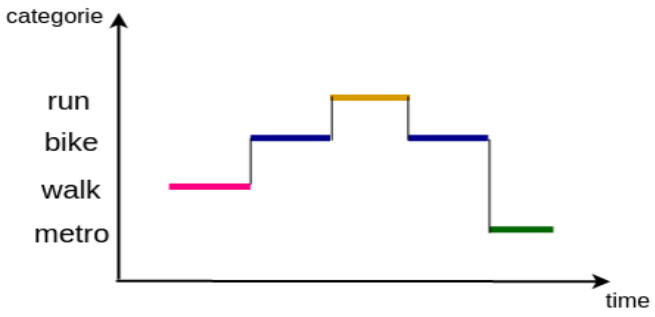



Figure 1.2: Stepwise representation of a symbolic trajectory representing the sequence of transportation means used by the individual in time

More specifically, a symbolic trajectory is a time dependent label, namely a function from time into labels. Such a function can be represented as a sequence of pairs, called units, $(I_1, l_1), ..., (I_n, l_n)$ where $I_j$ is a time interval and $l_j$ a label. For example, a symbolic trajectory can be written:

$$([8:30-8:45), walk)([8:45-9:13], train)([9:30-10:00], walk)$$

The symbolic trajectory data model includes a language for pattern matching and rewriting [31]. Pattern matching is used to retrieve symbolic trajectories matching a given pattern, whereas rewriting can be used to translate a symbolic

trajectory into some other form, classify it into certain categories, or retrieve the parts matching a pattern [31].

Departing from the traditional notion of trajectory, symbolic trajectories represent a major step ahead towards the definition of an effective and modern trajectory data model. However, important issues are still open. One of the major concerns is related to the fact that, in practice, symbolic and spatial trajectories are handled as if they were orthogonal concepts, despite the intuitive interrelationships that may exist in real applications between them. That has important consequences, in particular queries combining spatio-temporal and symbolic conditions cannot be straightforwardly solved by using one of the two models, alone. For example, even the processing of simple queries such as *determine how many people use public transport to move from Milan to Rome* can result to be extremely inefficient execution. For a more comprehensive solution, a tighter integration of spatial and contextual data is desirable. In this thesis we address such an issue.

### 1.1.3 Research Focus

This thesis focuses primarily on the problem of combining spatial and symbolic trajectories. That leads to the definition of trajectories containing both geometric information and textual information. These trajectories are called *spatio-textual*. Two main questions are addressed in this research:

(i) How to efficiently access large datasets of spatio-textual trajectories

(ii) How to extract textual annotations from spatial trajectories (and thus build spatio-textual trajectories).

**(i) Efficient Access to Spatio-Textual Trajectories.** We argue that the problem is rooted in real applications. For example a popular dataset containing both spatial and textual trajectories (separately) is GeoLife. This dataset is used throughout the thesis as running example.

GeoLife [67] is a well-known dataset reporting the traces of a group of individuals monitored in Beijing for over three years. GeoLife consists of two distinct datasets. The main dataset contains the spatial trajectories of 178 individuals in the form of timestamped point sequences, i.e., $\{(t_i, p_i)\}_{i\in[1,n]}$ where $t_i$ is the timestamp of point $p_i$. In addition, Geolife contains symbolic trajectories on the transportation modes for a subset of 69 individuals. A sequence takes the form $\{(I_i, l_i)\}_{i\in[1,m]}$, where $I_i$ is a time interval and $l_i$ a label in the set: {walk, bike, car, bus, airplane, other}. An example of symbolic trajectory is shown in Table 1.1.

| Start Time | End Time | Transportation Mode |
|---|---|---|
| 2007/10/19 05:23:15 | 2007/10/19 05:51:00 | taxi |
| 2007/10/19 05:52:18 | 2007/10/19 09:39:28 | walk |
| 2007/10/19 11:18:44 | 2007/10/19 11:53:40 | bike |

Table 1.1: GeoLife: a fragment of a symbolic trajectory

This dataset exemplifies a situation increasingly common in modern applications, that is the coexistence of heterogeneous and temporally aligned trajectories describing different aspects of the object's movement. To better illustrate the practical usefulness of spatio-textual trajectories resulting from the integration of the original trajectories, we report a few examples of queries over the Geolife dataset involving conditions on space, time and text. These queries could be of interest for example for a urban planner.

$Q_{1.1}$. *Find people who use bikes in a park region during the weekend*

$Q_{1.2}$. *Find people who move from region A to region B by their own cars in less than 2 hours*

$Q_{1.3}$. *Find the number of people in a city that use public transport to move from home to a far away workplace.*

These queries are difficult to solve efficiently, because the conditions to be satisfied concern all of the three dimensions, while existing query processing techniques and indexing frameworks are designed for space-time only. Moreover, queries may consist of sequences of conditions to be solved in the given order, thus differently from the standard SQL queries. All that calls for the development of a suitable query processing framework enabling the fast access to spatio-textual trajectories.

**(ii) Extracting Textual Annotations from Spatial Trajectories.** The textual annotations or labels appearing in spatio-textual trajectories have an application-dependent meaning. Such labels can be obtained in different ways. For example, labels can be added manually by users, for instance involved in a data collection campaign, or labels can denote mobility patterns extracted from spatial trajectories, for instance using data mining techniques. The latter case is the most interesting. A popular class of mobility patterns is called stop-and-move [45]. The patterns of this class express the movement as a sequence of transitions from one stop to another, where the stop indicates the temporary suspension

of the movement at the chosen level of abstraction. Interestingly, the stop-and-move pattern can be straightforwardly represented as symbolic trajectory with labels representing stops (e.g., places) and moves (e.g., transportation modes). Various techniques have been proposed in the literature to extract stop-and-move patterns from human and object trajectories such as [70]. As we will demonstrate later on, these techniques have several limitations, in particular rely on narrow assumptions that compromise the generality of the approach. Moreover, the validation on real data is often insufficient. In this thesis, the stop-and-move pattern is revisited and restrictive assumptions have been relaxed in favor of a more general solution which is subsequently applied to a case study in the field of animal ecology.

## 1.2 Contributions

The contribution of this thesis is two-fold. The first contribution is a spatio-textual trajectory data model together with a novel indexing framework, called IRWI, for the efficient access to spatio-textual trajectories. The goal in this case is not to propose another model alternative to symbolic trajectories but rather to devise efficient query processing techniques. The second contribution is a novel spatio-temporal clustering technique, called SeqScan for the detection of stops and moves in low-sampling rate spatial trajectories. The result is a summarized representation of the movement that can be expressed as spatio-textual trajectory. The key ideas developed for each topic are briefly summarized in the sequel.

### 1.2.1 The IRWI Indexing Framework

A spatio-textual trajectory is defined as a sequence of units $u_1, ..u_n$ with

$$u_i = (I_i, seg_i, l_i)$$

The unit indicates that during the time interval $I_i$ the moving object is located along the segment $seg_i$ and its activity or status is labeled $l_i$. We introduce a new class of queries, called *sequenced queries*. A sequenced query is defined as a sequence of *simple range queries* that are to be solved in the respect of the temporal order. A simple range query, denoted (I,L,R), consists of a time period I, a spatial region R and a set of labels L. Sequenced queries are important because they represent the skeleton of more complex queries. For the efficient processing of sequenced queries, we propose the IRWI indexing framework. A key feature of the IRWI index is that all of the dimensions of the trajectory (spatial, symbolic and temporal) are indexed simultaneously. Basically, the IRWI

index is an R-tree [32] augmented with inverted files [71]. The inverted files are stored in the internal nodes, in order to keep track of the labels present in each subtree. As a consequence, at the level of a simple query, all the range conditions (spatial, temporal and textual) can be evaluated together. The result is a faster query processing compared to evaluating each condition in a separate index. Another key feature of IRWI is that all of the simple queries in the sequence are processed concurrently. To make this strategy sustainable, a number of techniques are put in place to enable the early pruning of irrelevant trajectories, including a spatio-textual cost function. Experiments, conducted on both real and synthetic datasets, show a gain of 50% in CPU time and I/O over state-of-the-art techniques, i.e., IF-R* [69] and IR-tree [17]. In summary, the major contributions of this part of the thesis are as follows:

- Introduction of the novel concepts of spatio-textual trajectory and sequenced query. Proposal of a first data and query model.
- Development of the IRWI index for the efficient access to spatio-textual trajectories.
- Specification of usage scenarios for spatio-textual trajectories.

### 1.2.2 The Stay Region Model and the SeqScan Algorithm

The stay-region model is the conceptualization of a mobility pattern of type 'stop-and-move' and takes inspiration from the migratory behavior of animals [12]. Stops are called *stay regions*. A stay region is the residence of an object. An object can experience periods of absence from a stay region or, conversely, definitely leave the stay region for another stay region. It has been shown that this model cannot be built on existing segmentation and clustering techniques such as [20]. A novel stop-temporal clustering solution is thus proposed, called SeqScan. This technique partitions the spatial trajectory into a sequence of temporally disjoint and spatially separated sub-trajectories representing stay regions, excursions, and transitions from one region to another regions. The technique is built on an existing density-based clustering algorithm [24], from which it differs, however, in many aspects especially related to the use of time. For validation purposes, the algorithm has been applied to the study of animal migrations, in collaboration with a group of biologists (Fondazione E. Mach (TN)), and then contrasted with a more recent clustering algorithm developed by biologists [11, 12]. In summary, the main achievements of this part of the thesis are:

- The SeqScan clustering algorithm for the extraction of sequences of stay regions [20].

- An application of the technique for the study of roe deer migrations and comparison with a clustering solution developed by biologists.
- Use of the case study to represent the summarized movement in terms of spatio-textual trajectories.

## 1.3 Organization of the Thesis

The rest of the thesis is structured as follows:

- Chapter 2 reviews existing literature on trajectory data models, focusing in particular on the three major paradigms: spatial trajectories, semantic trajectories and symbolic trajectories
- Chapter 3 presents the framework for the efficient access to spatio-textual trajectories, finally focusing on the IRWI index
- Chapter 4 presents the spatio-temporal clustering algorithm SeqScan and its application to the study of the animals' migratory behavior
- Chapter 5 presents two usage scenarios illustrating practical applications of spatio-textual trajectories
- Chapter 6 concludes the thesis with a discussion on the open issues.

# Chapter 2

# Trajectory Data Models: Literature Review

In this chapter, we review related literature. In particular, we survey trajectory data models focusing on the three major paradigms of *spatial trajectories, semantic trajectories and symbolic trajectories*. In Section 2.1, we present popular data models for the representation of spatial trajectories together with major indexing techniques developed for the efficient access to spatial trajectories databases. In Section 2.2, we introduce the semantic trajectory as a conceptual model for annotated trajectories. Finally, Section 2.3 introduces the symbolic trajectory data model proposed for modeling and querying annotated trajectories in databases. Throughout this chapter, we refer to the Geolife dataset for the examples. We recall that Geolife contains both types of mobility information: spatial and contextual, the latter describing the transportation means used by the individuals during the observation period.

## 2.1 Spatial Trajectories

A spatial trajectory [68] is defined as a continuously time-varying position that describes the movement of a moving entity over time. At lower level of abstraction, a spatial trajectory consists of a sequence of timestamped positions. Over the past decades, massive amounts of trajectory data have been generated by GPS and other positioning devices for the benefit of different applications. However, traditional database management systems are unable to handle large amounts of spatial trajectories in an efficient way. In fact, a traditional database forces a decomposition of spatial trajectories into multiple attributes of simple type and in multiple rows. This makes even simple queries difficult to formulate, and furthermore hopelessly inefficient to process because the decomposed values must be reconstructed at query time. The limitations of the traditional database technology motivates the extensive research on novel database models conducted in the late 1990s and early 2000.

### 2.1.1 Spatial Trajectories Data Models

The foundational model is the Moving Object data model [29]. The model consists of a set of abstract data types for the representation and manipulation of time-varying objects, including spatial objects of type point, line and polygon and objects of simple type, e.g., real. Data types for time-varying objects are created using the *moving* type constructor: given an argument of type $\alpha$, $moving(\alpha)$ defines a type whose values are functions from time to the domain of $\alpha$. For example the values of type *moving(point)*, abbreviated in *mpoint*, are functions from time to space, while those of type *moving(real) (mreal)* are functions from time to the domain of real numbers. A value of type *mpoint* describes the movement of an object in space, namely a spatial trajectory, while an attribute of type *mreal*, for example, can represent the time varying distance between two moving points describing, e.g., two vehicles.

The rich set of types provided by the model offers a large set of operations. We list a subset of operators in Table 2.1 while the full set of operations can be found in [29]. For example, the **atinstant** operator computes the position of a moving point at a given instant of time (yielding a pair of an instant and a point), the **trajectory** operator projects a moving point into the 2-D space (resulting in a 2-D curve). The distance between two moving points is calculated by the **distance** operator, while **atmin** restricts the moving real to the instants where the value is minimum. While these data types characterize the *abstract* data model, the *discrete* data model [26] defines the concrete representations and data structures for all the types of the abstract model. In particular, the

Table 2.1: Moving Object data model: a subset of operators.

| Operator | Signature |
|---|---|
| **deftime** | $moving(\alpha) \rightarrow periods$ |
| **trajectory** | $moving(\text{point}) \rightarrow line$ |
| **initial** | $moving(\alpha) \rightarrow intime(\alpha)$ |
| **derivative** | $mreal \rightarrow mreal$ |
| **speed** | $mpoint \rightarrow mreal$ |
| **val** | $intime\ (real) \rightarrow real$ |
| **turn** | $mpoint \rightarrow mreal$ |
| **velocity** | $mpoint \rightarrow mpoint$ |
| **present** | $moving(\alpha) \times periods \rightarrow bool$ |
| **atinstant** | $moving(\alpha) \times instant \rightarrow intime(\alpha)$ |
| **at** | $moving(\alpha) \times \beta \rightarrow moving(\alpha)$ |
| **atperiods** | $moving(\alpha) \times periods \rightarrow moving(\alpha)$ |
| **atmin** | $moving(\alpha) \rightarrow moving(\alpha)$ |
| **atmax** | $moving(\alpha) \rightarrow moving(\alpha)$ |
| **when** | $moving(\alpha) \times (\ \alpha \rightarrow bool\ ) \rightarrow moving(\alpha)$ |

object's movement is represented using a *sliced representation*, namely the temporal development is subdivided into fragments, i.e., slices, and within the slice the location is determined using an *interpolation function*. Concretely, a spatial trajectory is represented by a sequence of *units* while the interpolation function is commonly a linear function. Each unit has a form $(I, seg)$, where $I$ represents the time interval of maximum width during which the time-varying location falls along the segment $seg$. The spatial trajectory can thus be represented by the sequence:

$$(i_1, seg_1), (i_2, seg_2), .., (i_n, seg_n)$$

The units are temporally disjoint, temporally ordered, and two adjacent units have different segments. The Moving Object model has been implemented as part of SECONDO, an extensible database system that provides a rich set of data types to support non-standard database applications [28]. Another system relying on the Moving Object data model is the HERMES database [46]. In the following, we illustrate a few examples of queries over the spatial trajectories of the Geolife dataset.

**Example 2.1.** The spatial trajectories are stored in a database table that contains two attributes, the identifier of the moving object and the spatial trajectory of type mpoint. Five SQL queries are reported below:

```
Geolife(Id:string, Trace:mpoint)
```

$Q_{2.1}$: Select the time periods for which the spatial trajectories in Geolife are defined:

```
SELECT  deftime(Trace)
    FROM Geolife
```

$Q_{2.2}$: Select the ids of individuals whose spatial trajectories are longer than 5km:

```
SELECT  Id
    FROM Geolife
    WHERE length(trajectory(Trace)) > 5000
```

$Q_{2.3}$: Select the individuals that move with a velocity less than 20 Km/h:

```
SELECT  Id
    FROM Geolife
    WHERE var(initial(atmax(velocity(Trace)))) < 20000/3600
```

$Q_{2.4}$: Select the individuals that stay in a region R for more than 1 hour:

```
 SELECT  Id
    FROM Geolife
    WHERE duration(deftime(at(Trace,R))) > 3600
```

$Q_{2.5}$: Select all the pairs of individuals who during their trips came closer to each other than 100 meters:

```
SELECT  p.Id, q.Id,
    FROM Geolife p, Geolife q
    WHERE var(initial(atmin(distance(p.Trace,q.Trace)))) < 100
```

A major stream of research concerns the development of indexing techniques for the fast retrieval of trajectories and their parts. Most of the proposed indexes for accessing spatial trajectories are based on the R-tree index [32]. Because the R-tree is also at the basis of the *IRWI* index we now describe the main features of this access method.

### 2.1.2 The R-tree Index

The R-tree is a balanced tree used for the dynamic organization of *d dimensional* objects [40]. For simplicity, we instantiate the description of the R-tree on the case of spatial objects (2-dimensional objects). Spatial objects are represented in the R-tree through a minimum bounding box (MBB). The MBB unifies the representation of spatial objects, and simplifies that of complex spatial objects. However, during the query processing, the MBB representation introduces an extra cost due to the need of filtering out the spatial objects that do not satisfy the query, despite their MBBs. In the R-tree, every node occupies one disk page and every non-root node contains a number of entries in the range $[m, M]$, where $m \leq M/2$ with $M$ depending on the disk page size.

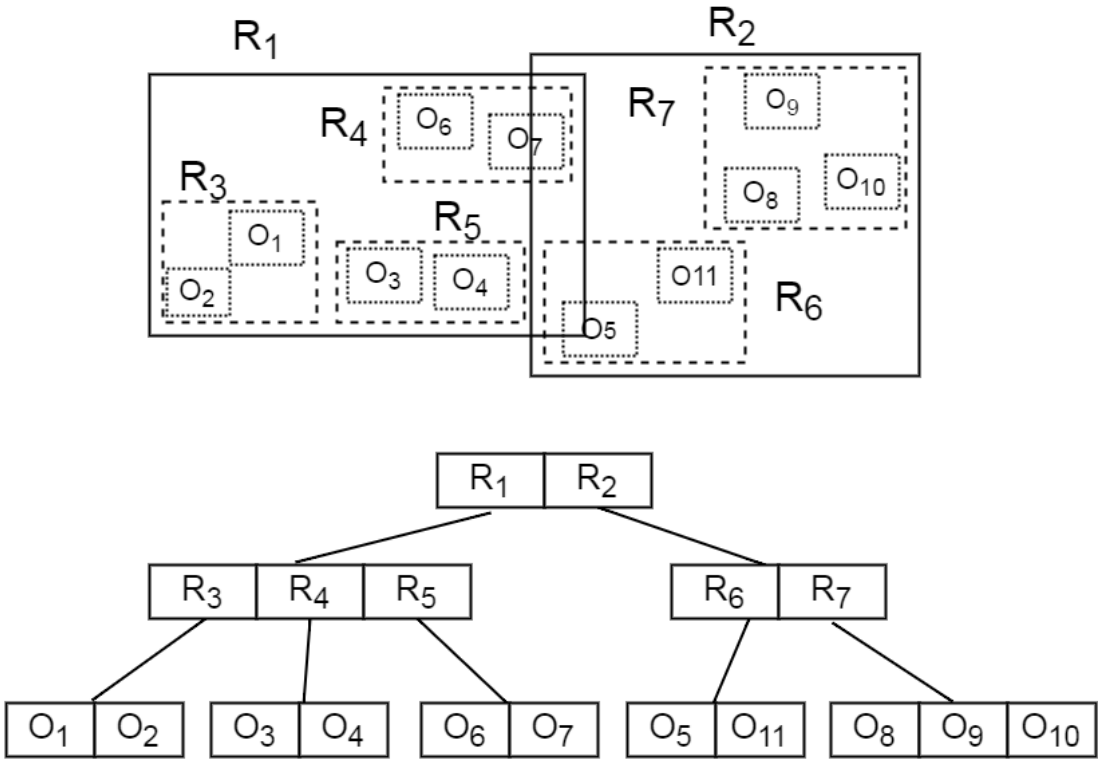



Figure 2.1: Example of R-tree

**Data structure.** Consider the R-tree shown in Figure 2.1. The tree consists of leaf and internal nodes. The objects (or pointers to objects) are inserted in the leaf nodes. The internal nodes split the spatial space covered by the objects into possibly overlapping regions while leaf nodes contain a set of leaf entries,

one for each object. More in detail: a) a leaf entry consists of a pair of logical pointers: a pointer to the object in the database, and the MBB of such object; b) An internal node contains a set of non-leaf entries, each containing a pointer to and the MBB of a child node. The MBB of the child node $n_i$ is the union of all the MBBs that are present in the entries of $n_i$.

**Search.** Consider a range query attempting to retrieve the objects contained in a given query-rectangle. The search process scans the R-tree starting from the root through the sub-trees whose MBB intersects the query rectangle until the leaf nodes are reached. It should be noted that minimizing the overlap between the MBBs of the internal nodes will reduce the number of candidate sub-trees that are to be scanned in the search process, thus making the processing more efficient. At the leaf level, the search process uses the logical pointers in the leaf entries to read the candidate spatial objects and filter out those that do not satisfy the range query. More details on the process will be given in Chapter 3 when we will introduce the *IRWI* index.

**Insertion.** For the insertion of a new spatial object, the R-tree is traversed starting from the root and searching for an appropriate leaf accommodating the new entry. At each level of the tree, a node is selected based on the *least enlargement criteria*, i.e., the selected node is the one whose corresponding MBB requires the least enlargement to enclose the MBB of the new object. If the selected leaf node can accommodate the new entry (i.e., contains less than the maximal allowed entries), the entry is then inserted and the ancestor nodes are updated accordingly. Otherwise, if the selected leaf node already contains the maximum number of entries, the node is split into two leaf nodes and the ancestor nodes are adjusted accordingly.

Splitting a node entails a re-distribution of its entries between two nodes. The objective of the splitting algorithms is to minimize the probability of visiting both nodes for the same query by minimizing the total area of the MBBs of both nodes. To that end, a straightforward approach is to compute all the possible groupings of entries and choose the one minimizing the MBB area. This solution is, however, time consuming therefore a trade-off between time complexity and minimization criteria satisfaction is needed. In this line a popular splitting algorithm is the *quadratic algorithm* presented by Guttman [32].

### 2.1.3 Access Methods based on R-tree

Numerous access methods based on R-tree have been proposed for the indexing of spatial trajectories [43, 40]. The basic technique employs two separated data

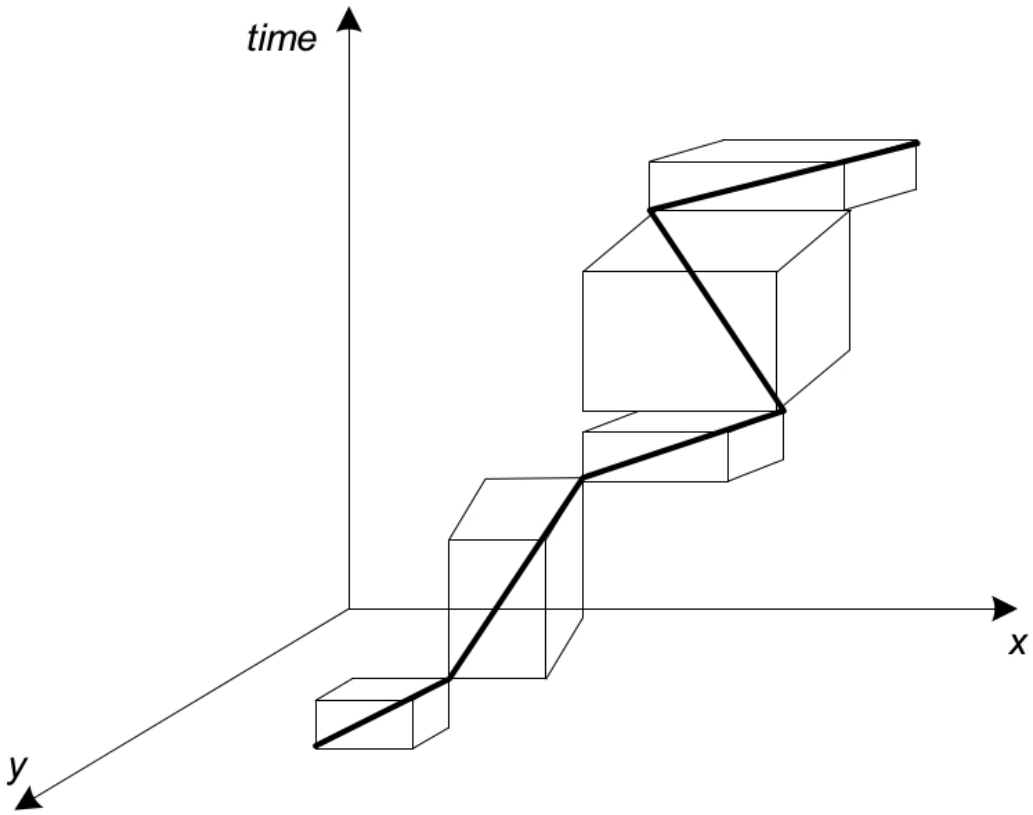



Figure 2.2: Discrete representation of spatial trajectory where every segment is bounded by the 3D MBB [40]

structures, a 2D R-tree and a 1D R-tree, the former for the indexing of the spatial component of the trajectories, i.e., the segments connecting consecutive points, and the latter for the indexing of their temporal component. The processing of a spatio-temporal range query consists of two steps. First, the two groups of candidate trajectories satisfying the spatial condition and the temporal condition, respectively, are selected. Next, the trajectories that are in common between these two groups are extracted.

A different technique employs a *3D R-tree* [55]. The idea is to use a unique data structure integrating both the spatial and temporal dimension. Accordingly every spatio-temporal segment of the trajectory is bounded by a 3D MBB, as shown in Figure 2.2, and inserted into the three-dimensional R-tree. The experiments show that, for spatio-temporal range queries, this index is more efficient than the basic access method while it is not equally performant for time-slice queries where the efficiency depends on the total number of entries in the history.

*Historical R-tree* and *HR-tree*, [42] try to overcome these limitations. In particular, the Historical R-tree creates an R-tree for each timestamp. The index data structure keeps a vector of pointers, each one pointing to one of these R-trees. In place of a complete R-tree, the Historical R-tree creates an R-tree exclusively for the objects whose location has changed since the last timestamp, maintaining as well a pointer to the unchanged nodes in the preceding R-tree. Given a time-sliced query, the search process scans the R-tree that contains the trajectories for the specific time slice. The drawback of the technique is that it requires the scanning of multiple R-trees in case of range queries. To overcome these limitations, Tao and Papadias propose the *Multi-version 3D R-tree* [54] which combines *Multi-version B-trees* (MVR tree) and 3D R-trees. The main idea is to build two trees, an MVR tree to process time-slice queries, and a 3D R-tree to process efficiently range queries. Other indexes have been proposed for the

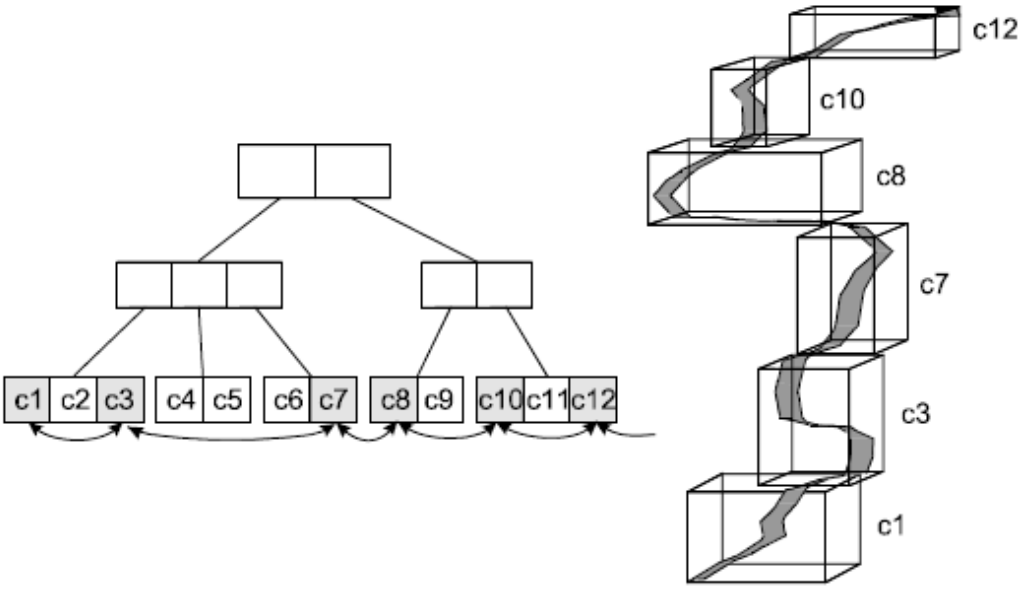



Figure 2.3: An example of a trajectory stored in a TB-tree [40].

efficient processing of topological queries. Pfoser et al. [47] propose the *STR-tree* and *TB-tree*. The STR-tree organizes the line segments forming a trajectory not only based on spatial properties, but also trying to group segments belonging to the same trajectory. By contrast, the TB-tree forces the line segments belonging to the same trajectory, to be stored in the same leaf node, ignoring thus their spatial proximity. To enable the fast access to the whole trajectory, leaf nodes belonging to the same trajectory are linked by means of a doubly linked list. The main problem with this technique is the dead space in each MBB that may

lead to the degradation of both update and query efficiency [40]. In general, the TB-tree performs better than the other access methods on queries involving the history while for queries involving the objects' coordinates, the performance of the TB-tree is worse than that of the 3D R-tree.

## 2.2 Semantic Trajectories

Representing the objects' movement in terms of spatial trajectories is useful when the goal is to locate objects in space and time or compute statistics on the spatio-temporal characteristics of trajectories, e.g., speed [45]. However, in modern applications, there is an increasing body of evidence that describing the movement exclusively in spatial terms is no longer sufficient [19]. For example, an important feature of the movement is the context in which such a movement takes place, such as the weather conditions during a travel, the places of interest being visited, the people in proximity and so forth. All that motivates the growing concern for solutions advancing the classical Moving Object data model.

*Semantic trajectories* is the term coined to indicate those data models that target the representation of knowledge about the movement of entities. In the recent years, the research on semantic trajectories has experienced a significant growth as demonstrated with a variety of research works, especially those carried out in the projects GeoPKDD in Europe [27] and Geolife in Asia [67]. In the sequel, we overview related works focusing on the aspects of conceptual modeling and semantic enrichment of trajectories.

### 2.2.1 Conceptual Models

The first conceptualization is known as *stop-and-move* model [45]. The basic idea is to describe the movement as a sequence of transitions from one stop to another where the stop indicates the temporary suspension of the movement at the chosen level of abstraction. For example, stops can describe the points of interest visited by tourists during their travels while the moves can indicate the use of transportation means. The concepts of *stop, move* have been next generalized in that of *episode* in [45]. Episodes result from the temporal partitioning of the movement history in semantically meaningful segments. The semantics is expressed through *annotations*. Under this view, stops and moves are simply different kind of episodes. Accordingly, a semantic trajectory is defined as sequence of episodes, where each episode holds an annotation. Figure 2.4 represents a semantic trajectory describing a touristic trip. The semantic trajectory consists of a sequence of alternating stop and move episodes, where stops are annotated with a place label and the moves with the label of a transportation means. *CONSTAnT* is another

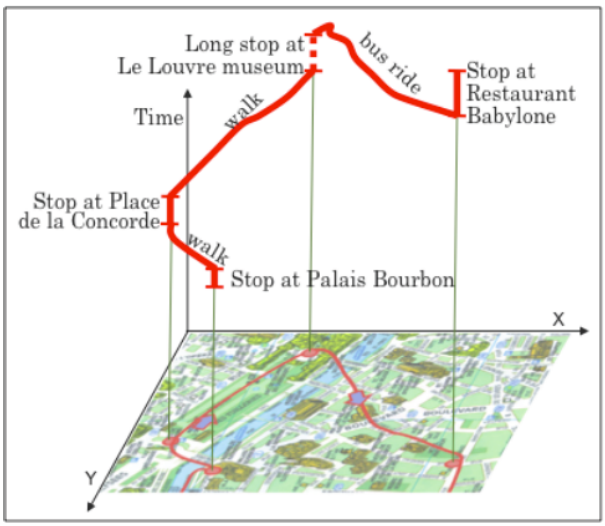



Figure 2.4: The semantic trajectory describing a touristic trip with annotated stop and move episodes [45]

conceptual model that extends the *stop-and-move* framework integrating several semantic dimensions for movement analysis (e.g., goal, behavior) [8]. In *CONSTAnT*, the movement is described as a sequence of semantic sub-trajectories, where each semantic sub-trajectory has multiple properties.

None of these models has been translated into a database system. We will come back to this point later on. Moreover none of these techniques specifies how to detect the episodes. Indeed, this problem is addressed by a different class of techniques also known as *semantic enrichment* techniques, briefly presented next.

### 2.2.2 Semantic Enrichment Frameworks

ST-DMQL [7] is proposed as semantic trajectory query language providing functionalities for trajectory processing and mining. The ST-DMQL query language provides an operator for generating stop and move episodes from spatial trajectories. The operator generates a stop episode when a certain constraint is satisfied, specifically if the spatial trajectory is inside one of the *candidate stops* for a minimum duration where the candidate stops are user-defined. A limitation of ST-DMQL is that it does not provide any other kind of episode besides these narrowly defined stop and move episodes.

The work of Yan et al. [61, 62] tries to overcome this limitation by defining a more general methodology. The authors propose a generic multi-layer framework called *SeMiTri* for the annotation of spatial trajectories at different levels of

abstraction. The idea is to first extract basic abstractions (e.g., stop, move) and then further characterize these episodes through the definition of higher-level abstractions. This methodology is applied as follows: during the annotation process, SeMiTri leverages contextual and geographical knowledge together with movement properties such as density, velocity and direction, to extract from the trajectory a sequence of stop and move episodes. Next, points of interest and activities are extracted from stop episodes while the information on the transportation means is extracted from the move episodes.

A more flexible and generic environment is provided by Baquara [25]. Baquara is an *ontology-based* framework for trajectory annotations where annotations can be selected from linked data. In general, an ontology describes taxonomies and classification networks, essentially defining the structure of knowledge for various domains. The ontology introduced in Baquara [25] defines concepts like events, places, moving objects, episodes and semantic trajectories. Driven by the ontology, an automated technique selects the appropriate linked data according to the spatio-temporal scope of the movement and the application domain (tourism, traffic, ecology).

Beyond the multiplicity of techniques, a question that remains unsolved is the scalability aspect, i.e., how to deal with large amounts of semantic trajectories. The issue poses several challenges [19]: it requires the definition of a rigorous data model, the specification of an effective query language enabling the retrieval of semantic trajectories or parts of them, and the development of efficient data access mechanisms. To deal with these aspects, two main directions have been explored: the first is presented below and implies the use of Semantic Web technologies, the second calls for novel database solutions and is discussed in the subsequent section.

### 2.2.3 Semantic Web Trajectory Databases

The early work in [63] starts from the stop-and-move conceptual model to define a *Semantic Trajectory Ontology* that is next stored onto a commercial database that has a reasoning extension, Oracle with OWL-Prime. Semantic Trajectory Ontology is a combination of three sub-ontologies: the *Trajectory Ontology* describes the trajectories in terms of stop and move, the *Geographic Ontology* describes the contextual information, and the *Application Domain Ontology* describes the application dependent concepts.

A different approach is presented in [36]. Instead of defining an application dependent ontology, the authors define the *Geo-ontology* as an ontology design pattern that serves as skeleton for the creation of complex ontologies. The Geo-ontology is application independent, while domain specific ontologies can be cre-

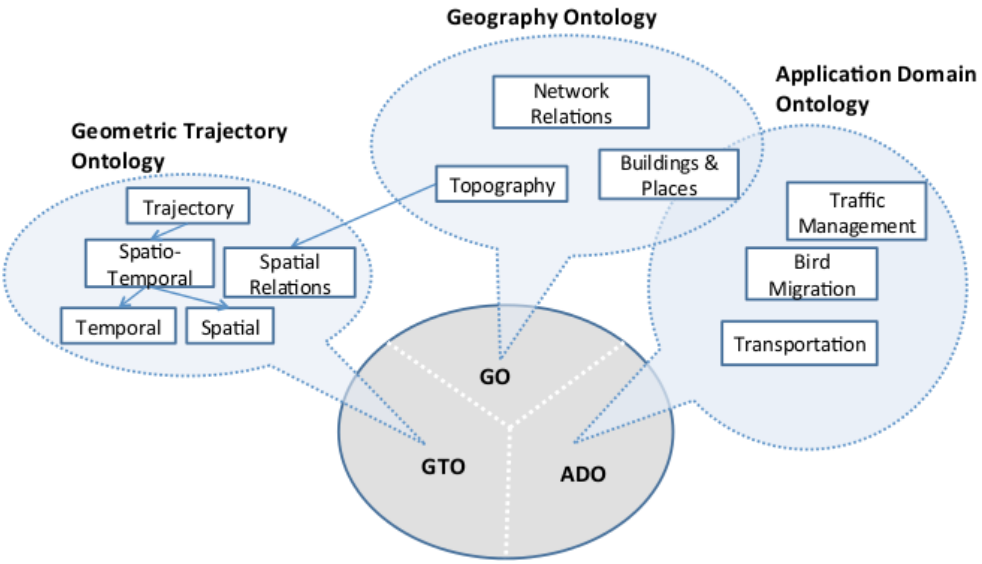



Figure 2.5: Semantic Trajectory Ontology [63]

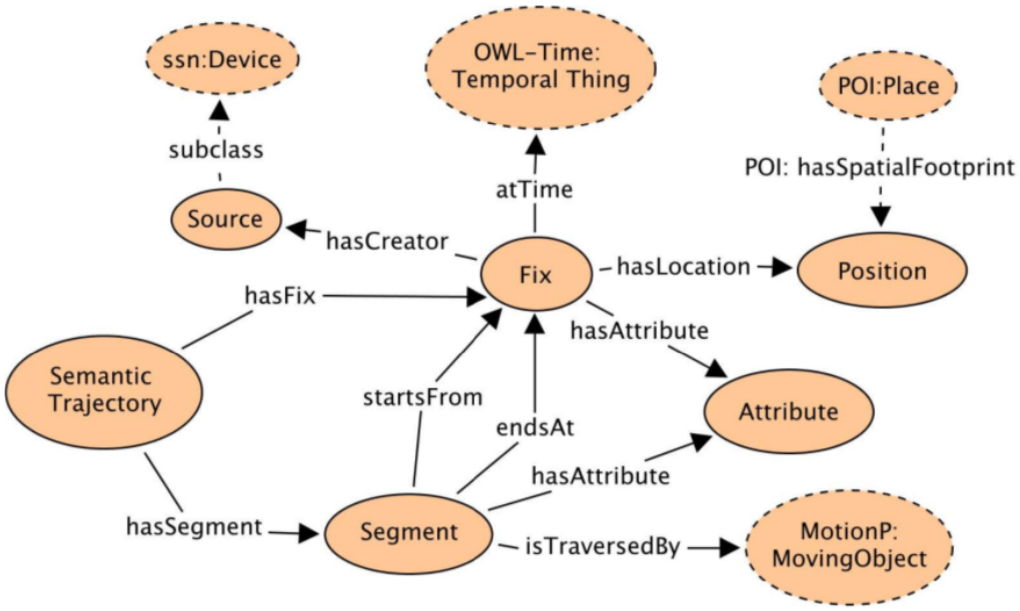



Figure 2.6: Geo-ontology [36]

ated by sub-typing (i.e., extending) the classes defined in the Geo-ontology. The Geo-ontology consists of basic classes (i.e., fix, segment, semantic trajectory) and a number of generic classes (Source, Position, Attribute) providing the application developer with the possibility of integrating geographic and application dependent knowledge into the final ontology. For example, the *fix* denotes the position of the moving object at a certain instant, the *segment* is defined by the starting and ending fixes, the *semantic trajectory* is a sequence of segments. The *Position* can represent any geographic feature of interest for the application (e.g., tourist attraction, schools). This concept can be extended for integrating any existing POI ontology. *Attribute* is another generic concept that connects fixes and segments to their attribute values, such as speed or the bearing of a segment. The model has been represented using the Web Ontology Language (OWL). Dealing with large amounts of trajectory data remains however an issue.

## 2.3 Symbolic Trajectories

A recent stream of research leverages database technology and the Moving Object data model, originally developed for the representation of spatial trajectories, to provide a system for the representation and querying of annotated trajectories stored in a database. A major result is the *Symbolic Trajectories* data model and its implementation [31].

In abstract terms, a symbolic trajectory is a time dependent name or *label* where the label is an element of an application-dependent vocabulary. Labels can represent for example the street of a city, the cell IDs of a cellular network, the different transportation modes (bus, metro, walk...) or even the speed profile of a movement (slow, medium, fast,...).

### 2.3.1 Trajectory Representation

In practice, the discrete model of symbolic trajectories consists of a sequence of units $u_1, .., u_n$ where $u_i = (I_i, l_i)$ is the generic unit defined by a time interval $I_i$ and label $l_i$. The interval $I_i$, in turn, is defined by the tuple $(start, end, lc, rc)$, specifying the beginning of the period ($start$), the end ($end$) and whether the interval is left or right closed respectively ($lc$, $rc$).

**Example 2.2.** A symbolic trajectory representing the sequence of transportation means used by a particular individual can be specified as follows:

```
([8:30-8:45), walk), ([8:45-9:13), train), ([9:13-9:19], walk)
```

This trajectory can describe, for example, the home-work travel of an individual. The individual walks for 15' from 8:30 until 8:45 before taking the train from where he/she gets off at 9:13. Next he/she walks again for a while to reach the destination.

The basic mechanism used for the representation of symbolic trajectories is the abstract data type. Accordingly, the symbolic trajectories are simply treated as values of a novel data type formally defined by a specific algebra encompassing a set of operations specific for the type. This new data type can be used to specify the attribute types in a database table, while SQL can be used to interrogate sets of symbolic trajectories. More in detail, the novel data type introduced for the representation of symbolic trajectories is called *mlabel* (moving label). A label is basically a string thus, literally, a moving label is a time-varying string. A pleasant feature is that the type *mlabel* inherits the generic operations defined for Moving Objects. For example, the operator **atinstant(o,t)**, that we have met in a previous section, returns the label of the symbolic trajectory $o$ at time $t$; **deftime(o)** returns the period in which the symbolic trajectory $o$ is defined.

**Example 2.3.** The sequences of transportation means used by the individuals in the Geolife dataset can be stored in a database table consisting of two attributes, one for the individual Id and the other for the symbolic trajectory (attribute Trans). Two basic examples of SQL queries are reported next:

```
Geolife(Id:string, Trans:mlabel)
```

$Q_{2.6}$: Find people who walk sometimes:

```
SELECT  Id
     FROM Geolife
     WHERE Trans passes "walk"
```

$Q_{2.7}$: Find walking time periods of geolife people:

```
SELECT  deftime(at(Trans,"walk"))
     FROM Geolife
     WHERE Trans passes "walk"
```

### 2.3.2 Pattern-based Query Language

A major feature of the symbolic trajectory data model is the pattern-based query language. Patterns can be expressed as regular expressions extended with supplementary conditions and variables. Queries can be performed using two different operators for matching and rewriting respectively. The two operators, **matches** and **rewrite**, have the following signatures:

$matches : mlabel \times pattern \rightarrow bool$ $rewrite : mlabel \times pattern \rightarrow set(mlabel)$

The operator *matches* returns true if the input symbolic trajectory matches the input pattern (otw false); *rewrite* returns the portions of symbolic trajectory matching the pattern. Operationally, the pattern matching algorithm loops over the units of a trajectory, updating the set of active states of a non-deterministic finite automaton (NFA). In the following we present a few examples of patterns and show how the queries can be specified.

**Example 2.4.** $Q_{2.8}$: Find all symbolic trajectories matching the pattern: walk at morning and evening of the same day. The corresponding pattern is:

```
X(morning,walk) * Y(evening,walk) // (Y.end-X.start)< 1 day
```

X and Y are variables bound to the first and last unit of the symbolic trajectory; the part on the right following // is a condition expressed over those variables specifying that the duration of the entire trip should be less than 1 day. The symbol $*$ is a wildcard matching any sequence of units.

The rewrite operation can be used for the extraction of parts of symbolic trajectories as well as for aggregating sub-sequences of units into higher-level concept. A few examples are reported next.

**Example 2.5.** $Q_{2.9}$: Find all symbolic trajectories matching the previous pattern and set the label of the output to home. The pattern is as follows:

```
X(morning,walk) * Y(evening,walk) // (Y.end-X.start)< 1 day
=> X Y // X.label := "Home", Y.label := "Home"
```

The part on the right following the symbol $=>$ specifies that the output trajectory should only contain the units denoted by the variables $X, Y$ while the respective labels are to be replaced by 'Home'.

**Example 2.6.** $Q_{2.10}$: rewrite the symbolic trajectories that contain bus, metro or train into a symbolic trajectory that contains 'Public Transport.'

```
SELECT  rewrite(Trans," *  X(_,{bus,metro,train}) *
                => X // X.label =Public transport")
     FROM Geolife
```

$Q_{2.11}$: retrieve the trajectories of people continuously walking for at least 1 hour:

```
SELECT  Id
     FROM Geolife
     WHERE Trans matches " * X(_,walk) *
                          // X.duration > 1 hour "
```

### 2.3.3 Indexing Symbolic Trajectories

To enable the fast processing of the matching and rewriting operations, an access method for the indexing of spatial trajectories has been defined by Valdes and Güting [57]. The index consists of two separated components: a trie, for the indexing of labels, and a 1D R-tree, for the indexing of time intervals. A symbolic trajectory is indexed as follows: for every unit $i$, the label, the unit position and the trajectory ID are stored all together in the trie as represented in Figure 2.7. In contrast, the time interval with the unit position and trajectory ID are stored in the R-tree. The textual and the time components are thus kept on distinct data structures.

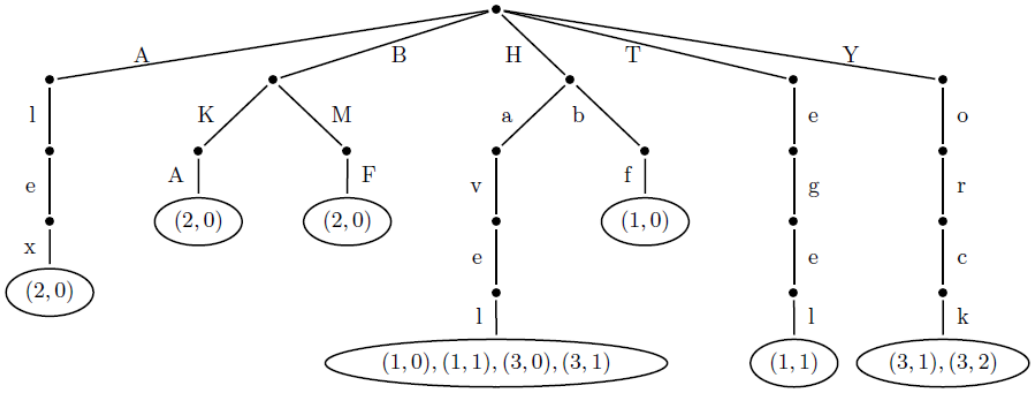



Figure 2.7: The trie used for the indexing of the symbolic trajectories labels. [57]

Given a query specifying a pattern represented by a sequence of atoms, the algorithm identifies the transitions that are mandatory for the automaton to reach any of the final states. The result is a set of candidate trajectories that are then evaluated for the exact matching. Compared to the linear scan, the computational cost of the index-based matching operation is reduced by more than an order of magnitude. The proposed index is implemented using the SECONDO platform.

## 2.4 Summary

In this section we have presented the three major paradigms for the representation of trajectories. The Moving Object data model is the reference model for the representation of spatial trajectories; the Semantic Trajectories paradigm includes different approaches for the semantic enrichment of trajectories; finally Symbolic Trajectories is the recent framework proposed for the handling of trajectories in textual form. In the next chapter we present the first major contribution

of the thesis, specifically a framework for the integration of spatial and symbolic trajectories.

# Chapter 3

# Modeling and Indexing Spatio-Textual Trajectories

## 3.1 Overview

Accessing large datasets of temporally aligned spatial and textual trajectories can provide valuable information on *where* certain behaviors take place. In this chapter we introduce the *spatio-textual trajectory* data model and a novel index framework, called *IRWI*, for the efficient processing of queries on aligned *spatio-textual* trajectories formulated as sequences of ordered spatio-textual range queries q = $q_1$,.., $q_n$ (*sequenced queries*). Our specific goal is not to propose an expressive data model somehow extending symbolic trajectories, but rather to focus on the efficiency of query processing. For that reason and also for the sake of generality, we prefer to use the term 'textual' in place of 'symbolic'. *IRWI* consists of a hybrid, i.e., spatial and textual, index data structure, enriched with a number of features that facilitate the early pruning of trajectories during the concurrent evaluation of the sub-queries $q_1$,.., $q_n$.

The chapter is organized as follows: Section 3.2 introduces the *spatio-textual trajectory* data model and the notion of *sequenced query*. Section 3.3 first discusses the baseline technique and next the *IRWI* solution. Sections 3.4 and 3.5, describe the index construction process and the query processing algorithm respectively. Experiments are reported in Section 3.6 where we contrast our approach with two methods, the first based on IR-tree [17], the latter on IF_R* [69], utilizing three different real and synthetic datasets.

## 3.2 Spatio-Textual Trajectories

### 3.2.1 Representation

At the *abstract* level, we define a *spatio-textual* trajectory as the function $tr(t)$ that for each instant of the temporal domain returns the pair $(l, p)$ where $p = f(t)$ is a point of space and $l = g(t)$ the label annotating this point. For example, the function $g(.)$ can return the transportation means used by the individual in time, while $f(.)$ is simply a time varying location. This abstract definition is mapped onto a *discrete* model. In practice, a spatio-textual trajectory is represented as sequence of units, e.g.:

$$u_i = (I_i, l_i, seg_i)$$

where the unit $u_i$ specifies that during the time interval $I_i$ the individual is located along the line segment $seg_i$, performing the activity labeled $l_i$. We recall from Chapter 2 that the discrete representation of a spatial trajectory consists of a sequence of segments. The spatial trajectory is thus annotated at the level of segment. Within the sequence, the time intervals of units are disjoint or adjacent and units are ordered by time intervals. In line with the Moving Object data model, time intervals are represented by the tuple $(s, e, lc, rc)$, where $s$ and $e$ are instants with $s < e$, and $lc$ and $rc$ boolean values denoting whether the interval is left-closed/open and right-closed/open.

Table 3.1 reports the sequence of units representing the trajectory in Figure 3.1. In this example, the labels indicate the transportation means used by the individual. The spatio-textual trajectory is visually displayed in Figure 3.1 as a spatial trajectory partitioned into a sequence of *colored sub-trajectories*, each indicating the transportation mode in the corresponding time period. In the

Table 3.1: The sequence of units of the *spatio-textual trajectory*

([7:00 7:30) **run** ($p_0$ $p_1$))
([7:30 8:00) **bike** ($p_1$ $p_2$))
([8:00 10:00) **metro** ($p_2$ $p_3$))
([10:00 10:10) **walk** ($p_3$ $p_4$)
([10:10 10:40] **run** ($p_4$ $p_5$))

rest of this chapter, unless specified otherwise, we use the term *trajectories* to refer to the unit-based representation of spatio-textual trajectories while a *sub-trajectory* is a sub-sequence of trajectory units. Back to our running example, note that the movement of the Geolife individuals can be compactly represented by spatio-textual trajectories.

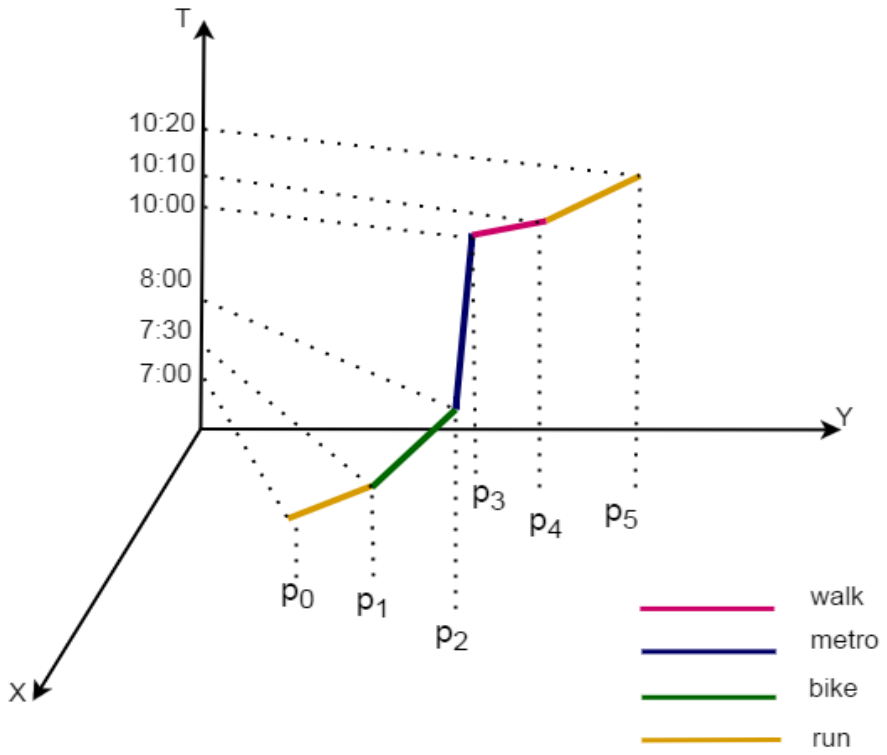



Figure 3.1: *Colored trajectory*: a simple graphical representation of a *spatio-textual* trajectory. The polyline displays a spatial trajectory, while the different symbology, e.g., color, highlights the time-varying categorical attribute, in this case the transportation mode

### 3.2.2 Sequenced Queries

We now turn our attention to the problem of interrogating a large dataset of spatio-textual trajectories through queries specifying non-trivial conditions on the textual, spatial and temporal components. We start with a discussion related to the following query over the Geolife dataset:

$Q_{3.1}$: *Find the individuals living in region $R_1$ that in the morning run to reach a station and then take some public transportation means to reach the workplace located in region $R_2$ after around one hour, between 9:00 and 10:00.*

The query $Q_{3.1}$ specifies two kinds of conditions: the first is on the textual dimension (i.e., the user runs then takes a public transportation means) and the second on the spatial-temporal dimension (i.e., morning, containment in region $R_1$ and in region $R_2$). Abstractly, this query belongs to a set of more general

queries that we call *sequenced queries*. Formally, we define a sequenced query as the sequence:

$$q = q_1, .., q_n \tag{3.1}$$

of *simple queries* with $q_i = (I_i, L_i, R_i)$ specifying an interval $I_i$, a set of $k$ labels $L_i = \{l_j, .., l_k\}$ and a rectangular region $R_i$. A trajectory $tr$ satisfies the *simple query* $q_i$ if the corresponding individual is located in region $R_i$ at some instant during the interval $I_i$ performing one of the activities in $L_i$. Rephrased in more formal terms, if it exists $t \in I_i$ such that $tr(t) = (l, p)$ with $l \in L_i$ and $p \in R_i$.

**Example 3.1.** The *simple query* asking for the individuals running in the early morning in region $R_1$ can be formulated as follows:

$$([7{:}00\ 8{:}00]\ \{\text{run}\}\ R_1))$$

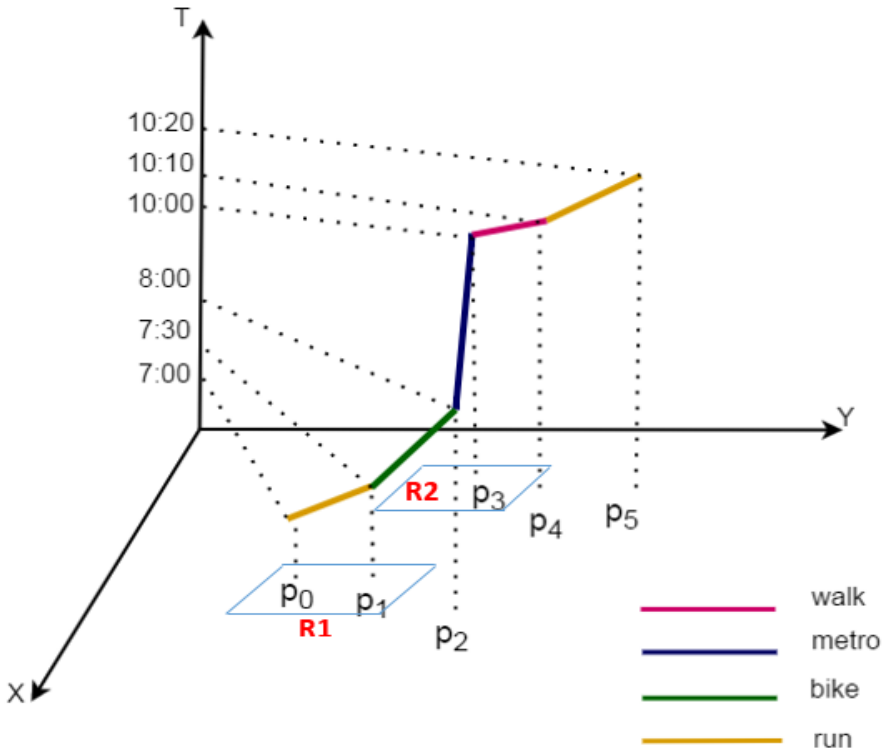



Figure 3.2: The rectangles $R_1$ and $R_2$ represent the spatial conditions of the sequenced query $Q_{3.1}$. The trajectory satisfies the query

This *simple query* is satisfied by the example trajectory in Figure 3.2 because the individual is running in $R_1$ at some time during [7:00 8:00].

For the sake of completeness, the notation used for the queries includes the symbol '_', used as placeholder for either the temporal or the textual conditions to mean 'any value'. Armed with this semantics, we say that the trajectory satisfies the sequenced query $q = q_1, .., q_n$ if there exist $n$ instants $t_1, .., t_n$ with $t_i < t_{i+1}$ such that for every instant $t_i$, $tr(t_i)$ satisfies the *simple query* $q_i$. The simple queries are thus to be solved in with respect to a given ordering, defined by the sequence in the query syntax.

**Example 3.2.** Back to the running example, Figure 3.2 shows two rectangles representing the spatial constraints informally specified in query $Q_{3.1}$. The full query for $Q_{3.1}$ consists of two simple queries expressed as follows:

([7:00 8:00] {run} $R_1$) ([9:00 10:00] {bus, train, metro} $R_2$)

It can be seen that the trajectory in Figure 3.2 satisfies the query because it intersects the rectangles in the right order, and moreover when inside $R_1$ the individual is running, next when inside $R_2$, he is using a public transportation.

Consider now a set of spatio-textual trajectories and a sequenced query. In order to retrieve the trajectories satisfying the query, a naive approach is to split the request into two pipelined sub-queries. The first sub-query is formulated over the textual trajectories to retrieve those matching the textual conditions, and the second over the spatial trajectories to retrieve those matching the spatial predicates in the periods in which the textual conditions are satisfied. This naive processing is shown to be extremely inefficient [21]. In the light of this experience, a challenging problem is to devise an appropriate index for the efficient processing of sequenced queries. This motivates the proposal of the IRWI technique. Before proceeding, we briefly overview related approaches in literature.

### 3.2.3 Related Approaches

Major streams of related research on query processing include pattern-based query languages over trajectories and spatial keywords-based queries, discussed in the next.

**Pattern-based query languages over trajectories.** Queries taking the form of sequential expressions over trajectories are typically supported by the so-called pattern-based languages. Pattern-based languages can be broadly classified into three classes: the languages that are only capable of expressing sequences of spatio-temporal predicates, e.g., [33, 49, 37]; the languages expressing queries as sequences of symbolic expressions, in the simplest case taking the form of

regular expressions, e.g., [31]; and the hybrid query languages combining spatio-temporal and symbolic expressions. Our model of sequenced query falls in the latter category. There exists a substantial difference, however, with respect to the state of the art, i.e., [58, 41], in that, in our model, the textual component describes a generic behavior or activity, thus it does not necessarily represent a spatial object, such as a region or a road, as in the aforementioned approaches. In this sense our notion of spatio-textual trajectory is more general and flexible.

**Spatial keywords-based query processing.** This class includes two categories of techniques, targeting the retrieval of static spatio-textual objects, e.g., [69, 56, 35, 38, 16, 17], and the retrieval of trajectories including both spatio-temporal and textual information, e.g., [34, 66], respectively.

The first category of techniques employs both textual and spatial criteria for the retrieval of, e.g., POIs [15]. The prevalent approach is to use a spatial index together with a textual index, the former typically based on R-tree [69, 35, 38, 17], grid or space filling curve [56, 16], the latter on *inverted files* and *bitmaps*. A major technique of this class is the *IR-tree* index. The IR-tree is essentially an R-tree, in which every internal node contains a pointer to an inverted file listing the spatio-textual objects contained in the sub-tree of the node [17]. However, these techniques are only applied to static objects and not to trajectories. The second category of techniques is more recent. A representative index is presented in [34]. This is an index that associates each keyword with an octree that partitions the space and time. In this case, however, the queries are simple range queries and not sequenced queries, as in our case.

In general, none of the above techniques is applied to trajectories while simply adapting one of the mechanisms in use for the processing of static spatio-textual objects does not ensure an adequate efficiency. Complementary to this, we note that trajectory indexing techniques are not designed to efficiently handle sequenced queries. All these observations motivate the study of a more efficient query processing strategy.

## 3.3 Accessing Spatio-Textual Trajectories: the IRWI Index

In this Section we present two techniques for the indexing of spatio-textual trajectories. We start with discussing a baseline technique, followed with a detailed presentation of our solution. To ease the presentation, we use as a running example the toy dataset of 4 trajectories represented in Figure 3.3. For the sake of

readability, the figure only displays the projection on the space of the trajectories, visualized as colored trajectories.

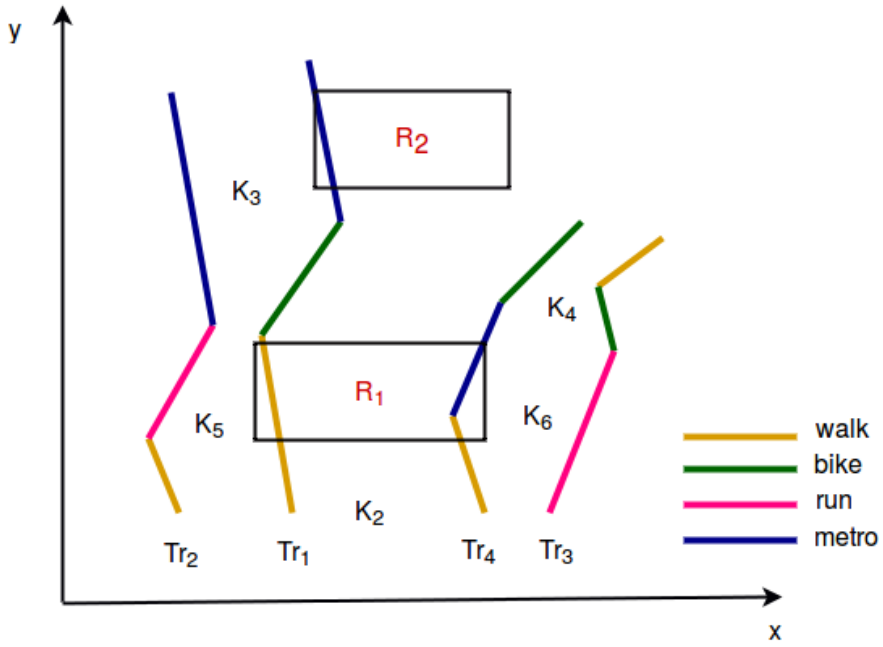



Figure 3.3: Running example: four colored trajectories projected on the 2D space (time is omitted). Each segment is associated with a different label, i.e., color. $R_1$ and $R_2$ are two regions of space.

### 3.3.1 Baseline Index

Consider a trajectory $u_1, .., u_n$ of length $n$. The idea is to use two separate data structures, specifically an inverted list for the indexing of the textual component of the units and a 3-dimensional R-tree for indexing the spatio-temporal part. A sequenced query is then processed *sequentially* solving each simple query of the sequence in the given order and, for each simple query, evaluating separately the textual and spatio-temporal conditions.

The inverted list is structured as follows. Each label is associated with a *posting list*. In particular, the posting list for label $l$ is a set of unit identifiers, each describing the position of one unit containing $l$. The unit identifier is the pair *(tid, num)* indicating the trajectory identifier and the position of the unit inside the trajectory, respectively. The three-dimensional *R-tree* stores in the leaf nodes an entry $(tid, num, I, seg)$ for each of the units specifying the unit identifier, the time interval and the segment.

Figure 3.4 shows the posting lists of the inverted file created for the trajectories of the example as well as the 2D representation of the R-tree for such trajectories.

| Label | Posting list |
|---|---|
| *walk* | (1,1)(2,1)(3,3)(4,1) |
| *bike* | (1,2)(3,2)(4,3) |
| *run* | (2,2)(3,1) |
| *metro* | (1,3)(2,3)(4,2) |

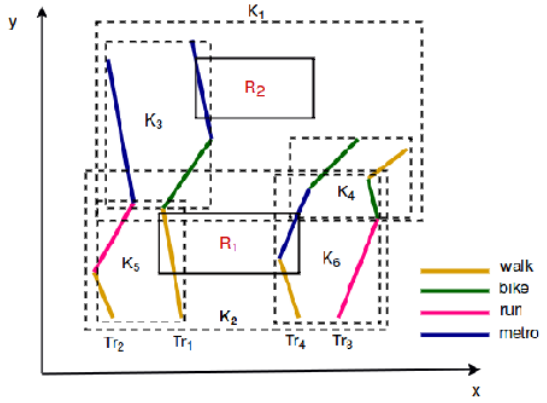



Figure 3.4: Baseline technique: *inverted list* and R-tree for the running example

The algorithm for processing the sequenced query $q =< q_1, .., q_n >$ comprises four steps:

- Step 1: For each query $q_i = (I_i, L_i, R_i)$ fetch from the inverted file the posting list of every label $l \in L_i$. This will result in the set $A_i$ of unit identifiers
- Step 2: For each query $q_i =(T_i, L_i, R_i)$ extract from the R-tree the set $B_i$ of unit identifiers for which the bounding box of the segment in the unit intersects the space-time box corresponding to $I_i$ and $R_i$
- Step 3: For each query $q_i =(T_i, L_i, R_i)$ compute the intersection $C_i = A_i \bigcap B_i$. This will result into a new set $C_i$ identifying the units satisfying both the textual and spatio-temporal conditions.
- Step 4: Given the sets $C_1, .., C_n$, determine for every candidate trajectory the time periods in which such trajectories satisfy the simple queries $q_1, .., q_n$. Next select only those trajectories that satisfy the simple queries in the appropriate order.

This technique presents two major shortcomings. Firstly, the unit content is split into two separate index data structures, therefore the textual and spatio-temporal conditions are to be evaluated separately. This would be similar to attempting to process a range query over a set of 2D points by providing separate binary index-structures over each coordinate. Such approach can generate significant overhead of false positives, which was one of the main motivations for the researchers to

investigate different 2D index structures [5, 32]. Secondly, every simple query of the sequenced query is evaluated independently from the other queries of the sequence. That hampers the pruning of the search space, moreover the R-tree has to be traversed every time a simple query is to be evaluated, that is, for a sequenced query of length $n$, the R-tree has to be visited $n$ times.

We speculate that a better performance can be obtained through a more integrated solution. That motivates the IRWI index presented next.

### 3.3.2 The IRWI Index

The key features of the IRWI index can be summarized as follows:

- The units $u_1, .., u_n$ are stored in the leaf nodes of a 3D R-tree while the internal nodes of the 3D R-tree summarize both the spatio-temporal and the textual information on the descending leaf nodes. In particular the labels in the leaf nodes are indexed using an inverted list. Thus the index consists of a unique, hybrid data structure

- Techniques are put in place to facilitate the early pruning of the irrelevant branches during the traversal of the tree. In particular, for every label, the identifiers of the trajectories containing such a label are explicitly stored in the data structure, using a compressed representation. The ultimate purpose is to support the *concurrent* processing of sequenced queries (as opposed to the sequential processing)

- The construction of the index is driven by two criteria: spatial proximity of line segments and textual homogeneity of labels. These criteria can be blended together through the specification of a spatio-textual cost function.

The full data structure is kept on disk. Every node of the tree is stored in a disk page of a system-defined size. Similarly, the inverted list is stored in a set of contiguous pages.

The index data structure is detailed below while the construction of the index and the search operation will be illustrated in the next sections.

**The data structure.** Figure 3.5 shows the IRWI-tree instantiated on the running example. The nodes have the following structure:

- The leaf nodes are vectors $e =< e_1, .., e_m >$ of *leaf entries* containing the spatio-textual units. Each entry consists of the pair: $e_i$=*(*$uid_i$*, *$unit_i$*)* where $uid_i$ is the unit identifier, i.e., *(tid, num)*, and $unit_i$ the spatio-textual unit, i.e., *(I, l, seg)*. The number of entries in a leaf node varies in the range

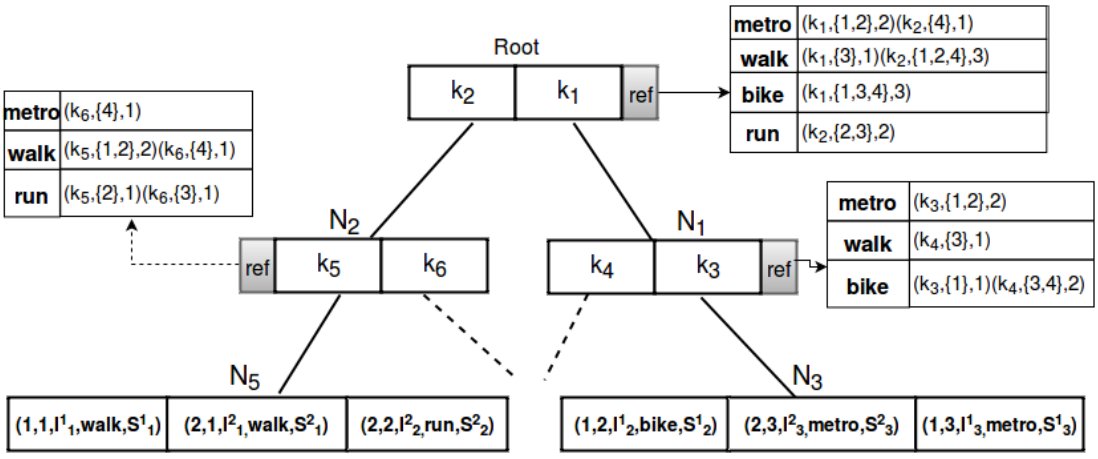



Figure 3.5: The IRWI tree: every non-leaf node contains a pointer to an inverted list reporting, for each label and sub-tree, the set of trajectories *Ids* and the number of units containing the label in the leaf nodes of the sub-tree. The leaf entries consist of the tuples: *(tr, num, I, l, seg)*, indicating the unit identifier (i.e., tr, num), the time interval, the label and the segment, respectively

$[min_1,max_1]$, where $min_1$, $max_1$ are two system defined parameters with $min_1 \in (0, max_1/2]$.

- The internal nodes contain a summary of the spatio-textual units stored in the descendant leaf nodes. In particular, a node $n = (< k_1, .., k_m >, ref)$, consists of a vector of *non-leaf entries* along with the pointer *ref* to an inverted file for the labels of the units located in the leaf nodes descending from $n$. Each label $l$ is associated with a posting list, where every posting specifies, besides a non-leaf entry, the set of trajectory identifiers and the total number of units containing the label $l$ in such a sub-tree. The set of trajectory identifiers is especially critical to reduce the search space during the processing of a query as shown next. One of the labels is the system-defined symbol ('_') standing for 'any value'. The corresponding posting list indicates for every entry the number of units in the leaf nodes. The number of entries in a non leaf node varies in $[min_2,max_2]$, where $min_2$, $max_2$ are system defined with $min_2 \in (0, max_2/2]$

We note that the idea of augmenting the internal nodes of a particular tree with entries that are themselves part of an index along a "other homogeneous dimension" has been introduced in the so-called *Range trees* [6]. However, our work differs in the sense that: (a) the spatial and temporal dimensions are indexed

in a single structure based on R-tree; (b) the "other dimension" is textual and thus not homogeneous with the R-tree, hence a different structure is used to index the textual dimension.

**Summarizing trajectory Ids.** Trajectory identifiers play an important role. Their handling, however, is not trivial. To illustrate, consider a query containing $n$ distinct labels $l_1, .., l_n$. Necessary condition for a trajectory to satisfy the query is that it contains all of those labels. For every label $l_i$, we can retrieve the sets of trajectories containing such label from the inverted file associated with the root node. The trajectories satisfying the query are necessarily included in the intersection of all those sets. Consequently, the entries that do not contain any of the trajectories in the intersection set can be pruned at early stage and the search space be reduced. The problem with this approach, that potentially can be very effective, is that, with large datasets, the storage of trajectory identifiers can span multiple disk pages. Fetching the posting lists thus becomes costly, and that can compromise the effectiveness of the pruning strategy. To mitigate the problem, the sets of trajectory *Ids* are recorded using a compressed representation. Specifically, the range of natural numbers encoding the *trajectories Ids* are partitioned in a small number of intervals, each clustering a subset of trajectories *Ids*. In this way, we can store the set of intervals in place of the full set of values and keep the data on a single disk page. The intervals are encoded using the *variable byte encoding* technique [39].

**Algorithm 1** Ids summarization

1: **Input:** $I =< I_1, .., I_n >$, $\lambda$, each $I_i$=$[s_i, e_i]$
2: **Output:** set of intervals $Ids$
3: $i \leftarrow 0$, gaps $\leftarrow \emptyset$
4: **while** $i <$ n **do**
5: /* compute the gap between two consecutive intervals */
6: $g \leftarrow s_{i+1} - e_i - 1$
7: $Insert(gaps, (i, g))$
8: $i \leftarrow i + 1$
9: **end while**
10: /* compute $\lambda$ -1 indices that have the largest gaps */
11: $Index_{larggaps} \leftarrow Max(gaps, \lambda - 1)$
12: /* partition the input I based on the indices positions that have largest gaps*/
13: $Ids \leftarrow Partition(I, Index_{larggaps})$

The operation takes as an input an ordered set of intervals partitioning a range of values and returns a coarser partition consisting of $\lambda$ intervals - with

$\lambda$ a system-defined parameter - that minimizes the total size of the intervals. For example, given the set of intervals: {[2, 4 ][5, 7][20, 25][26, 27]} and $\lambda = 2$ the operation returns {[2, 7][20, 27]}. The interval summarization is performed during the construction of the index based on the chosen value of $\lambda$.

Algorithm 1 details the operation of trajectory summarization. Given a sequence of $n$ intervals and a summarization parameter $\lambda$, the summarization process consists of the following steps: (1) for every two consecutive intervals, we compute the gap. This results in a set of $n-1$ gaps; (2) among the $n-1$ gaps, we find the indexes of the largest $\lambda$-1 gaps; (3) finally, we partition the input intervals over the computed maximal gaps. The complexity of the Ids summarization technique is linear with respect to the number $n$ of intervals.

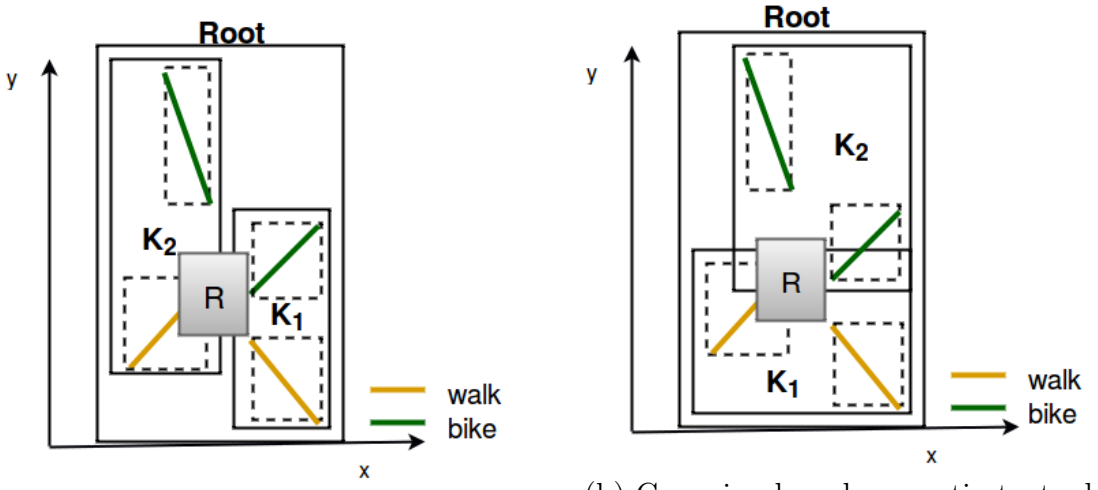
## 3.4 IRWI Construction

The building process of the IRWI index is similar to that of the R-tree but with a different grouping strategy. The idea is to supplement the geometric criterion, which drives the insertion of an object in a sub-tree of an R-tree, with another criterion that tries to keep the units sharing the same label as close as possible. The motivation is that the grouping of units containing the same label (i.e., homogeneous) allows a reduction in the number of disk accesses required to retrieve the units satisfying the textual conditions of the input query. The two criteria, i.e., spatial and textual, can however conflict because homogeneous labels can in reality be spread in space and time. An example can better illustrate the problem.

**Example 3.3.** Figure 3.6 illustrates 4 spatio-textual units that are differently grouped. The grouping in Figure 3.6 (a) is based on spatial proximity, that is the units that are spatially close to each other are also grouped together. The grouping in Figure 3.6 (b) is based on spatio-textual proximity, namely are grouped together the units with homogeneous labels that are close to each other. Now consider the following query that retrieves all of the individuals that pass through (i.e., intersect) a region R by bike:

```
(_, bike, R )
```

Depending on the grouping strategy, the number of I/O operations is different. In the first case (Figure 3.6 (a)) for processing the query the disk is accessed 3 times (i.e., for accessing the root, and the nodes $K_1$ and $K_2$); in the other case (Figure 3.6 (b)) the query only requires 2 accesses to disk (i.e., for accessing the root and $K_2$). Thus a trade-off between spatial and textual cost is needed and that motivates the investigation of the problem of generating a *hybrid cost* function.




(a) Grouping based on spatial cost

(b) Grouping based on spatio-textual cost

Figure 3.6: Dataset of 4 spatio-textual units with two different grouping strategies. $R$ is a region of space

### 3.4.1 Spatio-Textual Cost Function

We begin describing the cost function for the insertion of a unit, next we generalize the function to node splitting.

**Cost function for the insertion of a unit.** We define the normalized *geometric cost* $\mathcal{C}_G$, the *textual cost* $\mathcal{C}_T$ and finally the spatio-textual cost function $\mathcal{C}_{ST}$. Let $e = (tid, num, I, l, seg)$ be the leaf entry to insert and $n, k$ the current internal node and the entry in such a node, respectively.

- The *normalized geometric cost* function $\mathcal{C}_G(e, k)$ computes the cost of adding the segment $e.seg$ and the interval $e.I$ of the input leaf entry to the space-time minimum bounding box of the entry $k$. Such function is given by the ratio of the region enlargement and the maximum enlargement that can be achieved in the node, i.e.:

$$\mathcal{C}_G(e, k) = \frac{(Area(k.mbb') - Area(k.mbb))}{MaxEnl(e, n)}$$

  *MaxEnl(e,n)* is the maximum enlargement that can result from the insertion of $e$ in one of the entries $k_1, .., k_j$ of the current node; *mbb* and *mbb'* are the current and the enlarged minimum bounding box, respectively. In case there is no enlargement, the function returns 1. The cost varies in [0,1].

- The *textual cost* function $\mathcal{C}_T(e,k)$ is given by the percentage of units in the subtree pointed by $k$ which do not contain the label $e.l$, i.e.,

$$\mathcal{C}_T(e,k) = 1 - \frac{count(k, e.l)}{totalLabels(k)}$$

with *count(k, e.l)* the number of occurrences of the label $e.l$ in the subtree pointed by $k$ and *totalLabels(k)* the number of labels in the input entry $k$. It can be noticed that the cost is 0 if the entry contains only units with a label equal to $e.l$.

- *Cost function.* The spatio-textual cost function introduces the parameter $\beta$ to balance the contribution of the geometric and textual cost functions:

$$\mathcal{C}_{ST}(e,k) = \beta \times \mathcal{C}_G(e,k) + (1-\beta) \times \mathcal{C}_T(e,k)$$

$\beta \in (0,1]$ is the weight of the spatio-temporal component. Note that with $\beta = 1$, we obtain the cost function of R-tree.

**Generalization of the text cost function for node splitting.** If the insertion of a unit exceeds the capacity of the leaf node, a new leaf node is created and the leaf entries are split, while a new entry is added to the parent tree. The operation of node splitting is analogous to the one defined in the R-tree [32], except for the cost function. Thus, we limit the analysis to the text cost. The node splitting technique is the quadratic splitting [32]. Therefore the process consists of two steps: the 'pick seeds' step determines the pair of entries that are to be added for first in the two nodes. The best pair of seed entries are those that, if inserted in the same node, would maximize the cost. In the second step, the remaining entries are distributed in the two nodes. Let $e_i, e_j$ be a candidate pair of entries (where $e_i, e_j$ can represent either leaf or non-leaf entries) and $SL_{ij}$ the set of shared labels in the two entries. The generalized text cost function $\mathcal{C}_{\widehat{T}}(e_i, e_j)$ is defined as follow:

$$1 - \max_{l_k \in SL_{ij}} \frac{count(e_i, l_k) + count(e_j, l_k)}{totalLabels(e_i) + totalLabels(e_j)}$$

where the function *totalLabels()* returns the number of labels in the input entry (1 if the input entry is a leaf entry). Note that if the two entries do not share any label the cost is maximal (i.e., 1). It means that the labels in the respective subtrees are completely diverse and thus it is not worth keeping them together.

**Algorithm 2** SelectLeaf

1: **Input:** the leaf entry e=(tid, num, I, l, seg)
2: **Output:** a pair of a leaf node $N$ and the path from the root node to $N$
3: $N_c \leftarrow$ root node, Path $\leftarrow<>$
4: **while** $N_c$ is not a leaf node **do**
5: $Push(Path, N_c)$
6: /* returns the entry of $N_c$ with minimum cost */
7: $k_{min} \leftarrow \min_{k_i \in N_c} \mathcal{C}_{ST}(e, k_i)$
8: $N_c \leftarrow k_{min}.ref$
9: **end while**
10: $return(N_c, Path)$

### 3.4.2 Operations

In the following, we present the operations of insertion, removal and update of the entry *e=(tid, num, I, l, seg)* into IRWI, and the corresponding time complexity. We introduce the notations in Table 3.2. We note that the cost of the operations

Table 3.2: List of Notations

| Notations | Explanations |
|---|---|
| $N$ | Number of trajectory units |
| $L$ | Number of distinct labels |
| $T_{I/O}$ | Time needed to read a disk page |
| $k$ | Maximum length of inverted lists, |
| $min_1$ | Minimum # entries in leaf node |
| $min_2$ | Minimum # entries in non-leaf node |
| $max_1$ | Maximum # entries in leaf node |
| $max_2$ | Minimum # entries in non-leaf node |

is defined with respect to $N$ and $L$ whilst the other parameters are treated as constants.

**Insert.** Algorithm 3 illustrates the operation that inserts a leaf entry. Similarly to *R-tree*, the insertion starts by selecting the appropriate leaf node in which to insert the new leaf entry. This step is performed by Algorithm 2: we first traverse the *IRWI* tree from root to leaf nodes, selecting at each level the sub-tree that yields the *minimal cost*. Next, we get the best leaf node along with its ancestors.

The input entry is then inserted into the selected leaf node, Next the bounding box, inverted file, and summarization of *ids* of every ancestor node are updated.

**Algorithm 3** Insert

1: **Input:** the irwi index *IRWI*, the leaf entry *e=(tid, num, I, l, seg)*
2: $(N_{leaf}, Path) \leftarrow IRWI.SelectLeaf(e)$
3: $InsertEntry(N_{leaf}, e)$
4: **if** $N_{leaf}$ *needs to be split* **then**
5: $\{N_1, N_2\} \leftarrow Split(N_{leaf})$
6: **if** $N_{leaf}$ *is root node* **then**
7: /*Creates an empty internal node*/
8: $R \leftarrow InitializeNode()$
9: /*Entry(N) returns a tuple of bounding box, inverted file and pointer to N*/
10: $InsertEntry(R, Entry(N_1))$
11: $InsertEntry(R, Entry(N_2))$
12: **else**
13: $N_{leaf} = N_1$
14: $AdjustBoundingBox(Path)$
15: $AdjustInverIds(Path)$
16: $InsertEntry(Parent(N_{leaf}), Entry(N_2))$ ▷ If necessary, split intermediate nodes, adjust bounding box, inverted file and *Ids*.
17: **end if**
18: **else**
19: $AdjustBoundingBox(Path)$
20: $AdjustInverIds(Path)$
21: **end if**

The time complexity of the operation varies across scenarios. We start by computing the cost for a simple insertion that does not involve any node splitting as a function of $N$ and $L$. Later on, we discuss the worst case scenario where all the ancestors of the selected leaf node overflow and therefore need to be split.

The cost of the simplest insertion is dominated by the selection of the appropriate leaf node, which entails: (1) reading every non-leaf node since the root of the tree; (2) for every non-leaf node, read the inverted list of the input entry's label and compute the best entry. The total time of the second subroutine is

$$T_1 = T_{I/O}(1 + k) + T_c(max_2)$$

where $T_c$ is the cost of computing hybrid distance between one node entry and the input entry. As the maximum height of the tree is given by $\log_{min_2}(N/min_1)$,

the highest cost of the insertion operation that does not involve any splitting of a node is:

$$\mathcal{O}(T_1 * \log_{min_2}(N/min_1)) = \mathcal{O}(\log N)$$

Consider now the situation requesting the splitting of some intermediate nodes. The node splitting operation entails: (1) reading the full inverted lists in order to compute the hybrid cost, the cost of this subroutine is linear to the number of distinct labels $\mathcal{O}(L)$; (2) splitting the node with the quadratic technique, having cost $\mathcal{O}(max_2^2 * L) = \mathcal{O}(L)$. Thus, the splitting cost of one node is $\mathcal{O}(L)$. However, the worst insertion case occurs when the leaf node and all its ancestors need to be split and the height of the tree is maximal. This results in $\log_{min_2}(N/min_1)$ splitting operations. Therefore, the worst case insertion cost is equal to $\mathcal{O}(L * \log N)$.

**Delete.** Similarly to the *R-tree*, the removal of a unit from the index starts by retrieving the leaf node that contains this unit. Once the leaf node is found, the corresponding entry is deleted and the appropriate inverted files and bounding boxes are updated accordingly. If after the removal, there is an underflow of the corresponding leaf node, the remaining entries of such node are reinserted in the *IRWI* tree and the node itself is deleted. The worst deletion case is when the whole tree needs to be scanned in order to find the unit to delete and the remaining entries are reinserted with the worst insertion cost, yielding the worst deletion cost $\mathcal{O}(N) + \mathcal{O}(L * \log N)$.

**Update.** Updating a unit in the index requires two steps. First, the corresponding entry from IRWI is removed, next the new unit is inserted. Thus, the worst update cost is equal to $\mathcal{O}(N) + \mathcal{O}(L * \log N)$.

## 3.5 Concurrent Query Processing

We now turn to consider how the IRWI data structure is accessed during the processing of a sequenced query $q = q_1, .., q_n$. We remind that a sequenced query retrieves the set of trajectories satisfying the *simple queries* in the appropriate order. Formally, the result of a query is:

$$\{tr | \exists\ t_1 < .. < t_n, tr(t_1) \in Eval(q_1) \wedge .. \wedge tr(t_n) \in Eval(q_n)\}$$

where *Eval($q_i$)* yields the trajectories satisfying the *single query* $q_i = (I_i, L_i, R_i)$.

The key feature of the query processing strategy is that the simple queries are not evaluated sequentially, one after the other, but rather *in parallel*, at the

same stage. As a result the tree is traversed only once. Moreover, the subtrees that are not relevant for the query can be pruned at an earlier stage.

**Algorithm 4** SequencedQueryProcessing (q)

**Input:** the query $q = q_1, .., q_n$ with each $q_i$ a simple query
**Output:** res=$\{tr_i\}$, set of trajectories that satisfy the query
$\forall i \in [1, n]$ timeWindow[i]$\leftarrow \emptyset$, ids[i] $\leftarrow \emptyset$, nodes[i] $\leftarrow$ {RootNode}, entries[i] $\leftarrow \emptyset$
clevel $\leftarrow 0$
/* main cycle: for every non-leaf level...*/
**while** clevel $<$ height of the tree **do**
/* step: select candidate entries */
    **for** $1 \leq i \leq n$ **do**
        entries [i] $\leftarrow$ matchingEntries(nodes[i], $q_i$)
        timeWindow[i] $\leftarrow$ minTimeInterval(entries [i])
        Ids[i] $\leftarrow$ set of trajectories *Ids* pointed by entries[i]
    **end for**
     /* step: sets refinement */
    trim(timeWindow) ▷ Refine the candidate time periods of each simple query
    sharedIds $\leftarrow \bigcap_{1 \leq i \leq n} Ids[i]$
    /* step: filter entries */
    **for** $1 \leq i \leq n$ **do**
        filterByTime(entries[i], TimeWindow[i])
        filterByIds(entries[i], sharedIds)
        nodes [i] $\leftarrow$ getNodes(entries[i])
        **if** (nodes [i] is empty ) **then**
            return
        **end if**
    **end for**
    clevel $\leftarrow$ clevel + 1
**end while**
/* grouping of units by trajectory and $q_i$, */
trunits[i] $\leftarrow$ groupByTraj(nodes[i], $q_i$), $\forall i \in [1, n]$,
res $\leftarrow$ checkOrdering(Trunits)

The main steps of the process are as follows. The algorithm traverses the tree starting from the root node. At each level of the tree, the candidate entries for every simple query $q_i$ are first determined. A candidate entry $k$ is an entry

which satisfies $q_i = (I_i, L_i, R_i)$, namely: the space-time box defined by $R_i$ and $I_i$ overlaps $k.mbb$; and the set of labels contained in the descendant leaf nodes intersects $L_i$. Next the algorithm filters out those candidate entries that cannot contain trajectories satisfying all of the simple queries. The nodes referenced by the remaining entries are those traversed in the next step. Hence, the algorithm proceeds iteratively evaluating the query $q$ at every non-leaf level of the tree. As the leaf nodes are reached, the candidate units satisfying $q_i$ are grouped by trajectory id. The resulting trajectories are filtered out to extract only those satisfying the query $q$ in the correct order.

### 3.5.1 The Algorithm in Detail

The process is detailed in Algorithm 4. The array *nodes* stores for each $q_i$ the candidate nodes in the current level (variable *clevel*), namely the nodes that might point to units satisfying $q_1, .., q_n$. The candidate nodes are specific for each level of the tree. This array is initialized with the singleton containing the root node. At line 7, the algorithm starts scanning the tree, level by level, evaluating all of the simple queries in parallel. The progressive refinement of the search space is achieved in four main steps as follows.

- The first step is to compute the candidate entries. For each query $q_i = (I_i, L_i, R_i)$, *matchingEntries* accesses the inverted list associated with the labels in $L_i$ to find the entries pointing to the trajectory units containing such labels. Next, the subset of entries whose *mbb* does not overlap the bounding box formed by $(I_i, R_i)$ are filtered out. The remaining entries are recorded in the variable *entries[i]*. For each of the sets *entries[i],.., entries[n]*, the algorithm determines the minimum time interval, that is the time window containing all the units satisfying query $q_i$ (variable *timeWindow[i]*). Moreover the sets of trajectories identifiers pointed by the entries in *entries[i]* are retrieved from the inverted list and their union set stored in the variable *Ids[i]*. Ids[i] identifies thus a superset of the trajectories satisfying query $q_i$.

- In the second step, the time windows and the superset of trajectories are refined to account for the fact that the simple queries are all to be satisfied in the right order. As concerns the time ordering, assume that at the current level, the query $q_1$ can be solved in the interval $[s_1,e_1]$ starting at time $s_1$ and ending at time $e_1$, while the query $q_2$ in the interval $[s_2,e_2]$. If $q_1$ precedes $q_2$ in the ordering and $s_2 < s_1$, the candidate units of $q_2$ falling in the interval $[s_2,s_1]$ cannot be part of the final result. The algorithm thus checks whether consecutive time windows, i.e., timeWindow[i] and timeWindow[i+1], are

sound with respect to the temporal ordering. Moreover, if that is possible, the time windows are contracted. In the example the interval $[s_1,e_1]$ is trimmed to $[s_1,\min(e_1,e_2)]$ and $[s_2,e_2]$ to $[\max(s_1,s_2),e_2]$.

On the other side, the refined superset of trajectories containing units satisfying every simple query results from the intersection of *Ids[1],...,Ids[n]*. This operation can drastically reduce the number of candidate entries.

- In the third step the non-relevant entries are filtered out. For each set in $\{entries[i]\}_{i\in[1,n]}$ the algorithm detects and prunes those entries of the set that either do not intersect the corresponding time window, or do not point to any trajectory belonging to the intersection $\bigcap_{i\in[1,n]} Ids[i]$.

Finally, as the leaf nodes of the tree are reached, the trajectory identifiers and the units satisfying every query $q_i$ are returned. A final check is thus performed to extract the trajectories which contain properly ordered units.

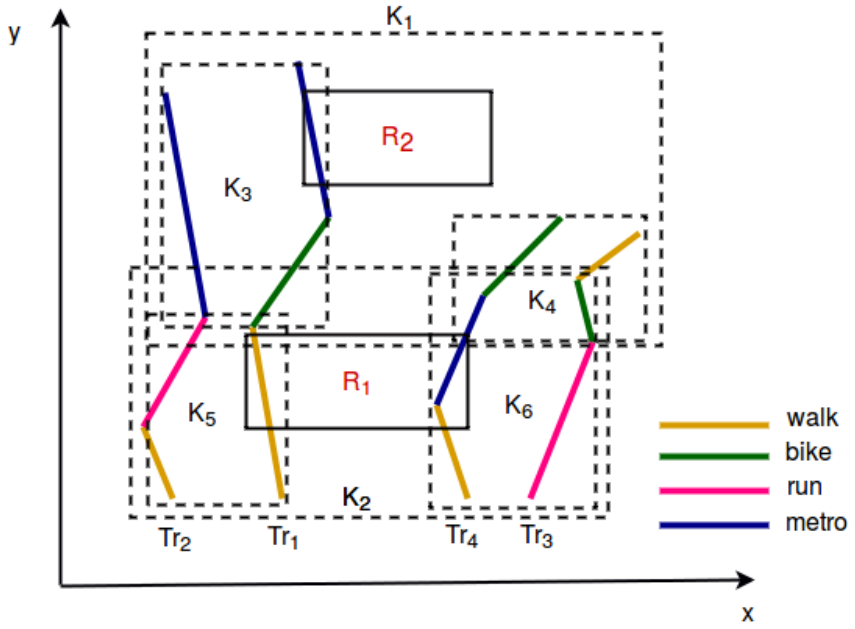



Figure 3.7: Spatial projection of bounding boxes of the *IRWI* entries represented in figure 3.5. $R_1$, $R_2$ are two spatial regions

### 3.5.2 The IRWI Index at Work

We continue using the running example in Figure 3.3 to illustrate the process step-by-step. The generated *IRWI* index is represented in Figure 3.5 and the

spatial projection of the appropriate bounding boxes is represented in Figure 3.7. We recall the query:

$$q_1 = ([7:00 \quad 8:00]\{walk\}R_1)\ q_2 = ([9:00 \quad 10:00]\{bus, train, metro\}R_2)$$

Notation: a candidate entry for the two queries is represented by tuple *(entry, int, trs)* where *entry* is the entry id, *int* the time interval of the entry, and *trs* the set of trajectories containing the labels associated with the entry. Initially, the candidate entries for the two queries $q_1, q_2$ are:

$E_{q_1}$={(Root,$I_{Root}$,{1,2,3,4})}
$E_{q_2}$={(Root,$I_{Root}$,{1,2,3,4})}

The steps of the different levels are described next:

- **Level 0**. The entry $k_2$ of the root node contains units labelled 'walk'. Moreover its spatial minimum bounding box overlaps $R_1$. The entry $k_1$ contains units labelled 'metro'( a public transport) and the spatial bounding box overlaps $R_2$

  $E_{q_1}$={($k_2$,$I_{k_2}$,{1,2,4})}

  $E_{q_2}$={($k_1$,$I_{k_1}$,{1,3,4})}

  Note that $\{1,2,4\} \cap \{1,3,4\} \neq \varnothing$ which means that the child nodes pointed by $k_2$ and $k_1$ are the candidate nodes in the next step.

- **Level 1**. The entries $k_5$ and $k_6$ in the node pointed by $k_2$ contain units labelled 'walk', moreover in both cases the spatial bounding boxes intersect $R_1$. The entry $k_3$ in the node pointed by $k_1$ contains units labelled 'metro' and its spatial bounding box intersects $R_2$. The candidate entries at this level are:

  $E_{q_1}$={($k_5$,$I_{k_5}$,{1,2}),($k_6$,$I_{k_6}$,{4})}

  $E_{q_2}$={($k_3$,$I_{k_3}$,{1})}

  Note that now we have $\{1,2\} \cap \{1\} \neq \varnothing$, $\{4\} \cap \{1\} = \varnothing$. The entry $k_6$ is thus filtered out, while only the child nodes pointed by $k_5$ and $k_3$ are candidates nodes for the next step:

  $E_{q_1}$={($k_5$,$I_{k_5}$,{1,2})}

  $E_{q_2}$={($k_3$,$I_{k_3}$,{1})}

- **Level 2: leaf level**. Filtering the leaves we obtain only two units satisfying the query: unit 1 of trajectory 1 pointed by $k_5$ satisfying $q_1$, unit 3 of trajectory 1 pointed by $k_3$ satisfying $q_2$. The result is as follows:

$Tr_{q_1}$={(1,{$u_1$})}
$Tr_{q_2}$={(1,{$u_3$})}

Units are grouped by trajectories and the temporal ordering is checked. Since $u_3$ occurs before $u_1$ the trajectory 1 satisfies the pattern. The resulting set of trajectories identifiers is thus {1}

## 3.6 Experimental Evaluation

In this section we contrast our system with two indexing techniques, both relying on *R-tree* or variants, and close to *IRWI*. The first technique is based on IR-tree [38]. The second technique is IF-R* [69]. Indeed, both these techniques are proposed for the indexing of static spatio-textual data (not for trajectories), so we have implemented the techniques with a 3-dimensional R-tree to include the temporal dimension. For the experiments we use three datasets presenting different features in terms of size, number of labels, distribution of labels. One of the datasets is the GeoLife subset. The other two datasets contain synthetic trajectories generated by the BerlinMod generator [23]. Two synthetic datasets are used called Syn10000 and Syn75000, respectively. The experimental setting is detailed in the following.

**Software and hardware environment.** IRWI is implemented using the Secondo environment [28](i.e., an extensible database providing a rich set of data structures and operations in the form of algebras). The experiments have been carried out on a PC equipped with Intel i7-3632QM, 2.20GHz processor with 8 GBytes of main memory, disk page size 4 KB, running Ubuntu 14.04.

### 3.6.1 Datasets

The GeoLife dataset contains a small set of long trajectories and very few labels indicating the transportation means. Syn10000 and Syn75000 are of different size, both use a few thousands of labels indicating street names, but the average number of labels per trajectory unit is significantly higher in Syn75000. The features of the datasets are reported in Table 3.3. In all the three cases, the *spatio-textual trajectories* are built on distinct sets of spatial and textual trajectories. In GeoLife the trajectories are in raw format, i.e., sequence of points and annotated intervals respectively. In Syn10000 and Syn75000 the trajectories are represented in terms of moving points (spatial trajectories) and symbolic trajectories [31], respectively.

Table 3.3: The three datasets used for the experiments

| Dataset | #Trajectories | #Units | #Unique labels |
|---|---|---|---|
| *Geolife* | 4484 | 4813552 | 11 |
| *Syn10000* | 10000 | 3828228 | 1896 |
| *Syn75000* | 75000 | 28749582 | 2288 |

### 3.6.2 Impact of System Defined Parameters

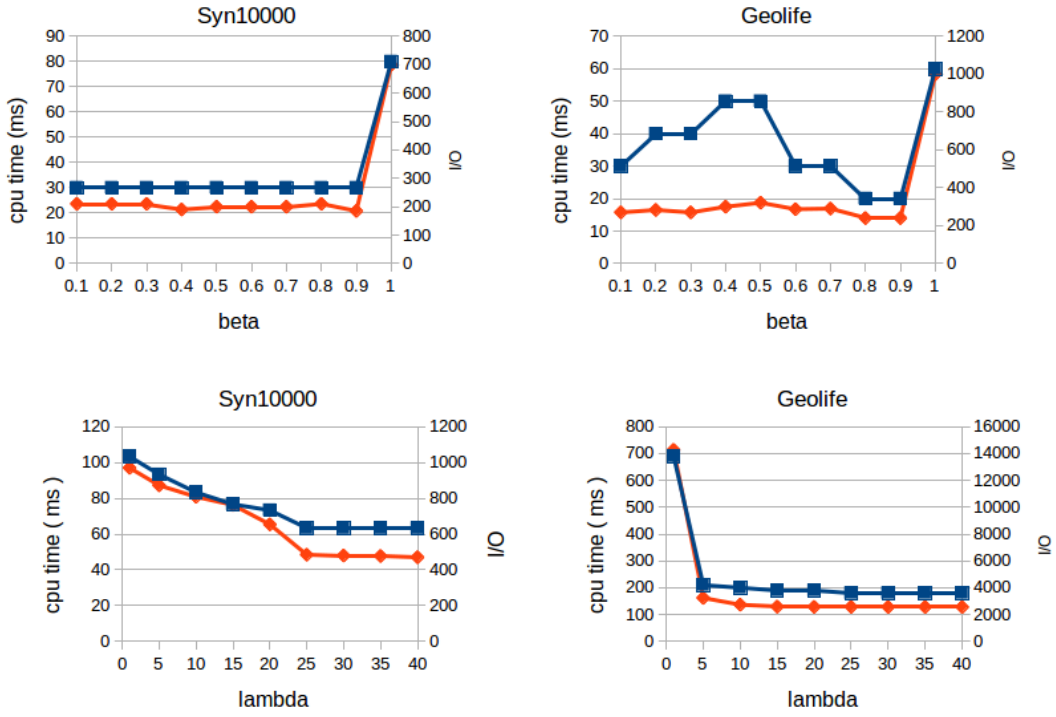



Figure 3.8: Impact of $\beta$ and $\lambda$ on CPU time (blue) and I/O(red)

Recall that we have two system defined parameters: $\beta$ and $\alpha$. We now start with considering the impact of the former one. We recall that $\beta$ is the weight of the spatial component (vs. the textual component) in the cost function, ranging in the interval (0,1]. For the study, we create an *IRWI*-tree for each value of $\beta$ ranging from 0.1 to 1. Next, for each instance of the tree we run 5 queries and take the average values of the measurements. The queries have length 1 and level of selectivity $\approx 0.1\%$. The two datasets Geolife and Syn10000 are used for this experiment. The results are reported in Figure 3.8. It can be seen that the search

is more efficient than in pure R-tree ($\beta = 1$). Consider that this result is obtained with query of length 1. Thus the gain gets more evident with longer queries. We can also see that whenever the labels are spread in space, as in Geolife, the performance is not significantly affected by values of $\beta < 1$. That means that the gain obtained from the grouping of homogeneous units is lost because the regions containing those labels get much larger. Let us turn to the $\lambda$ parameter. As reported in Figure 3.8, the summarization of the trajectories identifiers has a significant impact. In fact the highest CPU time and I/O is obtained with $\lambda = 1$ (i.e., there is only one group defined by the minimum and maximum trajectory identifier). With these datasets, reasonable values for $\lambda$ range between 20 and 40.

### 3.6.3 Comparison with IR-tree and IF-R*

**The IR-tree and IF-R* indexing techniques.** The data structure based on IR-tree is similar to the one we use. In particular, every node of the R-tree is augmented with an inverted list containing the keywords of the objects indexed in the corresponding subtree [17][1]. It should be noted, however, that this technique does not provide any hybrid cost function, neither does it support the summarization of trajectory *Ids*. Therefore, although the sequenced queries are processed concurrently, the early pruning of trajectories cannot be performed. The second index is the IF-R* index. The index data structure consists of an inverted file associating each label of the domain of concern with an R*-tree [69]. Thus the query evaluation starts from the textual condition and only if that is satisfied the spatio-temporal condition is evaluated. We use the index for the sequential evaluation of every simple query of the input sequenced query. If such query contains $n$ distinct labels, $n$ R*-trees are accessed. The indexing techniques are implemented using the Secondo environment.

**Performance metrics.** Two metrics are used for the evaluation of the query processing efficiency: *CPU* time and *I/O* as number of disk accesses. The number of disk accesses is estimated using a counter that is incremented by one every time a node of the index is retrieved, and by the number of uploaded disk pages, every time the posting list associated with a label is read from the inverted file.

**Queries creation of length $m$ and selectivity $s$.** We generate a set of queries of length $m$ and selectivity $s$. The process consists of 5 steps. First, we select a set of trajectories randomly from the dataset. The number of trajectories is

[1]The leaf nodes in our IR-tree implementation do not include pointers to inverted lists, moreover terms are not assigned a weight, as in [17]

related to the required query selectivity $s$. Second, we split every trajectory $tr$ into $m$ sub-trajectories of equal length. Third, we create $m$ sets of trajectory units where every set $i$ contains a random unit from the sub-trajectory $i$ of every trajectory $tr$. Fourth, we generate a simple query from each set of units, such that the temporal range of the query is the union of temporal ranges of the units within the set, and the spatial range is the union of spatial ranges. For the label condition, we randomly choose a number of labels from the units of this set. Fifth, if the temporal and spatial ranges are found to be large, they are split. Finally, after running the queries, we discard those whose selectivity value is very far from the targeted value $s$.

**Query processing efficiency.** We create three instances for each index (IRWI, IR-tree and IF-R*), one for each of the datasets. The parameters of the *IRWI*-tree are set to $\beta = 0.9$ and $\lambda = 25$. Queries are chosen of selectivity (0.1%). The query length ranges from 1 to 6. For every length query, we create a set of 5 queries. Queries are chosen so as to contain from 1 to 10 labels, and both small and big regions. We run the queries and report the average CPU time and I/O in the graphs of Figure 3.9. It can be seen that in all of the cases the CPU

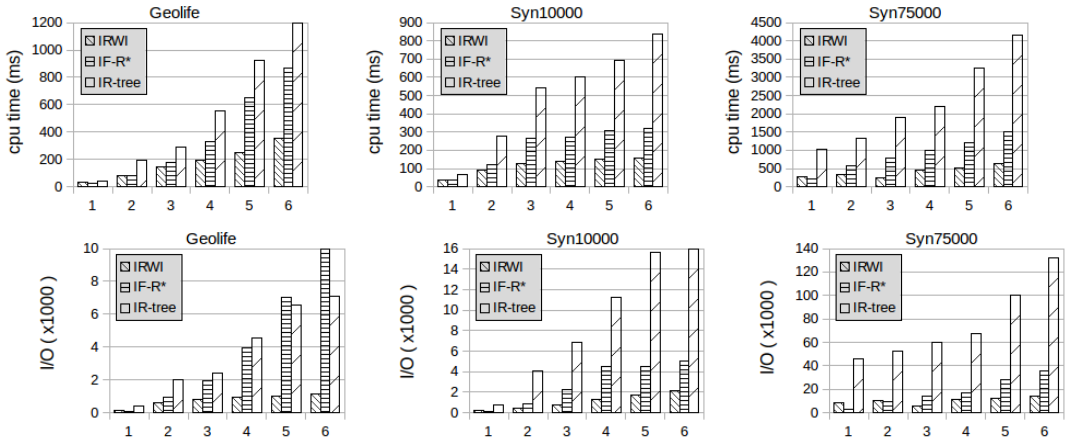



Figure 3.9: Comparing the three indexing techniques: CPU time and I/O operations with sequenced queries of different length (x-axis) ranging from 1 to 6

time and the number of I/O operations increase with the length of the sequenced query. For short sequenced queries, containing up to 2 simple queries, IF-R* and IRWI present similar results. Intuitively, short queries contain few labels, thus the selectivity of the whole query is generally close to that of the single sub-queries. However, as the length of the queries increases, we can observe

that IRWI outperforms both IF-R* and IR-tree. Notably, for 6-length queries, the CPU time consumed by the IRWI with the Geolife dataset is 50% the time consumed by IF-R* and 33% the time consumed by IR-tree, while, with synthetic data, the CPU time is 35% the time consumed by IF-R* and 20% that of IR-tree. We can also observe that the use of the hybrid cost function along with the summarized representation of the trajectories *Ids* leads to a significant reduction in the number of I/O operations. That is especially evident whenever queries contain selective labels, as in the case of the synthetic data.

**Index construction time cost and storage overhead.** Table 3.4 reports the index construction time cost and storage overhead for the different solutions. Not suprisingly, IRWI presents the highest time cost. That is essentially due to the introduction of the *spatio-textual cost* function. We recall that such a function is computed everytime a new unit (i.e., segment) is to be inserted. In terms of

Table 3.4: Index construction time cost (in hours) and storage overhead (MB)

| | Geolife | | Syn10000 | | Syn75000 | |
|---|---|---|---|---|---|---|
| | size | time | size | time | size | time |
| IF-R* | 694 | 3.8 | 528 | 0.6 | 3700 | 8.7 |
| IR-tree | 851 | 2.5 | 660 | 2.1 | 4500 | 16.9 |
| IRWI | 850 | 6.1 | 570 | 2.7 | 4000 | 24.4 |

storage overhead, IRWI requires more space than IF-R* (from 10% to 25%). This additional space is used by IRWI to maintain for every label in the internal nodes the summarized representation of the sets of trajectories containing such a label.

## 3.7 Summary

The main contribution of the thesis is the IRWI indexing framework. IRWI supports a novel type of queries, called sequenced queries, defined over spatio-textual trajectories. IRWI consists of a hybrid, spatial and textual, index data structure, enriched with a number of features that facilitate the early pruning of trajectories during the concurrent evaluation of the sub-queries $q_1, .., q_n$. As a result, the IRWI tree can be traversed only once. The experiments, conducted on both synthetic and real datasets, show a gain in performance with respect to state-of-the art techniques that increases significantly with the length of the query sequence. This gain is paid, however, in terms of time complexity of the index creation. Increasing the efficiency of this operation is a major goal of future research.

# Chapter 4

# A Clustering Technique for Spatial Trajectories Annotation

## 4.1 Overview

This chapter is about the extraction of textual annotations from spatial trajectories. Spatial trajectories contain valuable knowledge on the mobility behavior of individuals. If such a knowledge can be extracted and expressed in textual form, e.g., through a label, then it can be compactly represented through a spatio-textual trajectory. This would close the loop.

Textual annotations can be generated in different ways, for example they can be manually added to sub-trajectories, or acquired from some external source or associated with mobility patterns automatically extracted from the spatial trajectories. The work presented in this chapter fits into the last case. Specifically the problem to solve is to detect the *stay regions* of a moving object, where a stay region does not necessarily designate a precise geographical entity, such as a point of interest or a forest, but where an object is significantly present for a period of time, in spite of periods of absence. The problem cannot be solved using existing techniques for the detection of *stop-and-move* patterns. A novel clustering algorithm called *SeqScan* is thus proposed. For validating *SeqScan* on a real problem, the technique has been applied to the study of the migratory behavior of wild animals in collaboration with a group of biologists.

The rest of the chapter is organized as follows: Section 4.2 presents the motivations; Section 4.3 the *Stay Region* model and the SeqScan algorithm; Section 4.4 illustrates the validation efforts conducted in the field of animal ecology.

## 4.2 Motivations

### 4.2.1 Case Study and Requirements

A popular pattern of individual mobility is the stop-and-move pattern. This pattern describes the movement as a sequence of transitions from one stop to another, where a *stop* indicates a temporary suspension of the movement at the chosen level of abstraction. A stop implies that some important activity may have been carried out by the moving object at some location for some time. Examples of these activities include stopping for lunch in a restaurant during a trip, animal foraging and mating.

Discovering stops in spatial trajectories is a complex task. The main complexity comes from the diversity of stop definitions, which are application dependent and may exhibit diverse spatial, shape, size and temporal characteristics. For example, a *stop* may consist of a large number of points scattered around a *POI* for a small time period (circular shape) like in a touristic trip, or, differently, the stop may consist of points scattered in an arbitrarily wide area for large periods of time as in the case study illustrated next.

**The case study in animal ecology.** The case study regards the analysis of the movement of a number of roe deer equipped with a low-sampling-rate GPS collar and tracked for a period covering a few seasons. The overall goal is to extract the migratory behavior of the individuals. Actually, the animals of this species can either migrate or be stationary, a behavior known as *partial migration* [14]. Moreover, whenever an animal migrates, the migration takes place with modalities and times that - although respecting certain general patterns, e.g., seasonality - can vary from animal to animal. This means that every animal has its own migration behavior. At very coarse level, the behavior can be seen as a *stop-and-move* pattern [52], i.e., the animal stays in a region for some time and then moves to some other region. In reality the behavior is more complex. The animals spend most of their time inside a *home-range*. The concept of *home-range* is key in animal ecology. A popular definition of *home-range* is that of "area traversed by the individual in its normal activities of food gathering, mating and caring for young" [10]. Occasionally the animals can make *excursions* outside the home-range and possibly stay for short periods in a different area before returning to the home-range. A *migration* is a transition from one home-range to another home-range. During the migration the animal can stop in small areas for a short time (*stopover*). An example of migration pattern, which combines all these concepts, is reported in Figure 4.1. The migration behavior does not necessarily have a periodicity (e.g., animals may

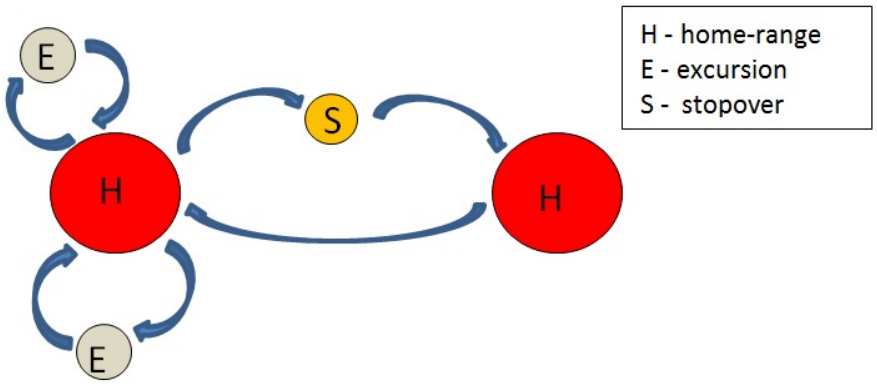



Figure 4.1: Example of animal migration pattern.

not migrate every year). Moreover the temporal and spatial extent of the regions in which the animals stay, as well as the duration of the moves can vary significantly [12]. In addition the location sampling frequency is low - a point every 4 hours or more - therefore movement attributes such as speed, velocity, heading are definitely inaccurate and cannot be used to discriminate whether the animal is stationary or migrates.

**Requirements**. Abstracting away from the case study, the problem can be framed as follows, i.e., to extract the sub-trajectories representing stops under the following assumptions: (i) A stop is a region dense of points. Moreover, the number of stops in a trajectory is not known. (ii) A stop can have an arbitrary spatial shape and temporal extent. (iii) The moving object can experience periods of absence from the stop region. (iv) The stops are to be temporally disjoint, i.e., the temporal extent of a stop does not overlap with the temporal extent of another stop. (v) Mobility attributes, such as speed, velocity, heading, are not reliable. (vi) There can be an arbitrary number of points in the trajectory that do not fall in any stop.

### 4.2.2 State-of-the art Techniques

A popular technique for the detection of stops is presented by Zheng et al. [65]. The approach is centered on the notion of *stay point* defined as a set of consecutive GPS points of the trajectory, close to each other (based on distance threshold) and in which the user stays for a minimum time (duration threshold). The temporal scale is a large scale (e.g., the stay lasts minutes, hours), moreover the semantics of the stay points is given by the geographical context. This solution does not fit well in scenarios in which the moving object can stay in a region for months

or years, and exhibit a complex behavior, such as moving back and forth from a region whose spatial and temporal boundaries are uncertain. Other techniques exist, however all of them rely on restrictive assumptions. For example in [4], stops are associated with the sub-trajectories where either there is absence of signal or the velocity is zero for a given temporal interval (e.g., the car is parked). In CB-SMoT [44] the stops are those sub-trajectories in which the speed is lower than the average speed along the trajectory. Another criterion used in DB-SMoT is the change of direction [48], specifically the angular value of the change in direction must be greater than a given threshold. If movement attributes are unreliable, all these techniques are useless.

A different paradigm is trajectory segmentation [9, 60, 1, 64, 2]. Trajectory segmentation is extensively applied to extract *stop-and-move* patterns such as in [1, 60]. The idea of segmentation is to partition the trajectory in segments of maximal length inside which the movement characteristics are *monotone*, i.e., the same characteristics hold in every subsegment of the segment [9]. Monotone characteristics include speed, velocity, heading, curvature, distance. One could define, for example, a stop as a segment in which the object's speed is below a threshold value. An extension of the study to segmentation based on non-monotone criteria has been recently presented [3]. Unfortunately, all these methods are sensible to noise, namely are not able to discriminate whether a point belongs to a segment or is an outlier, unless using trivial heuristics, such as bounding the number of noise points.

In this work, we present a novel approach to the stop discovery problem that combines the effectiveness of *density-based clustering* with the partitioning capability of trajectory segmentation without introducing any supplementary assumption on movement characteristics. The key idea is to only use *density* and *presence* as non-monotone criteria for the partitioning of a trajectory in a set of sub-trajectories of maximal length and temporally non-overlapping.

## 4.3 The Stay Region Model and the SeqScan Algorithm

We propose an approach grounded on the novel concept of *stay region*. In the next we elaborate on the stay region model and present the clustering technique for the discovery of stay regions.

### 4.3.1 Stay Region: a Conceptual View

A stay region denotes a stop where the moving object spends significant time. The time spent in the stop region can be arbitrarily long (compatible with the duration of the trajectory). In addition, a stay region does not designate a precise geographical entity (e.g., forest, city), rather, the region has fuzzy spatial and temporal boundaries. While residing in a stay region, the object can temporarily leave this region to perform *excursions*. In addition, objects can *move* from one stay region to another stay region. The number of moves/stay regions/excursions is not known. The intuition is that the object leaves a stay region for another stay region when the new region becomes 'more attractive' than the previous one. The concept of stay region is built on that of density-based cluster [24], from which it differs, however, in many aspects especially related to the use of time. For this reason, before presenting the stay region model, we review the fundamental concepts of DBSCAN.

### 4.3.2 Background: the DBSCAN Cluster Model

DBSCAN (density-based spatial clustering of applications with noise) is a clustering algorithm defined by Ester et al. in [24] that can detect clusters of arbitrary shape in presence of noise. Given a set of points in a d-dimensional data space, DBSCAN groups the points that are closely packed together (points with many near neighbors), marking as noise the points that lie in low-density regions (whose nearest neighbors are too far away). More formally, consider a database $P$ of points in space and the input parameters $\epsilon \in \mathbb{R}$ (i.e., the distance threshold), and $K \in \mathbb{N}$ (i.e., the minimum number of points that a cluster contains). Let $d()$ be a metric distance function in the reference space, e.g., the Euclidean distance. The fundamental concepts of the DBSCAN cluster model are as follows [24]:

(i) The $\epsilon$-*neighborhood* of $p \in P$, denoted $N_\epsilon(p)$, is the subset of points that are 'close' to $p$, i.e., $N_\epsilon(p) = \{p_i \in P, d(p, p_i) \leq \epsilon\}$.

(ii) Point $p$ is a *core point* if its $\epsilon$-neighborhood contains at least $K$ points, i.e., $|N_\epsilon(p)| \geq K$.

(iii) A point that is not a core point but belongs to the neighborhood of a core point is a *border point*

(iv) *Noise point* is a point that is neither core nor border point

(v) Point $p$ is *directly density-reachable* from $q$ if $q$ is a core point and $p \in N_\epsilon(q)$

(vi) Two points $p$ and $q$ are *density reachable* if there is a chain of points $p_1, .., p_n$, $p_1 = p, p_n = q$ such that $p_{i+1}$ is directly reachable from $p_i$

(vii) Points $p$ and $q$ are *density connected* if there exists a core point $o$ such that both $p$ and $q$ are density-reachable from $o$

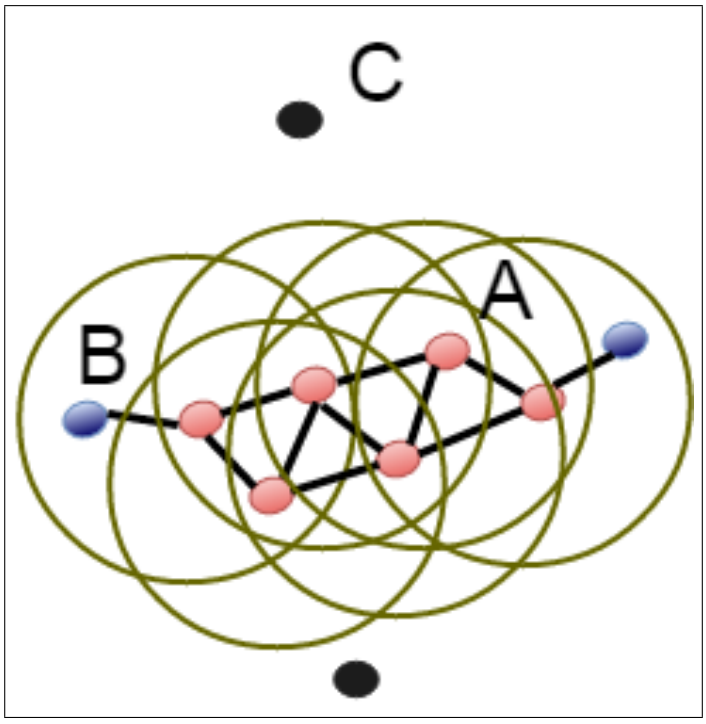



Figure 4.2: DBSCAN cluster over a set of 10 point with parameter K=4. The red and blue colors indicate the core and the border points, respectively, the black color the noise points

A cluster is finally defined as *a maximal set of density-connected points*, where maximal means that every point $p$ that is reachable from a core point $q$ belongs to the cluster containing $q$. For example Figure 4.2 shows a cluster within a set of 10 points ( K= 4. Point $A$ and the other red points are core points, because their $\epsilon$-neighborhood contain at least 4 points). Because they are all reachable from each other, they form a single cluster. Point $B$ is not a core point, but is reachable from A (via other core points) and thus belongs to the cluster as well. Point $C$ is a noise point being neither a core point nor density-reachable.

### 4.3.3 The Stay Region Model and Problem Formulation

We now introduce the stay region model in more formal terms. We begin by introducing some basic notation and assumptions.

A *trajectory* $T$ is a sequence of spatio-temporal points $T = [p_1, .., p_n]$ with $p_i = (l_i, t_i)$ where $l_i$, $t_i$ is the sampled location in space and time respectively with $t_i < t_{i+1}$ and $n$ the length of the trajectory. The trajectory has a begin point $p_{start}$, an end point $p_{end}$, a temporal extent $ext = [t_{start}, t_{end}]$, and a duration $dur = |t_{start} - t_{end}|$. The duration is measured in, e.g., days or in some other unit. No assumption is made on the sampling rate. However the distance in space covered by the object in the time interval $[t_i, t_{i+1}]$ is normally limited (in relation to the mobility pattern under consideration).

A *sub-trajectory* $S = [p_i^1, .., p_j^m] \subseteq T$ of length $m$ is a sequence of temporally ordered points of $T$ with index $1, .., m$. A sub-trajectory may contain *gaps*. A gap is the open interval $(t_i, t_j)$ indicating a 'hole' in the sequence, i.e., two points that are consecutive in $S$ are not consecutive in $T$, i.e., $p_i^x, p_j^{x+1} \in S \rightarrow j \neq i + 1$. For example, given $T = [p_1, .., p_9]$ the sub-trajectory $[p_3, p_5, p_8]$ contains two gaps, $(t_3, t_5)$ and $(t_5, t_8)$, respectively. The temporal extent of the sub-trajectory is $[t_3, t_8]$. Next, we introduce the notions of *dense region*, *presence* and *stay region*.

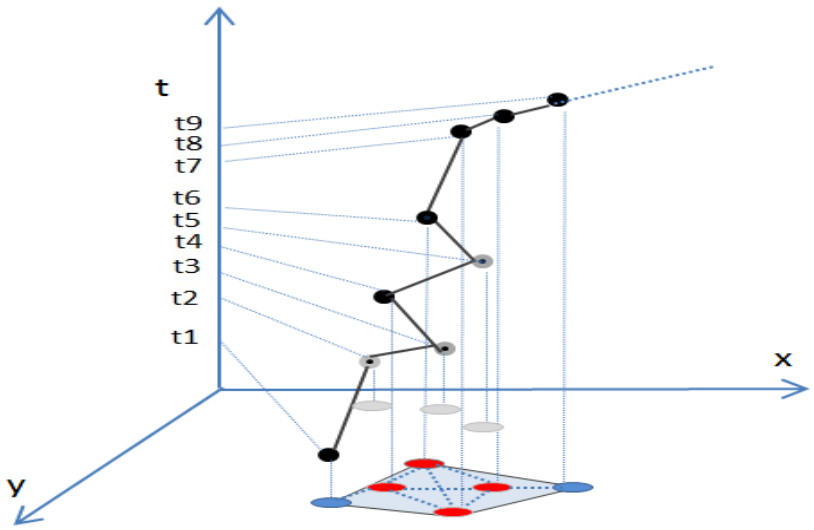



Figure 4.3: Spatio-temporal representation of the example trajectory. The points are projected on space. A subset of points forms a dense region with respect to $\epsilon, K = 4$. Red points are core points while blue points are border points

**Definition 4.1** (Dense region)**.** *A dense region $S \subseteq T$ is a sub-trajectory $S =$*

$[q^1, .., q^m]$ *such that the set of locations* $[l^1, .., l^m]$ *is a DBSCAN cluster with respect to* $\epsilon$ *and* $K$*. The points that do not belong to any dense region in* $T$ *are qualified as noise.* $\diamond$

**Example 4.1.** Consider the trajectory $T = [p_1, .., p_9]$ illustrated in the space-time cube of Figure 4.3. The points projected on the plane are numbered following the temporal order. For the sake of readability, the pairs of points that are at distance less or equal $\epsilon$ are connected by a line. The value of $K$ is set to 3. It can be noticed that the sequence $S = [p_1, p_4, p_6, p_7, p_8, p_9]$ is a dense region.The points $p_4, p_6, p_7, p_8$ are core points while $p_1, p_9$ are border points. The region $S$ contains two gaps, represented by the open intervals $(t_1, t_4)$ and $(t_4, t_6)$.

A dense region may contain gaps, where a gap indicates a period of absence from the region. We call *Time Segment* of a dense region the set of periods in which the object is present in such a region. A graphical representation of the Time Segment associated with the dense region in Example 4.1 is shown in Figure 4.4. This Time Segment consists of two instants and one period (the instant is a degenerated period where the two extremes coincide), i.e., $[t_1, t_1] \cup [t_4, t_4] \cup [t_6, t_9]$.

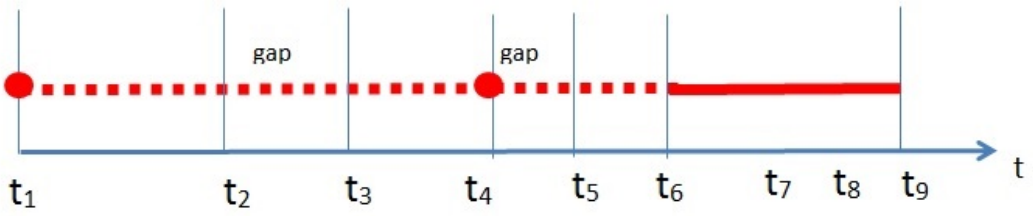



Figure 4.4: The Time Segment of the dense region in Example 4.1. The dotted lines indicate gaps. The circles correspond to instants.

We now introduce the notion of *presence* in an incremental fashion. Let us start considering the case in which $p_i, p_{i+1} \in T$ are two consecutive points both included in the dense region $S$. A sensible question is whether the intermediate points falling in the interval $[t_i, t_{i+1}]$ belong to $S$ or not. We know that the position is uncertain. Yet, we have assumed that the object's move between two subsequent samples is relatively limited in space. This legitimates the assumption that if $p_i, p_{i+1} \in S$ also the points in the interval $I = [t_i, t_{i+1}]$ are contained in $S$. Therefore the *presence* in $I$ is computed as duration of $I$. Consider now the case in which only one of the points is in $S$, namely the object moves somewhere outside the area. In this case, it is plausible that the object is outside the dense region for most of the time. In this case, the presence in $I$ is set to 0. By

extension, we define the presence in $S$ as the sum of the presence in each segment of $S$. A formal definition is given below:

**Definition 4.2** (Presence)**.** *Let $S = [q^1., .., q^m]$ be a dense region in $T = \{p_1, .., p_n\}$. Denote with $S[i, i+1]$ two consecutive points in $S$. We define:*

- *The presence in $S[i, i+1]$:*

$$P(S[i, i+1], T) = \begin{cases} |t_h - t_{h+1}|, & \text{if } \exists h, q^i = p_h, q^{i+1} = p_{h+1} \\ 0, & \text{otherwise} \end{cases}$$

- *The presence in the dense region $S$:*

$$P(S, T) = \sum_{i \in [1, n-1]} P(S[i, i+1], T)$$

- *We say that the presence in $S$ is persistent with respect to $\delta \geq 0$ (*presence threshold*) if it holds: $P(S, T) \geq \delta$ $\diamond$*

The presence can be easily computed from the *Time Segment*. For example, given the Time Segment in Figure 4.4 the presence in the dense region is $|t_9 - t_6|$. Based on these definitions, we introduce the key notion of stay region and next the problem to address.

**Definition 4.3** (Stay region)**.** *A stay region $S$ in the trajectory $T$ is a sub-trajectory of $T$ such that: (i) $S$ is a dense region with respect to $\epsilon$ and $K$. (ii)The object's presence in $S$ is persistent with respect to $\delta$.*

Finally the problem to address can be formulated as follows:

**Problem formulation.** To extract from a trajectory $T$ the sequence of stay regions with respect to the three parameters: $\epsilon, K, \delta$ such that:

(i) The stay regions are temporally disjoint, i.e., a point in a stay region cannot fall in the temporal extent of another stay region.

(ii) The stay regions are of maximal length, that is if a point can be added to region $S$, then it belongs to $S$. The points that do not fall in any stay region are *noise*.

The result is a set of stay regions and noise points. Every stay region $S$ contains a first (in time) minimal stay region denoted $\hat{S}$ called *core stay region*. This concept will be used later on. Moreover noise can be further categorized in

*transitions* and *excursions*. The points temporally falling between the end of one stay region $S_i$ and the beginning of the next one $S_{i+1}$ are *transitions* (denoted $S_i \rightarrow S_{i+1}$). The noise points falling in the temporal extent of a stay region $S_i$ are the *excursions*. Stay regions and transitions determine a partition of the temporal extent of $T$. As an example, Figure 4.5 shows a trajectory containing two stay regions, along with the respective transitions and excursions.

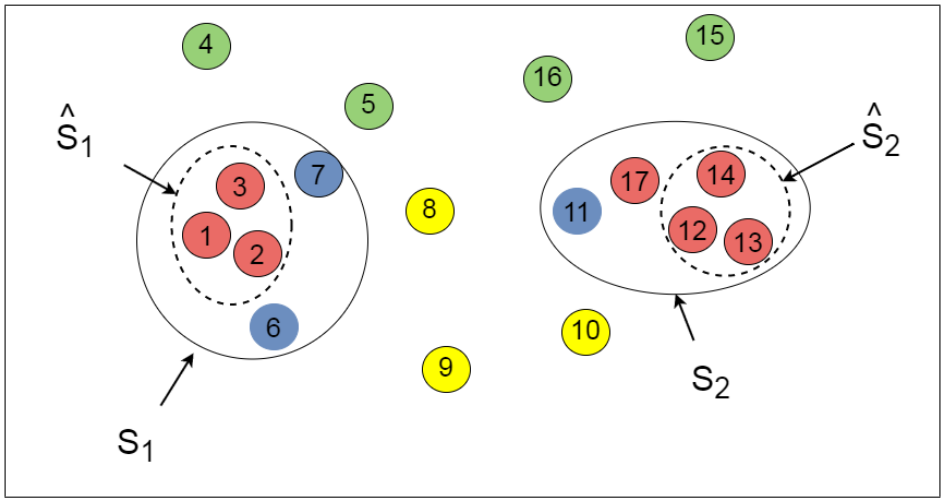



Figure 4.5: Sequence of two stay regions $S_1$ and $S_2$ in the trajectory $1, .., 17$ with respect to $k = 3$, $\epsilon, \delta = 0$. The sets $\widehat{S_1}$ =[1,3] and $\widehat{S_2}$= [12,14] are the core stay regions of $S_1$ and $S_2$ respectively. Core points are presented in red, the border points in blue, excursions and transitions in green and yellow respectively.

### 4.3.4 Extracting Stay Regions: the SeqScan Algorithm

To deal with the above problem, one could be attempted to run the DBSCAN algorithm [24] over the set of points (projected over space) to find the set of dense regions and then determine those in which the presence is persistent. The drawback is that the resulting sub-trajectories are not temporally disjoint. Thus the solution does not work. A different approach is to scan sequentially the trajectory and progressively aggregate the points creating one stay region at a time. In this case, the question is how to recognize the end of a stay region. We recall, in fact, that no supplementary movement characteristic can indicate whether the object is inside or outside a stay region. To deal with this problem, we propose the following approach: (i) a stay region is seen as an *attraction* area, i.e., an area when the object returns after periods of absence. (ii) A stay region remains attractive for the object until a new stay region is found. This means

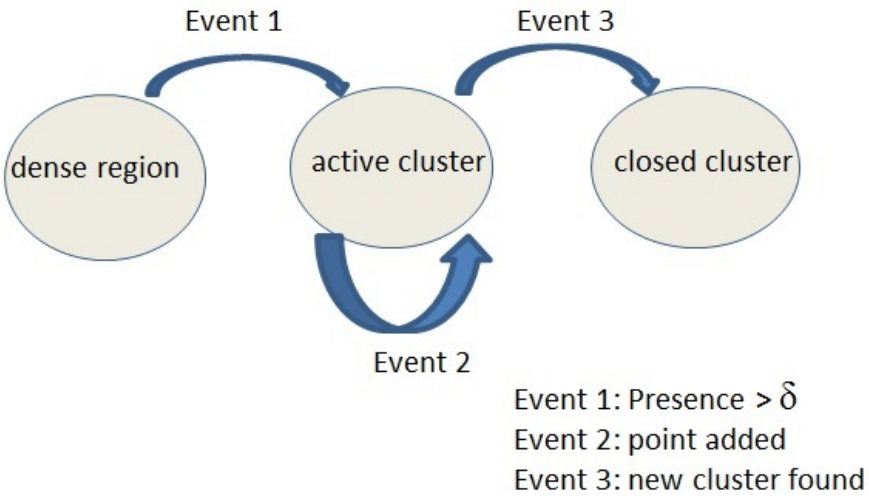



Figure 4.6: Cluster lifecycle

that the boundaries of a stay region are only known when the next stay region is detected (or the trajectory terminates). In the next, we refer to a stay region 'in progress' with the term of *cluster* (not to be confused with the DBSCAN cluster). Clusters are dynamic entities with a life cycle. As illustrated in the state diagram in Figure 4.6 a cluster originates from a dense region when the object's *presence* gets persistent, then the cluster is expanded with new points, and finally it is closed. The event that indicates the termination of the cluster expansion is the starting of a new and more recent cluster. At any instant, there is thus at most one *active* cluster and possibly one or more closed clusters. The algorithm is described in the next.

**Algorithm.** We start by describing the main operations, next we refine the implementation aspects. Every cluster is created and next expanded until it is closed. The period in between the activation of one cluster, say $C$, and the activation of the subsequent cluster is called *time context* of $C$. The notion of time context is introduced to restrict the portion of the input trajectory contributing to the generation of the cluster, namely only the points temporally contained in the time context, can be added to the active cluster. This is to ensure that stay regions are temporally disjoint. The *time context* for the active cluster starts when the previous cluster is closed (or at the beginning of the trajectory) and is updated every time a new point is read. Prior to detailing the Algorithm 5, we introduce two basic functions:

*findCluster(S)*: the function returns the first cluster in the input trajectory $S$ (i.e., the first in time order). If none is found, the function returns the empty-set.

*expand(activeCluster, timeContext, q)*: the function attempts to add the point $q$ to the active cluster, given the time context. If successful, it returns True, False otherwise.

Initially the function *findCluster()* is repeatedly called over sub-sequences of incremental length, e.g., $T[1,i], T[1,i+i]$.. until a cluster is possibly found at time $t_c$. Such a cluster, say $C$, thus becomes active. The cluster has a start point and an end point. The time context of the cluster is initialized to $[t_1, t_c]$. During the phase of *cluster expansion*, the algorithm continues scanning the trajectory until a new cluster is found or the trajectory is terminated. At each step, the algorithm tries to append the current point $p_c$ to the active cluster through the *expand()* function. There are two cases: (i) If the point can be added, the active cluster $C$ grows. The end time of the cluster is the timestamp of the current point. The scan proceeds. (ii) If the point cannot be added to the active cluster, the algorithm determines whether a cluster exists in the sub-sequence starting immediately after the end of the active cluster and the time of current point, i.e., $[t_{end+1}, t_c]$. If it is so, such cluster becomes the active one while $C$ is added to the sequence of *closed clusters*. The points falling in the time context but not belonging to the closed cluster are added to noise. Finally the time context is set to $[t_{end+1}, t_c]$. This guarantees that the cluster is temporally disjoint from the previous one. The cycle thus repeats with the expansion of the new active cluster. The final set of close clusters are the stay regions being discovered consisting of a temporal ordered sequence of sub-trajectories.

**Algorithmic Details.** We have seen that the resulting stay regions are temporally disjoint. Now we focus on the computation of the single clusters. A straightforward implementation of the function $findCluster(S)$ is to run the DBSCAN algorithm over the set of points in $S$ and then check whether the presence in any of the dense regions obtained in this way is persistent. The shortcoming of this approach is that, every time the function is called with an input trajectory that differs of one or few elements from the trajectory of the previous call, the dense regions have to be re-computed again. An alternative approach is to record the status of the dense regions that are progressively found and update such a status when a new point is added. The approach is described in what follows. Preliminarily, points and dense regions representations are described.

- *Points.* When the point $p_i = (x_i, y_i, t_i) \in T$ is read, the point is assigned index $i$, an identifier and a descriptor. The identifier is a pointer to the

**Algorithm 5** Stay Regions Discovery: SeqScan

```
procedure SeqScan(
    In: T = [p_1, .., p_n], ε, K, δ;
    Out: stayRegions, noise)
  c ← 1                                                  ▷ Index scan
  start ← 1, end ← 0
  activeCluster ← ∅
  stayRegions ← {∅}
  while c ≤ n do
    timeContext ← [t_start, t_c]
    if expand(activeCluster, timeContext, p_c) then
      end ← c
    else
      nextCluster ← findCluster(T[t_{end+1}, t_c])
      if nextCluster ≠ ∅ then
        stayRegions ← add(activeCluster)
        noise ← add(T[t_start, t_end] \ activeCluster)
        start ← end+1, end ← c
        activeCluster ← nextCluster
      end if
    end if
    c ← c+1
  end while
end procedure
```

actual coordinates. The descriptor is the pair: $Desc(p_i) = (Neighbours, R)$ where $Neighbours$ is the set of points (identifiers) in the $\epsilon$-neigborhood of $p_i$ and $R$ the possibly empty pointer to the descriptor of the dense region the points belongs to.

- *Dense Regions.* Every time a dense region is created it is assigned the descriptor $(Id, Ts)$ where $Id$ is the dense region identifier (e.g., progressive number) and $Ts$ the Time Segment, defined in the next. The points belonging to the dense region $j$ are thus the set: $\{p_i | Desc(p_i).R.Id = j\}$.

We say that a point $p$: (i) is a core point if it belongs to a dense regions, i.e., $Desc(p).R \neq Null$; (ii) is a border point if it is not a core point and exists a core point $q$ with a neighborhood containing $p$, i.e., $Desc(p).R.Id = null \wedge \exists q, p \in Desc(q).Neighbors$. For the sake of readability, we write $N_\epsilon(q)$ to indicate *Desc(q).Neighbors*.

We recall that the *Time Segment* of a dense region is defined by the set of intervals in which the object is present in the region. (i.e., the gaps are omitted). It is constructed as follows. The points of a dense region are qualified as *entrances* and *exits* where a point $p_i$: (i) is an entrance if the preceding point in the input trajectory $T$ does not belong to the dense region. (ii) $p_i$ is an exit if the subsequent point in $T$ does not belong to the dense region. A point can be both an entrance and an exit. Every pair of subsequent nodes (entrance, exit) specifies one period of presence in $S$. If entrance and exits coincide the period is an instant. The whole presence in $S$ is computed by summing up the duration of each period in the Time Segment.

Detecting whether one of the dense regions is a cluster is straightforward: it is sufficient to consider the Time Segment specified in each of the dense regions. More complex is the expansion phase which attempts to add a point $q$ to the active cluster. This operation is performed by the function *expand()* in the Algorithm 6. The function creates a new point descriptor for $q$ (line 4), while the neighborhoods of the points falling in the time context are updated; next the dense regions descriptors are updated (lines 5-7). In particular the last operation is as follows: for every point $p$ in the neighborhood of $q$ (i.e., $p \in N_\epsilon(q)$): (i) if $q$ is directly density reachable from $p$ (i.e., $p$ is a core point in a dense region $R$) then $q$ is added to $R$, i.e., the Time Segment of $R$, is updated (*LinkCorePoint(p,q)*). Note that the neighborhood $N_\epsilon(q)$ consists exclusively of points falling in the specified time context.(ii) If $p$ has become a core point, after the addition of $q$, but no dense region can contain it, then a new dense region (descriptor) is created. Conversely if $p$ is a core point but its neighbor points belong to different dense regions, these regions are merged into a unique one. Finally the Time Segment is updated accordingly (*LinkNeighbors(p)*). It can be shown that the *expand()* preserves the properties of *density-connectivity* and *maximality* of dense regions (we omit the demonstration for lack of space). An example illustrating how the function works is reported in the next.

**Example 4.2** (Updating operation)**.** Consider a trajectory of 10 points. The projection of the points on the plane is shown in Figure 4.7(a). Points are numbered from 1 to 10 based on the time order. The density parameters are $\epsilon$ and $N = 4$. The points are read in sequence from point 1:

- Point 1-4 are read. None of them is a core point as can be seen from the neighborhoods in Table 4.1 (first four lines):
- The first dense region is created at step 5 (see Figure 4.8(a)). After reading point 5, point 1 becomes a *core point* (the neighborhood includes 3 more

**Algorithm 6** Function expand()

```
1: function EXPAND(activeCluster, timeContext, q)
2:     Global variables : DR, T
3:     c ← 1
4:     createPointDesc(q)
5:     for all p ∈ N_ε(q) do                          ▷ N_ε(q) ⊆ T[timeContext]
6:         linkCorePoint(p, q)
7:         linkNeighbors(p)
8:     end for
9:     return(q ∈ activeCluster)
10: end function
11:
12: procedure LINKCOREPOINT(p, q)
13:     if Desc(p).R ≠ null then                       ▷ p is a core point
14:         Desc(q).R ← Desc(p).R
15:         updateTimeSegment(Desc(p).R, {q})
16:     end if
17: end procedure
18: procedure LINKNEIGHBORS(q)
19:     if isCorePoint(q) and Desc(q).R = null then
20:         if ∄dr ∈ DR where q ∈ dr then
21:             Desc(q).R = CreateNewRegionDesc()
22:             updateTimeSegment(Desc(q).R, N_ε(q))
23:         else
24:             for all dr_1, dr_2 ∈ DR where q ∈ dr_1, dr_2 do
25:                 merge(dr_1, dr_2)
26:             end for
27:             updateTimeSegment(Desc(q).R, N_ε(q) \ q)
28:         end if
29:     end if
30: end procedure
```

points) while points 2,3,5 are *border points*. Because no other dense region contains this core point, a new dense region descriptor is created named $dr_1$. The corresponding Time Segment is set to: $ts_1 = [1, 3] \cup [5, 5]$

- The dense region described by $dr_1$ is expanded (Figure 4.8(b). Point 6 is read. $N_\epsilon(6) = \{1, 5\}$. Point 5 becomes a core point (directly connected to points 1,2,6). Hence point 6 is added to $dr_1$ The time segment of $dr_1$ is updated to: $ts_1 = [1, 3] \cup [5, 6]$

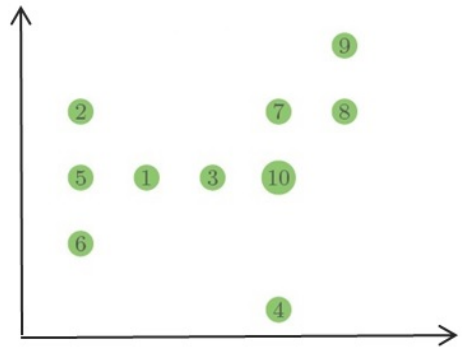



Figure 4.7: Set of 10 points projected on space

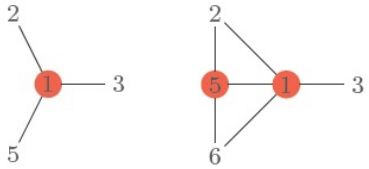



Figure 4.8: (a) Point 1 is a core point. A new dense region descriptor $dr_1$ is created. (b) Point 5 becomes a core point after point 6 is added

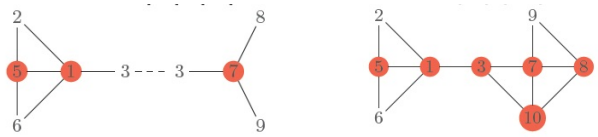



Figure 4.9: (a) A new dense region $dr_2$ is created; (b) the two regions $dr_1$ and $dr_2$ are merged because sharing a core point

- A new dense region is created (Figure 4.9(a)). The points from 7 to 9 are read. We obtain: $N_\epsilon(7) = \{3\}$, $N_\epsilon(8) = \{7\}$, $N_\epsilon(9) = \{7, 8\}$. At this point, 7 is a *core point* that however is not connected to $dr_1$, while 3,8,9 are *border points*. A new dense region $dr_2$ is created with Time Segment: $ts_2 = [3, 3] \cup [7, 9]$.
- Finally the point 10 is read (Figure 4.9(b)). Since $N_\epsilon(10) = \{3, 7, 8\}$, point 3 results to be a shared *core point* between $dr_1$ and $dr_2$. The two regions are merged. As a result also points 8, 10 become a core point. The time segment of the result is set to: $ts_1 \cup ts_2 = [1, 3] \cup [5, 10]$.

**Complexity of the algorithm.** The time complexity of the algorithm is quadratic with respect to the length of the trajectory. The time complexity

| |
|---|
| $N_\epsilon(1) = \{1\}$ |
| $N_\epsilon(2) = \{1,2\}, N_\epsilon(1) = \{1,2\}$ |
| $N_\epsilon(3) = \{1,3\}, N_\epsilon(1) = \{2,3,1\}$ |
| $N_\epsilon(4) = \{4\}$ |
| $N_\epsilon(5) = \{1,2,5\}, N_\epsilon(1) = \{2,3,5,1\}, N_\epsilon(2) = \{1,5,2\}$ |
| $N_\epsilon(6) = \{1,6,5\}, N_\epsilon(1) = \{2,3,5,1,6\}, N_\epsilon(5) = \{1,5,2,6\}$ |

Table 4.1: Neighborhoods of the first 6 points

is measured with respect to the costly operation computing the distance between the current point and the points in the Time Context. In the worst case, the input point $p_i$ is confronted with all the preceding points $p_1, .., p_{i-1}$. The total number of distance operations is thus $\sum_{i \in [1,n-1]} i = \frac{n(n-1)}{2}$, i.e., the time complexity is $O(n^2)$, that is the complexity of the DBSCAN algorithm, if no indexing mechanism is used [50]).

### 4.3.5 Parameter Analysis

One of the novelty of this model with respect to the state of art is the notion of *presence*. We remind that the presence lower bound, i.e., $\delta$, is one of the parameters of *SeqScan* and that this parameter defines the temporal granularity of the stay regions. In this section we study the relationship between presence and number of clusters.

#### Presence

Let $f : \mathbb{R} \to \mathbb{N}$ be the function expressing the relation between *presence* and number of stay regions (the density parameters $k$, $\epsilon$ are fixed). We show that the number of stay regions remains constant for values of $\delta$ ranging in properly defined intervals. The function $f$ can thus be discretized. The proof is reported in the following. This result is at the basis of the algorithm for the computation of the function $f$ that will be presented next.

The proof is in two steps: Lemma 4.1 demonstrates that the number of stay regions does not change if we consider the sequences of *core stay* regions in place of the maximal stay regions. We recall that the core stay regions are those that are detected for first during the scan of the trajectory, and next expanded. This result is used in Theorem 4.1. to prove that the number of stay regions does not change for values of presence ranging in a properly defined interval.

**Lemma 4.1.** *Let $\widehat{S} = \widehat{S}_1, \widehat{S}_2, ...$ and $\widehat{S^*} = \widehat{S^*_1}, \widehat{S^*_2}, ..$ the two sequences of minimal stay regions in T obtained with $\delta = p$ and $\delta = p^*$ respectively (with respect to density parameter K,$\epsilon$). These two sequences $\widehat{S}$ and $\widehat{S^*}$ are identical iff the corresponding stay regions are identical.*

$$\widehat{S} = \widehat{S^*} \Leftrightarrow S = S^*$$

*Proof.* We prove the implication $\widehat{S} = \widehat{S^*} \Rightarrow S = S^*$ (the other way is trivial). Consider for a generic index i, the equality $S_i = S^*_i$. By definition of minimal stay region, every element of $S_i$ is reachable (in the DBSCAN sense) from its minimal stay region $\widehat{S}_i$. Thus every element is also reachable from $\widehat{S^*_i}$ (as $\widehat{S^*_i} = \widehat{S}_i$). The two stay regions $S_i$ and $S^*_i$ are thus identical. □

**Theorem 4.1.** *Let $S_1, .., S_m$ the sequence of stay regions obtained with $\delta$ =p and $p_1, .., p_m$ the presence in the core stay regions. It holds that:*

$$\forall p^* \in [p, \min_{\forall i \in [1,m]} p_i], f(p^*) = f(p)$$

*Proof.* Let $p^* \in [p, \min_{\forall i \in [1,m]} p_i]$. We show that $\widehat{S^*} = \widehat{S}$ thus, for the previous Lemma, $S^* = S$. We recall that the SeqScan algorithm scans the trajectory until a core stay region is found, hence such region is expanded until a new, spatially separated core region is detected. As the value $p^*$ is equal or lower than the presence in the core stay regions, if we run SeqScan with $\delta = p^*$ none of the core stay region in $\widehat{S}$ is filtered out. For the same reason, no additional core stay region can be found. Thus $\widehat{S^*} = \widehat{S}$ that is what we wanted to demonstrate. □

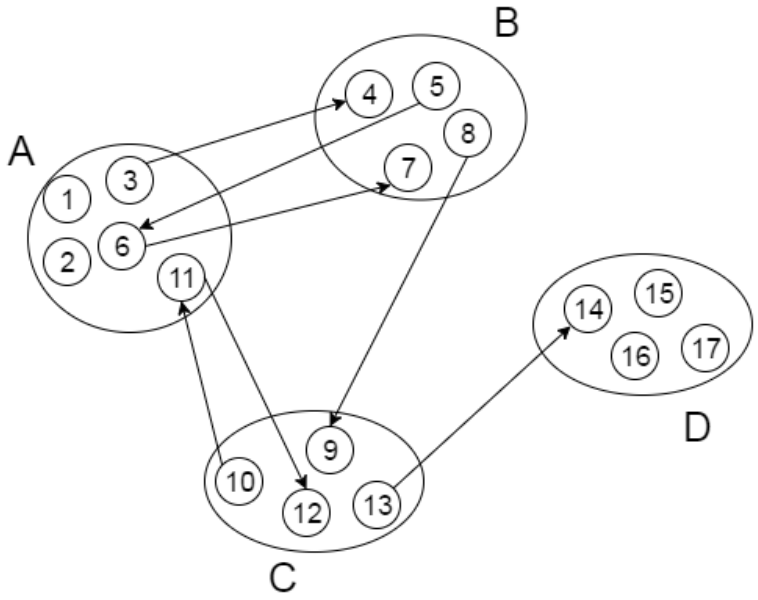
**Algorithm 7** Computing the function f

```
function F( In: T = [1, n], ε, K; Out: setOf(range,nstays))
    δ ← 0, setOf(range,nstays) ← ∅
    while δ ≠ −1 do
        SeqScan(T, ε, K, δ, S = S_1, .., S_m, noise)
        min ← min_{∀i∈[1,m]} Presence(Ŝ_i)
        if min = 0 then δ= -1 else
        setOf(range,nstays).add(([δ, min], Size(S)))
        δ ← min + constant                          ▷ constant > 0
        end if
    end while
end function
```

**Example 4.3.** We study the function f for the following trajectory of 17 points represented in Figure 4.10. We indicate for every pair of points the temporal distance (t-distance, in time units):

$1 -_1 2 -_1 3 -_1 4 -_2 5 -_1 6 -_1 7 -_2 8 -_1 9 -_2 10 -_1 11 -_1 12 -_2 13 -_1 14 -_2 15 -_2 16 -_1 17$

For example the t-distance between point 1 and point 2 is of 1 time unit, between point 4 and point 5 is of 2 time unit. Let us set the density parameter to k=3 (as usual $\epsilon$ is implicit). Steps of the algorithm:




| Ranges | #Clusters |
|---|---|
| [0,2] | 2 |
| (2,4] | 3 |
| (4,5] | 1 |
| > 5 | 0 |

Figure 4.10: Example of the clustering where the function f is represented in the table

- Run SeqScan with $\delta$=0. Two stay regions are detected, A and D, with $\widehat{A}$=[1,3] and $\widehat{D}$ =[14,16]. The presence in $\widehat{A}$ is 2 time units, while the presence in $\widehat{D}$ is 4. Therefore, for the values $\delta$ in [0,2], the number of clusters is 2.

- Run SeqScan with $\delta$= 2 + $\theta$ (where $\theta > 0$ is a constant used to handle the discontinuity). Three minimal stay regions are detected: the first is [4,5] ∪ [7,8] with presence= 4 time units, the second is [9,10]∪[12,13] with presence =4 time units, the third core stay region is [14,16] with presence=4. Thus, in the time interval (2,4], three stay regions are detected.

- Iterating: with $\delta = 4+ \theta$ there is 1 core stay region with presence=5, while for $\delta > 5$ the number of core stay regions is 0 that is the terminating condition.

## 4.4 Evaluation of SeqScan

A critical aspect of clustering is the validation of the results as well as of the technique itself. Ideally, the validation can be performed in different manners, for example through the use of synthetic data acting as ground truth. However, due to the lack of ground truth data, we have applied the technique on the case study discussed earlier in the chapter and we have analyzed the results in collaboration with domain experts. In particular, the validation activity that has been conducted consists of two phases: 1) Initially SeqScan has been used with real data reporting the movement of animals (roe deer) that are constantly monitored by the domain experts. The evaluation is qualitative, and based on the judgment of domain experts. 2) In the subsequent phase, SeqScan has been contrasted with an existing clustering-based algorithm recently proposed for the analysis of the migratory behavior of the same species, i.e., roe deer. We refer to this activity as quantitative evaluation. The methodology and the results are reported in the next.

### 4.4.1 Qualitative Evaluation

We analyze the spatial trajectories of 25 roe deer tracked in the period 2005-2008. The dataset is provided by the research institute Fondazione E.Mach. The total number of samples amounts to over 50000 points. The animals live in an area of about $30km^2$ on the Alps near the city of Trento (Italy). The history of these animals, since their capture for the installation of the GPS collar, is known [12]. The basic statistics of the dataset are reported in Table 4.5. The points are sampled approximately every 4 hours, to reduce battery consumption. Under normal conditions, the distances covered in that time period are a few hundreds of meters, thus the movement is generally limited (in relation to the phenomenon of migration). A significant number of samples in the dataset are missing, due to the fact that the satellite signal is lost, for example, when the animals stay inside a dense forest, therefore the temporal distance between subsequent points varies. Trajectories are of different length and duration as shown by the high standard deviation value($\sigma$).

**Methodology** The SeqScan algorithm is applied at two different stages. In the first stage, the algorithm is used to identify the large regions and runs over the

| Trajectories | $Avg$ | $\sigma$ |
|---|---|---|
| $\Delta s$ | 165 meters | 47 meters |
| $\Delta t$ | 5.17 hours | 0.93 hours |
| length | 1869 points | 503 points |
| duration | 401 days | 154 days |

Table 4.2: Summary statistics: average spatial distance between two consecutive points in every trajectory ($\Delta s$), average temporal distance between two consecutive points in every trajectory ($\Delta t$), length and duration of trajectories.

entire trajectory. This phase is called *coarse analysis*. In the second phase, the algorithm is run over the sub-trajectories representing *filtered noise*, i.e., long gaps and transitions, to detect the small regions. This analysis is called *fine-grained*. The values of the clustering parameters are reported in Table 4.3. Probably facilitated by the low number of parameters and the small group of animals, it has been possible to set a common set of values for all of the trajectories. The values are obtained empirically, on the basis of domain knowledge and data statistics. We will come back on this aspect later on.

| | $\epsilon$ | $K$ | $\delta$ |
|---|---|---|---|
| coarse analysis | 100 meters | 20 objects | 25 days |
| fine-grained analysis | 40 meters | 5 objects | 6 days |

Table 4.3: Clustering parameters value

The animal behavior extracted from the algorithm is described both in textual and map form. The textual form follows the syntax of symbolic trajectories [31]. Specifically, every stay region is given a label $l$ and a time period $I$. The label $l$ consists of two short texts, the former indicates the type of the stay region (e.g., $H$ stands for home-range, $S$ for stopover and $E$ for excursion). The time period $I$ is the period of presence in the stay region denoted by the label. The symbolic trajectory for each animal is thus obtained by concatenating the descriptions: $\{I_1\ label_1\}\ \{I_2,\ label_2\}$.... The stay regions are displayed on maps as sets of points enclosed in convex hulls (noise is omitted).

**Evaluation and discussion.** We show a few representative examples of migration behavior. In particular we consider three animals, nick-named Michela, Alessandra, Lara. The sets of sampled points for each of the animals are reported in Figure 4.12.

```
{ ["2005-12-10 00:01:00.023"  "2006-04-14 08:01:00.023" ]"H0"}
{ ["2006-04-14 12:01:00.053"  "2007-08-04 00:02:00.011" ]"H1"}
{ ["2007-08-04 08:02:00.018"  "2007-08-18 20:01:00.049" ]"E0"}
{ ["2007-08-19 00:01:00.041"  "2008-01-07 12:01:00.051" ]"H1"}
{ ["2008-01-08 20:02:00.023"  "2008-02-29 16:00:00.054" ]"H2"}
```

(a)

```
{ ["2008-01-09 04:02:00.024"  "2008-01-09 08:01:00.029" ]"H2"}
{ ["2008-01-09 20:00:00.054"  "2008-01-10 12:02:00.006" ]"H2"}
{ ["2008-01-12 12:02:00.009"  "2008-01-12 16:01:00.047" ]"H2"}
{ ["2008-01-14 08:00:00.054"  "2008-01-14 12:01:00.047" ]"H2"}
{ ["2008-01-20 08:00:00.054"  "2008-01-20 16:01:00.027" ]"H2"}
{ ["2008-01-23 08:01:00.012"  "2008-01-24 12:00:00.053" ]"H2"}
{ ["2008-01-25 00:01:00.022"  "2008-02-02 12:01:00.024" ]"H2"}
{ ["2008-02-03 00:02:00.015"  "2008-02-21 08:01:00.023" ]"H2"}
{ ["2008-02-22 16:01:00.046"  "2008-02-23 04:01:00.042" ]"H2"}
{ ["2008-02-24 00:02:00.054"  "2008-02-25 04:01:00.042" ]"H2"}
{ ["2008-02-26 16:01:00.024"  "2008-02-26 16:01:00.024" ]"H2"}
{ ["2008-02-27 08:02:00.059"  "2008-02-28 16:01:00.051" ]"H2"}
{ ["2008-02-29 08:01:00.055"  "2008-02-29 16:00:00.054" ]"H2"}
```

(b)

Figure 4.11: Symbolic trajectories: (a) Michela's behavior in synthetic form, i.e. short temporal gaps are omitted. (b) The Time Segment of one of the stay regions.

- The migration behavior of *Michela* is illustrated in Figures 4.13(a-b). The trajectory has about 4606 points over two years. The animal moves from the home-range $H0$ to the home-range $H1$ where it makes an excursion $E0$, then moves to home-range $H2$. The symbolic trajectory at two different levels of detail (synthetic and detailed) is reported in Figure 4.11

  The percentage of noise for Michela is 9.64%. The low percentage indicates that the points are highly aggregated. This measure is in line with the fact that the four stay regions are very close to each other; therefore the transitions do not generate significant noise. It can also be seen that the temporal extent of the regions is aligned with the typical seasonal behavior.

- Alessandra exhibits a different behavior as illustrated in Figure 4.13 (c). The trajectory consists of 2642 points. In this case the animal covers longer distances to move from one region to another and that explains the high percentage of noise, i.e., 32%.

- The behavior of *Lara* is displayed in Figure 4.13(d). The trajectory has

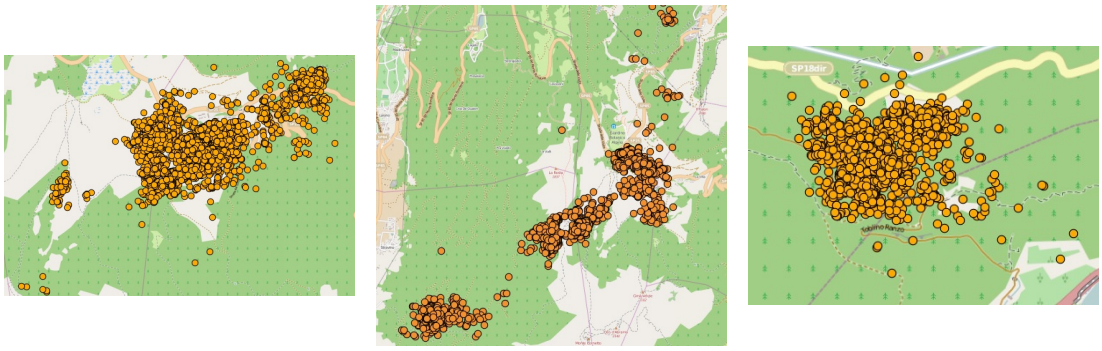


Figure 4.12: Sampled points for the example animals *Michela, Alessandra, Lara* (QGIS maps)

1543 points, thus the animal is observed for a shorter time. During this period, the animal is stationary. This behavior reflects itself in the low percentage of noise, 1.4%.

From the experiments it turns out that 10 animals out of 25 are stationary while the others migrate. While this matches previous results in [12], the algorithm helps to thoroughly describing and finding evidence of migratory behavior in uncertain cases (e.g., the animal Michela). An interesting aspect is that, as

| Group of animals | # | avg(%noise) | $\sigma$ |
|---|---|---|---|
| stationary | 10 | 2.8 | 1.9 |
| migrators | 15 | 20 | 9.9 |

Table 4.4: Average and deviation standard of the noise distribution for the groups of stationary and migrating animals

seen before, the percentage of noise varies significantly depending on whether the animal migrates or not, as reported in Table 4.4. This is relevant for two reasons: first, it shows that the algorithm is able to capture the migration behavior diversity; second, it shows that noise can be a good indicator of the animals' migratory attitude. Ideally, this can facilitate large scale analysis over larger groups of animals. We recall, however, that the animals' behaviors can be contrasted (based on noise) because of the choice of a common set of values for the clustering parameters.

In a few cases we have noticed an interesting divergence between the outcome of the algorithm and visual analysis. The specific situation is sketched in Figure 4.14: an animal stays in the region $A$ then moves to reach region $B$. Progressively however the region $B$ expands until it overlaps $A$. In this case the algorithm

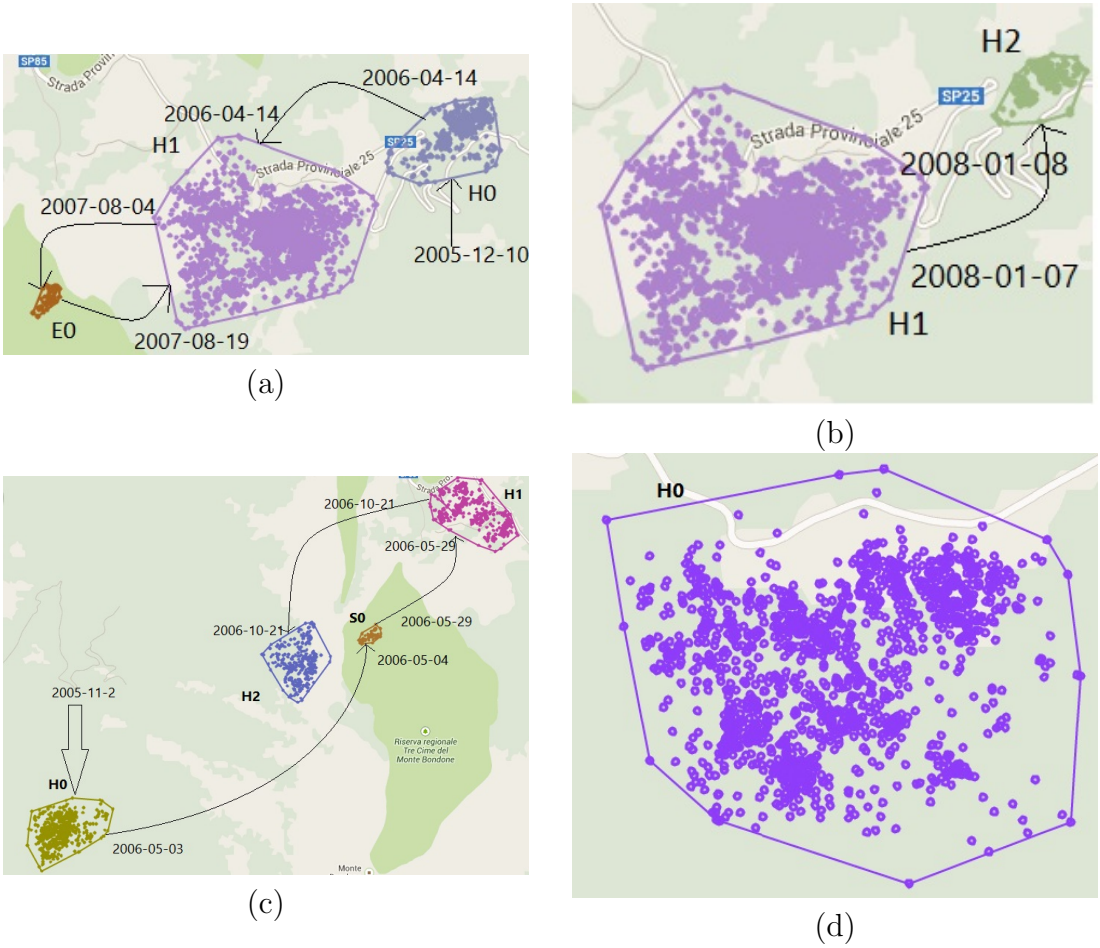



Figure 4.13: (a-b)*Migration pattern of Michela*: from H0 to H1; and from H1 to H2. (c)*Alessandra*: the animal covers longer distances. (d)*Lara*: stationary animal

considers $A$ and $B$ two different clusters, thus if the clusters are large regions, it means that the animal migrates. Conversely the visual analysis leads to consider this set of points as a unique region.

### 4.4.2 Quantitative Evaluation

A second form of evaluation has been conducted to refine the analysis of the tool. For this experiment two different datasets were used, reporting the trajectories of roe-deer (Capreolus capreolus) and red-deer (Cervus elaphus) living in the Bavarian region. These two species of deer, although part of the same Family (Cervidae), are characterized by different life-history traits, space use behavior,

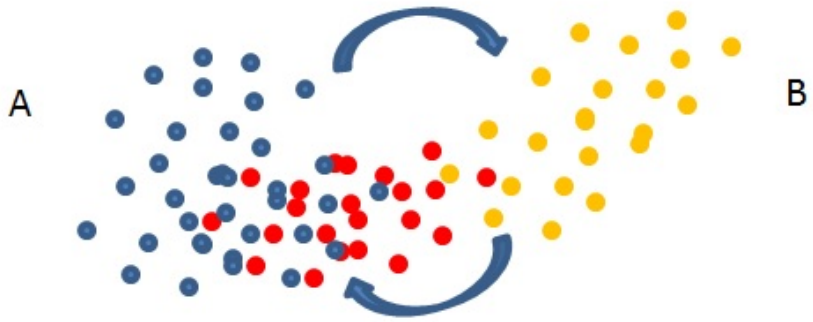



Figure 4.14: Cluster A: blue points; cluster B: yellow points in B. The red points highlight the expansion of B towards A

and size. Therefore, by using trajectories from different animal species within the same region, we evaluate the effectiveness of the method on different movement responses, within similar environmental conditions. Another goal is to contrast SeqScan with a recent technique developed in the context of animal ecology and grounded on statistical methods. In particular, we compare the classification of the migratory behavior, i.e., migrant vs. stationary, obtained by, SeqScan with that provided by the above mentioned reference technique. In the following, first we will outline the main features of the reference method, described in [12] and later developed in [11]. Next, we will describe the datasets. Finally, we present the experimental results along with discussion and analysis.

### The reference method

In this paragraph, we briefly present the steps performed by the reference method to extract the home ranges from animal trajectories. First, the method of Ward [59] is used to minimize the within-cluster sum-of-squares, so that each location is assigned to cluster 1 or 2. Then, with a nearest-neighbor procedure, outlying points are removed, to eliminate long range excursions outside the animal's home range. Then, clusters are 'assigned' to a season, by computing the temporal interval to 15th of July, and assigning the minimum interval to summer cluster, and the maximum interval to winter cluster. After these steps, each animal's trajectory corresponds to a sequence of consecutive points within a certain cluster, or between two clusters. Until this point, the temporal component is not considered by the method. This is done a-posteriori, by computing the maximum continuous time of residence in each cluster and expressing these times as proportion of the total monitoring time. For example, in case of roe and red deer, the large her-

bivores considered in this study, a reasonable threshold to discriminate between migrants and residents was fixed to the value 0.083, which corresponds, on an annual basis, to a 1 month period. In other words, the method assumes that a migrant should continuously stay at least one month in both the winter and summer ranges to be classified as such. In all other cases, the trajectories are classified as 'resident'. The threshold for the maximum continuous time of residence can be considered the main parameter of the reference method. The parameter is defined based on the definition of migration (here seen as a yearly phenomenon), and life-history considerations on the species of concern. For example, one month is here considered the minimum relevant time to large herbivores to access critical resources over a one-year time span.

### Datasets

The two datasets contain normalized trajectories obtained from the preprocessing of data downloaded from GPS sensors, so as to be ecologically comparable. More specifically, every trajectory covers at most one year. Thus for an animal tracked for, e.g., two years there are two normalized trajectories. Moreover, every trajectory starts at earliest on February 15 and ends at least on February 15 of the following year. In addition, every trajectory contains a *summer* range (i.e., the range whose temporal extent contains July) and two winter ranges (i.e., by exclusion those that are not summer ranges), so to identify the 'return' movement typical of seasonal migration. Note that the splitting of multi-annual trajectories in one-year trajectories is basically due the to the inherent difficulty of dealing with both the spatial and the temporal dimension of a series of locations at the same time. We will come back to this point later on. In both datasets, a significant number of GPS locations are missing due to lost satellite signal, due for example to rugged topography or thick forest cover. As a result, the points are sampled at varying frequencies.

**Summary statistics.** The two datasets have a different size. The *roe deer dataset* consists of 29 normalized trajectories with approximatively 60000 GPS points; the *red-deer dataset* has 20 trajectories with about 150000 GPS points. Table 4.5 reports the summary statistics on the length and duration of the trajectories in the two datasets.

### Experimental results and evaluation

We now present and discuss the results obtained by running SeqScan on the trajectories data with the chosen parameters. For every trajectory SeqScan returns:

Table 4.5: Average and standard deviation of trajectories length and duration

| | Roe deer | | Red deer | |
|---|---|---|---|---|
| | Avg | StDev | Avg | StDev |
| length(points) | 2078 | 1176 | 7205 | 4038 |
| duration(days) | 338 | 31 | 337 | 30 |

- A set of labeled points. The points are labeled with a cluster identifier. As effect of the normalization, for each trajectory, there can be only one, two or, more rarely, three ranges

- A set of labeled noisy points

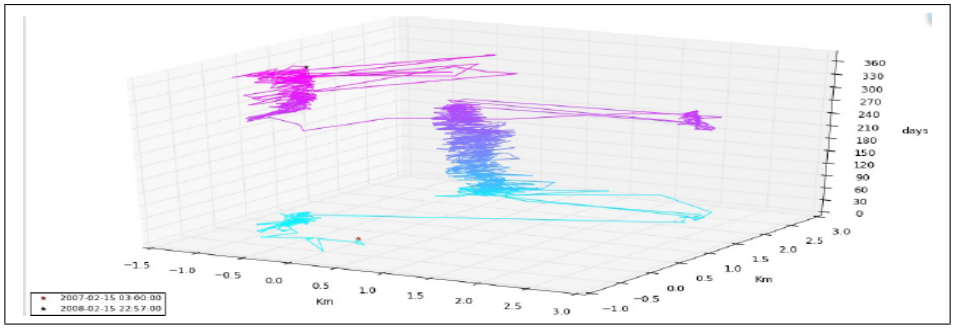



Figure 4.15: Space-time cube of the roe deer trajectory with Id=5. The coordinates indicate spatial and temporal distance from the first point of the trajectory, the color gradient indicates the temporal evolution.

We illustrate the outcome of the clustering through an example. In particular, we examine the trajectory of one animal, i.e., the roe deer with Id=5. To ease understanding, we show in Figure 4.15 the space-time cube of the trajectory. From the graph, it can be seen that the animal moves between two main areas whilst stops for a short period in a third area. We can achieve a better understanding of the movement, from the analysis of the clustering obtained from SeqScan and reported in Figure 4.16. In particular we can see that there are two home ranges, $A$ and $B$. While residing in $A$, the animal performs an excursion to the third area, next migrates to $B$ passing across the area where the animal stopped during the excursion (i.e., the area is somehow memorized). One could argue that this information can be extracted through a careful visual analysis. In reality, whenever there is no clear separation between the ranges, the visual representation of the trajectory is hard to interpret (e.g., see Figure 4.17).

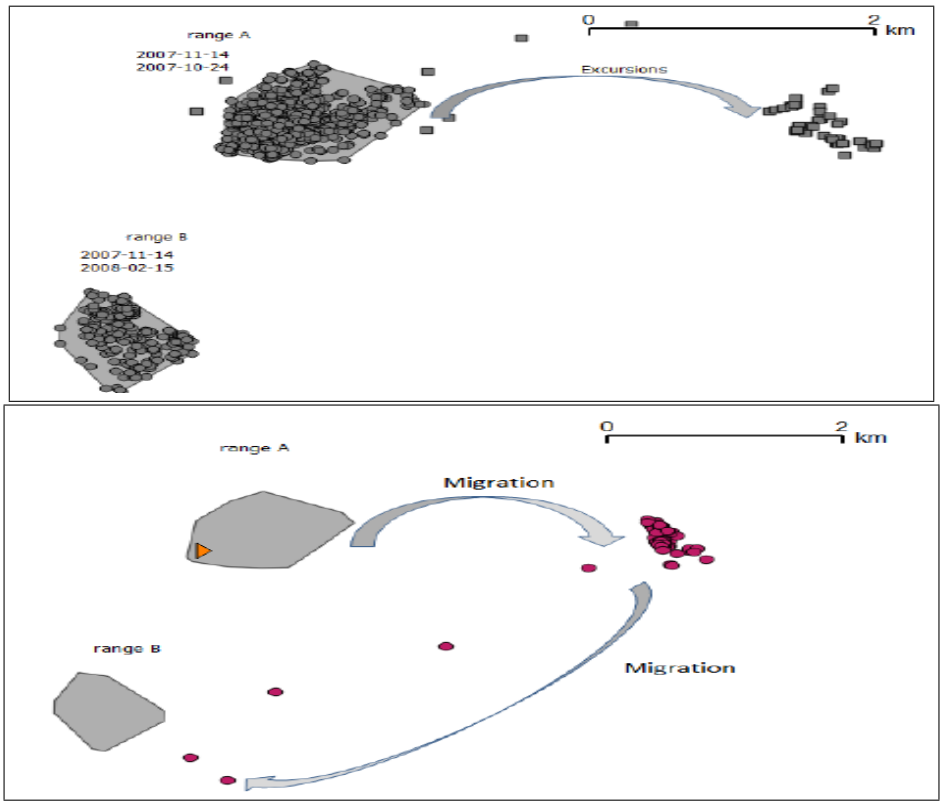



Figure 4.16: Clustering of the roe deer trajectory with ID=5. Figure shows two stay regions A and B. Top Figure shows the duration of each stay region and the excursion points while the animal is in A; bottom Figure shows the migration from A to B together with the stopover (in the same area of the excursion)

**Comparison with the reference technique.** We now turn to compare the outcome of SeqScan with the outcome of the reference technique applied to the same dataset. We base the comparison on two indicators defined in [11], i.e., :

- Animal's label. The first indicator simply specifies whether the animal observed in the time frame of the trajectory is migratory or stationary
- The migration distance. This is the distance between the centroids of the winter and summer clusters.

The aggregated results of the comparison are reported below while the details can be found in Table 4.7 and Table 4.6. In summary:

- In the roe deer dataset, 21 are classified as stationary (and 8 as migratory) by SeqScan. Of these 21, 17 are stationary also for the reference technique.

Thus the mismatches amount to 4 cases, i.e., trajectories that are classified as stationary by SeqScan and migratory by the reference technique.

- In the red deer dataset there is 1 mismatch, specifically one animal that is classified as migratory in SeqScan is classified as stationary by the reference method.

Of course, because the SeqScan parameters can be finely tuned, mismatching can be limited (we recall that in [11] the mismatching cases resulting from the comparison of three techniques to identify migration amounts even to 30%). The experiment demonstrates, however, that the outcome of SeqScan is sensible (from an ecological view point) while a suitable set of parameters can be found in a

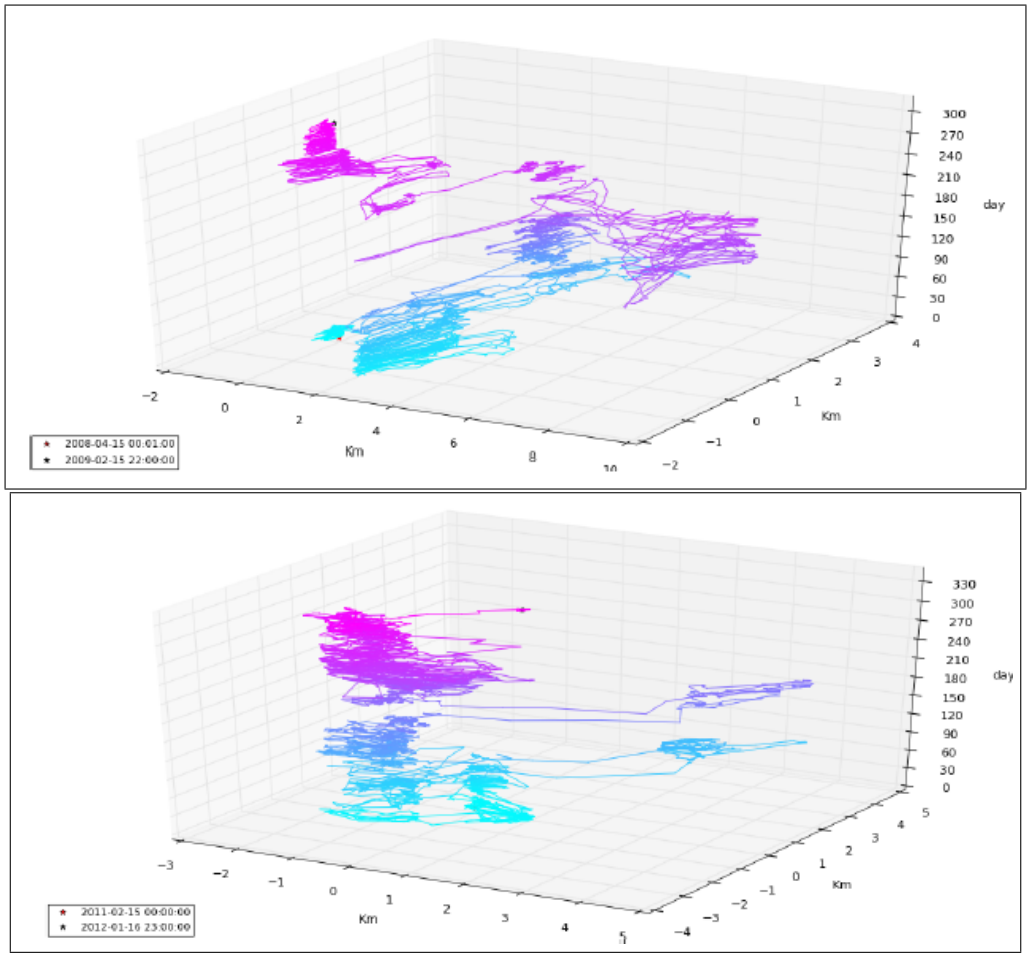



Figure 4.17: a) The trajectory of the red deer with ID=16; b) the trajectory of the red deer with ID=6

Table 4.6: Comparison between SeqScan (left side in every table) and the reference technique (right side) for roe-deer dataset.

| **ROE deer** | | | | | | |
|---|---|---|---|---|---|---|
| $ID_{Seq}$ | Label | $D_1$(m) | $ID_{RT}$ | Label | $D_2$(m) | $\frac{D_2-D_1}{D_2}$ |
| 1 | M | 1818 | 20 | M | 1995 | 0.09 |
| 2 | S | | 40 | S | | |
| 3 | S | | 50 | S | | |
| 4 | M | 3310 | 60 | M | 4313 | 0.23 |
| 5 | M | 1882 | 70 | M | 2010 | 0.06 |
| 6 | M | 26848 | 90 | M | 26873 | 0.00 |
| 7 | M | 1221 | 100 | M | 1119 | 0.09 |
| 8 | S | | 110 | S | | |
| 9 | M | 1214 | 120 | M | 1198 | 0.01 |
| **10** | **S** | | 130 | **M** | 1242 | |
| 11 | S | | 150 | S | | |
| 12 | S | | 160 | S | | |
| **13** | **S** | | 170 | **M** | 1189 | |
| 14 | S | | 171 | S | | |
| 15 | S | | 190 | S | | |
| 16 | S | | 210 | S | | |
| 17 | M | 3325 | 13670 | M | 3442 | 0.03 |
| 18 | S | | 13740 | S | | |
| 19 | S | | 13741 | S | | |
| 20 | S | | 13780 | S | | |
| 21 | M | 5144 | 13830 | M | 4836 | 0.06 |
| 22 | S | | 13840 | S | | |
| 23 | S | | 14021 | S | | |
| 24 | S | | 14030 | S | | |
| 25 | S | | 14110 | S | | |
| 26 | M | 1243 | 14111 | M | 983 | 0.26 |
| **27** | **S** | | 14150 | **M** | 1255 | |
| 28 | S | | 14180 | S | | |
| **29** | **S** | | 14310 | **M** | 575 | |

reasonably simple way. In this sense, we argue that SeqScan has performances comparable to the methods in use in animal ecology and thus can be of interest for the analysis of partial migrations, which is what we wanted to demonstrate.

Table 4.7: Comparison between SeqScan (left side in every table) and the reference technique (right side) for the red-deer dataset.

| **Red deer** | | | | | | |
|---|---|---|---|---|---|---|
| $ID_{Seq}$ | $Label_1$ | $D_1$(m) | $ID_{RT}$ | $Label_2$ | $D_2$(m) | $\frac{D_2-D_1}{D_2}$ |
| 1 | M | 1808 | 10 | M | | |
| 2 | M | 7325 | 20 | M | 6846 | 0.07 |
| 3 | M | 3942 | 30 | M | 3986 | 0.011 |
| 4 | S | | 40 | S | | |
| 5 | M | 4728 | 50 | M | 4848 | 0.025 |
| **6** | **M** | 1616 | 70 | **S** | | |
| 7 | M | 1912 | 80 | M | 1917 | 0.003 |
| 8 | M | 1449 | 81 | M | 1682 | 0.139 |
| 9 | S | | 90 | S | | |
| 10 | S | | 100 | S | | |
| 11 | S | | 110 | S | | |
| 12 | M | 7731 | 130 | M | 7496 | 0.031 |
| 13 | M | 30916 | 140 | M | 30583 | 0.011 |
| 14 | S | | 141 | S | | |
| 15 | M | 7121 | 151 | M | 7126 | 0.001 |
| 16 | M | 3007 | 160 | M | 5513 | 0.455 |
| 17 | S | | 170 | S | | |
| 18 | S | | 171 | S | | |
| 19 | S | | 180 | S | | |
| 20 | M | 1930 | 190 | M | 1900 | 0.016 |

## 4.5 Summary

Another major contribution of the thesis is the SeqScan algorithm for the discovery of stay regions from spatial trajectories. The algorithm presents novel features that seem particularly effective for the study of animal movement. The technique has been applied for validation purposes on real cases. Other forms of validation, based on the use of synthetic data, are currently under investigation.

# Chapter 5

# Scenario

After presenting the spatio-textual trajectory representation and the *SeqScan* clustering technique and its application to the analysis of animals' trajectories, we try close the loop and show the spatio-textual trajectories at work. In this chapter we utilize the data resulting from the clustering of the animals' spatial trajectories to illustrate how the spatio-textual trajectories can be built and queried in practice. Similarly, we present a few queries over the GeoLife dataset. The chapter is organized as follows. We start presenting the experimental setting and next we discuss the examples for the two datasets.

## 5.1 Usage setting

We consider trajectory data sources consisting of two parts: a set of textual trajectories and a set of aligned spatial trajectories. The individual's movement is thus characterized by the pair textual-spatial trajectory. We use the Secondo platform [28] as hosting database and software environment. Starting from source data, the process of data organization in the database consists of the two following steps. First, trajectories are uploaded and represented in terms of the system data types, as spatial and symbolic trajectories. Next, the combined spatio-textual trajectories are created through an operation that we call *temporal overlay*. The temporal overlay of a spatial trajectory $S$ and a symbolic trajectory $Y$ is performed by intersecting the time periods of the units in $S$ and $Y$, and then associating each of the resulting small periods, with the label and the segment of the units falling in that period. In summary, similarly to the operation of spatial overlay in GIS, temporal overlay is a composite operation consisting of a (temporal) intersection and a (temporal) join.

### 5.1.1 Implementation in Secondo

At implementation level, the process of construction of the spatio-textual trajectories is performed as follows: preliminarily a new abstract data type, called *trip*, has been created in *Secondo* for the representation of spatio-textual trajectories. The following operators have been developed for the Trip class:

- temporal *overlay*: this is a constructor that takes as input an object of type mpoint and an object of type mlabel and returns an object of type *trip*
- *match*: this is a boolean operator that takes as input a sequenced query in the form of text expression and returns true if the trip object satisfies the query

Finally, for each dataset, the trajectories are imported, and a table is created with the following simple schema:

```
TABLENAME(Id: int, Traj: trip)
```

## 5.2 Querying the Geolife Dataset

The spatio-textual trajectories derived from the Geolife dataset consist of trajectories that, we recall, are labeled with the transportation modes. Every trajectory represents a one-day trip. The dataset used for these experiments consists of the

trips of 69 users covering a period of about three years. We run a few queries over these trajectories and report the results visually, through the Secondo interface. Spatio-textual trajectories are stored in the table GeoLife.

**Example 5.1.** We start with a simple query that does not involve spatial conditions and that thus is entirely symbolic. The request has the following meaning: *retrieve the individuals who walk in the morning, bike in the afternoon, and then walk again.* The request is solved using the following sequenced query:

$$P1 = ([6:00\ 12:00], \{"walk"\}, \_)\ \ (\_, \{"bike"\}, \_)\ \ ([16:00\ 24:00], \{"walk"\}, \_)$$

In Secondo the query is formulated as follows:

```
query Geolife feed filter[.Traj match P1] consume
```

Figure 5.1 displays one of the resulting trajectories.

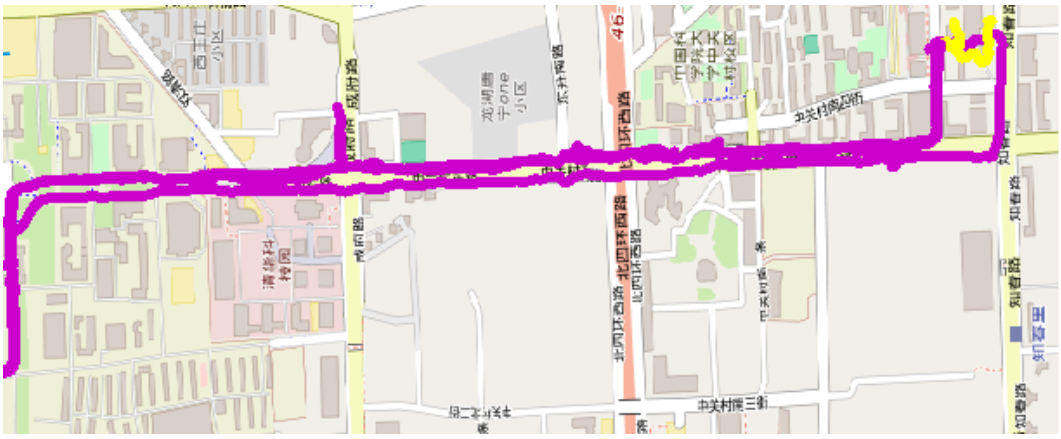
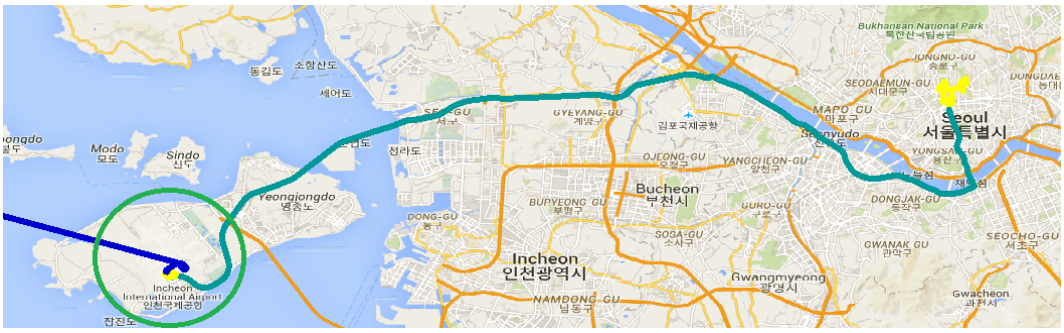


Figure 5.1: Example 5.1: one of the resulting spatio-textual trajectories. Bike and walk are displayed in purple and yellow color, respectively.

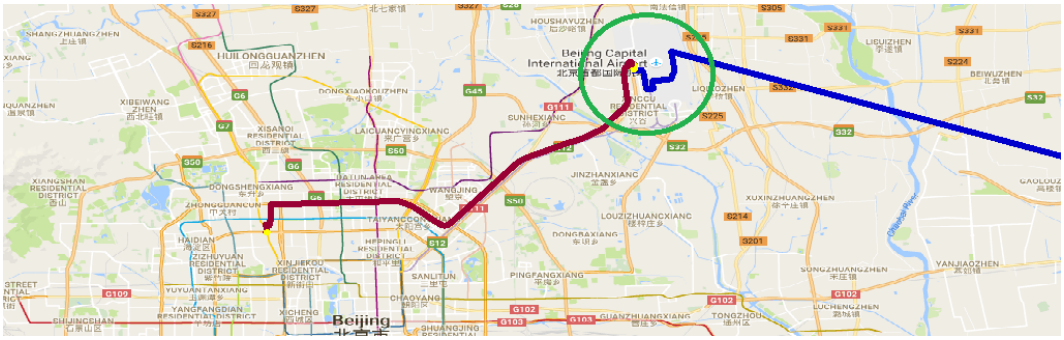
**Example 5.2.** In this example we introduce spatial conditions in the query. The request is: *to retrieve the trips of the individuals reaching a departure airport by car, taking a plane and finally a taxi at the arrival airport.* We refer to the Incheon International Airport for departure and Beijing International Airport for arrival, and denote the corresponding regions $A_1$ and $A_2$, respectively. The sequenced query is:

$$(\_\_, \{"car"\}, \_\_)\ (\_\_, \{"airplane"\}, A_1)\ (\_\_, \{"airplane"\}, A_2)\ (\_\_, \{"taxi"\}, \_\_)$$

One of the resulting trajectories is displayed in Figure 5.2.




(a)




(b)

Figure 5.2: Example 5.2: Figure (a) displays the first part of the trip before taking off; Figure (b) the second part of the trip after landing. The airport areas are enclosed in circles.

**Example 5.3.** We show another example combining spatial and textual conditions. The request is: *retrieve the trajectories of the individuals starting their trip by walking to a bus station, next take the bus and reach Kunming Lake (a touristic region in China) for a morning boat trip.* The sequenced query can be formulated as follows:

$$(\_\_, \{"walk"\}, \_\_) \quad (\_\_, \{"bus"\}, \_\_) \quad ([08:00\ 10:00], \{"boat"\}, Kunming)$$

One of the resulting trajectories is displayed in Figure 5.3.

**Example 5.4.** Sequenced queries can help extracting information about human mobility in cities. Such information may be useful for traffic analysis and transportation planning. In this example, the request is *to compute the percentage of trips occurring between two regions, where the individuals use exclusively public transportation.* For this example we consider two regions: Region1 and Region2,

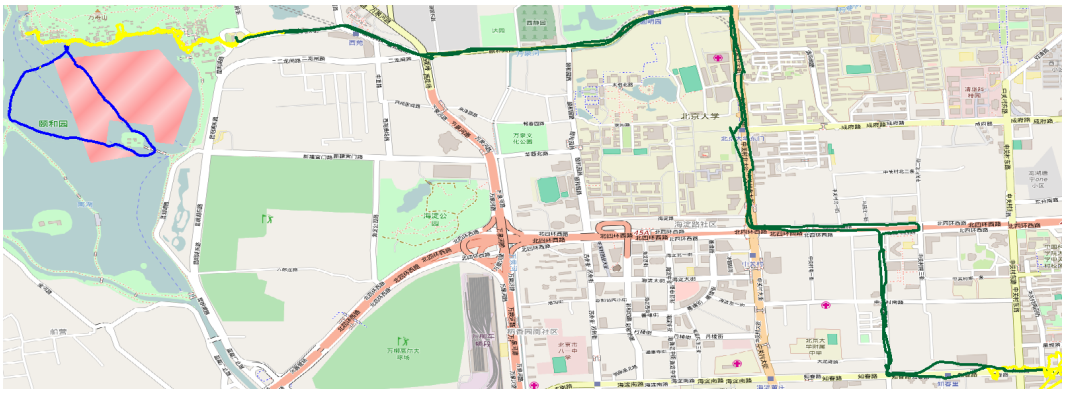


Figure 5.3: Example 5.3: one of the resulting trips. The individual walks for a while (yellow line), then takes a bus (green line) to reach the lake, then walks again to reach the boat (the blue line indicates the boat trip) and finally makes the trip back home

significantly far away from each other. The request can be solved performing two queries: the first is to retrieve the number of individuals that move from Region1 to Region2 and back by bus or train (i.e., public transportation) after walking to reach the respective departure locations; the second query is simply to retrieve the number of individuals that move from Region1 to Region2 and back. Finally the percentage can be easily computed as ratio. The former sequenced query is as follows:

$$(\_\_, \{"walk"\}, Region1)(\_\_, \{"bus", "train"\}, Region2)(\_\_, \{"walk"\}, Region1)$$

The resulting number of trips satisfying the query is 2 trips over 12. The remaining 10 trips over 12 employ private transportation, such as cars and taxi services. One of the trips is displayed in Figure 5.4.

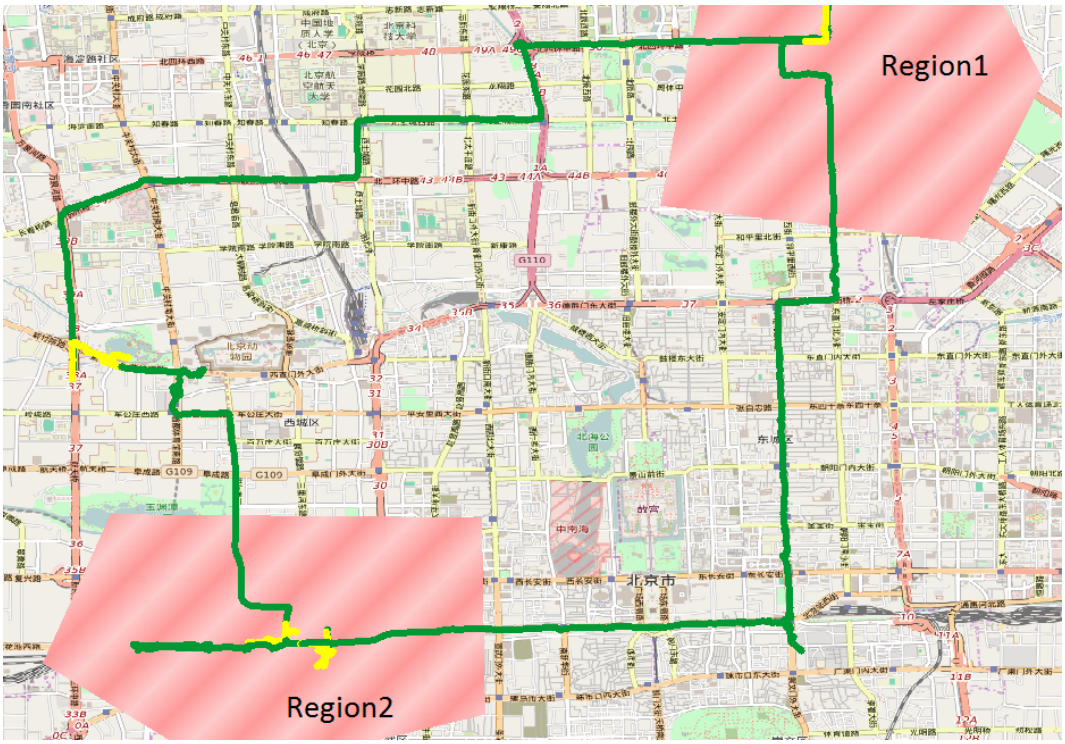



Figure 5.4: Example 5.4: the trip of an individual commuting between Region1 and Region2, and then returning to Region1. The individual located in Region1 walks (yellow line) for a while to get the bus (green line) next reaches Region2 and then walks again to reach the final destination from which he/she departs afterwards. Region1 and Region2 are represented by the polygonal areas.

## 5.3 Querying the Animals' Dataset

The datasets contains the symbolic trajectories describing the migratory behavior of the roe deer of the Tento datasets, analyzed in chapter 4.

As explained before, the SeqScan algorithm is applied in two steps. First, we scan the whole trajectory to extract the large regions, i.e., home ranges. Second, we run another scan over the sub-trajectories that represent filtered noise, long gaps and transitions, to detect the small regions. The output is a spatio-textual trajectory annotated with 3 labels: H, E, and S, where H stands for home range, S for stopover between two home ranges, and E presents an excursion between two points within the same home range. Running sequenced queries over the annotated trajectories may be the source of useful information about the animals behaviors. In particular, in the following examples, we try to understand the interaction between the animals and the environment, namely the land type. For

this purpose, we use the Corine Land Cover (CLC) project, which describes the land cover of most European countries. Figure 5.5 represents the distribution of CLC land classes in the Trento area.

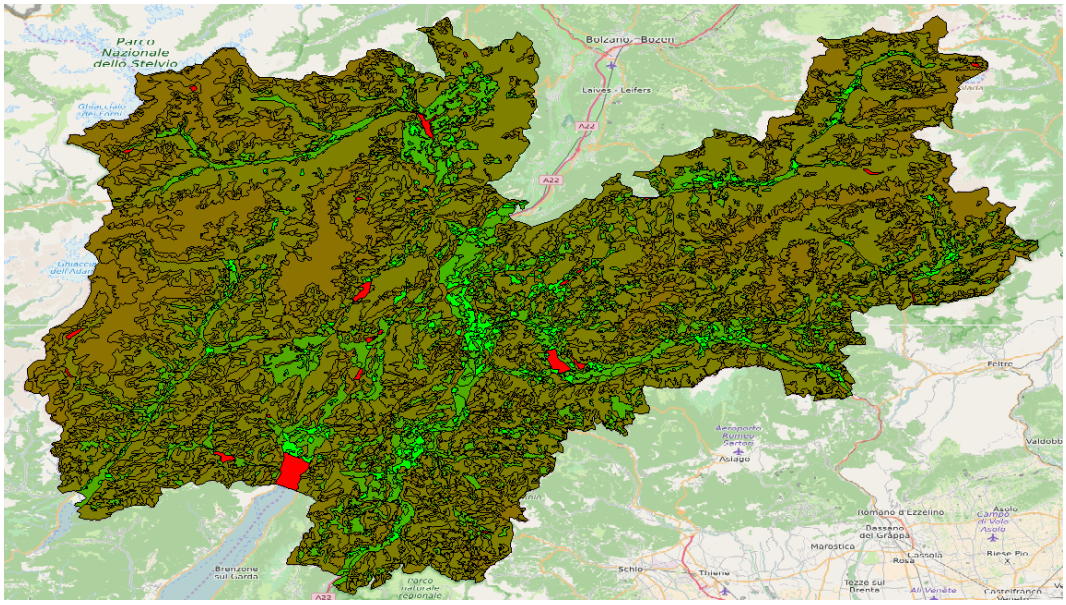



Figure 5.5: Land types in the region based on the Corine classification. Every land type is represented by a different color.

For exemplification purposes, we limit ourselves to consider the following land types:

- Land class 112: Urban area.
- Land class 231: Pastures.
- Land class 243: Rural area.
- Land class 313: Forest area.

**Example 5.5.** While the animals stay most of the time within forest, it may happen that they move to regions of different land type. In this example the request is to retrieve the trajectories of those animals passing unexpectedly through urban areas. The simple sequenced query is reported below. A result is shown in Figure 5.6.

$$(\_\_, \{"H"\}, Urban)$$

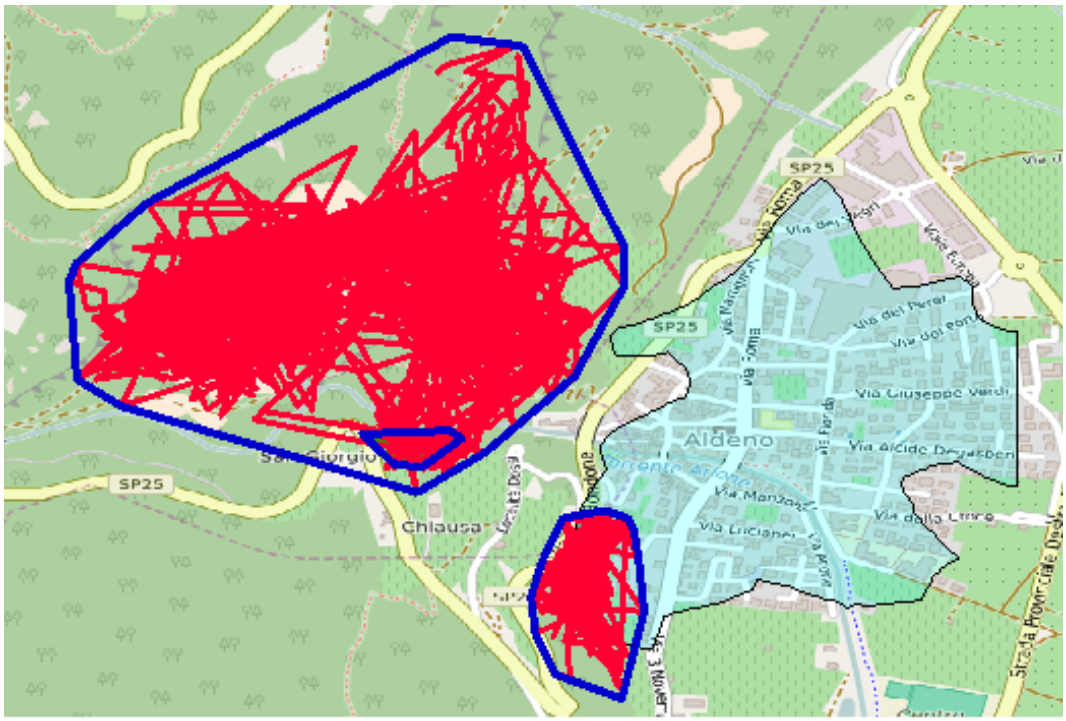



Figure 5.6: Example 5.5: Trajectory (red color) of an animal passing through an urban area (light blue color).

**Example 5.6.** In this example, we focus on the animal's migratory behavior. The request is to extract the trajectories of the animals migrating from one home range overlapping a forest to another home range overlapping pastures, with a stopover in between. A result is reported in Figure 5.7.

$$(\_\_, \{"H"\}, Forest)(\_\_, \{"S"\}, \_\_)(\_\_, \{"H"\}, Pastures)$$

**Example 5.7.** As we have seen before, animals can make occasional excursions outside the home range. In this example, the goal is to retrieve the trajectories of the animals making excursions in forest areas while the home range is a pasture area. Our query is composed of 3 simple queries, as shown below. A result is displayed in Figure 5.8.

$$(\_\_, \{"H"\}, Pastures)(\_\_, \{"E"\}, Forest)(\_\_, \{"H"\}, Pastures)$$

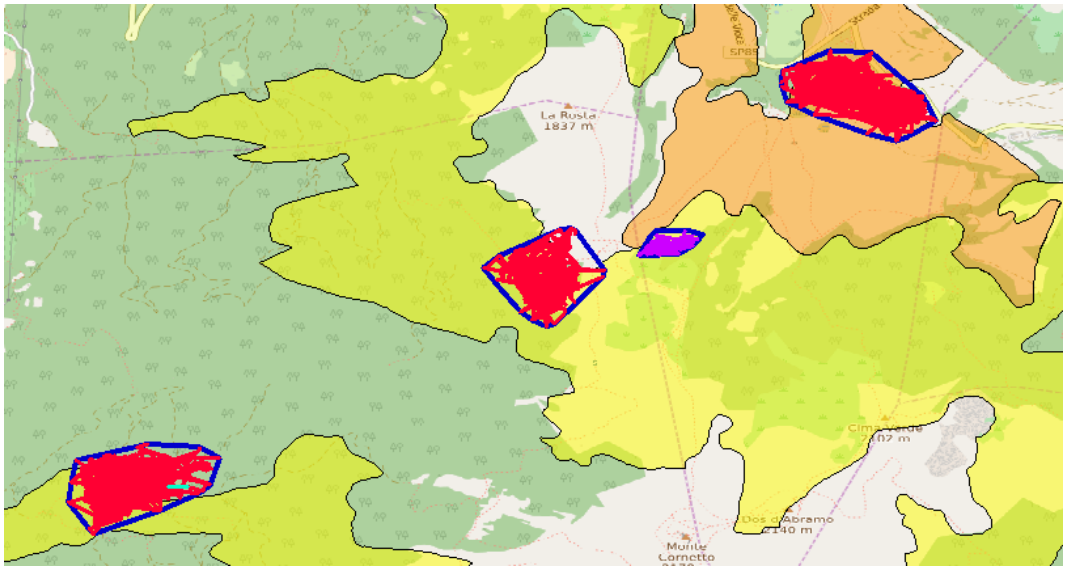



Figure 5.7: Example 5.6: Animal making a stopover between two home ranges of different land types. The red lines are segments within the home range, and the violet ones are segments within the stop area. Pasture area is in orange and Forest area is in yellow.

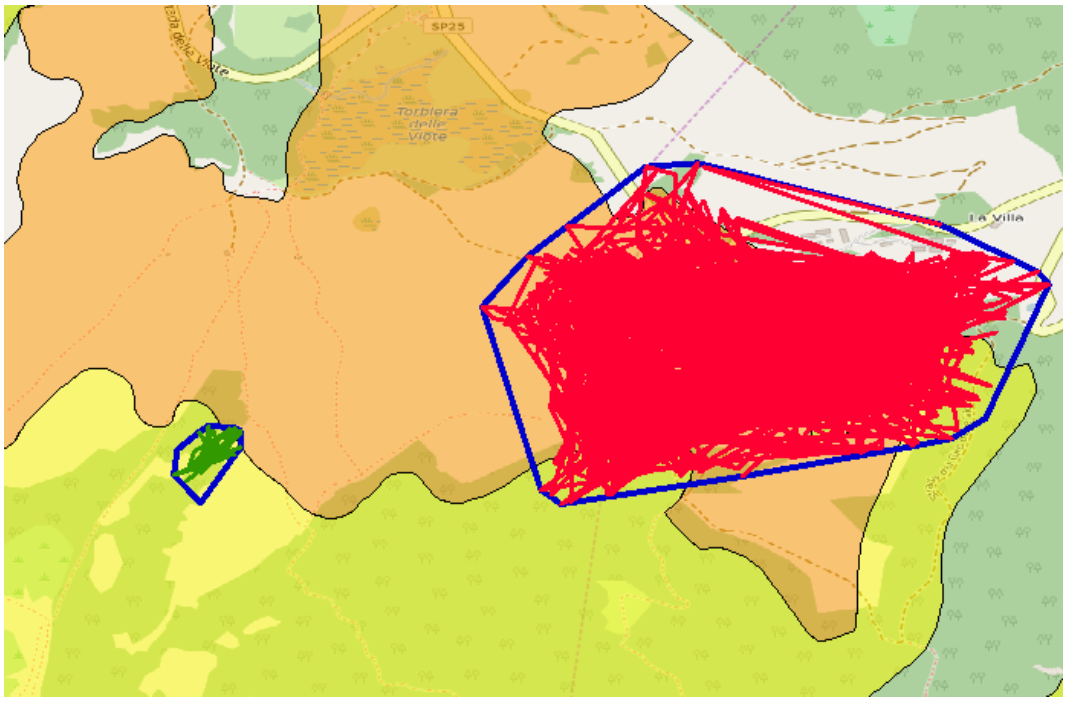



Figure 5.8: Example 5.7: Animal resident (red lines) in a pasture (orange polygon) and making an excursion (green lines) in a forest area (yellow polygon).

# Chapter 6

# Conclusions

In this chapter, we briefly summarize the main contributions of this thesis, followed by the prospect possible directions for future research.

## 6.1 Technical Contributions

The wide availability of time-varying data has led to massive amount of spatial and non-spatial trajectories. Therefore various data models have been proposed for the representation of different types of trajectories. The research presented in this thesis has been driven by a main question, whether blending together traditional spatial trajectories and emerging symbolic trajectories can lead to a more effective data model, combining the advantages of the two paradigms and thus potentially attracting a larger number of potential users (e.g. mobile application developers). The question is complex. Since the first experiments, however, it has been clear that the absolute inefficiency of naive techniques would have hampered the development of any realistic solution. From that it has been resulted the decision of focusing on the study of novel access methods integrating space-time-text and supporting sequenced queries. While spatio-textual indexes have been extensively investigated for static objects, the combined use of text and space-time for trajectories indexing is a novel topic. In this sense we believe the IRWI framework, for the innovative ideas it contains, can contribute to the foundation of a new research stream. This first result inevitably raises new questions suggesting new solutions and directions.

Working on the efficiency aspects, however, is not enough for answering the initial question on the usefulness of spatio-textual trajectories. For a data model to be meaningful, there must be a certain consensus on its 'usefulness'. This implies experimenting the system with real data and within the context of real applications. To that extent, the domain of animal ecology is particularly at-

tractive. This research focused on the extraction of the mobility behavior from the animals'spatial trajectories and then its encoding is terms of symbolic and next spatio-textual trajectories. In reality the case study has been particularly challenging. This work has led to the proposal of a novel spatio-temporal clustering technique, SeqScan, that has been successfully applied to the analysis of the mobility behavior of a species of wild animals. Definitely promising, this research has triggered a new stream of work.

## 6.2 Future Work and Research Directions

The research conducted in the thesis paves the way to further developments. A few possible directions are discussed next.

**Trajectories with multiple textual dimensions.** Currently, spatio-textual trajectories allow for the representation of a unique textual dimension. Real scenarios, however, may include multiple textual trajectories. For example, the movement of a vehicle can be characterized by the transportation means, the weather conditions, e.g. rainy, sunny weather, road typology, e.g. road type and so on. This extension impacts the query model, the access method, calling as well for novel visualization techniques. Another line of research, especially motivated by the presence of textual component is the investigation of similarity measures accounting for space, time and text.

**IRWI and query processing.** Currently, the IRWI index is used to support the processing of sequenced queries while its performance heavily depends on the characteristics of two key methods: the summarization of the trajectory Ids, and the hybrid spatio-textual cost function. These two methods can be improved in several ways. As concerns the summarization of trajectory Ids, an interesting direction is to investigate alternative solutions requiring less disk storage and performing a better filtering of trajectories. As concerns the spatio-textual cost function, the cost of the insertion into the IRWI tree depends on a system defined parameter $\beta$ which is a compromise between spatial and textual costs. An alternative approach is to leverage learning techniques to use $\beta$ in an adaptive way. Another line of research concerns the extension of the query processing techniques to handle different types of queries such as ranking functions (nearest neighbor, top-k queries).

**Stay region model and SeqScan.** The process of validation of SeqScan, which has led to the application of the technique on real data, is not completed

yet. A main direction for future work concerns the investigation of suitable internal and external quality indicators for the clustering.

# Bibliography


[1] L.O. Alvares, V. Bogorny, B. Kuijpers, J. de Macedo, B. Moelans, and A. Vaisman. A model for enriching trajectories with semantic geographical information. In *Proc. ACM GIS*, 2007.

[2] A. Anagnostopoulos, M. Vlachos, M. Hadjieleftheriou, E. Keogh, and P. Yu. Global distance-based segmentation of trajectories. In *Proc. ACM SIGKDD*, 2006.

[3] B. Aronov, A. Driemel, M. V. Kreveld, M. Löffler, and F. Staals. Segmentation of trajectories on nonmonotone criteria. *ACM Trans. Algorithms*, 2015.

[4] D. Ashbrook and T. Starner. Using gps to learn significant locations and predict movement across multiple users. *Personal Ubiquitous Comput*, 7(5):275–286, 2003.

[5] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 1975.

[6] J. L. Bentley. Decomposable searching problems. *Information Processing Letters*, 8(5):244 – 251, 1979.

[7] V. Bogorny, B. Kuijpers, and L. O. Alvares. ST-DMQL: A semantic trajectory data mining query language. *International Journal of Geographical Information Science*, 23(10):1245–1276, 2009.

[8] V. Bogorny, C. Renso, A. R. de Aquino, F. L. de Siqueira, and L. O. Alvares. CONSTAnT - a conceptual data model for semantic trajectories of moving objects. *Trans. GIS*, 2013.

[9] M. Buchin, A. Driemel, M. van Kreveld, and V. Sacrist. Segmenting trajectories: A framework and algorithms using spatiotemporal criteria. *Journal of Spatial Information Science*, 3(1):630–660, 2011.

[10] W. H. Burt. Territoriality and home range concepts as applied to mammals. *Journal of Mammalogy*, 24(3):346–352, 1943.

[11] F. Cagnacci, S. Focardi, A. Ghisla, B. van Moorter, E. H. Merrill, E. Gurarie, M. Heurich, A. Mysterud, J. Linnell, M. Panzacchi, R. May, T. Nygård, C. Rolandsen, and M. Hebblewhite. How many routes lead to migration? comparison of methods to assess and characterize migratory movements. *Journal of Animal Ecology*, 85(1):54–68, 2016.

[12] F. Cagnacci, S. Focardi, M. Heurich, A. Stache, A. J. Mark Hewison, N. Morellet, P. Kjellander, J. D. C. Linnell, A. Mysterud, M. Neteler, L. Delucchi, F. Ossi, and F. Urbano. Partial migration in roe deer: migratory and resident tactics are end points of a behavioural gradient determined by ecological factors. *OIKOS*, 120(12):1790–1802, 2011.

[13] V. P. Chakka, A. Everspaugh, and J. M. Patel. Indexing large trajectory data sets with SETI. In *CIDR*, 2003.

[14] B. B. Chapman, C. Brönmark, J. A. Nilsson, and L. A. Hansson. The ecology and evolution of partial migration. *OIKOS*, 120(12):1764–1775, 2011.

[15] L. Chen, G. Cong, C. S. Jensen, and D. Wu. Spatial keyword query processing: An experimental evaluation. *Proc. VLDB Endow*, 2013.

[16] M. Christoforaki, J. He, C. Dimopoulos, A. Markowetz, and T. Suel. Text vs. Space: Efficient geo-search query processing. In *Proc. ACM CIKM*, 2011.

[17] G. Cong, C. S. Jensen, and D. Wu. Efficient retrieval of the Top-k most relevant spatial web objects. *In Proc. VLDB Endow*, 2009.

[18] P. Corti, S. V. Mather, T. J. Kraft, and B. Park. *PostGIS Cookbook*. Packt Publishing, 2014.

[19] M. L. Damiani and R. H. Güting. Semantic trajectories and beyond. In *Proc. IEEE MDM*, 2014.

[20] M. L. Damiani, H. Issa, and F. Cagnacci. Extracting stay regions with uncertain boundaries from gps trajectories: A case study in animal ecology. In *Proc. ACM SIGSPATIAL*, 2014.

[21] M. L. Damiani, H. Issa, R. H. Güting, and F. Valdés. Hybrid queries over symbolic and spatial trajectories: A usage scenario. In *Proc. IEEE MDM*, 2014.

[22] M. L. Damiani, H. Issa, R. L. Güting, and F. Valdés. Symbolic trajectories and application challenges. *SIGSPATIAL Special*, 7(1):51–58, 2015.

[23] C. Düntgen, T. Behr, and R. H. Güting. BerlinMOD: A benchmark for moving object databases. *The VLDB Journal*, 18(6):1335–1368, 2009.

[24] M. Ester, H. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. *KDD*, 1996.

[25] R. Fileto, M. Krüger, N. Pelekis, Y. Theodoridis, and C. Renso. Baquara: A holistic ontological framework for movement analysis using linked data. In *ER*, 2013.

[26] L. Forlizzi, R. H. Güting, E. Nardelli, and M. Schneider. A data model and data structures for moving objects databases. In *Proc. ACM SIGMOD*, 2000.

[27] F. Giannotti and D. Pedreschi. *Mobility, Data Mining and Privacy: Geographic Knowledge Discovery*. Springer Publishing Company, Incorporated, 1 edition, 2008.

[28] R. H. Güting, V. Almeida, D. Ansorge, T. Behr, Z. Ding, T. Höse, F. Hoffmann, M. Spiekermann, and U. Telle. SECONDO: An extensible dbms platform for research prototyping and teaching. In *Proc. IEEE ICDE*, 2005.

[29] R. H. Güting, M. H. Böhlen, M. Erwig, C. S. Jensen, N. A. Lorentzos, M. Schneider, and M. Vazirgiannis. A foundation for representing and querying moving objects. *ACM Trans. Database Syst*, 2000.

[30] R. H. Güting and M. Schneider. *Moving Objects Databases*. Morgan Kaufmann Publishers, 2005.

[31] R.H. Güting, F. Valdés, and M. L. Damiani. Symbolic trajectories. *ACM Transactions on Spatial Algorithms and Systems*, 1(2):7, 2015.

[32] A. Guttman. R-trees: A dynamic index structure for spatial searching. *SIGMOD Rec*, 14(2):47–57, 1984.

[33] M. Hadjieleftheriou, G. Kollios, P. Bakalov, and V. J. Tsotras. Complex spatio-temporal pattern queries. In *Proc. VLDB*, 2005.

[34] Y. Han, L. Wang, Y. Zhang, W. Zhang, and X. Lin. Spatial keyword range search on trajectories. In *DASFAA (2)*, 2015.

[35] R. Hariharan, B. Hore, C. Li, and S. Mehrotra. Processing spatial-keyword (sk) queries in geographic information retrieval (gir) systems. In *Proc. IEEE SSDBM*, 2007.

[36] Y. Hu, K. Janowicz, D. Carral, S. Scheider, W. Kuhn, G. Berg-Cross, P. Hitzler, M. Dean, and D. Kolas. *A Geo-ontology Design Pattern for Semantic Trajectories*, pages 438–456. Springer International Publishing, 2013.

[37] B. Krogh, N. Pelekis, Y. Theodoridis, and K. Torp. Path-based queries on trajectory data. In *Proc. ACM SIGSPATIAL*, 2014.

[38] Z. Li, K. C. K. Lee, B. Zheng, W. Lee, D. Lee, and X. Wang. IR-Tree: An efficient index for geographic document search. *IEEE Trans. on Knowl. and Data Eng.*, 2011.

[39] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[40] Y. Manolopoulos, A. Nanopoulos, A. Papadopoulos, and Y. Theodoridis. *R-Trees: Theory and Applications*. Springer Publishing Company, Incorporated, 2005.

[41] C. Mouza and P. Rigaux. Mobility patterns. In *STDBM*, 2004.

[42] M. A. Nascimento and J. R. O. Silva. Towards historical r-trees. In *Proc. ACM SAC*, 1998.

[43] L. Nguyen-Dinh, W. G. Aref, and M. F. Mokbel. Spatio-Temporal Access Methods: Part 2 (2003 - 2010). *IEEE Data Eng. Bull.*, 33(2):46–55, 2010.

[44] A. T. Palma, V. Bogorny, B. Kuijpers, and L. O. Alvares. A clustering-based approach for discovering interesting places in trajectories. In *Proc. ACM SAC*, 2008.

[45] C. Parent, S. Spaccapietra, C. Renso, G. Andrienko, N. Andrienko, V. Bogorny, M. L. Damiani, A. Gkoulalas-Divanis, J. Macedo, N. Pelekis, Y. Theodoridis, and Z. Yan. Semantic trajectories modeling and analysis. *ACM Comput. Surv.*, 2013.

[46] N. Pelekis, Y. Theodoridis, S. Vosinakis, and T. Panayiotopoulos. Hermes – a framework for location-based data management. In *Proc. EDBT*, 2006.

[47] D. Pfoser, C. S. Jensen, and Y. Theodoridis. Novel approaches to the indexing of moving object trajectories. In *Proc. VLDB*, 2000.

[48] J. A. M. R. Rocha, V. C. Times, G. Oliveira, L. O. Alvares, and V. Bogorny. DB-SMoT: A direction-based spatio-temporal clustering method. In *IEEE IS*, 2010.

[49] M. A. Sakr and R. H. Güting. Spatiotemporal pattern queries in Secondo. In *Proc. SSTD*, 2009.

[50] J. Sander, M. Ester, H.P. Kriegel, and X. Xu. Density-Based Clustering in Spatial Databases: The Algorithm GDBSCAN and Its Applications. *Data Mining and Knowledge Discovery*, 2(2):169–194, 1998.

[51] J. Schiller and A. Voisard. *Location Based Services*. Morgan Kaufmann Publishers Inc., 2004.

[52] S. Spaccapietra, C. Parent, M. L. Damiani, J. A. F. de Macêdo, F. Porto, and C. Vangenot. A conceptual view on trajectories. *Data Knowl. Eng.*, 65(1):126–146, 2008.

[53] L. Sun and K. W. Axhausen. Understanding urban mobility patterns with a probabilistic tensor factorization framework. *Transportation Research Part B: Methodological*, 91:511–524, 2016.

[54] Y. Tao and D. Papadias. MV3R-Tree: A spatio-temporal access method for timestamp and interval queries. In *Proc. VLDB*, 2001.

[55] Y. Theodoridis, M. Vazirgiannis, and T. Sellis. Spatio-temporal indexing for large multimedia applications. In *Proc. IEEE ICMCS*, 1996.

[56] S. Vaid, C. B. Jones, H. Joho, and M. Sanderson. Spatio-textual indexing for geographical search on the web. In *Proc. SSTD*, 2005.

[57] F. Valdés and R. H. Güting. Index-supported pattern matching on symbolic trajectories. In *Proc. ACM SIGSPATIAL*, 2014.

[58] M. R. Vieira, P. Bakalov, and V. J. Tsotras. Querying trajectories using flexible patterns. In *Proc. ACM EDBT*, 2010.

[59] J. H. Ward. Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association*, 58(301):236–244, 1963.

[60] Z. Yan and D. Chakraborty. *Semantics in Mobile Sensing*. MorganClaypool, 2014.

[61] Z. Yan, D. Chakraborty, C. Parent, S. Spaccapietra, and K. Aberer. SeMiTri: A framework for semantic annotation of heterogeneous trajectories. In *Proc. EDBT/ICDT*, 2011.

[62] Z. Yan, D. Chakraborty, C. Parent, S. Spaccapietra, and K. Aberer. Semantic trajectories: Mobility data computation and annotation. *ACM Trans. Intell. Syst. Technol.*, 2013.

[63] Z. Yan, J. Macedo, C. Parent, and S. Spaccapietra. Trajectory ontologies and queries. *Transactions in GIS*, 12(1):75–91, 2008.

[64] H. Yoon and C. Shahabi. Robust time-referenced segmentation of moving object trajectories. In *Proc. ICDM*, 2008.

[65] Y. Zhengand L. Zhang, Z. Ma, X. Xie, and Wei-Ying W. Ma. Recommending friends and locations based on individual location history. *ACM Trans. Web*, 2011.

[66] K. Zheng, Y. Yang, S. Shang, and N. J. Yuan. Towards efficient search for activity trajectories. In *Proc. IEEE ICDE*, 2013.

[67] Y. Zheng, L. Wang, R. Zhang, X. Xie, and W. Ma. Geolife: Managing and understanding your past life over maps. In *Proc. IEEE MDM*, 2008.

[68] Y. Zheng and X. Zhou. *Computing with Spatial Trajectories*. Springer Publishing Company, Incorporated, 2011.

[69] Y. Zhou, X. Xie, C. Wang, Y. Gong, and W. Ma. Hybrid index structures for location-based web search. In *Proc. ACM CIKM*, 2005.

[70] M. Zimmermann, T. Kirste, and M. Spiliopoulou. Finding stops in error-prone trajectories of moving objects with time-based clustering. In *Intelligent Interactive Assistance and Mobile Multimedia Computing*, volume 53, pages 275–286. Springer Berlin Heidelberg, 2009.

[71] J. Zobel and A. Moffat. Inverted files for text search engines. *ACM Comput. Surv.*, 38(2), July 2006.

# Appendix

### Publications of Hamza Issa


[1] Hamza Issa and Maria Luisa Damiani, "Efficient access to temporally overlaying spatial and textual trajectories," in Proc. IEEE on Mobile Data Management(MDM), June 2016.

[2] Maria Luisa Damiani, Hamza Issa, Giuseppe Fotino, Marco Heurich, and Francesca Cagnacci, "Introducing 'presence' and 'stationarity index' to study partial migration patterns: an application of a spatio-temporal clustering technique," International Journal of Geographical Information Science, 30(5):907-928, 2016.

[3] Maria Luisa Damiani, Hamza Issa, Giuseppe Fotino, Fatima Hachem, Nathan Ranc, and Francesca Cagnacci, "Migro: A plug-in for the analysis of individual mobility behavior based on the stay region model," in Proc. ACM SIGSPATIAL, Nov 2015.

[4] Maria Luisa Damiani, Hamza Issa, Ralf Hartmut Güting, and Fabio Valdés, "Symbolic trajectories and application challenges," SIGSPATIAL Special, 7(1):51-58, Nov 2015.

[5] Maria Luisa Damiani, Hamza Issa, and Francesca Cagnacci, "Extracting stay regions with uncertain boundaries from gps trajectories: A case study in animal ecology," in Proc. ACM SIGSPATIAL, Nov 2014.

[6] Maria Luisa Damiani, Hamza Issa, Ralf Hartmut Güting, and Fabio Valdés, "Hybrid queries over symbolic and spatial trajectories: A usage scenario," in Proc. IEEE on Mobile Data Management(MDM), July 2014.

[7] Maria Luisa Damiani, Ralf Hartmut Güting, Fabio Valdés, and Hamza Issa, "Moving objects beyond raw and semantic trajectories," in Information Management in Mobile Applications(IMMoA), Aug 2013.